Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search

Arber Zela   zelaa@cs.uni-freiburg.de
Aaron Klein   kleinaa@cs.uni-freiburg.de
Stefan Falkner   sfalkner@cs.uni-freiburg.de
Frank Hutter   fh@cs.uni-freiburg.de
Department of Computer Science, University of Freiburg

Abstract

While existing work on neural architecture search (NAS) tunes hyperparameters in a separate post-processing step, we demonstrate that architectural choices and other hyperparameter settings interact in a way that can render this separation suboptimal. Likewise, we demonstrate that the common practice of using very few epochs during the main NAS and much larger numbers of epochs during a post-processing step is inefficient due to little correlation in the relative rankings for these two training regimes. To combat both of these problems, we propose to use a recent combination of Bayesian optimization and Hyperband for efficient joint neural architecture and hyperparameter search.

Keywords: Neural Architecture Search, Hyperparameter Optimization, Bayesian Optimization, Object Recognition

1. Introduction

Before the rise of deep learning and its success in end-to-end feature learning, manual feature engineering was arguably one of the most important steps in the machine learning workflow, but also very time-consuming and tedious. With an abundance of choices in designing the architecture of deep neural networks, manual feature engineering has nowadays to a certain degree been replaced by manual tuning of architectures. Recent work on *neural architecture search (NAS)* (Baker et al., 2017; Pham et al., 2018; Zoph and Le, 2017; Real et al., 2017; Zoph et al., 2017; Real et al., 2018; Liu et al., 2017; Cai et al., 2018; Elsken et al., 2018) automates this choice of network architecture. On some benchmarks, these NAS methods have indeed led to new state-of-the-art performance (Zoph and Le, 2017; Real et al., 2018), although only at extreme computational costs on the order of 800 GPUs for two weeks.

Many prominent NAS methods (Zoph and Le, 2017; Real et al., 2017; Zoph et al., 2017; Real et al., 2018) follow a two step process, which we argue is inefficient. During their main architecture search phase, they evaluate architectures with a fixed set of hyperparameters and a relatively small number of epochs (e.g., 20), and only after the search has finished, they optimize hyperparameters for the end result and evaluate it with a large number of epochs (e.g., 600). This process is suboptimal for various reasons:

- The resulting approach is not an anytime approach and does not satisfy the requirement of an automated machine learning (AutoML) system (Feurer et al., 2015) to make predictions after a given time budget.

- The sudden jump from a small budget of 20 to a large budget of 600 epochs leads to little correlation between the relative ranks on the small and the large budgets, potentially rendering most of the expensive optimization at the small budget void.

To combat these problems, we propose to use an approach for joint neural architecture and hyperparameter search that is anytime and gradually increases the computational budget for the best fraction of networks at lower compute budgets in order to yield a far more computationally efficient optimization procedure. Specifically, our contributions are as follows:

- We show how to use a recent combination (Falkner et al., 2018) of Bayesian optimization (Shahriari et al., 2016) and Hyperband (Li et al., 2017) to perform efficient joint neural architecture and hyperparameter search.

- We demostrate the weak correlation between performance after short and long training budgets, which potentially affects both architecture and hyperparameter choices when optimized with these two budgets, and show how to avoid this effect by incrementally increasing the budget during the optimization process.

- We show that for a limited training runtime of 3 hours we can achieve competitive performance on CIFAR-10 if we optimize the hyperparameters and architecture jointly.

2. Related Work

Melis et al. (2018) showed that a well-tuned LSTM (Hochreiter and Schmidhuber, 1997) with the right training pipeline was able to outperform a recurrent cell found by neural architecture search (Zoph and Le, 2017) on the Penn Treebank dataset. This underlines the effect hyperparameters can play in practice.

One of the most successful methods to optimize the hyperparameters of deep neural networks is Bayesian optimization (Snoek et al.; Hutter et al., b; Bergstra et al., a). Since each function evaluation requires to train and evaluate a deep neural network, more advanced Bayesian optimization methods try to speed up the optimization process by exploiting available fidelities of the objective function, such as learning curves (Domhan et al.; Klein et al., 2017b; Falkner et al., 2018) or dataset subsets (Klein et al., 2017a). While most Bayesian optimization methods only focus on a few continuous hyperparameters, Bergstra et al. (b) and Mendoza et al. (2016) optimized architecture and hyperparameters jointly and achieved, at this time, state-of-the-art performance for shallow convolutional neural networks and feed forward neural networks, respectively.

