AUTOHAS: EFFICIENT HYPERPARAMETER AND ARCHITECTURE SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep learning models often require extensive efforts in optimizing hyperparameters and architectures. Standard hyperparameter optimization methods are expensive because of their multi-trial nature: different configurations are tried separately to find the best. In this paper, we propose AutoHAS, an efficient framework for both hyperparameter and architecture search. AutoHAS generalizes the concept of efficient architecture search, ENAS and DARTS, to hyperparameter search and hence can jointly optimize both in a single training. A key challenge in such generalization is that ENAS and DARTS are designed to optimize discrete architecture choices, whereas hyperparameter choices are often continuous. To tackle this challenge, we discretize the continuous space into a linear combination of multiple categorical basis. Furthermore, we extend the idea of weight sharing and augment it with REINFORCE to reduce its memory cost. In order to decouple the shared network weights and controller optimization, we also propose to create temporary weights for evaluating the sampled hyperparameters and updating the controller. Experimental results show AutoHAS can improve the ImageNet accuracy by up to 0.8% for highly-optimized state-of-the-art ResNet/EfficientNet models, and up to 11% for less-optimized models. Compared to random search and Bayesian search, AutoHAS consistently achieves better accuracy with 10x less computation cost.

1 INTRODUCTION

Deep learning models require intensive efforts in optimizing architectures and hyperparameters. Standard hyperparameter optimization methods, such as grid search, random search (e.g., Bergstra & Bengio (2012)) or Bayesian optimization (e.g., Snoek et al. (2012)), are inefficient because they are multi-trial: different configurations are tried in parallel to find the best configuration. As these methods are expensive, there is a trend towards more efficient, single-trial methods for specific hyperparameters. For example, the learning rate can be optimized with the hypergradient method (Baydin et al., 2018). Similarly, many architecture search methods started out multi-trial (Zoph & Le, 2017; Baker et al., 2017; Real et al., 2019), but more recent proposals are single-trial (Pham et al., 2018; Liu et al., 2019). These efficient methods, however, sacrifice generality: each method only works for one aspect or a subset of the hyperparameters or architectures.

In this paper, we generalize those efficient, single-trial methods to include both hyperparameters and architectures[1]. One important benefit of the generalization is that we can have a general, efficient method for hyperparameter optimization as a special case. Another benefit is that we can now jointly search for both hyperparameters and architectures in a single model. Practically, this means that our method is an improvement over neural architecture search (NAS) because each model can potentially be coupled with its own best hyperparameters, thus achieving comparable or even better performance than existing NAS with *fixed* hyperparameters.

To this end, we propose AutoHAS, an efficient hyperparameter and architecture search framework. It is, to the best of our knowledge, the first method that can efficiently handle architecture space, hyperparameter space, or the joint search space. A challenge here is that architecture choices (e.g. kernel size) are often categorical values whereas hyperparameter choices (e.g. learning rate) are

[1]In this paper, hyperparameters refer all design choices that will affect the training procedure of a model, such as learning rate, weight decay, optimizer, dropout, augmentation policy, etc.

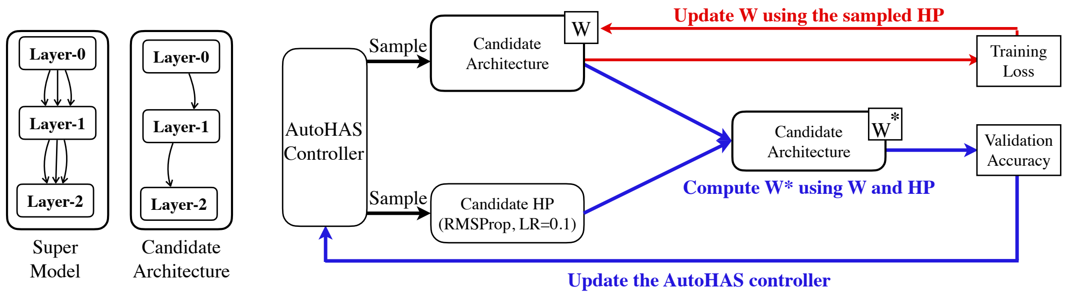

Figure 1: **The overview of AutoHAS method.** LEFT: Each candidate architecture's weights are shared with a super model, where each candidate is a sub model within this super model. RIGHT: During the search, AutoHAS alternates between optimizing the shared weights of super model $\mathcal{W}$ and updating the controller. It also creates temporary weights $\mathcal{W}^*$ by optimizing the sampled candidate architecture using the sampled candidate hyperparameter (HP). This $\mathcal{W}^*$ will be used to compute the validation accuracy as a reward so as to update the AutoHAS controller to select better candidates. Finally, $\mathcal{W}^*$ is discarded after updating the controller so as not to affect the original $\mathcal{W}$.

often continuous values. To address the mixture of categorical and continuous search spaces, we first discretize the continuous hyperparameters into a linear combination of multiple categorical basis. The discretization allows us to unify architecture and hyperparameter choices during search. As explained below, we will use a reinforcement learning (RL) method to search over these discretized choices in Fig. 1. The probability distribution over all candidates is naturally learnt by the RL controller, and it is used as the coefficient in the linear combination to find the best architecture and hyperparameters.

