10-2021

Direct differentiable augmentation search

Aoming LIU

Zehao HUANG

Zhiwu HUANG
Singapore Management University, zwhuang@smu.edu.sg

Huang

Naiyan WANG

Direct Differentiable Augmentation Search

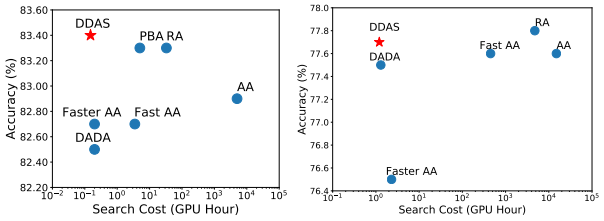
Aoming Liu[1], **Zehao Huang**[2], **Zhiwu Huang**[1], **Naiyan Wang**[2]
[1]ETH Zürich, Switzerland, [2]TuSimple, Beijing
{aoliu@student, zhiwu.huang@vision.ee}.ethz.ch, {zehaohuang18,winsty}@gmail.com

Abstract

Data augmentation has been an indispensable tool to improve the performance of deep neural networks, however the augmentation can hardly transfer among different tasks and datasets. Consequently, a recent trend is to adopt AutoML technique to learn proper augmentation policy without extensive hand-crafted tuning. In this paper, we propose an efficient differentiable search algorithm called Direct Differentiable Augmentation Search (DDAS). It exploits meta-learning with one-step gradient update and continuous relaxation to the expected training loss for efficient search. Our DDAS can achieve efficient augmentation search without relying on approximations such as Gumbel-Softmax or second order gradient approximation. To further reduce the adverse effect of improper augmentations, we organize the search space into a two level hierarchy, in which we first decide whether to apply augmentation, and then determine the specific augmentation policy. On standard image classification benchmarks, our DDAS achieves state-of-the-art performance and efficiency tradeoff while reducing the search cost dramatically, e.g. 0.15 GPU hours for CIFAR-10. In addition, we also use DDAS to search augmentation for object detection task and achieve comparable performance with AutoAugment [8], while being $1000\times$ *faster. Code will be released in* `https://github.com/zxcvfd13502/DDAS_code`

1. Introduction

Due to the "data hungry" nature of deep neural networks (DNN), data augmentation techniques, such as flipping, rotation, cropping, and color jittering, are essential tools to improve the performance. Data augmentation creates rich variation of data samples to reduce over-fitting issues caused by the high complexity of DNN. Although various hand-crafted data augmentation techniques [11, 19, 36, 39, 41, 45] are proposed recently, it's non-trivial to combine and adapt them when confronting a new task or dataset. This procedure usually requires expertise and extensive experiments to determine the optimal configuration. For example, an improper augmentation may not only be useless, but also may introduce harmful outliers in training.



(a) CIFAR-100 (b) ImageNet

Figure 1. Comparison between DDAS and other state-of-the-art Automatic Augmentation works, including AA [8], RA [9], PBA [18], Fast AA [25], Faster AA [14] and DADA [24]. Higher accuracy and lower search cost (upper left) are preferred. Our DDAS is significantly more efficient while achieving even better performance.

AutoAugment (AA) [8] is the pioneering work for automatic augmentation policy search. It utilizes an RL-based algorithm to search for optimal policy within a search space of 16 augmentation operations. During search, AA maximizes the accuracy on the validation set by optimizing 3 parameters: augmentation operation, its probability and magnitude, $(\texttt{op}, \texttt{prob}, \texttt{mag})$. Despite its impressive performance on image classification and detection tasks, the search cost of AA is still prohibitive for democratizing the technique, e.g. it requires thousands of GPU hours just for searching on a small dataset like CIFAR-10. A popular approach is to make the optimization differentiable, which makes the gradient estimation more efficient. Following works, such as [4, 14, 18, 24, 25], manage to decrease the search cost to $0.1 - 0.2$ hours, but with visible performance degradation.

On the other hand, AA only searches for a fixed augmentation policy while several works [17, 34] point out that dynamic augmentation policy may result in better performance. Following works such as OHL-Auto-Aug (OHLAA) [26] or Adverisial Augment (AdvAA) [42] propose online augmentation search manners that augmentation search is conducted together with training and different augmentation policies will be used for different training

epochs. These online approaches achieve significant improvements over the offline counterpart, however their costs remain high, and the policies can hardly transfer to different training tasks.

To achieve faster search with better performance, we introduce Direct Differentiable Augmentation Search (DDAS). Inspired by recent work on meta-learning [29,32], our goal is to find an augmentation policy which maximizes the network performance after one step gradient update. Inspired by OHLAA, we organize the augmentation policy into a two level hierarchy: we firstly determine the probability of augmenting the data, then we decide the probability of each augmentation operation. In this way, the policy has a chance to discard all augmentations. Then the differentiable search can be derived without tricks such as Gumbel-Softmax [20] or second order gradient approximation [29], as the probabilities naturally become weights of samples when considering the expectation of loss.

We verify the effectiveness of our DDAS on various models and datasets, including CIFAR-10/100 [22] and ImageNet [10]. All results show that our DDAS can achieve competitive or better performance while dramatically reducing the search cost. In Fig.1, we visualize the comparison between DDAS and other methods on CIFAR-100 and ImageNet. We can see that our proposed DDAS is on the Pareto optimal curve of the search cost v.s. testing accuracy. In addition, we also try to search for augmentation policy for object detection task. As far as we know, this task is rarely tackled due to its prohibitive cost. Thanks to our highly efficient search method, we could achieve results comparable with previous work [46] that adopted AutoAugment for object detection, while costing $1000\times$ less GPU hours. Our contributions can be summarized as follows:

1. We propose Direct Differentiable Augmentation Search (DDAS), an efficient differentiable augmentation policy search algorithm. Through meta-learning with one-step gradient update, we can achieve efficient and effective augmentation search.

2. We propose a compact yet flexible search space by explicitly modeling the probability of adopting augmentation. This design along with the epoch-wise policy reduces the adverse risk of aggressive augmentation.

3. Besides the thorough evaluation experiments for image classification, we are the first work to demonstrate efficient augmentation search for object detection (20 GPU hours) is feasible.

2. Related Works

2.1. Data Augmentation

Data Augmentation has been a standard technique for deep neural network training. Common data augmentations include rotation, translation, cropping and resizing. In recent years, several novel augmentation operations are designed manually according to domain knowledge or intuitions, such as Cutout [11], MixUp [41] and CutMix [39]. In addition to supervised learning, Data Augmentation is also important for semi-supervised learning [1, 2, 37], self-supervised learning [6], unsupervised learning [43] and reinforcement learning [21]. Although these augmentations are effective on a specific task, transferring them into other tasks or datasets still requires extensive workload. Furthermore, improper augmentation may hurt the model performance. For example, [8] reports that Cutout significantly hurts the performance on reduced SVHN. Thus it is valuable to automatically search for suitable augmentation policies for different tasks.

