Int J Comput Vis (2017) 121:253–280
DOI 10.1007/s11263-016-0929-y

Large-Scale Gaussian Process Inference with Generalized Histogram Intersection Kernels for Visual Recognition Tasks

Erik Rodner[1,2] · **Alexander Freytag**[1,2] · **Paul Bodesheim**[3] · **Björn Fröhlich**[4] · **Joachim Denzler**[1,2]

Received: 10 November 2014 / Accepted: 4 July 2016 / Published online: 27 July 2016

Abstract We present new methods for fast Gaussian process (GP) inference in large-scale scenarios including exact multi-class classification with label regression, hyperparameter optimization, and uncertainty prediction. In contrast to previous approaches, we use a full Gaussian process model without sparse approximation techniques. Our methods are based on exploiting generalized histogram intersection kernels and their fast kernel multiplications. We empirically validate the suitability of our techniques in a wide range of scenarios with tens of thousands of examples. Whereas plain GP models are intractable due to both memory consumption and computation time in these settings, our results show that exact inference can indeed be done efficiently. In consequence, we enable every important piece of the Gaussian process framework—learning, inference, hyperparameter optimization, variance estimation, and online learning—to be used in realistic scenarios with more than a handful of data.

Keywords Large-scale learning · Gaussian processes · Hyperparameter optimization · Visual recognition

Communicated by Raquel Urtasun.

✉ Erik Rodner
Erik.Rodner@uni-jena.de

[1] Friedrich Schiller University Jena, Jena, Germany

[2] Michael Stifel Center Jena, Jena, Germany

[3] Max Planck Institute for Biogeochemistry, Jena, Germany

[4] Daimler AG Research & Development, Böblingen, Germany

1 Introduction

Over the last years, visual image categorization has been dominated by a few classification concepts. Support Vector Machines represented the state-of-the-art for the last decade, e.g., the famous solvers LibLinear by Fan et al. (2008) as well as LibSVM of Chang and Lin (2011). More recently, they were replaced by Convolutional Neural Networks trained with deep learning techniques (Krizhevsky et al. 2012). Today, deep networks are established as excellent black-box tools which lead to impressive results in image categorization challenges (Russakovsky et al. 2015). However, these models only provide what they were designed and trained for: estimated class labels for previously unseen data. Are class labels and plain predictions the only thing we are interested in?

Nonparametric Bayesian methods based on Gaussian process models have the advantage of providing a complete probabilistic framework for inference. In consequence, they allow for estimating the variance of a prediction or finding suitable hyperparameters with marginal likelihood optimization as shown by Kapoor et al. (2010). Unfortunately, their application to large-scale learning scenarios is limited since required computation times and memory consumption scale cubically and quadratically with the number of collected examples, respectively. On the other hand, current research is shifting more and more towards large-scale learning scenarios due to the huge number of available image data. Thus, there exists an increasing gap between the benefits of the GP framework and their applicability to current visual recognition scenarios.

Contributions of this Article In this paper, we show how to overcome the scaling issues of the GP framework even in the presence of a large number of learning examples.

Our insights are based on inherent properties of histogram intersection kernels (HIK) which serve as similarity measure between histogram features. Histograms arise from a large number of popular image representations, e.g., SIFT, HOG, or visual Bag-of-words (BOW). Also more recent representations based on CNN activations are non-negative after being passed through rectified linear units. Thus, kernel functions particularly tailored to histogram characteristics are well applicable to a wide range of computer vision scenarios.

Exploiting HIK for efficient GP models has been inspired by several previous works also exploiting HIK properties to speed up kernel SVM learning and prediction (Maji et al. 2008; Wu 2010; Wang et al. 2012). Our first contribution is to transfer their insights to Bayesian methods, which allows for fast multiplications of the kernel matrix $\boldsymbol{K}$ with an arbitrary vector $\boldsymbol{v}$. Thereby, solving the GP inference equations is efficiently possible using iterative solvers.

We then go beyond pure classification aspects and provide efficient techniques for marginal likelihood optimization, variance prediction, and online learning. In particular, we demonstrate that hyperparameter optimization with the complete GP marginal likelihood can be performed with reduced computational costs by exploiting an upper bound for the log-determinant of the kernel matrix. We prove that our bound, which is a modification of the one provided by Bai and Golub (1997), indeed specifies an upper limit and show how to calculate it efficiently. For predictive variance estimation, we present several approximation methods with different asymptotic times and accuracies. The memory and runtime requirements of our methods are sub-quadratic allowing for scalability.

The main contributions of this article can thus be summarized as follows:

(1) We show how to perform learning and inference in a Bayesian manner with Gaussian processes and HIK for a large number of training examples.
(2) We introduce a technique for optimizing hyperparameters based on efficient evaluation of the GP marginal likelihood.
(3) We demonstrate how to estimate and approximate the GP predictive variance and show the implications for active learning.
(4) We derive efficient update routines for online learning, which are a pre-requisite for active learning.

In addition to theoretical derivations, we also empirically validate our techniques for image categorization, incremental learning, and active learning tasks in several experiments. For categorization, we use GP regression with binary labels in a one-vs-all manner similar to Kernel-SVM. Since our experiments are centered on visual object recognition, the application of HIK as a similarity measure is well suited due to the commonly used histogram representations of images. We further use parameterized versions of the kernel that are more flexible, e.g., by appropriate non-linear scaling or individual weighting of histogram elements. Our methods for hyperparameter optimization are useful to handle this increased flexibility and circumvent the necessity of techniques like cross-validation.

The article is based on previous conference publications. In detail, we presented the efficient GP multi-class classification with regression and hyperparameter optimization at ECCV (Rodner et al. 2012). Furthermore, we published the predictive variance approximation techniques and the incremental learning aspects at ACCV (Freytag et al. 2012b). In this article, we go beyond these individual publications and present all aspects in a coherent view. In addition, we extended them with detailed mathematical proofs, an error analysis of the quantization method, adaptive quantization as well as multiple additional experimental results and comparisons. An application of our approach towards semantic segmentation was demonstrated in an ICPR paper (Freytag et al. 2012a).

Structure of this Article We start by reviewing relevant literature for this article in Sect. 2. The Gaussian process framework for regression, classification, and hyperparameter optimization is reviewed in Sect. 3. In Sect. 4, we define the histogram intersection kernel as well as its parameterized versions and review how efficient kernel multiplications can be done. Furthermore, we present how a quantization approach leads to further improvements regarding computational speed.

Based on these foundations, we describe how to efficiently learn and test a GP model for multi-class classification with Gaussian noise models using HIK in Sect. 5. Fast hyperparameter optimization and speeding up predictive variance estimations are discussed in Sects. 6 and 7, respectively. How our methods can be extended to incremental learning as well as applied for active learning is presented in Sect. 8.

Our findings are complemented by extensive experiments on publicly available computer vision datasets. In Sect. 9, we experimentally prove the suitability of our approach for the tasks of large-scale classification and incremental learning. Efficient hyperparameter optimization with Gaussian processes is evaluated in Sect. 10. We further analyze our variance approximation techniques as well as their application for active learning in Sect. 11. A summary of our findings concludes the article.

2 Related Work

In the following, we review related work for different aspects of this article: learning with histogram intersection

kernels (Sect. 2.1), parameterized kernels (Sect. 2.2), current approaches for fast GP classification and regression (Sect. 2.3), and the application of the GP framework for visual recognition tasks (Sect. 2.4).

2.1 Fast Learning and Classification with HIK

For almost two decades, kernel methods have been popular tools for handling non-linear relations present in real world problems. The idea of kernels is to directly compute scalar products between transformed feature vectors without the necessity of actually computing the transformation. While model complexity and resulting accuracy can be dramatically improved via kernelization of linear algorithms, resulting benefits come at the cost of potentially increased memory consumption and computation time. To overcome these drawbacks, Vedaldi and Zisserman (2010) presented how to approximate the histogram intersection kernel with explicit feature transformations. These transformations can then be directly used in combination with linear classifiers. In contrast, Maji et al. exploited the properties of HIK directly for efficiently calculating SVM decisions (Maji et al. 2008, 2013). When m denotes the number of support vectors and D the number of feature dimensions, their technique scales only with $\mathcal{O}(\log(m)D)$ time compared to $\mathcal{O}(mD)$ for standard SVM inference. Going one step further, Wu (2010, 2012) presented fast SVM training by using the HIK properties to reformulate the SVM dual problem. Similar techniques have been applied later by Wang et al. (2012) for large-scale image similarity calculation. In the present article, we go beyond these works by showing that the special properties of the HIK can be exploited for GP classification with regression, hyperparameter optimization, and variance estimation.

2.2 Generalized HIK and Hyperparameter Optimization

Barla et al. (2003) applied the HIK for image classification and proved it to be a Mercer kernel for images having the same size. Since that time, a lot of improvements on this kernel have been proposed, e.g., HIK with polynomial transformations (Boughorbel et al. 2005) or the weighted multi-level extension known as pyramid match kernel (PMK) by Grauman and Darrell (2007). We show how to further generalize the HIK with order-preserving and positive-valued feature transformations and weights for each dimension. Therefore, our work is similar to the one of Ablavsky and Sclaroff (2011), where a cross-validation procedure is proposed to estimate multiple weights of histogram kernels. In contrast, our hyperparameter optimization is theoretically sound and directly optimizes the data likelihood. We compare our results to the ones achieved by Ablavsky and Sclaroff (2011) and show the resulting benefits of optimizing generalized histogram intersection kernels in Sect. 10.

Our approach for hyperparameter optimization should not be confused with the recent work on Bayesian optimization presented by Snoek et al. (2012). In fact, both approaches are orthogonal. Here, we present how to efficiently compute the marginal log-likelihood of a GP model. Thereby, it can be combined with the optimization method of Snoek et al. (2012) using GP regression for predicting suitable sample points of hyperparameters.

2.3 Fast GP Classification and Regression

Similar to Kernel-SVM, GP classifiers require computation time and memory cubic and quadratic in the number of training examples, respectively. Thus, their direct application to large-scale problems is limited. A growing number of publications tackle this problem by introducing sparse approximations. At the core of these approximations is the assumption of conditional independence between sets of certain examples. These examples could be a selection from the training set or can be learned during training (Quiñonero Candela and Rasmussen 2005). Although these techniques lead to impressive results, the necessary independence assumptions neglect information provided in training and test data. The only work we are aware of tackling full GP inference is the greedy block technique of Bo and Sminchisescu (2012), which circumvents storing the full kernel matrix in memory. However, kernel values have to be calculated explicitly, which is not necessary in our case. In Sect. 9, we empirically show that their method can be improved by orders of magnitude in computation time.

A complementary approach has been presented by Urtasun and Darrell (2008) who apply local learning to tackle the long training time and high memory demands of standard GP regression. In particular, they learn models from the k nearest neighbors selected by evaluating kernel distances. In contrast, our approach is sublinear during testing and does not involve an additional training step for each test example.

2.4 Applications of the GP Framework: Active Learning and Beyond

In active learning scenarios, we have been given a set of unlabeled examples, an expert who is willing to annotate a few of these examples, and a classifier which shall be trained. Thus, we search for the instances within the given set which are as informative as possible after being labeled and added to the model.

Active learning with the Gaussian process framework has been introduced by Kapoor et al. (2010). The authors present three different selection strategies based on the predictive mean, variance, and a combination of both. Thus, our tech-

niques presented in this article allow for using their ideas in large-scale scenarios with a large number of unlabeled examples and an increasing size of the training set. Note that this also holds for our recently introduced active learning methods that involve the computation of expected changes in model parameters for GP regression to develop a new active learning criterion (Freytag et al. 2013) as well as a generalization based on measuring the differences in expected model outputs (Freytag et al. 2014a; Käding et al. 2015).

Besides active learning, GPs serve as easy-to-use probabilistic model in a range of other applications. Three exemplary scenarios are their application for detector adaptation (Vázquez et al. 2014), for super-resolution tasks (He and Siu 2011), or for human pose inference (Urtasun and Darrell 2008).

3 GP Regression and Hyperparameter Optimization

In this section, we briefly introduce the Gaussian process framework for regression and classification to ensure theoretical basics important for understanding this article. An experienced reader may directly jump to Sect. 4 for a review on exploiting HIKs for efficient inference in general or directly to Sect. 5 for a description of efficient GP inference using HIKs.

3.1 Gaussian Process (Label) Regression

Let Ω be the space of all possible input data, e.g., D-dimensional feature vectors, which are often L_1-normalized in case of histogram representations. Furthermore, let $\mathcal{Y}$ be the space of possible targets. For now, let us focus on binary classification with $\mathcal{Y} = \{-1, 1\}$. Multi-class scenarios are discussed in Sect. 3.3. Given N training examples $\boldsymbol{x}_i \in \boldsymbol{X} \subset \Omega$ and corresponding binary labels $y_i \in \{-1, 1\}$, we would like to predict the label y_* of an unseen example $\boldsymbol{x}_* \in \Omega$.

The relationship between inputs and outputs can be modeled by a latent function f and an additional noise process $y = f(\boldsymbol{x}) + \varepsilon$. Well known frequentist approaches such as SVM would now seek for a single function f which optimizes some criterion such as the regularized risk (Vapnik 1998). In contrast, a Bayesian approach assumes that f is a sample of a stochastic process F and inference requires marginalization over all possible realizations. A specific choice of a stochastic process is a Gaussian process $\mathcal{GP}$. We can thus express our assumptions as a GP prior with zero-mean and covariance (kernel) function $\kappa(\cdot, \cdot)$, i.e., $f \sim \mathcal{GP}(0, \kappa)$. Furthermore, we assume that labels y_i are conditionally independent from $\boldsymbol{x}_i$ given $f(\boldsymbol{x}_i)$.

We follow Kapoor et al. (2010) and solve a given binary classification problem as a regression problem where labels y_i are treated as real-valued function values instead of discrete labels. This is very much related to Kernel-SVM but with an L_2-loss instead of a Hinge loss. Since it can be interpreted as kernelized least-squares regression, this technique is also known as label regression. In practical applications, it has proved to be useful for classification and it outperforms approximate inference techniques like Laplace approximation with more sophisticated noise models in most cases (Kapoor et al. 2010; Rodner et al. 2010; Kemmler et al. 2010).

For the noise process ε in label regression, a simple additive Gaussian noise model with variance σ_{n}^2 is used:

$$p(y_i \mid f_i) = \mathcal{N}(y_i \mid f_i, \sigma_{\mathsf{n}}^2). \tag{1}$$

An advantage of the Gaussian noise model is that the GP assumptions lead to analytic solutions of the involved marginalizations. In consequence, they allow for directly predicting the expectation μ_* as well as the variance σ_*^2 of the posterior distribution for the label y_* of a new example $\boldsymbol{x}_*$ (Rasmussen and Williams 2006):

$$\mu_* = \boldsymbol{k}_*^{\mathrm{T}} (\boldsymbol{K} + \sigma_{\mathsf{n}}^2 \cdot \boldsymbol{I})^{-1} \boldsymbol{y} = \boldsymbol{k}_*^{\mathrm{T}} \boldsymbol{\alpha}, \tag{2}$$

$$\sigma_*^2 = k_{**} - \boldsymbol{k}_*^{\mathrm{T}} (\boldsymbol{K} + \sigma_{\mathsf{n}}^2 \cdot \boldsymbol{I})^{-1} \boldsymbol{k}_* + \sigma_{\mathsf{n}}^2. \tag{3}$$

Here, the vector $\boldsymbol{k}_*$ contains the kernel values $(\boldsymbol{k}_*)_i = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*)$ corresponding to the test example $\boldsymbol{x}_*$, $\boldsymbol{K}$ is the kernel matrix of the training data with $(\boldsymbol{K})_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, $k_{**} = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*)$ is the prior variance of $\boldsymbol{x}_*$, and $\boldsymbol{y}$ is the vector containing all training labels. Although noted before, let us again emphasize that when we write "GP classification" in this article, we always refer to the label regression technique in Eq. (2) as presented by Kapoor et al. (2010). Thereby, we are able to estimate the predictive variance for classification settings which is one important benefit of the Bayesian framework. Another benefit arises when dealing with parameterized kernel functions, which is reviewed in the next section.

3.2 Hyperparameter Optimization

Kernel functions often depend on hyperparameters $\boldsymbol{\eta}$, which have an important impact on the resulting classification model. In contrast to SVM techniques, the GP framework allows for finding their optimal values by marginal likelihood maximization instead of expensive cross-validation. As shown by Rasmussen and Williams (2006) the negative marginal log-likelihood for GP regression models can be expressed as:

$$-\log p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\eta}) = \frac{1}{2}\boldsymbol{y}^{\mathrm{T}}\left(\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}\right)^{-1}\boldsymbol{y} + \frac{1}{2}\log\det\left(\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}\right) + \frac{N}{2}\log 2\pi. \quad (4)$$

We introduced the short-hand $\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}} = \left(\boldsymbol{K}_{\boldsymbol{\eta}} + \sigma_{\mathrm{n}}^2 \cdot \boldsymbol{I}\right)$ as the parameterized kernel matrix having the noise variance σ_{n}^2 added to the main diagonal. It should be mentioned here that for continuous hyperparameters and differentiable kernel functions, it is even possible to use the derivative of the negative marginal log-likelihood for the optimization (Rasmussen and Williams 2006).

3.3 Multi-class Classification

As common for SVM classifiers, GP multi-class classification can be done by utilizing the one-vs-all technique as shown by Kapoor et al. (2010). For each class m, a binary classifier with label regression (Nickisch and Rasmussen 2008) is trained which uses all images of m as positive examples and all remaining examples as negatives. Classification is then done by returning the class with largest predictive mean estimate of the corresponding binary classifier.

We can also perform model selection for the one-vs-all approach by jointly optimizing hyperparameters of all involved binary problems (Kapoor et al. 2010). Thereby, the objective function turns out to be the sum of all binary negative marginal log-likelihoods as given in Eq. (4). In addition, the predictive variance as shown in Eq. (3) is independent of the actual class labels and has therefore to be calculated only once for an arbitrary number of classes.

We can thus summarize that GP models can be seen as useful probabilistic counterparts to SVM classifiers with a range of interesting properties. Thus, efficient inference tools for GP models should be available to computer vision researchers for their "classification tool-boxes", which is one aim of the current article. To tackle this goal, we present in the next section how to exploit fast kernel multiplication for speeding up computation times by orders of magnitude and significantly reduce memory demands.

