DAAS: DIFFERENTIABLE ARCHITECTURE AND AUGMENTATION POLICY SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural architecture search (NAS) has been an active direction of automatic machine learning (Auto-ML), aiming to explore efficient network structures. The searched architecture is evaluated by training on datasets with fixed data augmentation policies. However, recent works on auto-augmentation show that the suited augmentation policies can vary over different structures. Therefore, this work considers the possible coupling between neural architectures and data augmentation and proposes an effective algorithm jointly searching for them. Specifically, 1) for the NAS task, we adopt a single-path based differentiable method with Gumbel-softmax reparameterization strategy due to its memory efficiency; 2) for the auto-augmentation task, we introduce a novel search method based on policy gradient algorithm, which can significantly reduce the computation complexity. Our approach achieves 97.91% accuracy on CIFAR-10 and 76.6% Top-1 accuracy on ImageNet dataset, showing the outstanding performance of our search algorithm.

1 INTRODUCTION

AutoML aims to automatically construct and train machine learning models with no need for human participation. Auto-augmentation (AA) and Neural Architecture Search (NAS) are two popular directions. A series of mechanisms have been designed for AA and NAS, such as reinforcement learning (Cubuk et al., 2019; Zoph et al., 2018), evolutionary algorithm (Real et al., 2019), Bayesian optimization (Lim et al., 2019; White et al., 2021), and gradient-based methods (Hataya et al., 2020; Li et al., 2020; Liu et al., 2019). These prior works are usually dedicated to either AA or NAS, but few explore joint searching for data augmentation policies and neural architectures.

In fact, there can be some connection between data augmentation policies and neural architectures. On the one hand, the performance of the architecture searched by NAS can be further improved by proper data augmentation. Fig. 1 shows an architecture searched by our method, which attains 97.36% accuracy on CIFAR-10 (Krizhevsky et al., 2009) under the default data augmentation policy of DARTS (Liu et al., 2019) but achieves 97.91% under our searched policies; On the other hand, the optimal augmentation policies for different architectures may also vary (see Sec. 5.2).

We argue and show that (see experiments): 1) It is beneficial to search for data augmentation policies and network architectures jointly; 2) The vanilla evaluation metric of NAS, i.e. training under a fixed policy, is biased and can mistakenly reject exemplary architectures with appropriate policy.

This paper proposes DAAS, a differentiable method, to jointly search for both data augmentation policies and neural architecture. For the AA task, we introduce a novel search method based on policy gradient algorithm rather than Gumbel reparameterization trick (Hataya et al., 2020; Li et al., 2020). Our analysis and experiment results show the high efficiency and effectiveness of our method. For the NAS task, we refer to single-path based methods (Dong & Yang, 2019; Wang et al., 2020a) and sample candidate architectures by activating a subset of edges in the supernet to reduce GPU memory cost. The contributions of this work can be summarized as follows.

1) Policy gradient based search algorithm for Auto-augmentation. For the AA task, unlike the works (Li et al., 2020; Hataya et al., 2020) that utilize Gumbel reparameterization trick, we propose to update policy parameters by policy gradient algorithm, which can simplify the second-order partial derivatives into a vector multiplication of two first-order derivatives.

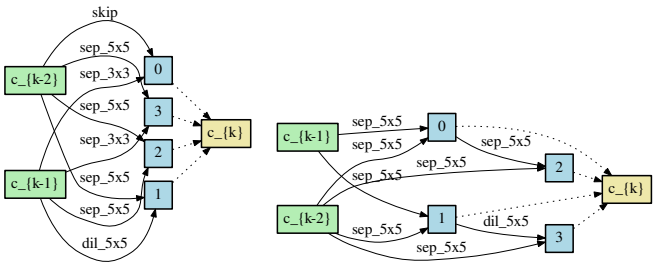
2) Differentiable joint searching framework for AA and NAS. To our best knowledge, this is the first work (except for the arxiv work Kashima et al. (2020)) to jointly search for AA and NAS in a differentiable manner. By constructing a bi-level optimization model to update the search parameters[1] and network weights alternately, an effective combination of architecture and augmentation policies can be found in a single GPU-day. In contrast, a

3) Strong performance. Our AA algorithm can achieve competitive and even better performance than current AA methods. Moreover, our framework performs better than prior NAS methods, showing the necessity to combine AA with NAS tasks. Under the search space of DARTS, the discovered architecture achieves 97.91% accuracy on CIFAR-10 and 76.6% top-1 accuracy on ImageNet.

2 Related Work

Differentiable Neural Architecture Search. DARTS (Liu et al., 2019) builds a cell-based supernet and introduces architecture parameters to represent the importance of operations. Though DARTS reduces the search cost to a few GPU-days, it suffers high GPU memory cost, and works revise the forward process. PC-DARTS (Xu et al., 2020) makes use of partial connections instead of full-fledged supernet. MergeNAS (Wang et al., 2020b) merges all parametric operations into one convolution, a similar super-kernel strategy is also used by Stamoulis et al. (2019). GDAS (Dong & Yang, 2019) adopts the Gumbel reparameterization technique to sample a sub-graph of the supernet at each iteration to reduce GPU memory cost. ROME (Wang et al., 2020a) reveals the instability issue in GDAS and stabilizes the search by topology disentanglement and gradient accumulation.

Auto-augmentation. Cubuk et al. (2019) first adopt reinforcement learning for auto-augmentation task but it requires to search for thousands of GPU-days. Fast-AA (Lim et al., 2019) introduces Bayesian Optimization to speed up the searching. Similar to GDAS, DADA (Li et al., 2020) introduces trainable policy parameters and adopts the Gumbel technique to update it with network weights by gradient descent algorithm alternately. Faster-AA Hataya et al. (2020) also utilizes the Gumbel technique and regards the AA task as a density matching problem. Unlike prior Gumbel-based methods (Li et al., 2020; Hataya et al., 2020), this work adopts policy gradient algorithm to update policy parameters, which can reduce the computation cost and achieves better performance.



(a) normal cell (b) reduction cell

Figure 1: Motivating example: the architecture of (a) normal and (b) reduction cells discovered by our method. We observe that cells are dominated by 5x5 separable convolutions, which may not be well recommended in NAS literature. Nevertheless, its performance can be improved by 0.5% by suited data augmentation policy, achieving 97.91% accuracy on CIFAR-10. Detailed comparison on the default augmentation policies and our searched ones is reported in Table 4.