There are many related methods for neural architecture search, for example based on reinforcement learning (Zoph and Le, 2017; Zoph et al., 2017) and evolutionary algorithms (Real et al., 2017; Liu et al., 2017; Real et al., 2018). All of these methods focus only on the neural architecture, keeping hyperparameter fixed during the search (and optimizing them in a post-hoc step).

3. Efficient Joint Hyperparameter Optimization and Architecture Search

In this section we discuss how to cast neural architecture search as a hyperparameter optimization problem and tackle it together with the standard hyperparameter optimization problem. We also discuss a method for efficiently searching in this joint space.

3.1. Neural Architecture Search as Hyperparameter Optimization

While the NAS literature casts the architecture search problem as very different from hyperparameter optimization, we argue that most NAS search spaces can be written as hyperparameter optimization search spaces (using the standard concepts of categorical and conditional hyperparameters). Indeed, this fact also enables the use of standard evolutionary algorithms in the literature Real et al. (2017, 2018).

As an example, take the parameterization of a convolutional Normal Cell introduced by Zoph et al. (2017). Each cell receives as inputs two previous hidden states (the feature maps of two cells in previous layers, or the input image directly), and outputs a new hidden state. The Normal Cell is composed of B (by default, $B = 5$) blocks and the $k - th$ block consists of 5 categorical choices: select a first hidden state (out of the cell's 2 inputs and the output hidden states of blocks $1, \ldots, k - 1$); select a second hidden state (out of the same domain as the first choice); select an operation (out of 13 operations including several types of convolutions, pooling operations and the identity; see Zoph et al. (2017) for the full list) to apply to the first hidden state; select an operation to apply to the second hidden state (out of the same 13 operations); and select a method (element-wise addition or concatenation) to combine the outputs of these operations to create a new hidden state (which is added to the existing set of hidden states). In the end, all the unused output hidden states from the cell's B blocks are concatenated together to form the final cell output (there are no free choices in this step). Therefore, in total, this search space is fully specified by $5B$ categorical hyperparameters.

Casting NAS as a hyperparameter optimization problem with categorical hyperparameters immediately suggests the possibility of joint architecture and hyperparameter search by just extending the hyperparameter space for the NAS part by the standard hyperparameters to be tuned. Likewise, it would be possible to sample hyperparameters in an RL approach or optimize them via genetic algorithms alongside the neural architecture.

3.2. Bayesian Optimization Hyperband

To efficiently optimize in the joint space of architectures and hyperparameters, we use BOHB (Falkner et al., 2018), a recent combination of Bayesian optimization (Shahriari et al., 2016) and Hyperband (Li et al., 2017). Due to space constraints, we refer the reader to Falkner et al. (2018) and Li et al. (2017) for full details on these methods and only provide the basics here.

Like Hyperband, BOHB uses evaluations on different budgets to accelerate the optimization by exploiting knowledge gained on cheap, lowfidelity observations. It dynamically allocates more resources to promising configurations by repeatedly invoking the Successive Halving (Jamieson and Talwalkar, 2016) subroutine. Successive Halving evaluates a large number of configurations using a small minimum budget b_{min} and continues to evaluate the best η^{-1} (by default best-performing third) of these with the next budget $\eta \cdot b_{min}$. This is repeated until reaching a maximum budget b_{max}. As an example, consider a budget based on the number of epochs a neural network is trained for; b_{min} could, e.g., be 10, and b_{max} 270.

Like Bayesian optimization, BOHB uses a probabilistic model to sample promising configurations rather than selecting these uniformly at random as done in Hyperband. BOHB

uses multivariate Kernel Density Estimators (KDEs) over the input configuration space to better model interactions between parameters and returns a new sample with the highest expected improvement (EI).