4 Efficient Kernel Multiplications with Histogram Intersection Kernels

Kernel methods are one of the fundamental tools used to handle the complexity of visual recognition and allow for expressing non-linear relations with otherwise linear models. A possible kernel function often used to compare histogram feature vectors $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^D$ is the histogram intersection kernel:

$$\kappa_{\mathrm{HIK}}(\boldsymbol{x}, \boldsymbol{x}') = \sum_{d=1}^{D} \min(\boldsymbol{x}(d), \boldsymbol{x}'(d)). \quad (5)$$

As shown by Maji et al. (2008) and Wu (2010), this kernel offers two important properties:

(1) for any test input $\boldsymbol{x}_*$, the computation of the inner product $\boldsymbol{k}_*^{\mathrm{T}}\boldsymbol{\alpha}$ between kernel vector $\boldsymbol{k}_*$ and weight vector $\boldsymbol{\alpha}$ of a trained representer model [e.g., GP regression in Eq. (2)] scales only sub-linear with the number of known examples, and
(2) for an arbitrary vector $\boldsymbol{v}$, the matrix-vector product $\boldsymbol{K}\boldsymbol{v}$ between the kernel matrix $\boldsymbol{K}$ and $\boldsymbol{v}$ scales only sub-quadratic.

While the first property allows for efficient inference with representer models, the second property enables reduced computation times for learning [as also noted in (Bottou et al. 2007, Sect. 9.4)]. We refer to these properties as fast kernel multiplications. In consequence, we refer to kernels fulfilling these properties as *fast multiplication kernels*. Note that these kernels should be not confused with multiplicative kernels as introduced by Yuan et al. (2008).

In the following, we review the work of Maji et al. (2008) and Wu (2010). Since the authors presented how fast HIK multiplications can be exploited for fast SVM learning and classification, we put their work into a Gaussian process perspective. In addition, we derive a worst-case bound for quantization errors of the techniques presented by Maji et al. (2008). The section closes by presenting generalizations of currently known histogram intersection kernels important for adaptations such as automated feature scaling or feature relevance determination.

4.1 Fast Kernel Multiplications

As we have seen in Eq. (2), the predictive mean is a weighted sum of kernel values. This property is shared by all representer models such as SVM and GP. The HIK allows for decomposing this sum into two parts (Maji et al. 2008):

$$\begin{aligned}\boldsymbol{k}_*^{\mathrm{T}}\boldsymbol{\alpha} &= \sum_{i=1}^{N}\boldsymbol{\alpha}(i)\sum_{d=1}^{D}\min\left(\boldsymbol{x}_i(d), \boldsymbol{x}_*(d)\right) \\ &= \sum_{d=1}^{D}\Bigg(\sum_{\{i\,:\,\boldsymbol{x}_i(d)<\boldsymbol{x}_*(d)\}}\boldsymbol{\alpha}(i)\,\boldsymbol{x}_i(d) + \ldots \\ &\quad \ldots\boldsymbol{x}_*(d)\sum_{\{j\,:\,\boldsymbol{x}_j(d)\geq\boldsymbol{x}_*(d)\}}\boldsymbol{\alpha}(j)\Bigg). \end{aligned} \quad (6)$$

From Eq. (6) we make the important observation, that for each dimension the predictive mean of Gaussian process

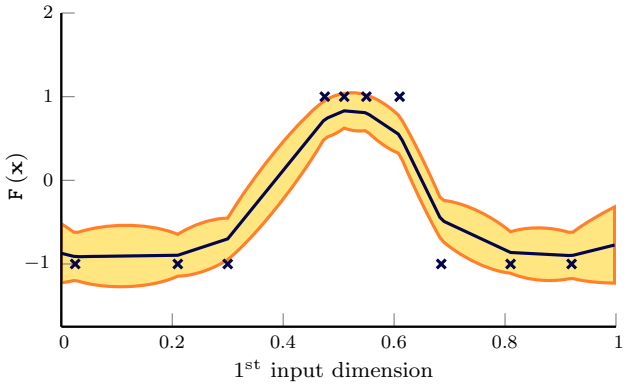

Fig. 1 Piecewise linearity of the regression function when using Gaussian process regression for classification (discrete $\boldsymbol{y}$) together with the histogram intersection kernel: 2-dimensional input vectors $\boldsymbol{x}$ are used but due to the normalization $\|\boldsymbol{x}\|_1 = 1$, we only display the predictive mean (*blue graph*) and confidence areas (*shaded area*) derived from the predictive variance with respect to the first dimension of the input vectors. Training points are shown as *blue crosses* and the noise variance is set to 0.1

regression with HIK is piecewise linear. This property is also visualized in Fig. 1 for a simple toy example.

We can now significantly reduce the computational costs using the following trick. Let us assume given permutations $\boldsymbol{\pi}_d$ which rearrange the training examples such that they are sorted in an ascending order in each dimension d. Then, we can rewrite Eq. (6) as

$$\begin{aligned}\boldsymbol{k}_*^{\mathrm{T}}\boldsymbol{\alpha} = \sum_{d=1}^{D}\Bigg(\underbrace{\sum_{i=1}^{r_d}\boldsymbol{\alpha}\left(\boldsymbol{\pi}_d^{-1}(i)\right)\boldsymbol{x}_{\boldsymbol{\pi}_d^{-1}(i)}(d)}_{\doteq \boldsymbol{A}(d,r_d)} + \ldots \\ \ldots \boldsymbol{x}_*(d)\underbrace{\sum_{i=r_d+1}^{N}\boldsymbol{\alpha}\left(\boldsymbol{\pi}_d^{-1}(i)\right)}_{\doteq \boldsymbol{B}(d,r_d)}\Bigg) \qquad (7)\\ = \sum_{d=1}^{D}\left(\boldsymbol{A}(d,r_d) + \boldsymbol{x}_*(d)\,\boldsymbol{B}(d,r_d)\right), \qquad (8)\end{aligned}$$

with r_d being the number of examples that are smaller than $\boldsymbol{x}_*$ in dimension d. We can precompute the two terms of the linear function during learning and store the values in look-up tables $\boldsymbol{A}$ and $\boldsymbol{B}$ as displayed in Eq. (8). Calculating the scores for test examples can be done by accessing only few elements of $\boldsymbol{A}$ and $\boldsymbol{B}$, in particular one element of both $\boldsymbol{A}$ and $\boldsymbol{B}$ for each dimension.

Given the vector $\boldsymbol{\alpha}$, the resulting computation time for building $\boldsymbol{A}$ and $\boldsymbol{B}$ is dominated by sorting, which requires $\mathcal{O}(ND\log N)$ operations. In terms of memory usage, we only have to store $\mathcal{O}(ND)$ elements in contrast to the kernel matrix of size $\mathcal{O}\left(N^2\right)$. This is especially important for large dataset sizes, for which the kernel matrix would not fit into memory. For calculating the score of a new example, we need $\mathcal{O}(D\log N)$ operations to find the correct position r_d in each dimension and compute the linear function in Eq. (8) using look-up tables $\boldsymbol{A}$ and $\boldsymbol{B}$.

Similar considerations hold for multiplications of an arbitrary vector $\boldsymbol{v} \in \mathbb{R}^N$ with the kernel matrix $\boldsymbol{K}$, which can be done in $\mathcal{O}(ND)$ based on already sorted features in each dimension:

$$\begin{aligned}(\boldsymbol{K}\boldsymbol{v})_i &= \sum_{j=1}^{N}\boldsymbol{v}(j)\cdot\kappa(\boldsymbol{x}_i,\boldsymbol{x}_j)\\ &= \sum_{j=1}^{N}\boldsymbol{v}(j)\sum_{d=1}^{D}\min(\boldsymbol{x}_i(d),\boldsymbol{x}_j(d)). \qquad (9)\end{aligned}$$

From this equation, we observe that we can further exploit sparsity of feature vectors since corresponding terms of zero-valued dimensions vanish.

As we will show in the next sections, the efficient computation of products $\boldsymbol{K}\boldsymbol{v}$ will be an essential part in our overall framework. In detail, we will require it for learning in Sect. 5.1 and for optimizing hyperparameters as shown in Sect. 6. Furthermore, computations of products $\boldsymbol{k}_*^{\mathrm{T}}\boldsymbol{\alpha}$ will allow for efficient classification as shown in Sect. 5.2.

4.2 Quantization of the Feature Space

The previously shown techniques for computing scores $\boldsymbol{k}_*^{\mathrm{T}}\boldsymbol{\alpha}$ already result in valuable savings of required computation times. Nonetheless, they still depend on the number of available training examples. If feature values in dimension d are bounded by $\boldsymbol{x}_*(d) \in [l_d, u_d]$, the evaluation can be further speeded up by quantizing the feature space (Maji et al. 2008) leading to an approximate inference method.

For L_1-normalized histogram features, all elements are obviously bounded by the interval $[0, 1]$. Using a quantization for each dimension with q bins, only q different prototypical outputs $\boldsymbol{p}(k)\quad(1 \le k \le q)$ are possible in each summand of Eq. (7). With already computed tables $\boldsymbol{A}$ and $\boldsymbol{B}$, we can proceed with building a final look-up table $\boldsymbol{T}$ of dimension $D \times q$:

$$\boldsymbol{T}(d,k) = \boldsymbol{A}(d,r_d) + \boldsymbol{p}(k)\cdot\boldsymbol{B}(d,r_d), \qquad (10)$$

with r_d being the number of examples that are smaller than $\boldsymbol{p}(k)$ in dimension d. Since permutations $\boldsymbol{\pi}_d$ are already computed, this step requires only $\mathcal{O}(D\max(q,N))$ operations.

As a result, the time spent for evaluating the score of a new test example decreases to $\mathcal{O}(D)$ if the quantizer works in $\mathcal{O}(1)$. Consequently, for a given number of dimensions, the score of a new test example can be computed in constant

time independent of N. It should be noted that in the case of quantization, GP regression can be restated as piecewise Bayesian linear regression.

Adaptive Quantization In contrast to previous works, we use an adaptive quantization for every single dimension of the input feature vectors. The necessity of our adaptive quantization becomes apparent when we replace BOW features by CNN activations. For the latter, the common activation strength typically varies heavily between dimensions. Furthermore, L_1-normalized activations come without theoretical motivation. Thus, neither are all values guaranteed to be smaller than one as for histogram entries, nor is a quantization equal for all dimensions suitable. We therefore compute the maximum value u_d in each feature dimension for the training set. For each dimension, we then use $[0, u_d]$ to define the bounds for a uniform quantization.

Quantization Error Analysis The quantization trick reduces the runtime during testing significantly. However, the question remains how much the approximation affects the classification scores. Therefore, we study the error induced by this trick when computing the predictive mean, an analysis that has not been performed in previous works. The proofs of the results are given in Appendix. The results are not restricted to Gaussian process regression and also hold for support vector machines, because of the decision made with a weighted sum of kernel values.

Theorem 1 (Worst-case error analysis) *If a quantization with a maximum quantization error of ε_q is used as well as L_1-normalized features, the error of the quantized predictive mean $\tilde{\mu}_*$ can be bounded as follows:*

$$|\mu_* - \tilde{\mu}_*| \leq \frac{D \cdot \varepsilon_q}{2} \cdot \|\boldsymbol{\alpha}\|_1. \tag{11}$$

Proof See Appendix. □

The L_1-term of the weights $\boldsymbol{\alpha}$ does not depend on the quantization, because the quantization trick is only applied when a new test input is given and the position in the sorted list has to be determined in constant time. However, the term can be further bounded for SVM models by using $\|\boldsymbol{\alpha}\|_1 \leq C \cdot N$:

$$|\mu_* - \tilde{\mu}_*| \leq \frac{D \cdot \varepsilon_q}{2} \cdot C \cdot N, \tag{12}$$

where C is the trade-off parameter used in the soft-margin version (Schölkopf and Smola 2001). For GP regression, we can use the bounds derived by Rodner (2011, p. 64), which lead to:

$$|\mu_* - \tilde{\mu}_*| \leq \frac{D \cdot \varepsilon_q}{2} \cdot \frac{N}{\sigma_\mathsf{n}^2}. \tag{13}$$

Conclusions of Theorem 1 What do we learn from the previous analysis? First, we observe that the upper bound of the error depends linearly on the dimension and the number of training examples. Furthermore, also the strength of regularization (for SVM determined by C and for GP regression by the noise variance σ_n^2) influences the error induced by the quantization. For a strong regularization (low C or high σ_n^2), the error decreases.

4.3 Very General Histogram Intersection Kernels

In the previous sections, we restricted our analysis to the standard HIK as introduced in Eq. (5). To increase the kernel functions flexibility, Boughorbel et al. (2005) have shown that the HIK equipped with any positive valued function $g(\cdot)$:

$$\kappa_{\text{GHIK}}\left(\boldsymbol{x}, \boldsymbol{x}'\right) = \sum_{d=1}^{D} \min\left(g\left(\boldsymbol{x}\left(d\right)\right), g\left(\boldsymbol{x}'\left(d\right)\right)\right) \tag{14}$$

still remains a positive definite kernel. The obvious question is whether efficient calculations as shown previously also hold for the generalized versions of HIK?

In fact, if $g(\cdot)$ is an automorphism, the relative order of the training elements stays valid after evaluating $g(\cdot)$. Therefore, the proposed techniques can also be applied to these generalized variants of the HIK which we denote with GHIK in the remainder of the article. Note that we can even use the same quantization by storing the original feature values. Two common examples are the polynomial feature transform:

$$g_{|\cdot|,\eta}\left(\boldsymbol{x}\left(d\right)\right) = \left|\boldsymbol{x}\left(d\right)\right|^{\eta}, \quad \eta \in \mathbb{R}_+, \tag{15}$$

and the exponential transformation:

$$g_{\exp,\eta}\left(\boldsymbol{x}\left(d\right)\right) = \frac{\exp(\eta\left|\boldsymbol{x}\left(d\right)\right|) - 1}{\exp(\eta) - 1}, \quad \eta \in \mathbb{R}_+. \tag{16}$$

In the remaining sections, we refer to them as HIK-POLY and HIK-EXP. Interestingly, Eq. (14) even holds if we consider functions g_d specifically parameterized for each dimension. Thereby, we can individually weight each feature dimension:

$$g_{\text{ard},\boldsymbol{\eta}}\left(\boldsymbol{x}\left(d\right)\right) = \boldsymbol{\eta}\left(d\right) \cdot \boldsymbol{x}\left(d\right), \quad \boldsymbol{\eta} \in \mathbb{R}_*^D. \tag{17}$$

In Sect. 6, we present how to optimize hyperparameters $\boldsymbol{\eta}$ of generalized HIKs even for large-scale training data.

5 Efficient GP Multi-Class Classification with GHIK

In this section, we demonstrate that learning and testing a GP model can be performed efficiently when using GHIKs.

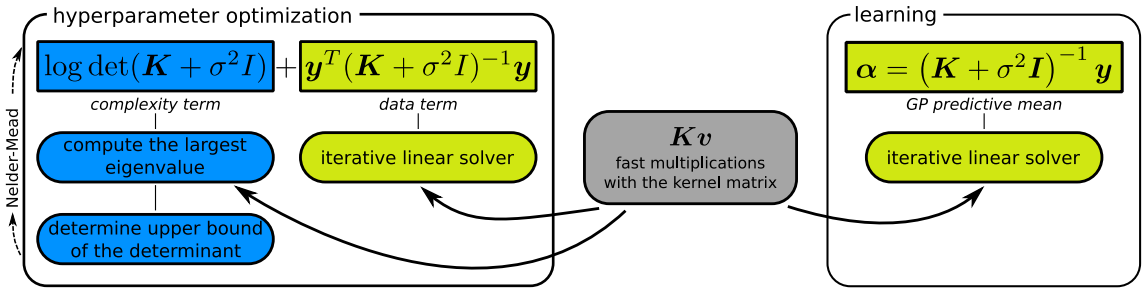

Fig. 2 Main outline of our approach for GP classification and hyperparameter optimization using fast multiplications with the kernel matrix. Details are given in Sects. 5 and 6

Most of our algorithms are also applicable for general fast multiplication kernels, but since GHIK is the only practical family of kernels known to fit to this class, we focus on the GHIK in our presentation of the algorithms. This theoretical investigation is complemented by experimental results given in Sect. 9 and following, where we tackle several applications such as classification of real-world large-scale datasets, model regularization, and feature relevance determination.

As shown in Eq. (2), inference with a GP model requires two steps: (1) solving the linear equation system $\tilde{\boldsymbol{K}}_\eta \cdot \boldsymbol{\alpha} = \boldsymbol{y}$ and (2) calculating the scalar product $\boldsymbol{k}_*^{\mathrm{T}}\boldsymbol{\alpha}$. Step (1) has to be done only once for each class of a given training set $\boldsymbol{X}$ and is known as learning. In contrast to that, the second step is used to test the learned model in order to infer labels for new test data. For transferring the techniques of Maji et al. (2008) to GP inference, we need to handle both steps. Let us start with step 1.

5.1 Step 1: Efficient Learning

For the training phase in step 1, we notice that storing the full kernel matrix is impossible for large-scale datasets. Furthermore, applying a Cholesky decomposition with a runtime of $\mathcal{O}\left(N^3\right)$ is far from being practical. However, the HIK explicitly allows for multiplications with the kernel matrix in linear time $\mathcal{O}(ND)$. Therefore, an iterative linear solver can be used to tackle the learning step and only needs to perform several cheap multiplications with the kernel matrix.

Wu (2010) used a coordinate descent method to solve the quadratic program related to SVM learning. In contrast, our experiments show that a linear conjugate gradients (CG) method converges faster for GP problems. Note that in absence of round-off errors, we obtain the exact solution after N iterations [see also (Hestenes and Stiefel 1952)]. In practice, we can even stop the iteration significantly earlier, e.g., when the maximum norm of the residual drops below a specified threshold.

Let us analyze the resulting asymptotic cost to compare against the GP baseline. To this end, let M denote the number of classes and let T_1 be the number of iterations the CG method performs, which depends on the condition number of the kernel matrix (Nocedal and Wright 2006). Among others, the condition of the kernel matrix itself depends on N and can be corrected by adapting the regularization parameter σ_{n}^2. Since the binary label vectors differ for each class, we need to compute M weight vectors $\boldsymbol{\alpha}^{(1)}, \ldots, \boldsymbol{\alpha}^{(M)}$. As previously noted, solving the resulting linear equation system using an iterative linear solver is possible in linear time, which leads to the first term $\mathcal{O}(DNT_1M)$. A second term $\mathcal{O}(DN\log N)$ arises from the effort for sorting all N examples in every dimension.

