Multi-Agent Automated Machine Learning

Zhaozhi Wang[123][†], Kefan Su[1], Jian Zhang[4], Huizhu Jia[1], Qixiang Ye[23], Xiaodong Xie[1], Zongqing Lu[1][‡]

[1]Peking University   [2]Peng Cheng Lab   [3]University of Chinese Academy of Sciences   [4]Huawei

Abstract

In this paper, we propose multi-agent automated machine learning (MA2ML) with the aim to effectively handle joint optimization of modules in automated machine learning (AutoML). MA2ML takes each machine learning module, such as data augmentation (AUG), neural architecture search (NAS), or hyper-parameters (HPO), as an agent and the final performance as the reward, to formulate a multi-agent reinforcement learning problem. MA2ML explicitly assigns credit to each agent according to its marginal contribution to enhance cooperation among modules, and incorporates off-policy learning to improve search efficiency. Theoretically, MA2ML guarantees monotonic improvement of joint optimization. Extensive experiments show that MA2ML yields the state-of-the-art top-1 accuracy on ImageNet under constraints of computational cost, e.g., 79.7%/80.5% with FLOPs fewer than 600M/800M. Extensive ablation studies verify the benefits of credit assignment and off-policy learning of MA2ML.

1. Introduction

Automated machine learning (AutoML) aims to find high-performance machine learning (ML) pipelines without human effort involvement. The main challenge of AutoML lies in finding optimal solutions in huge search spaces.

In recent years, reinforcement learning (RL) has been validated to be effective to optimize individual AutoML modules, such as data augmentation (AUG) [4], neural architecture search (NAS) [25, 53, 54], and hyper-parameter optimization (HPO) [38]. However, when facing the huge search space (Figure 1 *left*) and the joint optimization of these modules, the efficiency and performance challenges remain.

Through experiments, we observed that among AutoML modules there exists a cooperative relationship that facilies the joint optimization of modules. For example, a small network (ResNet-34) with specified data augmentation and op-
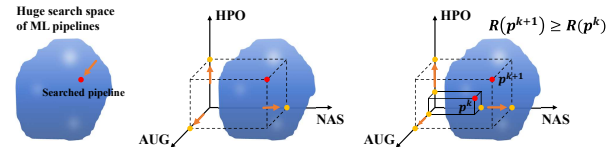
Figure 1. Search spaces of machine learning pipelines. *Left*: single agent controls all modules, and the huge search space makes it ineffective to learn. *Mid*: each agent controls one module, and the learning difficulty is reduced by introducing MA2ML. *Right*: MA2ML guarantees monotonic improvement of the searched pipeline, where p^k and $R(p^k)$ denote the k-th searched pipeline and its expected performance, respectively.

timized hyper-parameters significantly outperforms a large one (ResNet-50) with default training settings (76.8% vs. 76.1%). In other words, good AUG and HPO alleviate the need for NAS to some extent. Accordingly, we propose *multi-agent automated machine learning* (MA2ML), which explores the cooperative relationship towards joint optimization of ML pipelines. In MA2ML, ML modules are defined as RL agents (Figure 1 *mid*), which take actions to jointly maximize the reward, so that the training efficiency and test accuracy are significantly improved. Specially, we introduce credit assignment to differentiate the contribution of each module, such that all modules can be simultaneously updated. To handle both continuous (*e.g.*, learning rate) and discrete (*e.g.*, architecture) action spaces, MA2ML employs a multi-agent actor-critic method, where a centralized Q-function is learned to evaluate the joint action. Besides, to further improve search efficiency, MA2ML adopts off-policy learning to exploit historical samples for policy updates.

MA2ML is justified theoretically and experimentally. Theoretically, we prove that MA2ML guarantees monotonic policy improvement (Figure 1 *right*), *i.e.*, the performance of the searched pipeline monotonically improves in expectation. This enables MA2ML to fit the joint optimization problem and be adaptive to all modules in the ML pipeline, potentially achieving full automation. Experimentally, we take the combination of individual RL-based modules to form MA2ML-Lite, and compare their perfor-

mance on ImageNet [31] and CIFAR-10/100 [20] datasets. To better balance performance and computational cost, we add constraints of FLOPs in the experiment on ImageNet. Experiments show that MA2ML substantially outperforms MA2ML-Lite $w.r.t.$ both accuracy and sample efficiency, and MA2ML achieves remarkable accuracy compared with recent methods.

Our contributions are summarized as follows:

- We propose MA2ML, which utilizes credit assignment to differentiate the contributions of ML modules, providing a systematic solution for the joint optimization of AutoML modules.

- We prove the monotonic improvement of module policies, which enables to MA2ML fit the joint optimization problem and be adaptive to various modules in the ML pipeline.

- MA2ML yields the state-of-the-art performance under constraints of computational cost, $e.g.$, 79.7%/80.5% on ImageNet, with FLOPs fewer than 600M/800M, validating the superiority of the joint optimization of MA2ML.

2. Related work

Single-module Optimization. AutoML has been applied to different modules in the ML pipeline, such as AUG, NAS, and HPO. For AUG, AutoAugment [4] is an RL-based method to search augmentation policies. Fast AA [21] speeds up the search with less computing cost through Bayesian optimization. Faster AA [10] is a differentiable method to optimize augmentation policy, which reduces computing cost further. PBA [12] also reduces search cost by population-based training. Adversarial AA [49] uses the idea of GAN [8] to generate the hard augmentation policy for the model to improve robustness.

For NAS, there are three main types of methods, evolutionary methods [29, 30, 45], RL-based methods [25, 27, 35, 53, 54], and differentiable gradient-based methods [3, 11, 22, 44]. Evolutionary methods optimize the search through selection, recombination, and mutation in the neural architecture population. RL-based methods regard the neural architecture as a black box and the final accuracy as the reward. They typically use RL algorithms to solve the optimization problem. Differentiable gradient-based methods design a continuous representation for neural architecture space and make NAS a differentiable problem, which greatly improves search efficiency, compared with the other two types.