3.3. Joint Architecture and Hyperparameter Search Space

We picked a multiple-branch ResNet architecture as the basis for our search space (see Gastaldi (2017)). The first layer is a 3x3 convolution followed by 3 main blocks, a 8x8 average pooling and a fully connected layer in the end for discriminating between classes. We parametrized the number of filters $Filters_0$ for the first convolution and the number of residual blocks $ResBlocks_j$, branches $ResBranches_j$ and filters (determined by a widening factor $WidenFactor_j$) within each main block j. However, we kept a fixed structure (ReLU-Conv3x3-BN-ReLU-Conv3x3-BN-Mul) for each residual branch. Each sampled architecture configuration Net_i, is thus defined by these 10 architectural choices as:

$$
\begin{aligned}
Net_i &= stack_{j=1}^{3}\{MainBlock_j\}, \\
MainBlock_j &= stack_{k=1}^{ResBlocks_j}\{ResBlock_k\}, \\
ResBlock_k &= add_{l=1}^{ResBranches_j}\{ResBranch_l\}, \\
Filters_j &= round(WidenFactor_j \cdot Filters_{j-1}),
\end{aligned}
\tag{1}
$$

where $stack_{i=1}^{N}\{motif_i\}$ applies motifs sequentially and $add_{i=1}^{N}\{motif_i\}$ adds their outputs element-wise. For example, if $\mathcal{F}_1(\cdot)$ and $\mathcal{F}_2(\cdot)$ denote transformations applied to input x by $ResBlock_1$ and $ResBlock_2$ respectivelly, $stack\{ResBlock_1, ResBlock_2\}$ yields $\mathcal{F}_2(\mathcal{F}_1(x))$ as output. In the case of $ResBranch_1$ and $ResBranch_2$, $add\{ResBranch_1, ResBranch_2\}$ would result in $x + \mathcal{F}_1(x) + \mathcal{F}_2(x)$. One could combine these different operations and motifs recursively in order to search for more complex architectures as the system evolves, but we restrict ourselves to this 10-dimensional architecture space due to a limited compute budget. We leave for future work the design of a more generic search space which does not limit the network information into a fixed length vector.

We optimized this network using the regularization methods of CutOut (DeVries and Taylor, 2017), MixUp (Hongyi Zhang, 2018), Shake-Shake (Gastaldi, 2017), and Shake-Drop (Yamada et al., 2018)[1], as well as the optimization algorithm of stochastic gradient descent with restarts (SGDR; Loshchilov and Hutter (2017)). As hyperparameters, we tuned initial learning rate, batch size, momentum, L_2 regularization, CutOut length, and the α parameter for MixUp, as well as the death rate for ShakeDrop (Yamada et al., 2018). Table 3 in Appendix A summarizes these 10 architectural choices and 7 hyperparameters and provides the specific ranges we used.

4. Experiments

We now empirically study the results of applying BOHB to our joint architecture and hyperparameter search on CIFAR-10. Furthermore, we investigate the characteristics of

---

1. For residual blocks containing $b > 1$ residual branches, Shake-Shake scales the feature maps from branch i by a random factor a_i, such that $\sum_i^b a_i = 1$. Only in the case of $b = 1$ we apply ShakeDrop instead.

our search space accross different budgets, including the importance of architectural choices and hyperparameters on performance.

4.1. NAS Under Training Time Constraints

In our experiments, due to limited computational resources, and in order to explore a training-time-constrained NAS paradigm, we do not allow individual training times as large as those required to obtain the state-of-the-art performance, but limited the training time[2] to a maximum of three hours for each sampled configuration (the minimal budget we consider is 400 seconds). We used the default settings of BOHB, which results in budgets that differ by a multiplicative factor of $\eta = 3$; with a maximum budget of 3h, this yielded budgets of 400s, 1200s, 1h and 3h.

4.2. Results

We ran BOHB for an equivalent of 256 evaluations on the full budget of 3h (i.e., a total of 32 GPU days) and show the results in Figure 1. This provides the best performance found as a function of time, as well as the performance of all trained networks, which shows that our search did not only cover the good regions of the space, but also explored sufficiently. Surprisingly, the final performance reached within this 3h budget was as low as 3.18% test error; as Table 2 shows, this is lower than what can be obtained with several different standard architectures that were

Figure 1: Validation error of all configurations evaluated on the different budgets during the whole optimization procedure. The best performing configuration (incumbent) as a function of time is visualized by the black line.

also part of BOHB's search space, trained for 3h using the same optimization pipeline and hyperparameters. Training details for the other architectures can be found in Appendix B. This demonstrates the benefit of optimizing both architecture and hyperparameters.