In summary, we require $\mathcal{O}(ND(T_1M+\log N))$ operations for learning. Note that in practice, we often observe sparse features which leads to a significant reduction of the necessary computation times for this step. We also see that the runtime performance of our method is linear in the number of classes M allowing for scalability towards large-scale scenarios with hundreds or even thousands of classes.

5.2 Step 2: Efficient Testing

After estimating the coefficients $\boldsymbol{\alpha}$, the test step only involves evaluating inner products $\boldsymbol{k}_*^{\mathrm{T}}\boldsymbol{\alpha}$, which can be done in logarithmic time. Note that we can further reduce the asymptotic cost to constant time by applying the quantization idea of Maji et al. (2008) as reviewed in Sect. 4.

In summary of this section, we visualized the interplay of all steps of our approach in Fig. 2 including hyperparameter optimization as explained in the next section. An overview of asymptotic computation times and memory demand for our approach is finally given in Table 1.

6 Large-Scale Hyperparameter Optimization

While the previous section dealt with efficient large-scale classification, we are now interested in optimally adapting our system to a specific task. Here, we realize adaptations by optimizing involved hyperparameters. Due to our proba-

Table 1 Overview of asymptotic runtimes for learning, testing, and optimization of hyperparameters for baseline GP compared to our approach

Step	Asymptotic computation time		
	GP baseline	GP + (G)HIK	GP + (G) HIK + Quant.
Learning	$\mathcal{O}\left(\boldsymbol{N}^3 + M\boldsymbol{N}^2\right)$	$\mathcal{O}\left(D\boldsymbol{N}\left(\log \boldsymbol{N} + T_1 M\right)\right)$	$\mathcal{O}\left(D\boldsymbol{N}\left(\log \boldsymbol{N} + T_1 M\right)\right)$
Hyperp. opt.	$\mathcal{O}\left(T_2\left(\boldsymbol{N}^3 + M\boldsymbol{N}^2\right)\right)$	$\mathcal{O}\left(T_2 \boldsymbol{N} M D T_1\right)$	$\mathcal{O}\left(T_2 \boldsymbol{N} M D T_1\right)$
Testing	$\mathcal{O}\left(MD\boldsymbol{N}\right)$	$\mathcal{O}\left(MD\log \boldsymbol{N}\right)$	$\mathcal{O}\left(MD\right)$
Memory	$\mathcal{O}\left(\boldsymbol{N}^2\right)$	$\mathcal{O}\left(D\boldsymbol{N}\right)$	$\mathcal{O}\left(D\max\left(q, \boldsymbol{N}\right)\right)$

The parameter D denotes the number of dimensions, M the number of classes, and T_1 and T_2 the number of iterations used for the linear solver and the optimizer, respectively. We highlight the number of training examples N with bold font

bilistic model, this optimization can be done by minimizing the negative GP log-likelihood as given in Eq. (4).

Exact optimization of the negative GP log-likelihood is very time consuming and intractable for large-scale datasets. We show how to solve this drawback by optimizing a proxy function instead which can be evaluated efficiently. A theoretical analysis reveals that our proxy function is indeed an upper bound of the exact negative GP log-likelihood and thus results in similar optima.

6.1 An Upper Bound of the Log-Determinant

As we reviewed in Eq. (4), computing the negative log-likelihood mainly requires the evaluation of two terms: a quadratic data term $\boldsymbol{y}^{\mathrm{T}} \boldsymbol{K}_{\boldsymbol{\eta}}^{-1} \boldsymbol{y}$ and a complexity term $\log\det\left(\boldsymbol{K}_{\boldsymbol{\eta}}\right)$. Since the data term involves solving the same linear system as required for learning, we can compute it efficiently using the techniques presented previously. In contrast to that, the complexity term requires the determinant of the kernel matrix, which is a costly algebraic operation even with fast HIK multiplications (Yuster 2008). Due to this reason, we develop an upper bound of the log determinant which will ultimately lead to the upper bound of the negative log-likelihood.

The derived bound is based on the results of Bai and Golub (1997), which turns out to be efficiently computable with HIKs. Let us therefore assume that for a given positive definite matrix $\boldsymbol{M} \in \mathbb{R}^{N \times N}$, all eigenvalues λ_i can be bounded by $0 < \lambda_i \leq \beta$. Then, an upper bound of the log-determinant is given by:

$$\log\det(\boldsymbol{M}) \leq \left[\log\beta,\ \log\bar{t}\right] \begin{bmatrix} \beta & \bar{t} \\ \beta^2 & \bar{t}^2 \end{bmatrix}^{-1} \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} \tag{18}$$

$$\doteq \mathrm{ub}(\beta, \mu_1, \mu_2) \tag{19}$$

$$\text{with} \quad \bar{t} = \frac{\beta\mu_1 - \mu_2}{\beta n - \mu_1}, \tag{20}$$

where $\mu_1 = \mathrm{tr}\left(\boldsymbol{M}\right)$ is the trace of the matrix and $\mu_2 = \|\boldsymbol{M}\|_F^2$ is the Frobenius norm (Bai and Golub 1997).

While Bai and Golub (1997) proved the correctness of that bound, its tightness is what finally matters in practical applications. From this point of view, it is interesting to note that the bound is tight for regularized rank-1 matrices $\boldsymbol{M} = \boldsymbol{u}\boldsymbol{u}^{\mathrm{T}} + \tau\boldsymbol{I}$. This fact arises as a direct generalization of the result from Bai and Golub (1997) on Pei matrices. Fortunately, kernel matrices computed on common datasets are often of low rank as observed by Williams and Seeger (2000). Thus, we can expect that the bound can indeed offer sufficient accuracy in many scenarios. We give an empirical proof for this statement in Sect. 10.1.

How to Efficiently Evaluate the Upper Bound Function To calculate the bound given in Eq. (19) for the regularized kernel matrix $\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}$, we need its largest eigenvalue $\lambda_{\max}$, its trace μ_1, and its squared Frobenius norm μ_2. We first compute the largest eigenvalue $\lambda_{\max}$ with the Arnoldi iteration, which only requires matrix vector products. In our experiments, the algorithm needed approximately $T_3 \sim 10$ steps until convergence with high accuracy for various settings.

The trace of the regularized kernel matrix can be decomposed into two terms:

$$\begin{aligned} \mathrm{tr}(\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}) &= \sum_{i=1}^{N} \left(\kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) + \sigma_{\mathsf{n}}^2\right) \\ &= \sigma_{\mathsf{n}}^2 \cdot N + \sum_{i=1}^{N}\sum_{d=1}^{D} \min(\boldsymbol{x}_i(d), \boldsymbol{x}_i(d)). \end{aligned} \tag{21}$$

The first part arises from the noise model, whereas the sum of self-similarities constitutes the second part. For the specific choice of HIK, the latter one is equal to the sum of all feature values:

$$\mathrm{tr}(\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}) = \sigma_{\mathsf{n}}^2 \cdot N + \sum_{i=1}^{N}\sum_{d=1}^{D} \boldsymbol{x}_i(d). \tag{22}$$

If we use L_1-normalized histograms with $\|\boldsymbol{x}_i\|_1 = 1$, this further simplifies to $\mathrm{tr}(\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}) = (\sigma_{\mathsf{n}}^2 + 1) \cdot N$. Note that similar derivations hold for GHIKs. As we can see, incorporating

prior knowledge about kernels and features helps for speeding up the computations.

However, the squared Frobenius norm of $\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}$ is not directly available. Nonetheless, we can approximate it as shown next. To this end, let M again denote the number of classes of the classification task and let λ_i be the ith largest eigenvalue of the regularized kernel matrix such that $\lambda_1 \geq \cdots \geq \lambda_N$ are sorted in decreasing order. Then, an intuitive approximation is $\mu_2 = \sum_{i=1}^{N} \lambda_i^2 \geq \sum_{i=1}^{M} \lambda_i^2 = \tilde{\mu}_2$. The motivation for this approximation is as follows: if we have M classes with very compact clusters and large distances between each other, the kernel matrix should obey a simple block structure of rank M leading to M non-zero eigenvalues and hence $\mu_2 \approx \tilde{\mu}_2$.

Due to the fact that our approximation of μ_2 is a lower bound of $\|\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}\|_F^2$, the necessary computations in Eq. (19) are still well-defined. We verify in the next section that we still have a proper upper bound of the log-determinant. The final upper bound is:

$$\log\det(\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}) \leq \mathrm{ub}\left(\lambda_{\max},\ \mathrm{tr}\left(\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}\right),\ \sum_{i=1}^{M} \lambda_i^2\right). \tag{23}$$

With our experimental results in Sects. 9.3, 10.1, and 10.4, we show how to successfully utilize the resulting upper bound of the negative GP log-likelihood for hyperparameter optimization.

6.2 Proof of the Upper Bound in Case of Frobenius Norm Approximation

So far, we have proposed to use a lower bound for the Frobenius norm of $\tilde{\boldsymbol{K}}_{\boldsymbol{\eta}}$ based on the sum of the M largest eigenvalues to avoid expensive computations of the original negative log-likelihood. In the following, we prove that we are able to obtain a valid upper bound of the log-determinant with the bound of Bai and Golub (1997) even when using a lower bound of the Frobenius norm. Our proofs are completely algebraic and do not require knowledge of the Gaussian quadrature techniques used in Bai and Golub (1997). First of all, we show the validity of the modified upper bound for $\beta = 1$. The proofs of the results are given in Appendix.

Lemma 1 (Monotonicity for $\beta = 1$) *Let $\tilde{\mu}_2$ with $0 < \tilde{\mu}_2 \leq \mu_2$ be a lower bound of the squared Frobenius norm of a regular positive definite matrix $\boldsymbol{M}$, e.g., $\tilde{\mu}_2 = \sum_{i=1}^{M} \lambda_i^2$ with $M < N$. Then the following holds for every positive definite matrix $\boldsymbol{M}$ with $\mu_1 = \mathrm{tr}(\boldsymbol{M})$ and $\beta = 1$ being the largest eigenvalue of $\boldsymbol{M}$:*

$$\mathrm{ub}(1, \mu_1, \tilde{\mu}_2) \geq \mathrm{ub}(1, \mu_1, \mu_2). \tag{24}$$

The next lemma shows that scaling the matrix $\boldsymbol{M}$ with $\gamma > 0$ leads to an additive constant in the bound, which is independent of μ_1 and μ_2. This constant is equivalent to the one occurring in $\log\det(\gamma \boldsymbol{M}) = \log\det(\boldsymbol{M}) + N \log\gamma$, therefore, the quality of the bound is invariant with respect to γ. Note that the squared Frobenius norm scales with γ^2 and $\bar{t}$ with γ (see Eq. 20).

Lemma 2 (Multiplicative scaling) *For all suitable parameters β, μ_1, and μ_2 of a positive definite matrix and every positive factor $\gamma > 0$, the following holds:*

$$\mathrm{ub}(\gamma\beta, \gamma\mu_1, \gamma^2\mu_2) = \mathrm{ub}(\beta, \mu_1, \mu_2) + N \cdot \log\gamma. \tag{25}$$

The last step is to combine both lemmas, which leads directly to the validity of the bound of Bai and Golub for our Frobenius norm approximation:

Theorem 2 (Upper bound with $\tilde{\mu}_2$) *Let $\boldsymbol{M} \in \mathbb{R}^{N\times N}$ be a positive definite matrix with trace μ_1, squared Frobenius norm μ_2, and upper bound β for the eigenvalues (e.g., the largest eigenvalue). If $\tilde{\mu}_2$ is a lower bound of μ_2, the following holds:*

$$\log\det(\boldsymbol{M}) \leq \mathrm{ub}(\beta, \mu_1, \mu_2) \leq \mathrm{ub}(\beta, \mu_1, \tilde{\mu}_2). \tag{26}$$

With this theorem on hand, we are able to efficiently optimize hyperparameters even in large-scale scenarios as validated in the experimental sections.

6.3 Optimization Technique

The actual optimization is carried out with a method that does not require any gradient information, because calculating the gradient of the log-likelihood or the gradient of our upper bound is still a costly operation. A popular technique for this task is the downhill-simplex method, which is also known as Nelder–Mead method (Nelder and Mead 1965). Note that any other black-box optimization method could be applied as well. In experimental evaluations, however, we stick to the downhill-simplex technique.

7 Estimating the GP Predictive Variance

Up to now, we only considered fast computations of the predictive mean derived from GP regression. However, in many scenarios such as active learning or novelty detection it is important to get an estimate for the uncertainty of the prediction as well. The uncertainty is mostly measured in terms of class entropy. For Gaussian distributions, it is directly related to the variance. Due to this reason, we develop methods to efficiently compute the GP predictive variance also in large-scale scenarios.

As presented in Eq. (3), the predictive variance σ_*^2 depends on three terms. From these three terms, the a-priori variance is formed by the first term and the third term, i.e., by $k_{**} = \kappa(\boldsymbol{x}_*, \boldsymbol{x}_*)$ known as self-similarity and the noise variance σ_n^2. Thus, there are no previously known training examples considered so far. In contrast, the second term reduces this a-priori variance based on the similarities between test example $\boldsymbol{x}_*$ and training examples $\boldsymbol{X}$.

To efficiently compute σ_*^2, we start in Sect. 7.1 by applying the techniques for fast classification introduced in Sect. 5.2. However, since the second term is a quadratic form instead of a linear form in $\boldsymbol{k}_*$, these computations are not highly efficient. Therefore, we further show how to approximate the second term in an efficient manner using fast kernel evaluations (Sects. 7.2 and 7.3).

7.1 PUP: Precise Uncertainty Prediction

Naively computing the predictive variance for a single test example $\boldsymbol{x}_*$ involves three steps:

(1) explicitly computing the kernel vector $\boldsymbol{k}_*$,
(2) solving $\boldsymbol{\alpha}_* = \tilde{\boldsymbol{K}}^{-1}\boldsymbol{k}_*$ specific for $\boldsymbol{x}_*$, and
(3) computing the inner product of $\boldsymbol{k}_*$ and $\boldsymbol{\alpha}_*$.

For step 1, we require $\mathcal{O}(ND)$ operations to explicitly evaluate the kernel function for all examples and dimensions. After that, we can apply an iterative linear solver in step 2 to compute $\boldsymbol{\alpha}_*$. As noted previously, this requires $\mathcal{O}(NDT_1)$ operations. Finally, we can compute the product of $\boldsymbol{k}_*$ and $\boldsymbol{\alpha}_*$ in $\mathcal{O}(N)$ operations to obtain the desired data term. Since no approximation is involved, we refer to this method as Precise Uncertainty Prediction (PUP).

In total, we need $\mathcal{O}(NDT_1)$ operations to compute the exact predictive variance for an unseen example during testing. However, since T_1 is implicitly related to N, the resulting runtime might be too slow for large-scale applications. In addition, the exact values of classification uncertainties are not even required in certain scenarios. Active learning is one example, where only the relative order of uncertainty values is important. For these scenarios, we develop efficient approximations as shown in the following.

7.2 FAPU: Fine Approximation of the Predictive Uncertainty

To obtain efficient approximations of the predictive variance, we start by considering fundamental properties of the involved computations. Since the regularized kernel matrix $\tilde{\boldsymbol{K}}$ is symmetric and positive definite, the same holds for its inverse. Therefore, we can use upper and lower bounds for quadratic forms to obtain suitable approximations for σ_*^2 (see Appendix for details).

To this end, let $\boldsymbol{M} \in \mathbb{R}^{N\times N}$ be a positive definite matrix. Then, linear algebra provides us with the following lower bound on the quadratic form of $\boldsymbol{M}$ for any vector $\boldsymbol{x} \in \mathbb{R}^N$:

$$\begin{aligned}\boldsymbol{x}^\mathrm{T}\boldsymbol{M}\boldsymbol{x} \geq & \sum_{i=N-k+1}^{N} \lambda_i \tilde{\boldsymbol{x}}(i)^2 \\ & + \lambda_{N-k}\left(||\boldsymbol{x}||^2 - \sum_{j=N-k+1}^{N} \tilde{\boldsymbol{x}}(j)^2\right),\end{aligned} \tag{27}$$

where $\tilde{\boldsymbol{x}}(j)$ is the projection of $\boldsymbol{x}$ onto the jth eigenvector of $\boldsymbol{M}$. With the parameter k, we can influence the tightness of the bound which is exact for the extreme case of $k = N$. As before, each variable λ_j denotes the jth largest eigenvalue of $\boldsymbol{M}$.

We can now apply the lower bound of Eq. (27) to the data term of the predictive variance given in Eq. (3) by instantiating $\boldsymbol{M} = \tilde{\boldsymbol{K}}^{-1}$:

$$\begin{aligned}\sigma_*^2 \leq & k_{**} - \left(\sum_{j=N-k+1}^{N} \lambda_j \tilde{\boldsymbol{k}}_*(j)^2\right) \\ & - \lambda_{N-k}\left(||\boldsymbol{k}_*||^2 - \sum_{j=N-k+1}^{N} \tilde{\boldsymbol{k}}_*(j)^2\right) + \sigma_\mathrm{n}^2.\end{aligned} \tag{28}$$

While this result looks promising, we spent major efforts in previous sections to circumvent explicit computations of the inverse kernel matrix. In consequence, we can not access corresponding eigenvalues or eigenvectors directly. Fortunately, linear algebra provides us with relations between eigenvalues and eigenvectors of a matrix $\boldsymbol{M}$ and its inverse. In fact, for any symmetric and positive definite matrix $\boldsymbol{M}$, eigenvalues λ of $\boldsymbol{M}$ are in relationship with eigenvalues ξ of $\boldsymbol{M}^{-1}$ via $\lambda_j = \frac{1}{\xi_{N-j+1}}$. Furthermore, the eigenvector of $\boldsymbol{M}$ corresponding to λ_i is the same as the eigenvector of $\boldsymbol{M}^{-1}$ belonging to ξ_{N-i+1}. Consequently, we obtain $\boldsymbol{v}(i) = \tilde{\boldsymbol{k}}_*(N-i+1)$ for the projection of $\boldsymbol{k}_*$ onto the ith eigenvector of $\tilde{\boldsymbol{K}}$.