For HPO, black-box methods [1, 15] have been utilized for a long time. Meanwhile, some methods like [16, 41] use multi-fidelity ways to accelerate optimization through the evaluation on proxy tasks. Recently, gradient-based methods [23, 24, 26, 32] optimize hyper-parameters by calculat-

ing gradient respect to them, which reduces computing cost substantially.

The above methods for single module optimization leave other modules of the ML pipeline fixed during search, either by expert knowledge or empirical setting, which may not be optimal when combining them. Joint optimization of multiple modules is a more plausible way to advance the ML pipeline.

Joint Optimization. Recent studies [5, 6, 19, 46] search for NAS and HPO jointly through RL or training an accuracy predictor, or [17, 40] consider the joint optimization of AUG and NAS, and obtain convincing results by bi-level gradient-based optimization. DHA [52] explores the joint optimization for AUG, NAS, and HPO through one-level optimization, which is achieved in a differentiable manner by optimizing a compressed lower dimensional feature space for NAS. However, the alternate optimization of the gradient-based method may get stuck at the non-stationary point with limited-order gradient descent.

Unlike existing joint optimization methods for AutoML, our MA2ML aims to update all modules, as well as guarantee the convergence of joint optimization.

3. The Proposed Method

MA2ML is a general framework for joint optimization in AutoML and can be applied to any arbitrary combination of different modules. For ease of presentation, in the following, we use the joint optimization of AUG, NAS, and HPO for image classification tasks as an example to elaborate MA2ML.

3.1. Action Space

Before formulating the joint optimization as an MARL problem, we first define an action space for AUG, NAS, and HPO. In AUG, we adopt the setting in AutoAugment [4]. In NAS, we utilize the search space in NASNet [54] and FBNetV3 [5] on different datasets to show that MA2ML is naturally agnostic to search spaces. In HPO, we design a search space for hyper-parameters, including learning rate, weight decay, etc.

For each module, we implement a controller to sample different choices from the action space sequentially. Note that MA2ML can be easily extended to support more modules in the ML pipeline, such as loss function search and mix-precision configuration search, by simply implementing the module by a controller and adding it to the framework.

3.2. Joint Optimization

Assuming there are n modules for the AutoML pipeline, each module is modeled as an agent. For each agent i, it takes a random state S_i as initial input and outputs the se-
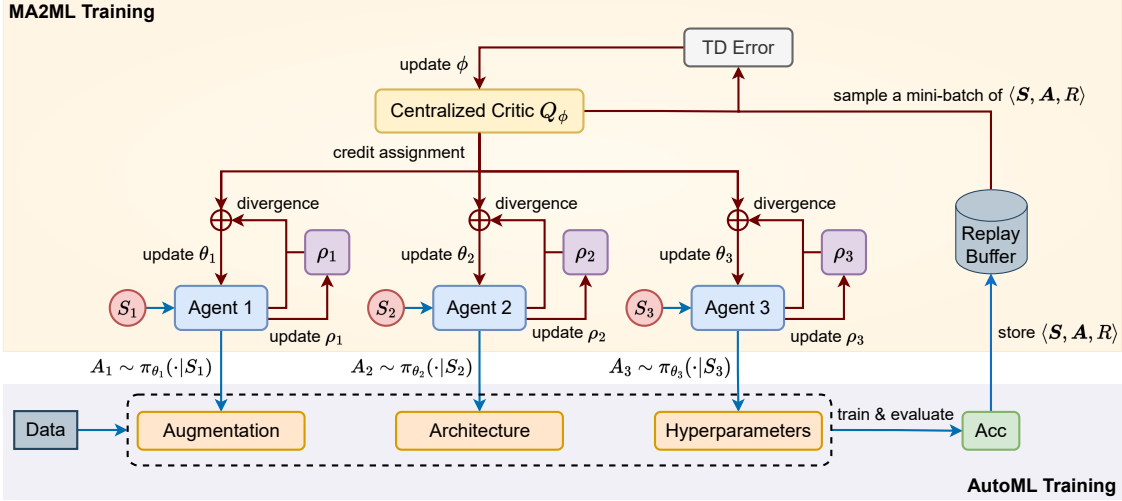
Figure 2. The framework of joint optimization with MA2ML. For AutoML training (lower panel), the ML pipeline is formed by the actions sampled from the policies of agents, then it is deployed for training on the dataset and to obtain the accuracy (reward). After that, the tuple $\langle S, A, R \rangle$ is stored in the replay buffer. For MA2ML training (upper panel), a mini-batch of $\langle S, A, R \rangle$ are sampled from the replay buffer to update the critic, policies, and target policies.

lected action A_i in the action space sampled from its policy π_i, which is denoted as $A_i \sim \pi_{\theta_i}(\cdot|S_i)$ where the policy is parameterized by θ_i. After all actions are determined, we train the network with the determined pipeline setting on the image classification task for epochs and evaluate top-1 accuracy on the validation set. We take the top-1 accuracy as the reward R for all the agents. The objective is to maximize the expected reward R, represented by $J(\Theta)$:

$$J(\Theta) = \mathbb{E}_{\pi_\Theta(A|S)}[R], \quad (1)$$

where $\pi_\Theta(A|S) \triangleq \prod_{i=1}^{n} \pi_{\theta_i}(A_i|S_i)$ is the joint policy, and Θ, S and A respectively denote the gather of θ_i, S_i and action A_i of all agents (we may drop Θ or θ for brevity if there is no confusion). Consequently, we transform the joint optimization of the ML pipeline as an MARL problem, then we can rely on MARL methods to solve the problem.