Joint searching for AutoML. Auto-augmentation (AA), Neural Architecture Search (NAS), and Hyper-parameter Optimization (HPO) are three popular branches of AutoML. Recent works (Dong et al., 2020; Dai et al., 2020; Klein & Hutter, 2019; Zela et al., 2018) have explored joint searching for NAS and HPO based on reinforcement learning and surrogate model to predict the performance.

Our work is one of the first works to explore the differentiable joint searching for AA and NAS. In particular, it essentially differs from the concurrent arxiv works (Kashima et al., 2020; Zhou et al., 2021). Specifically, the first work simply combines Faster-AA and DARTS, which requires large GPU memory and performs worse than independent search. The second work, named DHA (Zhou et al., 2021), regards AA and NAS as a one-level optimization problem, which oversimplifies the joint searching problem for NAS and AA. In contrast, bi-level optimization is adopted in our approach, which seeks better AA policies given the searched architecture and vice versa. Moreover, DHA only searches for augmentation policies but ignores the fine-grained search for application

[1]The search parameters are made of architecture parameters for NAS and policy parameters for AA.

Table 1: Detailed difference between our DAAS and two closely related concurrent works.

Method	Target	Search Space	Search Method
Kashima et al. (2020)	AA+NAS	**1) AA:** the same search space of Faster-AA; **2) NAS:** the same search space of DARTS.	**1)** Bi-level optimization for both AA and NAS; **2)** GPU memory consuming since it simply combines DARTS and Faster-AA; **3)** Gumbel reparameterization technique to estimate gradients for policy parameters.
DHA (Zhou et al., 2021)	AA+NAS+HPO	**1) AA:** types of augmentation policy; **2) NAS:** the same search space of DARTS; **3) HPO:** L2 regularization and learning rate.	**1)** One-level optimization for AA and NAS task, and bi-level optimization for HPO task; **2)** Memory efficient based on sparse coding (ISTA-NAS); **3)** Gumbel reparameterization technique to estimate gradients for policy parameters.
DAAS (ours)	AA+NAS	**1) AA:** types of augmentation policy, and the application probability and magnitude for each pre-processing operator; **2) NAS:** the same search space of DARTS.	**1)** Bi-level optimization for both AA and NAS; **2)** Memory efficient NAS method by sampling a subset of supernet edges at each iteration; **3)** Policy gradient based algorithm for AA task, which reduces the computation cost of gradient estimation for policy parameters.

probability and magnitude for pre-processing operations. While we search in a more detailed search space but is still cost-effective thanks to our differentiable search technique. Table 1 shows a detailed difference of our work to the two arxiv papers. As will be shown in our experiments, our method also outperforms DHA under the same setting.

3 Preliminaries and Search Space Choice

In this section, we first introduce the background of gradient-based methods for NAS and AA in Sec. 3.1, and then detail the search space of our joint searching task in Sec. 3.2.

3.1 Preliminaries of Differentiable Methods

Differentiable NAS (DARTS) is first introduced in (Liu et al., 2019). By constructing a supernet with normal cells and reduction cells, it introduces architecture parameters $\boldsymbol{\alpha}$ to represent the importance of candidate operations and connections and regards NAS as a bi-level optimization problem. To reduce the GPU memory and reduce the topology gap between the supernet and the final network, works (Dong & Yang, 2019; Xie et al., 2019; Wang et al., 2020a) utilize the Gumbel technique to sample and activate a subset of operations and construct the bi-level optimization model as Eq. 1, where $\boldsymbol{\theta}_z^*$ is the optimal operation parameters for the sampled architecture z.

$$\min_{\boldsymbol{\alpha}} \quad L_{val}(\boldsymbol{\theta}_z^*, z); \qquad \text{s.t.} \quad z \sim p(z; \boldsymbol{\alpha}), \quad \boldsymbol{\theta}_z^* = \arg\min_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta}, z) \tag{1}$$

Inspired by DARTS (Liu et al., 2019) and GDAS (Dong & Yang, 2019), DADA (Li et al., 2020) adopts the differentiable based method and Gumbel technique for auto-augmentation. It introduces policy parameters γ to represent the importance and hyper-parameters for augmentation policies and formulate the bi-level optimization model as Eq. 2, where Γ denotes the sampled augmentation policy and its hyper-parameters and D_{train}, D_{val} denotes the training set and validation set.

$$\min_{\boldsymbol{\gamma}} \quad L_{val}(\boldsymbol{\theta}^*; D_{val}); \qquad \text{s.t.} \quad \Gamma \sim p(\Gamma; \boldsymbol{\gamma}), \quad \boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta}; \Gamma(D_{train})) \tag{2}$$

3.2 Search Space for Architecture and Data Augmentation

In line with the mainstream of existing works, the joint search space is a Cartesian product of architecture and augmentation policy candidates. For the NAS task, we refer to DARTS (Liu et al., 2019) and construct a supernet containing all the candidate operations and connections; For the AA task, we refer to AA (Cubuk et al., 2019) and DADA (Li et al., 2020) and pair up candidate image pre-processing operations to build policies.

Architecture Search Space. A supernet is stacked by normal and reduction cells. Each cell contains N nodes $\{x_i\}_{i=1}^N$ representing latent feature maps. The outputs of all intermediate nodes are concatenated as the output of the cell. There is an parallel edge $\boldsymbol{e}_{i,j}$ between every two nodes x_i, x_j, which integrates all the candidate operations, i.e. $\boldsymbol{e}_{i,j} = \{o_{i,j} | o \in \mathcal{O}_{arch}\}$, where $\mathcal{O}_{arch}$ is the candidate operation set, including separable convolutions, pooling, and identity operations. We define architecture parameters $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ to represent the importance of edges and operations in the supernet, respectively. At each iteration, we sample two parallel edges for each node based on $\boldsymbol{\beta}$ and sample

one operation for each parallel edge based on $\boldsymbol{\alpha}$. The output of node x_j can be computed as follows:

$$x_j = \sum_{i<j} B_{i,j} \cdot \sum_{o \in \mathcal{O}_{arch}} A^o_{i,j} o_{i,j}(x_i); \quad \text{s.t.} \quad A^o_{i,j}, B_{i,j} \in \{0,1\}, \quad \sum_{i<j} B_{i,j} = 2, \sum_{o \in \mathcal{O}_{arch}} A^o_{i,j} = 1$$

where $\boldsymbol{B}_{:,j} = [B_{i,j}]_{i<j}$ is a two-hot vector denoting the sampled edges and $\boldsymbol{A}_{i,j} = [A^o_{i,j}]_{o \in \mathcal{O}_{arch}}$ is a one-hot vector denoting the sampled operation for the parallel edge $\boldsymbol{e}_{i,j}$.