Using both relations, we can reformulate the previous bound in terms of eigenvectors and eigenvalues of the implicitly accessible kernel matrix $\tilde{\boldsymbol{K}}$:

$$\begin{aligned}\sigma_*^2 \leq & k_{**} - \left(\sum_{i=1}^{k} \frac{1}{\xi_i}\boldsymbol{v}(i)^2\right) \\ & - \frac{1}{\xi_{k+1}}\left(||\boldsymbol{k}_*||^2 - \sum_{i=1}^{k} \boldsymbol{v}(i)^2\right) + \sigma_\mathrm{n}^2.\end{aligned} \tag{29}$$

Since we can adjust k for the desired precision, we call this technique a *fine* approximation of the predictive uncertainty (FAPU).

As noted previously, eigenvalues and eigenvectors can be computed using the Arnoldi iteration. For $k+1$ eigenvalues and k eigenvectors, this requires $\mathcal{O}(NkT_3)$ operations. Again, the number of iterations T_3 until convergence was almost constant about 10 in our experiments. For the explicit computation of the kernel vector $\boldsymbol{k}_*$, we still have to spend $\mathcal{O}(ND)$ operations. Squared projections $\tilde{\boldsymbol{k}}_*(i)^2$ of $\boldsymbol{k}_*$ onto eigenvectors can then be computed in $\mathcal{O}(N)$. Similar considerations hold for the norm $||\boldsymbol{k}_*||^2$.

In summary, we need $\mathcal{O}(ND + NkT_3)$ operations to compute the fine approximation of σ_*^2 as given in Eq. (29). Although we thereby reduce the quadratic scaling, we still depend linearly on N. Nonetheless, using the histogram intersection kernel, we can even develop a coarser approximation leading to a further speed-up as shown next.

7.3 CAPU: Coarse Approximation of the Predictive Uncertainty

The extreme case of the previous approximation is obtained with $k = 0$:

$$\sigma_*^2 \leq k_{**} - \frac{1}{\xi_1} ||\boldsymbol{k}_*||^2 + \sigma_{\mathrm{n}}^2. \tag{30}$$

Thus, even for the most extreme approximation using FAPU, we still require a computation time linear in N to compute $\boldsymbol{k}_*$ and its squared norm. However, for the specific choice of HIK as the kernel function, we note that $||\boldsymbol{k}_*||^2$ can be expressed as follows:

$$||\boldsymbol{k}_*||^2 = \boldsymbol{k}_*^{\mathrm{T}} \boldsymbol{k}_* = \sum_{i=1}^{N} \Big(\sum_{d=1}^{D} \min\left(\boldsymbol{x}_*(d), \boldsymbol{x}_i(d)\right) \Big)^2. \tag{31}$$

We will now exploit the properties of the HIK to approximate $||\boldsymbol{k}_*||^2$ by a lower bound. Thereby, we will still obtain a valid upper bound approximation for the predictive variance as given in Eq. (30).

One important aspect for the approximation arises from properties of sparse features. When features have only a few non-zero entries, the majority of mixed termss

$$\min\left(\boldsymbol{x}_*(d_1), \boldsymbol{x}_i(d_1)\right) \cdot \min\left(\boldsymbol{x}_*(d_2), \boldsymbol{x}_i(d_2)\right)$$

between different dimensions in Eq. (31) will vanish. For a sparsity ratio of 0.1, these are already 99 % of all terms! In these scenarios, neglecting the mixed terms is well justifiable and we obtain an expression that looks like a Parzen density estimation with squared kernel values:

$$\begin{aligned} ||\boldsymbol{k}_*||^2 &\geq \sum_{i=1}^{N} \sum_{d=1}^{D} \left(\min\left(\boldsymbol{x}_*(d), \boldsymbol{x}_i(d)\right)\right)^2 \\ &= \sum_{i=1}^{N} \sum_{d=1}^{D} \min\left(\boldsymbol{x}_*(d)^2, \boldsymbol{x}_i(d)^2\right). \end{aligned} \tag{32}$$

On a closer look, we notice that Eq. (32) is similar to Eq. (6) but with squared features and $\boldsymbol{\alpha} \equiv \mathbf{1}$. Therefore, we can apply the same techniques as described in Sect. 4.1 with squared feature values. Furthermore, we can even use the same permutations of the learning data. The only additional overhead comes from computing a new matrix $\boldsymbol{A}_{\sigma_*^2} \in \mathbb{R}^{D \times N}$ storing the cumulative sums of squared feature values similar to $\boldsymbol{A}$ of Sect. 4.1.

In consequence, we can compute the squared kernel vector within $\mathcal{O}(D \log N)$ operations for an unseen example $\boldsymbol{x}_*$. Note that we can now even apply the quantization idea described in Sect. 4.2. Thereby, the computation time is ultimately reduced to $\mathcal{O}(D)$ operations and only requires the additional computation of a look-up table $\boldsymbol{T}_{\sigma_*^2}$ similar to $\boldsymbol{T}$ of Sect. 4.2. We refer to this fastest uncertainty approximation with q-CAPU.

We visualized all approximations in Fig. 3 for the $2D$ scenario already used in Fig. 1. As we can nicely see, the approximation error of FAPU is inversely related to the number of eigenvectors used. Furthermore, the approximation converges from a piecewise quadratic function (FAPU) to a piecewise linear function (CAPU). While the precision is thereby reduced, we simultaneously reduce the required evaluation time as well.

A final overview of all the presented approaches for predictive variance computations as well as their resulting runtimes and decision functions is given in Table 2. In summary, we are able to efficiently compute the predictive variance with adjustable precision as well as adjustable time to spend. Note that the predictive variance is the same for all known classes (Rasmussen and Williams 2006), thus, our computation times are efficient even for an extremely large number of different classes. In Sect. 11, we compare our techniques in terms of runtimes needed in large-scale experiments and study their usability for the task of active learning.

8 Incremental and Active Learning

Large-scale learning is not only important for training based on a large chunk of data in batch mode, but also when the dataset is growing incrementally. We therefore show that incremental learning can be realized within our framework with just a few minor modifications. Furthermore, we also show how standard active learning methods can be directly used with our efficient estimates of the GP predictive mean and variance.

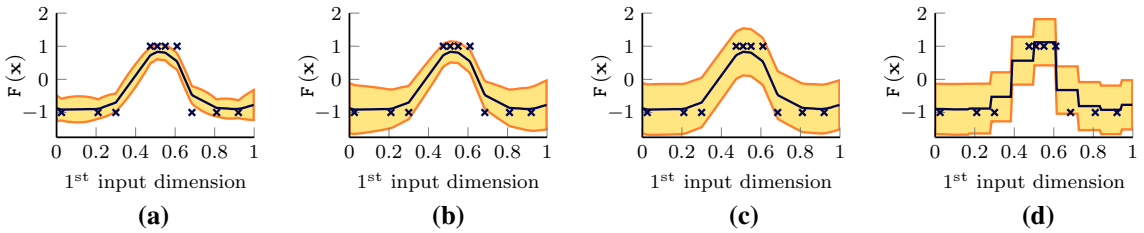

Fig. 3 Approximating the predictive variance σ_*^2 using our techniques which exploit properties of the HIK. Approximations are obtained with FAPU (*left*) for different numbers of k [see Eq. (29)] or using the the coarse approximation (*right*) without and with quantization of test inputs [see Eq. (30)]. The setup is identical to Fig. 1 where no approximation was applied. **a** FAPU ($k = 4$). **b** FAPU ($k = 1$). **c** CAPU. **d** q-CAPU ($q = 10$)

Table 2 Overview of the presented approaches to compute the predictive variance σ_*^2

Method	Asymptotic runtime	Resulting score	Approximation
GP-standard	$\mathcal{O}\left(\boldsymbol{N^2} + \boldsymbol{N}D\right)$	Eq. (3)	None (exact)
PUP (Sect. 7.1)	$\mathcal{O}\left(D\boldsymbol{N}T_1\right)$	Eq. (3)	None (exact)
FAPU (Sect. 7.2)	$\mathcal{O}\left(T_3 k D\boldsymbol{N}\right)$	Eq. (29)	Rank k approx. of $\tilde{\boldsymbol{K}}$
CAPU (Sect. 7.3)	$\mathcal{O}\left(D\boldsymbol{\log N}\right)$	Eq. (30)	Rank 1 approx. of $\tilde{\boldsymbol{K}}$, sparse feature assumption
q-CAPU (Sect. 7.3)	$\mathcal{O}\left(D\right)$	Eq. (30)	Rank 1 approx. of $\tilde{\boldsymbol{K}}$, sparse feature assumption, quantized test inputs

For details see the derivations in the corresponding sections. The parameters D and T_1 are defined as in Table 1, The number of eigenvectors used in the approximation is denoted with k. Variable T_3 is the number of iterations needed by the Arnoldi technique. We highlight the number of training examples N with bold font

8.1 Fast Incremental Learning

The usual blueprint for object recognition systems is to train a classifier on a given set of labeled data and to apply the resulting model on unseen examples. Although current research led to impressive results even on highly challenging datasets with this strategy (Lazebnik et al. 2006; Vedaldi et al. 2009; Kapoor et al. 2010), it suffers from two main drawbacks. First of all, there is no possibility to exploit labeled examples that are available after the training process. In consequence, we often neglect potentially useful information. Besides, this strategy will fail in situations where existing categories vary over time (e.g., cell phone designs) or new categories become available (e.g., Apple's iPod in 2001).

We can naively resolve these drawbacks by trivial training from scratch as soon as new data is accessible. However, we would thereby suffer from huge computational costs. For representer models such as Kernel-SVM or GP, every retraining would require $\mathcal{O}\left(N^3\right)$ since no information about the previously trained model is used. In contrast to that, incremental or online learning methods explicitly rely on previously trained models to efficiently adapt them over time. In the following, we show how to extend our GP/HIK for incremental learning.

As presented in the previous sections, training of GP/HIK models mainly consists of three stages: (1) sort training examples in every dimension, (2) compute the weight vector $\boldsymbol{\alpha}$ using an iterative linear solver, and (3) compute the matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ as well as the look-up table $\boldsymbol{T}$ if required. For new training examples, we can exploit the previous calculations in every stage to significantly speed-up the process of retraining:

(1) We can build on the given sorting of each dimension and find the correct position for a novel example in each dimension, which takes $\mathcal{O}\left(\log N\right)$ time for a single dimension and $\mathcal{O}\left(D \log N\right)$ in total.
(2) Using the previously calculated $\boldsymbol{\alpha}$ as an initialization for the iterative linear solver, we can significantly speed-up the process until convergence since the variations of $\boldsymbol{\alpha}$ are smooth and small, especially for large training sizes.
(3) For updating the arrays $\boldsymbol{A}$ and $\boldsymbol{B}$ as well as the look-up table $\boldsymbol{T}$, we only need to correct entries that are affected by new examples.

In addition to updates of the classification model, we could also adapt hyperparameters and optimize them on the fly. This would allow for adapting our model to new situations, e.g., when other feature dimensions become important to distinguish between categories. To speed up the optimization, we can easily use previous values of the hyperparameters as initial values for the optimization method, which is also known as warm-start.

In summary, we significantly benefit from previous calculations in every training step. We further validate this aspect in our experiments in Sect. 9.9.

8.2 Active Learning with Gaussian Processes

A main advantage of Gaussian processes is the possibility for giving feedback about how certain the classification result is. Aside from this apparent use of the predictive variance, it was successfully applied as a query strategy in active learning by Kapoor et al. (2010). The goal of active learning is to improve a classification model by selecting a few but highly informative examples for manual annotation. In this section, we briefly review the main ideas of Gaussian process based active learning.

For active learning, one typically has a small set $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ consisting of labeled data and a large set $\boldsymbol{X}_{\mathfrak{U}} = \{\boldsymbol{x}'_1, \ldots, \boldsymbol{x}'_{N'}\}$ of unlabeled examples. To obtain a classifier $\mathcal{A}$ trained with most informative examples, one exploits a query function $\mathcal{Q}$ that scores each unlabeled example. An oracle (e.g., a human expert) is then asked for the ground-truth label of the example $\boldsymbol{x}_*$ with best score. Consequently, an active learning scenario can be seen as a quadruple $(\mathcal{A}, \mathcal{Q}, \boldsymbol{X}, \boldsymbol{X}_{\mathfrak{U}})$.

One further distinguishes query strategies in two groups (Ebert et al. 2012). Exploitative methods utilize examples of $\boldsymbol{X}$ including their labels and rely on scores derived from outputs of the involved classifier. In contrast to that, explorative methods neglect the label information and query new examples only based on the distribution of the current examples. For the choice of Gaussian processes, Kapoor et al. (2010) introduced three possible query strategies which directly build on the trained model. Selecting examples based on smallest absolute predictive mean:

$$\mathcal{Q}_{\mu_*}\left(\boldsymbol{x}'\right) = -\left|\mu_*\left(\boldsymbol{x}'\right)\right|, \quad \boldsymbol{x}_* = \operatorname*{argmin}_{\boldsymbol{x}' \in \boldsymbol{X}_{\mathfrak{U}}} \left|\mu_*\left(\boldsymbol{x}'\right)\right| \tag{33}$$

is an exploitative method and selects examples close to the current decision boundary. Complementary, selecting examples with large predictive variance:

$$\mathcal{Q}_{\sigma_*^2}\left(\boldsymbol{x}'\right) = \sigma_*^2\left(\boldsymbol{x}'\right), \quad \boldsymbol{x}_* = \operatorname*{argmax}_{\boldsymbol{x}' \in \boldsymbol{X}_{\mathfrak{U}}} \sigma_*^2\left(\boldsymbol{x}'\right) \tag{34}$$

is explorative and prefers examples with highest classification uncertainty regarding the known training examples. Finally, Kapoor et al. (2010) propose to select examples with small *uncertainty*[1]:

$$\mathcal{Q}_{unc}\left(\boldsymbol{x}'\right) = -\frac{\left|\mu_*\left(\boldsymbol{x}'\right)\right|}{\sqrt{\sigma_*^2\left(\boldsymbol{x}'\right)}}, \quad \boldsymbol{x}_* = \operatorname*{argmin}_{\boldsymbol{x}' \in \boldsymbol{X}_{\mathfrak{U}}} \frac{\left|\mu_*\left(\boldsymbol{x}'\right)\right|}{\sqrt{\sigma_*^2\left(\boldsymbol{x}'\right)}}. \tag{35}$$

as a combination of $\mathcal{Q}_{\mu_*}$ and $\mathcal{Q}_{\sigma_*^2}$. The motivation here is to obtain a query function similar to the minimum margin approach suitable for SVMs (Tong and Koller 2001) but with the additional consideration of the classification uncertainty. We apply our techniques for efficient GP inference to these query strategies in Sect. 11.3 enabling active learning with a large pool of unlabeled examples.

9 Experimental Analysis: Large-Scale Classification

Our experimental evaluation is divided in three parts, thereby following the structure of the previous theoretical sections. In the current section, we analyze our techniques for efficient classification in large-scale scenarios. The suitability of our optimization approach is evaluated in Sect. 10 and experiments with our variance computation techniques are finally presented in Sect. 11.

9.1 Main Experimental Datasets

The majority of evaluations in the following three sections is done on two datasets which we shortly introduce here.

ImageNet for Multi-Class Classification To evaluate computational scalability of our introduced techniques, the ImageNet dataset as used for the ILSVRC'10 competition (Berg et al. 2010) serves as a perfect benchmark. We use in total 150,000 images from 1000 different categories from this dataset. Learned models are evaluated on 50,000 examples from the ILSVRC'10 validation dataset. As commonly done for ImageNet experiments, the flat-1-error is used as a measure of accuracy indicating the ratio of correctly classified examples among all test data.

ImageNet for Binary Classification Training of multi-class GP models with 1000 categories and hundreds of thousands of training examples is extremely time consuming even with our efficient GP/HIK techniques. Therefore, we also derive binary classification tasks from this dataset to allow for further analyses that have taken less time. Binary tasks are derived in a one-vs-all manner, i.e., we use all images of a single class as positive examples and ℓ examples from each of the other 999 categories as negative examples. Thereby, we obtain 200 tasks from the first 200 categories. Models are evaluated on the validation set using average AUC as a measure of accuracy.

[1] Note that in the remainder of the article, the term *uncertainty* refers to classification uncertainty, and not to the query strategy introduced by Kapoor et al. (2010).

Table 3 Large-scale learning and classification for 200 binary ImageNet tasks: computation times are given as median values of measurements for each task (learning) and each test example (classification)

Method	10, 090 examples ($\ell = 10$)			50, 050 examples ($\ell = 50$)		
	AUC	learning time	classif. time	AUC	learning time	classif. time
GP with HIK (cholesky)	0.836	>3.5 h	1.1 s	—*	—*	—*
GP with HIK (ours)	0.836	64 s	44 μs	0.856	321 s	44 μs

* Not possible due to excessive memory demand

15Scenes for Detailed Analyses We also use the 15Scenes dataset as a small-scale benchmark (Lazebnik et al. 2006), where all 15 classes are used for multi-class classification. Following the suggestion of Quattoni and Torralba (2009), all images are scaled to a size of 256×256 pixels to get results which are not biased on different characteristic image sizes for specific categories. Accuracy of learned models is measured with the averaged class-wise recognition rate (ARR).

9.2 Experimental Setup

Generalized histogram intersection kernels are designed to compare histogram-like image representations. Therefore, we represent images using either BOW features or non-negative activations of convolutional neural networks (CNNs). BOW features are computed using the available toolkit[2] provided with the ILSVRC'10 challenge (Berg et al. 2010). We use the provided visual codebook with 1, 000 elements to allow for easy reproducibility. CNN activations are obtained from the AlexNet CNN learned on ImageNet (Donahue et al. 2013) and extracted using the Caffe toolbox.

For optimization of hyperparameters, we use the Nelder–Mead technique (Nelder and Mead 1965) as mentioned in Sect. 6.3.

9.3 Large-Scale Experiments with ImageNet

The first question we are particularly interested in is whether our provided techniques allow for applying GP models to large-scale data. Therefore, we investigate two scenarios on the ImageNet dataset: binary classification and multi-class classification.