AutoHAS uses the weight sharing technique proposed by (Pham et al., 2018; Liu et al., 2019). The main idea is to train a super model, where each candidate in the architecture space is its sub-model. Using a super model can avoid training millions of candidates from scratch (Liu et al., 2019; Dong & Yang, 2019a; Cai et al., 2019; Pham et al., 2018). AutoHAS extends its scope from architecture search to both architecture and hyperparameter search. We not only share the weights of super model with each architecture but also share this super model across hyperparameters. At each search step, AutoHAS optimizes the sampled sub-model by a combination of the sampled hyperparameter choices, and the shared weights of super model serves as a good initialization for all hyperparameters at the next step of search (see Fig. 1 and Sec. 2). In order to decouple the shared network weights ($\mathcal{W}$ in Fig. 1) and controller optimization, we also propose to create temporary weights ($\mathcal{W}^*$ in Fig. 1) for evaluating the sampled hyperparameters and updating the controller. With weight sharing, AutoHAS reduces the search cost by an order of magnitude than random search and Bayesian search. In experiments, AutoHAS shows non-trivial improvements on **seven** datasets, such as 0.8% accuracy gain on highly-optimized EfficientNet and 11% accuracy gain on less-optimized models.

2 AutoHAS

In this section, we elaborate the design philosophy of AutoHAS. We introduce the background of AutoHAS in Sec. 2.1, how to represent architectures and hyperparameters in a unified way in Sec. 2.2, how to search in Sec. 2.3, and how to derive the final architectures and hyperparameters in Sec. 2.4.

2.1 Preliminaries

AutoHAS should be able to handle the general case of NAS and HPO – jointly find architecture α and hyperparameters h that achieve high performance on the validation set. This objective can be formulated as a bi-level optimization problem:

$$\min_{\alpha,h} \mathcal{L}(\alpha, h, \omega_\alpha^*, \mathbb{D}_{val}) \quad \text{s.t.} \;\; \omega_\alpha^* = f_h(\alpha, \omega_\alpha^0, \mathbb{D}_{train}), \tag{1}$$

where $\mathcal{L}$ is the objective function (e.g., cross-entropy loss) and ω_α^0 is the initial weights of the architecture α. $\mathbb{D}_{train}$ and $\mathbb{D}_{val}$ denote the training data and the validation data, respectively. f_h represents the algorithm with hyperparameters h to obtain the optimal weights ω_α^*, such as using SGD to minimize the training loss. In that case, $\omega_\alpha^* = f_h(\alpha, \omega_\alpha^0, \mathbb{D}_{train}) = \arg\min_{\omega_\alpha} \mathcal{L}(\alpha, h, \omega_\alpha^0, \mathbb{D}_{train})$. We can also use HyperNetwork (Ha et al., 2017) to generate weights ω_α^*.

AutoHAS generalizes both NAS and HPO by introducing a broader search space. On one-hand, NAS is a special case of HAS, where the inner optimization $f_h(\alpha, \omega_\alpha^0, \mathbb{D}_{train})$ uses fixed α and h to optimize $\min_\omega \mathcal{L}(\alpha, h, \omega, \mathbb{D}_{train})$. On the other, HPO is a special case of HAS, where α is fixed in Eq. (1).

2.2 Unified Representation of the Search Space in AutoHAS

The search space in AutoHAS is a Cartesian product of the architecture and hyperparameter candidates. To search over this mixed search space, we need a unified representation of different searchable components, i.e., architectures, learning rates, optimizers, etc.

Architectures Search Space. We use the simplest case as an example. First of all, let the set of predefined candidate operations (e.g., 3x3 convolution, pooling, etc.) be $\mathcal{O} = \{O_1, O_2, ..., O_n\}$, where the cardinality of $\mathcal{O}$ is n for each layer in the architecture. Suppose an architecture is constructed by stacking multiple layers, each layer takes a tensor F as input and output $\pi(F)$, which serves as the next layer's input. $\pi \in \mathcal{O}$ denotes the operation at a layer and might be different at different layers. Then a candidate architecture α is essentially the sequence for all layers $\{\pi\}$. Further, a layer can be represented as a linear combination of the operations in $\mathcal{O}$ as follows:

$$\pi(F) = \sum\nolimits_{i=1}^{n} C_i^\alpha \, O_i(F) \qquad \text{s.t.} \;\; \sum\nolimits_{i=1}^{n} C_i^\alpha = 1, C_i^\alpha \in \{0, 1\} \tag{2}$$

where C_i^α (the i-th element of the vector C^α) is the coefficient of operation O_i for a layer.

