DHA: End-to-End Joint Optimization of Data Augmentation Policy, Hyper-parameter and Architecture

Kaichen Zhou  
University of Oxford

Lanqing Hong†  
Huawei Noah’s Ark Lab

Shoukang Hu  
The Chinese University of Hong Kong

Fengwei Zhou  
Huawei Noah’s Ark Lab

Binxin Ru  
SailYond Technology & Research Institute of Tsinghua University in Shenzhen

Jiashi Feng  
National University of Singapore

Zhenguo Li  
Huawei Noah’s Ark Lab

Reviewed on OpenReview: https://openreview.net/forum?id=MHOAEiTlen

Abstract

Automated machine learning (AutoML) usually involves several crucial components, such as Data Augmentation (DA) policy, Hyper-Parameter Optimization (HPO), and Neural Architecture Search (NAS). Although many strategies have been developed for automating these components in separation, joint optimization of these components remains challenging due to the largely increased search dimension and the variant input types of each component. In parallel to this, the common practice of searching for the optimal architecture first and then retraining it before deployment in NAS often suffers from low performance correlation between the searching and retraining stages. An end-to-end solution that integrates the AutoML components and returns a ready-to-use model at the end of the search is desirable. In view of these, we propose DHA, which achieves joint optimization of Data augmentation policy, Hyper-parameter and Architecture. Specifically, end-to-end NAS is achieved in a differentiable manner by optimizing a compressed lower-dimensional feature space, while DA policy and HPO are regarded as dynamic schedulers, which adapt themselves to the update of network parameters and network architecture at the same time. Experiments show that DHA achieves state-of-the-art (SOTA) results on various datasets and search spaces. To the best of our knowledge, we are the first to efficiently and jointly optimize DA policy, NAS, and HPO in an end-to-end manner without retraining.

1 Introduction

While deep learning has achieved remarkable progress in various tasks such as computer vision and natural language processing, the design and training of a well-performing deep neural architecture for a specific task usually requires tremendous human involvement (He et al. [2016] Sandler et al. 2018). To alleviate such
burden on human users, AutoML algorithms have been proposed in recent years to automate the pipeline of designing and training a model, such as automated Data Augmentation (DA), Hyper-Parameter Optimization (HPO), and Neural Architecture Search (NAS) (Cubuk et al., 2018; Mittal et al., 2020; Chen et al., 2019). All of these AutoML components are normally processed independently and the naive solution of applying them sequentially in separate stages, not only suffers from low efficiency but also leads to sub-optimal results (Dai et al., 2020; Dong et al., 2020). Indeed, how to achieve full-pipeline “from data to model” automation efficiently and effectively is still a challenging and open problem.

One of the main difficulties lies in understanding how to automatically combine the different AutoML components (e.g., NAS and HPO) appropriately without human expertise. FBNetV3 (Dai et al., 2020) and AutoHAS (Dong et al., 2020) investigated the joint optimization of NAS and HPO, while (Kashima et al., 2020) focused on the joint optimization of neural architectures and data augmentation policies. The joint optimization of NAS and quantization policy were also investigated in APQ (Wang et al., 2020). Clear benefits can be seen in the above works when optimizing two AutoML components together, which motivates the further investigation of “from data to model” automation. However, with the increasing number of AutoML components, the search space complexity is increased by several orders of magnitudes and it is challenging to operate in such a large search space. In addition, how these AutoML components affect each other when optimized together is still unclear. Thus, further investigation is needed to open the black box of optimizing different AutoML components jointly.

Another main challenge of achieving the automated pipeline “from data to model” is understanding how to perform end-to-end searching and training of models without the need of parameter retraining. Current approaches, even those considering only one AutoML component such as NAS algorithms, usually require two stages, one for searching and one for retraining (Liu et al., 2019; Xie et al., 2019). Similarly, automatic DA methods such as FastAA (Lim et al., 2019) also need to retrain the model parameters once the DA policies have been searched. In these cases, whether the searched architectures or DA policies would perform well after retraining is questionable, due to the inevitable difference of training setup between the searching and retraining stages (Yang et al., 2019). To improve the performance correlation between searching and retraining stages, DSNAS (Hu et al., 2020) developed a differentiable NAS method to provide direct NAS without parameter retraining. OnlineAugment (Tang et al., 2020) and OnlineHPO (Im et al., 2021) design direct DA or HPO policy, respectively, without model retraining.