From Table 2, we also note that, amongst the three different sizes of multi-branch residual architecture regularized with Shake-Shake, within the 3h time budget the medium-sized architecture (26 2x64d) performed best, not the largest one (26 2x96d). This is in stark contrast to comparisons with a large budget of 1800 epochs, for which the largest architecture performs much better (Gastaldi, 2017; DeVries and Taylor, 2017). We believe this effect is due to the strong stochasticity introduced by Shake-Shake regularization, which prevents obtaining very good results quickly. Indeed, in a follow-up experiment with a WRN architecture (Zagoruyko and Komodakis, 2016) outside of BOHB's design space, which was incompatible with Shake-Shake regularization, we obtained even slightly better results for a time budget of 3h (but worse results for large budgets). This demonstrates an interaction effect between the efficacy of the regularization method used and the available time budget.
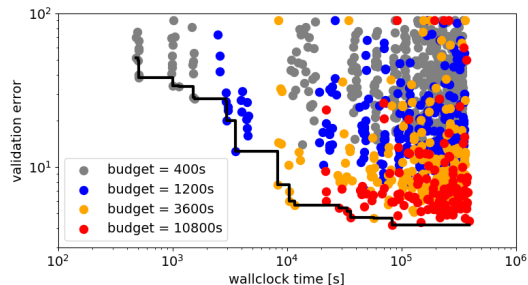
---

2. We only measured the actual time spend in forward and backward passes during training and exclude any preprocessing and validation overhead.

Table 1: Spearman rank correlation coefficients of the validation errors between different budgets. The correlation is high between every budget and the next larger one, but degrades quickly beyond that.

	1200s	1h	3h
400s	0.87	0.31	0.05
1200s		0.88	0.64
1h			0.86

Table 2: Comparison of test performance between manually-constructed architectures and the network configuration found by BOHB when trained for a 3 hour budget. All networks used the same optimization pipeline and hyperparameters.

Network	Params	Test error (%)
ResNet-18	11.2M	3.34 ± 0.11
Shake-Shake 26 2x32d	2.9M	3.91 ± 0.09
Shake-Shake 26 2x64d	11.7M	3.38 ± 0.07
Shake-Shake 26 2x96d	26.2M	4.22 ± 0.06
Ours	27.6M	$\mathbf{3.18 \pm 0.16}$

4.3. Analysis

We now analyse the characteristics of our search space on the different budgets. By design, BOHB optimizes the validation performance for each budget starting with the smallest one and moving to the next larger one as soon as enough evaluations have finished successfully. Consequently, the error rates for 400s have been extensively optimized, with coverage of the search space gradually decreasing for larger budgets.

First, we studied the rank correlation of the final validation error of all configurations that were trained on any particular pair of budgets.[3] The results, summarized in Table 1, clearly show that the relative performance of two configurations generalizes to *somewhat* longer training times (e.g., correlations of roughly 0.87 for 3-fold increases of training time), but that it quickly degrades with larger differences in budget and almost vanishes when jumping directly from 400s to 3h. This means in particular that the ranking of the top configurations cannot be deduced from the ranking of much shorter runs only, which is a common practice in the current neural architecture search literature.

We now study the characteristics of the search space using functional analysis of variance (fANOVA; Hutter et al. (a)). This method allows us to quantify the importance of architecture choices and hyperparameters based on the whole search space by marginalizing performances over all possible values that other hyperparameters could take based on a model fit on the observations. The importance of a single choice, or a combination of any number of choices, is quantified by the percentage of the performance variation that is explained by only this choice/these choices. To improve the quality of the analysis, we focus on the results after 400s up to the 1h budget, since enough evaluations have finished on these allowing us to draw meaningful conclusions for these budgets.

We can observe different behavior for the various choices: The learning rate remains important throughout all budgets and its value in the best found configuration only changes slightly. On the other hand, some architectural parameters that affect the number of network parameters, e.g., the number of residual blocks within a block, can be heavily influenced by a constrained budget; this is, e.g., demonstrated in the plots in the middle of Fig. 2. For the 400s training budget, the networks with less residuals blocks in the first block perform better on average and the importance of this choice is quite high. With more training time, this choice becomes rather unimportant, and good configurations can be found with almost any value. For the CutOut length and the number of residual blocks