3.3. MA2ML-Lite

For the MARL problem defined in (1), one method to solve it is to learn a policy individually for each module of AUG, NAS, and HPO. In the `AutoML` training phase, we sample an action according to the policy of each agent, and train the model according to the searched ML pipeline to calculate the reward R (top-1 accuracy). In the `MA2ML` training phase, we use the reward R to directly guide the update of each agent's policy. REINFORCE [34] which can handle both discrete and continuous action, is used to calculate policy gradient, which is formulated as

$$\nabla_{\theta_i} J(\Theta) = \mathbb{E}_{\pi(A|S)}[\nabla_{\theta_i} \log \pi_{\theta_i}(A_i|S_i)(R - b)], \quad (2)$$

where b is an exponential moving average of the previous rewards. The moving average is beneficial for reducing the variance of gradient estimate. Then, the policy of each agent i is updated by gradient ascent using (2). Although $J(\Theta)$ depends on the joint policy of all agents, their policies are not *really* jointly optimized by (2). In other words, for the obtained R, it is not able to tell the contribution of each agent, and thus cannot explicitly update their policies towards better ones. We term this RL method MA2ML-Lite, which is the lite version of MA2ML.

3.4. Credit Assignment

As AUG, NAS, and HPO jointly determine the final performance, if we use the reward R to directly feedback to each agent as MA2ML-Lite does, each agent cannot distinctly determine whether the performed action is good or not. This poses a great challenge to the learning of policies.

To solve the challenge, we train a centralized critic to learn the action-value function (Q-function) and add a counterfactual baseline, inspired by [7]. The counterfactual baseline marginalizes out an agent's action, while keeping other agents' actions fixed. The difference between the value of taken actions and the counterfactual baseline measures the marginal contribution of each agent. The difference is used to update each agent's policy.

The centralized critic is denoted as $Q(S, A)$ or $Q(S, A_i, A_{-i})$, where A_{-i} represents the joint action of all agents except agent i. The counterfactual baseline denoted as $b(S, A_{-i})$ for each agent i is calculated as

$$b(S, A_{-i}) = \mathbb{E}_{A_i \sim \pi_{\theta_i}}[Q(S, A_i, A_{-i})]. \quad (3)$$

In AutoML, considering the large action space of different modules and to reduce the computation cost, we can use the sampled action A_i as input, while keeping other actions fixed, to approximate the counterfactual baseline.

3.5. Off-Policy Learning

REINFORCE is an on-policy method, meaning that current policy can only be updated using the experiences obtained by itself. Considering it takes much time to train the model to receive top-1 accuracy, on-policy methods are very *inefficient*. Thereby, we incorporate off-policy learning, such that the current policy can also use historical experiences generated during learning for updates, which helps a lot in improving search efficiency.

To enable off-policy learning for actor-critic methods, a popular method is maximum-entropy RL, like SAC [9], which adds an entropy regularization in the objective as

$$
\begin{aligned}
J(\Theta) &= \mathbb{E}_{\boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})}[Q(\boldsymbol{S}, \boldsymbol{A}) - \lambda \log \boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})] \\
&= \mathbb{E}_{\boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})}[Q(\boldsymbol{S}, \boldsymbol{A})] + \lambda \mathcal{H}(\boldsymbol{\pi}(\cdot|\boldsymbol{S})),
\end{aligned} \tag{4}
$$

where $\mathcal{H}(\boldsymbol{\pi}(\cdot|\boldsymbol{S}))$ represents the entropy of $\boldsymbol{\pi}(\cdot|\boldsymbol{S})$, and λ is the coefficient of entropy regularization. However, the entropy regularization biases the converged policy, *i.e.*, the converged policy maximizes (4) instead of the original objective (1).

To eliminate the bias of the converged policy as well as implement off-policy learning, inspired by [33], we maintain a target policy ρ_i for each agent, in addition to the original policy π_i, and add the divergence regularization between π_i and ρ_i in the reward,

$$
\begin{aligned}
J(\Theta) &= \mathbb{E}_{\boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})}[Q(\boldsymbol{S}, \boldsymbol{A}) - \lambda \log \frac{\boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})}{\boldsymbol{\rho}(\boldsymbol{A}|\boldsymbol{S})}] \\
&= \mathbb{E}_{\boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})}[Q(\boldsymbol{S}, \boldsymbol{A})] - \lambda D_{\mathrm{KL}}(\boldsymbol{\pi}(\cdot|\boldsymbol{S})\|\boldsymbol{\rho}(\cdot|\boldsymbol{S})),
\end{aligned} \tag{5}
$$

where $\boldsymbol{\rho} \triangleq \prod_{i=1}^{n} \rho_i$ denotes the joint target policy and $D_{\mathrm{KL}}(\boldsymbol{\pi}(\cdot|\boldsymbol{S})\|\boldsymbol{\rho}(\cdot|\boldsymbol{S}))$ denotes the KL divergence between two distributions, $\boldsymbol{\pi}(\cdot|\boldsymbol{S})$ and $\boldsymbol{\rho}(\cdot|\boldsymbol{S})$. The divergence regularization is beneficial for exploration and stable policy improvement. More importantly, based on divergence policy iteration [33] we can further derive the theoretical result as follows.

Theorem 1. By iteratively applying divergence policy iteration and taking $\boldsymbol{\pi}^k$ as the joint target policy $\boldsymbol{\rho}^{k+1}$, the policy sequence of $\{\boldsymbol{\pi}^k\}$ converges and monotonically improves upon the *original* optimization problem.