Binary Classification Scenarios Let us start with evaluating computational benefits arising from GP/HIK in comparison with the GP baseline implementation using a Cholesky decomposition and explicit kernel evaluations. Here, we only create binary classification scenarios to obtain at least some results for the baseline approach in affordable time. Images are represented using provided BOW features with 1, 000 dimensions. Results are shown in Table 3 for $\ell = 10$ and $\ell = 50$ resulting in 10, 090 and 50, 050 training examples, respectively. Computation times are measured on a single-core Intel 2.6GHz machine and our method makes use of a quantization with 100 bins to speed up classification.

As can be seen in Table 3, we are able to learn GP classifiers within a few minutes without loss in accuracy. In contrast to that, the baseline GP implementation is not applicable to more than some thousands of examples—due to computation time and memory demand. In particular, standard GP regression for $\ell = 50$ exceeded our memory capacities by resulting in a 19 GB kernel matrix.

In summary, we observe that our approach for training and classification is significantly faster than the baseline GP (speed-up factor: 196) and has only a linear memory requirement. Due to both aspects, it allows for learning on large-scale datasets that are otherwise intractable for exact GP inference.

Multi-class Classification Scenarios For a multi-class classification experiment on ImageNet, we compare against SVM solvers publicly available. Specifically, we choose the popular LibSVM package as Kernel-SVM solver with default parameter settings. Since κ_{HIK} is not directly supported by LibSVM, we follow provided suggestions and apply an RBF kernel instead. Furthermore, we compare against LibLinear as standard solver for linear SVM models in large-scale image classification scenarios (Deng et al. 2010). We apply an adaptive quantization with $q = 100$ bins for each dimension as introduced in Sect. 4.2. As before, we use provided BOW features with 1, 000 dimensions. For different numbers of training examples, we average over ten random splits. Results in terms of Flat-1 error rates are shown in Fig. 4.

We observe that GP/HIK can successfully be used in large multi-class classification scenarios with 1000 categories. For a small number of training examples N, GP/HIK even outperforms LibLinear but is slightly inferior for increasing training sets. In addition, LibSVM leads to significantly larger errors. We believe that this results from fixed default settings for regularization parameters used in our experiment.

[2] http://www.image-net.org/challenges/LSVRC/2011/ILSVRC2011_devkit-2.0.tar.gz.

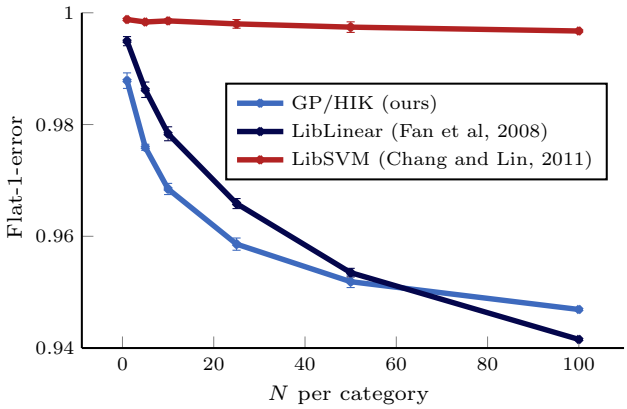

Fig. 4 Flat-1 error for multi-class classification using all 1000 categories of ImageNet and provided BOW features

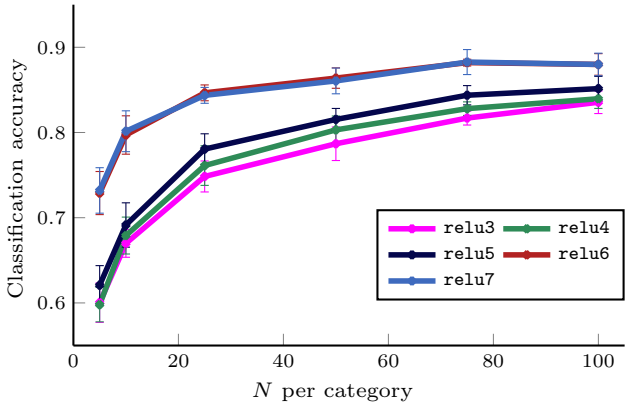

Fig. 5 Performance of GP/HIK on 15Scenes with CNN features extracted from different `relu` layers and a varying number of training examples per class

In summary, we observe that GP/HIK is a useful alternative to established SVM solvers in large multi-class scenarios.

9.4 Detailed Analysis Using the 15Scenes Dataset

The previous experiments confirmed the applicability of our GP/HIK to large-scale data. Nonetheless, a simple BOW image representation can not compete with state-of-the-art today. Thus, previous results only serve as proof-of-concept. We are now interested in investigating whether GP/HIK is also useful to work on recent image representations extracted from CNN activations. We chose the 15Scenes dataset to guarantee that all training data fit into our available memory even with the largest applied image representation of $D = 64,896$ feature dimensions (`relu3`).

GP/HIK on CNN Activations For different numbers of training examples, we trained GP/HIK models on top of different CNN activations. We used layers `relu3` to `relu7` due to their non-negativity. Results are shown in Fig. 5.

First of all, we note that GP/HIK is indeed applicable as model on top of CNN activations. Besides, we find that high-level activations of layers `relu6` and `relu7` give the best performance for this task. Interestingly, earlier experiments in (Freytag et al. 2014b) showed that humans only obtain an accuracy level of 85.67 % on this dataset. Thus, the accuracy of 87.98 % with `relu7` and the largest train size is indeed remarkable.

GP/HIK versus SVM Baselines As in the previous section, we also compare against linear and kernelized SVM as baselines. Based on the previous results, we use `relu7` features as representations. For consistency, we also add results for BOW representations and an overview is given in Table 4. As can be seen, we again outperform the SVM baselines when the same features are used. The recognition rate of a linear SVM is even lower when an L_1-normalization of the features is applied, a pre-processing technique which we use for our GP methods. Another not so surprising fact is the superiority of CNN compared to BOW or SPMK (BOW with spatial pyramid matching) features. The only technique currently outperforming our method is an AlexNet carefully fine-tuned on 15Scenes . However, this method often requires a grad-student tweaking hyperparameters for a day.

In summary, we find that our techniques can serve as powerful, probabilistic classification technique on top of recent image representations.

9.5 Evaluation of Linear Solvers with Fast HIK Multiplications

In the following, we compare the performance of conjugate gradients with fast HIK matrix multiplications as presented in Sect. 4 against two coordinate descent approaches: (1) the coordinate descent method of Wu (2010) applied to GP and (2) the greedy block coordinate descent (GBCD) approach of Bo and Sminchisescu (2012). The first one was originally presented for fast SVM learning with HIK and directly operates on the look-up table $\boldsymbol{T}$ (Sect. 4). GBCD calculates parts of the kernel matrix on the fly to solve sub-problems. For our experiments, the size of the sub-problems is set to 10 and the number of components for greedy selection is 20. We also tested other values like a sub-problem size of 60 and 500 number of components as suggested by (Bo and Sminchisescu 2012), but did not achieve a significant speed-up. Note that our approach and (Wu 2010) exploit fast HIK matrix multiplications, while (Bo and Sminchisescu 2012) can be applied for every kernel function.

We use again the ImageNet dataset with binary tasks and solve the linear system $\tilde{\boldsymbol{K}}_{\eta} \cdot \boldsymbol{\alpha} = \boldsymbol{y}$ with all three methods. Since we only care about the speed of convergence, we use

Table 4 Evaluation of GP/HIK on the 15Scenes dataset with standard BOW features (*upper part*) and CNN features (*lower part*)

Method	Features	**ARR**
Linear SVM (Fan et al. 2008)	BOW	68.4 %
SVM/HIK (Quattoni and Torralba 2009)	BOW	64.1 %
SVM/HIK (Quattoni and Torralba 2009)	GIST	73.0 %
SVM/HIK (Quattoni and Torralba 2009)	SPMK	73.4 %
GP/HIK (Ours)	BOW	70.8 %
Linear SVM (Sun and Ponce 2013)	learned patches	86.00 %
Linear SVM (Fan et al. 2008)	AlexNet-relu7	85.87 %
GP/HIK (Ours)	AlexNet-relu7	87.95 %
AlexNet CNN with fine-tuning	–	90.92 %

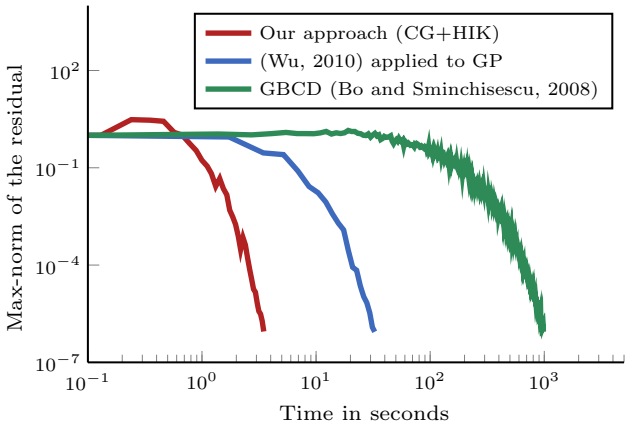

Fig. 6 Evaluation of the runtime and convergence of linear solvers: (1) our conjugate gradients method, (2) the coordinate descent method of Wu (2010), and (3) greedy block coordinate descent of Bo and Sminchisescu (2012)

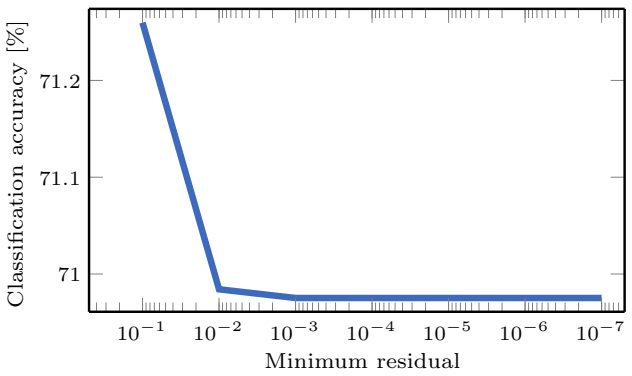

Fig. 7 Evaluation of model regularization by early stopping. Desired accuracies for the iterative linear solver to stop the computation are displayed on the x axis, whereas the y axis shows the resulting recognition rate as a measure of generalization abilities. The standard HIK serves as kernel function

a rather small-scale setup with $\ell = 1$. Figure 6 shows the residual of the linear system with respect to the computation time needed. Termination is done when the maximum norm of the residual drops below 10^{-6}. Computation times are measured on a single-core Intel 2.6 GHz machine.

As can be seen in Fig. 6, there are orders of magnitude between all three methods. Conjugate gradients reaches a solution in 3.7 seconds, which is superior to the coordinate descent method of Wu (2010) applied to GP (32s until convergence). GBCD is slow (convergence after 16 minutes) due to the long time needed for explicit calculation of kernel values for features of dimension $D = 1,000$. It should be noted that solving the linear system of GP regression needs more time than solving an SVM optimization problem as presented by Wu (2010). This is due to the additional sparsity constraints of SVM. Furthermore, runtime results presented by Bo and Sminchisescu (2012) looked more promising, which is likely due to the low dimensionality of chosen features ($D \leq 37$) in the paper. In summary, we find that the CG method nicely fits to our techniques for fast matrix-vector multiplications using HIKs.

9.6 Early Stopping of the Linear Solver

Early stopping refers to performing optimization not until convergence but only up to the point when the residual is lower than a predefined threshold. For large-scale SVMs, Perronnin et al. (2012) figured out that regularization by early stopping leads to suitable generalization abilities with the additional benefit of computation time saved. Since the iterative linear solver in our proposed methods allows for early stopping as well, we evaluate in the following whether their findings also hold here.

For an experimental evaluation, we conduct experiments on the 15Scenes database (Lazebnik et al. 2006) using our GP/HIK and BOW features. We stopped the process of training at different levels of accuracy reached by the iterative linear solver, i.e., if the residual dropped below predefined values. Experimental results are shown in Fig. 7.

First of all, we notice a rapid convergence of the resulting classification accuracy even if we stop the linear solver with an extremely high residual. In fact, an early stopping might even lead to better generalization performance. We therefore conclude that early stopping the calculation of

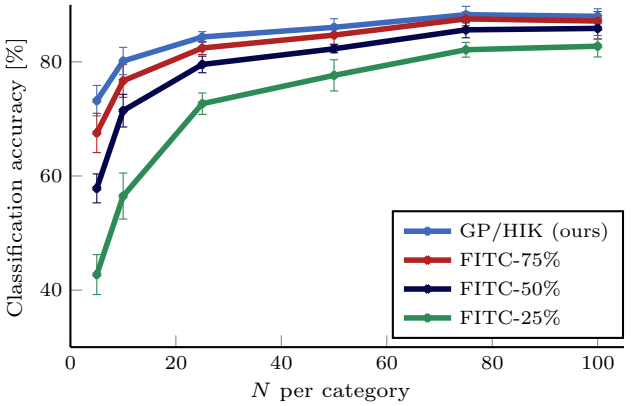

Fig. 8 Comparison of our approach with the FITC method of Quiñonero Candela and Rasmussen (2005) on the 15Scenes dataset with `relu7` features. For FITC, different relative sizes of the inducing set are used

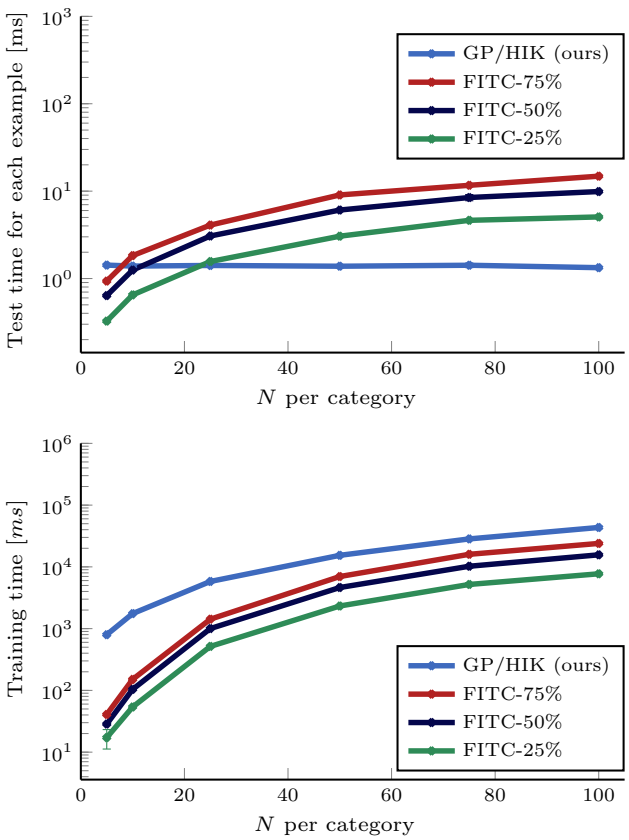

Fig. 9 Comparison of our approach with FITC with respect to (*top*) time needed during inference for each test example and (*bottom*) total time needed for training. See Fig. 8 for an analysis of the resulting accuracies

weights $\boldsymbol{\alpha}$ is advisable. In our experiments, the number of iterations needed until reaching the stopping criterion grew exponentially with the desired accuracy. This fact additionally highlights the benefit of early stopping.

In summary, we find that early stopping of the iterative linear solver leads to well regularized models and significantly saves computation times.

9.7 Comparison with GP Sparsity Methods

An important family of methods for large-scale Gaussian process inference covers sparse approximation techniques. Among them, the Fully Independent Training Conditional Approximation (FITC) is presumably the most powerful representative (Quiñonero Candela and Rasmussen 2005). It is therefore interesting to compare our efficient techniques for exact inference against FITC as a representative for sparse GP approximations. Similar to previous experiments, we use the 15Scenes dataset and `relu7` features. Results for other CNN layers lead to comparable conclusions. Again, we average over ten random splits. For FITC, we evaluate different sizes of the inducing set. The accuracies depending on the number of training examples per category are visualized in Fig. 8. Furthermore, we compare required computation times for training and inference in Fig. 9.

With respect to classification accuracy (Fig. 8), we outperform FITC especially for small inducing sets with a large margin. We can thus conclude that our techniques allow for exact GP inference and circumvent the necessity of sparse approximations. Regarding time for inference, we observe in Fig. 9 that the our quantization approach leads to constant computation time. In contrast, FITC's computation time increases with the number of known examples. For training, however, we notice that FITC results in faster learning times. Although this does not hold asymptotically (see Table 1), the investigated setting has too few examples and too large feature dimensions to fully unveil the gain in computation time. In direct comparison, we thus conclude that our techniques is especially beneficial for learning from large datasets without requiring sparse approximations.

9.8 Evaluation of the Quantization

We already applied the quantization idea in Sect. 9.3 to evaluate GP models trained with hundreds of thousands of examples within milliseconds. For simplicity, we set the number of bins per dimension to $q = 100$ and obtained identical accuracies as the baseline GP. Let us now evaluate the resulting classification accuracy as well as required computation times for training and inference with varying numbers of quantization bins per dimension.