2.2. Automatic Data Augmentation

Inspired by Neural Architecture Search (NAS) [29, 38, 47], recent works attempt to search for data augmentation policy automatically for different tasks, such as image classification [8, 15, 18, 25, 30, 33, 34, 35], 2D object detection [9, 46] and 3D point clouds related tasks [7, 23]. AutoAugment (AA) [8], the pioneering work for automatic augmentation, proposes a novel search space consisting of 16 augmentation operations and adopts a reinforcement learning based algorithm for search. AA achieves promising performance improvement on classification task but the search cost is prohibitive (5000 GPU hours on CIFAR-10). Following AA, Population Based Augmentation (PBA) [18] applies Population Based Training (PBT), an evolution based hyper-parameter optimization algorithm for augmentation policy search. Fast AutoAugment (Fast AA) [25] treats augmentation search as a density matching problem and uses Bayesian Optimization to solve it. Both of these two methods reduce the search cost from thousands of GPU hours to several hours, e.g. Fast AA only costs 3.5 GPU hours on CIFAR-10.

More recently, Faster AutoAugment (Faster AA) [14] and Differentiable Automatic Data Augmentation (DADA) [24] propose differentiable relaxations to this challenging problem. DADA relaxes the original optimization with RELAX gradient estimator. Faster AA attempts to address this problem from another perspective: it replaced the original augmentation search task with a distribution matching task between the augmented data and original data. They both significantly improve the efficiency (0.1 - 0.2 GPU hour on CIFAR-10), but suffer from performance degradation compared with AA.

All the above methods need to firstly search for augmentation policy on an offline proxy dataset, and then transfer it to the final training. On the other hand, RandAugment (RA) [9], OHL-Auto-Aug (OHLAA) [26] and Adversarial AutoAugment (AdvAA) [42] propose a different

search paradigm. They argue that the augmentation policy searched on proxy task is sub-optimal for target task, and a fixed augmentation policy is suboptimal for different training stage. Specifically, RA directly uses naive grid search to find the best policy, while OHLAA and AdvAA propose online search manners by jointly optimizing augmentation policies and training the target networks. Recently, MetaAugment [44] further proposes sample-aware online data augmentation search. Despite their superior results, their online search costs are quite large and not affordable on large dataset. Our proposed DDAS is an offline method, however with the help of dynamic interpolation method, our method can apply different augmentation policies during different training stages. As a result, our method can get the best of two worlds.

2.3. Comparison between DADA and DDAS

As mentioned above, DADA is close to our DDAS. Thus we highlight some key differences here. Both DADA and DDAS use meta-learning with one-step gradient update and differentiable optimization. However, as DADA follows the parameterization of AA, it has to use Gumbel-Softmax parameterization and RELAX gradient estimator [13] to make the optimization differentiable. In addition, second order gradient approximation [29] is necessary for DADA to achieve efficient search, which may result in inaccurate gradient approximation according to [3]. In contrast, our DDAS directly uses the expectation of training loss to derive the differentiable search formula without Gumbel-Softmax, gradient estimator and second order gradient approximation.

A primary advantage of DDAS over DADA is the generalisability to complex tasks such as object detection. When applying DADA to object detection, the search cannot finally converge. We attribute this convergence problem to gradient noise and variance, which may be caused by inaccurate gradient approximation. In contrast, DDAS can achieve efficient augmentation search for object detection.

3. Method

3.1. Search Space Reformulation

As aforementioned, one notable difference between our method and previous works is that we explicitly model the probability of applying augmentation in our search space. The search space of previous works indeed contains the option of no augmentation, however the augmentation is conducted in a multi-step manner. The probability of sampling each augmentation is independent at each step, consequently it may rarely sample a policy that applying no augmentation at all, and result in "over-strong" augmentation that deteriorates accuracy.

Formally, suppose we sample N_o operations from K candidate augmentation operations $\{o_1, ..., o_K\}$ sequentially to form an augmentation policy φ_l. Thus the number of all possible augmentation policies is $L = K^{N_o}$. Note that we discretize the continuous magnitude of augmentation and consider the same augmentation with different magnitudes as multiple operations. For example, $Rotation\ 15^\circ$ and $Rotation\ 20^\circ$ are regarded as two different candidate operations in our search space. Specifically, we model the augmentation policy with two cascaded probabilities: **total probability** p_{tp} and **operation probability** $\mathbf{p}_o = \{p_o^{o_1}, p_o^{o_2}, \ldots, p_o^{o_K}\}$. p_{tp} decides whether to apply augmentation on the original data and then $\mathbf{p}_o$ decides the probability for a certain augmentation. Fig.2 shows the chain of constructing an augmentation policy.

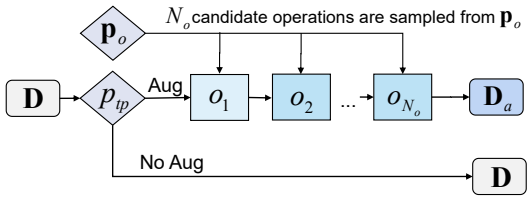


Figure 2. Augmentation pipeline. First step is to decide whether to augment by sampling from p_{tp}. Second step is to decide `op` and `mag` by sampling from $\mathbf{p}_o$. In the second step, we sample N_o operations from K candidate operations and apply them sequentially for the input data.

3.2. Formulation for Augmentation Search

Augmentation search can be formulated as a bi-level optimization problem:

$$\begin{aligned} &\min_{\mathbf{p}} \mathcal{J}(\mathbf{p}) = \mathcal{L}_{val}(\mathbf{V}, \boldsymbol{\theta}^*_{\mathbf{p}}) \\ &\text{s.t.} \quad \boldsymbol{\theta}^*_{\mathbf{p}} = \arg\min_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{p}} \left[\mathcal{L}_{train}(\mathbf{D}, \boldsymbol{\theta})\right], \end{aligned} \tag{1}$$

where $\mathcal{L}_{train}$ and $\mathcal{L}_{val}$ indicate the total training and validation loss; $\boldsymbol{\theta}$ represents the weights of DNN, and $\mathbf{D}$ and $\mathbf{V}$ denote the training samples and validation samples, respectively. Note that we assume that $\mathbf{D}$ in training is further augmented based on original data $\mathbf{D}_0$ and augmentation probability $\mathbf{p} = \{\mathbf{p}_o, p_{tp}\}$, while $\mathbf{V}$ for validation is kept as original as common practice.

It's non-trivial to solve this optimization problem since the inner optimization needs a full training of a given DNN, which is very time-consuming. Inspired by DARTS [29], we adopt the idea of meta-learning to relax Eqn.1 as a differentiable optimization problem. The inner optimization is approximated with one-step gradient update, instead of training until convergence. Thus we could get the approximated gradient of validation loss wrt. augmentation parameters:

$$\begin{aligned} &\nabla_{\mathbf{p}} \mathcal{L}_{val}(\mathbf{V}, \boldsymbol{\theta}^*_{\mathbf{p}}) \\ \approx &\nabla_{\mathbf{p}} \mathcal{L}_{val}(\mathbf{V}, \boldsymbol{\theta} - \eta \cdot \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{p}} \left[\mathcal{L}_{train}(\mathbf{D}, \boldsymbol{\theta})\right]), \end{aligned} \tag{2}$$

where η is the learning rate.

