# HYPERBAND: BANDIT-BASED CONFIGURATION EVALUATION FOR HYPERPARAMETER OPTIMIZATION

**Lisha Li**[*], **Kevin Jamieson**[**], **Giulia DeSalvo**[†], **Afshin Rostamizadeh**[‡], **and Ameet Talwalkar**[*]
[*]UCLA, [**]UC Berkeley, [†]NYU, and [‡]Google
{lishal,ameet}@cs.ucla.edu, kjamieson@berkeley.edu
desalvo@cims.nyu.edu, rostami@google.com

## ABSTRACT

Performance of machine learning algorithms depends critically on identifying a good set of hyperparameters. While recent approaches use Bayesian Optimization to adaptively select configurations, we focus on speeding up random search through adaptive resource allocation. We present HYPERBAND, a novel algorithm for hyperparameter optimization that is simple, flexible, and theoretically sound. HYPERBAND is a principled early-stoppping method that adaptively allocates a predefined resource, e.g., iterations, data samples or number of features, to randomly sampled configurations. We compare HYPERBAND with popular Bayesian Optimization methods on several hyperparameter optimization problems. We observe that HYPERBAND can provide more than an order of magnitude speedups over competitors on a variety of neural network and kernel-based learning problems.

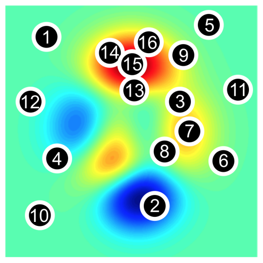
## 1 INTRODUCTION

The task of hyperparameter optimization is becoming increasingly important as modern data analysis pipelines grow in complexity. The quality of a predictive model critically depends on its hyperparameter configuration, but it is poorly understood how these hyperparameters interact with each other to affect the quality of the resulting model. Consequently, practitioners often default to either hand-tuning or automated brute-force methods like random search and grid search.

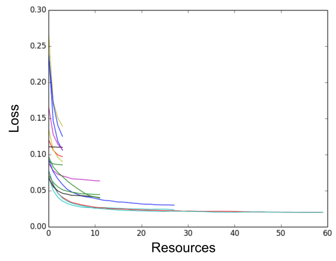
In an effort to develop more efficient search methods, the problem of hyperparameter optimization has recently been dominated by *Bayesian optimization* methods (Snoek et al., 2012; Hutter et al., 2011; Bergstra et al., 2011) that focus on optimizing hyperparameter *configuration selection*. These methods aim to identify good configurations more quickly than standard baselines like random search by selecting configurations in an adaptive manner; see Figure 1(a). Existing empirical evidence suggests that these methods outperform random search (Thornton et al., 2013; Eggensperger et al., 2013; Snoek et al., 2015). However, these methods tackle a fundamentally challenging problem of simultaneously fitting and optimizing a high-dimensional, non-convex function with unknown smoothness, and possibly noisy evaluations. To overcome these difficulties, some Bayesian optimization methods resort to heuristics, at the expense of consistency guarantees, to model the objective function or speed up resource intensive subroutines.[1] Moreover, these adaptive configuration selection methods are intrinsically sequential and thus difficult to parallelize.

An orthogonal approach to hyperparameter optimization focuses on speeding up *configuration evaluation*; see Figure 1(b). These methods are adaptive in computation, allocating more resources to promising hyperparameter configurations while quickly eliminating poor ones. Resources can take various forms, including size of training set, number of features, or number of iterations for iterative algorithms. By adaptively allocating these resources, these methods aim to examine orders of magnitude more hyperparameter configurations than methods that uniformly train all configurations to completion, thereby quickly identifying good hyperparameters. While there are methods that combine Bayesian optimization with adaptive resource allocation (Swersky et al., 2013; 2014; Domhan et al., 2015), we focus on speeding up random search as it offers a simple, parallelizable, and theoretically principled launching point and is shown to outperform grid search (Bergstra & Bengio, 2012).

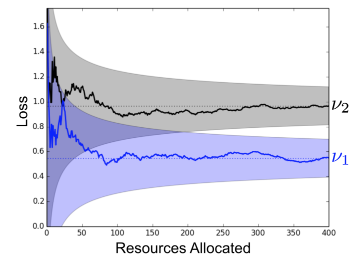
[1]Consistency can be restored by allocating a fraction of resources to performing random search.



(a) Configuration Selection



(b) Configuration Evaluation



(c) Envelopes

Figure 1: (a) The heatmap shows the validation error over a two dimensional search space, with red corresponding to areas with lower validation error, and putative configurations selected in a sequential manner as indicated by the numbers. (b) The plot shows the validation error as a function of the resources allocated to each configuration (i.e., each line in the plot). Configuration evaluation methods allocate more resources to promising configurations. (c) The validation loss as a function of total resources allocated for two configurations. The shaded areas bound the maximum distance from the terminal validation loss and monotonically decreases with the resource.

Our novel configuration evaluation method, HYPERBAND, relies on a principled early-stopping strategy to allocate resources, allowing it to evaluate orders of magnitude more configurations than uniform allocation strategies. HYPERBAND is a general-purpose technique that makes minimal assumptions, unlike prior configuration evaluation approaches (Swersky et al., 2013; Domhan et al., 2015; Swersky et al., 2014; György & Kocsis, 2011; Agarwal et al., 2011). In this work, we describe HYPERBAND, provide intuition for the algorithm through a detailed example, and present a wide range of empirical results comparing HYPERBAND with well established competitors. We also briefly describe the theoretical underpinnings of HYPERBAND, however a thorough theoretical treatment is beyond the scope of this paper and is deferred to Li et al. (2016).

## 2 Related Work

Bayesian optimization techniques model the conditional probability $p(f|\lambda)$ of a configuration's performance on a metric $f$ given a set of hyperparameters $\lambda$. For instance, SMAC uses random forests to model $p(f|\lambda)$ as a Gaussian distribution (Hutter et al., 2011). TPE is a non-standard Bayesian optimization algorithm based on tree-structured Parzen density estimators (Bergstra et al., 2011). A third popular method, Spearmint, uses Gaussian processes (GP) to model $p(f|\lambda)$ and performs slice sampling over the GP's hyperparameters (Snoek et al., 2012).

Adaptive configuration evaluation is not a new idea. Maron & Moore (1993) considered a setting where training time is negligible (e.g., $k$-nearest-neighbor classification) and evaluation on a large validation set is accelerated by evaluating on an increasing subset of the validation set, stopping early configurations that are performing poorly. Since subsets of the validation set provide unbiased estimates of its expected performance, this is an instance of the *stochastic* best-arm identification problem for multi-armed bandits (see Jamieson & Nowak (2014) for a brief survey).