Proof. Let $J_{\mathrm{init}}(\boldsymbol{\pi})$ be the original optimization objective and $J_{\mathrm{reg}}(\boldsymbol{\pi}, \boldsymbol{\rho})$ be the optimization objective with divergence regularization given the fixed target policy $\boldsymbol{\rho}$. Then

they have the following definitions and relations.

$$
J_{\mathrm{init}}(\boldsymbol{\pi}) = \mathbb{E}_{\boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})}[R], \tag{6}
$$

$$
J_{\mathrm{reg}}(\boldsymbol{\pi}, \boldsymbol{\rho}) = \mathbb{E}_{\boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})}[R - \lambda \log \frac{\boldsymbol{\pi}(\boldsymbol{A}|\boldsymbol{S})}{\boldsymbol{\rho}(\boldsymbol{A}|\boldsymbol{S})}], \tag{7}
$$

$$
J_{\mathrm{reg}}(\boldsymbol{\pi}, \boldsymbol{\rho}) = J_{\mathrm{init}}(\boldsymbol{\pi}) - \lambda D_{\mathrm{KL}}(\boldsymbol{\pi}(\cdot|\boldsymbol{S})\|\boldsymbol{\rho}(\cdot|\boldsymbol{S})). \tag{8}
$$

Iteratively applying divergence policy iteration and taking $\boldsymbol{\pi}^k$ as the joint target policy $\boldsymbol{\rho}^{k+1}$ can be formulated as

$$
\boldsymbol{\pi}^{k+1} = \arg \max_{\boldsymbol{\pi}} J_{\mathrm{reg}}(\boldsymbol{\pi}, \boldsymbol{\rho}^{k+1}) = \arg \max_{\boldsymbol{\pi}} J_{\mathrm{reg}}(\boldsymbol{\pi}, \boldsymbol{\pi}^k). \tag{9}
$$

Substituting (9) to (8), we have

$$
\begin{aligned}
J_{\mathrm{init}}(\boldsymbol{\pi}^{k+1}) &\geq J_{\mathrm{init}}(\boldsymbol{\pi}^{k+1}) - \lambda D_{\mathrm{KL}}(\boldsymbol{\pi}^{k+1}(\cdot|\boldsymbol{S})\|\boldsymbol{\pi}^k(\cdot|\boldsymbol{S})) \\
&= J_{\mathrm{reg}}(\boldsymbol{\pi}^{k+1}, \boldsymbol{\pi}^k) \\
&\geq J_{\mathrm{reg}}(\boldsymbol{\pi}^k, \boldsymbol{\pi}^k) \\
&= J_{\mathrm{init}}(\boldsymbol{\pi}^k).
\end{aligned} \tag{10}
$$

The first inequality is from the non-negativity of the KL-divergence and the second inequality is from the definition of $\boldsymbol{\pi}^{k+1}$ in (9).

The conclusion $J_{\mathrm{init}}(\boldsymbol{\pi}^{k+1}) \geq J_{\mathrm{init}}(\boldsymbol{\pi}^k)$ (defined by (10)) means the original objective monotonically improves though we optimize the objective with divergence regularization.

Moreover, the reward R is bounded, so the original objective $J_{\mathrm{init}}(\boldsymbol{\pi})$ is bounded. The boundness and monotonic improvement of the sequence $J_{\mathrm{init}}(\boldsymbol{\pi}^k)$ guarantees the convergence of the original objective sequence. $\qquad\square$

Remark 1. Theorem 1 guarantees the monotonic policy improvement and sub-optimum in the original optimization objective (1), not just the divergence-regularized objective as in [33]. This is crucial for AutoML as we can control the search of MA2ML by λ, without deteriorating the optimality of the searched ML pipeline. Therefore, Theorem 1 lays the theoretical foundation of MA2ML.

3.6. Training

In the AutoML training phase, we sample actions $\boldsymbol{A}$ from $\boldsymbol{\pi}$ to obtain the ML pipeline and train the model according to the pipeline to receive top-1 accuracy as the reward R. After that, we store the experience $\langle \boldsymbol{S}, \boldsymbol{A}, R \rangle$ in the replay buffer $\mathcal{D}$.

In the MA2ML training phase, we sample a mini-batch $\mathcal{B}$ of experiences from the replay buffer $\mathcal{D}$. We first update the centralized critic, parametrized by ϕ, by gradient descent of the loss function $\mathcal{L}_Q$ as

$$
\begin{aligned}
\mathcal{L}_Q &= \mathbb{E}[(Q_\phi(\boldsymbol{S}, \boldsymbol{A}) - R)^2], \\
\phi &= \phi - \eta_\phi \nabla_\phi \mathcal{L}_Q,
\end{aligned} \tag{11}
$$

Algorithm 1 MA2ML

1: initialize the critic ϕ, and the policy θ_i and target policy $\bar{\theta}_i$ for each agent i
2: **for** $iter = 1$ to max_iter **do**
3:     receive initial state S_i for each agent i
4:     sample $A_i \sim \pi_{\theta_i}(\cdot|S_i)$ for each agent i
5:     determine the pipeline setting according to $\boldsymbol{A}$
6:     train the pipeline for a number of epochs on dataset
7:     obtain the top-1 accuracy on the validation set as R
8:     store $\langle \boldsymbol{S}, \boldsymbol{A}, R \rangle$ in the replay buffer $\mathcal{D}$
9:     sample a mini-batch $\mathcal{B}$ from $\mathcal{D}$
10:    update the critic ϕ by (11)
11:    update each agent's policy θ_i by (13)
12:    update each agent's target policy $\bar{\theta}_i$ by (14)
13: **end for**
14: choose top-k pipelines in terms of top-1 accuracy
15: retrain the pipelines till convergence
16: obtain the highest accuracy on the test set

Figure 3. Comparison of MA2ML with other AutoML methods on ImageNet.

where η_ϕ denotes the learning rate of ϕ. Then, we update the policy and the target policy for each agent. The counterfactual baseline with divergence regularization is modified as

$$b(\boldsymbol{S}, A_{-i}) = \mathbb{E}_{A_i \sim \pi_i}[Q(\boldsymbol{S}, A_i, A_{-i}) - \lambda \log \frac{\pi(\boldsymbol{A}|\boldsymbol{S})}{\rho(\boldsymbol{A}|\boldsymbol{S})}]. \quad (12)$$