Given an input mini-batch data $\mathbf{D}_0^t$ at each step t, we could generate a set of different augmented mini-batches $\{\mathbf{D}_1^t, ..., \mathbf{D}_L^t\}$ by applying the L augmentation policies, $\varphi_1, ..., \varphi_L$, to it. We further define an augmentation policy $\varphi_l = \{o_l^1, \ldots, o_l^{N_o}\}$ with $o_l^i \in \{o_1, \ldots, o_K\}$ denoting the augmentation index chosen in the i-th step. Then we can define the probability to get an augmented mini-batch $\mathbf{D}_l^t$ with augmentation policy $\varphi_l = \{o_l^1, \ldots, o_l^{N_o}\}$ as

$$P(\mathbf{D}_l^t) = p_{tp} \cdot P(\varphi_l) = p_{tp} \cdot \prod_{i=1}^{N_o} p_o^{o_l^i}, \tag{3}$$

Obviously, the probability of sampling a mini-batch without any augmentation can be represented as $P(\mathbf{D}_0^t) = 1 - \sum_{l=1}^{L} P(\mathbf{D}_l^t) = 1 - p_{tp}$. Given the probability of each augmented mini-batch, we can expand the expected training loss as:

$$\begin{aligned}&\mathbb{E}_{\mathbf{p}}\left[\mathcal{L}_{train}(\mathbf{D}_0^t, \boldsymbol{\theta}_t)\right] \\ =&\sum_{l=1}^{L} P(\mathbf{D}_l^t)\mathcal{L}_{train}(\mathbf{D}_l^t, \boldsymbol{\theta}^t) + P(\mathbf{D}_0^t)\mathcal{L}_{train}(\mathbf{D}_0^t, \boldsymbol{\theta}^t).\end{aligned} \tag{4}$$

Then the gradient wrt. $\boldsymbol{\theta}^t$ can be easily derived by

$$\begin{aligned}\mathbf{g}^t &= \sum_{l=1}^{L} P(\mathbf{D}_l^t)\frac{\partial \mathcal{L}_{train}(\mathbf{D}_l^t, \boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}^t} + P(\mathbf{D}_0^t)\frac{\partial \mathcal{L}_{train}(\mathbf{D}_0^t, \boldsymbol{\theta}^t)}{\partial \boldsymbol{\theta}^t} \\ &= \sum_{l=1}^{L} P(\mathbf{D}_l^t)\mathbf{g}_l^t + (1 - \sum_{l=1}^{L} P(\mathbf{D}_l^t))\mathbf{g}_0^t.\end{aligned} \tag{5}$$

3.3. Meta-Learning with One-Step Gradient Update

Following the approximation in Eqn.2, network parameters are updated for one step with learning rate η:

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^t - \eta \cdot \mathbf{g}^t. \tag{6}$$

Then assume we sample a validation mini-batch $\mathbf{V}^t$ from validation dataset, the validation loss becomes $\mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})$, and we can get the gradient of validation loss wrt. $\mathbf{p}_o$ and p_{tp} as :

$$\frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial p_o^{o_k}} = \sum_{l=1}^{L} \frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial P(\mathbf{D}_l^t)} \cdot \frac{\partial P(\mathbf{D}_l^t)}{\partial p_o^{o_k}}. \tag{7}$$

$$\frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial p_{tp}} = \sum_{l=1}^{L} \frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial P(\mathbf{D}_l^t)} \cdot \frac{\partial P(\mathbf{D}_l^t)}{\partial p_{tp}}. \tag{8}$$

The partial derivative of $\mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})$ wrt. $P(\mathbf{D}_l^t)$ can be further derived by:

$$\begin{aligned}\frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial P(\mathbf{D}_l^t)} &= \frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial \boldsymbol{\theta}^{t+1}}^{\top} \cdot \frac{\partial \boldsymbol{\theta}^{t+1}}{\partial P(\mathbf{D}_l^t)} \\ &= \eta \cdot {\mathbf{g}_{val}^t}^{\top} \cdot (\mathbf{g}_0^t - \mathbf{g}_l^t),\end{aligned} \tag{9}$$

where $\mathbf{g}_{val}^t = \frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial \boldsymbol{\theta}^{t+1}}$. Then Eqn.7 and Eqn.8 can be reformulated as:

$$\frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial p_o^{o_k}} = \sum_{l=1}^{L} \eta \cdot {\mathbf{g}_{val}^t}^{\top} \cdot (\mathbf{g}_0^t - \mathbf{g}_l^t) \cdot p_{tp} \cdot \frac{\partial P(\varphi_l)}{\partial p_o^{o_k}}, \tag{10}$$

$$\frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial p_{tp}} = \sum_{l=1}^{L} \eta \cdot {\mathbf{g}_{val}^t}^{\top} \cdot (\mathbf{g}_0^t - \mathbf{g}_l^t) \cdot P(\varphi_l). \tag{11}$$

We denote $\frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial p_o^{o_k}}$ as $\mathbf{g}_{p_o^{o_k}}$ and $\frac{\partial \mathcal{L}_{val}(\mathbf{V}^t, \boldsymbol{\theta}^{t+1})}{\partial p_{tp}}$ as $\mathbf{g}_{p_{tp}}$, then $p_o^{o_k}$ and p_{tp} can be updated by gradient descent with $\mathbf{g}_{p_o^{o_k}}$ and $\mathbf{g}_{p_{tp}}$, respectively.

3.4. Stochastic Policy Sampling

In practice, it will be intractable if we directly calculate the gradient of $\mathbf{p}$ following Eqn.10 and Eqn.11, as we need to sample all the L possible augmentation polices where L can be very large, e.g. $L = 4624$ for CIFAR experiments. In order to search efficiently, we only randomly sample $\hat{L}$ policies $\{\varphi_1, ..., \varphi_{\hat{L}}\}$ to calculate the expected training loss as described in Eqn 4:

$$\begin{aligned}&\mathbb{E}_{\mathbf{p}}\left[\mathcal{L}_{train}(\mathbf{D}_0^t, \boldsymbol{\theta}^t)\right] \\ \approx&\sum_{l=1}^{\hat{L}} \frac{P(\mathbf{D}_l^t)}{Z}\mathcal{L}_{train}(\mathbf{D}_l^t, \boldsymbol{\theta}^t) + (1 - \sum_{l=1}^{\hat{L}} \frac{P(\mathbf{D}_l^t)}{Z})\mathcal{L}_{train}(\mathbf{D}_0^t, \boldsymbol{\theta}^t),\end{aligned} \tag{12}$$

where $Z = \sum_{l=1}^{\hat{L}}(P(\varphi_l))$ is a normalization factor to make sure that the probability of no augmentation equals to $1 - p_{tp}$. Based on Eqn.12, we have:

