Bayesian Optimization with an Empirical Hardness Model for Approximate Nearest Neighbour Search

Julieta Martinez James J. Little Nando de Freitas
University of British Columbia
{julm, little, nando}@cs.ubc.ca

Abstract

Nearest Neighbour Search in high-dimensional spaces is a common problem in Computer Vision. Although no algorithm better than linear search is known, approximate algorithms are commonly used to tackle this problem. The drawback of using such algorithms is that their performance depends highly on parameter tuning. While this process can be automated using standard empirical optimization techniques, tuning is still time-consuming. In this paper, we propose to use Empirical Hardness Models to reduce the number of parameter configurations that Bayesian Optimization has to try, speeding up the optimization process. Evaluation on standard benchmarks of SIFT and GIST descriptors shows the viability of our approach.

1. Introduction

Finding the Nearest Neighbour of vectors in high-dimensional spaces is a common problem in Computer Vision. It is a central and time-consuming part of non-parametric classification [3, 22, 23], and different indexing and retrieval techniques [9, 19]. While no algorithm better than linear search is known, numerous Approximate algorithms for Nearest Neighbour search (ANN) have been proposed [18, 24].

It is well known that parameter tuning is crucial for the performance of many algorithms commonly used in Computer Vision. In fact, it has been noted that richly parameterized models can exhibit performance that ranges from chance to state-of-the-art depending solely on their tuning [5]. Nearest Neighbour Search is not the exception; as noted by Muja and Lowe [13], the performance of different ANN algorithms varies widely depending on the characteristics of a dataset such as correlation, dimensionality, cluster prevalence and size, as well as on the parameters of the algorithms themselves. This makes it hard to find the best algorithm and its optimal parameters for arbitrary datasets. Moreover, since the performance of an algorithm depends on both its intrinsic quality and its parameter tuning, it is not always entirely clear whether new ANN algorithms are better than classical ones. In this paper, we investigate the optimization of rather classical algorithms implemented in FLANN, a pervasive ANN library that has been extensively used as a benchmark for new ANN methods.

Motivated by the relatively recent success of Bayesian Optimization as a powerful tool to optimize expensive-to-evaluate functions [25, 2], in this paper we explore, for the first time, its applicability to boost the performance of ANN algorithms. We also propose a novel procedure to speed up the optimization process: by learning an extended Empirical Hardness Model (EHM) [11] offline, we obtain a proxy to the target function, whose evaluation can be bypassed depending on the quality of the EHM. We test the feasibility of our approach on large datasets of SIFT and GIST descriptors, and show improvements in speedup over linear search compared to plain Bayesian Optimization and Grid Search – the current method implemented by FLANN.

The rest of this paper is organized as follows: in Section 2 we briefly review FLANN, Bayesian Optimization and Empirical Hardness Models. In Section 3 we formally postulate the problem and in Section 4 we expose the proposed solution. In Section 5 we describe our experimental setup, and in Section 6 we analyze the experimental results. Finally, we conclude and outline future work in Section 7.

2. Background and Related Work

In 2009 Muja and Lowe [13] published FLANN, a C++ library intended to speed up the matching of nearest neighbours in high-dimensional vector spaces with a certain trade-off in precision. FLANN incorporates two algorithms for search: a randomized kd-tree as described in [18] and a hierarchical k-means clustering novel to the FLANN implementation. The authors found that the performance of each algorithm varies widely depending on properties of the dataset such as dimensionality, correlations, cluster preva-

This work was supported in part by the Institute for Computing, Information and Cognitive Systems (ICICS) at UBC.

lence and size, as well as the desired precision. Since choosing and configuring the algorithm that will work best for a given dataset can be hard for an inexperienced user, they also offered a functionality to do this automatically. Finding the best configuration of each algorithm is posed as an optimization problem in parameter space, which can be solved via standard optimization techniques.

FLANN was released as open source and since then has become widely popular. It is currently integrated with the Computer Vision framework OpenCV and with the Robotics Operating System (ROS) [17], and benefits of an active community on GitHub.[1] FLANN has also been widely used as a baseline to demonstrate the effectiveness of new ANN algorithms [9, 24], and it is often outperformed only by a small margin. In many comparisons, since FLANN is assumed to automatically choose the best of a randomized kd-tree or a hierarchical k-means index, it is considered to be at least as good as the best of both algorithms.