In contrast, this paper assumes that evaluation time is negligible and the goal is to early-stop long-running training procedures by evaluating partially trained models on the validation set. Previous approaches either require strong assumptions or use heuristics to perform adaptive resource allocation. Several works propose methods that make strong assumptions on the convergence behavior of training algorithms, providing theoretical performance guarantees under these assumptions (György & Kocsis, 2011; Agarwal et al., 2011; Swersky et al., 2013; 2014; Domhan et al., 2015; Sabharwal et al., 2016). Unfortunately, these assumptions are often hard to verify, and empirical performance can drastically suffer when they are violated. One recent work of particular interest proposes a heuristic based on sequential analysis to determine stopping times for training configurations on increasing subsets of the data (Krueger et al., 2015). However, it has a few shortcomings: (1) it is designed to speedup multi-fold cross-validation and is not significantly faster than standard holdout, (2) it is not an anytime algorithm and requires the set of configurations to be evaluated as an input, and (3) the theoretical correctness and empirical performance of this method are highly dependent on a user-defined "safety-zone."[2] Lastly, in an effort avoid heuristics and strong assumptions, Sparks et al. (2015) proposed a halving style algorithm that did not require explicit convergence behavior, and Jamieson & Talwalkar (2015) analyzed a similar algorithm, providing theoretical guarantees and encouraging empirical results. Unfortunately, these halving style algorithms suffer from the $n$ vs $B/n$ issue which we will discuss in Section 3.

Finally, Klein et al. (2016) recently introduced Fabolas, a Bayesian optimization method that combines adaptive selection and evaluation. Similar to Swersky et al. (2013; 2014), it models the conditional validation error as a Gaussian process using a kernel that captures the covariance with downsampling rate to allow for adaptive evaluation. While we intended to compare HYPERBAND with Fabolas, we encountered some technical difficulties when using the package[3] and are working with the authors of Klein et al. (2016) to resolve the issues.

## 3 HYPERBAND ALGORITHM

HYPERBAND extends the SUCCESSIVEHALVING algorithm proposed for hyperparameter optimization in Jamieson & Talwalkar (2015) and calls it as a subroutine. The idea behind SUCCESSIVEHALVING follows directly from its name: uniformly allocate a budget to a set of hyperparameter configurations, evaluate the performance of all configurations, throw out the worst half, and repeat until one configurations remains. The algorithm allocates exponentially more resources to more promising configurations. Unfortunately, SUCCESSIVEHALVING requires the number of configurations $n$ as an input to the algorithm. Given some finite time budget $B$ (e.g. an hour of training time to choose a hyperparameter configuration), $B/n$ resources are allocated on average across the configurations. However, for a fixed $B$, it is not clear a priori whether we should (a) consider many configurations (large $n$) with a small average training time; or (b) consider a small number of configurations (small $n$) with longer average training times.

We use a simple example to better understand this tradeoff. Figure 1(c) shows the validation loss as a function of total resources allocated for two configurations with terminal validation losses $\nu_1$ and $\nu_2$. The shaded areas bound the maximum deviation from the terminal validation loss and will be referred to as "envelope" functions. It is possible to differentiate between the two configurations when the envelopes diverge. Simple arithmetic shows that this happens when the width of the envelopes is less than $\nu_2 - \nu_1$, i.e. when the intermediate losses are guaranteed to be less than $\frac{\nu_2-\nu_1}{2}$ away from the terminal losses. There are two takeaways from this observation: more resources are needed to differentiate between the two configurations when either (1) the envelope functions are wider or (2) the terminal losses are closer together.

However, in practice, the optimal allocation strategy is unknown because we do not have knowledge of the envelope functions nor the distribution of terminal losses. Hence, if more resources are required before configurations can differentiate themselves in terms of quality (e.g., if an iterative training method converges very slowly for a given dataset or if randomly selected hyperparameter configurations perform similarly well) then it would be reasonable to work with a small number of configurations. In contrast, if the quality of a configuration is typically revealed using minimal resources (e.g., if iterative training methods converge very quickly for a given dataset or if randomly selected hyperparameter configurations are of low-quality with high probability) then $n$ is the bottleneck and we should choose $n$ to be large.

### 3.1 HYPERBAND

HYPERBAND, shown in Algorithm 1, addresses this "$n$ versus $B/n$" problem by considering several possible values of $n$ for a fixed $B$, in essence performing a grid search over feasible value of $n$. Associated with each value of $n$ is a minimum resource $r$ that is allocated to all configurations before some are discarded; a larger value of $n$ corresponds to a smaller $r$ and hence more aggressive early stopping. There are two components to HYPERBAND; (1) the inner loop invokes SUCCESSIVEHALVING for fixed values of $n$ and $r$ (lines 3-9) and (2) the outer loop which iterates over different values

[2]The first two drawbacks prevent a full comparison to HYPERBAND on our selected empirical tasks, however, for completeness, we provide a comparison in Appendix A to Krueger et al. (2015) on some experimental tasks replicated from their paper.

[3]The package provided by Klein et al. (2016) is available at `https://github.com/automl/RoBO`.

of $n$ and $r$ (lines 1-2). We will refer to each such run of SUCCESSIVEHALVING within HYPERBAND as a "bracket." Each bracket is designed to use about $B$ total resources and corresponds to a different tradeoff between $n$ and $B/n$. A single execution of HYPERBAND takes a finite number of iterations, and in practice can be repeated indefinitely.

HYPERBAND requires two inputs (1) $R$, the maximum amount of resource that can be allocated to a single configuration, and (2) $\eta$, an input that controls the proportion of configurations discarded in each round of SUCCESSIVEHALVING. The two inputs dictate how many different brackets are considered; specifically, $s_{\max}+1$ different values for $n$ are considered with $s_{\max} = \lfloor \log_\eta(R) \rfloor$. HYPERBAND begins with the most aggressive bracket $s = s_{\max}$, which sets $n$ to maximize exploration, subject to the constraint that at least one configuration is allocated $R$ resources. Each subsequent bracket reduces $n$ by a factor of approximately $\eta$ until the final bracket, $s = 0$, in which every configuration is allocated $R$ resources (this bracket simply performs classical random search). Hence, HYPERBAND performs a geometric search in the average budget per configuration to address the "$n$ versus $B/n$" problem, at the cost of approximately $s_{\max}+1$ times more work than running SUCCESSIVEHALVING for a fixed $n$. By doing so, HYPERBAND is able to exploit situations in which adaptive allocation works well, while protecting itself in situations where more conservative allocations are required.

