An Analysis of Methods of Integration of Hyperparameter Optimization on Neuroevolutionary Models

Krishna Chittur

Supervised by Dr. Bruce Porter

A thesis submitted to
the University of Texas at Austin
for the Turing Scholars program

Department of Computer Science
College of Natural Sciences
University of Texas at Austin
May 2020

ABSTRACT

The field of automated machine learning has developed numerous methods for the problem of neural architecture search (NAS). One of these is neuroevolution, where evolutionary algorithms are used to evolve model architectures. When employing this approach, the question remains of how best to optimize hyperparameters of the training process such as batch size, learning rate, and the choice of optimizer. To this end, we survey various methods of integrating hyperparameter optimization into a neuroevolutionary pipeline to determine their effect on a wide variety of datasets. We further analyze meta-features of these datasets to see if the efficacy of hyperparameter optimization could be contingent on these meta-features.

ACKNOWLEDGMENTS

This research was funded and made possible by SparkCognition's Darwin project. I would like to acknowledge the support of my advisor, Dr. Bruce Porter, as well as Tyler McDonnell, my manager, who suggested this line of research. The template for this thesis was created by Luuk Earl of the University of Birmingham.

Table of Contents

1 Introduction and Background

1.1 Neuroevolution

Deep neural networks have been historically employed to use statistical inference to solve difficult problems in a wide variety of domains, including vision, language, and signal processing. By capturing hidden patterns and structures in labeled training data, a neural network can accurately predict labels on unseen data. However, the performance of such a network can be heavily contingent on one's choice of model architecture, and it is still an open problem as to how best to determine this architecture for a given labeled training set.

In recent years, much research has been done on algorithms to automatically find effective architectures for neural networks to train on a given labeled dataset. These algorithms are known as neural architecture search (NAS) algorithms. Generally these algorithms work by fixing a given search space of architectures to search over, and then sampling architectures from that space to train models. Common NAS methods include reinforcement learning, as popularized by Zoph and Le, 2016 and Baker et al., 2016, as well as evolutionary algorithms, as in Real, Moore, et al., 2017. There are also many approaches that are designed for efficiency, e.g. via sharing parameters between the trained models as in Pham et al., 2018.

Here, we focus on the approach of neuroevolution, where model architectures are evolved over time, specifically in the context of Darwin^{TM} (McDonnell et al., 2018), an enterprise automated machine learning platform[1]. Darwin uses a hybrid approach to train both neural networks and other types of models such as gradient boosted decision trees and random forests, while also incorporating ensemble methods (McDonnell et al., 2018). Neural networks are evolved using a genetic algorithm that gradually constructs more complex models from simpler initial genomes by combining layers or groups of layers ("genes") of different structures and sizes. As Darwin has extensive support for time-series models, these genes may, for example, be taken as intermediate layers from an LSTM or a temporal convolutional

[1]https://www.sparkcognition.com/product/darwin/

network (TCN) (Bai, Kolter, and Koltun, 2018).

1.2 Motivation: Hyperparameter Optimization

The models generated by Darwin's genetic algorithm are trained using backpropagation and a standard set of default hyperparameters. This means that the batch size, learning rate, and choice of optimizer, among other aspects of the training process, are fixed across all neural networks trained on all datasets. While it is not uncommon for architecture search algorithms to fix these hyperparameters across all models[2], most neural architecture search papers tend to focus on a narrowly defined problem like CIFAR-10 (Krizhevsky, Hinton, et al., 2009). In contrast, when training a variety of models over a wide range of datasets as in Darwin, is it not unreasonable to imagine that a single set of default hyperparameters is no longer sufficient to handle all possible use cases.

In order to judge the usefulness of hyperparameter optimization, we added to Darwin the ability to run a hyperparameter optimizer as an optional tune step after the main neuroevolution algorithm. This tune step takes the top k best-performing models and re-trains them with a different set of hyperparameters. For the purposes of testing, we set $k = 2$ and tried two different optimizers, random search and Hyperband (Elsken, Metzen, and Hutter, 2018), with varying amounts of time allocated for hyperparameter optimization.

In this document, we further analyze if the potential accuracy improvements from having this additional tune step in any way correlate with meta-features of the dataset the models are being trained on, such as skew, kurtosis, and various metrics of dataset difficulty. More on this later.

[2] e.g. see Real, Aggarwal, et al., 2019 for one such example in neuroevolution, or Zoph and Le, 2016 for reinforcement learning-based NAS.

Hyperband

Hyperband is a hyperparameter search algorithm introduced in Elsken, Metzen, and Hutter, 2018. The Hyperband algorithm is based on a simpler algorithm known as Successive Halving, as introduced by Jamieson and Talwalkar, 2016, which is a solution to the best arm problem for multi-armed bandits. In the multi-armed bandit problem, one is given a set of competing choices ("arms") where each arm provides a random reward drawn from an unknown probability distribution each time some resource is allocated. The overall challenge is to balance exploration and exploitation between the arms in a way that minimizes cumulative regret. Exploring this tradeoff is highly relevant to hyperparameter search over neural networks, as these arms can be thought of as competing hyperparameter configurations. In this case, the process of allocating resources and collecting rewards can be thought of as training a model with a given hyperparameter configuration for a certain number of epochs and observing the accuracy of the trained model on a validation set.

The Successive Halving algorithm takes as input two parameters: n, the initial number of samples, and r, the required minimum amount of resources to allocate to each sample. The implementation of Successive Halving in Darwin also takes another parameter, j, which is the conversion ratio of resources to epochs. After training the n samples for rj epochs each, the less performant half of the samples are discarded, and the remaining samples are trained for $2rj$ epochs instead. Likewise, in the third round, the best $\lceil \frac{n}{4} \rceil$ samples are trained for $4rj$ epochs, and so on. A total of $\lceil \log_\eta n \rceil$ rounds are needed for a single sample to remain.

The variant of Successive Halving used by Hyperband, shown in Algorithm 1, generalizes this further by substituting 2, the proportion of samples discarded each round, with a parameter $\eta \in \mathbb{Z}, \eta >= 2$. Furthermore, rather than running for the full $\lceil \log_\eta n \rceil$ rounds as in traditional Successive Halving, another parameter s is taken to indicate the number of rounds of Successive Halving to run.