$$\mathbf{g}_{p_o^{o_k}} = \frac{1}{Z} \cdot \sum_{l=1}^{\hat{L}} \eta \cdot {\hat{\mathbf{g}}_{val}^t}^{\top} \cdot (\hat{\mathbf{g}}_0^t - \hat{\mathbf{g}}_l^t) \cdot p_{tp} \cdot \frac{\partial P(\varphi_l)}{\partial p_o^{o_k}}, \tag{13}$$

$$\mathbf{g}_{p_{tp}} = \frac{1}{Z} \cdot \sum_{\hat{l}=\hat{1}}^{\hat{L}} \eta \cdot {\hat{\mathbf{g}}_{val}^t}^{\top} \cdot (\hat{\mathbf{g}}_0^t - \hat{\mathbf{g}}_l^t) \cdot P(\varphi_l), \tag{14}$$

where $\hat{\mathbf{g}}_{val}^t$ is the validation gradient obtained after using approximated loss expectation in Eqn.12 to do one-step gradient update, and $\hat{\mathbf{g}}_l^t$ is the gradient of training loss corresponding to $\mathbf{D}_l^t$.

In the actual search process, we find that there is obvious scale difference on gradients caused by different data at each step, as each $\mathbf{D}_0^t$ could contribute differently to the validation loss. In order to achieve stable search, we normalize the gradients for $\mathbf{D}_0^t$ and $\{\mathbf{D}_1^t \ldots \mathbf{D}_{\hat{L}}^t\}$ with a scale factor Z_g calculated by different data at each step:

$$Z_g = \sum_{l=1}^{\hat{L}} \left| {\hat{\mathbf{g}}_{val}^t}^{\top} \cdot (\hat{\mathbf{g}}_0^t - \hat{\mathbf{g}}_l^t) \right|. \tag{15}$$

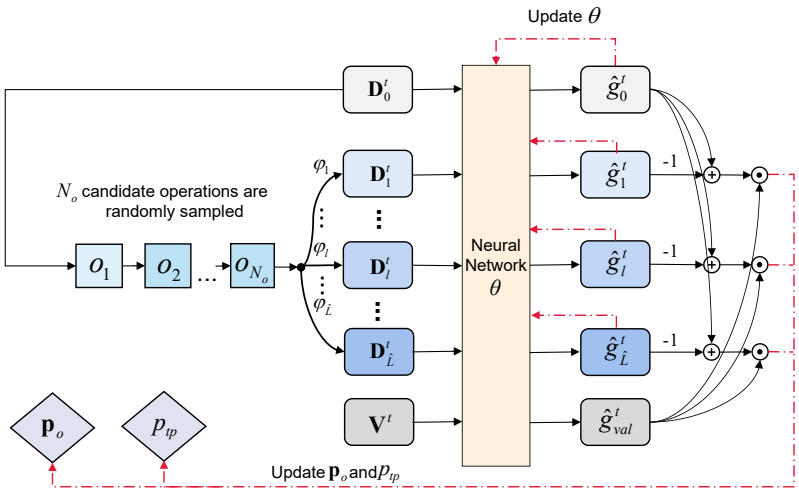


Figure 3. Overview of our proposed Directly Differentiable Automatic Augmentation pipeline. $\hat{L}$ augmented mini-batches $\mathbf{D}_1^t, \ldots, \mathbf{D}_{\hat{L}}^t$ are generated by applying augmentation policies $\varphi_{\hat{1}}, ..., \varphi_{\hat{L}}$ sampled with uniform probability distribution to original mini-batch $\mathbf{D}_0^t$. Augmentation parameters, p_{tp} and $\mathbf{p}_o$, are updated with approximated gradients according to Eqn.16 and Eqn.17.

With this approximation, $\mathbf{g}_{p_o^{o_k}}$ and $\mathbf{g}_{p_{tp}}$ finally become:

$$\mathbf{g}_{p_o^{o_k}} = \frac{1}{Z \cdot Z_g} \cdot \sum_{l=1}^{\hat{L}} \eta \cdot \hat{\mathbf{g}}_{val}^t{}^\top \cdot (\hat{\mathbf{g}}_0^t - \hat{\mathbf{g}}_l^t) \cdot p_{tp} \cdot \frac{\partial P(\varphi_l)}{\partial p_o^{o_k}}, \quad (16)$$

$$\mathbf{g}_{p_{tp}} = \frac{1}{Z \cdot Z_g} \cdot \sum_{\hat{l}=\hat{1}}^{\hat{L}} \eta \cdot \hat{\mathbf{g}}_{val}^t{}^\top \cdot (\hat{\mathbf{g}}_0^t - \hat{\mathbf{g}}_l^t) \cdot P(\varphi_l). \quad (17)$$

We find that we can achieve efficient optimization with the derived $\mathbf{g}_{p_o^{o_k}}$ and $\mathbf{g}_{p_{tp}}$, and there is no need to apply second order gradient approximation [29] to further improve the search efficiency.

3.5. Implementation Details and Algorithms

In this section, we will introduce implementation details of our proposed algorithm.

3.5.1 Search Details

We search for augmentation policies on proxy tasks by splitting the original dataset into smaller train and validation dataset, S_{train} and S_{val}. And the proxy search runs for T_{max} epochs, which is smaller than that in common training. Note that we sample augmentations from a uniform distribution during search to avoid Matthew Effect: operations with large probabilities to be continuously selected and promoted.

3.5.2 Probability Implementation

To make p_{tp} fall within valid range, we implement it using sigmoid function with its input noted as α_{tp}: $p_{tp} = \frac{1}{1+e^{-\alpha_{tp}}}$. To make elements of $\mathbf{p}_o$ sum to 1, we implement $\mathbf{p}_o$ using softmax function with its input noted as $\boldsymbol{\alpha_o}$: $p_o^{o_k} = \frac{e^{\alpha_o^k}}{\sum_{i=1}^{K} e^{\alpha_o^i}}$, where α_o^k is the k-th item of $\boldsymbol{\alpha_o}$.

3.5.3 Augmentation Operations and Magnitudes

Similar as [26], we combine `op` and `mag` by discrete sampling of magnitude for each operation. Each operation requiring magnitude is combined with the selected magnitude values to form independent candidate operations: $o_i = (\texttt{op}_i,\ \texttt{mag}_i)$. Other operations that do not need magnitude are regarded as single candidate operations $o_i = (\texttt{op}_i, \texttt{None})$. We follow the setting of `mag` in RandAugment [8], which is elaborated in the appendix detailedly.

3.5.4 Dynamic Offline Policy

There are evidences suggesting that different training stage may prefer different augmentation policy [17, 34]. Consequently, we design a dynamic offline augmentation policy which saves the snapshot of augmentation parameters $\mathbf{p}$ for every epoch during search, and then replay them during final re-training. However, the re-training typically takes more epochs than search, we further propose to upsample the policy with nearest neighbor interpolation. When applied to training stage, we use 1-D mean filter with size F_s to smooth the p_{tp} as there may be fluctuations of p_{tp} which will deteriorate training accuracy.