Hyperparameter Search Space. Now we can define the hyperparameter search space in a similar way. The major difference is that we have to consider both categorical and continuous cases:

$$h = \sum\nolimits_{i=1}^{m} C_i^h \, \mathcal{B}_i \qquad \text{s.t.} \;\; \sum\nolimits_{i=1}^{m} C_i^h = 1, \; C_i^h \in \begin{cases} [0, 1], & \text{if continuous} \\ \{0, 1\}, & \text{if categorical} \end{cases} \tag{3}$$

where $\mathcal{B}$ is a predefined set of hyperparameter basis with the cardinality of m and $\mathcal{B}_i$ is the i-th basis in $\mathcal{B}$. C_i^h (the i-th element of the vector C^h) is the coefficient of hyperparameter basis $\mathcal{B}_i$. If we have a continuous hyperparameter, we have to discretize it into a linear combination of basis and unify both categorical and continuous. For example, for weight decay, $\mathcal{B}$ could be {1e-1, 1e-2, 1e-3}, and therefore, all possible weight decay values can be represented as a linear combination over $\mathcal{B}$. For categorical hyperparameters, taking the optimizer as an example, $\mathcal{B}$ could be {Adam, SGD, RMSProp}. In this case, a constraint on C_i^h is applied: $C_i^h \in \{0, 1\}$ as in Eq. (3).

2.3 AutoHAS: Efficient hyperparameter and Architecture Search

Given the discretizing strategy in Sec. 2.2, each candidate in the search space can be represented by the value of $\mathbb{C} = \{C^\alpha \text{ for all layers}, C^h \text{ for all types of hyperparameter}\}$, which represents the coefficients for all architecture and hyperparameter choices. As a result, AutoHAS converts the searching problem to obtaining the coefficients $\mathbb{C}$.

Algorithm 1 AutoHAS Training

```
Input: Randomly initialize $\mathcal{W}$ and $\mathbb{P}$
Input: Split the available data into two disjoint sets: $\mathbb{D}_{train}$ and $\mathbb{D}_{val}$
1: while not converged do
2:     Sample ($\alpha, h \in \mathcal{B}$) from the controller
3:     Estimate the quality $Q(\alpha, h)$ as the reward to update controller by REINFORCE
4:     $\mathcal{W}_\alpha \leftarrow f_h(\alpha, \mathcal{W}_\alpha, \mathbb{D}_{train})$
5: end while
6: Derive the final architecture $\alpha$ and hyperparameters $h$ by $\mathbb{P}$ (Sec. 2.4)
```

AutoHAS applies reinforcement learning together with weight sharing to search over the discretized space. During search, we learn a controller to sample the candidate architecture and hyperparameters from the discretized space. In AutoHAS, this controller is parameterized by a collection of independent multinomial variables $\mathbb{P} = \{P^\alpha \text{ for all layers}, P^h \text{ for all types of hyperparameter}\}$[2], which draws the probability distribution of the discretized space. AutoHAS also leverages a super model to share weights $\mathcal{W}$ among all candidate architectures, where each candidate is a sub-model in this super model (Pham et al., 2018; Liu et al., 2019). Furthermore, AutoHAS extends the scope of weight sharing from architecture to hyperparameters, where $\mathcal{W}$ also serves as the initialization for the algorithm f_h.

We describe AutoHAS in Algorithm 1. It alternates between learning the shared weights $\mathcal{W}$ and learning the controller using REINFORCE (Williams, 1992). Specifically, at each iteration, the

[2] P^α and P^h are n- and m-dimensional vectors, respectively. Each vector sums up to 1.

controller samples a candidate — an architecture α and basis hyperparameter $h \in \mathcal{B}$. We estimate its quality $Q(\alpha, h)$ by utilizing the temporary weights to maintain the value of $f_h(\alpha, \mathcal{W}_\alpha, \mathbb{D}_{train})$. Using temporary weights, we can measure the validation accuracy of α and h as $Q(\alpha, h)$, and in the same time, avoid the side effect of $f_h(\alpha, \mathcal{W}_\alpha, \mathbb{D}_{train})$ w.r.t. $\mathcal{W}_\alpha$. In our experiment, $f_h(\alpha, \mathcal{W}_\alpha, \mathbb{D}_{train})$ is approximately calculated as one-step gradient descent using the algorithm determined by h. This estimated quality is used as a reward to update the controller's parameters $\mathbb{P}$ via REINFORCE. Then, we optimize the shared weights $\mathcal{W}$, where the weights corresponding to the sampled architecture $\mathcal{W}_\alpha$ is updated as $f_h(\alpha, \mathcal{W}_\alpha, \mathbb{D}_{train})$.