Despite being remarkably popular, the library performs only Grid Search to pick the best algorithm and its corresponding parameters. We believe that this has hampered the performance of the library in previous benchmarks, and in this paper intend to bring it to its full potential. Moreover, we want this performance to be available to the users at a minimum computational cost.

Finally, even though FLANN aims to be agnostic to the underlying ANN algorithm, and is built to accommodate new implementations, no additional ANN methods have been added to the library since it was published. It is likely that the underlying optimization method is partly to blame, as Grid Search would suffer from a combinatorial explosion if faced with more options. We also expect this work will also shed some light on what the most suitable optimization technique for a larger scale FLANN would be.

2.1. Automatic Algorithm Configuration

Automatic Algorithm Configuration deals with the problem of finding good algorithm parameters for different problems. It is often posed as a special case of black-box optimization: if we let $f : \mathcal{X} \to \mathbb{R}$ be a black-box function (*i.e.*, a function with no closed-form expression or available derivatives), the task at hand is to automatically solve the global optimization problem

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{X}}{\operatorname{argmin}} f(\mathbf{x}). \tag{1}$$

In the context of Algorithm Configuration, $\mathbf{x} \in \mathbb{R}^d$ is a particular configuration of d parameters drawn from the configuration space, $\mathcal{X}$, and $f(\mathbf{x})$ is the performance of the algorithm as defined by a desired metric. In the case of instance-specific algorithm configuration, the metric is usually defined as the performance of the algorithm on a particular problem.

In Computer Vision the most common optimization method is Grid Search, *i.e.*, different uniformly spaced values of $\mathbf{x} \in \mathcal{X}$ are evaluated, and a finer grid is evaluated in areas that seem promising. This is, in fact, the method that FLANN currently implements for parameter tuning. While Grid Search has the advantages of being intuitive, easy to implement and embarrassingly parallelizable, it inevitably suffers from the curse of dimensionality, and has a poor performance for problems whose *intrinsic* dimensionality is low, *i.e.*, when only a few parameters matter (see figure 1 in [1]).

While in this work we explore our approach in the context of ANN algorithms, our method is applicable to general optimization problems.

2.2. Bayesian Optimization for Parameter Tuning

Here we cover Bayesian Optimization with Gaussian Processes very briefly, and direct the unfamiliar reader to more tutorial treatments of the subject [4, 14, 16].

Bayesian Optimization is a powerful technique to optimize expensive-to-evaluate functions. Despite being a relatively new technology, it has demonstrated impressive performance on parameter optimization in a variety of tasks [7, 20, 25, 2]. Bayesian Optimization has two main ingredients: first, a *performance model* of the algorithm whose value is known at certain points, and second, an *acquisition function* that directs the optimization as a trade-off between exploration and exploitation.

Performance model. As a performance model we use Gaussian Processes (GPs). GPs are a popular method to perform non-linear regression. For every entry in the parameter space, a Gaussian distribution estimating the mean and the confidence (i.e. the variance) of the value of the function at a given point is computed. A GP can be completely specified by its mean function m and covariance function k

$$f(\mathbf{x}) \sim \mathrm{GP}(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')). \tag{2}$$

We assume the mean to be the zero function, $m(\mathbf{x}) = 0$, which simplifies notation. We use the squared Gaussian kernel as our covariance function k:

$$k(\mathbf{x}, \mathbf{x}') = \sigma^2 \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\ell^2}\right), \tag{3}$$

where ℓ controls the width of the kernel and σ controls the amount of modelled noise. This kernel incorporates the assumption of smoothness by requiring entries that are close together in parameter space to have similar values, while relaxing the constraint for entries that are farther apart.

[1] https://github.com/mariusmuja/flann

For regression, given a series of previous observations $\{\mathbf{x_{1:t}}, \mathbf{f_{1:t}}\}$, we evaluate a new point $\mathbf{x_{t+1}}$. If $f_{t+1} = f(\mathbf{x_{t+1}})$, by definition $\mathbf{f_{1:t}}$ and $\mathbf{f_{t+1}}$ are jointly Gaussian:

$$\begin{bmatrix} \mathbf{f_{1:t}} \\ \mathbf{f_{t+1}} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} & \mathbf{k} \\ \mathbf{k}^\top & k(\mathbf{x_{t+1}}, \mathbf{x_{t+1}}) \end{bmatrix}\right), \quad (4)$$