Data Augmentation Search Space. Similar to DADA (Li et al., 2020) and Fast-AA (Lim et al., 2019), we construct a set of image pre-processing operations $\mathcal{O}_{aug}$ with 15 candidates, including rotation, translation, and etc. Each policy Γ contains K image pre-processing operations: $\Gamma = \{o_k\}_{k=1}^K$, where $o_k \in \mathcal{O}_{aug}$. So there are total $|\mathcal{O}_{aug}|^K$ policies in our search space. The policy parameters $\boldsymbol{\gamma}$ contain three parts: the sampling weights $\boldsymbol{\pi}$, the operation probability $\boldsymbol{p}$, and the operation magnitude $\boldsymbol{\delta}$. Specifically, we define sampling weights $\boldsymbol{\pi} \in \mathbb{R}^{K \times |\mathcal{O}_{aug}|}$ to represent the sampling probability for each candidate operation. Therefore, the k-th image operation can be sampled as: $o_k \sim p(o; \boldsymbol{\pi}_k) = \text{softmax}(\boldsymbol{\pi}_k) \in \mathbb{R}^{|\mathcal{O}_{aug}|}$. Additionally, For i-th policy, each image operation o^i_k owns two parameters: the probability p^i_k and magnitude m^i_k to apply the operation. Referring to AA (Cubuk et al., 2019), we discretize the range of magnitude into 10 values (uniform spacing) and use $\boldsymbol{\delta}^i_k \in \mathbb{R}^{10}$ to represent the importance of each magnitude candidate, so that the output of image pre-processing $\tilde{o}^i_k$ can be computed as follows, where $m^i_k \sim p(m^i_k; \boldsymbol{\delta}^i_k) = \text{softmax}(\boldsymbol{\delta}^i_k)$.

$$\tilde{o}^i_k(D; p^i_k, m^i_k) = \begin{cases} o^i_k(D; m^i_k), & \text{with probability } \left(p^i_k \cdot p(m^i_k; \boldsymbol{\delta}^i_k)\right) \\ D, & \text{with probability } \left(1 - p^i_k\right) \end{cases}. \tag{3}$$

Therefore, the probability of a specific policy $\Gamma^i = \{o^i_k, y^i_k, m^i_k\}_{k=1}^K$ can be formulated as Eq. 4, where $y^i_k \in \{0,1\}$ indicates whether to apply the operation o^i_k, and m^i_k is the sampled magnitude.

$$p(\Gamma^i) = \prod_{k=1}^K p(o^i_k; \boldsymbol{\pi}_k) \cdot (1 - p^i_k)^{(1-y^i_k)} \cdot \left[p^i_k \cdot p(m^i_k; \boldsymbol{\delta}^i_k)\right]^{y^i_k} \tag{4}$$

4 METHODOLOGY

In this section, we first introduce the bi-level optimization model for joint searching in Sec. 4.1 and then detail the algorithm in Sec. 4.2. Next, we introduce the strategy to derive the final augmentation policy and architecture in Sec. 4.3. Finally, we analyze the superiority of our method in Sec. 4.4.

4.1 BI-LEVEL OPTIMIZATION FOR JOINT SEARCHING

Jointly searching for NAS and AA can be formulated as a bi-level optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}} \quad & \bar{L} = \mathbb{E}_{z \sim p(z; \boldsymbol{\alpha}, \boldsymbol{\beta})} \left[L_{val}(\boldsymbol{\theta}^*_z(\boldsymbol{\gamma}), z, D_{val}) \right] \\ \text{s.t.} \quad & \boldsymbol{\theta}^*_z(\gamma) = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\Gamma \sim p(\Gamma; \boldsymbol{\gamma})} \left[L_{train}(\boldsymbol{\theta}, z, \Gamma(D_{train})) \right] \end{aligned} \tag{5}$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ represent the importance of candidate operations and connections, $\boldsymbol{\gamma}$ represent the importance of candidate augmentation policies and the corresponding hyper-parameters, and $\boldsymbol{\theta}$ is the network weights in the supernet. Our bi-level optimization is not only combine the optimization model of NAS and AA. On the one hand, different from the optimization in GDAS (Eq. 1) that only sample once to compute L_{val}, we argue that the expectation of L_{val} w.r.t. $p(z)$ should be considered and propose to sample n_{arch} architectures and average the loss; On the other hand, unlike the optimization in DADA (Eq. 2), we also consider the expectation of L_{train} w.r.t. $p(\Gamma)$ and sample n_{aug} policies for each sampled architecture z.

This work adopts differentiable method to solve the above optimization method, and the difficulty to lies in how to estimate the gradient for architecture parameters $\boldsymbol{\alpha}$ and policy parameters $\boldsymbol{\gamma}$. Specifically, we utilize first-order approximation as prior works (Liu et al., 2019; Dong & Yang, 2019; Wang et al., 2020a) and estimate gradients for $\boldsymbol{\alpha}$ as $\nabla_z L_{val} \cdot \nabla_{\boldsymbol{\alpha}} z$. However, according to the chain rule, the gradients for $\boldsymbol{\gamma}$ is $\nabla_{\boldsymbol{\gamma}} L_{val} = \nabla_{\boldsymbol{\theta}^*} L_{val} \cdot \nabla_{\boldsymbol{\gamma}} \boldsymbol{\theta}^*$ in which $\boldsymbol{\theta}^*$ is intractable. DADA refers to the second-order approximation in DARTS and approximate $\boldsymbol{\theta}^* \approx \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} L_{train}$, where η is the learning rate to train operation weights. The gradient $\nabla_{\boldsymbol{\gamma}} L_{val}$ can be then approximated as:

$$\nabla_{\boldsymbol{\gamma}} L_{val} \approx \nabla_{\boldsymbol{\theta}^*} L_{val} \cdot (-\eta) \nabla^2_{\Gamma, \boldsymbol{\theta}} L_{train} \cdot \nabla_{\boldsymbol{\gamma}} \Gamma. \tag{6}$$

Algorithm 1: DAAS: Joint Searching for Data Augmentation Policies and Neural Architecture

Parameters of supernet:

Initialized operation weights $\boldsymbol{\theta}$, architecture parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and policy parameters $\boldsymbol{\gamma}$;

while *not converged* **do**

Sample batches of data D_{train}, D_{val}, candidate architectures $\{z^{(i)}\}$, and policies $\{\Gamma^{(i,j)}\}$;

Estimate the gradients for the architectures parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ by Eq. 9 and policy parameters $\boldsymbol{\gamma}$ by Eq. 13, and update them by gradient descent: $\boldsymbol{a} \leftarrow \boldsymbol{a} - \xi_{\boldsymbol{a}} \nabla_{\boldsymbol{a}} \bar{L}, \boldsymbol{a} \in \{\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}\}$;

Estimate gradients for supernet weights by Eq. 14 and update by $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \xi_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \bar{L}_{train}$;

The second-order partial derivative in Eq. 6 requires complex computation, so DADA has to shrink the search dataset to speed up the search process. Moreover, we notice the policy Γ is usually non-differentiable w.r.t. $\boldsymbol{\gamma}$, i.e. $\nabla_{\boldsymbol{\gamma}}\Gamma$ is nonexistent. DADA manually defines $\nabla_{\boldsymbol{\gamma}}\Gamma \equiv 1$, which is groundless and incorrect. In this work, we utilize policy gradient algorithm to estimate the $\nabla_{\boldsymbol{\gamma}}\Gamma$ instead of a manually-designed constant.

4.2 Proposed DAAS Algorithm

Gradients for architecture parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}$. Referring to GDAS (Dong & Yang, 2019) and ROME (Wang et al., 2020a), we adopt Gumbel-softmax reparameterization trick to sample architectures. As shown in Eq. 7 and Eq. 8, $\tilde{\boldsymbol{\alpha}}^o_{i,j} = \frac{\exp \alpha^o_{i,j}}{\sum_{o' \in \mathcal{O}} \exp \alpha^{o'}_{i,j}}$ and $\tilde{\boldsymbol{\beta}}_{i,j} = \frac{\exp \beta i,j}{\sum_{k<j} \exp \beta_{k,j}}$ are normalized architecture parameters, $B_{i,j} = 1$ denotes that $\boldsymbol{e}_{i,j}$ is sampled and activated, and $A^o_{i,j} = 1$ denotes the operation o is sampled on edge $\boldsymbol{e}_{i,j}$, $\boldsymbol{g}$ are sampled from Gumbel(0,1) distribution.

$$\tilde{\boldsymbol{A}}^o_{i,j} = \frac{\exp\left[(\log \tilde{\boldsymbol{\alpha}}^o_{i,j} + \boldsymbol{g}^o_{i,j})/\tau\right]}{\sum_{o'=1}^{|\mathcal{O}|} \exp\left[(\log \tilde{\boldsymbol{\alpha}}^{o'}_{i,j} + \boldsymbol{g}^{o'}_{i,j})/\tau\right]}; \quad \boldsymbol{A}_{i,j} = \text{one\_hot}\left[\arg\max_{o \in \mathcal{O}} \tilde{\boldsymbol{A}}^o_{i,j}\right] \tag{7}$$

$$\tilde{\boldsymbol{B}}_{i,j} = \frac{\exp\left((\log \tilde{\boldsymbol{\beta}}_{i,j} + \boldsymbol{g}_{i,j})/\tau\right)}{\sum_{i'<j} \exp\left((\log \tilde{\boldsymbol{\beta}}_{i',j} + \boldsymbol{g}_{i',j})/\tau\right)}; \quad \boldsymbol{B}_{i,j} = \begin{cases} 1, & i \in \arg\operatorname{top2}_{i'<j}(\tilde{\boldsymbol{B}}_{i',j}) \\ 0, & otherwise \end{cases}. \tag{8}$$

Therefore, the sampled architecture z can be determined by: $z = \{\boldsymbol{B}_{i,j} | 2 \le j \le N, 1 \le i < j\} \cup \{\boldsymbol{A}^o_{i,j} | 2 \le j \le N, 1 \le i < j, o \in \mathcal{O}_{arch}\}$. In the forward pass, only the sampled operations are activated so that the computation cost can be reduced. While in the backward pass, we estimate the gradients $\nabla_{\boldsymbol{\alpha}} z = \nabla_{\boldsymbol{\alpha}} \tilde{\boldsymbol{A}}$ and $\nabla_{\boldsymbol{\beta}} z = \nabla_{\boldsymbol{\beta}} \tilde{\boldsymbol{B}}$. In our experiments, we sample $n_{arch} = 5$ architectures $\{z^{(i)} | 1 \le i \le n_{arch}\}$ and estimate the expectation by mean value. Therefore, the gradient of search loss w.r.t. architecture parameters can be formulated as follows, where D_{val} is a batch of data.

$$\nabla_{\boldsymbol{a}} \bar{L} = \frac{1}{n_{arch}} \sum_{i=1}^{n_{arch}} \nabla_z L_{val}(\boldsymbol{\theta}, z^{(i)}, D_{val}) \cdot \nabla_{\boldsymbol{a}} z^{(i)}, \quad \boldsymbol{a} \in \{\boldsymbol{\alpha}/\boldsymbol{\beta}\} \tag{9}$$

Gradients for policy parameters $\boldsymbol{\gamma}$. Similar to DADA in Eq. 6, we adopt the second-order approximation to estimate the gradients $\nabla_{\boldsymbol{\gamma}} \bar{L}$. Given a sampled architecture z, we apply one-step SGD algorithm to estimate the optimal network weights in Eq. 5 as follows:

$$\boldsymbol{\theta}^* \approx \boldsymbol{\theta} - \xi \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\Gamma \sim p(\Gamma;\boldsymbol{\gamma})} \left[L_{train}(\boldsymbol{\theta}, z, \Gamma(D_{train}))\right] \triangleq \boldsymbol{\theta}' \tag{10}$$

where ξ is the learning rate and Γ is the augmentation policy. Therefore, the gradient of search loss w.r.t. policy parameters $\boldsymbol{\gamma}$ can be formulated as follows:

$$\nabla_{\boldsymbol{\gamma}} \bar{L} = \nabla_{\boldsymbol{\theta}'} \mathbb{E}_z \left\{L_{val}(\boldsymbol{\theta}', z, D_{val}) \cdot (-\xi) \cdot \nabla^2_{\boldsymbol{\theta},\boldsymbol{\gamma}} \mathbb{E}_{\Gamma \sim p(\Gamma;\boldsymbol{\gamma})} \left[L_{train}(\boldsymbol{\theta}, z, \Gamma(D_{train}))\right]\right\} \tag{11}$$