Notice that when $s = 0$, Successive Halving only runs for a single iteration and allocates

Algorithm 1: Successive Halving with additional parameters η, s.

```
1 General Successive Halving (r, j, η, n, s);
  Input  : r, j, η, n, s where $0 \leq s \leq \lceil \log_\eta n \rceil$
  Output: $T_s$, a set of $n_s$ samples
2 $T_0 \leftarrow$ get_samples($n$);
3 for $i \leftarrow 0$ to $s$ do
4     $n_i \leftarrow \lfloor n\eta^{-i} \rfloor$;
5     $r_i \leftarrow \lceil r\eta^{i} \rceil$;
6     $T_i \leftarrow$ run_and_get_top_k($S_{i-1}, jr_i, n_i$);
7 end
8 return $T_s$;
```

equal resources to all models, making it identical to random search. Conversely, when $s = \lceil \log_\eta n \rceil$, we have the original Successive Halving algorithm. Given a fixed initial budget B, there is a tradeoff between n and B/n, as having a larger n allows for taking more samples from the search space and thus more exploration, whereas a smaller n allows for more resources to be allocated to each model i.e. exploitation.

Hyperband, shown in Algorithm 2, takes two parameters: η, which is the same as before, and R, the maximum amount of resources to be allocated to any single given model. It then performs a 1D grid search over the possible variations of Successive Halving with the given value for R. As it starts with full Successive Halving and ends with random search, Hyperband is an anytime algorithm, and is implemented as such in Darwin.

Why Hyperband?

While random search is already know to be very effective for hyperparameter optimization (Bergstra and Bengio, 2012), there were several reasons we chose to also implement Hyperband as an optional optimization algorithm when benchmarking the efficacy of hyper-

Algorithm 2: Hyperband as implemented in Darwin$^{\mathrm{TM}}$.

```
1  Hyperband (η, R, j, k);
   Input  : η, R, j, k
   Output: The top k performing samples, trained for jR epochs each
2  $s_{\max} \leftarrow \lceil \log_\eta R \rceil$;
   // budget for each iteration
3  $B \leftarrow R\,(s_{\max} + 1)$;
   // priority queue of trained samples, sorted by fitness
4  $Q \leftarrow$ empty list;
5  for $s \leftarrow s_{max}$ to 0 do
6      $n \leftarrow \lceil \frac{B\eta^s}{R(s+1)} \rceil$;
7      $r \leftarrow R\eta^{-s}$;
8      $T \leftarrow$ draw_samples($n$);
9      $T' \leftarrow$ SuccessiveHalving($r, j, \eta, n, s$);
10     extend ($Q, T'$);
       // Terminate early if time limit is reached
11 end
12 return take($Q, k$);
```

parameter optimization in Darwin. As an anytime algorithm, Hyperband can be made to adhere to strict time limitations. This ability to terminate after an arbitrary amount of time is highly relevant to the implementation of Darwin, which takes a user-provided time limit, and it works well with the existing anytime nature of neuroevolution.

Furthermore, Hyperband does not make any assumptions about the sampling distribution. While we currently use uniform random sampling, it would not be difficult to later change the sampling algorithm to correlate certain hyperparameters or to condition the search space on the chosen model architecture or even model parameters.

Another valuable aspect of Hyperband, and of bandit-based algorithms in general, is their ability to stop training early for poorly-performing models. During testing, we found that the early performance of poorly-performing configurations tended to be relatively predictive of later performance. Being able to leverage early stopping thus allows for the configuration space to be searched more efficiently, in contrast to random search and Bayesian approaches, where the process of training is more likely to be treated as evaluating a black-box function (Snoek, Larochelle, and Adams, 2012).

Finally, Hyperband is a very simple algorithm, making it easy to implement in Darwin as well as to test programmatically.

1.3 Related Work

The general problem of simultaneously choosing a learning algorithm and optimizing its hyperparameters for a given dataset has been called the *combined algorithm selection and and hyperparameter optimization problem* (CASH) in the context of Auto-WEKA (Thornton et al., 2013) (Kotthoff et al., 2017), although these papers do not include neural networks in their model search space. They approach the combined problem with a hierarchical hyperparameter search using modern Bayesian algorithms such as SMAC (Hutter, Hoos, and Leyton-Brown, 2011). A similar approach is taken by Auto-sklearn (Feurer et al., 2018), which uses a hybrid Bayesian and bandit-based algorithm based on BOHB (Falkner, Klein, and Hutter, 2018) to optimize the entire ML pipeline, including the process of data preprocessing. Like Auto-WEKA, deep neural networks are not part of the search space, partially due to tight time and memory constraints.

2 Dataset Exploration

2.1 Meta-Features

Automated machine learning, or AutoML, is a subfield of machine learning that concerns itself with automating the standard machine learning process of building, training, and testing models to solve one or more problems. Closely related to AutoML is the field of meta learning, which is a subfield of machine learning that deals with the challenge of generalizing automated machine learning techniques across different tasks by learning from experience. Often, this experience is characterized by metadata about previously solved problems. When this metadata is in the form of statistics computed across the training dataset, they are often referred to as meta-attributes (Bensusan, Giraud-Carrier, and Kennedy, 2000) or, more commonly, as meta-features (Bilalli, Abelló, and Aluja-Banet, 2017).

Meta-features are generally divided into a small group of categories, including, but not limited to: statistical methods, information-theory based characterizations, landmarking, model-based methods (especially decision-tree-based methods) (Bensusan, Giraud-Carrier, and Kennedy, 2000) (Bilalli, Abelló, and Aluja-Banet, 2017), and other methods, such as clustering-based and concept-based measures (Rivolli et al., 2018). In particular, Rivolli et al., 2018 goes further when devising its taxonomy of meta-features and distinguishes between two components of a given meta-feature as a characterization measure m and a summarization function σ, as follows[3].