The gradient for the policy of each agent is calculated and the policy is updated by gradient ascent as

$$\begin{aligned}
\nabla_{\theta_i} \mathcal{L}_{\pi_{\theta_i}} = \mathbb{E}[\nabla_{\theta_i} \log \pi_{\theta_i}(A_i|S_i)(Q(\boldsymbol{S}, \boldsymbol{A}) \\
- \lambda \log \frac{\pi_{\theta_i}(A_i|S_i)}{\rho_i(A_i|S_i)} - \mathbb{E}_{A_i \sim \pi_i}[Q(\boldsymbol{S}, \boldsymbol{A})] \\
+ \lambda D_{\mathrm{KL}}(\pi_i(\cdot|S_i)\|\rho_i(\cdot|S_i)))], \\
\theta_i = \theta_i + \eta_\theta \nabla_{\theta_i} \mathcal{L}_{\pi_{\theta_i}},
\end{aligned} \quad (13)$$

where η_θ is the learning rate of the policy. The target policy of each agent is parameterized by $\bar{\theta}_i$, and is updated as

$$\bar{\theta}_i = (1 - \tau)\bar{\theta}_i + \tau\theta_i, \quad (14)$$

where τ is an empirically determined parameter.

For completeness, the framework of MA2ML is depicted in Figure 2 and the learning algorithm of MA2ML is summarized in Algorithm 1.

4. Experiments

Image classification experiments of MA2ML and MA2ML-Lite are performed on ImageNet while ablation studies are carried on CIFAR-10/100[1].

4.1. ImageNet

Setting. Considering the large search cost, we adopt the setting in FBNetV3 [5], *i.e.*, randomly sampling 200 classes from the entire dataset as the training set and taking out 10K images from it to form a validation set. The sub-set with 200 classes is termed ImageNet-200.

The action space mentioned in Section 3.1 is employed for the experiment on ImageNet. For AUG, each pipeline is configured with an augmentation policy, which consists of 25 sub-policies. Each sub-policy contains two augmentation operations, which are determined by three dimensions including operation type, probability, and magnitude. Operation type has 15 options, probability ranges from 0.0 to 1.0 with step 0.1, and magnitude consists of 10 levels.

For NAS and HPO, we use the search space in FBNetV3. We search for input resolution, kernel size, expansion, number of channels per layer, and depth as architecture configuration, which is based on the inverted residual block [13]. For HPO, we search for the optimizer type, initial learning rate, weight decay, mixup ratio [47], drop out ratio, stochastic depth drop ratio [14], and whether to use exponential moving average (EMA) [18]. Overall, AUG, NAS, and HPO contain 10^{160}, 10^{17}, 10^7 candidates, respectively.

To better balance the performance and computational cost of the searched AutoML pipeline, constraints of FLOPs are added. We adopt the multi-objective reward function $R = \mathrm{Acc}(m) \times \left[\frac{\mathrm{FLOPs}(m)}{\mathrm{FLOPs\_constraint}}\right]^w$ in [36], where $\mathrm{Acc}(m)$ and $\mathrm{FLOPs}(m)$ respectively denote top-1 accuracy and floating point operations of the model, $\mathrm{FLOPs\_constraint}$ represents the constraint of FLOPs, and w is used to adjust the tradeoff between accuracy and FLOPs. We set $w = -0.07$ and $\mathrm{FLOPs\_constraint} = 600M/900M$ for two trials using MA2ML. Besides, the MA2ML-Lite runs $\mathrm{FLOPs\_constraint} = 600M$ and compared with MA2ML[2].

We generate 24 ML pipelines for each batch, and train

Table 1. Top-1 accuracy (%) and FLOPs of state-of-the-art AutoML methods on ImageNet, where NARS denotes neural architecture-recipe search. All compared models have computational cost close to 600M FLOPs for a fair comparison.

Model	Acc (%)	FLOPs (M)	Method	Search Modules
DARTS [22]	73.3	574	gradient	NAS
NASNet [54]	74.0	564	RL	NAS
MiLeNAS [11]	75.3	584	gradient	NAS
RMI-NAS [50]	75.3	657	Random Forest	NAS
RegNetY [28]	75.5	600	pop. param.*	NAS
ROME [39]	75.5	556	gradient	NAS
AmoebaNet-C [29]	75.7	570	evolution	NAS
PC-DARTS [43]	75.8	597	gradient	NAS
BaLeNAS [48]	75.8	597	gradient	NAS
ISTA-NAS [44]	76.0	638	gradient	NAS
Shapley-NAS [42]	76.1	582	gradient	NAS
DAAS [40]	76.6	698	gradient	AUG+NAS
DHA [52]	77.4	-	gradient	AUG+NAS+HPO
MIGO-NAS [51]	78.3	595	MIGO	NAS
OFA† [2]	79.0	595	gradient	NAS
EfficientNet-B1 [37]	79.1	700	RL	NAS
FBNetV3‡ [5]	79.2	550	NARS	NAS+HPO
L^{2}NAS [25]	79.3	618	RL	NAS
MA2ML-A	79.3	490	MARL	AUG+NAS+HPO
MA2ML-B	**79.7**	596	MARL	AUG+NAS+HPO

*Population parameterization. †Results are given in [25] without distillation.

‡Results are reproduced according to 600M FLOPs constraint without distillation.

Table 2. Comparison of MA2ML-A/B/C with MA2ML-Lite-A/B/C on ImageNet.

Model	A/B/C	
	Acc (%)	FLOPs (M)
MA2ML-Lite	78.6 / 79.1 / 79.5	498 / 597 / 697
MA2ML	79.3 / 79.7 / 80.1	490 / 596 / 694

each model for 100 epochs on ImageNet-200 to receive top-1 accuracy. For one trial, 1992 pipelines are searched on ImageNet-200 in total. After the search, 20 pipelines of the highest accuracy are retrained under the given constraint on the entire ImageNet dataset for 400 epochs. The pipeline of the highest accuracy is selected as the search result. For the search results obtained by MA2ML and MA2ML-Lite with FLOPs_constraint = 600M, we retrain those with FLOPs fewer than 500M/600M/700M and pick the best as MA2ML-A/B/C and MA2ML-Lite-A/B/C. For the search results obtained with FLOPs_constraint = 900M, we retrain those with FLOPs fewer than 800M/900M/1G and pick the best as MA2ML-D/E/F.

