Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures

Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures

J. Bergstra BERGSTRA@ROWLAND.HARVARD.EDU
Rowland Institute at Harvard
100 Edwin H. Land Boulevard
Cambridge, MA 02142, USA

D. Yamins YAMINS@MIT.EDU
Department of Brain and Cognitive Sciences
Massachusetts Institute of Technology
Cambridge, MA 02139, USA

D. D. Cox DAVIDCOX@FAS.HARVARD.EDU
Rowland Institute at Harvard
100 Edwin H. Land Boulevard
Cambridge, MA 02142, USA

Abstract

Many computer vision algorithms depend on configuration settings that are typically hand-tuned in the course of evaluating the algorithm for a particular data set. While such parameter tuning is often presented as being incidental to the algorithm, correctly setting these parameter choices is frequently critical to realizing a method's full potential. Compounding matters, these parameters often must be re-tuned when the algorithm is applied to a new problem domain, and the tuning process itself often depends on personal experience and intuition in ways that are hard to quantify or describe. Since the performance of a given technique depends on both the fundamental quality of the algorithm and the details of its tuning, it is sometimes difficult to know whether a given technique is genuinely better, or simply better tuned.

In this work, we propose a meta-modeling approach to support automated hyperparameter optimization, with the goal of providing practical tools that replace hand-tuning with a reproducible and unbiased optimization process. Our approach is to expose the underlying expression graph of how a performance metric (e.g. classification accuracy on validation examples) is computed from hyperparameters that govern not only how individual processing steps are applied, but even which processing steps are included. A hyperparameter optimization algorithm transforms this graph into a program for optimizing that performance metric. Our approach yields state of the art results on three disparate computer vision problems: a face-matching verification task (LFW), a face identification task (PubFig83) and an object recognition task (CIFAR-10), using a single broad class of feed-forward vision architectures.

Proceedings of the 30th International Conference on Machine Learning, Atlanta, Georgia, USA, 2013. JMLR: W&CP volume 28. Copyright 2013 by the author(s).

1. Introduction

Many computer vision algorithms depend on *hyperparameter* choices such as the size of filter bank, the strength of classifier regularization, and positions of quantization levels. These choices can have enormous impact on system performance: e.g. in (Pinto & Cox, 2011), the authors extensively explored a single richly-parameterized model family, yielding classification performance that ranged from chance to state-of-the-art performance, depending solely on hyperparameter choices. This and other recent work show that the question of "how good is this model on that dataset?" is ill-posed. Rather, it makes sense to speak of the quality of the best configuration that can typically be discovered by a particular search procedure in a given amount of time, for a task at hand. From this perspective, the tuning of hyperparameters is an important part of understanding algorithm performance, and should be a formal and quantified part of model evaluation.

On the other hand, *ad hoc* manual tuning by the algorithm inventor, while generally hard to reproduce or compare with fairly, can be efficient. A system's designer has expectations for how his or her system should work, and he or she can quickly diagnose unexpected deviations.

In this work we explore the possibility that manual optimization is no longer efficient enough to justify the lack of formalization that it entails. Recent developments in algorithm configuration raise the efficiency of automatic search, even in mathematically awkward search spaces, to a level where the result of hand-tuning can be matched and exceeded in a matter of hours on a small cluster of computers. Using these ideas, we implemented a broad class of feed-forward image feature extraction and classification models in order to formalize the steps of selecting the parameters of a model, and evaluating that model on a task. We compared random search in that model class with a more sophisticated algorithm for hyperparameter optimization, and found that the optimization-based search strategy recovered or improved on the best known configurations for all three image classification tasks in our study. This success motivates us to suggest that questions regarding the utility of modeling ideas should generally be tested in this style. Automatic search is reproducible, and thus supports analysis that is impossible for human researchers to perform fairly (e.g. "How would you have tuned approach Y if you had not already learned to optimize approach X?") To support research in hyperparameter optimization, we provide our optimization algorithms and specification language for download as free open source software. This software replicates the results presented in this work, and provides a foundation for general algorithm configuration in future work.

2. Previous Work

Our work extends two veins of research with little historical overlap: feed-forward model architectures for computer vision, and techniques for algorithm configuration.