where

$$\mathbf{k}^\top = \begin{bmatrix} k(\mathbf{x_{t+1}}, \mathbf{x_1}), & \ldots, & k(\mathbf{x_{t+1}}, \mathbf{x_t}) \end{bmatrix}, \quad (5)$$

and $\mathbf{K}$ is the Gram matrix defined by k on $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_t}$. The predictive distribution is

$$P(\mathbf{f_{t+1}}|\mathbf{f_{1:t}}, \mathbf{x_{1:t+1}}) = \mathcal{N}(\mathbf{f_{t+1}}|\mu(\mathbf{x_{t+1}}), \sigma^2(\mathbf{x_{t+1}})), \quad (6)$$

where

$$\mu(\mathbf{x_{t+1}}) = \mathbf{k}^\top \mathbf{K}^{-1} \mathbf{f_{1:t}}, \quad (7)$$

$$\sigma^2(\mathbf{x_{t+1}}) = k(\mathbf{x_{t+1}}, \mathbf{x_{t+1}}) - \mathbf{k}^\top \mathbf{K}^{-1} \mathbf{k}. \quad (8)$$

Acquisition function. The choice of the next sampling point, $\mathbf{x_{t+1}}$, is defined as the maximum of the acquisition function. The job of the acquisition function is to guide the exploration towards the minimum in as few steps as possible. To do so, it balances the trade-off between *exploiting* points where the objective is likely to be high (i.e. the GP mean is high) and *exploring* others where the uncertainty (i.e. the GP variance) is high. In our experiments we use the GP-Upper Confidence Bound (GP-UCB) [21] as our acquisition function

$$\text{GP-UCB}(\mathbf{x}) = \mu(\mathbf{x}) + \sqrt{\nu\tau_t}\sigma(\mathbf{x}), \quad (9)$$

where $\sqrt{\nu\tau_t}$ controls the importance of the variance (exploration) against the mean (exploitation).

Bayesian Optimization in conjunction with Gaussian Processes has recently gained popularity as an auto-tuning tool for Machine Learning algorithms but, to the best of our knowledge, remains unexplored in the context of ANN algorithm configuration. Evaluating the performance of an ANN algorithm is an expensive operation, taking from several seconds to half an hour depending on the dataset size and algorithm parameters. Therefore, it is critical that we find the right parameters in as few evaluations as possible. This is the reason why we advocate the use of Bayesian Optimization instead of simpler optimization techniques.

2.3. Empirical Hardness Models for Performance Prediction

First proposed by Leyton-Brown *et al*. [11], Empirical Hardness Models (EHMs) are a tool that predict the performance of an algorithm on a particular problem.

Formally, given a set of particular problem instances, $p_i \in \mathcal{P}$, and an algorithm, $a \in \mathcal{A}$, the objective of performance prediction is to build a model that predicts the performance of a when run on an arbitrary $p \in \mathcal{P}$. EHMs frame performance prediction as a machine learning problem, by characterizing each p_i with a set of features, $\mathbf{c}_p = [c_{p1}, c_{p2}, ..., c_{pn}] \in \mathbb{R}^n$, where c_{pi} encode expert knowledge. By treating the performance of a on p as an observation, $y \in \mathbb{R}$, the aim is to learn a function, $\phi : \mathbb{R}^n \to \mathbb{R}$, that allows to predict the performance of algorithm a on any problem instance $p \in \mathcal{P}$.

EHMs are modelling *algorithm* performance. Note that although the name of EHMs suggests that they are modelling how hard a particular problem is, in practice this translates to modelling how hard a particular problem is *for a particular algorithm*. The rationale is that a problem that might be hard to solve for one algorithm might be easy to solve for another, and vice versa.

Extending EHMs to account for algorithm parameters. EHMs work pretty well when the algorithms whose performance is predicted are deterministic and parameter-free. However, this is hardly the case in practice. A workaround to this limitation was proposed in [6], where EHMs were extended to take parameter configurations as input too. In our notation, we extend $\mathbf{c}_p$ to $\mathbf{c}_{p,a} = [c_{p1}, c_{p2}, \ldots, c_{pn}, x_{a1}, x_{a2}, \ldots, x_{am}] \in \mathbb{R}^{n+m}$, where $x_{a1}, x_{a2}, \ldots, x_{am}$ are tunable algorithm parameters. Naturally, the function that we aim to learn is now $\phi : \mathbb{R}^{n+m} \to \mathbb{R}$. A recent evaluation of Machine Learning models for performance prediction has shown both Random Forests and GPs to be the best options [8]. Due to the ease of implementation, we chose GPs to build our EHM.