Targeting the challenging task-specific end-to-end AutoML, we propose DHA, a differentiable joint optimization solution for efficient end-to-end AutoML components, including DA, HPO and NAS. Specifically, the DA and HPO are regarded as dynamic schedulers, which adapt themselves to the update of network parameters and network architecture. At the same time, the end-to-end NAS optimization is realized in a differentiable manner with the help of sparse coding method, which means that instead of performing our search in a high-dimensional network architecture space, we optimize a compressed lower-dimensional feature space. With this differentiable manner, DHA can effectively deal with the huge search space and the high optimization complexity caused by the joint optimization problem. To summarize, our main contributions are as follows:

- We propose an AutoML method, DHA, for the concurrent optimization of DA, HPO, and NAS. To the best of our knowledge, we are the first to efficiently and jointly realize DA, HPO, and NAS in an end-to-end manner without retraining.

- Experiments show that DHA achieves a state-of-the-art (SOTA) accuracy on ImageNet with both cell-based and Mobilenet-like architecture search space. DHA also provides SOTA results on various datasets, including CIFAR10, CIFAR100, SPORT8, MIT67, FLOWERS102 and ImageNet with relatively low computational cost, showing the effectiveness and efficiency of joint optimization (see Fig. 2).

- Through extensive experiments, we demonstrate the advantages of doing joint-training over optimizing each AutoML component in sequence. Besides, higher model performance and a smoother loss landscape are achieved by our proposed DHA method.
2 Related Works

**Data augmentation.** Learning data augmentation policies for a target dataset automatically has become a trend, considering the difficulty to elaborately design augmentation methods for various datasets (Cubuk et al., 2018; Zhang et al., 2020; Lin et al., 2019). Specifically, AutoAugment (Cubuk et al., 2018) and Adversarial AutoAugment (Zhang et al., 2020) adopt reinforcement learning to train a controller to generate policies, while OHL-Auto-Aug (Lin et al., 2019) formulates augmentation policy as a probability distribution and adopts REINFORCE (Williams, 1992) to optimize the distribution parameters along with network training. PBA (Ho et al., 2019) and FAA (Lin et al., 2019) use population-based training method and Bayesian optimization respectively to reduce the computing cost of learning policies. Cubuk et al. (2020) argues that the search space of policies used by these works can be reduced greatly and simple grid search can achieve competitive performance. They also point out that the optimal data augmentation policy depends on the model size, which indicates that fixing an augmentation policy when searching for neural architectures may lead to sub-optimal solutions.

**Hyper-parameter optimization.** Hyper-parameters also play an important role in the training paradigm of deep neural network models. Various black-box optimization approaches have been developed to address hyper-parameter tuning tasks involving multiple tasks (Mittal et al., 2020; Perrone et al., 2018) or mixed variable types (Ru et al., 2020). Meanwhile, techniques like multi-fidelity evaluations (Kandasamy et al., 2017; Wu et al., 2019), parallel computation (González et al., 2016; Kathuria et al., 2016; Alvi et al., 2019), and transfer learning (Swersky et al., 2013; Min et al., 2020) are also employed to further enhance the query efficiency of the hyper-parameter optimization. In addition, several works (Bengio, 2000; Lorraine et al., 2020; MacKay et al., 2019; Shaban et al., 2019; Maclaurin et al., 2015) have proposed to use gradient-based bi-level optimization to tune large number of hyper-parameters during model training. Although many HPO strategies have been adopted in NAS strategies, methods that jointly optimize both architectures and hyper-parameters are rarely seen except the ones discussed below.

**Neural architecture search.** NAS has attracted growing attention over the recent years and provided architectures with better performance over those designed by human experts (Pham et al., 2018; Real et al., 2018).
The rich collection of NAS literature can be divided into two categories: query-based methods and gradient-based ones. The former includes powerful optimization strategies such as reinforcement learning (Zoph & Le, 2017; Pham et al., 2018), Bayesian optimization (Kandasamy et al., 2018; Ru et al., 2020b) and evolutionary algorithms (Elsken et al., 2019; Lu et al., 2019). The latter enables the use of gradients in updating both architecture parameters and network weights, which significantly reducing the computation costs of NAS via weight sharing (Liu et al., 2019; Chen et al., 2019; Xie et al., 2019; Hu et al., 2020). To reduce searching cost, most NAS methods search architectures in a low-fidelity set-up (e.g., fewer training epochs, smaller architectures) and retrain the optimal architecture using the full set-up before deployment. This separation of search and evaluation is sub-optimal (Hu et al., 2020), which motivates the development of end-to-end NAS strategies (Xie et al., 2019; Hu et al., 2020) that return read-to-deploy networks at the end of the search. Our work also proposes an end-to-end solution.