Feed-forward models in computer vision. There is a long tradition of basing computer vision systems on models of biological vision (Fukushima, 1980; LeCun et al., 1989; Riesenhuber & Poggio, 1999; Lowe, 1999; Hinton et al., 2006; DiCarlo et al., 2012). Such efforts have arrived at a rough consensus model in which nonlinear image features are computed by a feed-forward neural network. Each layer of the network comprises a relatively standard set of transformations, including: (i) dimensionality expansion (e.g. by convolution with a filter bank), (ii) dynamic-range reduction (e.g. by thresholding), (iii) spatial smoothing (e.g. pooling or soft-max), (iv) local competition (e.g. divisive normalization), and (v) dimensionality reduction (e.g. sub-sampling or PCA). Feature extraction is usually followed by a simple classifier read-out trained on labeled data.

Beyond this high-level consensus, however, many details remain unresolved: which specific operations should be involved, in what order should they be applied, how many layers should be used, what kinds of classifier(s) should be used, and how (if at all) should the filter values be learned from statistics of input data. Many competing modeling approaches can roughly be thought of as having made different design choices within a larger unformalized space of feed-forward algorithm configurations.

Algorithm configuration. Algorithm configuration is a branch of optimization dealing with mathematically difficult search spaces, comprising both discrete and continuous variables as well as conditional variables that are only meaningful for some combinations of other variables. *Bayesian optimization* approaches have proved useful in these difficult domains (Mockus et al., 1978). A Bayesian optimization approach centers on a probability model for $P(\text{score}|\text{configuration})$ that is obtained by updating a prior from a history H of (configuration, score) pairs. This model can be queried more quickly than the original system in order to find promising candidates. Search efficiency comes from only evaluating these most promising candidates on the original system. Gaussian processes (Rasmussen & Williams, 2006) have often been used as the probability model, but other regression models such as decision trees have also proved successful (Hutter, 2009; Brochu, 2010; Bardenet & Kégl, 2010; Hutter et al., 2011; Bergstra et al., 2012). In these approaches, the criterion of Expected Improvement (EI) beyond a threshold μ is a popular heuristic for making proposals (Jones, 2001). In that approach, the optimization algorithm repeatedly suggests a configuration c that optimizes $EI(c) = \int_{y<\mu} yP(y|c, H)$ while the experimental history of (score, configuration) pairs, H, accumulates and changes the model. Recently Bergstra et al. (2011) suggested an approach to Bayesian optimization based on a model of $P(c|y)$ instead. Under some assumptions this approach can also be seen to optimize EI.

Hyperparameter optimization in computer vision is typically carried out by hand, by grid search, or by random search. We conjecture that Bayesian optimization is not typically used because it is relatively new technology, and because it requires a layer of abstraction between the researcher toying with settings at a command prompt and the system being optimized. We show that although algorithm configuration is a young discipline, it already provides useful techniques for formalizing the difficult task of simultaneous optimization of many hyperparameters. One of the contributions of our work is to show how useful Bayesian optimization can be, even when optimizing *hundreds* of hyper-parameters.

3. Automatic Hyperparameter Optimization

Our approach to hyperparameter optimization has four conceptual components:

1. Null distribution specification language. We propose an expression language for specifying the hyperparameters of a search space. This language describes the distributions that would be used for random, unoptimized search of the configuration space, and encodes the bounds and legal values for any other search procedure. A null prior distribution for a search problem is an expression G written in this specification language, from which sample configurations can be drawn.

For example:

$$\begin{aligned} G = \{ & a = \mathbf{normal}(0,1), \\ & b = \mathbf{choice}(0,\ \log(\mathbf{uniform}(2,10)),\ a)\} \end{aligned}$$

specifies a joint distribution in which a is distributed normally with mean 0 and variance 1, and b takes either value 0, or a, or a value drawn uniformly between 2 and 10. There are three hyperparameters at play here, shown in bold: the value of a, the value of the choice, and the value of the uniform.

More generally, the expressions that make up the null distribution specification can be arbitrarily nested, composed into sequences, passed as arguments to deterministic functions, and referenced internally, to form an directed acyclic expression graph (DAG).

2. Loss Function. The loss function is the criterion we desire to minimize. It maps legal configurations sampled from G to a real value. For example, the loss functions could extract features from a particular image dataset using configuration parameters specified by the random sample from G, and then report mis-classification accuracy for those features. Typically the loss function will be intractable analytically and slow enough to compute that doing so imposes a meaningful cost on the experimenter's time.