3. Problem Formulation

ANN algorithms typically offer a trade-off between speedup and build time or memory constraints. Muja and Lowe [13] proposed to define the cost of a particular algorithm conufiguration i as

$$J_i = \frac{s + w_b b}{(s + w_b b)_{opt}} + w_m m. \quad (10)$$

where s is the search cost, the time that it takes to find the nearest neighbours of a randomly sampled set of vectors. As more precision is required by the user, this is expected to increase. The build time cost, b, is the time that it takes to build the index. As the size of the dataset increases, this is expected to increase as well.

The memory cost, $m = m_t/m_d$, is the fraction of the memory used by the index relative to the memory used to store the dataset. The weights, w_b and w_m, are specified

by the user. The build time weight, w_b, controls the ratio of importance of the build time with respect to the search time. The memory weight, w_m, determines the importance given to the memory used by the index. Finally, $(s + w_b b)_{opt} = \min_i(s + w_b b)$, is the minimum search plus build time cost found during (Grid Search) optimization.

3.1. Optimizing Speed for a Fixed Recall

The optimization problem consists of finding the algorithm and its corresponding parameter configuration that achieves a minimum J_i. For a given set of vectors $\mathbf{\Theta}$, we randomly sample a subset $\boldsymbol{\theta}$ and optimize for the speed of a fixed recall@1, i.e. we build an index for $\mathbf{\Theta}$ and for each vector $v \in \boldsymbol{\theta}$ we search for its nearest neighbor in $\mathbf{\Theta}$, until at least a fraction ρ of them are correct: our goal is to perform this task as fast as possible. This is the original metric that FLANN aims to optimize, although in [13] recall@1 is referred to as *precision*.

4. Solution Methodology

Our proposed solution consists of two parts. First, offline, we build an EHM for each of the two ANN algorithms that FLANN implements. Second, online, we use this model to get approximate evaluations of parameter configurations, and use the evaluations to speed up the optimization on particular instances.

4.1. Bayesian Optimization as Performance Prediction

It is important to remember that Bayesian Optimization builds a model of the current function it is trying to optimize. In our case, the model is given by the Gaussian Process at time t, which is updated in every iteration by the observation at time $t + 1$. It should be straighforward to see, then, that *Bayesian Optimization is also in the business of performance prediction*. In fact, if we could use equation (6) to correctly predict the value of any arbitrary $\mathbf{f_{t+1}}$, then we would be effectively predicting the performance of any algorithm configuration. If we think of the algorithm selection as a categorical parameter, then obtaining an accurate GP model is equivalent to both a) converging the GP optimization, and b) accurately predicting the performance of different algorithms (and different parameter configurations) on a particular problem.

4.2. Empirical Hardness Models as Parameter Tuning

It is also important to note that Empirical Hardness Models can be though of as a method for parameter tuning. In fact, if we let the model include algorithm parameters, then EHMs are effectively predicting the performance of different algorithms and different parameter configurations on a particular problem. Particularly, finding the best algorithm configuration amounts to finding the argmin of the EHM.

4.3. Should we trust the model?

Now that we have seen that both Bayesian Optimization and EHMs can be seen as a tool to tackle the same problem, it seems natural to ask: how can they help each other? Suppose that we have a series of EHMs learned offline: should we trust this model during Bayesian Optimization? On one hand, if the answer is "yes" then the natural way to proceed is to not query the original function at all. According to this logic, if the EHMs have all the answers, then the configuration that minimizes the value of the EHM *should* be the optimal configuration that Bayesian Optimization is looking for. On the other hand, if the answer is "no", then perhaps we should throw the EHM away, since it can only mislead and hamper the optimization process.

The former two extreme cases rarely happen in practice. In reality, the EHMs that we learn will not predict the performance of different algorithms perfectly on every instance, but will rather be an approximation. Moreover, the quality of such approximation will most likely be unknown during runtime, and it will possibly be accurate for some configurations, and inaccurate for others. Therefore, it seems more natural to assess the quality of the model online, and find out, based on empirical evidence, whether the model learnt offline is appropriate to tune the instance at hand. It is in this spirit that we propose algorithm 1.