Joint optimization of AutoML components. Conventional neural architecture search methods perform a search over a fixed set of architecture candidates and then adopt a separate set of hyper-parameters when retraining the best architecture derived from the architecture search phase. Such search protocol may lead to sub-optimal results (Zela et al., 2018; Dong et al., 2020) as it neglects the influence of training hyper-parameters on architecture performance and ignores superior architectures under alternative hyper-parameter values (Dai et al., 2020). Given this, several works have been proposed to jointly optimize architecture structure and training hyper-parameters (Dai et al., 2020; Wang et al., 2020; Dong et al., 2020; Zela et al., 2018) introduces the use of multi-fidelity Bayesian optimization to search over both the architecture structure and training hyper-parameters. Dai et al. (2020) trains an accuracy predictor to estimate the network performance based on both the architecture and training hyper-parameters and then uses an evolutionary algorithm to perform the search. Both these methods are query-based and require a relatively large number of architecture and hyper-parameter evaluations to fine-tune predictors or obtain good recommendations. To improve the joint optimization efficiency, AutoHAS (Dong et al., 2020) introduces a differentiable approach in conjunction with weight sharing for the joint optimization task, which empirically demonstrates that such a differentiable one-shot approach achieves superior efficiency over query-based methods. In addition to jointly optimizing neural architectures and hyper-parameters, the other line of research focuses on the joint optimization of neural architectures and data augmentation hyper-parameters (Kashima et al., 2020). Our proposed method differs from the above works in mainly two aspects: first, our method is more efficient than AutoHAS since our method has no need to update the whole network and only needs to update the sampled sub-network at each optimization step. Second, the joint optimization scope is further extended from NAS and training hyper-parameters to include DA, HPO and NAS.

3 Methodology

Consider a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, where $N$ is the size of this dataset, and $y_i$ is the label of the input sample $x_i$. We aim to train a neural network $f(\cdot)$, which can achieve the best accuracy on the test dataset $D^{test}$. Multiple AutoML components are considered, including DA, HPO, and NAS. Let $\tau$, $\eta$, $\alpha$, and $\theta$ represent the data augmentation parameters, the hyper-parameters, the architecture parameters, and the objective neural network parameters, respectively. This problem can be formulated as

$$\argmin_{\tau, \eta, \alpha, \theta} \mathcal{L}(\tau, \eta, \alpha, \theta; D)$$

s.t. $c_i(\alpha) \leq C_i, i = 1, ..., \gamma$, \hspace{1cm} (1)

where $\mathcal{L}(\cdot)$ represents the loss function, $D$ denotes the input data, $c_i(\cdot)$ refers to the resource cost (e.g., storage or computational cost) of the current architecture $\alpha$, which is restricted by the $i$-th resource constraints $C_i$, and $\gamma$ denotes the total number of resource constraints. Considering the huge search space, it is challenging to achieve the joint optimization of $\tau$, $\eta$, $\alpha$, and $\theta$ within one-stage without parameter retraining. In this work, we propose to use the differentiable method to provide a computationally efficient solution. See Fig. 1 for an illustration.
3.1 Data augmentation parameters

For every mini-batch of training data $B^{tr} = \{(x_k, y_k)\}_{k=1}^{n^{tr}}$ with batch size $n^{tr}$, we conduct data augmentation to increase the diversity of the training data. We consider $K$ data augmentation operations, and each training sample is augmented by a transformation consisting of two successive operations (Cubuk et al., 2018; Lim et al., 2019). Each operation is associated with a magnitude that is uniformly sampled from $\{-\mathcal{K}, 0, \mathcal{K}\}$, where $\mathcal{K}$ is a constant. Given a sampled architecture, the loss function for each augmented sample is denoted by $L^{tr}$ via reinforcement learning (Zhang et al., 2020) or training an extra teacher model to generate additional data. Each operation is associated with a magnitude that is uniformly sampled from $\{-\mathcal{K}, 0, \mathcal{K}\}$. For every mini-batch of training data $B^{tr}$, we directly sample a subset from training set to update the unweighted training loss, which can be written as:

$$L^{tr}(\alpha^t; T_k(x_k)) = \frac{1}{n^{tr}} \sum_{k=1}^{n^{tr}} L^{tr}(f(\alpha^t; T_k(x_k))).$$

With this loss function, DHA intends to increase the sampling probability of those transformations that can generate samples with high training loss. By sampling such transformations, DHA can pay more attention to more aggressive DA strategies and increase model robustness against difficult samples (Zhang et al., 2020). However, blindly increasing the difficulty of samples may cause the augment ambiguity phenomenon (Wei et al., 2020): augmented images may be far away from the majority of clean images, which could cause the under-fitting of model and deteriorate the learning process. Hence, besides optimizing the probability matrix of DA strategies, we randomly sample the magnitude of each chosen strategy from an uniform distribution, which can prevent learning heavy DA strategies: augmenting samples with large magnitude strategies. Moreover, instead of training a controller to generate adversarial augmentation policies via reinforcement learning (Zhang et al., 2020) or training an extra teacher model to generate additional labels for augmented samples (Wei et al., 2020), we search for the probability distribution of augmentation transformations directly via gradient-based optimization. In this way, the optimization of data augmentation is very efficient and hardly increases the computing cost.