2.4 Deriving hyperparameters and Architecture

After AutoHAS optimizes $\mathbb{P} = \{P^\alpha, P^h\}$ via Algorithm 1, we can derive the coefficient $\mathbb{C}$ as follows:

$$C^\alpha = \text{one\_hot}(\arg\max_i P^\alpha), \tag{4}$$

$$C^h = \begin{cases} P^h & \text{if continuous} \\ \text{one\_hot}(\arg\max_i P^h) & \text{if categorical} \end{cases}, \tag{5}$$

Together with Eq. (2) and Eq. (3), we can derive the final architecture α and hyperparameters h. Intuitively speaking, the selected operation in the final architecture has the highest probability over other candidates, and so does the categorical hyperparameter. For the continuous hyperparameter, the final one is the weighted sum of the learnt probability P^h with its basis $\mathcal{B}$.

To evaluate whether the AutoHAS-discovered α and h is good or not, we will use h to re-train α on the whole training set and report its performance on the test sets.

2.5 Discussion

Generalizability. AutoHAS can be applied to searching for architecture only, hyperparameter only, or both. Moreover, unlike previous HPO methods that require the hyperparamter optimization formulation f_h to be differentiable for computing gradient w.r.t. the hyperparameters, AutoHAS treats the inner optimization f_h as a block-box, and thus is applicable for both differentiable and non-differentiable hyperparmaters.

Phase-wise AutoHAS. It is challenging to search over the large joint HAS space. Since the sampled architecture and hyperparameters change at every iteration, the gradients w.r.t. the shared weights in super model might dramatically change. Consequently, the shared weights can not be trained well and insufficiently indicative of the RL reward. To alleviate this problem, we propose an alternative, i.e., Phase-wise AutoHAS, which split the whole search procedure into two (or multiple) phases. In the first phase, it will use Algorithm 1 to search for the choices of some components and keep other components fixed as the default value. In the second phase, it will re-use the discovered components in the first phase and search for others. We found this Phase-wise AutoHAS works better than (single-phase) AutoHAS in most cases, at the cost of doubling computational resources. More empirical analysis can be found in Sec. 3.3.

Why do we need temporary weights? There is an interaction between architecture optimization and hyperparameter optimization in AutoHAS. If we implement f_h in a straightforward solution, it will overwrite the original weights $\mathcal{W}$ when we compute f_h. Consequently, the updating of $\mathcal{W}$ in the red branch in Fig. 1 becomes unsafe. Here, we utilize the temporary weights $\mathcal{W}^*$ to maintain the value of f_h. This strategy allows us to decouple the training of shared weights and the update of the AutoHAS controller, and thus effectively optimize over the hyperparameter space.

3 Experiments

We evaluate AutoHAS on seven datasets, including two large-scale datasets, ImageNet (Deng et al., 2009) and Places365 (Zhou et al., 2017). We will briefly introduce the experimental settings in Sec. 3.1. We compare AutoHAS with other SOTA methods/models in Sec. 3.2. Lastly, we ablatively study AutoHAS in Sec. 3.3.

3.1 Experimental Settings

Datasets. We leverage seven datasets to comprehensively evaluate our AutoHAS. Their details (Deng et al., 2009; Zhou et al., 2017; Xiao et al., 2016; Krizhevsky & Hinton, 2009; Krause et al., 2013; Nilsback & Zisserman, 2010) are described in Table 1.

Table 1: **Benchmark datasets** – ImageNet and Places365 are two commonly used large-scale datasets for image classification, while the other five are small-scaled datasets.

Name	#Classes	#Train Data	#Eval Data	Hold-out $\mathbb{D}_{train}$	Hold-out $\mathbb{D}_{val}$
ImageNet	1000	1.28M	50K	1.23M	50K
Places365	365	1.8M	50K	1.69M	112K
CIFAR-10	10	50K	10K	45K	5K
CIFAR-100	100	50K	10K	45K	5K
Stanford Cars	196	8144	8041	6494	1650
Oxford Flower	102	2040	6149	1020	1020
SUN-397	397	19850	19850	15880	3970

Searching settings. We call the hyperparameters that control the behavior of AutoHAS as meta hyperparameters – the optimizer and learning rate for RL controller, the momentum ratio for RL baseline, and the warm-up ratio. Warm-upping the REINFORCE algorithm indicates that we do not update the parameters of the controller at the beginning. In addition, when the search space includes architecture choices, we also uses the warm-up technique described in Bender et al. (2020). For these meta hyperparameters, we use Adam, momentum as 0.95, warm-up ratio as 0.3. The meta learning rate is selected from $\{0.01, 0.02, 0.05, 0.1\}$ according to the validation performance. When the architecture choices are in the search space, we will use the absolute reward function (Bender et al., 2020) to constrain the FLOPs of the searched model to be the same as the baseline model. For experiments on ImageNet and Places365, we use the batch size of 4096, search for 100 epochs, and use 4×4 Cloud TPU V3 chips. For experiments on other datasets, we use the batch size of 512, search for 15K steps, and use 2×2 Cloud TPU V3 chips.

