Differentiable Automatic Data Augmentation

Li, Y., Hu, G., wang, Y., Hospedales, T., Robertson, N., & Yang, Y. (2020). Differentiable Automatic Data Augmentation. In *European Conference on Computer Vision 2020: Proceedings* (Vol. 12367, pp. 580–595). (Lecture Notes in Computer Science). Springer. https://doi.org/10.1007/978-3-030-58542-6_35

Published in:
European Conference on Computer Vision 2020: Proceedings

Document Version:
Publisher's PDF, also known as Version of record

Queen's University Belfast - Research Portal:
Link to publication record in Queen's University Belfast Research Portal

DADA: Differentiable Automatic Data Augmentation

Yonggang Li*[1], Guosheng Hu*[2,3], Yongtao Wang†[1], Timothy Hospedales[4], Neil M. Robertson[2,3], and Yongxin Yang[4]

[1] Wangxuan Institute of Computer Technology, Peking University, Beijing, China
wyt@pku.edu.cn
[2] Anyvision, Queens Road, Belfast, United Kingdom
[3] Queens University of Belfast, Belfast, United Kingdom
[4] School of Informatics, The University of Edinburgh, Edinburgh, United Kingdom

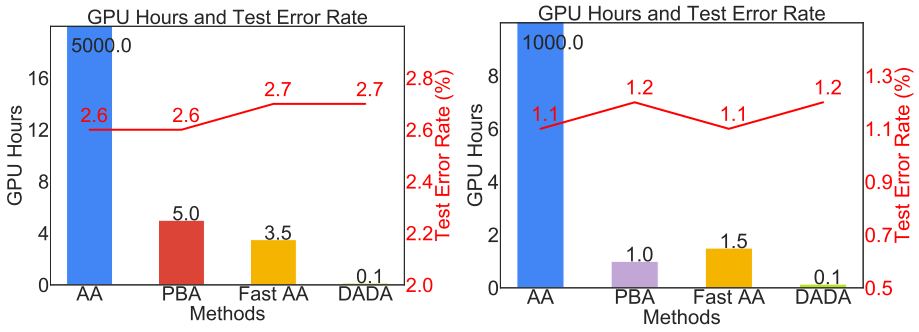
Abstract. Data augmentation (DA) techniques aim to increase data variability, and thus train deep networks with better generalisation. The pioneering AutoAugment automated the search for optimal DA policies with reinforcement learning. However, AutoAugment is extremely computationally expensive, limiting its wide applicability. Followup works such as Population Based Augmentation (PBA) and Fast AutoAugment improved efficiency, but their optimization speed remains a bottleneck. In this paper, we propose Differentiable Automatic Data Augmentation (DADA) which dramatically reduces the cost. DADA relaxes the discrete DA policy selection to a differentiable optimization problem via Gumbel-Softmax. In addition, we introduce an unbiased gradient estimator, RELAX, leading to an efficient and effective one-pass optimization strategy to learn an efficient and accurate DA policy. We conduct extensive experiments on CIFAR-10, CIFAR-100, SVHN, and ImageNet datasets. Furthermore, we demonstrate the value of Auto DA in pre-training for downstream detection problems. Results show our DADA is at least one order of magnitude faster than the state-of-the-art while achieving very comparable accuracy. The code is available at https://github.com/VDIGPKU/DADA.

Keywords: AutoML, Data Augmentation, Differentiable Optimization

1 Introduction

Data augmentation (DA) techniques, such as geometric transformations (e.g., random horizontal flip, rotation, crop), color space augmentations (e.g., color jittering), are widely applied in training deep neural networks. DA serves as a regularizer to alleviate over-fitting by increasing the amount and diversity of the training data. Moreover, it is particularly important in scenarios where big training data is not available, e.g. medical image analysis.

∗ indicates equal contribution.
† indicates corresponding author.



(a) CIFAR-10 (b) SVHN

Fig. 1: DADA achieves an order of magnitude reduction in computation cost while maintaining comparable test error rate to state of the art DA methods (AA [3], PBA [10] and Fast AA [15]) using WRN-28-10 backbone.