Let $\mathcal{D}$ be a set of datasets, where a dataset $D \in \mathcal{D}$ is a sequence of labeled training data points, $D = \{(\vec{x}_i, y_i) \mid 1 \leq i \leq n\}$. A meta-feature f is then defined as a function:

[3]There is actually an error in the preprint of Rivolli et al., 2018 used when writing this paper, namely, that $\mathcal{D}$ is defined as a single dataset rather than as a set of datasets, which leads to repeated use of incorrect function domain notation. This error is corrected in the reproduction here.

$$f : \mathcal{D} \to \mathbb{R}^n$$
$$f : D \mapsto \sigma\left(m\left(D, h_m\right), h_s\right)$$

where $m : \mathcal{D} \to \mathbb{R}^{k'}$ is a characterization measure that generates a vector of one or more elements to describe some meta-feature of the dataset, $\sigma : \mathcal{D} \to \mathbb{R}^k$ is a summarization statistic such as max, min, mean, standard deviation, skewness, and kurtosis, and h_m and h_s are hyperparameters. σ may also be the identity function if m generates a single value, as is the case for some meta-features.

For the remainder of this article, we use the library `pymfe`[4] from Rivolli et al., 2018 to generate the meta-features for the datasets that are used to benchmark Darwin™. Correspondingly, we here summarize some of the common categories of meta-features as distinguished by `pymfe`.

— **General** or **Simple** measures are those that are commonly known and easy to compute, such as the number of attributes that are categorical or numeric in nature and the overall number of data points. We also use this category for some other dataset information that is tracked by Darwin, such as: whether a problem is a classification or regression task, whether a problem involves timeseries (and therefore benefits from recurrent models), and whether a problem has a heavy class imbalance.

— **Statistical** measures are a broad variety of measures from statistics that characterize higher-order properties of data distribution, such as correlation/covariance and metrics of dispersion (e.g. skewness and kurtosis).

— **Information-theoretic** measures are, as the name suggests, from information theory. These measures are based on entropy, and include metrics such as mutual information and noise-to-signal ratio. Generally, they are useful for characterizing how immediately

[4]https://pypi.org/project/pymfe/

predictive attributes are of the labels and each other, which is useful for characterizing redundancy.

— **Model-based** measures are particularly useful for the task of determining aspects of the difficulty a given dataset (Peng et al., 2008). Usually, these boil down to training a decision tree on the dataset and making inferences from the characteristics of the trained tree. For example, if the decision tree has a large number of leaves per class or nodes per attribute, certain aspects of the underlying problem could be considered more difficult.

— **Landmarking** measures are similar to model-based measures in that they involve training a simple, fast model on the dataset. This model can be a decision tree, a naïve Bayes classifier, or something else entirely. Rather than making inferences from properties of the trained model as with model-based methods, the performance of the model on a validation set is taken directly as a characterization measure.

2.2 Meta-Feature Distributions

The set of datasets used to benchmark Darwin is compiled from a variety of sources, including the UCI Machine Learning Repository (Dua and Graff, 2017) and the AutoML Challenge series (Guyon et al., 2019). There are also a number of synthetic datasets. In Figure 1, we break down the general proportions of datasets in the dataset pool. Darwin's dataset pool features a large number of timeseries problems, in accordance with its most frequent use cases.

In Figure 2, we show the distributions of a select few meta-features in Darwin's dataset pool. Mean skewness is chosen for its theoretical significance, while the others are chosen for their relevance later in Section 3.2. A more detailed summary of a larger number of meta-features can be found in Appendix A.

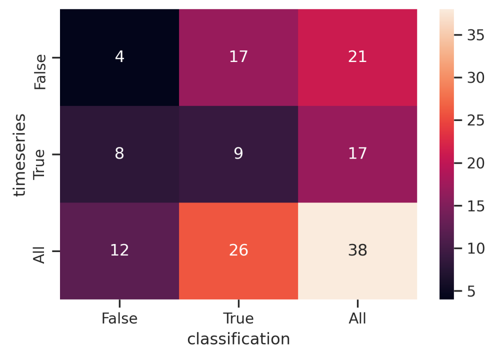


Figure 1 Confusion matrix of classification/regression problems vs recurrent/non-recurrent problems. Generally, timeseries problems tend to be less common than non-timeseries problems, but still very common, and classification problems tend to be much more common than non-classification problems.

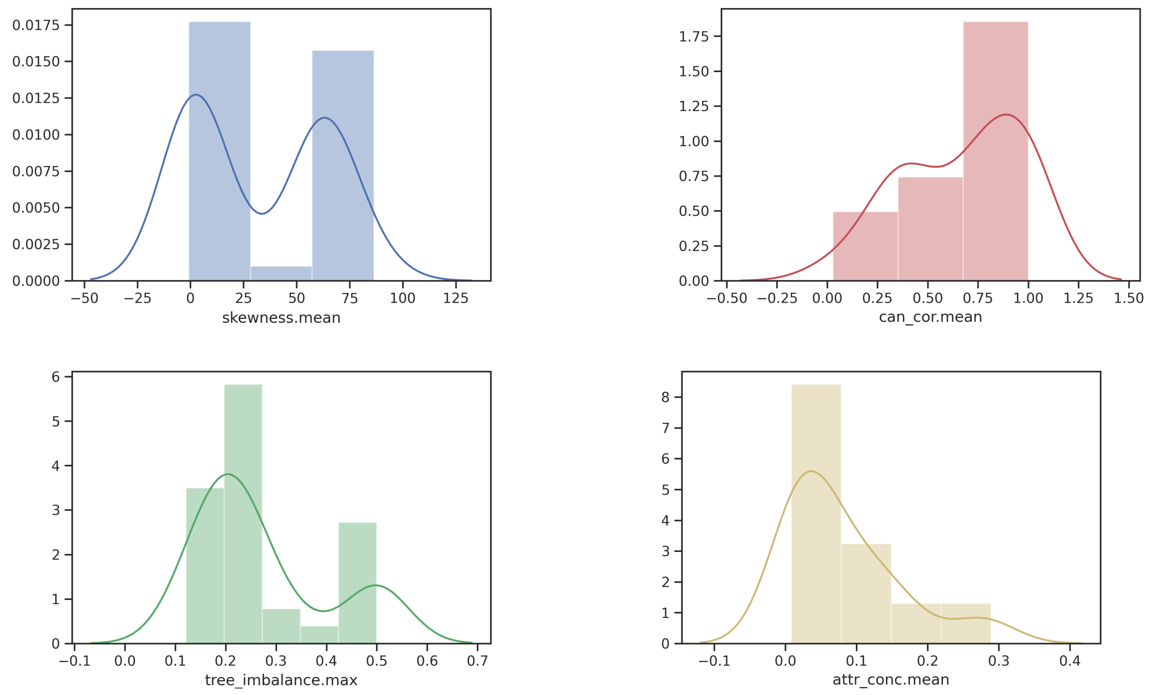


Figure 2 Some specific meta-features and how they are distributed in Darwin's dataset pool. From top left, clockwise: mean skewness, mean canonical correlation between the data points and labels, max tree imbalance (model-based metric), and mean attribute concentration (Goodman-Kruskal τ). For a more detailed explanation of these meta-features, why they were chosen, and what they mean, see Section 3.2, as well as Appendix A in Rivolli et al., 2018.