Performance. On ImageNet, MA2ML-A/B/C/D/E/F achieves 79.3%, 79.7%, 80.1%, 80.5%, 80.7%, and 81.1%

top-1 accuracy. Figure 3 gives the accuracy under different FLOPs of MA2ML and other methods. Since we do not use distillation in our experiments, results achieved by implementing distillation are not drawn here. The curve of MA2ML is plotted with the results of MA2ML-A/B/C/D/E/F, which is the highest among all the compared methods with a large margin. Figure 3 also indicates that MA2ML can always perform better with larger constraints of computation cost.

In Table 1, we compared MA2ML-A, MA2ML-B, and other methods with similar FLOPs. MA2ML-A outperforms all listed AutoML methods with only 490M FLOPs (L^{2}NAS has the same performance but with much larger FLOPs), and MA2ML-B with 596M FLOPs obtains the performance with large improvement compared with the results of other methods close to 600M FLOPs.

In Table 2, MA2ML-Lite-A/B/C achieves 78.6%, 79.1% and 79.5%. The comparison shows that MA2ML outperforms MA2ML-Lite by 0.6%, which is ***attributed to credit assignment and off-policy learning of MA2ML***.

Figure 4 presents the average reward curves of top-20 pipelines and the scatter plot of average reward in each batch of MA2ML and MA2MLs-Lite on ImageNet-200 (FLOPs_constraint = 600M). The curves illustrate that the superiority of MA2ML at the early stage is not signif-
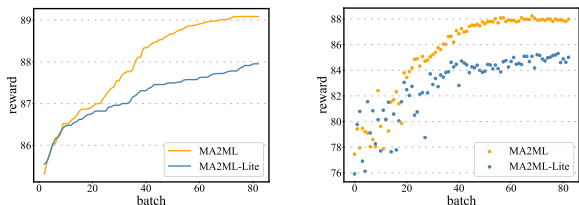
Figure 4. Learning patterns of MA2ML and MA2ML-Lite on ImageNet-200 (FLOPs_constraint = 600M). *Left*: the average reward curves of top-20 pipelines in terms of patch numbers. *Right*: the scatter plot for average rewards of pipelines in each batch.

icant, but surpasses MA2ML-Lite a lot in the mid-term of the search and keeps the lead till the end. The scatter depicts the batch-level average reward, where MA2ML also greatly outperforms MA2ML-Lite from the mid-term. Moreover, the batch-level average reward of MA2ML is also more stable than MA2ML-Lite, which demonstrates *the benefit of monotonic policy improvement*.

MA2ML takes almost the same search cost as MA2ML-Lite and other RL-based methods, because the major part of search cost is the training time to receive top-1 accuracy and the cost of MARL in MA2ML is negligible. So the performance gain of MA2ML over MA2ML-Lite takes no more computation cost. The performance on ImageNet proves the effectiveness and generality of MA2ML, and jointly searching for ML modules is necessary.

4.2. CIFAR

Setting. We also use the action space mentioned in Section 3.1 on CIFAR-10 and CIFAR-100. For AUG, we generate 25 augmentation sub-policies, which is the only difference from AUG module on ImageNet. For NAS, we utilize the search space in NASNet, and search for a normal cell and a reduction cell. We construct the whole network with 17 cells, and the 5th and 11th are designed as reduction cells. A cell is composed of 5 sequential blocks. To create a new block, we search for two existing blocks, choose two operations and apply them to the selected blocks, and select a way to combine the outputs of the last step. The operation space consists of 13 options and the combination contains element-wise addition and concatenation. For HPO, we choose SGD as the optimizer, fix the batch size, and search for warmup learning rate and weight decay, since these two hyper-parameters affect much the performance by experience. We discretize the search space of them into 4 concrete values and use 10 steps to determine them on average. Overall, AUG, NAS and HPO space contains 10^{32}, 10^{37}, 10^{12} candidates, respectively.

On CIFAR-10 and CIFAR-100, we sample 24 ML pipelines for each batch according to the policies of agents, and train the model using each pipeline for 12 epochs to re-

ceive top-1 accuracy as the reward. We have in total 5016 pipelines on CIFAR-10 and 4008 on CIFAR-100 and choose the top 30 pipelines in terms of top-1 accuracy to train for 600 epochs. As the parameter size is not constrained on CIFAR-10 and CIFAR-100, the searched models tend to be larger than those of the compared methods. The ML pipeline of the highest top-1 accuracy on the test set is selected to train the final models. The pipeline is trained for 5 trials for both MA2ML and MA2ML-Lite. The mean and standard deviation are reported as the performance.

Results on CIFAR-10/100. MA2ML respectively achieves $97.77 \pm 0.07\%$ and $85.08 \pm 0.14\%$ top-1 accuracies on CIFAR-10 and CIFAR-100, higher than those ($97.70 \pm 0.10\%$ and $84.80 \pm 0.12\%$) of MA2ML-Lite. Table 3 shows the results achieved by MA2ML, MA2ML-Lite, and other outstanding AutoML methods. Compared with the state-of-the-art AutoML methods, MA2ML achieves very competitive performance.