Training settings. Once we complete the searching procedure, we re-train the model using the AutoHAS-discovered hyperparameter and architecture. For the components that are not searched for, we keep it the same as the baseline models. For each experiment, we run three times and report the mean (and variance) of the accuracy.

3.2 Comparison With HPO and NAS

AutoHAS shows better performance than other HPO methods. We choose MobileNet-V2 as the baseline model. We search for the mixup ratio from [0, 0.2] and drop-path ratio from [0, 0.5] for each MBConv layer. We use the training schedule in (Bender et al., 2020). Results compared with four representative HPO methods are shown in Fig. 2. Multi-trial search methods, Random Search (Bergstra & Bengio, 2012) or Bayesian optimization (Golovin et al., 2017), must train and evaluate many candidates, and thus are inefficient. Even using $10\times$ more time, they still cannot match the accuracy of AutoHAS. HGD (Baydin et al., 2018) can only search for the learning rate and the searched learning rate is much worse than the baseline. IFT (Lorraine et al., 2020) is an efficient gradient-based HPO method. With the same search space, AutoHAS gets higher accuracy than IFT.

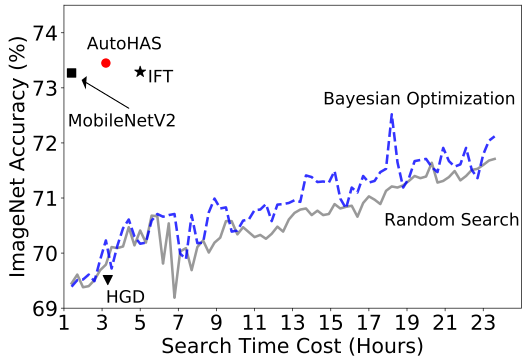

Figure 2: Comparison between AutoHAS and previous HPO methods on ImageNet. AutoHAS achieves better accuracy than HGD, and uses much less search time cost than others.

AutoHAS is feasible for jointly searching hyperparameter and architecture. As a proof of concept for the joint search, we follow MNasNet (Tan et al., 2019) and ProxylessNAS (Cai et al.,

2019) to design a architecture search space (i.e., kernel size {3x3, 5x5} and expansion ratio {3, 6} on top of MobileNetV2), and a joint search space with additional hyperparmater search options (i.e., mixup and dropout ratio). We then compare AutoHAS performance on these two search spaces. With architecture-only search, AutoHAS achieves comparable results (e.g., 74% accuracy @ 300M flops) as MnasNet/ProxylessNAS, but with the joint search, AutoHAS can further improve accuracy by 0.2% with the same FLOPs, suggesting the potential benefit of jointly optimizing architectures and hyperparameters. Notebly, NAS methods are infeasible to optimze the hyperparameters.

Table 2: AutoHAS improves ResNet-50 and EfficientNet-B0 on ImageNet – For each training, we repeat the training three times and the variance is less than 0.16.

Model	Method	#Params	#FLOPs	Top-1 Accuracy
ResNet-50 (He et al., 2016)	Human	25.6 M	4110 M	77.20
	AutoHAS	25.6 M	4110 M	**77.83** (**+0.63**)
EfficientNet-B0 (Tan & Le, 2019)	NAS	5.3 M	398 M	77.15
	AutoHAS	5.2 M	418 M	**77.92** (**+0.77**)

AutoHAS improves SoTA ImageNet models. To investigate the effect of AutoHAS over the state-of-the-art models. We apply AutoHAS to two strong baselines. Firstly, we choose ResNet-50. The baseline strategy is to train it by 200 epochs, start the learning rate at 1.6 and decay it by 0.1 for every $\frac{1}{3}$ of the whole training procedure, use EMA with the decay rate of 0.9999, and apply SGD with the momentum of 0.9. This can provide higher accuracy than the original paper. For reference, the reported top-1 accuracy is 76.15% for ResNet-50 in TorchVision, whereas our baseline is 77.2% accuracy. Since previous methods usually do not tune the architecture of ResNet-50, we only use AutoHAS to search for its hyperparameters including learning rate and mixup ratio for data augmentation. From Table 2, AutoHAS improves this strong baseline by 0.63%.