3. Hyperparameter Optimization algorithm (HOA). The HOA is an algorithm which takes as inputs the null prior expression G and an experimental history H of values of the loss function, and returns suggestions for which configuration to try next. Random sampling from the prior distribution specification G is a perfectly valid HOA. More sophisticated HOAs will generally commandeer the random nodes within the null prior expression graph, replacing them with expressions that use the experimental history in a nontrivial way (e.g. by replacing a **uniform** node with a Gaussian mixture whose number of components, means, and variances are refined over the course of the experiment).

4. Database. Our approach relies on a database to store the experimental history H of configurations that have been tried, and the value of the loss function at each one. As a search progresses, the database grows, and the HOA explores different areas of the search space.

The stochastic **choice** node, which randomly chooses an argument from a list of possibilities, is an important aspect of our approach. Choice nodes make it possible to encode *conditional parameters* in a search space (Hutter, 2009). To continue the example above, if the choice node is evaluated such that b takes the value of a, then our parameterization of G allows the optimizer to infer that whatever score we obtain has nothing to do with the hyperparameter associated with $\mathbf{uniform}(2,10)$. Visual system models have many configurable components, and entire components can be omitted from a particular pipeline configuration, so it is natural to describe the parameters of an optional component using conditional parameters. The use of conditional parameters makes credit assignment among a set of hyperparameters more efficient.

Our implementation of these four components is available for download as both a general purpose tool for program optimization and a specific visual system model for new image classification data sets (Bergstra, 2013; Bergstra et al., 2013).

4. Object Recognition Model Family

We evaluate the viability of automatic parameter search by encoding a broad class of feed-forward classification models in terms of the null distribution specification language described in the previous section. This space is a combination of the work of Coates & Ng (2011) and Pinto et al. (2009), and is known to contain parameter settings that achieve the state of the art performance on three data sets (i.e, loss functions): LFW, Pubfig83, and CIFAR-10.

The full model family that we explore is illustrated in Figure 1. Like Coates & Ng (2011), we include ZCA-based filter-generation algorithms (Hyvärinen & Oja, 2000) and coarse histogram features (described in their work as the R-T and RP-T algorithms). Like Pinto et al. (2009), we allow for 2-layer and 3-layer sequences of filtering and non-linear spatial pooling. Our search space is configured by a total of 238 hyperparameters – far too large for brute force search, and an order of magnitude larger *in dimensionality* than the 32-dimensional space searched by Bergstra et al. (2011). The remainder of this section describes the components of our model family. An implementation of the model is available for download (Bergstra et al., 2013).

The *inter-layers* (Figure 1a) perform a filter bank normalized cross-correlation, spatial pooling, and possibly sub-sampling. These layers are very much in the spirit of the elements of the Pinto & Cox (2011) model, except that we have combined the normalization and filter bank cross-correlation into a single mathematical operation (fbncc, Equations 1-2).

$$y = \mathrm{fbncc}(x, f) \tag{1}$$

$$y_{ijk} = \frac{\check{f}_k * \check{u}_{ij}}{\sqrt{\rho \max(||\check{u}_{ij}||^2, \beta) + (1-\rho)(||\check{u}_{ij}||^2 + \beta)}} \tag{2}$$

The fbncc operation is a filter bank convolution of each filter f_k with a multi-channel image or feature map x, in which each patch $\check{x}_{ij}$ of x is first shifted by its mean $\epsilon\check{m}$ (motivating $\check{u}_{ij} \doteq \check{x}_{ij} - \epsilon\check{m}$) then scaled to have approximately unit norm. Whereas Pinto & Cox (2011) employed only random uniform filters f_k, we include also some of the filter-generation strategies employed in Coates & Ng (2011): namely random projections of ZCA components, and randomly chosen ZCA-filtered image patches. Filter-generation is parametrized by a filter count $K \in [16, 256]$), a filter size $S_f \in [2, 10]$, a random seed, and a band-pass parameter in the case of ZCA. The pair-indexed hat-notation $\check{x}_{ij}$ refers to a *patch volume* from x at row i and column j that includes S_f rows and columns as well as all channels of x; Our fbncc implementation is controlled by log-normally distributed hyperparameter β which defines a low-variance cutoff, a binary-valued hyperparameter ρ that determines whether that cutoff is soft or hard, and a binary-valued parameter ϵ that determines whether the empirically-defined patch mean $\check{m}$ should be subtracted off or not.