Data augmentation is very useful for training neural networks, however, it is nontrivial to automatically select the most effective DA policy (a set of augmentation operations) for one particular task and dataset. The pioneering work, AutoAugment (AA) [3], models the process of policy selection as an optimization problem: the objective is to maximize the accuracy on the validation set, the parameters optimized are i) the probabilities of applying different augmentation functions and ii) the magnitudes of chosen functions. Reinforcement learning is used to optimize this problem. AutoAugment achieved excellent performance on image classification, however, the optimization is very expensive: ∼5000 GPU hours for one augmentation search. Despite the effectiveness of AutoAugment, the heavy computational cost limits its value to most users. To address the efficiency problem, Population Based Augmentation (PBA) [10] and Fast AutoAugment (Fast AA) [15] are proposed. PBA introduces an efficient population based optimization, which was originally used for hyper-parameter optimization. Fast AA models the data augmentation search as a density matching problem and solves it through bayesian optimization. Though PBA and Fast AA greatly improve search speed, augmentation policy learning remains rather slow, e.g. PBA still needs ∼5 GPU hours for one search on the reduced CIFAR-10 dataset.

The inefficiency of the existing DA optimization strategies arises from the fact that the optimization (selecting discrete augmentation functions) is intrinsically non-differentiable, thus precluding joint optimization of network weights and DA parameters, and requiring resorting to multi-pass reinforcement learning, BayesOpt, and evolutionary strategies. Intuitively, optimization efficiency can be greatly improved if we can relax the optimization to a differentiable one and jointly optimize network weights and DA parameters in a single-pass way. Motivated by the differentiable neural architecture search [5,19,26], we propose a Differentiable Automatic Data Augmentation (DADA) to relax the optimization problem to be differentiable and then use gradient-based optimization to jointly

train model weights and data augmentation parameters. In this way, we can achieve a very efficient and effective DA policy search.

DADA follows AA [3] in using a search space where a policy contains many sub-policies, each of which has two operations (each with probability and magnitude of applying the DA function). DADA first reformulates the discrete search space to a joint distribution that encodes sub-policies and operations. Specifically, we treat the sub-policy selection and augmentation application as sampling from a Categorical distribution and Bernoulli distribution, respectively. In this way, DA optimization becomes a Monte Carlo gradient estimate problem [21]. However, Categorical and Bernoulli distributions are not differentiable. To achieve differentiable optimization, we relax the two non-differentiable distributions to be differentiable through Gumbel-Softmax gradient estimator [12] (a.k.a. concrete distribution [20]). Furthermore, DADA minimizes the loss on validation set rather than the accuracy (used by AutoAugment) to facilitate gradient computation.

To realise this differentiable optimization framework, we introduce 1) an efficient optimization strategy and 2) an accurate gradient estimator. For 1), a straightforward solution for sampling-based optimization is to iterate two sub-optimizations until convergence: i) optimizing DA policies ii) training neural networks. Clearly, this sequential optimization is very slow. Motivated by DARTS [19], we *jointly* optimize parameters of DA policies and networks through stochastic gradient descent. This *one-pass* strategy greatly reduces the computational cost. For 2), the classic gradient estimator is the Gumbel-Softmax estimator. In the field of network architecture search (NAS), SNAS [26] and GDAS [5] use this estimator and achieved good performance. However, the gradient estimated by Gumbel-Softmax estimator is biased. To overcome this, we propose to use the RELAX [7] estimator, which can provide an unbiased gradient estimator unlike Gumbel-Softmax, and thus improved policy search.

We conduct extensive experiments using a variety of deep models and datasets, e.g. CIFAR-10, SVHN [22]. As shown in Fig. 1, our method achieves comparable performance with the state-of-the-art, while requiring significantly less compute.

The contributions of this work are threefold:

1. We propose Differentiable Automatic Data Augmentation (DADA), which uses an efficient *one-pass* gradient-based optimization strategy and achieves at least one order of magnitude speedup over state-of-the-art alternatives.
2. DADA relaxes the DA parameter optimization to be differentiable via Gumbel-Softmax. To achieve accurate gradient estimation, we introduce an unbiased gradient estimator, RELAX [7], to our DADA framework.
3. We perform a thorough evaluation on CIFAR-10, CIFAR-100 [13], SVHN [22] and ImageNet [24], as well as object detection benchmarks. We achieve at least an order of magnitude speedup over state-of-the-art while maintaining accuracy. Specifically, on ImageNet, we only use 1.3 GPU (Titan XP) hours for searching and achieve 22.5% top1-error rate with ResNet-50.