Algorithm 1 Warm-starting BO

Require: γ, the *a priori* confidence. β, the EHM confidence tolerance. α, the confidence change pace.

1: **for** each Bayesian Optimization iteration i **do**
2: $p \sim$ uniformly at random in $[0, 1]$
3: **if** $p > \gamma$ **then**
4: $m \leftarrow$ EHM value
5: $q \leftarrow$ function value
6: **if** $|m - q| > \beta$ **then**
7: $\gamma \leftarrow \min(0.001, \gamma * \alpha)$
8: **else**
9: $\gamma \leftarrow \max(0.999, \gamma * (1 + \alpha))$
10: **else**
11: $q \leftarrow$ model value
12: Continue the optimization iteration normally

Intuitively, we are testing the quality of the model with probability inverse to the current confidence. In line 6 we probe the difference between what the model predicts and what the value of the actual function is. The rest of the algorithm takes care of keeping a belief of the confidence in the EHM. The proposed algorithm is general enough to consider the two extreme cases of model confidence outlined before. Absolute confidence in the model can be expressed

with the parameters $\gamma = 1, \beta = 1$ and α being irrelevant, while a total lack of confidence in the model will be given by parameter values $\gamma = 0, \beta = 0$, and α being irrelevant as well. Values in between account for different initial degrees of confidence in the EHM.

Search initialization. Note that algorithm 1 does not dictate how the first query to the function should be made. In practice, this is usually user-specified or set at random. We make the first query at the middle value of the respective configuration spaces.

Hyperparameter optimization. In Bayesian Optimization, it is often hard to estimate the true width of the Gaussian kernel ℓ. While ℓ lets the user input their knowledge about the problem, in the context of black-box optimization the user often knows little about the problem in the first place. The width of the kernel is often set by maximizing marginal likelihood, or estimated using Grid Search. We follow the approach of [25], exploring the bounds, $\ell_{\text{lb}} = 0.01$ and $\ell_{\text{ub}} = 50$, every 10 iterations.

Parameter values. In the Gaussian kernel we set $\sigma = 0.01$, and in GP-UCB $\sqrt{\nu\tau_t} = 10$. For algorithm 1, we use $gamma = 0.5$, $\beta = 0.1$ and $\alpha = 0.1$.

5. Experimental Setup

We evaluate our approach empirically. We compare Grid Search (the current method implemented in FLANN), against plain GP-UCB (In Figures 1 and 2 "Bayesian Optimization") and GP-UCB using the EHM trained in batch (denoted "W-BO", as for "warm-started" Bayesian Optimization). For completeness, we also compare against Random Search and SMAC [7], a competing optimization approach. We set w_b and w_m to zero in our objective function (eq. 10), and therefore proceed to optimize for the best speedup.

5.1. Parameter Spaces

As implemented in FLANN, a kd-tree index can be parameterized by the number of dimensions with largest variance over which a tree split is considered, n, and the number of trees it contains, t, while a hierarchical k-means index is parameterized by the clustering method, c, a branching factor, b, and the number of clustering iterations it performs, i. We define 2 parameter spaces to perform optimization, summarized in Table 1. Both n and c are fixed on values that were noted to work well in [13]. While with Bayesian Optimization we can explore arbitrary integers (full configuration), the current optimization is constrained to a set of predefined values outlined in the partial configuration.

Parameter		Partial Conf.	Full Conf.
kd-tree	n	{5}	{5}
	t	{1, 4, 8, 16, 32}	{1, 2, ..., 64}
k-means	c	{random}	{random}
	b	{16, 32, 64, 128, 256}	{2, 3, ..., 512}
	i	{1, 5, 10, 15}	{1, 2, ..., 32}

Table 1. Different parameter spaces for FLANN. Grid Search, the current method implemented by the library, explores all the combinations of the partial configuration, while in Bayesian Optimization we can explore the full configuration space.

In our experiments, we let Bayesian Optimization run for 100 iterations, while Grid search is limited to $5+5\times4 = 25$ trials (which is what currently FLANN does). Nonetheless, the final comparison is made at 25 iterations to keep it fair.