**Algorithm 1:** HYPERBAND algorithm for hyperparameter optimization.

```
   input          : $R$, $\eta$ (default $\eta = 3$)
   initialization : $s_{\max} = \lfloor \log_\eta(R) \rfloor$, $B = (s_{\max}+1)R$
1  for $s \in \{s_{\max}, s_{\max}-1, \ldots, 0\}$ do
2      $n = \lceil \frac{B}{R}\frac{\eta^s}{(s+1)} \rceil$,        $r = R\eta^{-s}$
       // begin SUCCESSIVEHALVING with $(n,r)$ inner loop
3      $T$ =get_hyperparameter_configuration($n$)
4      for $i \in \{0, \ldots, s\}$ do
5          $n_i = \lfloor n\eta^{-i} \rfloor$
6          $r_i = r\eta^i$
7          $L = \{$run_then_return_val_loss$(t, r_i) : t \in T\}$
8          $T$ =top_k$(T, L, \lfloor n_i/\eta \rfloor)$
9      end
10 end
11 return Configuration with the smallest intermediate loss seen so far.
```

$R$ represents the maximum amount of resources that can be allocated to any given configuration. In most cases, there is a natural upper bound on the maximum budget per configuration that is often dictated by the resource type (e.g., training set size for dataset downsampling; limitations based on memory constraint for feature downsampling; rule of thumb regarding number of epochs when iteratively training neural networks). $R$ is also the number of configurations evaluated in the bracket that performs the most exploration, i.e $s = s_{\max}$. In practice one may want $n \leq n_{\max}$ to limit overhead associated with training many configurations on a small budget, i.e. costs associated with initialization, loading a model, and validation. In this case, set $s_{\max} = \lfloor \log_\eta(n_{\max}) \rfloor$.

The value of $\eta$ can be viewed as a knob that can be tuned based on *practical* user constraints. Larger values of $\eta$ correspond to a more aggressive elimination schedule and thus fewer rounds of elimination; specifically, each round retains $1/\eta$ configurations for a total of $\lfloor \log_\eta(n) \rfloor + 1$ rounds of elimination with $n$ configurations. If one wishes to receive a result faster at the cost of a sub-optimal asymptotic constant, one can increase $\eta$ to reduce the budget per bracket $B = (\lfloor \log_\eta(R) \rfloor + 1)R$. We stress that results are not very sensitive to the choice of $\eta$. In practice we suggest taking $\eta$ to be equal to 3 or 4.

HYPERBAND requires the following methods to be defined for any given learning problem: `get_hyperparameter_configuration(n)` returns a set of $n$ i.i.d. samples from some distribution defined over the hyperparameter configuration space; `run_then_return_val_loss(t, r)` takes a hyperparameter configuration ($t$) and resource allocation ($r$) and returns the validation loss after training for the allocated resources; and `top_k(configs, losses, k)` takes a set of configurations as well as their associated losses and returns the top $k$ performing configurations.

## 3.2 Example application with iterations as a resource: LeNet

We next present a simple example to provide intuition. We work with the MNIST dataset and optimize hyperparameters for the LeNet convolutional neural network trained using mini-batch SGD. Our search space includes learning rate, batch size, and number of kernels for the two layers of the network as hyperparameters (details are shown in Table 3 in Appendix A).

We further define the number of iterations as the resource to allocate, with one unit of resource corresponding to one epoch or a full pass over the dataset. We set $R$ to 81 and use the default value of $\eta = 3$, resulting in $s_{\max} = 4$ and thus 5 brackets of SuccessiveHalving with different tradeoffs between $n$ and $B/n$. The resources allocated within each bracket are displayed in Table 1.

| | $s = 4$ | | $s = 3$ | | $s = 2$ | | $s = 1$ | | $s = 0$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| $i$ | $n_i$ | $r_i$ | $n_i$ | $r_i$ | $n_i$ | $r_i$ | $n_i$ | $r_i$ | $n_i$ | $r_i$ |
| 0 | 81 | 1 | 27 | 3 | 9 | 9 | 6 | 27 | 5 | 81 |
| 1 | 27 | 3 | 9 | 9 | 3 | 27 | 2 | 81 | | |
| 2 | 9 | 9 | 3 | 27 | 1 | 81 | | | | |
| 3 | 3 | 27 | 1 | 81 | | | | | | |
| 4 | 1 | 81 | | | | | | | | |

Table 1: Values of $n_i$ and $r_i$ for the brackets of Hyperband when $R = 81$ and $\eta = 3$.

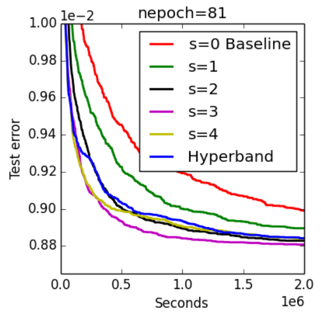


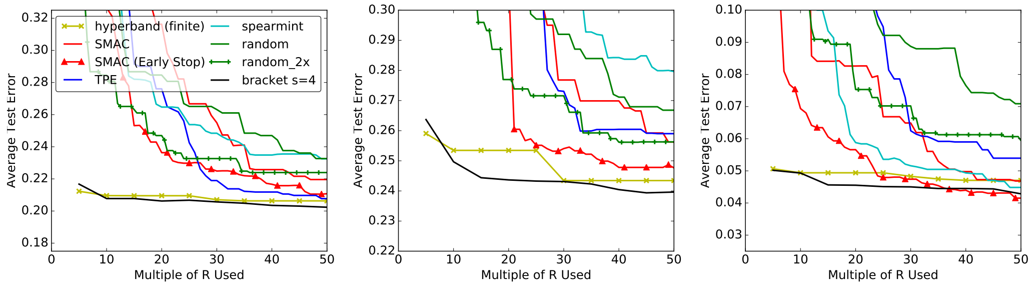
Figure 2: Performance of individual brackets $s$ and Hyperband.

Figure 2 compares the empirical performance of the different brackets of Hyperband if they were used separately, as well as standard Hyperband (all results are averaged over 70 trials). In practice we do not know a priori which bracket $s \in \{0, \ldots, 4\}$ will be most effective, and in this case neither the most ($s = 4$) nor least aggressive ($s = 0$) setting is optimal. However, note that Hyperband does nearly as well as the optimal bracket ($s = 3$) and vastly outperforms the baseline uniform allocation (i.e. random search), which is equivalent to bracket $s = 0$.