3 Experiments

3.1 Setup

For all tests, we ran Darwin's main neurovolution algorithm for 20 minutes with stagnation stopping disabled, so that training would be performed for the full allotted timeframe. 20 minutes is a common default in Darwin's documentation and is in practice enough time for LSTM and TCN-based genomes to outperform other types of neural networks for timeseries problems. We ignored ensembling results and any models that were not neural networks, e.g. gradient-boosted decision trees and random forest models, so that we could focus solely on the impact of hyperparameter optimization on the performance of the optimized models on the validation set.

By default, training in Darwin's neuroevolutionary algorithm uses a batch size of 256 and learning rate of 10^{-3}, with the the Adam learning rate optimizer[5] (Kingma and Ba, 2014). After neuroevolution had completed, we ran a hyperparameter optimizer (either random search or Hyperband) for 2, 5, or 10 minutes, along with a control group that did not use an optimizer. These optimizers used the same training and validation set as the neuroevolutionary algorithm.

Testing was performed in two rounds, one with and one without different learning rates as part of the hyperparameter search space. One of the reasons this was done was to see if Hyperband would perform worse in the latter situation by selecting for models with higher learning rates. The Successive Halving algorithm prioritizes models with strong early performance, and models with higher learning rates tend to have better performance in the first few epochs. This, however, can backfire, since those same models may perform comparatively worse once fully optimized. Thus, it would not be unreasonable to assume

[5]To avoid ambiguity with the word "optimizer", we here use "learning rate optimizer" for optimizers such as Adam, Adamax, RMSProp, etc. and "hyperparameter optimizer" for those implemented in Darwin (Hyperband, random search).

that bandit-based algorithms tend to perform worse than other (e.g. Bayesian) methods when the learning rate is part of the search space.

All testing was performed on a single machine running Ubuntu 16.04 with a single NVIDIA GeForce GTX 1080. While Darwin has the ability to train models on a distributed cluster of servers, we did not use that mode in benchmarking.

3.2 Results

General Meta-Feature Correlations

First, we examine the relationship between dataset meta-features and average model performance. Model accuracy on categorical datasets was measured via the F1 score, and model accuracy on regression problems was measured via R^2. While it may seem at first glance that Darwin was generally more likely to have a very high accuracy score on classification problems (Fig. 3), this was primarily due to the large number of synthetic regression problems that exist in Darwin's standard test bench. This becomes clearer when the recurrent nature of datasets is taken into account (Fig. 4). Most of the more difficult problems in Darwin's pool of benchmark datasets tended to be regression problems.

Again, note that this is something of an apples-to-oranges comparison, as F1 and R^2 are different measures.

We then see which dataset meta-features were most predictive of test set accuracy in Table 1. We find that canonical correlation analysis (CCA) is one highly effective way to estimate the difficulty of a dataset, suggesting that easier datasets tend to possess correlations of hidden variables that are linear combinations of the training data, and that neural networks are quick to pick up on these correlations.

We likewise found model-based methods to be highly predictive of dataset difficulty, with more complex models predicting poorer test set accuracy, as expected. Model-based methods generally tended to dominate the list, suggesting that they are particularly effective among

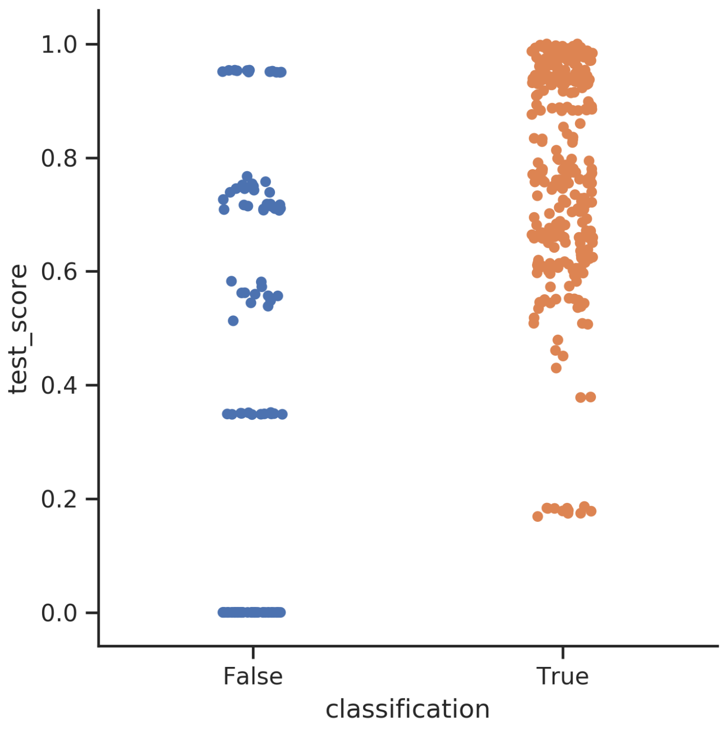


Figure 3 Classification datasets in the pool skewed easier, but this was partially due to the predominance of synthetic recurrent datasets in Darwin's benchmarks.

meta-features for judging the difficulty of a dataset.

Hyperparameter Optimization: Score Improvements

Before assessing what impact dataset meta-features may have on the efficacy of hyperparameter optimization on the performance of our models, it makes sense to first examine how much of an impact, if at all, adding hyperparameter optimization to the machine learning pipeline actually has at all on test set performance. In Table 2, we find that overall, when averaging across the entire dataset pool, there doesn't seem to be a significant improvement to model performance.