Unlike DADA (Li et al., 2020) that manually defines $\nabla_{\boldsymbol{\gamma}} L \equiv 1$, we adopt policy gradient algorithm. Specifically, the expectation $\mathbb{E}_{\Gamma \sim p(\Gamma;\boldsymbol{\gamma})} \left[L_{train}(\boldsymbol{\theta}, z, \Gamma(D_{train}))\right]$ in Eq. 11 can be formulated as $\sum \left[L(\boldsymbol{\theta}, z, \Gamma(D_{train})) \cdot p(\Gamma)\right]$, then the second-order partial derivative can be simplified as:

$$\begin{aligned} &\nabla^2_{\boldsymbol{\theta},\boldsymbol{\gamma}} \mathbb{E}_{\Gamma \sim p(\Gamma;\boldsymbol{\gamma})} \left[L(\boldsymbol{\theta}, z, \Gamma(D))\right] = \nabla^2_{\boldsymbol{\theta},\boldsymbol{\gamma}} \sum \left[L(\boldsymbol{\theta}, z, D') \cdot p(\Gamma)\right] \\ &= \sum \left[\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, z, D') \cdot \nabla_{\boldsymbol{\gamma}} \log p(\Gamma) \cdot p(\Gamma)\right] = \mathbb{E}_{\Gamma \sim p(\Gamma;\boldsymbol{\gamma})} \left[\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, z, D') \cdot \nabla_{\boldsymbol{\gamma}} \log p(\Gamma)\right] \end{aligned} \tag{12}$$

where $D' = \Gamma(D)$ is the augmented data, and $p(\Gamma)$ is the probability of a specific policy Γ as formulated as Eq. 4. Based on Eq. 12, we observe that the intractable second-order partial derivative in fact can be exactly computed by a vector multiplication of two first-order gradients. Therefore, Eq. 11 can be simplified as follows, where $\boldsymbol{\theta}'$ is defined in Eq. 10.

$$\begin{aligned}\nabla_{\boldsymbol{\gamma}} \bar{L} &= -\xi \mathbb{E}_z \left\{ \nabla_{\boldsymbol{\theta}'} L_{val}(\boldsymbol{\theta}', z, D_{val}) \cdot \mathbb{E}_{\Gamma \sim p(\Gamma;\boldsymbol{\gamma})} \left[\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, z, D') \cdot \nabla_{\boldsymbol{\gamma}} \log p(\Gamma) \right] \right\} \\ &= -\xi \sum_{i=1}^{n_{arch}} \left[\nabla_{\boldsymbol{\theta}'} L_{val}(\boldsymbol{\theta}', z^{(i)}, D_{val}) \cdot \sum_{j=1}^{n_{aug}} \left[\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, z^{(i)}, D') \cdot \nabla_{\boldsymbol{\gamma}} \log p(\Gamma^{(i,j)}) \right] \right]\end{aligned} \tag{13}$$

Gradients for supernet weights $\boldsymbol{\theta}$. As pointed out by the recent work (Wang et al., 2020a), single-path based methods can mislead the training for supernet weights due to insufficient architecture sampling. Therefore, we adopt the gradient accumulation technique (Wang et al., 2020a) and sample multiple architectures to accumulate the gradients of $\boldsymbol{\theta}$. Specifically, one augmentation policy Γ is first sampled and fixed to apply to the input data. We then sample $n_w = 10$ architectures and conduct backward pass independently for these samples. Gradients for supernet weights $\boldsymbol{\theta}$ are accumulated:

$$\nabla_{\boldsymbol{\theta}} \bar{L}_{train} = \sum_{i=1}^{n_w} \nabla_{\boldsymbol{\theta}} L_{train}(\boldsymbol{\theta}, z^{(i)}, \Gamma(D_{train})). \tag{14}$$

The algorithm of our DAAS is outlined in Alg. 1.

4.3 Deriving Augmentation Policies and Architecture

For NAS, the final architecture is inferred according to the magnitude of architecture parameters. Similar to (Liu et al., 2019; Wang et al., 2020a), we preserve two edges for each node (based on $\boldsymbol{\beta}$) and one operation on each selected edge (based on $\boldsymbol{\alpha}$). For AA task, we can pair up the data augmentation operation and enumerate all policies and its sampling probability: for policy $\Gamma^i = \{o_k^i\}_{k=1}^K$, its sampling probability is $p(\Gamma^i) = \prod_{k=1}^K p(o_k^i; \boldsymbol{\pi}_k)$. For each operation o_k^i in policy Γ^i, its application probability is p_k^i and its application magnitude is $m_k^i = \arg\max p(m_k^i; \boldsymbol{\delta}_k^i)$.

Our method is evaluated by training the discovered architectures from scratch with corresponding data augmentation policies. At each iteration, we sample one policy $\Gamma = \{o_k\}_{k=1}^K$ based on the sampling probability $p(\Gamma)$ and augment the input data by the K pre-processing operations in sequence.

4.4 Discussion

Strength of policy gradient algorithm for AA. Recent differentiable based AA works (Li et al., 2020; Hataya et al., 2020) adopt Gumbel reparameterization trick to estimate the gradient of loss w.r.t. the policy parameters. However, such estimation is biased due to *inaccurate gradient definition for magnitude parameters*. The augmentation operation $o(D; p, m)$ is non-differentiable w.r.t. the magnitude m, but they have to manually define $\nabla_m o(D; p, m) \equiv 1$ to satisfy the chain rule for gradient estimation. In this work, we adopt a policy gradient algorithm to estimate the gradient for $\boldsymbol{\gamma}$. On the one hand, our algorithm does not need to define inaccurate gradients for magnitude parameters. On the other hand, thanks to the policy gradient algorithm, the second-order partial derivative can be simplified as a multiplication of two first-order derivatives (Eq. 12).

Strength of multiple sampling. In many prior differentiable based works for AA (Li et al., 2020; Hataya et al., 2020) and NAS (Dong & Yang, 2019; Xie et al., 2019), only one sampling for augmentation policy or network architecture is considered for computing the search loss. Rethinking the optimization target, we aim to obtain the probability for optimal policy $p(\Gamma; \boldsymbol{\gamma})$ and architecture $p(z; \boldsymbol{\alpha}, \boldsymbol{\beta})$ given a fixed batch of data D. Therefore, a single sample is insufficient to represent all candidates in the distribution, resulting in biased gradient estimation for architecture and policy parameters. Recent works (Liu et al., 2021; Wang et al., 2020a) also point out the importance of multiple sampling for AA and NAS. In this work, we refine the optimization target for NAS and AA and construct the bi-level optimization model for joint searching in Eq. 5.