Ablation on off-policy learning and credit assignment. To show search efficiency improved by off-policy learning, we implement MA2ML on-policy version as an ablation on CIFAR-10. Figure 5 illustrates the average accuracy curves of top-30 pipelines and the scatter plot for average accuracy in each batch of MA2ML, MA2ML-Lite, and MA2ML on-policy during learning on CIFAR-10. The performance of MA2ML surpasses MA2ML-Lite at early stages and keeps the lead all the way, which indicates the effectiveness and high sample efficiency of MA2ML on CIFAR-10. From the scatter diagram, one can see that MA2ML outperforms MA2ML-Lite by a large margin from the start. Compared with MA2ML on-policy, MA2ML surpasses it a lot at the beginning of the search and leads all the way, which validates the search efficiency gains caused by off-policy learning. Besides, the performance gap between the MA2ML on-policy and MA2ML-Lite in the late search stages indicates the superiority of credit assignment[3].
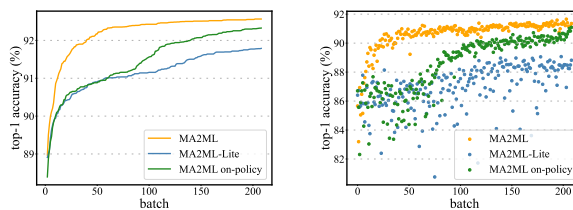

Figure 5. Learning patterns of MA2ML, MA2ML on-policy, and MA2ML-Lite on CIFAR-10. *Left*: average accuracy curve of top 30 pipelines in terms of the number of batches. *Right*: the scatter plot for the average accuracy of different pipelines in each batch. MA2ML outperforms MA2ML on-policy and MA2ML-Lite consistently in terms of accuracy and sample efficiency.

---

[3]The learning pattern of MA2ML and MA2ML-Lite on CIFAR-100 is available in the supplementary material.

Table 3. Top-1 accuracy (%) and parameter size (MB) of compared AutoML methods on CIFAR-10 and CIFAR-100. The results of MA2ML and MA2ML-Lite are mean and standard deviation of 5 trials of the best ML pipeline.

Model	CIFAR-10		CIFAR-100		Method
	Acc (%)	Param (M)	Acc (%)	Param (M)	
NASNet-A [54]	97.60	27.6	-	-	RL
ENAS [27]	97.11	4.6	80.57	4.6	RL
L^{2}NAS [25]	$97.51_{\pm0.12}$	3.8	$82.24_{\pm0.19}$	3.5	RL
AmoebaNet [29]	$97.45_{\pm0.05}$	2.8	81.07	3.1	evolution
DARTS [22]	$97.24_{\pm0.09}$	3.3	$82.64_{\pm0.44}$	3.3	gradient
DARTS- [3]	$97.41_{\pm0.08}$	3.5	$82.49_{\pm0.25}$	3.3	gradient
MiLeNAS [11]	$97.49_{\pm0.11}$	3.9	-	-	gradient
ISTA-NAS [44]	$97.64_{\pm0.06}$	3.4	$83.10_{\pm0.11}$	-	gradient
AutoHAS [6]	95.0	-	78.4	-	RL
Joint Search [17]	$97.46_{\pm0.09}$	-	$83.81_{\pm0.49}$	-	gradient
DAAS [40]	$97.76_{\pm0.10}$	4.0	$84.63_{\pm0.31}$	3.8	gradient
DHA [52]	$98.11_{\pm0.26}$	-	$83.93_{\pm0.23}$	-	gradient
MA2ML-Lite	$97.70_{\pm0.10}$	7.8	$84.80_{\pm0.12}$	9.0	MARL
MA2ML	$97.77_{\pm0.07}$	9.0	$85.08_{\pm0.14}$	7.7	MARL

Table 4. Comparison of NAS+AUG+HPO, AUG+HPO, and HPO on CIFAR-100.

Module	Baseline	HPO	AUG+HPO	NAS+AUG+HPO
Acc(%)	73.50	74.39	77.37	85.08

Ablation on ML Modules. On CIFAR-100, the ablation studies for ML modules are performed. We fix the network architecture as ResNet-56 and run the optimization of HPO and AUG+HPO. In Table 4, one can see that HPO improves 0.89% accuracy (74.39% vs. 73.50%) over the baseline (ResNet-56 using default hyper-parameters) while AUG and NAS respectively improve 2.98% and 7.71%, which are all significant margins.

5. Conclusion

We proposed MA2ML, a general framework for the joint optimization of ML pipelines. MA2ML transforms the joint optimization of modules as an MARL problem, and provides a method with theoretical guarantee. Empirically, we investigated the joint optimization of AUG, NAS, and HPO for image classification tasks on ImageNet and CIFAR-10/100. MA2ML yields state-of-the-art top-1 accuracy on ImageNet under constraints of computational cost with similar settings, *e.g.*, 79.7%/80.5% with FLOPs fewer than 600M/800M. MA2ML substantially outperforms MA2ML-Lite and MA2ML on-policy, which empirically verified the benefit of credit assignment and off-policy learning of MA2ML.

MA2ML has several advantages over other AutoML methods. First, MA2ML can be applied to the joint optimization of any arbitrary combination of different ML modules. Second, MA2ML is agnostic to search spaces, so we can choose suitable search spaces for different datasets. Third, MA2ML can also be easily generalized to different tasks by taking the evaluation (e.g., mIOU and pixel accuracy for semantic segmentation, and mAP for object detection) as the reward. Fourth, MA2ML is an RL-based method, which optimizes reward instead of training loss. Consequently, the incompatibility of different ML modules that may exist in the methods based on training loss, *e.g.*, gradient-based methods, is naturally avoided. Last but not the least, the search of MA2ML can be easily controlled without affecting the optimality of the performance.

MA2ML may have some limitations. From the experiments, we can find that there exists a gap between the rank of short-term and final long-term training accuracy, which causes the superiority of MA2ML's final performance over MA2ML-Lite not as large as in the search. Moreover, although MA2ML provides theoretical guarantee of the performance, as an MARL-based method, the search cost is large as other RL-based AutoML methods, since receiving the final reward costs much. How to address these two limitations is left as future work.