Our experiments in this section are performed on 15Scenes with L_1-normalized `relu7` features. For ten random splits, each setting is evaluated to allow for statistically significant but comparable results. Computation times are measured on an Intel Core i7-3930K CPU desktop computer with 3.20 GHz and without any parallelization. All

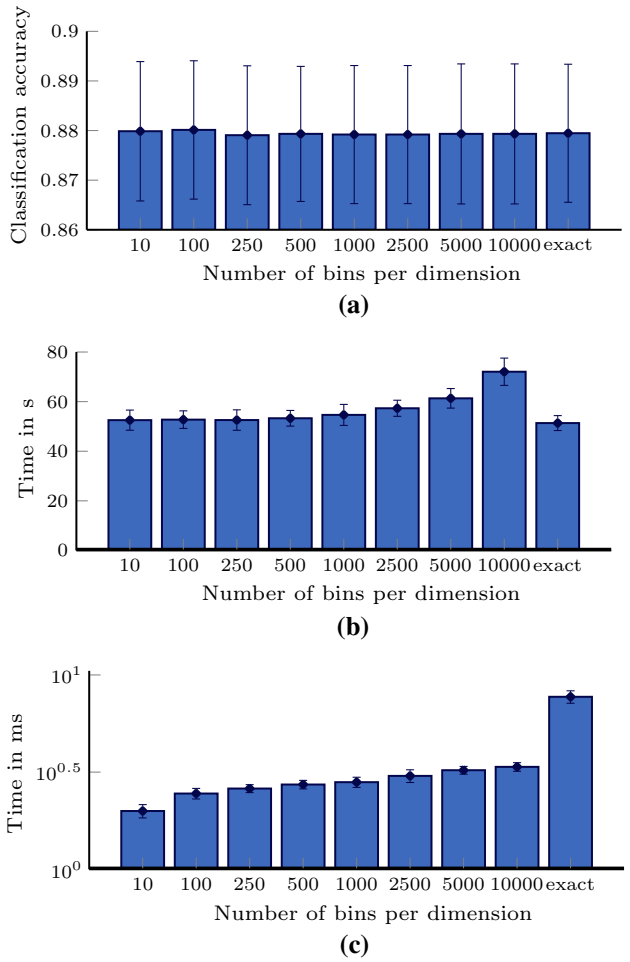

Fig. 10 Effect of different quantization levels on classification accuracy (*top*), required computation times during training (*middle*), and for inference (*bottom*). Results are obtained on the 15Scenes dataset and averaged over ten random splits. **a** Accuracy (note the small range of the *y* axis). **b** Training time. **c** Classification time

computation times include overhead arising from converting features from Matlab data structures to C++ pendants in Mex-interfaces. Results are shown in Fig. 10.

As we can nicely observe in Fig. 10a, quantizing feature values does not negatively affect the classification accuracy. In fact, for all evaluated settings, classification results are comparable to those achieved with exact inference. It should be noted that this effect is also due to the adaptive quantization. For a uniform quantization equal in all dimensions, accuracy drops significantly for small number of bins (not shown here).

From Fig. 10b, we further observe that the training time grows over the exact baseline the more bins are required, since larger LUTs need to be computed and stored. Finally, classification times are shown in Fig. 10c. We observe clear speed-ups compared to the exact baseline.

In summary, we find that quantization of features saves valuable time during classification at the cost of an affordable overhead. Already with 100 bins per dimension, classification results are on-par with the exact baseline.

9.9 Comparing Incremental and Batch Learning

In Sect. 8.1, we analyzed how to efficiently handle new data without the necessity of retraining the classifier from scratch. To evaluate the resulting benefit, we show results of experiments conducted on the 15Scenes dataset with BOW features. In 100 runs, we randomly pick 10 examples per class as an initialization. During each run, we incrementally add 1 example per class over 50 iterations resulting at most 900 examples used for training the model. Every iteration consists of training the classifier as well as optimizing kernel hyperparameters to perform parameter optimization on the fly. Performances are evaluated on a disjoint test set consisting of 50 examples per class and the results are visualized in Fig. 11.

From the plot in Fig. 11a, we make the well-known observation that using more examples is beneficial for training better models. In addition, we notice that the models learned in an incremental manner lead to almost identical results as those from models trained from scratch. However, when taking the computation times given in the plot of Fig. 11b into account, we obtain a clear advantage of our incremental learning approach compared to simple retraining. This speed-up increases with the number of examples and is therefore especially useful for large-scale scenarios.

Note that the major update time is spent for finding optimal hyperparameters during updates. Thus, we could obtain further speed-ups by running optimization steps only after a batch of new data with several examples has been recorded. Since hyperparameters only vary slowly, this would be well justifiable in practice.

Summarizing, we are able to efficiently update our model when new data is available even with an involved parameter optimization, which allows for using Gaussian processes for large-scale scenarios in lifelong or active learning.

10 Experimental Analysis: Hyperparameter Optimization

In this section, we are interested in evaluating our approach for efficient hyperparameter optimization as presented in Sect. 6. The results of this section can be summarized as follows:

(1) Optimizing the exact log-likelihood and our upper bound approximation lead to similar optima in practice (Sect. 10.1).

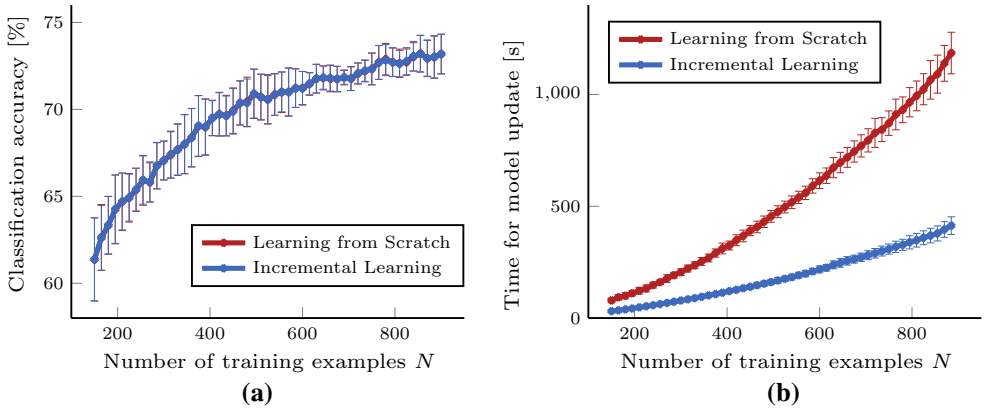

Fig. 11 Comparison of incremental learning and learning from scratch: **a** classification results achieved by both methods (the graphs are identical), **b** corresponding times for retraining the classifier when new data is accessible

(2) Generalized histogram intersection kernels improve the classification performance significantly compared to standard HIK (Sect. 10.2 and Sect. 10.3).
(3) Feature relevance determination can be done by optimizing the marginal likelihood of a weighted HIK (Sect. 10.4).
(4) Early stopping is also applicable when hyperparameters are optimized (Sect. 10.5).

10.1 Verifying the Bound of the Negative Marginal Log-Likelihood

Before we evaluate potential benefits which arise from optimizing hyperparameters, we are first of all interested in the tightness of our introduced bounds for the negative GP marginal log-likelihood presented in Sect. 6. We therefore train GP models on the 15Scenes dataset with BOW features as image representations and different values of η for HIK-POLY. Then, we evaluate our upper bound approximation as well as the exact value for the negative log-likelihood $-\log p(\boldsymbol{y} \mid \boldsymbol{X}, \eta)$. Furthermore, we evaluate trained models on hold-out test data and report average recognition rates to investigate the relation between likelihood and accuracy. Results are shown in Fig. 12.

First of all, we notice that the exact log-likelihood is closely connected to the resulting accuracy. Thus, the log-likelihood is a useful criterion for adapting hyperparameters of models to training data. Furthermore, it can be seen that our bound closely matches the true negative marginal log-likelihood in this setup. In consequence, the minima of both curves only differ slightly and our optimization technique can be successfully applied. For higher values of η, our bound converges to the exact value because the influence of the log-determinant term compared to the data term of the marginal log-likelihood decreases. Consequently, possible approximation errors become less important and the data term can be

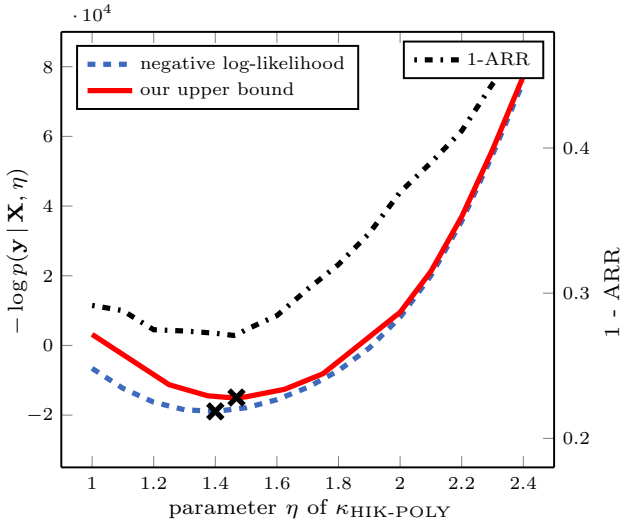

Fig. 12 Comparison between our upper bound of the negative marginal log-likelihood, the real negative marginal log-likelihood (*left* y axis), and resulting ARR on the test set (*right* y axis). Different hyperparameter values for HIK-POLY are shown on the x axis

computed without any approximation even for large-scale datasets. We observed a similar behavior for other datasets and settings.

In summary, we find that our upper bound approximation is well suited for optimization of hyperparameters.

10.2 Different Generalized HIK for Binary Classification

Since we found that our upper bound tightly matches the exact log-likelihood, we are now interested in the resulting benefits of optimizing generalized variants of the HIK. We again start with evaluations on binary classification tasks similar to Sect. 9.3. The experimental setup is kept identical but with activated optimization of hyperparameters for

Table 5 Benefit of hyperparameter estimation for 200 binary ImageNet tasks: Computation times are given as median values of measurements for each task (learning) and each test example (classification)

Method	10,090 examples ($\ell = 10$)			50,050 examples ($\ell = 50$)		
	AUC	Learning time	Classif. time	AUC	Learning time	Classif. time
GP/HIK	0.836	64 s	44 μs	0.856	321 s	44 μs
GP/HIK-POLY	0.865	435 s	44 μs	0.883	2815 s	44 mus
GP/HIK-EXP	0.889	579 s	44 μs	0.893	2578 s	44 μs

Optimization was done for the hyperparameters of $\kappa_{\text{HIK-POLY}}$ and $\kappa_{\text{HIK-EXP}}$, respectively

Table 6 Evaluation of hyperparameter optimization for $\kappa_{\text{HIK-POLY}}$ and $\kappa_{\text{HIK-EXP}}$: classification accuracy is obtained on the 15Scenes dataset with standard BOW features (upper part) and CNN features (lower part)

Method	Features	ARR
GP/HIK	BOW	70.8 %
GP/HIK-POLY	BOW	72.9 %
GP/HIK-EXP	BOW	74.0 %
GP/HIK	AlexNet-`relu7`	87.95 %
GP/HIK-POLY	AlexNet-`relu7`	87.07 %
GP/HIK-EXP	AlexNet-`relu7`	87.87 %

Compare also against Table 4

$\kappa_{\text{HIK-POLY}}$ and $\kappa_{\text{HIK-EXP}}$. Experimental results are shown in Table 5.

As we can see, our optimization technique based on the upper bound approximation is able to handle datasets with tens of thousands of training examples. Thereby, we obtain accuracy gains statistically significant with $p < 10^{-7}$ measured by a paired t test.

In summary, we find that our optimization technique leads to valuable accuracy gains in binary classification settings.

10.3 Different Generalized HIK for Multi-class Classification

For an analysis of hyperparameters in multi-class classification scenarios, we build on the previous evaluations of Sect. 9.4 and use the 15Scenes dataset with BOW and CNN features. In contrast to the previous evaluation, we now optimize hyperparameters of HIK-POLY and HIK-EXP with our GP marginal likelihood optimization technique. Thus, this analysis complements the previous results shown in Table 4. Results are given in Table 6.

For the BOW features, we observe that optimizing hyperparameters of GP/HIK-EXP results in the highest accuracy. In fact, it is even comparable to the result of the spatial pyramid matching kernel (SPMK) by Quattoni and Torralba (2009) as shown in Table 4. This highlights the power of generalized HIK and our hyperparameter optimization.

When applying CNN features (lower part of Table 6), results show a huge increase of performance in general leading to state-of-the-art results. It should be noted that in contrast to results by Sun and Ponce (2013), we do not perform patch discovery or fine-tuning to obtain features especially suited for the dataset, but still improve on their results listed in Table 4. Interestingly, the generalized HIK does not further increase classification accuracy—it even reduces the performance slightly. We have not yet a convincing explanation for this phenomenon, which requires further research.

In summary, we find that our GP marginal likelihood optimization method is technically suited to optimize hyperparameters in multi-class classification scenarios. Useful adaptations and other kernel parameterizations for recent CNN features remain an open question.

10.4 Feature Relevance Estimation

We have already seen that Gaussian processes allow for hyperparameter optimization by marginal likelihood estimation. In this experiment, we show the suitability of GP equipped with optimized weighted HIK for efficient feature relevance determination leading to superior results compared to those of SVM-based estimations.

Since there is no exact gradient information during the optimization available, the Nelder–Mead method converges poorly for huge numbers of parameters to be optimized. Consequently, computing feature relevance for features with thousands of dimensions, as in our previous experiments, is almost impossible right now.

Nevertheless, as a proof of concept we follow the same synthetic experimental setup as Ablavsky and Sclaroff (2011): for different numbers of training examples, we randomly sample eight-dimensional feature vectors with relevant information only available in the first two dimensions. The performance is estimated with 500 tests. For the specific random distributions, we refer the reader to the work of Ablavsky and Sclaroff (2011) and references therein. The results of our experiments can be seen in Fig. 13.

The information included in each dimension is well reflected by the estimated relative weights η_i, which can be observed from the plot in Fig. 13a. Furthermore, the plot in Fig. 13b shows the recognition accuracy for standard and weighted HIK with respect to the training size. The improvement is highly significant with $p < 10^{-7}$ using the paired t test. In comparison with Ablavsky and Sclaroff (2011),

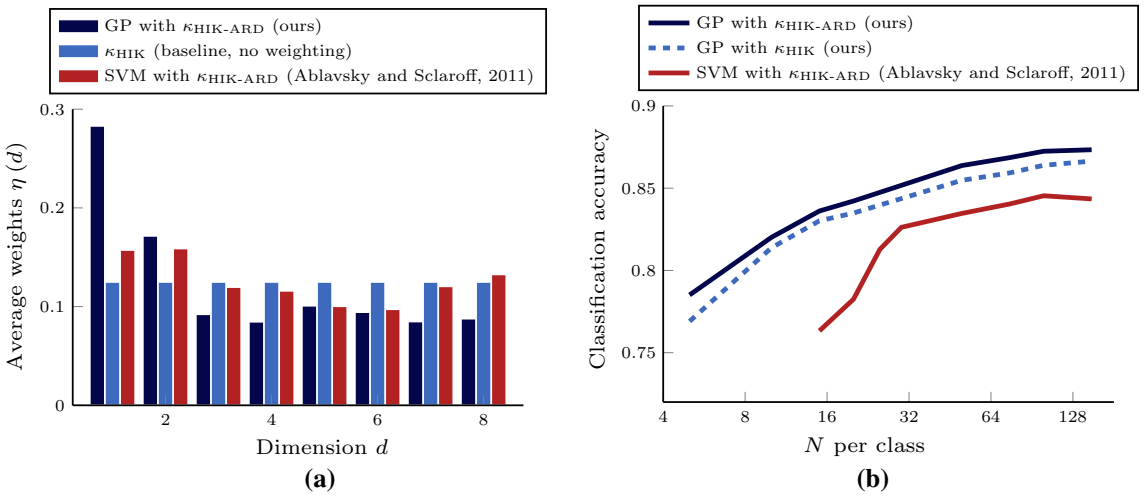

Fig. 13 Relevance determination with very generalized histogram intersection kernels and GP hyperparameter optimization. The first two features contain most of the discriminative information: **a** feature weights estimated with 5 examples per class, **b** performance compared to non-weighted histogram intersection kernels. Results are averaged over 500 runs

our approach additionally leads to more consistent weights and higher accuracies. The experimental results emphasize the benefits of a probabilistic framework for hyperparameter optimization.

In addition, regularization terms could be added to the objective, such as terms based on the minimum description length principle. However, this is beyond the scope of this paper.

10.5 Early Stopping of the Linear Solver

In Sect. 9.6, we investigated the effect of early stopping for GP/HIK . However, we did not optimize hyperparameters and applied the plain κ_{HIK} instead. Let us therefore investigate whether the same findings hold if optimization is additionally activated. We therefore repeat the same experiments as in Sect. 9.6 but optimize parameters of HIK-POLY using our marginal likelihood optimization technique. Results are shown in Fig. 14.

As for the plain κ_{HIK}, we again notice a rapid convergence of the resulting classification accuracy. In contrast, to the results without hyperparameter optimization there is no benefit of early stopping in terms of accuracy. In addition, it should be noted that the optimal hyperparameter value remained unchanged for all settings of the stopping criterion throughout our experiments.

We conclude that early stopping is well applicable for hyperparameter optimization using our upper bound for the negative log-likelihood, since it leads to a significant speed-up due to a decrease in iterations of the linear solver. However, an increase in accuracy with early stopping cannot be expected.

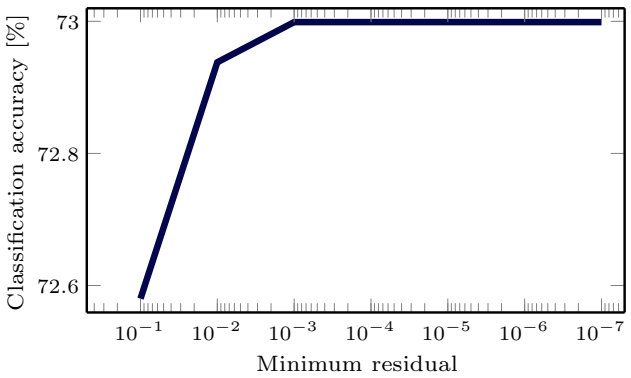

Fig. 14 Evaluation of model regularization by early stopping. Desired accuracies for the iterative linear solver to stop the computation are displayed on the x axis. The y axis shows the resulting recognition rate as a measure of generalization abilities. In contrast to the previous evaluation in Sect. 9.6, we now optimize the hyperparameter η of the HIK-POLY

11 Experimental Analysis: Uncertainty Prediction

The third part of our experimental analysis deals with the uncertainty approximation introduced in Sect. 7. We can summarize the results of this section as follows:

(1) The predictive variance of Gaussian processes can be approximated efficiently even in large-scale scenarios with a speed-up of up to 45, 000× (Sect. 11.1).
(2) Our upper bound approximations closely follow the exact variance scores (Sect. 11.2).
(3) Approximation of the predictive variance leads to active learning results comparable to the ones achieved with the exact predictive variance (Sect. 11.1).