Strength of joint searching. On one hand, optimal augmentation policy can be coupled with architecture. Recent works on AA (Li et al., 2020; Hataya et al., 2020; Cubuk et al., 2019) search augmentation policy for different network architectures, including Wide-ResNet (Zagoruyko & Komodakis, 2016), ShakeShake (Gastaldi, 2017), and Pyramid (Yamada et al., 2019), and the searched

Table 2: Comparison of the state-of-the-art models on CIFAR-10 (left) and CIFAR-100 (right). We report the best (in the first block) and averaged (in the second block) performance over four parallel tests by searching under different random seeds. *: The results are obtained by training the best architecture for multiple times rather than searching for multiple times. †: Results of joint searching for AA and NAS reported in DHA (Zhou et al., 2021).

CIFAR-10	**Params** (M)	**Error** (%)	**Cost** GPU Days
NASNet-A (2018)	3.3	2.65	2000
ENAS (2018)	4.6	2.89	0.5
DARTS (2019)	3.3	3.00±0.14*	0.4
P-DARTS (2019)	3.4	2.50	0.3
SNAS (2019)	2.8	2.85±0.02*	1.5
GDAS (2019)	3.4	2.93	0.2
PC-DARTS (2020)	3.6	2.57	0.1
DARTS- (2021)	3.5	2.50	0.4
DAAS (best)	4.4	**2.09**	1.0
R-DARTS (2020)	-	2.95±0.21	1.6
SDARTS-ADV (2020)	3.3	2.61±0.02	1.3
ROME (2020a)	3.7	2.58±0.07	0.3
DARTS- (2021)	3.5	2.59±0.08	0.4
DHA†(AA+NAS) (2021)	-	2.22±0.13	2.7
DAAS (avg)	4.0	**2.24±0.10**	1.0

CIFAR-100	**Params** (M)	**Error** (%)	**Cost** GPU Days
AmoebaNet (2019)	3.1	18.93	3150
PNAS (2018)	3.2	19.53	150
ENAS (2018)	4.6	19.43	0.45
DARTS (2019)	-	20.58±0.44*	0.4
P-DARTS (2019)	3.6	17.49	0.3
GDAS (2019)	3.4	18.38	0.2
ROME (2020a)	4.4	17.33	0.3
DARTS- (2021)	3.4	17.16	0.4
Ours (best)	3.7	**15.20**	1.0
R-DARTS (2020)	-	18.01±0.26	1.6
ROME (2020a)	4.4	17.41±0.12	0.3
DARTS- (2021)	3.3	17.51±0.25	0.4
DHA†(AA+NAS) (2021)	-	16.45±0.03	2.7
Ours (avg)	3.8	**15.37±0.31**	1.0

optimal policies differs over architectures. On the other hand, optimal network architecture is also related to data augmentation policy because architecture can be regarded as a feature extractor and is sensitive to some specific data distributions which can be affected by augmentation policy. Consequently, joint searching for NAS and AA considers the coupling between network architecture and augmentation policy and can find optimal combinations. Results in Sec. 5.2 verify our analysis.

5 Experiments

Search Settings. We follow DARTS (Liu et al., 2019) and construct a supernet by stacking 8 cells with 16 initial channels. Each cell contains $N = 6$ nodes, two of which are input nodes. Two reduction cells are located at $1/3$ and $2/3$ of the total depth of the supernet. In the search stage, we first warmup the supernet by alternately updating operation parameters $\boldsymbol{\theta}$ and architecture parameters ($\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$) for 20 epochs, and then jointly search for architectures and policies for another 30 epochs. We set the sampling number $n_{arch} = 5$ and $n_{aug} = 2$ by default. For operation parameters, we use SGD optimizer with 0.9 momentum; For architecture and policy weights, we adopt Adam optimizer with $\beta = (0.5, 0.999)$. To search on CIFAR-10 and CIFAR-100, we split the training set into two parts as D_{train} and D_{val} to train supernet weights $\boldsymbol{\theta}$ and search parameters $\boldsymbol{\alpha}, \boldsymbol{\beta}, \boldsymbol{\gamma}$ respectively. Moreover, due to the high efficiency of our method, we directly search on ImageNet. We follow DADA (Li et al., 2020) and construct a surrogate dataset by selecting 120 classes.

Evaluation Settings. We use standard evaluation settings as DARTS (Liu et al., 2019) by training the inferred model for 600 epochs using SGD with a batch size of 96 for CIFAR-10 and CIFAR-100. The searched architecture is also transferred to ImageNet by stacking 14 cells with 48 initial channels. The transferred and searched models on ImageNet are trained for 250 epochs by SGD with a batch size of 1024. In the evaluation stage, we preserve all possible policy strategies and sample one policy based on the learned sampling parameter $\boldsymbol{\pi}$ at each iteration. Note that both the search and evaluation experiments are conducted on NVIDIA 2080Ti.

5.1 Performance Evaluation

In this section, we first report the performance of our method on CIFAR datasets and ImageNet. Then, we show the superiority of the searched policy against the default policies used by prior NAS works. Note that four parallel tests are conducted on each benchmark by jointly searching for NAS and AA under different random seeds.

Performance on CIFAR datasets. Table 2 shows the best and averaged performance over four parallel tests by searching under different random seeds. Compared with prior NAS works, our

Table 3: Comparison on ImageNet. The first block reports the performance of models transferred from CIFAR, and the second block reports performance of models directly searched on ImageNet.