Acknowledgement

References

[1] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(2), 2012. 2

[2] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations (ICLR)*, 2020. 6

[3] Xiangxiang Chu, Xiaoxing Wang, Bo Zhang, Shun Lu, Xiaolin Wei, and Junchi Yan. Darts-: Robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations (ICLR)*, 2020. 2, 8

[4] Ekin D Cubuk, Barret Zoph, Dandelion Mané, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 1, 2

[5] 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 predictor pretraining. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021. 2, 5, 6, 11

[6] Xuanyi Dong, Mingxing Tan, Adams Wei Yu, Daiyi Peng, Bogdan Gabrys, and Quoc V Le. Autohas: Efficient hyperparameter and architecture search. *arXiv preprint arXiv:2006.03656*, 2020. 2, 8

[7] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2018. 3

[8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2014. 2

[9] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning (ICML)*, 2018. 4

[10] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster autoaugment: Learning augmentation strategies using backpropagation. In *European Conference on Computer Vision (ECCV)*, 2020. 2

[11] Chaoyang He, Haishan Ye, Li Shen, and Tong Zhang. Milenas: Efficient neural architecture search via mixed-level reformulation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2, 6, 8

[12] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In *International Conference on Machine Learning (ICML)*, 2019. 2

[13] Andrew Howard, Andrey Zhmoginov, Liang-Chieh Chen, Mark Sandler, and Menglong Zhu. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. In *IEEE/CVF International Conference on Computer Vision (CVPR)*, 2018. 5

[14] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision (ECCV)*, 2016. 5

[15] 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 (LION)*, 2011. 2

[16] Kirthevasan Kandasamy, Gautam Dasarathy, Jeff Schneider, and Barnabás Póczos. Multi-fidelity bayesian optimisation with continuous approximations. In *International Conference on Machine Learning (ICML)*, 2017. 2

[17] Taiga Kashima, Yoshihiro Yamada, and Shunta Saito. Joint search of data augmentation policies and network architectures, 2021. 2, 8

[18] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015. 5

[19] Aaron Klein and Frank Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*, 2019. 2

[20] Alex Krizhevsky. Learning multiple layers of features from tiny images. 2009. 2

[21] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast autoaugment. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019. 2

[22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. In *International Conference on Learning Representations (ICLR)*, 2018. 2, 6, 8

[23] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020. 2

[24] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning (ICML)*, 2015. 2

[25] Keith G Mills, Fred X Han, Mohammad Salameh, Seyed Saeed Changiz Rezaei, Linglong Kong, Wei Lu, Shuo Lian, Shangling Jui, and Di Niu. L2nas: Learning to optimize neural architectures via continuous-action reinforcement learning. In *ACM International Conference on Information & Knowledge Management (CIKM)*, 2021. 1, 2, 6, 8

[26] Fabian Pedregosa. Hyperparameter optimization with approximate gradient. In *International Conference on Machine Learning (ICML)*, 2016. 2

[27] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning (ICML)*, 2018. 2, 8

[28] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020. 6

[29] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2019. 2, 6, 8

[30] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. In *International Conference on Machine Learning (ICML)*, 2017. 2

[31] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 2

[32] 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)*, 2019. 2

[33] Kefan Su and Zongqing Lu. Divergence-regularized multi-agent actor-critic. In *International Conference on Machine Learning (ICML)*, 2022. 4

[34] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2000. 3

[35] 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 *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 2

[36] 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 *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019. 5

[37] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*, 2019. 6

[38] H. Tobias, NG Nicolás, M. Sven, and W. Stefan. Speeding up the hyperparameter optimization of deep convolutional neural networks. *International Journal of Computational Intelligence and Applications*, 17(02):1850008–, 2018. 1

[39] 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. *arXiv preprint arXiv:2011.11233*, 2020. 6

[40] Xiaoxing Wang, Xiangxiang Chu, Junchi Yan, and Xiaokang Yang. Daas: Differentiable architecture and augmentation policy search, 2021. 2, 6, 8

[41] Jian Wu, Saul Toscano-Palmerin, Peter I Frazier, and Andrew Gordon Wilson. Practical multi-fidelity bayesian optimization for hyperparameter tuning. In *Uncertainty in Artificial Intelligence (UAI)*, 2020. 2

[42] Han Xiao, Ziwei Wang, Zheng Zhu, Jie Zhou, and Jiwen Lu. Shapley-nas: Discovering operation contribution for neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11892–11901, June 2022. 6

[43] 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 *International Conference on Learning Representations (ICLR)*, 2020. 6

[44] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020. 2, 6, 8

[45] Zhaohui Yang, Yunhe Wang, Xinghao Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. Cars: Continuous evolution for efficient neural architecture search. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 2

[46] Arber Zela, Aaron Klein, Stefan Falkner, and Frank Hutter. Towards automated deep learning: Efficient joint neural architecture and hyperparameter search. *arXiv preprint arXiv:1807.06906*, 2018. 2

[47] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018. 5

[48] Miao Zhang, Shirui Pan, Xiaojun Chang, Steven Su, Jilin Hu, Gholamreza (Reza) Haffari, and Bin Yang. Balenas: Differentiable architecture search via the bayesian learning rule. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11871–11880, June 2022. 6

[49] Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial autoaugment. In *International Conference on Learning Representations (ICLR)*, 2019. 2

[50] Xiawu Zheng, Xiang Fei, Lei Zhang, Chenglin Wu, Fei Chao, Jianzhuang Liu, Wei Zeng, Yonghong Tian, and Rongrong Ji. Neural architecture search with representation mutual information. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11912–11921, June 2022. 6

[51] Xiawu Zheng, Rongrong Ji, Yuhang Chen, Qiang Wang, Baochang Zhang, Jie Chen, Qixiang Ye, Feiyue Huang, and Yonghong Tian. Migo-nas: Towards fast and generalizable neural architecture search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):2936–2952, 2021. 6

[52] 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. *arXiv preprint arXiv:2109.05765*, 2021. 2, 6, 8

[53] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2017. 1, 2

[54] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 1, 2, 6, 8, 11