That is not to say that no datasets see any improvements. In Table 3, we pick out the five datasets that saw the largest increase in performance with the introduction of hyperparameter optimization. One particular synthetic dataset saw a significant performance increase

Meta-feature	Correlation	Meta-feature	Correlation
can_cor.mean	0.803502	tree_imbalance.max	-0.524518
can_cor.min	0.716742	nodes_per_level.max	-0.556851
ns_ratio	0.683231	nodes_per_level.sd	-0.557699
tree_shape.max	0.679986	leaves_corrob.max	-0.558824
can_cor.max	0.573044	nodes_repeated.min	-0.561584
naive_bayes.sd	0.504130	nodes_per_level.mean	-0.567421
tree_shape.sd	0.477376	tree_imbalance.mean	-0.736365
one_nn.sd	0.458157	leaves_corrob.mean	-0.771872
leaves_per_class.mean	0.453305	tree_imbalance.min	-0.775395
tree_imbalance.sd	0.451577	leaves_corrob.min	-0.776016

Table 1 Meta-features most predictive of Darwin's performance on a given dataset, with correlation coefficients.

Optimization time (min)	score (lr fixed)	score (lr searched)
No optimization	0.721584	0.610847
5	0.728147	0.610278
10	0.726328	0.604280

Table 2 Breakdown of test score given hyperparameter optimization level. In aggregate, improvements seem insignificant.

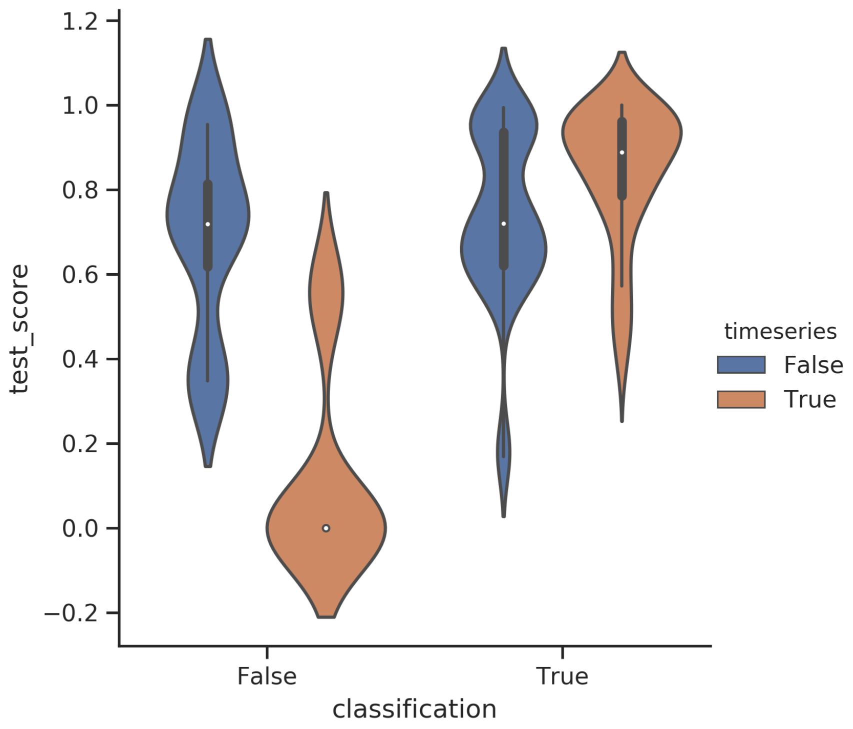


Figure 4 Violin plot of test score of Darwin across all runs. Regression datasets with a timeseries component, such as `UCI_Beijing`, tended to be the most difficult.

with larger batch sizes, and several other large recurrent datasets similarly saw non-negligible improvements.

In Table 4, we see which meta-features are the most correlated with the average improvement in test set accuracy seen for a given dataset once hyperparameter optimization is introduced. Notably, several information-theoretic measures stand out as unusually predictive of the efficacy of hyperparameter optimization - attribute concentration, class concentration, Shannon entropy, and mutual information. In particular, we find that datasets where the categorical features are strongly associated with each other or with the target label (as measured by Goodman-Kruskal τ, as described in Alexandros and Melanie, 2001) tend to see larger improvements when hyperparameter optimization is introduced. That being said, it can be hard to be sure as to what extent these correlations may be spurious.

As an example, upon examination, some meta-features that are seemingly strongly cor-

Dataset	Improvement (5)	Dataset	Improvement (10)
ComplexRecurrent...	0.505774	ComplexRecurrent...	0.409648
UCI_EEG_eye_state_v2	0.064226	UCI_room_occupancy	0.039599
beijing	0.034135	philippine	0.035232
sklearn_boston	0.024234	ComplexRecurrent...	0.026638
SimpleRecurrent...	0.021039	beijing	0.023344

Table 3 Datasets that saw the largest increase in test score performance with hyperparameter optimization. Datasets ending in ... are synthetic.

Meta-feature	Correlation	Meta-feature	Correlation
attr_conc.mean	0.395148	h_mean.max	-0.664021
class_conc.max	0.321885	num_to_cat	-0.493990
attr_ent.sd	0.317389	h_mean.sd	-0.381053
mut_inf.sd	0.316393	g_mean.sd	-0.358495
nodes_repeated.max	0.299559	leaves_per_class.min	-0.315663
joint_ent.sd	0.298292	leaves_per_class.mean	-0.290889
var_importance.sd	0.294308	attr_ent.min	-0.289855
nodes_repeated.mean	0.294278	nr_outliers	-0.257503
nodes_repeated.sd	0.283038	leaves_per_class.max	-0.255589
attr_conc.sd	0.279396	kurtosis.min	-0.229514

Table 4 Meta-features most predictive of impact of hyperparameter optimization on test set performance.

related with test set accuracy improvements, such as the harmonic or geometric mean of the data points in a regression problem, are in fact heavily influenced by a small set of outliers and are not generally correlated across the bulk of datasets. As a result, it is important to take a closer look the relationships between these measures and the datasets before prematurely concluding that they are entirely predictive of the efficacy of hyperparameter optimization. In Figure 5, we illustrate some of the more promising meta-features listed in Table 4 to see how they correlate with score improvements across the distribution of datasets in Darwin's benchmark pool.