Models	FLOPs (M)	Params (M)	Top-1 (%)	Cost GPU days	Way
AmoebaNet-A (Real et al., 2019)	555	5.1	74.5	3150	TF CIFAR10
NASNet-A (Zoph et al., 2018)	564	5.3	74.0	2000	TF CIFAR10
PNAS (Liu et al., 2018)	588	5.1	74.2	225	TF CIFAR10
DARTS (Liu et al., 2019)	574	4.7	73.3	0.4	TF CIFAR10
P-DARTS (Chen et al., 2019)	577	5.1	75.3	0.3	TF CIFAR100
FairDARTS-B (Chu et al., 2020)	541	4.8	75.1	0.4	TF CIFAR10
SNAS (Xie et al., 2019)	522	4.3	72.7	1.5	TF CIFAR10
PC-DARTS (Xu et al., 2020)	586	5.3	74.9	0.1	TF CIFAR10
GDAS (Dong & Yang, 2019)	581	5.3	74.0	0.2	TF CIFAR10
ROME (Wang et al., 2020a)	576	5.2	75.3	0.3	TF CIFAR10
DAAS (ours)	698	6.1	**76.6**	1.0	TF CIFAR10
PC-DARTS (Xu et al., 2020)‡	597	5.3	75.4	3.8	DS ImageNet
GDAS (Dong & Yang, 2019)	405	3.6	72.5	0.8	DS ImageNet
ROME (Wang et al., 2020a)	556	5.1	75.5	0.5	DS ImageNet
DAAS (ours)	661	5.9	**76.5**	1.8	DS ImageNet

method achieves 97.91% accuracy on CIFAR-10 and 84.80% accuracy on CIFAR-100, surpassing DARTS (Liu et al., 2019) by nearly 1%, as shown in Table 2. Additionally, the averaged performance is also reported. Our discovered architectures achieve state-of-the-art on both CIFAR-10 and CIFAR-100 datasets, showing that our joint searching method can stably improve the performance of NAS. Moreover, our method can discover effective architectures and policies in 1 GPU-days, showing the high efficiency of our joint searching algorithm.

Performance on ImageNet dataset. We also conduct experiments on ImageNet to verify the effectiveness of our method on complex datasets. We follow DARTS Liu et al. (2019) and transfer the cells searched on CIFAR-10 to ImageNet. Specifically, models are constructed by stacking 14 cells with 48 initial channels and are trained from scratch for 250 epochs by SGD with 0.5 initial learning rate. Table 3 shows that our transferred model achieves 76.6% top-1 accuracy on validation set, outperforming all prior differentiable NAS methods. Additionally, due to the high efficiency of our method, we can also directly search on ImageNet. In the search stage, we follow DADA Li et al. (2020) and randomly select 120 classes. A supernet is constructed by stacking 8 cells with 16 initial channels. We first warm up the operation weights and architecture parameters for 30 epochs and then jointly train architecture parameters and policy parameters for another 20 epochs. As shown in Table 3, DAAS achieves 76.5% top-1 accuracy on ImageNet validation set.

Superiority of the searched augmentation policies. Prior differentiable NAS methods (Liu et al., 2019; Dong & Yang, 2019; Wang et al., 2020a) manually design a default data augmentation policy for all networks. To verify the superiority of our joint searching framework against the NAS framework, we first search on our joint search space and then train the discovered architectures with default data augmentation policies in prior works and our searched policies for 600 epochs. The results are reported in Table 4, showing that suited augmentation policies can significantly improve the performance of architectures.

Table 4: Performance on CIFAR-10 of four architectures trained with default augmentation policies in prior NAS works and the policies discovered in our search space.

	Params (M)	Test Error (%) Default	Test Error (%) Searched
Arch-1	4.40	2.58	**2.09**
Arch-2	4.06	2.57	**2.30**
Arch-3	4.12	2.64	**2.36**
Arch-4	3.99	2.64	**2.20**

5.2 Ablation Study

Effectiveness of AA algorithm. Table 5 compares our AA algorithm with prior methods on CIFAR-10 and CIFAR-100 with various classic CNN networks, including Wide-ResNet (Zagoruyko & Komodakis, 2016), Shake-Shake (Gastaldi, 2017), and PyramidNet (Yamada et al., 2019). After the search process, we follow AA (Cubuk et al., 2019) and DADA (Li et al., 2020) by training Wide-

Table 5: Ablation study on AA algorithm. We compare with other AA methods on CIFAR-10 and CIFAR-100 with various classic networks. Our results are averaged over three parallel tests. †: The results are obtained from Cubuk et al. (2019); Lim et al. (2019). '-': The results are not provided by the prior work. WRN and SS are the shorthand of Wide-ResNet and Shake-Shake, respectively.

Dataset	Model	Baseline†	Cutout†	AA	PBA	Fast-AA	Faster-AA	DADA	DDAS	DAAS (ours)
CIFAR-10	WRN-40-2	5.3	4.1	3.7	-	3.6	3.7	3.6	-	**3.5**
CIFAR-10	WRN-28-10	3.9	3.1	**2.6**	2.6	2.7	**2.6**	2.7	2.7	**2.6**
CIFAR-10	SS(26 2x32d)	3.6	3.0	**2.5**	2.5	2.7	2.7	2.7	-	2.7
CIFAR-10	SS(26 2x96d)	2.9	2.6	2.0	2.0	2.0	2.0	2.0	2.0	**1.8**
CIFAR-10	SS(26 2x112d)	2.8	2.6	**1.9**	2.0	2.0	2.0	2.0	-	**1.9**
CIFAR-10	PyramidNet	2.7	2.3	**1.5**	**1.5**	1.8	-	1.7	-	1.6
CIFAR-100	WRN-40-2	26.0	25.2	**20.7**	-	**20.7**	21.4	20.9	-	21.0
CIFAR-100	WRN-28-10	18.8	18.4	17.1	16.7	17.3	17.3	17.5	**16.6**	16.9
CIFAR-100	SS(26 2x96d)	17.1	16.0	14.3	15.3	14.9	15.0	15.3	15.0	**14.6**
CIFAR-100	PyramidNet	14.0	12.2	**10.7**	10.9	11.9	-	11.2	-	11.0

ResNets for 200 epochs, Shake-Shakes for 1,800 epochs, and PyramidNets for 1,800 epochs. Our results are averaged over three parallel tests. Table 5 shows that our AA algorithm outperforms peer differentiable AA methods and even outperforms AA (Cubuk et al., 2019) on 4 benchmarks, whose search cost requires thousands of GPU-days while ours is fewer than 1 GPU-days.

Comparison between joint searching and independent searching for NAS and AA. To further verify our analysis on the strength of joint searching in Sec. 4.4, we compare with independent searching for NAS and AA. For the settings of independent search, we first search architectures by alternately training operation weights and architecture parameters for 50 epochs and derive the final network according to the discovered cells. Then, we search augmentation policies by alternately training operation weights and policy parameters for another 50 epochs with fixed architecture. Four parallel tests are conducted for both joint searching and independent searching, and the average performance is reported in Table 6, showing that joint searching outperforms independent searching by almost 0.1%. However, unlike the independent searching scheme with two separate stages where the network architecture is fixed when searching augmentation policies, joint searching is more efficient and can adjust architectures and policies simultaneously during the search process.