---

3. We want to emphasize that most of the evaluations BOHB performed with a 3h budget are for configurations that perform well on smaller budgets, and thus our samples are skewed towards good configurations.
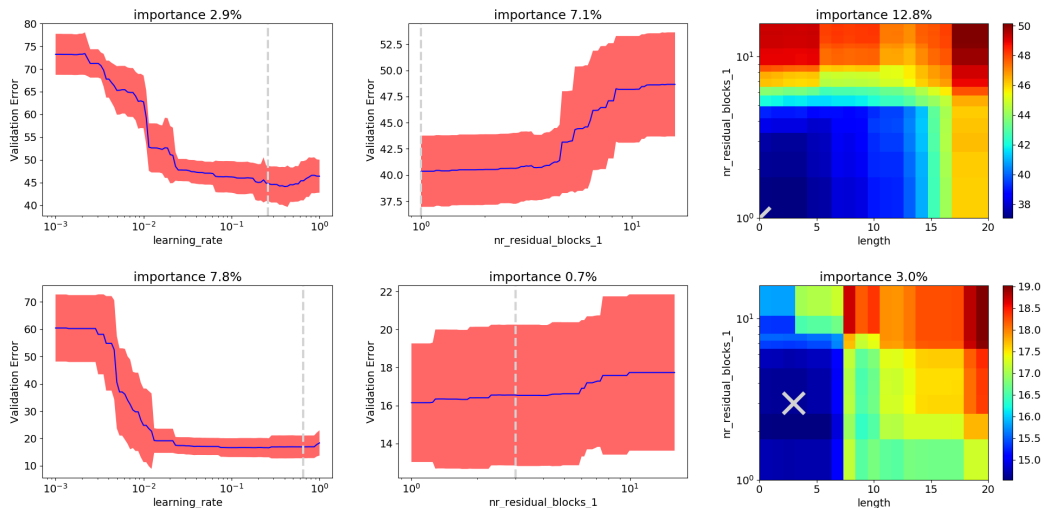
Figure 2: Parameter importance plots for three hyperparameters for training 400s (top row) and 1h (bottom row). The importance indicates the fraction of the variance explained by the individual choice(s). The value of the best found configuration on this budget is indicated by the dashed line/ gray x. While the learning rate remains the same throughout the budgets, other choices are heavily influenced by the constrained training time. For example, the characteristics for the number of residual blocks (middle) and the interaction between the number of residual blocks and the CutOut lengths (right), as well as their importance, change with the budget.

in the first block we find an interesting pattern (see plots on the right of Fig. 2). While the good range for the CutOut length on the 400s budget is quite large (as long as the number of residual blocks is small enough), the picture changes for the larger budget. There, the CutOut length becomes more restricted, and the number of blocks becomes less relevant.

Overall, from this analysis, we conclude that there is a close interaction between good architectural choices and hyperparameters on the one hand, and the runtime budget on the other hand. The common practice of optimizing just on the smallest budget and evaluating on the largest budget would be very wasteful in this case.

5. Conclusions

We showed that it is desirable and feasible to optimize neural architectures and hyperparameters jointly. We also demonstrated the potentially strong effects that short training exerts on both architectural choices and hyperparameters, resulting in a poor correlation between the performance after short and long training periods, and showed how to sidestep this effect with incremental budget increases in the optimization process as implemented by the recent BOHB approach.

Acknowledgements

References

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *ICLR*, 2017.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. pages 2546–2554, a.

J. Bergstra, D. Yamins, and D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. pages 115–123, b.

Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *AAAI*, 2018.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. pages 3460–3468.

T. Elsken, J. H. Metzen, and F. Hutter. Multi-objective architecture search for cnns. April 2018.

S. Falkner, A. Klein, and F. Hutter. Practical hyperparameter optimization for deep learning. *ICLR 2018 Workshop*, 2018.

M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter. Efficient and robust automated machine learning. pages 2962–2970, 2015.

Xavier Gastaldi. Shake-shake regularization. *ICLR 2017 Workshop*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *CVPR*, 2016.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Yann N. Dauphin David Lopez-Paz Hongyi Zhang, Moustapha Cisse. mixup: Beyond empirical risk minimization. *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1Ddp1-Rb.

F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. pages 754–762, a.

F. Hutter, H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. pages 507–523, b.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. 2016.

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian hyperparameter optimization on large datasets. In *Electronic Journal of Statistics*, volume 11, 2017a.

A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with Bayesian neural networks. 2017b.

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

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: Bandit-based configuration evaluation for hyperparameter optimization. 2017.