3.5.5 DDAS Overview and Algorithm

We summarize the whole method in Algorithm 1, and a visual demonstration is shown in Fig. 3.

4. Experiments

We apply our DDAS to search for augmentation for two tasks: image classification and object detection. First we perform a sanity check on a toy experiment to show that our DDAS can learn reasonable augmentation. Then, for image

Table 1. Comparison of search time in GPU hours.

	AA	RA†	PBA	Fast AA	Faster AA	DADA	**DDAS**
GPU	P100	2080Ti	Titan XP	V100	V100	Titan XP	2080Ti
CIFAR-10/100	5000	33	5	3.5	0.2	0.1/0.2	0.15
ImageNet	15000	4750	-	450	2.3	1.3	1.2

† Estimated based on our experiment settings.

Algorithm 1 Search Algorithm of DDAS

INPUT: S_{train} and S_{val} from original dataset.
Initialize $\mathbf{p}_o$, p_{tp} and network parameter $\boldsymbol{\theta}$.
for $T = 1, 2, ..., T_{max}$ **do**
  for $t = 1, 2, ..., max\_iter$ **do**
    Sample $\mathbf{D}_0^t$ from S_{train} and compute $\hat{\mathbf{g}}_0^t$.
    Sample $\mathbf{V}^t$ from S_{val}.
    for l = 1,...,$\hat{L}$ **do**
      Randomly sample φ_l with equal probabilities.
      Apply φ_l to $\mathbf{D}_0^t$ to generate $\mathbf{D}_l^t$.
      Compute $\hat{\mathbf{g}}_l^t$ with the training loss of $\mathbf{D}_l^t$.
    end for
    Do one-step gradient update and compute $\hat{\mathbf{g}}_{val}^t$.
    Update $\mathbf{p}_o^*$ and p_{tp}^* according to Eqn.16 and Eqn.17.
  end for
  $\mathbf{p}_o(T) \leftarrow \mathbf{p}_o^*$. $p_{tp}(T) \leftarrow p_{tp}^*$.
end for
Smooth $p_{tp}(1), .., p_{tp}(T_{max})$ with mean filter.
return $p_{tp}(1), .., p_{tp}(T_{max})$ and $\mathbf{p}_o(1), ..., \mathbf{p}_o(T_{max})$.

classification, we carry out experiments on three datasets: CIFAR-10, CIFAR-100 [22] and ImageNet [10]. For object detection, we evaluate our method on the popular COCO dataset [28].

We mainly compare our performance and search cost with offline augmentation search, including AA [8], Fast AA [25], PBA [18], RA [9], Faster AA [14] and DADA [24]. Note that AdvAA [42], OHLAA [26] and MetaAugment [44] are not compared here as it is hard to compare with them in a fair setting. For example, online methods usually adopt large batch sizes, which has significant impact on the performance.

4.1. Sanity Check

Experiment Setting. To demonstrate the effectiveness and correctness of our DDAS, we firstly design a toy example on the CIFAR-10 dataset. In normal training, it's hard to verify the correctness of an augmentation policy, as multiple augmentations may work in synergy to improve the performance. Therefore, we design the following sanity check: in contrast to the normal setting, we rotate the images in the validation set for 90°. And we add rotation for 90° as a candidate augmentation, which is noted as `Rotate90`. Then obviously, a reasonable policy should assign higher probability to the `Rotate90` operation. For simplicity, we replace the original rotation operation with `Rotate90` and only sample one magnitude for each operation. Other details are listed in the appendix.

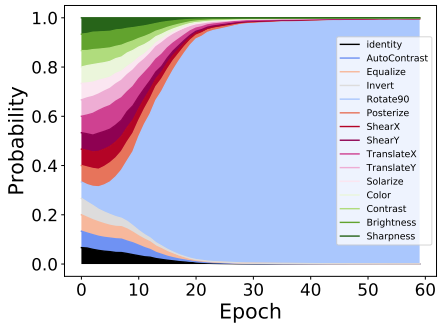


Figure 4. $\mathbf{p}_o$ searched in sanity check experiment. `Rotate90` is the dominant operation.

Result. We visualize the $\mathbf{p}_o$ searched in Fig 4. We can see that the `Rotate90` rapidly becomes the dominant operation, which proves the correctness of our DDAS.

4.2. CIFAR-10 and CIFAR-100

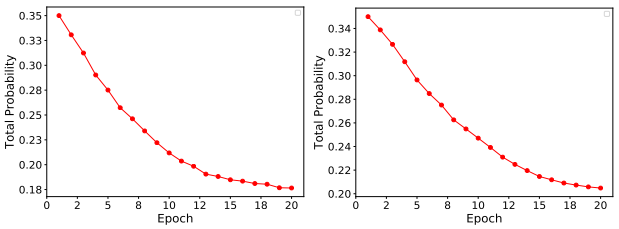
Search Setting. For CIFAR-10 and CIFAR-100, we conduct experiments with two different networks. We firstly search for augmentation policy on a proxy task with a small network, Wide-ResNet-40-2 [40], on part of the dataset for 20 epochs, with 2000 images for training and 2000 images for validation. We run the search for 20 epochs on both CIFAR-10 and CIFAR-100. The training mini-batch size is 32 while the validation mini-batch size is 256. At each step, we sample $\hat{L} = 3$ augmentation policies randomly according to a uniform distribution. There are $N_o = 2$ operations in an augmentation policy. The search is carried out on a RTX 2080Ti. We initialize $\mathbf{p}_o$ equally and p_{tp} as 0.35.

Training Setting. After search, we apply the searched augmentation to two prevalent networks, Wide-ResNet-28-10 [40] and Shake-Shake(26 2x96d) [12]. We train the networks on the full training set and report the performance evaluated on test set. Wide-ResNet-28-10 is trained for 200 epochs and Shake-Shake(26 2x96d) is trained for 1800 epochs. We run every experiment for three times and report the average test error rate. Other details are in the appendix.

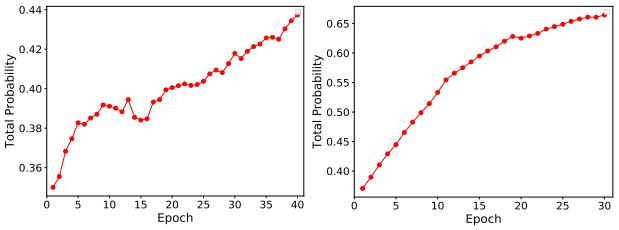
Result. The search costs of DDAS are listed in Table 1. And the test error rates are shown in Table 2. Results of other automatic augmentation search works are also listed for comparison. As shown in Table 1, our DDAS costs only

Table 2. CIFAR-10 and CIFAR-100 test error rates (%). WRN and SS are the shorthand of Wide-ResNet and Shake-Shake respectively.