## 3.3 Overview of theoretical results

Although a detailed theoretical analysis is beyond the scope of this paper, we provide an intuitive, high-level description of theoretical properties of Hyperband. Suppose there are $n$ configurations, each with a given terminal validation error $\nu_i$ for $i = 1, \ldots, n$. Without loss of generality, index the configurations by performance so that $\nu_1$ corresponds to the best performing configuration, $\nu_2$ to the second best, and so on. Now consider the task of identifying the best configuration. The optimal strategy would allocate to each configuration $i$ the minimum resource required to distinguish it from $\nu_1$, i.e., enough so that the envelope functions depicted in Figure 1(c) bound the intermediate loss to be less than $\frac{\nu_i - \nu_1}{2}$ away from the terminal value. As shown in Jamieson & Talwalkar (2015) and Li et al. (2016), the budget required by SuccessiveHalving is in fact only a small factor away from this optimal approach because it capitalizes on configurations that are easy to distinguish from $\nu_1$. In contrast, the naive uniform allocation strategy, which allocates $B/n$ to each configuration, has to allocate to every configuration the resource required to distinguish $\nu_2$ from $\nu_1$.

The relative size of the budget required for uniform allocation and SuccessiveHalving depends on the envelope functions bounding deviation from terminal losses as well as the distribution from which $\nu_i$'s are drawn. The budget required for SuccessiveHalving is smaller when the optimal $n$ versus $B/n$ tradeoff requires fewer resources per configuration. Hence, if the envelope functions tighten quickly as a function of resource allocated, or the average distances between terminal losses is large, then SuccessiveHalving can be substantially faster than uniform allocation. Of course we do not have knowledge of either function in practice, so we will hedge our aggressiveness with Hyperband. Remarkably, despite having no knowledge of the envelope functions or the distribution of $\nu_i$'s, Hyperband requires a budget that is only log factors larger than the optimal for SuccessiveHalving. See Li et al. (2016) for details.



(a) CIFAR-10 (b) MRBI (c) SVHN

Figure 3: Average test error across 10 trials is shown in all plots. Label "SMAC_early" corresponds to SMAC with the early stopping criterion proposed in Domhan et al. (2015) and label "bracket $s = 4$" corresponds to repeating the most exploratory bracket of HYPERBAND.

## 4 HYPERPARAMETER OPTIMIZATION EXPERIMENTS

In this section, we evaluate the empirical behavior of HYPERBAND with iterations, data subsamples, and features as resources. For all experiments, we compare HYPERBAND with three well known Bayesian optimization algorithms — SMAC, TPE, and Spearmint. Additionally, we show results for SUCCESSIVEHALVING corresponding to repeating the most exploration bracket of HYPERBAND. Finally for all experiments, we benchmark against standard random search and random_2×, which is a variant of random search with twice the budget of other methods.

### 4.1 EARLY STOPPING ITERATIVE ALGORITHMS FOR NEURAL NETWORKS

We study a convolutional neural network with the same architecture as that used in Snoek et al. (2012) and Domhan et al. (2015) from cuda-convnet. The search spaces used in the two previous works differ, and we used a search space similar to that of Snoek et al. (2012) with 6 hyperparameters for stochastic gradient decent and 2 hyperparameters for the response normalization layers. In line with the two previous works, we used a batch size of 100 for all experiments. For these experiments, we also compare against a variant of SMAC named SMAC_early that uses the early termination criterion proposed in Domhan et al. (2015) for neural networks. We view SMAC with early stopping to be a combination of adaptive configuration selection and configuration evaluation. See Appendix A for more details about the experimental setup.

**Datasets:** We considered three image classification datasets: CIFAR-10 (Krizhevsky, 2009), rotated MNIST with background images (MRBI) (Larochelle et al., 2007), and Street View House Numbers (SVHN) (Netzer et al., 2011). CIFAR-10 and SVHN contain 32 × 32 RGB images while MRBI contains 28 × 28 grayscale images. The splits used for each dataset are as follows: (1) CIFAR-10 has 40k, 10k, and 10k instances; (2) MRBI has 10k, 2k, and 50k instances; and (3) SVHN has close to 600k, 6k, and 26k instances for training, validation, and test respectively. For all datasets, the only preprocessing performed on the raw images was demeaning.

**HYPERBAND Configuration:** For these experiments, one unit of resource corresponds to 100 mini-batch iterations. For CIFAR-10 and MRBI, $R$ is set to 300 (or 30k total iterations). For SVHN, $R$ is set to 600 (or 60k total iterations) to accommodate the larger training set. $\eta$ was set to 4 for all experiments, resulting in 5 SUCCESSIVEHALVING brackets for HYPERBAND.

**Results:** Ten independent trials were performed for each searcher. For CIFAR-10, the results in Figure 3(a) show that HYPERBAND is more than an order of magnitude faster than its competitors. In Figure 6 of Appendix A, we extend the $x$-axis for CIFAR-10 out to $100R$. The results show that Bayesian optimization methods ultimately converge to similar errors as HYPERBAND. For MRBI, HYPERBAND is more than an order of magnitude faster than standard configuration selection approaches and 5× faster than SMAC with early stopping. For SVHN, while HYPERBAND finds a good configuration faster, Bayesian optimization methods are competitive and SMAC with early stopping outperforms HYPERBAND. This result demonstrates that there is merit to incorporating early stopping with configuration selection approaches.

Across the three datasets, HYPERBAND and SMAC_early are the only two methods that consistently outperform random_2×. On these datasets, HYPERBAND is over $20\times$ faster than random search while SMAC_early is $\leq 7\times$ faster than random search within the evaluation window. In fact, the first result returned by HYPERBAND after using a budget of $5R$ is often competitive with results returned by other searchers after using $50R$. Additionally, HYPERBAND is less variable than other searchers across trials, which is highly desirable in practice (see Appendix A for plots with error bars).

For computationally expensive problems in high dimensional search spaces, it may make sense to just repeat the most exploratory brackets. Similarly, if meta-data is available about a problem or it is known that the quality of a configuration is evident after allocating a small amount of resource, then one should just repeat the most exploration bracket. Indeed, for these experiments, repeating the most exploratory bracket of HYPERBAND outperforms cycling through all the brackets. In fact, bracket $s = 4$ vastly outperforms all other methods on CIFAR-10 and MRBI and is nearly tied with SMAC_early for first on SVHN.