Secondly, we choose a NAS-searched model, EfficientNet-B0. The baseline strategy is to train it by 600 epochs and use the same learning rate schedule as in the original paper. As EfficientNet-B0 already tunes the kernel size and expansion ratio, we choose a different architecture space. Specifically, in each MBConv layer, we search for the number of groups for all the 1-by-1 convolution layer, the number of depth-wise convolution layer, whether to use a residual branch or not. In terms of the hyperparameter space, we search for the per-layer drop-connect ratio, mixup ratio, and the learning rate. We use phase-wise AutoHAS to first search for the architecture and then for the hyperparameters. From Table 2, we improves the strong EfficientNet-B0 baseline by 0.77% ImageNet top-1 accuracy.

AutoHAS improves SoTA Places36 models. Beside ImageNet, we have also evaluated AutoHAS on another popular dataset: Places365 (Zhou et al., 2017). Similarly, we apply AutoHAS to EfficientNet-B0 to search for better architectures and hyperparameters on this dataset. Fig. 3 shows the results: Although EfficientNet-B0 is a strong baseline with significantly better parameter-accuracy trade-offs than other models, AutoHAS can still further improve its accuracy 1% and obtain a new state-of-the-art accuracy on Places365. Note that `B0` and `B0 + AutoHAS` only uses single crop evaluation, while other models use 10 crops.

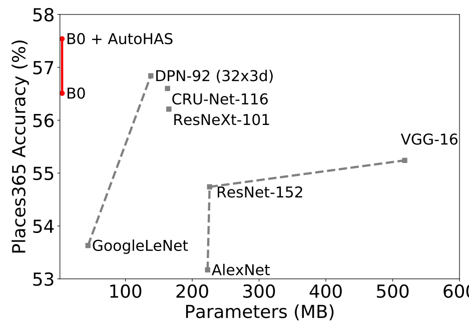

Figure 3: AutoHAS improves accuracy by 1% for EfficientNet-B0 on Places365.

3.3 Ablation Studies

Why choose RL instead of a differentiable strategy? Differentiable search methods have been extensively studied for its simplicity in many previous literature (Liu et al., 2019; Dong & Yang, 2019a; Wan et al., 2020; Xie et al., 2019), but these methods usually require much higher memory cost in order to train the entire super model. In our AutoHAS framework, we employ a simple reinforcement learning algorithm – REINFORCE Williams (1992) – to optimize the controller: instead of training the whole super model, we only train a subset of the super model and therefore significantly reduce the training memory cost. Notably, the REINFORCE could also be simply replaced by a differentiable-based algorithm with the supervision of validation loss. We investigate the difference between differentiable and REINFORCE search in Table 3. We use a small variant of MobileNetV2 with depth multiplier 0.3 as our baseline model (in order to fit our device memory constraint for the differentiable approach), and then apply them to the same search space. Not surprisingly, differentiable search requires much higher memory cost (6.1x more than baseline) as it needs to maintain the feature or gradient tensors for all the super model, whereas our REINFORMCE-based AutoHAS is much more memory efficient: reducing the memory cost by 70% than the differentiable approach. Empirically, we observe they achieve similar accuracy gains in this case, but AutoHAS enables us to search for much larger models such as EfficientNet-B0 and ResNet-50 as shown in Table 2.

Table 3: Differentiable Search vs. AutoHAS REINFORCE Search – Both are applied to the same baseline model with the same hyperparamter and architecture search space. Baseline model has no search cost, but we list its standalone training cost as a reference. Compared to the differentiable search, our AutoHAS achieves slightly better accuracy with much less search memory cost.

	#Params (M)	#FLOPs (M)	Accuracy (%)	Search Cost Memory(GB)	Search Cost Time(Hour)
Baseline model	1.5	35.9	50.96	(1.0)	(1.4)
Differentiable	1.5	36.1	52.17	6.1	2.9
AutoHAS(REINFORCE)	1.5	36.3	53.01	**1.8**	**1.7**

AutoHAS on different search spaces and datasets. To evaluate the generalization ability, we have evaluated AutoHAS in different hyperparameter and architecture spaces for five more datasets. For simplicity, we choose the standard MobileNetV2 as our baseline model. Table 4 shows the results. We observe: (1) The accuracy gains for many of these datasets are much larger than ImageNet/Places365, possible because the hyperparameter and architecture of the baseline are not heavily optimized on these scenarios, leaving us a larger headroom for performance optimization. In particular, AutoHAS achieves up to 11% accuracy gain on Flower dataset, suggesting that AutoHAS could be more useful for less optimized or new model/dataset scenarios. (2) Joint search and phase-wise search have similar performance, possibly due to the difficulty of navigating through a large and complex search space and the interactions between different hyperparamters. Suppose phase-wise search has two phases with search space size O(m) and O(n), then its total search space size is O(m + n), but its corresponding joint search space size would be much larger O(m * n), making the joint search problem much more difficult. While this paper mainly focuses on unifying the architecture and hyperparameter search, it is still an open challenge how to navigate through the very large joint search space while still obtaining the optimal solution, which would be our future work.