Since there is randomness in both the ANN search and the optimization algorithms, we made 5 runs of each optimization method and report the mean and, for visual clarity, a half standard deviation in our results.

5.2. Data

We evaluate the performance of the different optimization techniques on two different datasets. The first one is SIFT1M, introduced in [9]. It consists of one million 128-dimensional SIFT [12] descriptors. To observe the effects of dataset size, we randomly sampled groups of 10,000, and 100,000 descriptors from the original dataset. For the second dataset, we took the 60,000 images from CIFAR-10 [10] and computed 384-dimensional GIST [15] descriptors from them. Later, we computed PCA and kept the 64 principal components, which is a common practice that reduces complexity while maintaining a competitive performance. Similarly, we randomly sampled a smaller dataset of 6,000 vectors to see the effect of dataset size increase. In each dataset, we took a sample $\boldsymbol{\theta}$ of 1,000 vectors and optimized for speed as described in Section 3.1. We show results for a fixed recall@1 of 0.6 and 0.9.

5.3. Dataset Features

To train the batch EHMs, it is not only necessary to define the parameter spaces, but also the problem features that will be used in the model. Instead of coming up with features ourselves, we note that important features have been pointed out in the past. In fact the original FLANN paper [13] suggests that dimensionality, size, clustering characteristics and correlations play an important role in the performance of different algorithms. Next, we explain how we compute these features.

Dimensionality and size. This is straightforward. It corresponds to the number of datapoints and their dimension-

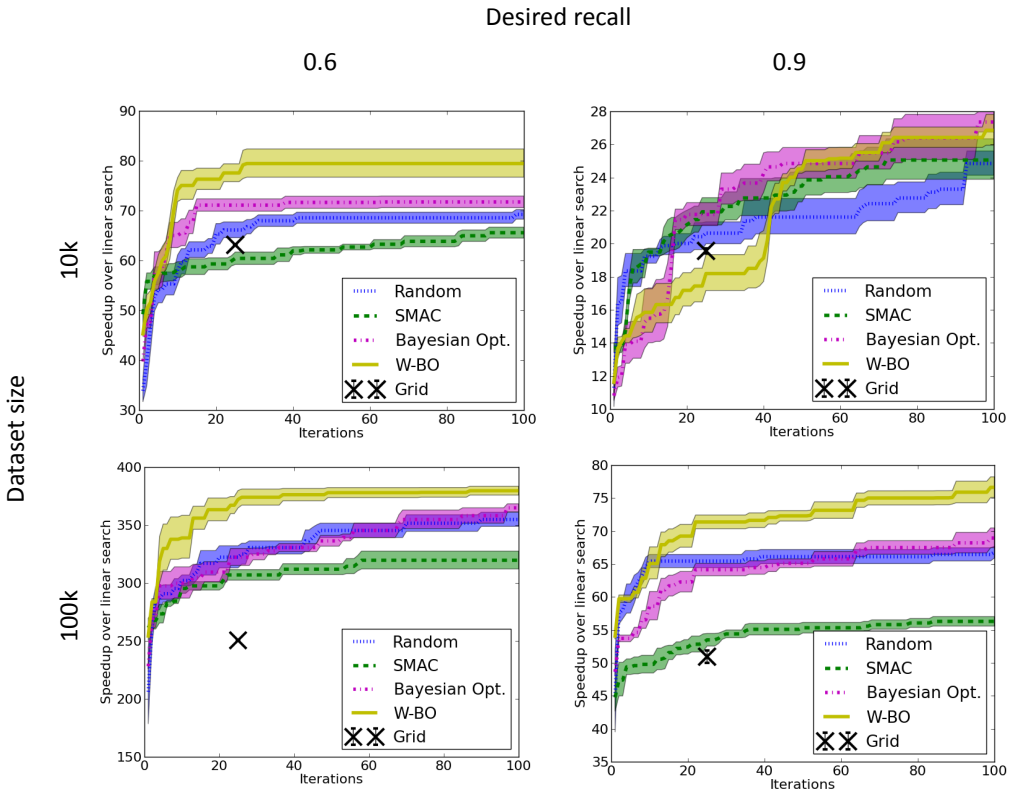

Figure 1. Performance on the 100k SIFT dataset under different optimizers.

ality, which are assumed to be known in advance.