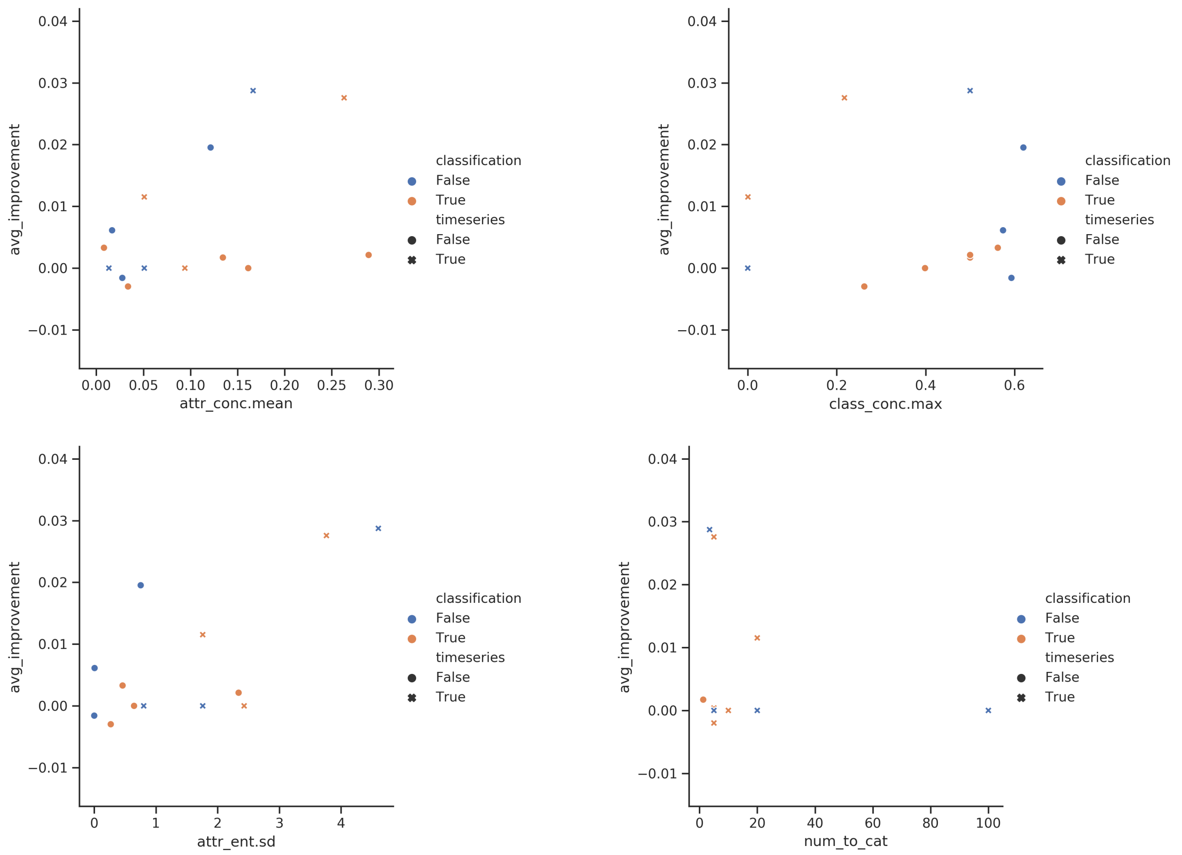
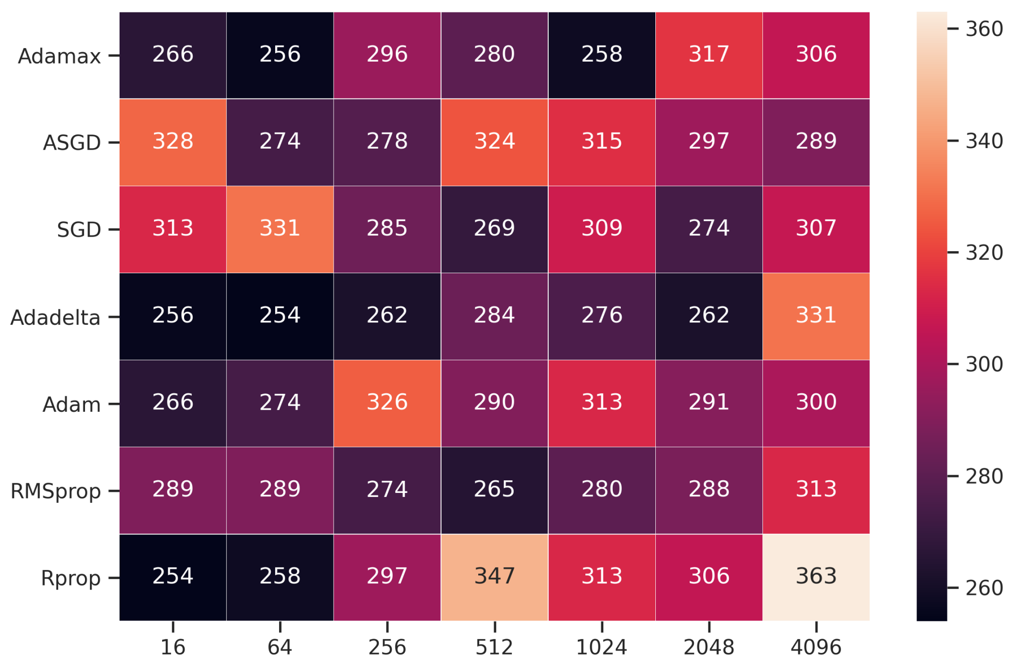


Figure 5 Some specific meta-features and how they correlate with average improvement with hyperparameter optimization.

Hyperparameter Choices

While there are some datasets, such as the UCI Beijing dataset, that see consistent nontrivial score increases with the introduction of hyperparameter optimization, it is also true that most datasets do not seem to benefit greatly. Thus, another reasonable question one may ask is whether any particular hyperparameter configuration is consistently effective, and is thus desirable as a sane default across a wide variety of datasets and model architectures.



	16	64	256	512	1024	2048	4096
Adamax	266	256	296	280	258	317	306
ASGD	328	274	278	324	315	297	289
SGD	313	331	285	269	309	274	307
Adadelta	256	254	262	284	276	262	331
Adam	266	274	326	290	313	291	300
RMSprop	289	289	274	265	280	288	313
Rprop	254	258	297	347	313	306	363

Figure 6 Heatmap of best-performing hyperparameter configurations across dataset pool. Larger batch sizes seemed generally more effective across the board, while most learning rate optimizers performed adequately.