Local spatial pooling (lpool, Equation 3) was implemented as in Pinto & Cox (2011).

$$y = \mathrm{lpool}(x) \quad \Leftrightarrow \quad y_{ijk} = x_{i'j'k}/||\check{x}_{i'j'k}||_p \tag{3}$$

The operation is parameterized by a patch size $S_p \in [2, 8]$, a sub-sampling stride $i'/i = j'/j \in \{1, 2\}$, and a log-normally distributed norm parameter p. The triple-indexed $\check{x}_{ijk}$ refers to a single-channel *patch surface* from x at row i, column j, and channel k that extends spatially to include S_p rows and columns.

The *outer-layers* (Figure 1b) combine the fbncc operation of inter-layers with different pooling options. Rather than sampling or optimizing the filter count, the filter count is determined analytically so that the number of image features approaches but does not exceed sixteen thousand (16,000). Pooling is done either (1) with lpool and lnorm (Equation 4) as in Pinto & Cox (2011), or (2) with spatial summation of positive and negative half-rectified filter bank responses (dihist, Equation 5). Within pooling strategy (2) we used two strategies to define the spatial patches used in the summation: either (2a) grid cell summation as in Coates & Ng (2011), or (2b) box filtering. The difference between (2a) and (2b) is a trade-off between spatial resolution and depth of filter bank in making up the output feature set.

$$y = \mathrm{lnorm}(x) \quad \Leftrightarrow \quad y_{ij} = \begin{cases} \frac{x_{ijk}}{\check{x}_{ij}} & \text{if } ||\check{x}_{ij}||_2 > \tau \\ x_{ijk} & \text{otherwise} \end{cases} \tag{4}$$

$$y = \mathrm{dihist}(x) \quad \Leftrightarrow \quad y_{ijk} = \begin{bmatrix} ||\max(\check{x}_{ijk} - \alpha, 0)||_1 \\ ||\max(-\check{x}_{ijk} - \alpha, 0)||_1 \end{bmatrix} \tag{5}$$

Hyperparameter τ of the lnorm operation was log-normally distributed, as was the α hyperparameter of dihist. In approach (2a) we allowed 2x2 or 3x3 grids. In approach (2b) we allowed for sub-sampling by 1, 2, or 3 and square summation regions of side-length 2 to 8.

The last step in our image-processing pipeline is a *classifier*, for which we used an ℓ_2-regularized, linear, L2-SVM. For the smaller training sets we used liblinear via sklearn as the solver(Fan et al., 2008; Pedregosa et al., 2011), for larger ones we used a generic L-BFGS algorithm in the primal domain (Bergstra et al., 2010). Training data were column-normalized. The classifier components had just two hyperparameters: the