3.2 Hyper-parameters

As shown in Fig. 1, given the batch of augmented training data $\{((T_k(x_k), y_k))\}_{k=1}^{n^{tr}}$ and the sampled child network, we need to optimize the differentiable hyper-parameters $\eta$, such as learning rate and L2 regularization. At the training stage, we alternatively update $\theta$ and $\eta$. In $t$-th iteration, we can update $\theta^t$ based on the gradient of the unweighted training loss $L^{tr}(f(\alpha^t, \theta^t; B^{tr}))$, which can be written as:

$$\theta^{t+1} = OP(\theta^t, \eta^t, \nabla_{\theta} L^{tr}(f(\alpha^t, \theta^t; B^{tr}))),$$

where $OP(\cdot)$ is the optimizer. To update the hyper-parameters $\eta$, we regard $\theta^{t+1}$ as a function of $\eta$ and compute the training loss $L^{tr}(f(\alpha^t, \theta^{t+1}(\eta^t); B^{tr}))$ with network parameters $\theta^{t+1}(\eta^t)$ on a mini-batch of training data $B^{tr}$. Then, $\eta^t$ is updated with $\nabla_{\eta} L^{tr}(f(\alpha^t, \theta^{t+1}(\eta^t); B^{tr}))$ by gradient descent:

$$\eta^{t+1} = \eta^t - \beta \nabla_{\eta} L^{tr}(f(\alpha^t, \theta^{t+1}(\eta^t); B^{tr})), \quad (4)$$

where $\beta$ is a learning rate. Even $\theta^t$ can also be deployed to $\theta^{t-1}$ whose calculation also involves $\eta^{t-1}$, we take an approximation method in Eqn. (4) and regard $\theta^t$ here as a variable independent of $\eta^{t-1}$. Instead of splitting an extra validation set for HPO, we directly sample a subset from training set to update $\eta$, which could ensure that the whole training set is used in updating $\tau$, $\eta$, $\alpha$, and $\theta$. This avoids the final learned weight decay coefficient to be zero as non-zero weight decay coefficient can help avoid the model to overfit to the training data.

3.3 Architecture parameters

With the augmented data in previous section, we achieve the optimization of the architecture parameter $\alpha$ through end-to-end NAS, motivated by SNAS (Xie et al., 2019), DNAS (Hu et al., 2020) and ISTA-
NAS (Yang et al., 2020). Following Liu et al. (2019), we denote the each space as a single directed acyclic graph (DAG), where the probability matrix $\alpha$ consists of vector $\alpha_{i,j}^T = [\alpha_{i,j}^1, ..., \alpha_{i,j}^r, ..., \alpha_{i,j}^k]$ and $\alpha_{i,j}^r$ represents the probability of choosing $r$th operation associated with the edge $(i, j)$. Instead of directly optimizing $\alpha \in \mathbb{R}^n$, we adopt ISTA-NAS to optimize its compressed representation $b \in \mathbb{R}^m$ where $m << n$, which can be written as:

$$\begin{equation}
b = A\alpha + \epsilon, \quad (5)
\end{equation}$$

where $\epsilon \in \mathbb{R}^m$ represents the noise and $A \in \mathbb{R}^{m \times n}$ is the measurement matrix which is randomly initialized. Eqn. (5) is solved through using LASSO loss function (Tibshirani, 1994) and the $\alpha$ is optimized by using iterative shrinkage thresholding algorithm (Daubechies et al., 2004), which can be written as:

$$\begin{equation}
\alpha^{t+1} = \eta_{\lambda/L}(\alpha^t - \frac{1}{L}A^T(A\alpha - b)), t = 0, 1, ..., \quad (6)
\end{equation}$$

where $L$ represents the LASSO formulation which can be written as $\min_{\alpha} \frac{1}{2}||A\alpha - b||_2^2 + \lambda||\alpha||_1$; the $\lambda$ represents the regularization parameters and the $\eta_{\lambda/L}$ is the shrinkage operator as defined in (Beck & Teboulle, 2009). Thus we have:

$$\begin{equation}
\alpha^o_j = (b_j^T A_j - [\alpha_j(b_j)]^T E_j) o_j, \quad (7)
\end{equation}$$

where $o_j$ refers to all possible operations connected to node $j$ and $E_j = A_j^T A_j - I$. With this relaxation, $b$ can be optimized through calculating the gradient concerning training loss. The main reason for using this optimization algorithm is that it can optimize the high-dimensional architecture parameters through optimizing low-dimensional embeddings, which can largely decrease the optimization difficulty and increase the optimization efficiency. Moreover, this algorithm also adopt the weights sharing in the optimization process which can be readily combined with our proposed data augmentation and hyper-parameter optimization method.