We examine the most commonly-chosen intermediate hyperparameter configurations in Figure 6. Overall, it seems that higher batch sizes tend to be unilaterally more effective, while there is no clear consensus on the best learning rate optimizer. Rprop ekes out a lead as the most popular optimizer, followed by ASGD, SGD, and Adam, in that order, but the differences are largely negligible.

One question that had been raised earlier was whether it made sense to allow a bandit-based algorithm to modify learning rates, given that models trained with higher learning rates could be prioritized by the algorithm in the short run at the expense of future performance. It is definitely true that the average test set accuracy across all runs with all optimizers was only 0.608 with alternative learning rates as part of the search space, whereas it was 0.725 when the learning rate was simply fixed at 10^{-3}. At first glance, therefore, one would suspect that Hyperband performed much worse when searching over learning rates, and therefore brought the average down. However, that would not be quite correct.

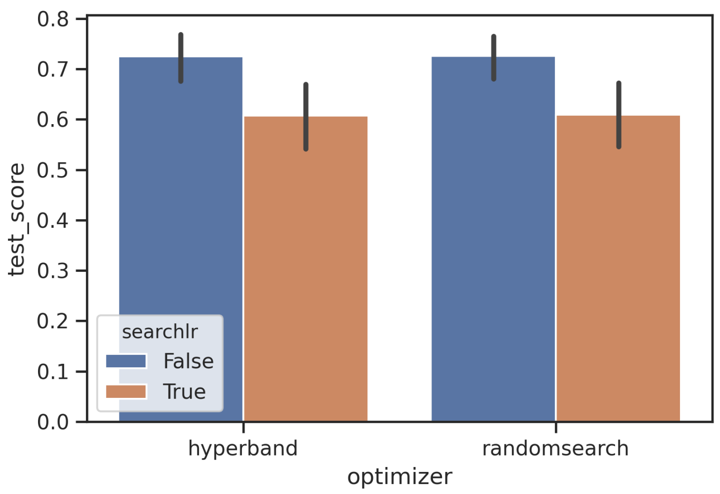


Figure 7 Average test set accuracy with learning rate searched vs not searched. A surprisingly large disparity emerges.

In Figure 7, we find that both Hyperband and random search did significantly worse with the introduction of alternative learning rates into the pool, but otherwise, there was no significant difference between them. (That is not to say that Hyperband could not have had any benefits relative to random search; it may still perform better under very strict time limitations, but we have not tested it rigorously under those conditions.) We hypothesize that significant changes to the default learning rate of 1e-3 are largely unhelpful, and that adding this dimension to the hyperparameter search space tends to make it harder for a hyperparameter search algorithm to find a performant model under a strict time limit as in Darwin. Thus, it is probably better to keep a sane default for learning rate across all datasets.

4 Conclusion

In this article, we attempted to analyze the impact of integrating hyperparameter optimization into a neural architecture search (NAS) pipeline designed to use evolutionary algorithms to build deep neural networks to solve a wide variety of problems. We assessed the efficacy of both Hyperband (Elsken, Metzen, and Hutter, 2018) and random search when run after the main neuro-evolutionary algorithm as an optional tune step with a fixed time budget. We found various relationships of potential interest, such as the unusual effectiveness of model-based methods in estimating dataset difficulty, and the potential effectiveness of information theory-based methods in predicting the efficacy of hyperparameter optimization.

Overall, we found hyperparameter optimization to be useful for some datasets but not generally necessary in the context of Darwin^{TM} given the selection of sane defaults. We found larger batch sizes to be generally more effective across the pool of datasets tested, while neither the choice of learning rate optimizer nor the choice of hyperparameter optimizer tended to have a significant impact on performance under the chosen testing configurations.

5 Future Work

Regarding the implementation of hyperparameter optimization in a neuroevolutionary pipeline such as Darwin^{TM}, there are still several avenues that remain to be explored. Due to implementation limitations, we were unable to examine the effectiveness of embedding hyperparameters into the genome and evolving them alongside the model architecture, but this would be a important avenue for future research. Furthermore, we did not experiment much with searching over learning rate schedulers, parameter decay, warm restarts, dropout, and other general hyperparameters of the supervised learning process. As Darwin continues to develop and support a wider variety of datasets, potentially such as language and vision datasets, it could be beneficial to re-visit these conclusions with a broader set of datasets and hyperparameters in mind. Finally, while we did note some correlations between certain information-theoretic measures and the potential efficacy of hyperparameter optimization in Section 3.2, we do not currently possess a strong theoretical justification for the existence of such correlations, and further research would be required in order to not dismiss these correlations as potentially spurious.

APPENDICES

Appendix A: Extended Meta-Feature Analysis Results

In the interest of reproducibility, we here share more comprehensive information about the distribution of meta-features across the datasets used internally in Darwin^{TM} to benchmark performance. In Table 5, we list some general statistics, aggregated across the subset of the dataset pool where those statistics are applicable. (In other words, if a statistic is only computed for categorical datasets, it will only be aggregated across categorical datasets.) For a detailed glossary of the meaning of each meta-feature, see Appendix A in Rivolli et al., 2018.

Meta-feature	Mean	Standard deviation
`attr_to_inst`	0.035	0.076
`best_node.max`	0.581	0.342
`best_node.mean`	0.564	0.338
`best_node.min`	0.544	0.334
`best_node.sd`	0.012	0.016
`cat_to_num`	0.068	0.141
`cor.max`	0.580	0.440
`cor.mean`	0.077	0.127
`cor.min`	0.003	0.017
`cor.sd`	0.082	0.099
`cov.min`	0.296	1.480
`eigenvalues.min`	0.439	1.444
`elite_nn.max`	0.563	0.345
`elite_nn.mean`	0.537	0.345
`elite_nn.min`	0.504	0.347

Continued on next page

Meta-feature	Mean	Standard deviation
elite_nn.sd	0.018	0.019
freq_class.max	0.478	0.337
freq_class.mean	0.432	0.341
freq_class.min	0.395	0.357
freq_class.sd	0.056	0.129
iq_range.min	0.717	2.117
kurtosis.min	-0.839	1.164
leaves_branch.max	40.618	55.804
leaves_branch.mean	16.799	20.150
leaves_branch.min	3.147	2.105
leaves_corrob.max	0.291	0.402
leaves_corrob.mean	0.208	0.409
leaves_corrob.min	0.206	0.410
linear_discr.max	0.693	0.285
linear_discr.mean	0.662	0.286
linear_discr.min	0.628	0.291
linear_discr.sd	0.020	0.033
mad.min	0.565	1.523
max.min	14.673	82.393
naive_bayes.max	0.606	0.309
naive_bayes.mean	0.581	0.312
naive_bayes.min	0.556	0.312
naive_bayes.sd	0.018	0.024