Table 6: Comparison of joint and independent searching for NAS and AA on CIFAR-10. Models are searched on our search space.

Search Mode	**Params** (M)	**Test Error** (%)
Joint	3.98	2.25
Independent	4.00	2.38

Ablation study by mixing up the discovered architectures and policies. Ablation studies are conducted to show the validity of the discovered combinations of architecture and policies. We randomly mix up three discovered combinations and full train the mixed combinations for 600 epochs on CIFAR-10. The results are reported in Table 7, showing that architectures achieve the best performance when the related policies are utilized.

Table 7: We randomly mix up the architecture and policies discovered by three parallel tests. The experiments are conducted on CIFAR-10.

	Policy-1	**Policy-2**	**Policy-3**
Arch-1	**2.09**	2.29	2.37
Arch-2	2.56	**2.30**	2.31
Arch-3	2.37	2.31	**2.20**

6 Conclusion

In this work, we have proposed an efficient differentiable joint searching algorithm named DAAS to discover efficient architecture and augmentation policies simultaneously. By constructing a bi-level optimization for joint searching, we further adopt the Gumbel technique to train architecture parameters and utilize the policy gradient algorithm to optimize augmentation policy parameters. Extensive experiments and ablation studies verify the effectiveness of our method. In particular, DAAS achieves 97.91% accuracy on CIFAR-10 and 76.6% top-1 accuracy on ImageNet in only 1 GPU-days' search. This work shows the superiority of joint searching for NAS and AA, implying that NAS evaluation should consider appropriate data augmentation policies.

REFERENCES

Xiangning Chen and Cho-Jui Hsieh. Stabilizing differentiable architecture search via perturbation-based regularization. In *ICML*, 2020.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive Differentiable Architecture Search: Bridging the Depth Gap between Search and Evaluation. In *ICCV*, 2019.

Xiangxiang Chu, Tianbao Zhou, Bo Zhang, and Jixiang Li. Fair darts: Eliminating unfair advantages in differentiable architecture search. *ECCV*, 2020.

Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. DARTS-: robustly stepping out of performance collapse without indicators. In *ICLR*. OpenReview.net, 2021.

Ekin D. Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V. Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, pp. 113–123. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00020.

Xiaoliang Dai, Alvin Wan, Peizhao Zhang, Bichen Wu, Zijian He, Zhen Wei, Kan Chen, Yuandong Tian, Matthew Yu, Peter Vajda, and Joseph E. Gonzalez. Fbnetv3: Joint architecture-recipe search using neural acquisition function. *CoRR*, abs/2006.02049, 2020.

Xuanyi Dong and Yi Yang. Searching for a Robust Neural Architecture in Four GPU Hours. In *CVPR*, pp. 1761–1770, 2019.

Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V. Le. Autohas: Differentiable hyper-parameter and architecture search. *CoRR*, abs/2006.03656, 2020.

Xavier Gastaldi. Shake-shake regularization of 3-branch residual networks. In *ICLR*. OpenReview.net, 2017.

Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm (eds.), *ECCV*, volume 12370 of *Lecture Notes in Computer Science*, pp. 1–16. Springer, 2020. doi: 10.1007/978-3-030-58595-2_1.

Taiga Kashima, Yoshihiro Yamada, and Shunta Saito. Joint search of data augmentation policies and network architectures. *CoRR*, abs/2012.09407, 2020.

Aaron Klein and Frank Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *CoRR*, abs/1905.04970, 2019.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning Multiple Layers of Features from Tiny Images. Technical report, Citeseer, 2009.

Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy M. Hospedales, Neil Martin Robertson, and Yongxing Yang. DADA: differentiable automatic data augmentation. *CoRR*, abs/2003.03780, 2020.

Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In Hanna M. Wallach, Hugo Larochelle, Alina Beygelzimer, Florence d'Alché-Buc, Emily B. Fox, and Roman Garnett (eds.), *NeurIPS*, pp. 6662–6672, 2019.

Aoming Liu, Zehao Huang, Zhiwu Huang, and Naiyan Wang. Direct differentiable augmentation search. *CoRR*, abs/2104.04282, 2021.

Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive Neural Architecture Search. In *ECCV*, pp. 19–34, 2018.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable Architecture Search. In *ICLR*, 2019.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient Neural Architecture Search via Parameter Sharing. In *ICML*, 2018.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, volume 33, pp. 4780–4789, 2019.

Dimitrios Stamoulis, Ruizhou Ding, Di Wang, Dimitrios Lymberopoulos, Bodhi Priyantha, Jie Liu, and Diana Marculescu. Single-Path NAS: Designing Hardware-Efficient ConvNets in less than 4 Hours. *ECML PKDD*, 2019.

Xiaoxing Wang, Xiangxiang Chu, Yuda Fan, Zhexi Zhang, Xiaolin Wei, Junchi Yan, and Xiaokang Yang. ROME: robustifying memory-efficient NAS via topology disentanglement and gradients accumulation. *CoRR*, abs/2011.11233, 2020a.

Xiaoxing Wang, Chao Xue, Junchi Yan, Xiaokang Yang, Yonggang Hu, and Kewei Sun. Mergenas: Merge operations into one for differentiable architecture search. In *IJCAI*, 2020b.

Colin White, Willie Neiswanger, and Yash Savani. BANANAS: bayesian optimization with neural architectures for neural architecture search. In *AAAI*, pp. 10293–10301. AAAI Press, 2021.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic Neural Architecture Search. In *ICLR*, 2019.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. In *ICLR*, 2020.

Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise. Shakedrop regularization for deep residual learning. *IEEE Access*, 7:186126–186136, 2019. doi: 10.1109/ACCESS.2019.2960566.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Richard C. Wilson, Edwin R. Hancock, and William A. P. Smith (eds.), *BMVC*. BMVA Press, 2016.

Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *CoRR*, abs/1807.06906, 2018.

Arber Zela, Thomas Elsken, Tonmoy Saikia, Yassine Marrakchi, Thomas Brox, and Frank Hutter. Understanding and robustifying differentiable architecture search. In *ICLR*, 2020.

Kaichen Zhou, Lanqing Hong, Shoukang Hu, Fengwei Zhou, Binxin Ru, Jiashi Feng, and Zhenguo Li. DHA: end-to-end joint optimization of data augmentation policy, hyper-parameter and architecture. *CoRR*, abs/2109.05765, 2021.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning Transferable Architectures for Scalable Image Recognition. In *CVPR*, volume 2, 2018.