2 Related Work

In the past few years, handcrafted data augmentation techniques are widely used in training deep Deep Neural Network (DNN) models for image recognition, object detection, etc. For example, rotation, translation, cropping, resizing, and flipping [14,25] are commonly used to augment training examples. Beyond these, there are other techniques manually designed with some domain knowledge, like Cutout [4], Mixup [30], and CutMix [28]. Although these methods achieve promising improvements on the corresponding tasks, they need expert knowledge to design the operations and set the hyper-parameters for specific datasets.

Recently, inspired by the neural architecture search (NAS) algorithms, some methods [3, 10, 15, 16] attempted to automate learning data augmentation policies. AutoAugment [3] models the policy search problem as a sequence prediction problem, and uses an RNN controller to predict the policy. Reinforcement learning (RL) is exploited to optimize the controller parameters. Though promising results are achieved, AutoAugment is extremely costly (e.g., 15000 GPU hours on ImageNet) due to the low efficiency of RL and multi-pass training. PBA [10] proposes to use population based training to achieve greater efficiency than AutoAugment, and evaluates on small datasets like CIFAR-10 and SVHN. OHL-Auto-Aug [16] employs online hyper-parameter learning in searching for an auto-augmentation strategy, leading to a speedup over AutoAugment of 60× on CIFAR-10 and 24× on ImageNet. Fast AutoAugment (Fast AA) [15] treats policy search as a density matching problem and applies Bayesian optimization to learn the policy, and achieves, e.g., 33× speedup over AutoAugment on ImageNet. Although Fast AA achieves encouraging results, its cost is still high on large datasets. E.g., 450 GPU (Tesla V100) hours on ImageNet.

Since our DADA is inspired by the differentiable neural architecture search [5, 19, 26], we briefly review these methods here. DARTS [19] first constructs a super-network of all possible operations and controls the super-network with architecture parameters. It then models neural architecture search as a bi-level optimization problem and optimizes architecture parameters through stochastic gradient descent. In order to remove the bias of DARTS for operation selection, SNAS [26] and GDAS [5] add stochastic factors to the super-network and utilize the Gumbel-Softmax gradient estimator [12, 20] to estimate the gradient. Our baseline method, which also uses the Gumbel-Softmax gradient estimator to optimize the data augmentation policy, is most motivated by these methods.

3 Differentiable Automatic Data Augmentation (DADA)

We first introduce the search space of DADA in Section 3.1. Then we model DA optimization as Monte Carlo sampling of DA policy in Section 3.2. After that, we introduce the Gumbel-Softmax [12, 20] as a relaxation of the categorical distribution in Section 3.3. In Section 3.4, we introduce an unbiased gradient estimator with small variance, RELAX [7], to compute gradients in our DADA framework. Finally, we propose an efficient *one-pass* optimization strategy to jointly optimize the network weights and the DA parameters in Section 3.5.

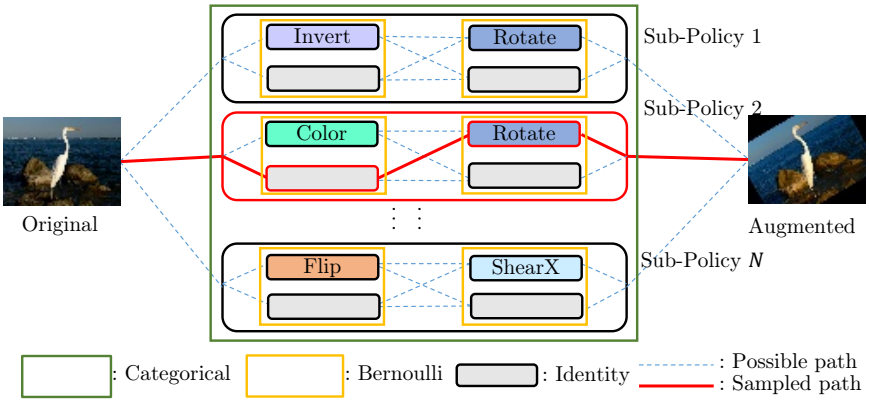


Fig. 2: The framework of DADA. The sub-policies and operations are sampled from Categorical and Bernoulli distributions respectively.