Table 7 Runtimes needed for the computation of the predictive variance using the presented techniques from Sect. 7 in comparison to the baseline GP on two image categorization datasets (see Sect. 11.1)

Method	Number of training examples		
	1, 500	10, 090	50, 050
GP, plain formulation	60.36 ms	1.54 s	—*
PUP (Sect. 7.1)	2.11 s	22.92 s	>1 min
FAPU (Sect. 7.2), $k = 8$	13.73 ms	105.63 ms	1.47 s
FAPU (Sect. 7.2), $k = 2$	13.35 ms	105.37 ms	1.15 s
CAPU (Sect. 7.3)	5.92 ms	62.07 ms	266.97 ms
q-CAPU (Sect. 7.3)	13.32 μs	33.89 μs	33.89 μs

* Not possible due to excessive memory demand

11.1 Fast Computation of the Predictive Variance

We start this section by evaluating the efficiency of our proposed techniques for computing the predictive variance in terms of computation times needed. As in the previous sections, we conduct experiments on the 15Scenes dataset (Lazebnik et al. 2006) as well as on the large-scale ImageNet dataset. For the 15Scenes dataset, we randomly pick 100 examples of each class for training resulting in 1500 training examples in total. Training on the large-scale ImageNet dataset with binary tasks is carried out using $\ell = 10$ or $\ell = 50$ randomly chosen examples per negative class and 100 examples for the positive class, which results in 10, 090 and 50, 050 training examples in total. Computation times are averaged over all remaining examples. Experiments in this section are conducted on a 3.4 GHz CPU without any parallelization.

Experimental results are shown in Table 7. Especially for large-scale datasets, we obtain a significant speed-up compared to the direct computation of the variance (GP-standard). For rapid uncertainty prediction, the CAPU method turns out to be highly suitable with computation times in the order of microseconds. It should be noted that the PUP method is relatively slow due to the involved computations of the iterative linear solver. Although we have already seen the efficiency of a linear conjugate gradient method for this problem, it still needed some hundreds of iterations until convergence, especially for large training sets. Therefore, we argue to use the precise method only in cases where time is not the limiting factor, but the GP baseline can not be computed explicitly due to the huge memory demand.

11.2 Investigating the Tightness of Variance Approximations

In a second experiment, we investigate the deviation between exact variance scores and our introduced approximations. As argued before, two outcomes can be acceptable which depends on the application scenario: (1) either the approximations should be close to the exact scores (e.g., for security applications with fixed thresholds), or (2) only the correct relative order is of interest (e.g., active learning). We analyze our techniques regarding both aspects in the following.

As previously, we use the 15Scenes dataset for evaluation. Images are represented using L_1-normalized `relu7` features. For training, we select three categories with 50 randomly chosen examples of each category. All remaining data serves for model evaluation. Thereby, we obtain a small set of examples from known categories and a significantly larger pool of unknown categories. The noise level for model regularization is fixed to $\sigma_n^2 = 0.1$. For FAPU, we spend $T_3 = 100$ iterations for the Arnoldi technique to estimate even 16 eigenvectors reliably The quantization for q-CAPU is done with $q = 100$ bins per dimension and dimension-adaptive.

To analyze the deviation of scores, we sort variance scores of the exact PUP technique increasing order. Predictions of the remaining techniques are re-arranged accordingly. Results are shown in the left part of Fig. 15. For visualization, only every 5th sample is plotted.

The relative order of scores is additionally analyzed by normalizing scores of each method individually into [0, 1]. Again, scores of PUP are ordered decreasingly and scores of remaining methods are plotted accordingly. Results are shown in the right part of Fig. 15.

First of all, we clearly observe the upper bound relationship among our introduced techniques. Regarding exact scores, it can be seen that PUP is closed matched by the FAPU approximation. Hence, FAPU should be used when approximation errors are undesirable. Besides, we observe that q-CAPU and CAPU lead to identical results although their scores differ strongly from the exact counterparts. Nonetheless, the relative order is comparable (right figure). Hence, CAPU is a reasonable choice if only the relative order of variance estimates is required.

11.3 Application for Active Learning

In a final experiment, we are interested in applying our GP/HIK to active learning scenarios. Therefore, we apply the three query strategies by Kapoor et al. (2010) as reviewed in Sect. 8.2. We evaluate our methods on the difficult real-world ImageNet dataset. For each experiment, we randomly pick a single positive class as well as four, nine, or nineteen classes providing negative examples. Starting with two randomly chosen examples per class, we query new examples using the proposed methods. Each task is repeated with 100 random initializations. Final results are achieved by averaging over 100 different tasks. For variance estimation, we apply our FAPU technique with $k = 2$ eigenvectors for the approximation. Experimental results are given in Fig. 16.

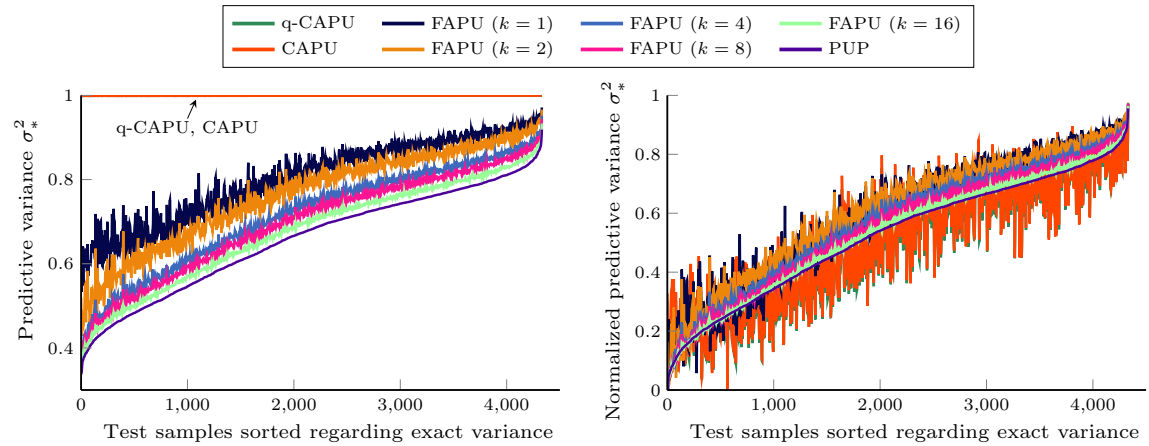

Fig. 15 Tightness of GP/HIK variance approximations on the 15Scenes dataset with `relu7` features. Scores of the exact method are increasingly ordered. Predictions of remaining techniques are re-ordered accordingly. GP/HIK is trained on L_1-normalized `relu7` activations. *Left*: variance predictions of each method. *Right*: we normalized variance predictions of each method individually into [0, 1] to better visualize the overall trend. Best viewed in color

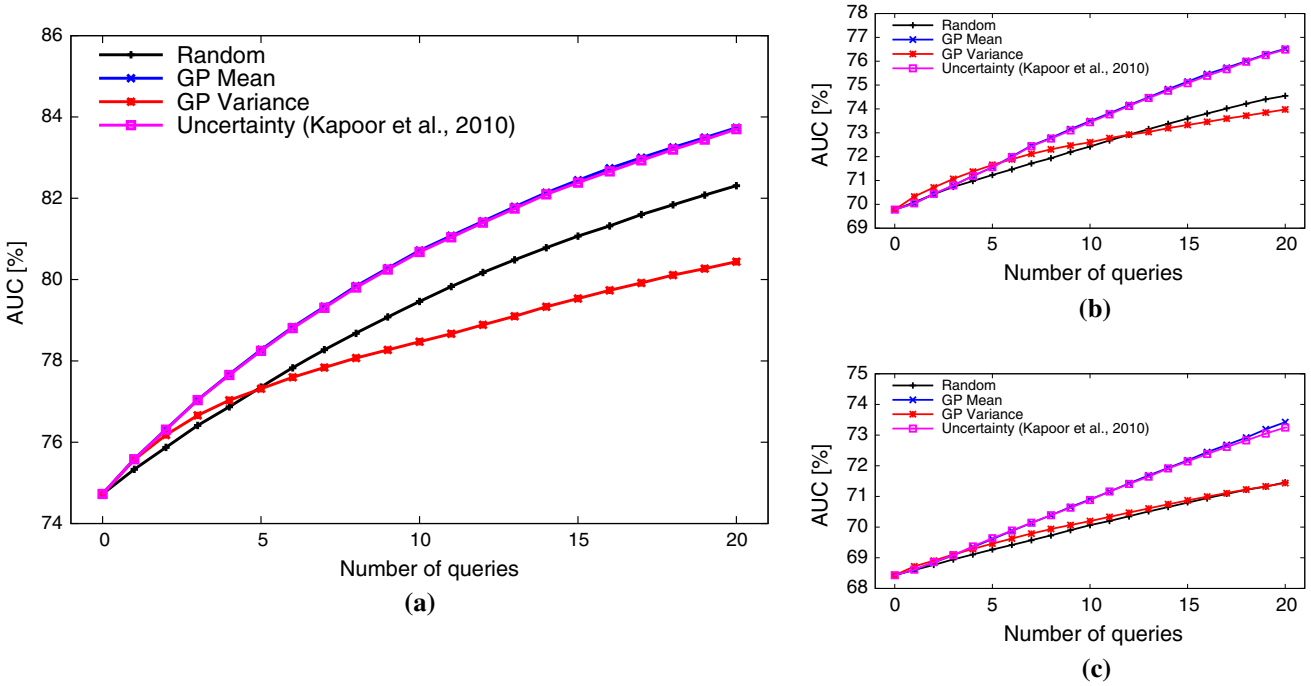

Fig. 16 Active learning results on 100 binary classification tasks derived from the ImageNet dataset. Best viewed in color. **a** 4 negative classes. **b** 9 negative classes. **c** 19 negative classes

We first notice that concerning $\mathcal{Q}_{\sigma_*^2}$, we again obtain interesting results, since it is inferior to random sampling for a small number of negative classes but slightly superior for larger number of classes. Query strategies $\mathcal{Q}_{\mu_*}$ and $\mathcal{Q}_{\text{unc}}$ tend to pick similar examples even on this challenging dataset resulting in almost identical performances, which is due to the strong influence of the mean term in $\mathcal{Q}_{\text{unc}}$.

As a concluding remark we state that for active learning, a suitable combination of mean and variance is beneficial to obtain satisfying learning rates, and our techniques are appropriate for computing query scores even for thousands of possible queries in large-scale learning scenarios.

12 Conclusions

In this article, we presented solutions for efficient Gaussian process inference in large-scale scenarios. Our techniques cover all aspects of inference, i.e., exact multi-class classification with label regression, hyperparameter optimization, and uncertainty prediction. A key aspect of all techniques

is to exploit generalized histogram intersection kernels, which have proved to be highly suitable for measuring similarities between histogram-like representations. Thereby, our derived methods yield significant asymptotic as well as practical speed-ups (several orders of magnitude) while requesting only a linear amount of memory. We empirically validated our techniques for a wide range of application scenarios and provided detailed analyses of all involved aspects. Experimental results disprove common belief that full non-parametric Bayesian methods are too expensive for large-scale data.

We believe that the presented techniques can be applied to a wide range of other uses of a GP model (Bonilla et al. 2008; Pillonetto et al. 2010; Bo and Sminchisescu 2010), where the inversion of the kernel matrix or computing its log-determinant is the main bottleneck. Finally, the joint possibility for efficient and exact classification, uncertainty prediction, hyperparameter adaptation, and online learning ultimately allows for life-long learning applications with never-ending data streams.

Our developed source code is publicly available at https://github.com/cvjena/gp-hik-core and licensed under LGPL. Since we provide C++ implementation as well as Matlab interfaces, we hope that other computer vision researchers can use our techniques as additional classification tool besides LibSVM or LibLinear.

Acknowledgements This research was partially supported by grant DE 735/10-1 of the German Research Foundation (DFG).

Quantization Error Analysis

Proof of Theorem 1 The quantization trick approximates a new example $\boldsymbol{x}_*$ with a quantized version $\tilde{\boldsymbol{x}}_*$. In the following, we assume D quantizations with q bins are given for each dimension. For each input value $\boldsymbol{x}(d) \in [l_d, u_d]$, we can compute a quantized value $\tilde{z}$. The maximum quantization error ε_q is then defined as follows:

$$\varepsilon_q = \max_{\boldsymbol{x}(d)\in[l,u]} \max_{1\leq d\leq D} |\boldsymbol{x}(d) - \tilde{z}|. \tag{36}$$

Our goal is to analyze the difference between the exact predictive mean μ_* and the one computed with the quantization trick $\tilde{\mu}_*$:

$$\begin{aligned} |\mu_* - \tilde{\mu}_*| &= |\boldsymbol{k}_*^{\mathrm{T}}\boldsymbol{\alpha} - \tilde{\boldsymbol{k}}_*^{\mathrm{T}}\boldsymbol{\alpha}| \\ &= |(\boldsymbol{k}_* - \tilde{\boldsymbol{k}}_*)^{\mathrm{T}}\boldsymbol{\alpha}| \leq \sum_{i=1}^{N} |\triangle k(i)|\,|\boldsymbol{\alpha}(i)| \end{aligned} \tag{37}$$

where we have defined $\triangle k(i) = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*) - \kappa(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}_*)$ as an abbreviation. Let us analyze this term in more detail using the relation $\min(a,b) = \frac{1}{2}\cdot(a+b-|a-b|)$:

$$\begin{aligned} |\triangle k(i)| &= |\kappa(\boldsymbol{x}_i, \boldsymbol{x}_*) - \kappa(\boldsymbol{x}_i, \tilde{\boldsymbol{x}}_*)| \\ &= \left| \sum_{d=1}^{D} \min(\boldsymbol{x}_i(d), \boldsymbol{x}_*(d)) - \min(\boldsymbol{x}_i(d), \tilde{\boldsymbol{x}}_*(d)) \right| \\ &= \frac{1}{2} \left| \sum_{d=1}^{D} \boldsymbol{x}_i(d) + \boldsymbol{x}_*(d) - |\boldsymbol{x}_i(d) - \boldsymbol{x}_*(d)| \ldots \right. \\ &\quad \left. \ldots - \boldsymbol{x}_i(d) - \tilde{\boldsymbol{x}}_*(d) + |\boldsymbol{x}_i(d) - \tilde{\boldsymbol{x}}_*(d)| \right| \\ &= \frac{1}{2} \left| \sum_{d=1}^{D} \boldsymbol{x}_*(d) - \tilde{\boldsymbol{x}}_*(d) - |\boldsymbol{x}_i(d) - \boldsymbol{x}_*(d)| + \ldots \right. \\ &\quad \left. \ldots |\boldsymbol{x}_i(d) - \tilde{\boldsymbol{x}}_*(d)| \right|. \end{aligned} \tag{38}$$

We can now exploit the fact that $\boldsymbol{x}_*$ is normalized to sum up to a constant value, e.g., 1. Furthermore, we assume that this also holds for $\tilde{\boldsymbol{x}}_*$. This assumption is reasonable for a high dimension D and a quantization error with zero mean. Putting both aspects together, we obtain the following equality:

$$\begin{aligned} |\triangle k(i)| &= \frac{1}{2} \left| \sum_{d=1}^{D} |\boldsymbol{x}_i(d) - \tilde{\boldsymbol{x}}_*(d)| - |\boldsymbol{x}_i(d) - \boldsymbol{x}_*(d)| \right| \\ &= \frac{1}{2} \left| \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_*\|_1 - \|\boldsymbol{x}_i - \boldsymbol{x}_*\|_1 \right| \end{aligned} \tag{39}$$

Due to the symmetry in the inequality with respect to $\boldsymbol{x}_*$ and $\tilde{\boldsymbol{x}}_*$, we can assume without loss of generality that $\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_*\|_1 \geq \|\boldsymbol{x}_i - \boldsymbol{x}_*\|_1$. This allows us to apply the triangle inequality and we finally obtain:

$$\begin{aligned} |\triangle k(i)| &= \frac{1}{2} \left(\|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_*\|_1 - \|\boldsymbol{x}_i - \boldsymbol{x}_*\|_1 \right) \\ &\leq \frac{1}{2} \left(\|\boldsymbol{x}_i - \boldsymbol{x}_*\|_1 + \|\boldsymbol{x}_* - \tilde{\boldsymbol{x}}_*\|_1 - \|\boldsymbol{x}_i - \boldsymbol{x}_*\|_1 \right) \\ &= \frac{1}{2} \|\boldsymbol{x}_* - \tilde{\boldsymbol{x}}_*\|_1. \end{aligned} \tag{40}$$

This directly leads to $|\triangle k(i)| \leq \frac{D\cdot\varepsilon_q}{2}$. Combined with Eq. (37), this proofs the final result of the Theorem.

The bound can be also improved by considering the sum of quantization errors in each dimension, but we skipped this fact for brevity and ease of notation. □

Proof of the Bound

Proof of Lemma 1 First note that due to the conditions of the lemma, the following holds: $1 \leq \mu_2 < \mu_1 \leq N$ and $\bar{t} > 0$. Furthermore, the bound is only valid for $\beta \neq \bar{t}$, because otherwise the 2×2 matrix within the bound would be singular.