Chenxi Liu, Barret Zoph, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive Neural Architecture Search. In *arXiv:1712.00559 [cs, stat]*, December 2017.

I. Loshchilov and F. Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR) 2017 Conference Track*, April 2017.

G. Melis, C. Dyer, and P. Blunsom. On the state of the art of evaluation in neural language models. In *International Conference on Learning Representations (ICLR) 2018 Conference Track*, 2018.

H. Mendoza, A. Klein, M. Feurer, J. Springenberg, and F. Hutter. Towards automatically-tuned neural networks. In *ICML 2016 AutoML Workshop*, 2016.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *Arxiv, 1802.03268*, 2018.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V. Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2902–2911, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized Evolution for Image Classifier Architecture Search. In *arXiv:1802.01548 [cs]*, February 2018.

B. Shahriari, K. Swersky, Z. Wang, R. Adams, and N. de Freitas. Taking the human out of the loop: A review of Bayesian optimization. *Proceedings of the IEEE*, 104(1):148–175, 2016.

J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. pages 2960–2968.

Yoshihiro Yamada, Masakazu Iwamura, and Koichi Kise. Shakedrop regularization. *arXiv preprint arXiv:1802.02375*, 2018.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR) 2017 Conference Track*, 2017.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *arXiv:1707.07012 [cs.CV]*, 2017.

Appendix A. Joint Archicture and Hyperparameter Search Space

Table 3 summarizes the 10 architectural choices and 7 hyperparameters we used, along with specific ranges and values of the best performing found configuration on the 3h budget.

Table 3: The configuration space for joint architecture and hyperparameter optimization. Each numbered Residual Block, Residual Branches and Widen Factor correspond to each of the 3 main blocks of the architecture. The last column represents the configuration which performs the best after optimizing the search space with BOHB.

Hyperparameter	Range	Log-transform	Value
Initial Learning Rate	$[0.001, 1.0]$	yes	0.648188
Batch Size	$[32, 128]$	yes	89
L_2 regularization	$[0.00001, 0.001]$	yes	0.000339
Momentum	$[0.001, 0.99]$	no	0.099601
MixUp α	$[0.0, 1.0]$	no	0.492042
CutOut *length*	$[0, 20]$	no	3
ShakeDrop *death rate*	$[0.0, 1.0]$	no	0.038439
$ResBlocks_1$	$[1, 16]$	yes	3
$ResBlocks_2$	$[1, 16]$	yes	4
$ResBlocks_3$	$[1, 16]$	yes	2
$ResBranches_1$	$[1, 5]$	no	1
$ResBranches_2$	$[1, 5]$	no	1
$ResBranches_3$	$[1, 5]$	no	4
$Filters_0$	$[8, 32]$	yes	16
$WidenFactor_1$	$[0.5, 8.0]$	yes	6.241141
$WidenFactor_2$	$[0.5, 8.0]$	yes	1.388867
$WidenFactor_3$	$[0.5, 8.0]$	yes	3.344766

Appendix B. Training details

We use the PreAct ResNet-18 (He et al., 2016) and WideResNet-28-10 (Zagoruyko and Komodakis, 2016) architectures with Shake-Shake regularization (S-S-I; see Gastaldi (2017) for details). All networks were trained on one Nvidia GTX 1080Ti GPU. The parameters of each model are initialized as described by He et al. (2016) and trained using SGD with an initial learning rate of 0.1 and with Nesterov's momentum of 0.9. All the models in 2 are trained for 3h with the initial learning rate annealed using a cosine function with $T_0 = 720s$, $T_{mult} = 2$ (Loshchilov and Hutter, 2017). Furthermore, we also apply L_2 regularization with a factor of 10^{-4} and $5 \cdot 10^{-4}$ to 3-branch and 2-branch networks, respectively.

During optimization, we split the CIFAR-10 (Krizhevsky, 2009) datasets into a 45k data points for training, 5k for validation and a 10k for testing and we normalized the features by the per-channel mean and standard deviation. For the training set we used standard data augmentation scheme used for CIFAR-10 (He et al., 2016), i.e. we first padded each image by 4 pixels, cropped a random 32x32 patch and flipped 50% horizontally. Furthermore, we applied CutOut (DeVries and Taylor, 2017) with a mask length of 16 and MixUp (Hongyi Zhang, 2018) with an α value of 0.2.