Table 4: AutoHAS Accuracy for Different Search Space on five Datasets – Weight decay and MixUp are for hyperparameters, and Arch is for architectures. `joint` indicates the joint search; `phase` indicates the phase-wise search. Each experiment is repeated three times and the average accuracy is reported (standard deviation is about 0.2%).

	Image Classification Top-1 Accuracy (%) CIFAR-10	CIFAR-100	Stanford Cars	Oxford Flower	SUN-397
Baseline	94.1	76.3	83.8	74.0	46.3
WeightDecay	95.0	77.8	89.0	84.4	49.1
MixUp	94.1	77.0	85.2	79.6	47.4
Arch	94.5	76.8	84.1	76.4	46.3
MixUp + Arch (joint)	94.4	77.4	84.8	78.2	47.3
MixUp + Arch (phase)	94.4	77.6	85.5	79.6	48.3
WeightDecay + MixUp (joint)	**95.0** (+0.9)	**78.4** (+2.1)	89.9	84.4	50.5
WeightDecay + MixUp (phase)	94.9	78.2	**90.5** (+6.8)	**85.4** (+11.4)	**50.8** (+4.5)

4 Related Works

Neural Architecture Search (NAS). Since the seminal works (Baker et al., 2017; Zoph & Le, 2017) show promising improvements over manually designed architectures, more efforts have been devoted to NAS. The accuracy of NAS models has been improved by carefully designed search space (Zoph et al., 2018), better search method (Real et al., 2019), or compound scaling (Tan & Le, 2019). The model size and latency have been reduced by Pareto optimization (Tan et al., 2019; Wu et al., 2019; Cai et al., 2019; 2020) and enlarged search space of neural size (Cai et al., 2020; Dong & Yang, 2019b). The efficiency of NAS algorithms has been improved by weight sharing (Pham et al., 2018), differentiable optimization (Liu et al., 2019), or stochastic sampling (Dong & Yang, 2019a; Xie et al., 2019). As these NAS methods use fixed hyperparamters during search, we have empirically observed that they often lead to sub-optimal results, because different architectures tend to favor their own hyperparameters. In addition, even if the manual optimization of architecture design is avoided by NAS, they still need to tune the hyperparameters after a good architecture is discovered.

Hyperparameter optimization (HPO). Black-box and multi-fidelity HPO methods have a long standing history (Bergstra & Bengio, 2012; Hutter, 2009; Hutter et al., 2011; 2019; Kohavi & John, 1995; Hutter et al., 2019). Black-box methods, e.g., grid search and random search (Bergstra & Bengio, 2012), regard the evaluation function as a black-box. They sample some hyperparameters and evaluate them one by one to find the best. Bayesian methods can make the sampling procedure in random search more efficient (Jones et al., 1998; Shahriari et al., 2015; Snoek et al., 2015). They employ a surrogate model and an acquisition function to decide which candidate to evaluate next (Thornton et al., 2013). Multi-fidelity optimization methods accelerate the above methods by evaluating on a proxy task, e.g., using less training epochs or a subset of data (Domhan et al., 2015; Jaderberg et al., 2017; Kohavi & John, 1995; Li et al., 2017). These HPO methods are computationally expensive to search for deep learning models (Krizhevsky et al., 2012).

Recently, gradient-based HPO methods have shown better efficiency (Baydin et al., 2018; Lorraine et al., 2020), by computing the gradient with respect to the hyperparameters. For example, Maclaurin et al. (2015) calculate the extract gradients w.r.t. hyperparameters. Pedregosa (2016) leverages the implicit function theorem to calculate approximate hypergradient. Following that, different approximation methods have been proposed (Lorraine et al., 2020; Pedregosa, 2016; Shaban et al., 2019). Despite of their efficiency, they can only be applied to differentiable hyperparameters such as weight decay, but not non-differentiable hyperparameters, such as learning rate (Lorraine et al., 2020) or optimizer (Shaban et al., 2019). Our AutoHAS is not only as efficient as gradient-based HPO methods but also applicable to both differentiable and non-differentiable hyperparameters. Moreover, we show significant improvements on state-of-the-art models with large-scale datasets, which supplements the lack of strong empirical evidence in previous HPO methods.

Hyperparameter and Architecture Search. Few approaches have been developed for the joint searching of hyperparameter and architecture (Klein & Hutter, 2019; Zela et al., 2018). However, they focus on small datasets and small search spaces. These methods are more computationally expensive than AutoHAS. Concurrent to our AutoHAS, FBNet-V3 (Dai et al., 2020) learns an acquisition function to predict the performance for the pair of hyperparameter and architecture. They require to evaluate thousands of pairs to optimize this function and thus costs much more computational resources than ours.