3.4 Joint-optimization

Based on the above analysis of each AutoML module, DHA realizes end-to-end joint optimization of automated data augmentation parameters $\tau$, hyper-parameters $\eta$, and architecture parameters $\alpha$. The DHA algorithm is summarized in Algorithm 1. One-level optimization is applied to $\tau$ and $\alpha$ as in Line 6 and Line 8, while bi-level optimization is applied to $\eta$ as in Line 9.

One thing worth mentioning is that different optimizers are adopted for different parameters. There are two main reasons behind this choice. Firstly, $\tau, \eta, \alpha$ and $\theta$ work differently in the optimization process, e.g., $\tau$ controls the transformation strategy for the training set while $\alpha$ is related to the architecture selection. Besides $\theta$, other parameters could not directly be optimized through the gradient descent on training set. These differences cause the different optimization methods for $\tau, \alpha, \eta$ and $\theta$. Secondly, $\tau, \alpha, \eta$ and $\theta$ have
different dimensions and different scales, which makes the joint-optimization with a uniform optimizer extremely impracticable. This is why current works concerning DA (Ho et al., 2019), NAS (Yao et al., 2020; Nekrasov et al., 2019) and HPO (Falkner et al., 2018) always adopt different optimizer for network’s weights and hyper-parameters they aim to optimize. During the training process, DA and HPO adapt the online optimization strategy. The DA strategy and HPO settings are evolving with the weights of the parameters.

Moreover, the main reason that DHA could optimize large-scale search space in an effectively manner, is that DHA delicately adopt weight-sharing in the joint-optimization for different parameters. Instead of only optimizing a sub-network with a DA strategy and a hyper-parameter setting to check the performance of certain setting, we realize the joint-optimization with the help of a super-net network, a DA probability matrix and continuous hyper-parameter setting. In that way, DHA can make use of previous trained parameter weights to check the performance setting, which largely decreases the computational request.

4 Experiments

In this section, we empirically compare DHA against existing AutoML algorithms on various datasets. With extensive experiments, we demonstrate the benefits of joint-optimization over sequential-optimization in terms of generalization performance and computational efficiency.

4.1 Experiment setting

Datasets. Following Ru et al. (2020b), we conducted experiments on various datasets, including CIFAR10 and CIFAR100 for the object classification task (Krizhevsky et al., 2009), SPORT8 for the action classification task (Li & Fei-Fei, 2007), MIT67 for the scene classification task (Quattoni & Torralba, 2009), FLOWERS102 for the fine object classification task (Nilsback & Zisserman, 2008), and ImageNet (Russakovsky et al., 2015) for the large-scale classification task. The accuracy is calculated on the test set.

Search space. (1) Automated DA. Following Ho et al. (2019), we consider \( K = 14 \) different operations for data augmentation, such as AutoContrast and Equalize. The magnitude of each operation is randomly sampled from the uniform distribution. Note that we follow a widely used data augmentation setting (Zhang et al., 2020; Zhou et al., 2021), which applies two successive data augmentation strategies on each training
which resembles the common practice by human to optimize different components in sequence. Specifically,
we directly refer to the model performance reported in papers cited in Table 2. To further demonstrate the
we could extend the size of probability matrix from

2020), and DSNAS (Hu et al., 2020). Experiments are conducted on NVIDIA V100 under PyTorch-1.3.0 and
Python 3.6. Detailed settings of baselines are provided in Appendix A.

the online DA and HPO strategy proposed in our paper and trains the architecture derived from the first
optimal architecture under certain hyper-parameter settings. In the next stage, Sequential DHA performs
Sequential DHA consists of two stages. During the first stage, Sequential DHA performs NAS to find the
benefits of joint optimization of multiple AutoML components, we also include a baseline,
re-implementation code which achieves similar performance described in the original papers. As for ImageNet,
for papers with code, including ENAS, NSGA-NET, P-DARTS and One-Stage ISTA, we directly use their
CIFAR10, CIFAR100, SPORT8, MIT67, and FLOWERS102, we re-implement the baseline methods. Specifically,