We now start by deriving the coefficients for μ_1 and μ_2. The first part of Eq. (19) can be written as:

$$\begin{aligned}
&\left[\log\beta,\ \log\bar{t}\right]\begin{bmatrix}\beta & \bar{t}\\ \beta^2 & \bar{t}^2\end{bmatrix}^{-1}\\
&= \left[\log\beta,\ \log\bar{t}\right]\left(\frac{1}{\beta\bar{t}^2-\bar{t}\beta^2}\begin{bmatrix}\bar{t}^2 & -\bar{t}\\ -\beta^2 & \beta\end{bmatrix}\right)\\
&= \frac{1}{\beta\bar{t}^2-\bar{t}\beta^2}\left[\bar{t}^2\log\beta-\beta^2\log\bar{t},\ \beta\log\bar{t}-\bar{t}\log\beta\right]\\
&= \frac{1}{\bar{t}-\beta}\left[\frac{\log\beta}{\beta}\bar{t}-\frac{\log\bar{t}}{\bar{t}}\beta,\ \frac{\log\bar{t}}{\bar{t}}-\frac{\log\beta}{\beta}\right]. \qquad (41)
\end{aligned}$$

Therefore, we get the following short form of Eq. (19) with $\beta = 1$:

$$\begin{aligned}
\mathrm{ub}(1,\mu_1,\mu_2) &= \frac{\log\bar{t}}{\bar{t}\,(\bar{t}-1)}\,(\mu_2-\mu_1)\\
\boxed{\textit{definition of }\bar{t}} &= \log\left(\frac{\mu_1-\mu_2}{N-\mu_1}\right)\cdot\\
&\quad\left(\frac{\mu_1-\mu_2}{N-\mu_1}\left(\frac{\mu_1-\mu_2}{N-\mu_1}-1\right)\right)^{-1}(\mu_2-\mu_1)\\
\boxed{\textit{simplify}} &= \log\left(\frac{\mu_1-\mu_2}{N-\mu_1}\right)\cdot\\
&\quad\frac{(N-\mu_1)^2(\mu_2-\mu_1)}{(\mu_1-\mu_2)(\mu_1-\mu_2-N+\mu_1)}\\
\boxed{\textit{cancel }\mu_1-\mu_2} &= \log\left(\frac{\mu_1-\mu_2}{N-\mu_1}\right)\frac{(N-\mu_1)^2}{N-2\mu_1+\mu_2}. \qquad (42)
\end{aligned}$$

Let $\tilde{\mu}_2$ with $0 < \tilde{\mu}_2 \le \mu_2$ be a lower bound of the squared Frobenius norm. If we replace μ_2 with $\tilde{\mu}_2$ in Eq. (42), we notice that the log-term increases and the denominator of the second part decreases. This directly leads us to the validity of the lemma. □

Proof of Lemma 2

$$\begin{aligned}
\mathrm{ub}(\gamma\beta,\gamma\mu_1,\gamma^2\mu_2) &= \left[\log\gamma\beta,\ \log\gamma\bar{t}\right]\cdot\\
&\quad\left(\begin{bmatrix}\gamma & 0\\ 0 & \gamma^2\end{bmatrix}\begin{bmatrix}\beta & \bar{t}\\ \beta^2 & \bar{t}^2\end{bmatrix}\right)^{-1}\begin{bmatrix}\gamma & 0\\ 0 & \gamma^2\end{bmatrix}\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}\\
&= \left(\left[\log\beta,\ \log\bar{t}\right]+\left[\log\gamma,\ \log\gamma\right]\right)\cdot\\
&\quad\begin{bmatrix}\beta & \bar{t}\\ \beta^2 & \bar{t}^2\end{bmatrix}^{-1}\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}\\
\boxed{\textit{definition of ub}} &= \mathrm{ub}(\beta,\mu_1,\mu_2)+\left[\log\gamma,\ \log\gamma\right]\cdot\\
&\quad\begin{bmatrix}\beta & \bar{t}\\ \beta^2 & \bar{t}^2\end{bmatrix}^{-1}\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}\\
&= \mathrm{ub}(\beta,\mu_1,\mu_2)\ +\ \tilde{\mathrm{ub}}_\gamma(\beta,\mu_1,\mu_2). \qquad (43)
\end{aligned}$$

Now, we show that the second term equals to $N\cdot\log\gamma$ by using the definition of $\bar{t}$ and the calculation of the weights for μ_1 and μ_2 as done in the beginning of the proof of Lemma 1:

$$\begin{aligned}
\tilde{\mathrm{ub}}_\gamma(\beta,\mu_1,\mu_2) &= (\log\gamma)\left[1,\ 1\right]\cdot\begin{bmatrix}\beta & \bar{t}\\ \beta^2 & \bar{t}^2\end{bmatrix}^{-1}\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}\\
\boxed{\textit{see proof of L1}} &= \frac{\log\gamma}{\bar{t}-\beta}\left[\frac{\bar{t}}{\beta}-\frac{\beta}{\bar{t}},\ \frac{1}{\bar{t}}-\frac{1}{\beta}\right]\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}\\
&= \frac{\log\gamma}{(\bar{t}-\beta)\,\bar{t}\beta}\left[\bar{t}^2-\beta^2,\ \beta-\bar{t}\right]\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}\\
&= \frac{\log\gamma}{\bar{t}\beta}\left[\bar{t}+\beta,\ -1\right]\begin{bmatrix}\mu_1\\ \mu_2\end{bmatrix}\\
&= \frac{\log\gamma}{\bar{t}\beta}\left(\left(\bar{t}+\beta\right)\mu_1-\mu_2\right)\\
\boxed{\textit{definition of }\bar{t}} &= (\log\gamma)\,\frac{\beta N-\mu_1}{\beta^2\mu_1-\beta\mu_2}\cdot\\
&\quad\left(\left(\frac{\beta\mu_1-\mu_2+\beta^2N-\beta\mu_1}{\beta N-\mu_1}\right)\mu_1-\mu_2\right)\\
&= (\log\gamma)\,\frac{-\mu_1\mu_2+\beta^2N\mu_1-\beta N\mu_2+\mu_1\mu_2}{\beta^2\mu_1-\beta\mu_2}\\
&= (\log\gamma)\,\frac{\beta^2N\mu_1-\beta N\mu_2}{\beta^2\mu_1-\beta\mu_2}\\
&= N\cdot\log\gamma. \qquad (44)
\end{aligned}$$

□

Proof of Theorem 2 The first part of the inequality was proved by Bai and Golub (1997) and the proof for the second part is straightforward by applying Lemma 2 with $\gamma = \frac{1}{\beta}$ followed by using Lemma 1:

$$\begin{aligned}
\mathrm{ub}(\beta,\mu_1,\mu_2) &= \mathrm{ub}\left(1,\frac{\mu_1}{\beta},\frac{\mu_2}{\beta^2}\right)-N\cdot\log\left(\frac{1}{\beta}\right) \quad \boxed{L2}\\
&\le \mathrm{ub}\left(1,\frac{\mu_1}{\beta},\frac{\tilde{\mu}_2}{\beta^2}\right)-N\cdot\log\left(\frac{1}{\beta}\right) \quad \boxed{L1}\\
&= \mathrm{ub}(\beta,\mu_1,\tilde{\mu}_2). \quad \boxed{L2} \qquad (45)
\end{aligned}$$

□

Bounds on Quadratic Forms

From linear algebra we know that any real-valued, symmetric matrix $\boldsymbol{M}\in\mathbb{R}^{N\times N}$ can be transformed into $\boldsymbol{M}=\boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^{\mathrm{T}}$ where $\boldsymbol{D}$ is a positive definite diagonal matrix containing the eigenvalues of $\boldsymbol{M}$ and $\boldsymbol{U}$ is an orthogonal matrix of the same size as $\boldsymbol{M}$. Therefore, we notice that for any vector $\boldsymbol{x}\in\mathbb{R}^N$ the following holds:

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x}=\boldsymbol{x}^{\mathrm{T}}\boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{x}=\tilde{\boldsymbol{x}}^{\mathrm{T}}\boldsymbol{D}\tilde{\boldsymbol{x}}=\sum_{i=1}^{N}\lambda_i\tilde{\boldsymbol{x}}\,(i)^2\,. \qquad (46)$$

We denoted with λ_i the decreasingly ordered eigenvalues of $\boldsymbol{M}$, i.e., $\lambda_1 \geq \cdots \geq \lambda_N$, and $\tilde{x}_i$ contains the projection of $\boldsymbol{x}$ onto the ith column of $\boldsymbol{U}$, which is the ith eigenvector of $\boldsymbol{M}$. As a result, we can bound the quadratic form in Eq. (46) as follows:

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x} = \sum_{i=1}^{k} \lambda_i \tilde{\boldsymbol{x}}(i)^2 + \sum_{j=k+1}^{N} \lambda_j \tilde{\boldsymbol{x}}(j)^2 \tag{47}$$

$$\leq \sum_{i=1}^{k} \lambda_i \tilde{\boldsymbol{x}}(i)^2 + \lambda_{k+1} \sum_{j=k+1}^{N} \tilde{\boldsymbol{x}}(j)^2 . \tag{48}$$

Since $\boldsymbol{U}$ is an orthonormal basis, it does not influence the length of vectors, i.e., $||\boldsymbol{U}\boldsymbol{x}|| = ||\boldsymbol{x}||$. Therefore, we can obtain the following upper bound:

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x} \leq \sum_{i=1}^{k} \lambda_i \tilde{\boldsymbol{x}}(i)^2 + \lambda_{k+1}\left(||\boldsymbol{x}||^2 - \sum_{i=1}^{k} \tilde{\boldsymbol{x}}(i)^2\right). \tag{49}$$

Equivalently, we get the following lower bound considering the k smallest eigenvalues of $\boldsymbol{M}$ and bounding the remaining eigenvalues with the $(k+1)$th smallest:

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x} \geq \sum_{j=N-k+1}^{N} \lambda_j \tilde{\boldsymbol{x}}(j)^2 + \lambda_{N-k}\left(||\boldsymbol{x}||^2 - \sum_{j=N-k+1}^{N} \tilde{\boldsymbol{x}}(j)^2\right). \tag{50}$$

For the special cases of $k = 0$ in Eq. (49) and Eq. (50), we obtain the well known bounds for any positive definite matrix $\boldsymbol{M}$ and any vector $\boldsymbol{x}$ of corresponding size:

$$\lambda_N(\boldsymbol{M})\,||\boldsymbol{x}||^2 \leq \boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x} \leq \lambda_1(\boldsymbol{M})\,||\boldsymbol{x}||^2 . \tag{51}$$

References

Ablavsky, V., & Sclaroff, S. (2011). Learning parameterized histogram kernels on the simplex manifold for image and action classification. In *IEEE international conference of computer vision (ICCV)* pp. 1473–1480.

Bai, Z., & Golub, G. H. (1997). Bounds for the trace of the inverse and the determinant of symmetric positive definite matrices. *Annals of Numerical Mathematics*, *4*(1–4), 29–38.

Barla, A., Odone, F., & Verri, A. (2003). Histogram intersection kernel for image classification. In *IEEE international conference on image processing (ICIP)*, pp. 513–516.

Berg, A.C., Deng, J., & Fei-Fei, L. (2010). Large scale visual recognition challenge. http://www.imagenet.org/challenges/LSVRC/2010/.

Bo, L., & Sminchisescu, C. (2008). Greedy block coordinate descent for large scale gaussian process regression. In *Uncertainty in artificial intelligence (UAI)*.

Bo, L., & Sminchisescu, C. (2010). Twin gaussian processes for structured prediction. *International Journal of Computer Vision (IJCV)*, *87*(1–2), 28–52.

Bo, L., & Sminchisescu, C. (2012). Greedy block coordinate descent for large scale gaussian process regression. Computing Research Repository (CoRR) abs/1206.3238, previous publication in Uncertainty for Artificial Intelligence (UAI).

Bonilla, E.V., Chai, K.M.A., & Williams, C.K.I. (2008). Multi-task gaussian process prediction. In *Advances in Neural Information Processing Systems (NIPS)* (pp. 153–160). Cambridge: MIT Press.

Bottou, L., Chapelle, O., DeCoste, D., & Weston, J. (Eds.). (2007). *Large-scale kernel machines*. Cambridge: MIT Press.

Boughorbel, S., Tarel, J.P., & Boujemaa, N. (2005). Generalized histogram intersection kernel for image recognition. In *IEEE international conference on image processing (ICIP)*, pp. 161–164.

Chang, C. C., & Lin, C. J. (2011). Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, *2*, 1–27.

Deng, J., Berg, A., Li, K., & Fei-Fei, L. (2010). What does classifying more than 10,000 image categories tell us? In *European Conference on Computer Vision (ECCV)*, pp. 71–84.

Donahue, J., Jia, Y., Vinyals, O., Hoffman, J., Zhang, N., Tzeng, E., & Darrell, T. (2013). Decaf: A deep convolutional activation feature for generic visual recognition. arXiv preprint arXiv:1310.1531.

Ebert, S., Fritz, M., & Schiele, B. (2012). Ralf: A reinforced active learning formulation for object class recognition. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 3626–3633.

Fan, R. E., Chang, K. W., Hsieh, C. J., Wang, X. R., & Lin, C. J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research (JMLR)*, *9*, 1871–1874.

Freytag, A., Fröhlich, B., Rodner, E., & Denzler, J. (2012a). Efficient semantic segmentation with gaussian processes and histogram intersection kernels. In *International conference on pattern recognition (ICPR)*, pp. 3313–3316.

Freytag, A., Rodner, E., Bodesheim, P., & Denzler, J. (2012b). Rapid uncertainty computation with gaussian processes and histogram intersection kernels. In: *Asian conference on computer vision (ACCV)*, pp. 511–524.

Freytag, A., Rodner, E., Bodesheim, P., & Denzler, J. (2013). Labeling examples that matter: Relevance-based active learning with gaussian processes. In *German conference on pattern recognition (GCPR)*, pp. 282–291.

Freytag, A., Rodner, E., & Denzler, J. (2014a). Selecting influential examples: Active learning with expected model output changes. In: *European conference on computer vision (ECCV)*.

Freytag, A., Rühle, J., Bodesheim, P., Rodner, E., & Denzler, J. (2014b). Seeing through bag-of-visual-word glasses: towards understanding quantization effects in feature extraction methods. In: *International conference on pattern recognition (ICPR)—FEAST workshop*.

Grauman, K., & Darrell, T. (2007). The pyramid match kernel: Efficient learning with sets of features. *Journal of Machine Learning Research (JMLR)*, *8*(Apr), 725–760.

He, H., & Siu, W.C. (2011). Single image super-resolution using gaussian process regression. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 449–456.

Hestenes, M. R., & Stiefel, E. (1952). Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, *49*(6), 409–436.

Käding, C., Freytag, A., Rodner, E., Bodesheim, P., & Denzler, J. (2015). Active learning and discovery of object categories in the presence of unnameable instances. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 4343–4352.

Kapoor, A., Grauman, K., Urtasun, R., & Darrell, T. (2010). Gaussian processes for object categorization. *International Journal of Computer Vision (IJCV)*, *88*(2), 169–188.

Kemmler, M., Rodner, E., & Denzler, J. (2010). One-class classification with gaussian processes. In: *Asian conference on computer vision (ACCV)*, vol. 2, pp. 489–500.

Krizhevsky, A., Sutskever, I., & Hinton, G.E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*.

Lazebnik, S., Schmid, C., & Ponce, J. (2006). Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 2169–2178.

Maji, S., Berg, A.C., & Malik, J. (2008). Classification using intersection kernel support vector machines is efficient. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 1–8.

Maji, S., Berg, A. C., & Malik, J. (2013). Efficient classification for additive kernel svms. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *35*(1), 66–77.

Nelder, J. A., & Mead, R. (1965). A simplex method for function minimization. *Computer Journal*, *7*(4), 308–313.

Nickisch, H., & Rasmussen, C. E. (2008). Approximations for binary gaussian process classification. *Journal of Machine Learning Research*, *9*(10), 2035–2078.

Nocedal, J., & Wright, S. J. (2006). *Conjugate gradient methods*. New York: Springer.

Perronnin, F., Akata, Z., Harchaoui, Z., & Schmid, C. (2012). Towards good practice in large-scale learning for image classification. In *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 3482–3489.

Pillonetto, G., Dinuzzo, F., & Nicolao, G. D. (2010). Bayesian online multitask learning of gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *32*(2), 193–205.

Quattoni, A., & Torralba, A. (2009). Recognizing indoor scenes. In: *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 413–420.

Quiñonero Candela, J., & Rasmussen, C. E. (2005). A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research (JMLR)*, *6*, 1939–1959.

Rasmussen, C. E., & Williams, C. K. I. (2006). *Gaussian processes for machine learning. Adaptive computation and machine learning*. Cambridge: The MIT Press.

Rodner, E. (2011). Learning from few examples for visual recognition problems. Hut Verlag München, http://herakles.inf-cv.uni-jena.de/biborb/bibs/cv/papers/Rodner11:Diss.pdf.

Rodner, E., Hegazy, D., & Denzler, J. (2010). Multiple kernel gaussian process classification for generic 3d object recognition from time-of-flight images. In: *International conference on image and vision computing New Zealand (IVCNZ)*, pp. 1–8.

Rodner, E., Freytag, A., Bodesheim, P., & Denzler, J. (2012). Large-scale gaussian process classification with flexible adaptive histogram kernels. In *European conference on computer vision (ECCV)*, vol. 4, pp. 85–98.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*, 1–42.

Schölkopf, B., & Smola, A. J. (2001). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. Cambridge: MIT Press.

Snoek, J., Larochelle, H., & Adams, R.P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 2951–2959.

Sun, J., & Ponce, J. (2013). Learning discriminative part detectors for image classification and cosegmentation. In *IEEE International conference on computer vision (ICCV)*.

Tong, S., & Koller, D. (2001). Support vector machine active learning with applications to text classification. *Journal of Machine Learning Research (JMLR)*, *2*(Nov), 45–66.

Urtasun, R., & Darrell, T. (2008). Sparse probabilistic regression for activity-independent human pose inference. In: *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 1–8.

Vapnik, V. (1998). *Statistical learning theory* (Vol. 2). New York: Wiley.

Vázquez, D., Marin, J., Lopez, A. M., Geronimo, D., & Ponsa, D. (2014). Virtual and real world adaptationfor pedestrian detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *36*(4), 797–809.

Vedaldi, A., & Zisserman, A. (2010). Efficient additive kernels via explicit feature maps. In: *IEEE conference on computer vision and pattern recognition (CVPR)*, pp. 3539–3546.

Vedaldi, A., Gulshan, V., Varma, M., & Zisserman, A. (2009). Multiple kernels for object detection. In *IEEE international conference on computer vision (ICCV)*, pp. 606–613.

Wang, G., Hoiem, D., & Forsyth, D. (2012). Learning image similarity from flickr groups using fast kernel machines. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, *34*(11), 2177–2188.

Williams, C.K.I., & Seeger, M. (2000). The effect of the input density distribution on kernel-based classifiers. In: *International Conference on Machine Learning (ICML)*, pp. 1159–1166.

Wu, J. (2010). A fast dual method for hik svm learning. In: *European conference on computer vision (ECCV)*, pp. 552–565.

Wu, J. (2012). Efficient hik svm learning for image classification. *IEEE Transactions on Image Processing (TIP)*, *21*(10), 4442–4453.

Yuan, Q., Thangali, A., Ablavsky, V., & Sclaroff, S. (2008). Multiplicative kernels: Object detection, segmentation and pose estimation. In *Computer vision and pattern recognition (CVPR), IEEE*, pp. 1–8.

Yuster, R. (2008). Matrix sparsification for rank and determinant computations via nested dissection. In: *IEEE symposium on foundations of computer science (FOCS)*, pp. 137–145.