	Baseline	Cutout	AA	PBA	Fast AA	RA	Faster AA	DADA	**DDAS**
CIFAR-10									
WRN-28-10	3.9	3.1	2.6	2.6	2.7	2.7	2.6	2.7	2.7 ± 0.1
SS (26 2x96d)	2.9	2.6	2.0	2.0	2.0	2.0	2.0	2.0	2.1 ± 0.1
CIFAR-100									
WRN-28-10	18.8	18.4	17.1	16.7	17.3	16.7	17.3	17.5	16.6 ± 0.2
SS (26 2x96d)	17.1	16.0	14.3	15.3	14.9	-	15.0	15.3	15.1 ± 0.2



(a) CIFAR-10 (b) CIFAR-100



(c) ImageNet (d) MS-COCO

Figure 5. Total Probability p_{tp} learned on different datasets. The trend of p_{tp} varies among different datasets, which validates the necessity of learning it automatically.

0.15 GPU hours to search on CIFAR-10 and CIFAR-100, which is significantly faster than AA, Fast AA and PBA. Comparing to those efficient search methods, such as Faster AA and DADA, our DDAS yields better performances as shown in Table 2. The advantage is even more significant in complicated datasets such as CIFAR-100. Jointly considering these two aspects, our method achieves the sweet spot for accuracy and efficiency tradeoff for automatic augmentation policy search.

Another desired property of our DDAS is that it enables the transfer of augmentation policy searched with small network to large networks. Our experiments further verify this claim: The policy obtained using Wide-ResNet-40-2 can transfer well to Wide-ResNet-28-10 and Shake-Shake(26 2x96d).

We further visualize the p_{tp} of the augmentations obtained on CIFAR-10 and CIFAR-100 in Fig.5 (a) & (b). We can see that the total probabilities keep decreasing, which means our DDAS prefers less augmentation as epoch number increases on CIFAR-10/100.

4.3. ImageNet

Due to the different data distributions, the augmentation policy obtained on CIFAR may not be useful when applied to large-scale datasets. For example, Cutout [11] shows less improvement on ImageNet than that on CIFAR. Thus we apply our DDAS to directly search on ImageNet.

Search Setting. As for ImageNet, we choose ResNet-18 [16] as our proxy model for search. The proxy dataset setting is the same as that in AA [8]. We randomly sample 120 classes from the 1000 classes of ImageNet. 6000 images are sampled (50 images for each class) as the training dataset S_{train} and 1200 images (20 images for each class) are sampled as the validation dataset S_{val}.

We run the search for 40 epochs. The training mini-batch size is 32 and validation mini-batch size is 64. At every step, we sample $\hat{L} = 2$ augmentation policies according to a uniform distribution. The operation number in a sampled augmentation policy is set to $N_o = 2$. The search is carried out on a single RTX 2080Ti GPU. Similar as the CIFAR experiments, we initialize $\mathbf{p}_o$ equally and p_{tp} with 0.35.

Training Setting. As for training, we train two networks: ResNet-50 and ResNet-200. Both of these two networks are trained for 270 epochs. Other training and search details are listed in the appendix.

Result. The search costs are shown in Table 1 and the test error rates on ImageNet are reported in Table 3. Our DDAS is on the Pareto optimal curve of the search cost v.s. testing error. The results show the consistent superiority across datasets of different scales.

We also visualize the p_{tp} obtained on ImageNet in Fig.5 (c). In contrast to CIFAR-10/100 experiment, we can see that p_{tp} obtained on ImageNet increases as training epoch increases. We hypothesize the reason may be the distribution variation between training and validation set of ImageNet data is larger than that in CIFAR, thus aggressive augmentations are needed to fill the distribution gap.

4.4. Object Detection on COCO

In addition to image classification, we further evaluate our DDAS on object detection task. Augmentation search for object detection is rarely tackled. As far as we know, only AA and RA are applied on object detection. [46] specialized AA to object detection by proposing new search

Table 3. ImageNet test error rates (%). Every model is trained for 270 epochs if not noted.

Model	Baseline	AA	Fast AA	Faster AA	DADA	RA†	**DDAS**
ResNet-50	23.7	22.4	22.4	23.5¶	22.5	22.2	22.3 ± 0.1
ResNet-200	21.5	20.0	19.4	-	-	-	19.7

† Reproduced RA with our settings for comparison. ¶ Trained for 200 epochs.

space. RA also tried augmentation search within the same search space. The central issue of both methods is they are very time-consuming. AA takes 19200 TPU hours to search for the augmentation policies. As far as we know, our DDAS is the first work to achieve efficient augmentation search for object detection. We implement our search algorithm based on MMDetection [5].

Search Setting. We use RetinaNet [27] with ResNet-50 as backbone to search for augmentation policies. We randomly sample 3000 images as the training dataset S_{train} and 1500 images as the validation dataset S_{val} from *train* split. We run the search for 30 epochs. Both the training mini-batch size and validation mini-batch size are 4 images per GPU. At every step, we will sample $\hat{L} = 3$ augmentation policies according to a uniform distribution. The operation number N_o in an sampled augmentation policy is set to 2. As for the augmentation operations, we follow the setting of [46]. The search is carried out on 4 RTX 2080Ti GPUs.

Training Setting. After obtaining the augmentation policies, we then train two networks following [46] from scratch: RetinaNet with ResNet-50 and ResNet-101 as backbones on the full COCO dataset. The two networks are trained for 150 epochs. For fair comparison, we reproduce AA based on our implementation with the searched policy described in [46]. As for RA [9], we simply copy the result of ResNet-101 from the paper, which is trained for 300 epochs, since we can not reproduce a decent performance.

Result. The search costs and mean Average Precision (mAP) on COCO *val* split are reported in Table 4. Our DDAS significantly reduces the search cost for object detection task on COCO, which is $1000\times$ faster than AA [46] while achieving slightly lower results. We also visualize the p_{tp} searched on COCO in Fig.5 (d), and find that p_{tp} increases as training epoch increases, which is similar as the search on ImageNet.

5. Ablation

In this section, we further analyze the results and do ablation experiments to show some interesting insights. The experiments in this section are all conducted with Wide-ResNet-28-10 on CIFAR-100 and ResNet-50 on ImageNet.

The effectiveness of p_{tp}. A notable difference between DDAS and previous works is that we model the probability of applying augmentation separately. To prove the superiority of learning an adaptive p_{tp}, we manipulate p_{tp} to a fixed value in both search and training to test the results.

Table 4. Mean Average Precision (mAP) and search cost on COCO dataset for object detection.

Backbone	Method	mAP	Search Time (hours)
ResNet-50	Baseline	36.9	0
	AA†	38.4	19800¶
	DDAS	38.1 ± 0.1	20^*
	DADA	not converge⋆	
ResNet-101	Baseline	38.6	0
	AA†	40.0	19800¶
	RA [9]	40.1	4600^*
	DDAS	39.8	20^*

¶ Evaluated on TPU. * Evaluated on 2080Ti.
† Reproduced with searched policy from [46].
⋆ Confirmed with authors of DADA

The results are reported in Table 5, which show that our DDAS could automatically learn the best p_{tp}. The results are slightly better than the best fixed p_{tp}. Moreover, as illustrated in Fig.5, different datasets prefer different p_{tp}s. Our method can save the effort of manually tuning p_{tp}.