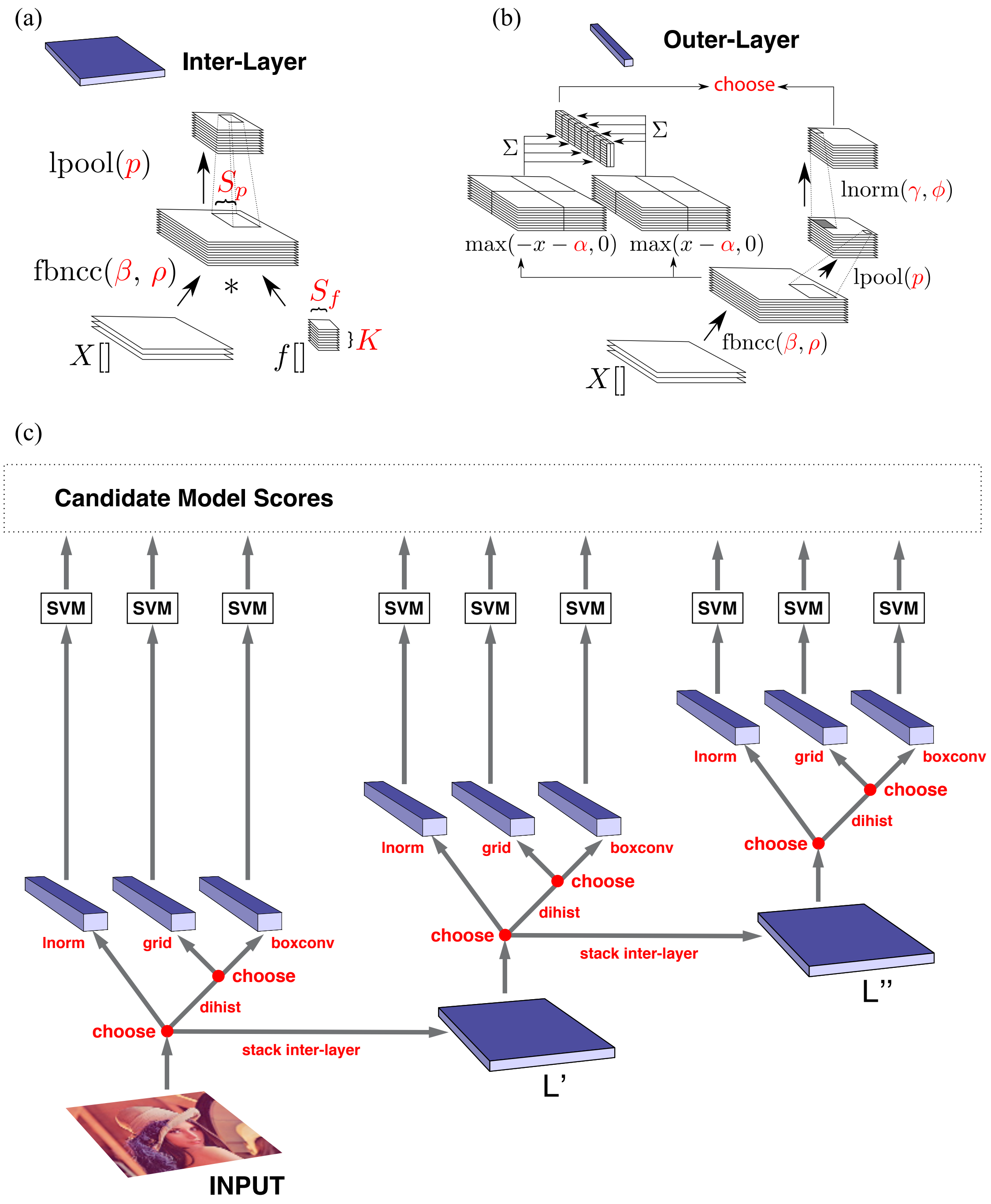


Figure 1. Our experiments search a class of image classification pipelines (c) that include 0, 1, or 2 *inter-layers* (a), an *outer-layer* (b) that extracts features, and a support vector machine (SVM) classifier. Inter-layers perform filter-bank normalized cross-correlation (fbncc) and local spatial pooling (lpool). Outer-layers are similar, but may additionally perform quadrant pooling and local normalization (lnorm). Hyperparameters govern the number of inter-layers, the type of outer-layer, and a host of configuration options within each layer. Although many of the hyperparameters are mutually exclusive (e.g. only one outer-layer is active per pipeline) there are 238 hyperparameters in the full search space.

strength of regularization and a cutoff for low-variance feature columns. Code for these experiments was written in Python, with the feature extraction carried out by Theano (Bergstra et al., 2010) and hyperparameter optimization carried out by the hyperopt package (Bergstra, 2013).

5. Results

We evaluate the technique of automatic hyperparameter configuration by comparing two hyperparameter optimization algorithms: random search versus a Tree of Parzen Estimators (TPE) (Bergstra et al., 2011). The TPE algorithm is an HOA that acts by replacing stochastic nodes in the null description language with ratios of Gaussian Mixture Models (GMM). On each iteration, for each hyperparameter, TPE fits one GMM $\ell(x)$ to the set of hyperparameter values associated with the smallest (best) loss function values, and another GMM $g(x)$ to the remaining hyperparameter values. It chooses the hyperparameter value x that maximizes the ratio $\ell(x)/g(x)$. Relative to Bergstra et al. (2011) we made two minor modifications to the search algorithm. The first modification was to downweight trials as they age so that old results do not count for as much as more recent ones. We gave full weight to the most recent 25 trials and applied a linear ramp from 0 to 1.0 to older trials. This is a heuristic concession to TPE's assumption of hyperparameter independence: as our search moves through space, we use temporal distance within the experiment as a surrogate for distance in the search space. The second modification was to vary the fraction of trials used to estimate $\ell(x)$ and $g(x)$ with time. Out of T observations of any given variable, we used the top-performing $\sqrt{T}/4$ trials to estimate the density of ℓ. We initialized TPE with 50 trials drawn from the null configuration description. These hyper-hyperparameters were chosen manually by observing the shape of optimization trajectories on LFW view 1. We did not reconfigure TPE for the other data sets. The TPE algorithm took up to one or two seconds to suggest new points, so it contributed a negligible computational cost to these experiments.