Finally, CIFAR-10 is a very popular dataset and state-of-the-art models achieve much better accuracy than what is shown in Figure 3. The difference in performance is mainly attributable to higher model complexities and data manipulation (i.e. using reflection or random cropping to artificially increase the dataset size). If we limit the comparison to published results that use the same architecture and exclude data manipulation, the best human expert result for the dataset is 18% error and hyperparameter optimized result is 15.0% for Snoek et al. (2012)[4] and 17.2% for Domhan et al. (2015). These results are better than our results on CIFAR-10 because they use 25% more data by including the validation set and also train for more epochs. The best model found by HYPERBAND achieved a test error of 17.0% when trained on the combined training and validation data for 300 epochs.

### 4.2 DATA DOWNSAMPLING KERNEL REGULARIZED LEAST SQUARES CLASSIFICATION

In this experiment, we use HYPERBAND with data samples as the resource to optimize the hyperparameters of a kernel-based classification task on CIFAR-10. We use the multi-class regularized least squares classification model which is known to have comparable performance to SVMs (Rifkin & Klautau, 2004; Agarwal et al., 2014) but can be trained significantly faster. The hyperparameters considered in the search space include preprocessing method, regularization, kernel type, kernel length scale, and other kernel specific hyperparameters (see Appendix A for more details). HYPERBAND is run with $\eta = 4$ and $R = 400$, with each unit of resource representing 100 datapoints. Similar to previous experiments, these inputs result in a total of 5 brackets. Each hyperparameter optimization algorithm is run for ten trials on Amazon EC2 `m4.2xlarge` instances; for a given trial, HYPERBAND is allowed to run for two outer loops, bracket $s = 4$ is repeated 10 times, and all other searchers are run for 12 hours.

Figure 4 shows that HYPERBAND returns a good configuration after just the first SUCCESSIVEHALVING bracket in approximately 20 minutes; other searchers fail to reach this error rate on average even after the entire 12 hours. Notably, HYPERBAND was able to evaluate over 250 configurations in this first bracket of SUCCESSIVEHALVING, while competitors were able to evaluate only three configurations in the same amount of time. Consequently, HYPERBAND is over $30\times$ faster than Bayesian optimization methods and $70\times$ faster than random search. Bracket $s = 4$ sightly outperforms HYPERBAND but the terminal performance for the two algorithms are the same. Random_2× is competitive with SMAC and TPE.

### 4.3 FEATURE SUBSAMPLING TO SPEED UP APPROXIMATE KERNEL CLASSIFICATION

We next demonstrate the performance of HYPERBAND when using features as a resource, focusing on random feature approximations for kernel methods. Features are randomly generated using the method described in Rahimi & Recht (2007) to approximate the RBF kernel, and these random features are then used as inputs to a ridge regression classifier. We consider hyperparameters of a random feature kernel approximation classifier trained on CIFAR-10, including preprocessing method, kernel length scale, and $l_2$ penalty. We impose an upper bound of 100k random features for the kernel approximation so that the data will comfortably fit into a machine with 60GB of

[4]We were unable to reproduce this result even after receiving the optimal hyperparameters from the authors through a personal communication.

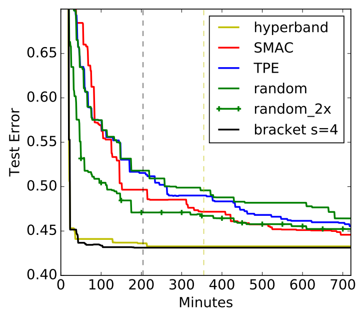


Figure 4: Average test error of the best kernel regularized least square classification model found by each searcher on CIFAR-10. The color coded dashed lines indicate when the last trial of a given searcher finished.

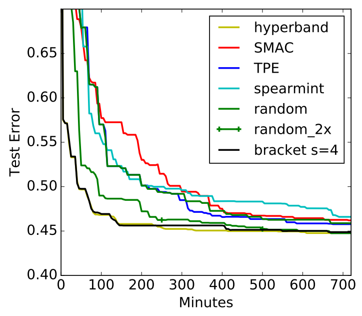


Figure 5: Average test error of the best random features model found by each searcher on CIFAR-10. The test error for HYPERBAND and bracket $s = 4$ are calculated in every evaluation instead of at the end of a bracket.

memory. Additionally, we set one unit of resource to be 100 features for an $R = 1000$, which gives 5 different brackets with $\eta = 4$. Each searcher is run for 10 trials, with each trial lasting 12 hours on a `n1-standard-16` machine from Google Cloud Compute. The results in Figure 5 show that HYPERBAND is around 6x faster than Bayesian methods and random search. HYPERBAND performs similarly to bracket $s = 4$. Random_$2\times$ outperforms Bayesian optimization algorithms.

### 4.4 EXPERIMENTAL DISCUSSION

For a given $R$, the most exploratory SUCCESSIVEHALVING round performed by HYPERBAND evaluates $\eta^{\lfloor \log_\eta(R) \rfloor}$ configurations using a budget of $(\lfloor \log_\eta(R) \rfloor + 1)R$, which gives an upper bound on the potential speedup over random search. If training time scales linearly with the resource, the maximum speedup offered by HYPERBAND compared to random search is $\frac{\eta^{\lfloor \log_\eta(R) \rfloor}}{(\lfloor \log_\eta(R) \rfloor+1)}$. For the values of $\eta$ and $R$ used in our experiments, the maximum speedup over random search is approximately $50\times$ given linear training time. However, we observe a range of speedups from $6\times$ to $70\times$ faster than random search. The differences in realized speedup can be explained by two factors: (1) the scaling properties of total evaluation time as a function of the allocated resource and (2) the difficulty of finding a good configuration.

If training time is superlinear as a function of the resource, then HYPERBAND can offer higher speedups. More generally, if training scales like a polynomial of degree $p > 1$, the maximum speedup of HYPERBAND over random search is approximately $\frac{\eta^{p-1}-1}{\eta^{p-1}} \eta^{\lfloor \log_\eta(R) \rfloor}$. Hence, higher speedups were observed for the the kernel least square classifier experiment discussed in Section 4.2 because the training time scaled quadratically as a function of the resource.