3.1 Search Space

Following Fast AutoAugment [15], a DA policy P contains several sub-policies s_i $(1 \leq i \leq |P|)$. Each sub-policy s includes k image operations (e.g. flipping, rotation) $\{\bar{O}_i^s(x; p_i^s, m_i^s)|1 \leq i \leq k\}$ which are applied in sequence. Each operation $\bar{O}_i^s$ is applied to the image x with two continuous parameters: p_i^s (the probability of applying the operation) and m_i^s (the magnitude of the operation):

$$\bar{O}_i^s(x; p_i^s, m_i^s) = \begin{cases} O_i^s(x; m_i^s) & : \text{with probability } p_i^s, \\ x & : \text{with probability } 1 - p_i^s. \end{cases} \tag{1}$$

Therefore the sub-policy s can be represented by a composition of operations:

$$\begin{aligned} s(x; \mathbf{p}^s, \mathbf{m}^s) =& \bar{O}_k^s(x; p_k^s, m_k^s) \circ \bar{O}_{k-1}^s(x; p_{k-1}^s, m_{k-1}^s) \circ \cdots \circ \bar{O}_1^s(x; p_1^s, m_1^s) \\ =& \bar{O}_k^s(\bar{O}_{k-1}^s(...\bar{O}_1^s(x; p_1^s, m_1^s); p_{k-1}^s, m_{k-1}^s); p_k^s, m_k^s). \end{aligned} \tag{2}$$

3.2 Policy Sampling from a Joint Distribution

We model the sub-policy selection and operation applying as sampling from Categorical and Bernoulli distributions, respectively. Then the DA optimization is modelled as a Monte Carlo gradient estimate problem from two distributions.

Sub-Policy Sampling Let $\mathcal{O}$ be the set of all image pre-processing operations in our search space. Then the candidate N sub-policies $\mathcal{S}$ are all the combinations of k operations in $\mathcal{O}$. To determine which sub-policies should be selected, we sample the sub-policy from a Categorical distribution $p(c|\pi)$ (where we can sample a one-hot random variable) with probabilities π. Probabilities are computed

as a softmax over parameters $\alpha = \alpha_{1:N}$ defining preference for sub-policies:

$$\bar{s}(x) = \sum_{s \in \mathcal{S}} c_s s(x), \quad c \sim p(c|\pi),\ \pi_s = \frac{exp(\alpha_s)}{\sum_{s' \in \mathcal{S}} exp(\alpha_{s'})}, \quad \text{for } s = 1, \ldots, N. \quad (3)$$

After α optimized, we can select the sub-policies with the top probabilities as the final DA policy P. Therefore, to optimize the DA policy, our task becomes optimizing the parameters α of the sub-policy sampling distribution.

Operation Sampling Given a particular sub-policy choice, the constituent operations are executed or not according to a Bernoulli distribution sample. We start from the simple example: a sub-policy s contains only one operation O_1^s. Then, the image operation O (we omit the superscript and subscript of O_1^s for simplicity) with application probability β and magnitude m can be represented:

$$s(x) = \bar{O}(x; m) = bO(x; m) + (1 - b)x, \quad b \sim \text{Bernoulli}(\beta). \quad (4)$$

To generalize Eq. (4) to a sub-policy with multiple operations, we formulate the composition of O_i and O_{i-1} in Eq. (2) as below:

$$\begin{aligned} \bar{O}_i(x; \beta_i, m_i) \circ \bar{O}_{i-1}(x; \beta_{i-1}, m_{i-1}) = {} & b_i b_{i-1} O_i(O_{i-1}(x; m_{i-1}); m_i) \\ & + b_i(1 - b_{i-1}) O_i(x; m_i) \\ & + (1 - b_i) b_{i-1} O_{i-1}(x; m_{i-1}) \\ & + (1 - b_i)(1 - b_{i-1}) x. \end{aligned} \quad (5)$$

where $b_{i-1} \sim \text{Bernoulli}(\beta_{i-1})$, $b_i \sim \text{Bernoulli}(\beta_i)$.