5.1. TPE vs. Random Search: LFW and PubFig83

Random search in a large space of biologically-inspired models has been shown to be an effective approach to face verification (Pinto & Cox, 2011) and identification (Pinto et al., 2011). Our search space is similar to the one used in those works, so LFW (Huang et al., 2007) and PubFig83 (Pinto et al., 2011) provide fair playing fields for comparing TPE with random search.

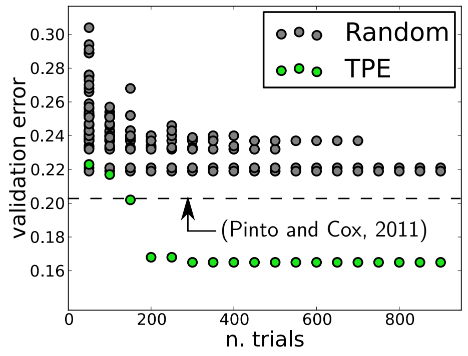
For experiments on LFW, we follow Pinto & Cox (2011) in using the *aligned* image set, and resizing the gray scale images to 200×200. We followed the official evaluation protocol – performing model selection on the basis of one thousand images from "view 1" and testing by re-training the classifier on 10 "view 2" splits of six thousand pairs. We transformed image features into features of image pairs by applying an element-by-element *comparison* function to the left-image and right-image feature vectors. Following Pinto & Cox (2011) we used one comparison function for model selection (square root of absolute difference) and we concatenated four comparison functions for the final "view 2" model evaluation (product, absolute difference, squared difference, square root of absolute difference).

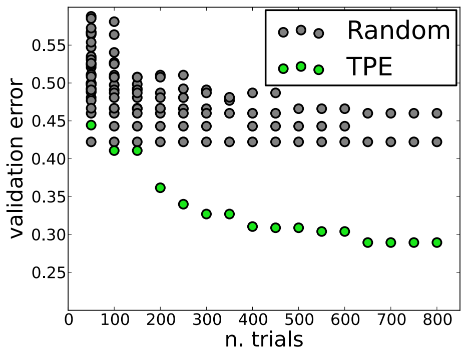
The PubFig83 data set contains 8300 color images of size 100×100, with 100 pictures of each of 83 celebrities (Pinto et al., 2011). For our PubFig83 experiments we converted the *un-aligned* images to gray scale and screened models on the 83-way identification task using 3 splits of 20 train/20 validation examples per class, running two simultaneous TPE optimization processes for a total of 1200 model evaluations. Top-scoring configurations on the screening task were then tested in a second phase, consisting of five training splits of 90 train/10 test images per class. Each of the five second phase training sets of 90 images per class consisted of the 40 images from the first phase and 50 of the 60 remaining images.

The results of our model search on LFW are shown in Figure 2. The TPE algorithm exceeded the best random search view 1 performance within 200 trials, for both our random search and that carried out in Pinto & Cox (2011). TPE converged within 1000 trials to an error rate of 16.2%, significantly lower than the best configuration found by random search (21.9%). On LFW's test data (view 2) the optimal TPE configuration also beats those found by our random search (84.5% vs. 79.2% accurate). The best configuration found by random search in Pinto & Cox (2011) does well on View 2 relative to View 1 (84.1% vs. approximately 79.5%) and is approximately as accurate as TPE's best configuration on the test set. On PubFig83, the optimal TPE configuration outperforms the best random configuration found by our random search (86.5% vs 81.0% accurate) and the previous state of the art result (85.2%) Pinto et al. (2011).