We compare DHA with various AutoML algorithms, such as ENAS (Pham et al., 2018), NSGA-
NET (Lu et al., 2019), NSGA-NET (Lu et al., 2019), P-DARTS (Chen et al., 2019), MANAS (Carlucci et al.,
2019) and One-Stage ISTA (Yang et al., 2020)(see Table 1 and Table 2). For datasets including CIFAR10,
CIFAR100, SPORT8, MIT67, and FLOWERS102, we re-implement the baseline methods. Specifically,
for papers with code, including ENAS, NSGA-NET, P-DARTS and One-Stage ISTA, we directly use their
official implementation code. For paper without code, including DARTS and MANAS, we use the unofficial
re-implementation code which achieves similar performance described in the original papers. As for ImageNet,
we directly refer to the model performance reported in papers cited in Table 2. To further demonstrate the
benefits of joint optimization of multiple AutoML components, we also include a baseline, Sequential DHA,
which resembles the common practice by human to optimize different components in sequence. Specifically,
Sequential DHA consists of two stages. During the first stage, Sequential DHA performs NAS to find the
optimal architecture under certain hyper-parameter settings. In the next stage, Sequential DHA performs
the online DA and HPO strategy proposed in our paper and trains the architecture derived from the first
stage from scratch. Detailed algorithm of Sequential DHA could be found in Appendix B.

**Baselines.** We compare DHA with various AutoML algorithms, such as ENAS (Pham et al., 2018), NSGA-
NET (Lu et al., 2019), NSGA-NET (Lu et al., 2019), P-DARTS (Chen et al., 2019), MANAS (Carlucci et al.,
2019) and One-Stage ISTA (Yang et al., 2020)(see Table 1 and Table 2). For datasets including CIFAR10,
CIFAR100, SPORT8, MIT67, and FLOWERS102, we re-implement the baseline methods. Specifically,
for papers with code, including ENAS, NSGA-NET, P-DARTS and One-Stage ISTA, we directly use their
official implementation code. For paper without code, including DARTS and MANAS, we use the unofficial
re-implementation code which achieves similar performance described in the original papers. As for ImageNet,
we directly refer to the model performance reported in papers cited in Table 2. To further demonstrate the
benefits of joint optimization of multiple AutoML components, we also include a baseline, Sequential DHA,
which resembles the common practice by human to optimize different components in sequence. Specifically,
Sequential DHA consists of two stages. During the first stage, Sequential DHA performs NAS to find the
optimal architecture under certain hyper-parameter settings. In the next stage, Sequential DHA performs
the online DA and HPO strategy proposed in our paper and trains the architecture derived from the first
stage from scratch. Detailed algorithm of Sequential DHA could be found in Appendix B.

**Implementation details.** For hyper-parameters of DHA, we simply adopt the hyper-parameters used in
previous works without many modifications, including Meta-Aug (Zhou et al., 2021), ISTA-NAS (Yang et al.,
2020), and DSNAS (Hu et al., 2020). Experiments are conducted on NVIDIA V100 under PyTorch-1.3.0 and
Python 3.6. Detailed settings of baselines are provided in Appendix A.