Table 5. Test error rates (%) v.s. p_{tp}.

p_{tp}	0.2	0.4	0.6	0.8	1.0	p^*_{tp}
CIFAR-100	16.9	17.0	16.6	18.5	24.7	16.6
ImageNet	22.8	22.2	22.7	22.7	23.1	22.3

The effectiveness of dynamic policy. Then we show the necessity of dynamic policies. We fix p_{tp} and $\mathbf{p}_o$ to their last searched values, which is similar as AutoAugment. The results are summarized in Table 6. The results show that the dynamic policy indeed improves the performance slightly.

Table 6. Test error rates (%) v.s. Dynamic augmentation policy.

Fixed	p_{tp}	p_o	p_{tp}, p_o	Dynamic
CIFAR-100	16.8	16.7	16.9	16.6
ImageNet	22.4	22.2	22.4	22.3

6. Conclusion

In this paper, we propose Direct Differentiable Augmentation Search (DDAS) for automatic augmentation policy search. DDAS firstly reorganizes the search space with a two-level hierarchy, and then make the search differentibale with one-step gradient update meta-learning and continuous relaxation of the expected training loss. Our DDAS achieves state-of-the-art accuracy and efficiency tradeoff on image classification task on various datasets. Moreover, we are the first work to make efficient search on object detection while achieving competitive performance. We believe this concise and effective framework can serve as baseline for many subsequent explorations.

References

[1] David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. ReMixMatch: Semi-supervised learning with distribution alignment and augmentation anchoring. *arXiv preprint arXiv:1911.09785*, 2019.

[2] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. MixMatch: A holistic approach to semi-supervised learning. In *NeurIPS*, 2019.

[3] Kaifeng Bi, Changping Hu, Lingxi Xie, Xin Chen, Longhui Wei, and Qi Tian. Stabilizing DARTS with amended gradient estimation on architectural parameters. *ArXiv*, abs/1910.11831, 2019.

[4] Chih-Yang Chen, Che-Han Chang, and Edward Y Chang. Hypernetwork-based augmentation. *arXiv preprint arXiv:2006.06320*, 2020.

[5] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[7] Shuyang Cheng, Zhaoqi Leng, Ekin Dogus Cubuk, Barret Zoph, Chunyan Bai, Jiquan Ngiam, Yang Song, Benjamin Caine, Vijay Vasudevan, Congcong Li, et al. Improving 3d object detection through progressive population based augmentation. In *ECCV*, 2020.

[8] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. AutoAugment: Learning augmentation strategies from data. In *CVPR*, 2019.

[9] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. RandAugment: Practical automated data augmentation with a reduced search space. In *NeurIPS*, 2020.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

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

[12] Xavier Gastaldi. Shake-shake regularization. *arXiv preprint arXiv:1705.07485*, 2017.

[13] Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *ICLR*, 2018.

[14] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Faster AutoAugment: Learning augmentation strategies using backpropagation. In *ECCV*, 2020.

[15] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Meta approach to data augmentation optimization. *arXiv preprint arXiv:2006.07965*, 2020.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[17] Zhuoxun He, Lingxi Xie, Xin Chen, Ya Zhang, Yanfeng Wang, and Qi Tian. Data augmentation revisited: Rethinking the distribution gap between clean and augmented data. *arXiv preprint arXiv:1909.09148*, 2019.

[18] Daniel Ho, Eric Liang, Xi Chen, Ion Stoica, and Pieter Abbeel. Population based augmentation: Efficient learning of augmentation policy schedules. In *ICML*, 2019.

[19] Hiroshi Inoue. Data augmentation by pairing samples for images classification. *arXiv preprint arXiv:1801.02929*, 2018.

[20] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *ICLR*, 2016.

[21] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[22] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. *Technical report*, 2009.

[23] Ruihui Li, Xianzhi Li, Pheng-Ann Heng, and Chi-Wing Fu. PointAugment: an Auto-Augmentation framework for point cloud classification. In *CVPR*, 2020.

[24] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy M. Hospedales, Neil Martin Robertson, and Yongxin Yang. DADA: differentiable automatic data augmentation. In *ECCV*, 2020.

[25] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. Fast AutoAugment. In *NeurIPS*, 2019.

[26] Chen Lin, Minghao Guo, Chuming Li, Xin Yuan, Wei Wu, Junjie Yan, Dahua Lin, and Wanli Ouyang. Online hyper-parameter learning for auto-augmentation strategy. In *ICCV*, 2019.

[27] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014.

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *ICLR*, 2019.

[30] Saypraseuth Mounsaveng, Issam Laradji, Ismail Ben Ayed, David Vazquez, and Marco Pedersoli. Learning data augmentation with online bilevel optimization for image classification. *arXiv preprint arXiv:2006.14699*, 2020.

[31] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPSW*, 2011.

[32] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *ICML*, 2018.

[33] Zhiqiang Tang, Yunhe Gao, Leonid Karlinsky, Prasanna Sattigeri, Rogerio Feris, and Dimitris Metaxas. OnlineAugment: Online data augmentation with less domain knowledge. In *ECCV*, 2020.

[34] Keyu Tian, Chen Lin, Ming Sun, Luping Zhou, Junjie Yan, and Wanli Ouyang. Improving Auto-Augment via augmentation-wise weight sharing. In *NeurIPS*, 2020.

[35] Sen Wu, Hongyang R Zhang, Gregory Valiant, and Christopher Ré. On the generalization effects of linear transformations in data augmentation. *arXiv preprint arXiv:2005.00695*, 2020.

[36] Cihang Xie, Mingxing Tan, Boqing Gong, Jiang Wang, Alan L. Yuille, and Quoc V. Le. Adversarial examples improve image recognition. In *CVPR*, 2020.

[37] Qizhe Xie, Zihang Dai, E. Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training. *arXiv: Learning*, 2019.

[38] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: stochastic neural architecture search. In *ICLR*, 2018.

[39] Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

[40] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

[41] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. Mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

[42] Xinyu Zhang, Qiang Wang, Jian Zhang, and Zhao Zhong. Adversarial AutoAugment. In *ICLR*, 2020.

[43] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. In *NeurIPS*, 2020.

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

[45] Sharon Zhou, Jiequan Zhang, Hang Jiang, Torbjörn Lundh, and Andrew Y Ng. Data augmentation with mobius transformations. *arXiv preprint arXiv:2002.02917*, 2020.

[46] Barret Zoph, Ekin D Cubuk, Golnaz Ghiasi, Tsung-Yi Lin, Jonathon Shlens, and Quoc V Le. Learning data augmentation strategies for object detection. In *ECCV*, 2020.

[47] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.