If 10 randomly sampled configurations is sufficient to find a good hyperparameter setting, then the benefit of evaluating orders of magnitude more configurations is muted. Generally the difficulty of the problem scales with the dimension of the search space since coverage diminishes with dimensionality. For low dimensional problems, the number of configurations evaluated by random search and Bayesian methods is exponential in the number of dimensions so good coverage can be achieved; i.e. if $d = 3$ as in the features subsampling experiment, then $n = O(2^d = 8)$. Hence, HYPERBAND is only $6\times$ faster than random search on the feature subsampling experiment. For the neural network experiments however, we hypothesize that faster speedups are observed for HYPERBAND because the dimension of the search space is higher.

## 5 FUTURE WORK

We have introduced a novel bandit-based method for adaptive configuration evaluation with demonstrated competitive empirical performance. Future work involves exploring (i) embedding HYPERBAND into parallel and distributed computing environments; (ii) adjusting for training methods with different convergence rates; and (iii) combining HYPERBAND with non-random sampling methods.

## REFERENCES

A. Agarwal, J. Duchi, P. L. Bartlett, and C. Levrard. Oracle inequalities for computationally budgeted model selection. In *COLT*, 2011.

A. Agarwal, S. Kakade, N. Karampatziakis, L. Song, and G. Valiant. Least squares revisited: Scalable approaches for multi-class prediction. In *ICML*, 2014.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. In *JMLR*, 2012.

J. Bergstra et al. Algorithms for hyper-parameter optimization. In *NIPS*, 2011.

T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *IJCAI*, 2015.

K. Eggensperger et al. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS Bayesian Optimization Workshop*, 2013.

A. György and L. Kocsis. Efficient multi-start strategies for local search algorithms. *JAIR*, 41, 2011.

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

K. Jamieson and R. Nowak. Best-arm identification algorithms for multi-armed bandits in the fixed confidence setting. In *Information Sciences and Systems (CISS), 2014 48th Annual Conference on*, pp. 1–6. IEEE, 2014.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *AISTATS*, 2015.

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. *arXiv preprint arXiv:1605.07079*, 2016.

A. Krizhevsky. Learning multiple layers of features from tiny images. In *Technical report, Department of Computer Science, Univsersity of Toronto*, 2009.

T. Krueger, D. Panknin, and M. Braun. Fast cross-validation via sequential testing. *Journal of Machine Learning Research*, 16:1103–1155, 2015.

H. Larochelle et al. An empirical evaluation of deep architectures on problems with many factors of variation. In *ICML*, 2007.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *arXiv:1603.06560*, 2016.

O. Maron and A. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In *NIPS*, 1993.

Y. Netzer et al. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

G. Rätsch, T. Onoda, and K.R. Müller. Soft margins for adaboost. *Machine Learning*, 42:287–320, 2001.

R. Rifkin and A. Klautau. In defense of one-vs-all classification. *JMLR*, 2004.

A. Sabharwal, H. Samulowitz, and G. Tesauro. Selecting near-optimal learners via incremental data allocation. In *AAAI*, 2016.

P. Sermanet, S. Chintala, and Y. LeCun. Convolutional neural networks applied to house numbers digit classification. In *ICPR*, 2012.

J. Snoek, H. Larochelle, and R. Adams. Practical bayesian optimization of machine learning algorithms. In *NIPS*, 2012.

J. Snoek et al. Bayesian optimization using deep neural networks. In *ICML*, 2015.

E. Sparks, A. Talwalkar, D. Haas, M. J. Franklin, M. I. Jordan, and T. Kraska. Automating model search for large scale machine learning,. In *Symposium on Cloud Computing*, 2015.

K. Swersky, J. Snoek, and R. Adams. Multi-task bayesian optimization. In *NIPS*, 2013.

K. Swersky, J. Snoek, and R. P. Adams. Freeze-thaw bayesian optimization. *arXiv:1406.3896*, 2014.

C. Thornton et al. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *KDD*, 2013.

## A Additional Experimental Results

In this section, we present a comparison of Hyperband with the CVST algorithm from Krueger et al. (2015) and provide additional details for experiments presented in Section 3 and 4.

### A.1 Comparison with CVST

The CVST algorithm from Krueger et al. (2015) focuses on speeding up standard k-fold cross-validation. We did not include it as one of the competitors in Section 4 because the experiments we selected were too computational expensive for multi-fold cross-validation and CVST is not an any time algorithm. Nonetheless, the CVST algorithm is an interesting approach and was shown to have promising empirical performance in Krueger et al. (2015). Hence, we performed a small scale comparison modeled after their empirical studies between CVST and Hyperband.

We replicated the classification experiments in Krueger et al. (2015) that train a support vector machine on the datasets from the IDA benchmark (Rätsch et al., 2001). All experiments were performed on Google Cloud Compute's `n1-standard-1` instances. Following Krueger et al. (2015), we evaluated Hyperband and CVST on the same 2d grid of 610 hyperparameters and recorded the best test error and duration for 50 trials . The only modification we made to their original experimental setup was the data splits; instead of half for test and half for training, we used 1/11th for test and 10/11th for training. Hyperband performed holdout evaluation using 1/10th of the training data as the validation set. We set $\eta = 3$, and $R$ was set for each dataset so that a minimum resource of 50 datapoints is allocated to each configuration. Table 2 shows that CVST and Hyperband achieve comparable test errors (the differences are well within the error bars), while Hyperband is significantly faster than CVST on all datasets. More granularly, while CVST on average has slightly lower mean error, Hyperband is within $0.2\%$ of CVST on 5 of the 7 datasets. Additionally, for each of the 7 datasets, Hyperband does as well as or better than CVST in over half of the trials.

| | CVST | | Hyperband | | Ratio |
|---|---|---|---|---|---|
| Dataset | Test Error | Duration | Test Error | Duration | Duration |
| banana | 9.8%±1.6% | 12.3±5.0 | 9.9%±1.5% | 1.8±0.1 | 6.7±2.8 |
| german | 26.0%±4.5% | 2.7±1.1 | 27.6%±4.8% | 0.7±0.0 | 4.1±1.7 |
| image | 2.9%±1.1% | 3.5±1.0 | 3.3%±1.4% | 1.0±0.0 | 3.4±0.9 |
| splice | 8.6%±1.8% | 10.6±3.1 | 8.7%±1.8% | 3.9±0.1 | 2.7±0.8 |
| ringnorm | 1.4%±0.4% | 21.3±2.3 | 1.5%±0.4% | 6.5±0.3 | 3.3±0.4 |
| twonorm | 2.4%±0.5% | 27.9±10.0 | 2.4%±0.5% | 6.5±0.2 | 4.3±1.5 |
| waveform | 9.3%±1.3% | 13.7±2.7 | 9.5%±1.3% | 2.9±0.2 | 4.8±1.0 |

Table 2: The test error and duration columns show the average value plus/minus the standard deviation across 50 trials. Duration is measured in minutes and indicates how long it took each method to evaluate the grid of 610 hyperparameters used in Krueger et al. (2015). The ratio column shows the ratio of the duration for Hyperband over that for CVST with associated standard deviation.