<table>
<thead>
<tr>
<th>Model</th>
<th>CIFAR Acc</th>
<th>CIFAR100 Acc</th>
<th>SPORT8 Acc</th>
<th>MIT67 Acc</th>
<th>FLOWERS102 Acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAS (Pham et al., 2018)</td>
<td>95.85±0.17</td>
<td>78.02±0.55</td>
<td>94.54±0.35</td>
<td>71.05±0.29</td>
<td>96.56±0.20</td>
</tr>
<tr>
<td>NSGA-NET (Lu et al., 2019)</td>
<td>96.18±0.37</td>
<td>77.31±0.14</td>
<td>93.87±0.35</td>
<td>70.20±0.41</td>
<td>95.30±0.26</td>
</tr>
<tr>
<td>DARTS (Lu et al., 2019)</td>
<td>97.24±0.10</td>
<td>82.37±0.34</td>
<td>92.53±0.34</td>
<td>70.73±0.24</td>
<td>96.30±0.24</td>
</tr>
<tr>
<td>P-DARTS (Chen et al., 2019)</td>
<td>97.13±0.07</td>
<td>82.46±0.37</td>
<td>93.45±0.66</td>
<td>70.70±0.29</td>
<td>95.09±0.26</td>
</tr>
<tr>
<td>MANAS (Carlucci et al., 2019)</td>
<td>97.18±0.07</td>
<td>82.07±0.14</td>
<td>94.46±0.22</td>
<td>71.36±0.19</td>
<td>95.67±0.18</td>
</tr>
<tr>
<td>One-Stage ISTA (Yang et al., 2020)</td>
<td>97.64±0.20</td>
<td>83.10±0.11</td>
<td>94.33±0.12</td>
<td>72.12±0.03</td>
<td>96.59±0.16</td>
</tr>
<tr>
<td>Sequential DHA</td>
<td>97.77±0.14</td>
<td>83.51±0.12</td>
<td>94.55±0.09</td>
<td>72.34±0.14</td>
<td>96.67±0.17</td>
</tr>
<tr>
<td>DHA</td>
<td>98.11±0.26</td>
<td>83.93±0.23</td>
<td>95.06±0.13</td>
<td>73.35±0.19</td>
<td>97.41±0.09</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Time</th>
<th>Time</th>
<th>Time</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ENAS (Pham et al., 2018)</td>
<td>67.2±2.1</td>
<td>67.2±4.4</td>
<td>4.92±0.5</td>
<td>46.2±2.6</td>
</tr>
<tr>
<td>NSGA-NET (Lu et al., 2019)</td>
<td>151.2±4.1</td>
<td>151.2±6.5</td>
<td>11.06±0.8</td>
<td>103.9±4.1</td>
</tr>
<tr>
<td>DARTS (Lu et al., 2019)</td>
<td>151.2±2.9</td>
<td>151.2±3.8</td>
<td>11.06±1.4</td>
<td>103.9±3</td>
</tr>
<tr>
<td>P-DARTS (Chen et al., 2019)</td>
<td>76.8±1.1</td>
<td>76.8±2.6</td>
<td>5.62±0.8</td>
<td>52.8±2</td>
</tr>
<tr>
<td>MANAS (Carlucci et al., 2019)</td>
<td>122.4±1.8</td>
<td>122.4±2.9</td>
<td>8.96±0.7</td>
<td>84.1±1</td>
</tr>
<tr>
<td>One-Stage ISTA (Yang et al., 2020)</td>
<td>55.2±1.5</td>
<td>55.2±1.2</td>
<td>4.1±0.3</td>
<td>37.9±0.4</td>
</tr>
<tr>
<td>Sequential DHA</td>
<td>101.9±1.6</td>
<td>101.9±2.3</td>
<td>7.3±0.5</td>
<td>67.9±1</td>
</tr>
<tr>
<td>DHA</td>
<td>76.6±3.2</td>
<td>76.6±3.1</td>
<td>5.6±0.5</td>
<td>52.5±1.4</td>
</tr>
</tbody>
</table>
Sequential DHA and DHA consistently outperform those NAS algorithms with manual designed DA and HPO. Specifically, DHA achieves SOTA results on all datasets. This shows the clear performance gain of extending the search scope from architecture to including also data augmentation and hyper-parameters, justifying the need for multi-component optimization in AutoML. Moreover, despite optimising over a larger search space, DHA remains cost efficiency. For example, on CIFAR100, DHA enjoys 1.56% higher test accuracy than DARTS but requires 42% less time. Besides, the comparison between DHA and Sequential DHA reveals the evident advantage of doing DA, HPO and NAS jointly over doing them separately in different stages.

Table 3: Top-1 accuracy (%) and computational time (GPU-hour) of different combination of AutoML components on CIFAR10, CIFAR100, SPORT8, MIT67 and FLOWERS102. Listed algorithms are described in Ablation study.

4.2 Results

The test accuracy and computational time of various AutoML algorithms are summarized in Table 1 and Table 2. The timing results in these two tables measure the computational time taken to obtain a ready-to-deploy network, which corresponds to the sum of search and retrain time for two-stage NAS methods and the search time for end-to-end methods like one-stage NAS method as well as our DHA.

Small-scale datasets. As shown in Table 1, methods optimizing all of DA, HPO and NAS automatically (i.e., Sequential DHA and DHA) consistently outperform those NAS algorithms with manual designed DA and HPO. Specifically, DHA achieves SOTA results on all datasets. This shows the clear performance gain of extending the search scope from architecture to including also data augmentation and hyper-parameters, justifying the need for multi-component optimization in AutoML. Moreover, despite optimising over a larger search space, DHA remains cost efficiency. For example, on CIFAR100, DHA enjoys 1.56% higher test accuracy than DARTS but requires 42% less time. Besides, the comparison between DHA and Sequential DHA reveals the evident advantage of doing DA, HPO and NAS jointly over doing them separately in different stages.

Large-scale dataset. Results of the large-scale dataset ImageNet with cell-based search space and MobileNet search space are shown in Table 2. DHA consistently outperforms various NAS methods which only involves
architecture optimization, demonstrating the benefits of joint-optimization. Even when compared with One-stage NAS methods like ISTA, DHA achieves up to 1.7% TOP-1 accuracy improvement. Moreover, in comparison with the joint-optimization algorithm APQ (Wang et al., 2020) and AutoHAS (Dong et al., 2020), DHA outperforms APQ by 2.5% and outperforms AutoHAS by 4.3%. These comparisons reveal that DHA proposes an efficient and high-performed joint-optimization algorithm. The Top-1 accuracy and computation time of these AutoML algorithms are also summarized in Fig. 2. As can be seen, DHA consistently gains highest test accuracy on all five datasets while being more cost efficient than NAS methods and Sequential DHA. The validation accuracy of Sequential-DHA and DHA during the training is shown in Fig. 3. We could notice that a traditional sequential optimization algorithm would disrupt the learning process, as it has to retrain the model after the searching phase. While benefiting from the end-to-end optimization process, DHA would have a smoother learning curve and also achieve better performance.