A. Addtional Experiments

A.1. Experiment on SVHN

Search Setting. We also search augmentation on SVHN [31]. Similar as the experiments on CIFAR-10/100, we first search augmentation on a proxy task with a small network, Wide-ResNet-40-2 on part of the dataset for 20 epochs. We split 3000 images for training dataset S_{train} and 3000 images for S_{val}. The training mini-batch size is 32 while the validation mini-batch size is 256. At each step, we sample $\hat{L} = 3$ augmentation policies randomly according to a uniform distribution. The number of operation N_o in an augmentation policy is 2. The search is carried out on a single RTX 2080Ti GPU. As for initialization, we initialize $\mathbf{p}_o$ equally and p_{tp} as 0.35.

α_{tp} and $\boldsymbol{\alpha_o}$ are updated with Adam optimizers. The learning rate for $\boldsymbol{\alpha_o}$ is 0.005, and that for α_{tp} is 0.001. For the 2 optimizers, we set $\beta_1 = 0.5$, $\beta_2 = 0.999$. The network parameters of Wide-ResNet-40-2 are updated with SGD optimizer with momentum as 0.9. The learning rate is 0.05, with cosine decay, and the weight decay is 0.0005.

Training Setting. After search, we apply the searched augmentation to Wide-ResNet-28-10. We train the network for 160 epochs on the full training set and report the performance evaluated on test set. We set initial learning as 0.005, batch size as 128, momentum as 0.9, weight decay as 0.001, and cosine learning rate decay.

Result. The search cost and test error rate are shown in Table.7. We run the experiment for three times and report the average test error rate. Compared with other efficient automatic augmentation methods, our DDAS can also achieve comparable performance and efficiency.

A.2. The effectiveness of policy number $\hat{L}$.

Our DDAS achieves efficient search because it can search for good augmentation policies with limited sampled augmentation policies at each step ($\hat{L} = 2,\ 3$). We further explore the effectiveness of sampled augmentation policy number $\hat{L}$ on the performance of searched policies. We search for augmentation policies with different $\hat{L}$ values and show the results in Table 8. We can see that simply increasing the sampled augmentation policy number $\hat{L}$ does not increase the performance.

B. Implemntation Details

B.1. Image Classsification

Operations. Here we list the augmentation operations used for image classification.

- `Identity`
- `Rotate`
- `Posterize`
- `Sharpness`
- `Translate-x/y`
- `FlipLR`
- `AutoContrast`
- `Solarize`
- `Contrast`
- `Shear-x/y`
- `Invert`
- `FlipUD`
- `Equalize`
- `Color`
- `Brightness`
- `Smooth`
- `Blur`

Similar as AA [8], the operations are from PIL, a popular Python image library.[1] In addition, we add `Cutout` to search space of ImageNet, and use it defaultly with region size as 16 pixels on CIFAR-10/100 and SVHN.

Magnitude. As for the magnitude, we follow RA [9] and use the same linear scale for indicating the magnitude (strength) of each transformation. As mentioned in Method, we discretely sampled magnitude value, and the selected magnitude values for CIFAR-10/100, SVHN and ImageNet are listed in Table 9.

B.2. Object Detection

For object detection, we use the operations in [46]. The magnitude setting keeps the same as [46]. The selected magnitude values for COCO is listed in Table 9.

C. Experiments Details

C.1. CIFAR-10/100 & Sanity Check

Search Setting. As for search both α_{tp} and $\boldsymbol{\alpha_o}$ are updated with Adam optimizers. The learning rate for $\boldsymbol{\alpha_o}$ is 0.005, and that for α_{tp} is 0.001. For both of the 2 optimizers, we set $\beta_1 = 0.5$, $\beta_2 = 0.999$. The network parameters of Wide-ResNet-40-2 are updated with SGD optimizer with momentum as 0.9. The learning rate is 0.05, with cosine decay, and the weight decay is 0.0005.

As for Sanity Check, the settings are basically similar as that of CIFAR-10/100 experiments. The difference is that we only use $N_o = 1$, magnitude as 2 and p_{tp} initialized as 0.75.

Training Setting. Both of the 2 training networks are trained with SGD optimizer whose momentum is 0.9. The batch size is 128 and the cosine LR schedule is adopted for all the models. For Wide-ResNet-28-10, we set initial learning as 0.1 and weight decay as 0.0005. For Shake-Shake(26 2x96d), we set initial learning as 0.01 and weight decay as 0.001. The $p_t(1), ..., p_t(T_{max})$ are smoothed with a mean filter whose size $F_s = 2$.

C.2. ImageNet

Search Setting. Both α_{tp} and $\boldsymbol{\alpha_o}$ are optimized with Adam optimizer. The learning rate for $\boldsymbol{\alpha_o}$ is 0.003, and that for α_{tp} is 0.002. For the 2 optimizers, we set $\beta_1 = 0.5$, $\beta_2 = 0.999$. The network parameters of ResNet-18 are updated with SGD optimizer. The learning rate is 0.01, with cosine decay, the momentum is 0.9 and the weight decay is 0.0001.

[1] https://pillow.readthedocs.io/en/5.1.x/

Table 7. SVHN test error rates (%) and search cost (GPU hour). WRN is shorthand of Wide-ResNet.

	Baseline	Cutout	AA	PBA	Fast AA	RA	Faster AA	DADA	**DDAS**
Error									
WRN-28-10	1.5	1.3	1.1	1.2	1.1	1.0	1.2	1.2	1.2
Cost									
WRN-28-10	-	-	1000	1	1.5	-	0.06	0.1	0.1

Table 8. Sampled augmentation policy number $\hat{L}$ ablation.

$\hat{L}$	3	7	11
Error	16.6	16.8	17.1

Table 9. Manitudes sampled for different datasets

Dataset	mag
CIFAR-10/ CIFAR-100/ SVHN	$\{2, 6, 10, 14\}$
ImageNet	$\{7, 14\}$
COCO	$\{4, 6, 8\}$

Training Setting. As for training, the 2 networks are trained with SGD optimizer whose momentum is 0.9. We set initial learning rate as 0.2, batch size as 512, momentum as 0.9, weight decay as 0.0001 and step learning decay by 0.1 at epoch 90, 180 and 240. For fair comparison we reproduce RA on ResNet-50 with our training setting and report the result. The p_ts are smoothed with a mean filter whose size $F_s = 6$.

C.3. COCO

Search Setting. Both $\boldsymbol{\alpha_o}$ and α_{tp} are updated with Adam optimizers. The learning rate for $\boldsymbol{\alpha_o}$ is 0.001, and that for α_{tp} is 0.001. And we set $\beta_1 = 0.5,\ \beta_2 = 0.999$ for the 2 optimizers. The parameters of RetinaNet are updated with SGD optimizer. The learning rate is 0.04, with step learning decay at epoch 24 and 28, the momentum is 0.9 and the weight decay is 0.0001.

Training Setting. Both of the 2 training networks are trained with SGD optimizer whose momentum is 0.9. We set the initial learning rate as 0.08, batch size as 64, momentum as 0.9, weight decay as 0.0001 and step learning rate decay at epoch 120 and 140.