Correlation. We use the simple correlation coefficient to measure the clustering between dimensions i and j, given by $P_{ij} = C_{ij}/\sqrt{C_{ii} \times C_{jj}}$, where C stands for the covariance matrix. This returns a value between -1 and 1 for each dimension pair. We noticed that taking the mean of all these values results in numbers very close to zero. Therefore, we take all the positive values and keep the mean, and do the same for negative values, resulting in 2 features to account for clustering.

Clustering characteristics. Clustering is accounted for by measuring the k-means clustering quantization distortion. This is defined as $\sum_{i=1}^{n} ||v_i - c_i||^2$, where v_i is each vector, and c_i is the k-means cluster centroid assigned to such vector. We use $\lceil n^{1/2} \rceil$ as the number of cluster centers, and perform 5 k-means iterations on the data.

Desired recall (ρ). This value is passed by the user. While it is not a characteristic of the dataset or of the algorithm, it turns out to affect the performance, and must therefore be passed to the EHM as well. We test for values of 0.6 and 0.9, as in [13].

5.4. Building the EHM

To build the EHMs, we took disjoint random subsamples from the SIFT1M dataset, and used PCA-projected 384-dimensional GIST descriptors from CIFAR-100 [10], a dataset similar in spirit to CIFAR-10. We tried 500 random configurations on each dataset, resulting in $16 \times 500 = 8,000$ configurations to train. We then fit a Gaussian Process to this data, and set its hyperparameters via 10-fold cross-validation.

6. Experimental Results

Results for the SIFT1M and CIFAR-10 datasets are summarized in Figures 1 and 2.

The performance of plain Bayesian Optimization is already quite good, achieving a 16% improvement in speedup over linear search when compared to Grid Search for the same number of iterations, which is the current performance of FLANN. Using the Empirical Hardness Model (W-BO) does improve Bayesian Optimization in 3 of the 4 cases

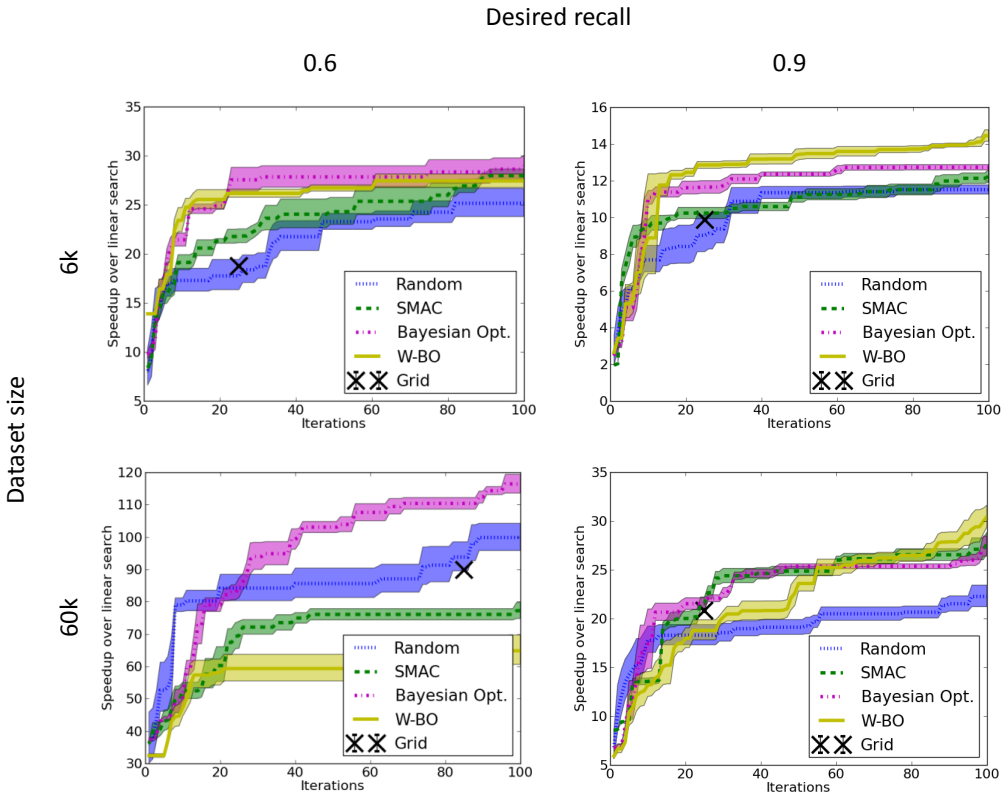

Figure 2. Performance on the CIFAR-10 dataset under different optimizers.

for the SIFT1M dataset, and in 2 cases for the CIFAR-10 dataset. Overall, W-BO is 18% better than linear search for the same number of iterations.