### A.2 LeNet experiment

We trained the LeNet convolutional neural network on MNIST using mini-batch SGD. Code is available for the network at `http://deeplearning.net/tutorial/lenet.html`. The search space for the LeNet example discussed in Section 3.2 is shown in Table 3.

| Hyperparameter | Scale | Min | Max |
|---|---|---|---|
| Learning Rate | log | 1e-3 | 1e-1 |
| Batch size | log | 1e1 | 1e3 |
| Layer-2 Num Kernels (k2) | linear | 10 | 60 |
| Layer-1 Num Kernels (k1) | linear | 5 | k2 |

Table 3: Hyperparameter space for the LeNet application of Section 3.2. Note that the number of kernels in Layer-1 is upper bounded by the number of kernels in Layer-2.

### A.3 EXPERIMENTS USING ALEX KRIZHEVSKY'S CNN ARCHITECTURE

For the experiments discussed in Section 4.1, the exact architecture used is the 18% model provided on cuda-convnet for CIFAR-10.[5]

| Hyperparameter | Scale | Min | Max |
|---|---|---|---|
| *Learning Parameters* | | | |
| Initial Learning Rate | log | $5 * 10^{-5}$ | 5 |
| Conv1 $l_2$ Penalty | log | $5 * 10^{-5}$ | 5 |
| Conv2 $l_2$ Penalty | log | $5 * 10^{-5}$ | 5 |
| Conv3 $l_2$ Penalty | log | $5 * 10^{-5}$ | 5 |
| FC4 $l_2$ Penalty | log | $5 * 10^{-3}$ | 500 |
| Learning Rate Reductions | integer | 0 | 3 |
| *Local Response Normalization* | | | |
| Scale | log | $5 * 10^{-6}$ | 5 |
| Power | linear | 0.01 | 3 |

Table 4: Hyperparameters and associated ranges for the three-layer convolutional network.

**Search Space:** The search space used for the experiments is shown in Table 4. The learning rate reductions hyperparameter indicates how many times the learning rate was reduced by a factor of 10 over the maximum iteration window. For example, on CIFAR-10, which has a maximum iteration of 30,000, a learning rate reduction of 2 corresponds to reducing the learning every 10,000 iterations, for a total of 2 reductions over the 30,000 iteration window. All hyperparameters with the exception of the learning rate decay reduction overlap with those in Snoek et al. (2012). Two hyperparameters in Snoek et al. (2012) were excluded from our experiments: (1) the width of the response normalization layer was excluded due to limitations of the Caffe framework and (2) the number of epochs was excluded because it is incompatible with dynamic resource allocation.

**Datasets:** CIFAR-10 and SVHN contain $32 \times 32$ RGB images while MRBI contains $28 \times 28$ grayscale images. For all datasets, the only preprocessing performed on the raw images was demeaning. For CIFAR-10, the training (40,000 instances) and validation (10,000 instances) sets were sampled from data batches 1-5 with balanced classes. The original test set (10,000 instances) is used for testing. For MRBI, the training (10,000 instances) and validation (2,000 instances) sets were sampled from the original training set with balanced classes. The original test set (50,000 instances) is used for testing. Lastly, for SVHN, the train, validation, and test splits were created using the same procedure as that in Sermanet et al. (2012).

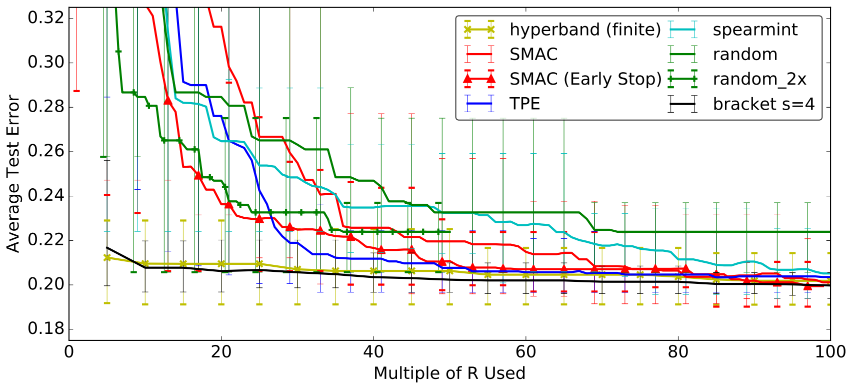
**Computational Considerations:** The experiments took the equivalent of over 1 year of GPU hours on NVIDIA GRID K520 cards available on Amazon EC2 `g2.8xlarge` instances. We set a total budget constraint in terms of iterations instead of compute time to make comparisons hardware independent.[6] Comparing progress by iterations instead of time ignores overhead costs not associated with training like cost of configuration selection for Bayesian methods and model initialization and validation costs for HYPERBAND. While overhead is hardware dependent, the overhead for HYPERBAND is below 5% on EC2 `g2.8xlarge` machines, so comparing progress by time passed would not impact results significantly.

Due to the high computational cost of these experiments, we were not able to run all searchers out to convergence. However, we did double the budget for each trial of CIFAR-10 to allow for a comparison of the searchers as they near convergence. Figure 6 shows while Bayesian optimization methods achieve similar performance as HYPERBAND and SUCCESSIVEHALVING, it takes them much longer to achieve a comparable error rate.

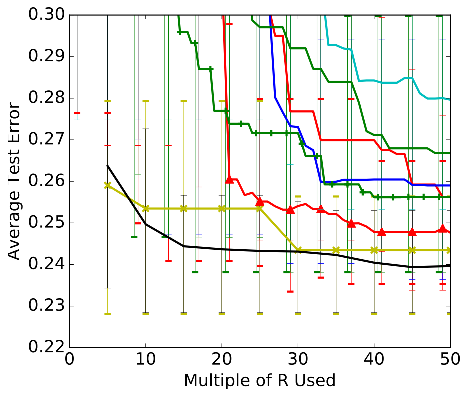
**Comparison with Early Stopping:** Adaptive allocation for hyperparameter optimization can be thought of as a form of early stopping where less promising configurations are halted before completion. Domhan et al. (2015) propose an early stopping method for neural networks and combine it

[5]The model specification is available at `http://code.google.com/p/cuda-convnet/`.