5 Conclusion

In this paper, we proposed an automated and unified framework AutoHAS, which can efficiently search for both hyperparameters and architectures. AutoHAS provides a novel perspective of AutoML algorithms by generalizing the weight sharing technique from architectures to hyperparameters. Specifically, AutoHAS first unifies the representation of both continuous and categorical choices by the discretizing strategy. Then AutoHAS leverages the weight sharing technique to train a single super model for different hyperparameter and architecture candidates. In parallel, AutoHAS introduces REINFORCE to learn a controller that can sample good hyperparameter and architecture candidates. Experimentally, AutoHAS significantly improves the baseline models on *seven* datasets. For the highly-optimized ResNet/EfficientNet, AutoHAS improves ImageNet top-1 accuracy by 0.8%; for other less-optimized scenarios (e.g., Oxford Flower), it improves the accuracy by 11.4%.

REFERENCES

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017.

Atılım Güneş Baydin, Robert Cornish, David Martínez Rubio, Mark Schmidt, and Frank Wood. Online learning rate adaptation with hypergradient descent. In *International Conference on Learning Representations (ICLR)*, 2018.

Gabriel Bender, Hanxiao Liu, Bo Chen, Grace Chu, Shuyang Cheng, Pieter-Jan Kindermans, and Quoc V Le. Can weight sharing outperform random architecture search? an investigation with tunas. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14323–14332, 2020.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research (JMLR)*, 13(Feb):281–305, 2012.

Han Cai, Ligeng Zhu, and Song Han. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations (ICLR)*, 2019.

Han Cai, Chuang Gan, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR)*, 2020.

Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, et al. FBNetV3: Joint architecture-recipe search using neural acquisition function. *arXiv preprint arXiv:2006.02049*, 2020.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 248–255, 2009.

Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 3460–3468, 2015.

Xuanyi Dong and Yi Yang. Searching for a robust neural architecture in four gpu hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1761–1770, 2019a.

Xuanyi Dong and Yi Yang. Network pruning via transformable architecture search. In *The Conference on Neural Information Processing Systems (NeurIPS)*, pp. 760–771, 2019b.

Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google vizier: A service for black-box optimization. In *The SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017.

David Ha, Andrew Dai, and Quoc V Le. HyperNetworks. In *International Conference on Learning Representations (ICLR)*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

Frank Hutter. *Automated configuration of algorithms for solving hard computational problems*. PhD thesis, University of British Columbia, 2009.

Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pp. 507–523, 2011.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning*. Springer, 2019.

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, et al. Population based training of neural networks. *arXiv preprint arXiv:1711.09846*, 2017.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.

Aaron Klein and Frank Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019.

Ron Kohavi and George H John. Automatic parameter selection by minimizing estimated error. In *Machine Learning Proceedings*, pp. 304–312, 1995.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *International IEEE Workshop on 3D Representation and Recognition (3dRR)*, 2013.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *The Conference on Neural Information Processing Systems (NeurIPS)*, pp. 1097–1105, 2012.

Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The Journal of Machine Learning Research (JMLR)*, 18(1):6765–6816, 2017.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *The International Conference on Machine Learning (ICML)*, pp. 2113–2122, 2015.

Maria-Elena Nilsback and Andrew Zisserman. Delving deeper into the whorl of flower segmentation. *Image and Vision Computing*, 28(6):1049–1062, 2010.

Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *The International Conference on Machine Learning (ICML)*, pp. 737–746, 2016.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. In *The International Conference on Machine Learning (ICML)*, pp. 4092–4101, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, pp. 4780–4789, 2019.

Amirreza Shaban, Ching-An Cheng, Nathan Hatch, and Byron Boots. Truncated back-propagation for bilevel optimization. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 1723–1732, 2019.

Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, 104(1): 148–175, 2015.

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

Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable bayesian optimization using deep neural networks. In *The International Conference on Machine Learning (ICML)*, pp. 2171–2180, 2015.

Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *The International Conference on Machine Learning (ICML)*, pp. 6105–6114, 2019.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. MNasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2820–2828, 2019.

Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *The SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855, 2013.

Alvin Wan, Xiaoliang Dai, Peizhao Zhang, Zijian He, Yuandong Tian, Saining Xie, Bichen Wu, Matthew Yu, Tao Xu, Kan Chen, et al. FBNetV2: Differentiable neural architecture search for spatial and channel dimensions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12965–12974, 2020.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10734–10742, 2019.

Jianxiong Xiao, Krista A Ehinger, James Hays, Antonio Torralba, and Aude Oliva. Sun database: Exploring a large collection of scene categories. *International Journal of Computer Vision (IJCV)*, 119(1):3–22, 2016.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2019.

Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. In *The International Conference on Machine Learning (ICML) Workshop*, 2018.

Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 40(6):1452–1464, 2017.

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

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8697–8710, 2018.