![Learning Curve On Sequential DHA and DHA](image)

**Figure 3**: The validation accuracy of Sequential-DHA and DHA over the training time on ImageNet with Cell-Based Structure.

4.3 Analysis of loss landscape

![Figure 4: Loss-landscape of the trained models on SPORT8. (a) One-Stage ISTA (b) Sequential DHA and (c) DHA.](image)

The strong test performance of DHA across various datasets motivates us to further check the geometry of the minimiser achieved by our final trained model. We employ the filter-normalisation method proposed in Li et al. (2017) to visualise the local loss landscape around the minimiser achieved by One-Stage ISTA, Sequential DHA and DHA. We choose a center point \( \hat{\theta} \) in the graph, and two direction vector \( \delta \) and \( \eta \). We the plot the 2D contour with the function \( f(\alpha, \beta) = L(f(\hat{\theta} + \alpha \delta + \beta \eta; B^{test})) \), where \( B^{test} \) represents the test set. The resultant contour plots are shown in Fig. 4. As can be seen, the minimum reached by optimising over more AutoML components tend to be flatter than NAS with manual designed DA and HPO.
The loss landscape of DHA is also flatter and smoother than that of Sequential DHA accounting for the better generalisation performance of DHA in previous experiments, explaining the superior test accuracy achieved by DHA in Tables 1 and 2 (Keskar et al., 2016; Xu & Mannor, 2012).

4.4 Ablation study of AutoML components

To further verify the effectiveness of our proposed extended search scope and search strategy, we empirically investigate the performance of considering all three components (i.e., DA, HPO and NAS) against combining any two of them. We examine both the sequential-optimization and joint-optimization settings with four designed optimization algorithms. **Sequential NAS + DA** first conducts NAS and during the tuning stage, the proposed DA optimization is applied. Similar to the Sequential NAS+DA, **Sequential NAS + HPO** firstly conducts NAS. Then with the fixed architecture, Sequential NAS + DA optimizes the HPO and networks’ weights simultaneously. In contrast to them, **Joint-optimization NAS + DA** simultaneously conduct DA strategy optimization, NAS and parameter weights optimization. **Joint-optimization NAS + HPO** simultaneously conducts HPO, NAS and parameter weights optimization.

The comparison results are presented in Table 3. We can notice that performing joint optimization for either NAS + DA or NAS + HPO, achieves higher test accuracy than doing them in a pipeline. This reconfirms our previous conclusion that optimising different AutoML components jointly is better than doing them in sequence. Moreover, by comparing the results of the joint-optimization NAS+DA and the joint-optimization NAS+HPO in Table 3 against the One-Stage ISTA in Table 1, it is clear that considering one more AutoML component on top of NAS can lead to clear performance gain of the final model. While such gain is higher for incorporating DA than HPO, it is maximised when all three components are considered; our DHA obtains the best test accuracy among all joint-optimization baselines across all datasets.

We also provide (1) more ablation studies of DA and HPO (see Appendix C), (2) analysis of the searched architectures (see Appendix D) and comparison in terms of FLOPs and Latency (see Appendix E). Please refer to the Appendix for more details.

5 Conclusion

In this work, we present DHA, an end-to-end joint-optimization method for three important components of AutoML, including DA, HPO and NAS. This differentiable joint-optimization method can efficiently optimize larger search space than precious AutoML methods and achieve SOTA results on various datasets with a relatively low computational cost. Specifically, DHA achieves 77.4% Top-1 accuracy on ImageNet with cell based search space, which is higher than current SOTA by 0.5%. With DHA, we show the advantage of doing joint-optimization of AutoML over doing co-optimization in sequence, and conclude that joint optimization of multiple AutoML components is necessary.

6 Acknowledgments

We gratefully acknowledge the support of MindSpore, CANN (Compute Architecture for Neural Networks) and Ascend AI Processor used for this research.

References


Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In ICML, 2018.


Fengwei Zhou, Jiawei Li, Chuanlong Xie, Fei Chen, Lanqing Hong, Rui Sun, and Zhenguo Li. Metaaugment: Sample-aware data augmentation policy learning. In AAAI, 2021.