We hypothesize that the EHM hampers Bayesian Optimization in CIFAR-10 because the classes used to train the on 64-dimensional data were drawn from a different dataset. If in practice we want to ship FLANN with a built-in performance model, this is a major drawback. It is, however, promising that the optimization is improved in 2 out of 4 cases, particularly for size 6k and $\rho = 0.9$, where plain Bayesian Optimization does not reach W-BO even after 100 iterations.

7. Conclusions and Future Work

We have demonstrated that FLANN can benefit from a better optimization technique than its current Grid Search. We have also introduced a new algorithm that incorporates Empirical Hardness Models into the Bayesian Optimization loop, and demonstrated that this can be used to find better configurations with less queries to the objective function.

Since FLANN has been compared in previous benchmarks using only Grid Search for parameter tuning, we think that a more thorough evaluation – one that takes into account parameter tuning – should be performed to settle the score between FLANN and other ANN approaches. Finally, we also want to make improvements to our warm-starting algorithm to account for model confidence; after all, it does not make much sense to lose confidence in a model that was not confident of its predictions in the first place. These are all interesting areas for future work.

Acknowledgements

The authors would like to thank Sancho McCann and Marius Muja for helpful discussions and valuable suggestions; and Matthew Hoffman for providing his implementation of Bayesian Optimization with Gaussian Processes.

References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13:281–305, 2012. 2

[2] J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization

in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 115–123, 2013. 1, 2

[3] O. Boiman, E. Shechtman, and M. Irani. In defense of nearest-neighbor based image classification. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 1

[4] E. Brochu, V. M. Cora, and N. de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*, 2010. 2

[5] D. Cox and N. Pinto. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. In *Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*, pages 8–15. IEEE, 2011. 1

[6] F. Hutter, Y. Hamadi, H. H. Hoos, and K. Leyton-Brown. Performance prediction and automated tuning of randomized and parametric algorithms. In *Principles and Practice of Constraint Programming-CP 2006*, pages 213–228. Springer, 2006. 3

[7] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, page 507523, 2011. 2, 5

[8] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 2013. 3

[9] H. Jégou, M. Douze, and C. Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, Jan. 2011. QUAERO. 1, 2, 5

[10] A. Krizhevsky. Learning multiple layers of features from tiny images. *Technical Report, Department of Computer Science, University of Toronto*, 2009. 5, 6

[11] K. Leyton-Brown, E. Nudelman, and Y. Shoham. Learning the empirical hardness of optimization problems: The case of combinatorial auctions. In *Principles and Practice of Constraint Programming-CP 2002*, pages 556–572. Springer, 2002. 1, 3

[12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004. 5

[13] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP (1)*, pages 331–340, 2009. 1, 3, 4, 5, 6

[14] K. P. Murphy. *Machine learning: a probabilistic perspective*. The MIT Press, 2012. 2

[15] A. Oliva and A. Torralba. Building the gist of a scene: The role of global image features in recognition. *Progress in brain research*, 155:23–36, 2006. 5

[16] M. A. Osborne, R. Garnett, and S. J. Roberts. Gaussian processes for global optimization. In *3rd international conference on learning and intelligent optimization (LION3)*, pages 1–15, 2009. 2

[17] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, 2009. 2

[18] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 1

[19] J. Sivic and A. Zisserman. Video google: A text retrieval approach to object matching in videos. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1470–1477. IEEE, 2003. 1

[20] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Neural Information Processing Systems*, 2012. 2

[21] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. *arXiv preprint arXiv:0912.3995*, 2009. 3

[22] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(11):1958–1970, 2008. 1

[23] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. 1

[24] J. Wang, N. Wang, Y. Jia, J. Li, G. Zeng, H. Zha, and X. Hua. Trinary-projection trees for approximate nearest neighbor search. 2013. 1, 2

[25] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. De Freitas. Bayesian optimization in high dimensions via random embeddings. In *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pages 1778–1784. AAAI Press, 2013. 1, 2, 5