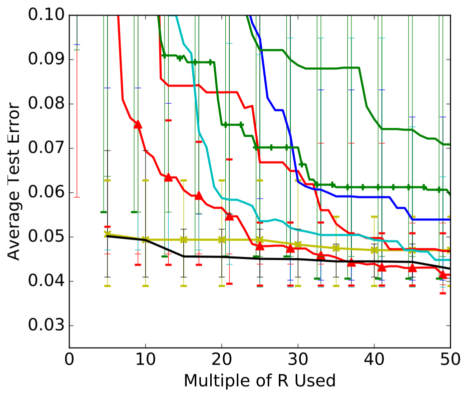
[6]Most trials were run on Amazon EC2 g2.8xlarge instances but a few trials were run on different machines due to the large computational demand of these experiments.



(a) CIFAR-10



(b) MRBI



(c) SVHN

Figure 6: Average test error across 10 trials is shown in all plots. Error bars indicate the maximum and minimum ranges of the test error corresponding to the model with the best validation error

with SMAC to speed up hyperparameter optimization. Their method stops training a configuration if the probability of the configuration beating the current best is below a specified threshold. This probability is estimated by extrapolating learning curves fit to the intermediate validation error losses of a configuration. If a configuration is terminated early, the predicted terminal value from the estimated learning curves is used as the validation error passed to the hyperparameter optimization algorithm. Hence, if the learning curve fit is poor, it could impact the performance of the configuration selection algorithm. While this approach is heuristic in nature, it does demonstrate promising empirical performance so we included SMAC with early termination as a competitor. We used the conservative termination criterion with default parameters and recorded the validation loss every 400 iterations and evaluated the termination criterion 3 times within the training period (every 8k iterations for CIFAR-10 and MRBI and every 16k iterations for SVHN).[7] Comparing performance by the total multiple of $R$ used is conservative because it does not account for the time spent fitting the learning curve in order to check the termination criterion.

## A.4 Kernel classification experiments

We trained the regularized least-squares classification model using a block coordinate descent solver. Our models take less than 10 minutes to train on CIFAR-10 using an 8 core machine, while the default SVM method in Scikit-learn is single core and takes hours. Table 5 shows the hyperparameters and associated ranges considered in the kernel least squares classification experiment discussed in

[7]We used the code provided at https://github.com/automl/pylearningcurvepredictor.

| Hyperparameter | Type | Values |
|---|---|---|
| preprocessor | Categorical | min/max, standardize, normalize |
| kernel | Categorical | rbf, polynomial, sigmoid |
| C | Continuous | log $[10^{-3}, 10^{5}]$ |
| gamma | Continuous | log $[10^{-5}, 10]$ |
| degree | if kernel=poly | integer [2,5] |
| coef0 | if kernel=poly,sigmoid | uniform [-1.0, 1.0] |

Table 5: Hyperparameter space for kernel regularized least squares classification problem discussed in Section 4.2.

Section 4.2. The cost term C is divided by the number of samples so that the tradeoff between the squared error and the $l_2$ penalty would remain constant as the resource increased (squared error is summed across observations and not averaged). The regularization term $\lambda$ is equal to the inverse of the scaled cost term C. Additionally, the average test error with associated minimum and maximum ranges across 10 trials are show in Figure 7.

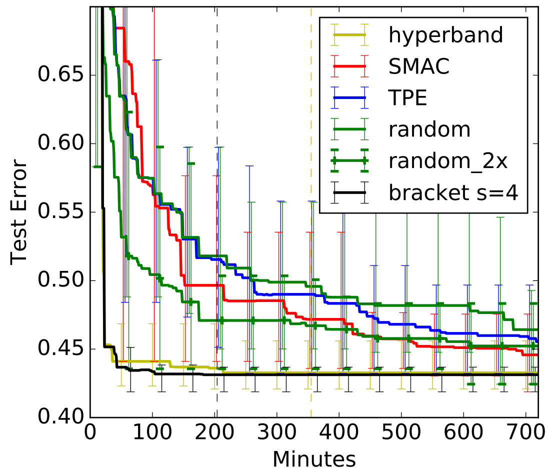


Figure 7: Average test error of the best kernel regularized least square classification model found by each searcher on CIFAR-10. The color coded dashed lines indicate when the last trial of a given searcher finished. Error bars correspond to observed minimum and maximum test error across 10 trials.

| Hyperparameter | Type | Values |
|---|---|---|
| preprocessor | Categorical | none, min/max, standardize, normalize |
| $\lambda$ | Continuous | log $[10^{-3}, 10^{5}]$ |
| gamma | Continuous | log $[10^{-5}, 10]$ |

Table 6: Hyperparameter space for random feature kernel approximation classification problem discussed in Section 4.3.

Table 6 shows the hyperparameters and associated ranges considered in the random features kernel approximation classification experiment discussed in Section 4.3. The regularization term $\lambda$ is divided by the number of features so that the tradeoff between the squared error and the $l_2$ penalty would remain constant as the resource increased. Additionally, the average test error with associated minimum and maximum ranges across 10 trials are show in Figure 8.

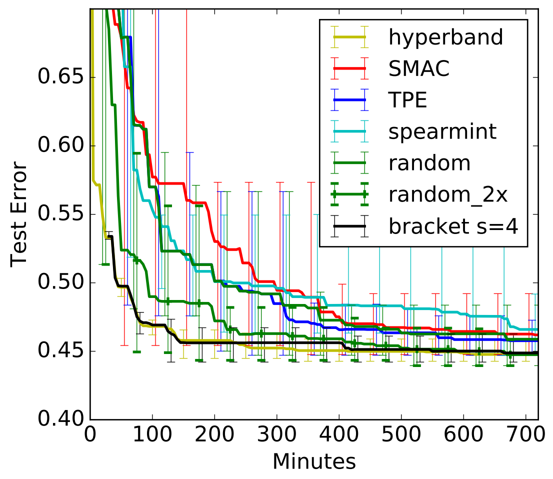


Figure 8: Average test error of the best random features model found by each searcher on CIFAR-10. The test error for HYPERBAND and bracket $s = 4$ are calculated in every evaluation instead of at the end of a bracket. Error bars correspond to observed minimum and maximum test error across 10 trials.