Continued on next page

Meta-feature	Mean	Standard deviation
nodes_per_inst	0.179	0.225
nr_bin	4.514	22.426
nr_cat	0.737	1.639
nr_cor_attr	0.056	0.120
nr_disc	8.500	18.453
one_nn.max	0.635	0.329
one_nn.mean	0.606	0.330
one_nn.min	0.578	0.332
one_nn.sd	0.018	0.022
random_node.max	0.559	0.346
random_node.mean	0.540	0.338
random_node.min	0.508	0.337
random_node.sd	0.017	0.043
sd.min	0.701	1.724
skewness.mean	32.467	31.707
skewness.min	-18.777	83.697
skewness.sd	11.426	35.199
sparsity.max	0.200	0.351
sparsity.mean	0.034	0.086
sparsity.min	0.002	0.010
sparsity.sd	0.045	0.097
tree_depth.max	40.618	55.804
tree_depth.mean	16.019	19.978

Continued on next page

Meta-feature	Mean	Standard deviation
`tree_depth.min`	0.000	0.000
`tree_imbalance.max`	0.270	0.129
`tree_imbalance.mean`	0.130	0.192
`tree_imbalance.min`	0.103	0.205
`tree_shape.max`	0.224	0.171
`tree_shape.mean`	0.012	0.026
`tree_shape.min`	0.001	0.003
`var.min`	3.204	10.761
`var_importance.max`	0.157	0.223
`var_importance.mean`	0.042	0.061
`var_importance.min`	0.019	0.045
`var_importance.sd`	0.036	0.081
`worst_node.max`	0.538	0.339
`worst_node.mean`	0.516	0.330
`worst_node.min`	0.494	0.323
`worst_node.sd`	0.013	0.024

Table 5 Aggregated meta-feature values across datasets.

References

Alexandros, Kalousis and Hilario Melanie (2001). “Model selection via meta-learning: a comparative study”. In: *International Journal on Artificial Intelligence Tools* 10.04, pp. 525–554.

Bai, Shaojie, J Zico Kolter, and Vladlen Koltun (2018). “An empirical evaluation of generic convolutional and recurrent networks for sequence modeling”. In: *arXiv preprint arXiv:1803.01271*.

Baker, Bowen et al. (2016). “Designing neural network architectures using reinforcement learning”. In: *arXiv preprint arXiv:1611.02167*.

Bensusan, Hilan, Christophe G Giraud-Carrier, and Claire Julia Kennedy (2000). “A Higher-order Approach to Meta-learning.” In: *ILP Work-in-progress reports* 35.

Bergstra, James and Yoshua Bengio (2012). “Random search for hyper-parameter optimization”. In: *Journal of machine learning research* 13.Feb, pp. 281–305.

Bilalli, Besim, Alberto Abelló, and Tomas Aluja-Banet (2017). “On the predictive power of meta-features in OpenML”. In: *International Journal of Applied Mathematics and Computer Science* 27.4, pp. 697–712.

Dua, Dheeru and Casey Graff (2017). *UCI Machine Learning Repository*. URL: http://archive.ics.uci.edu/ml.

Elsken, Thomas, Jan Hendrik Metzen, and Frank Hutter (2018). “Neural architecture search: A survey”. In: *arXiv preprint arXiv:1808.05377*.

Falkner, Stefan, Aaron Klein, and Frank Hutter (2018). “BOHB: Robust and efficient hyper-parameter optimization at scale”. In: *arXiv preprint arXiv:1807.01774*.

Feurer, Matthias et al. (2018). “Practical automated machine learning for the automl challenge 2018”. In: *International Workshop on Automatic Machine Learning at ICML*, pp. 1189–1232.

Guyon, Isabelle et al. (2019). “Analysis of the AutoML Challenge series 2015-2018”. In: *AutoML*. Springer series on Challenges in Machine Learning. URL: https://www.automl.org/wp-content/uploads/2018/09/chapter10-challenge.pdf.

Hutter, Frank, Holger H Hoos, and Kevin Leyton-Brown (2011). “Sequential model-based optimization for general algorithm configuration”. In: *International conference on learning and intelligent optimization*. Springer, pp. 507–523.

Jamieson, Kevin and Ameet Talwalkar (2016). “Non-stochastic best arm identification and hyperparameter optimization”. In: *Artificial Intelligence and Statistics*, pp. 240–248.

Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization*. arXiv: 1412.6980 [cs.LG].

Kotthoff, Lars et al. (2017). “Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA”. In: *The Journal of Machine Learning Research* 18.1, pp. 826–830.

Krizhevsky, Alex, Geoffrey Hinton, et al. (2009). “Learning multiple layers of features from tiny images”. In:

McDonnell, Tyler et al. (2018). “Divide and conquer: neuroevolution for multiclass classification”. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 474–481.

Peng, Yonghong et al. (2008). “Decision Tree-Based Data Characterization for Meta-Learning”. In:

Pham, Hieu et al. (2018). “Efficient neural architecture search via parameter sharing”. In: *arXiv preprint arXiv:1802.03268*.

Real, Esteban, Alok Aggarwal, et al. (2019). “Regularized evolution for image classifier architecture search”. In: *Proceedings of the aaai conference on artificial intelligence*. Vol. 33, pp. 4780–4789.

Real, Esteban, Sherry Moore, et al. (2017). “Large-scale evolution of image classifiers”. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 2902–2911.

Rivolli, Adriano et al. (2018). “Characterizing classification datasets: a study of meta-features for meta-learning”. In: *arXiv preprint arXiv:1808.10406*.

Snoek, Jasper, Hugo Larochelle, and Ryan P Adams (2012). “Practical bayesian optimization of machine learning algorithms”. In: *Advances in neural information processing systems*, pp. 2951–2959.

Thornton, Chris et al. (2013). “Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms”. In: *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 847–855.

Zoph, Barret and Quoc V Le (2016). “Neural architecture search with reinforcement learning”. In: *arXiv preprint arXiv:1611.01578*.