5.2. Matching Hand-Tuning: CIFAR-10

Coates & Ng (2011) showed that single-layer ap-





Method (# configurations)	LFW View 2 Error (%)	PubFig83 View 2 Error (%)
TPE-optimized (750)	**15.5** ± .7	**13.50** ± .7
High-throughput (15K)	**15.9** ± .7 (Pinto & Cox, 2011)	14.78 ± .45 (Pinto et al., 2011)
Random search (2K)	20.8 ± .8	19.0 ± .8
Chance	50.0	98.8

Figure 2. Optimization of validation set performance on data sets LFW (top left) and PubFig83 (top right). The TPE algorithm finds configurations with significantly better validation set error than a 2000-trial random search or the 15,000-trial random searches carried out in Pinto & Cox (2011) and Pinto et al. (2011). Grey dots in the top panels within a column represent the lowest error among T random trials (as T increases to the right); green dots denote the lowest error observed within the first T suggestions by the TPE algorithm. On test data ("view 2", bottom), TPE has discovered the best known model configuration in the search space within 750 trials, but our 2000-trial random search has not come close. View 2 error rates are given with a 95% confidence interval assuming Bernoulli-distributed errors.

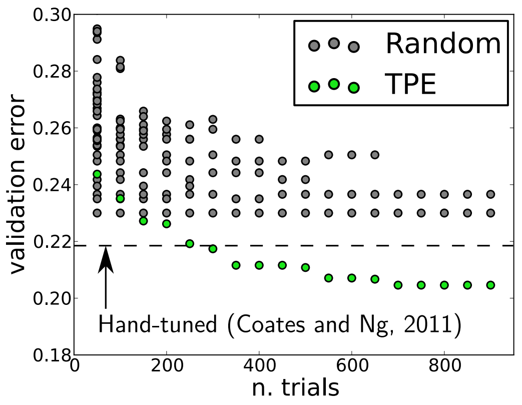
proaches are competitive with the best multi-layer alternatives for 10-way object classification using the CIFAR-10 data set (Krizhevsky, 2009). The success of their single-layer approaches depends critically on correct settings for several hyperparameters governing pre-processing and feature extraction. CIFAR-10 images are low-resolution color images (32×32) but there are fifty thousand labeled images for training and ten thousand for testing. We performed model selection on the basis of a single random, stratified subset of ten thousand training examples.

The results of TPE and random search are reported in Figure 3. TPE, starting from broad priors over a wide variety of processing pipelines, was able to match the performance of a skilled and motivated domain expert. With regards to the wall time of the automatic approach, our implementation of the pipeline was designed for GPU execution and the loss function required from 0 to 30 minutes. TPE found a configuration very similar to the one found by in Coates & Ng (2011) within roughly 24 hours of processing on 6 GPUs. Random search was not able to approach the same level of performance.

6. Discussion

In this work, we have described a conceptual framework to support automated hyperparameter optimization, and demonstrated that it can be used to quickly recover state-of-the-art results on several unrelated image classification tasks from a large family of computer vision models, with no manual intervention. On each of three datasets used in our study we compared random search to a more sophisticated alternative: TPE. A priori, random search confers some advantages: it is trivially parallel, it is simpler to implement, and the independence of trials supports more interesting analysis (Bergstra & Bengio, 2012). However, our experiments found that TPE clearly outstrips random search in terms of optimization efficiency. TPE found best known configurations for each data set, and did so in only a small fraction of the time we allocated to random search. TPE, but not random search, was found to match the performance of manual tuning on the CIFAR-10 data set. With regards to the computational overhead of search, TPE took no more than a second or two to suggest new hyperparameter assignments, so it added a negligible computational cost to the experiments overall.

This work opens many avenues for future work. One direction is to enlarge the model class to include a greater variety of components, and configuration strategies for those components. Many filter-learning and feature extraction techniques have been proposed in the literature beyond the core implemented in our experiment code base. Another direction is to improve



Method (# configs)	Test Acc. (%)
Hand-tuned	**20.9** ± .8
TPE (800)	**21.2** ± .8
Random (2K)	23.4 ± .8
Chance	90.0

Figure 3. On the CIFAR-10 object classification data set, TPE minimizes validation set error (left) better than manual tuning and a 2000-point random search. On test data (right) the best model found by TPE matches the performance of hand-tuning within the model class. Test errors are given with a 95% confidence interval assuming Bernoulli-distributed errors. The best configurations from the TPE and random searches are both better on validation than test; this overfitting is expected when the validation set is not perfectly representative of the test set.

the search algorithms. The TPE algorithm is conspicuously deficient in optimizing each hyperparameter independently of the others. It is almost certainly the case that the optimal values of some hyperparameters depend on settings of others. Algorithms such as SMAC (Hutter et al., 2011) that can represent such interactions might be significantly more effective optimizers than TPE. It might be possible to extend TPE to profitably employ non-factorial joint densities $P(\text{config}|\text{score})$. Relatedly, such optimization algorithms might permit the model description language to include constraints in the form of distributional parameters that are themselves optimizable quantities (e.g. $\textbf{uniform}(0, \textbf{lognormal}(0, 1))$). Another important direction for research in algorithm configuration is a recognition that not all loss function evaluations are equally expensive in terms of various limited resources, most notably in terms of computation time. All else being equal, configurations that are cheaper to evaluate should be favored (Snoek et al., 2012).

Our experiments dealt with the optimization of classification accuracy, but our approach extends quite naturally to the optimization (and constrained optimization via barrier techniques) of any real-valued criterion. We could search instead for the smallest or fastest model that meets a certain level of classification performance, or the best-performing model that meets the resource constraints imposed by a particular mobile platform. Having to perform such searches by hand may be daunting, but when the search space is encoded as a searchable model class, automatic optimization methods can be brought to bear.

7. Acknowledgements

This work was funded by the Rowland Institute of Harvard, and the National Science Foundation (IIS 0963668).

References

Bardenet, R. and Kégl, B. Surrogating the surrogate: accelerating Gaussian Process optimization with mixtures. In *ICML*, 2010.

Bergstra, J. Hyperopt: Distributed asynchronous hyperparameter optimization in Python. http://jaberg.github.com/hyperopt, 2013.

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

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., and Bengio, Y. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In *NIPS*24*, pp. 2546–2554, 2011.

Bergstra, J., Pinto, N., and Cox, D. D. Machine learning for predictive auto-tuning with boosted regression trees. In *INPAR*, 2012.

Bergstra, J., Yamins, D., and Pinto, N. Hyperparameter optimization for convolutional vision architectures. https://github.com/jaberg/hyperopt-convnet, 2013.

Brochu, E. *Interactive Bayesian Optimization: Learning Parameters for Graphics and Animation.* PhD thesis, University of British Columbia, December 2010.

Coates, A. and Ng, A. Y. The importance of encoding versus training with sparse coding and vector quantization. In *Proc. ICML-28*, 2011.

DiCarlo, J. J., Zoccolan, D., and Rust, N. C. How does the brain solve visual object recognition? *Neuron*, 73:415–34, 2012 Feb 9 2012. ISSN 1097-4199.

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

Fukushima, K. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, 1980.

Hinton, G. E., Osindero, S., and Teh, Y. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Huang, G. B., Ramesh, M., Berg, T., and Learned-Miller, E. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

Hutter, F. *Automated Configuration of Algorithms for Solving Hard Computational Problems.* PhD thesis, University of British Columbia, 2009.

Hutter, F., Hoos, H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In *LION-5*, 2011. Extended version as UBC Tech report TR-2010-10.

Hyvärinen, A. and Oja, E. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4–5):411–430, 2000.

Jones, D.R. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

Lowe, D. G. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision 2 (ICCV)*, pp. 1150–1157, 1999. doi: 10.1109/ICCV.1999.790410.

Mockus, J., Tiesis, V., and Zilinskas, A. The application of Bayesian methods for seeking the extremum. In Dixon, L.C.W. and Szego, G.P. (eds.), *Towards Global Optimization*, volume 2, pp. 117–129. North Holland, New York, 1978.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Pinto, N. and Cox, D. D. Beyond simple features: A large-scale feature search approach to unconstrained face recognition. In *Proc. Face and Gesture Recognition*, 2011.

Pinto, N., Doukhan, D., DiCarlo, J. J., and Cox, D. D. A high-throughput screening approach to discovering good forms of biologically inspired visual representation. *PLoS Comput Biol*, 5(11):e1000579, 11 2009.

Pinto, N., Stone, Z., Zickler, T., and Cox, D. D. Scaling-up Biologically-Inspired Computer Vision: A Case-Study on Facebook. In *IEEE Computer Vision and Pattern Recognition, Workshop on Biologically Consistent Vision*, 2011.

Rasmussen, C. E. and Williams, C. K. I. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

Riesenhuber, M. and Poggio, T. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

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