Optimization Objective With the above formulations, the DA policy can be sampled from a joint distribution $p(c, b|\alpha, \beta) = p(b|\beta, c)p(c|\alpha)$ (we use $p(c|\alpha)$ to describe Eq. (3)). Therefore, our objective can be represented as:

$$\mathbb{E}_{c,b \sim p(c,b|\alpha,\beta)}[\text{Reward}(c, b)] = \mathbb{E}_{c,b \sim p(c,b|\alpha,\beta)}[\mathcal{L}_w(c, b)]. \quad (6)$$

where $\mathcal{L}_w(c, b)$ is the *validation-set loss* achieved by the network (with weights w) which is trained using the DA policy $\{c, b\}$ on training set. Unlike AutoAugment which uses validation *accuracy* as reward, we use the validation *loss* to facilitate gradient computation.

3.3 Differentiable Relaxation with Gumbel-Softmax

To learn the data augmentation policy, we need to estimate the gradient of the objective w.r.t. the parameters $\{\alpha, \beta\}$ of the categorical (sub-policy) and bernoulli (operation) distributions. In this section, we use Gumbel-Softmax reparameterization trick [12] (a.k.a. concrete distribution [20]) to reparameterize the parameters $\{\alpha, \beta\}$ to be differentiable. Then we detail the estimation of the gradient of magnitudes m using the straight-through gradient estimator [1].

Differentiable Relaxation of Sub-Policy Selection The Categorical distribution is not differentiable w.r.t. the parameter α, therefore we use the Gumbel-Softmax reparameterization trick to achieve the differentiable relaxation. This reparameterization is also used in network architecture search, e.g SNAS [26] and GDAS [5]. With the Gumbel-Softmax reparameterization, Eq. (3) becomes:

$$\begin{aligned} \bar{s}(x) &= \sum_{s \in \mathcal{S}} c_s s(x), \\ c \sim \text{RelaxCategorical}(\alpha, \tau) &= \frac{\exp((\alpha_s + g_s)/\tau)}{\sum_{s' \in \mathcal{S}} \exp((\alpha_{s'} + g_{s'})/\tau)}, \end{aligned} \tag{7}$$

where $g = -\log(-\log(u))$ with $u \sim \text{Uniform}(0, 1)$, and τ is the temperature of softmax function. In our implementation, the straight-through gradient estimator is applied: the backward pass uses differentiable variables as Eq. (7), the forward pass uses discrete variables as shown:

$$\bar{s}(x) = \sum_{s \in \mathcal{S}} h_s s(x), \quad \text{where } h = \text{one\_hot}(\text{argmax}_s(\alpha_s + g_s)). \tag{8}$$

Differentiable Relaxation of Augmentation Operator Sampling Similar to the Categorical distribution, the Bernoulli distribution is not differentiable w.r.t. β. We apply the same reparameterization trick to the Bernoulli distribution:

$$\text{RelaxBernoulli}(\lambda, \beta) = \sigma((\log \frac{\beta}{1-\beta} + \log \frac{u}{1-u})/\lambda), \ u \sim \text{Uniform}(0, 1). \tag{9}$$

Similar to Eq. (8), we only execute the operation if $b > 0.5$ but backpropagate using the gradient estimated by Eq. (9).

Optimization of Augmentation Magnitudes Different from optimizing the discrete distribution parameters, we optimize augmentation magnitudes by approximating their gradient. Since some operations (e.g. flipping and rotation) in our search space are not differentiable w.r.t. the magnitude, we apply the straight-through gradient estimator [1] to optimize the magnitude. For an image $\hat{x} = s(x)$ augmented by sub-policy s, we approximate the gradient of magnitude (of the sampled operations) w.r.t. each pixel of the augmented image:

$$\frac{\partial \hat{x}_{i,j}}{\partial m} = 1. \tag{10}$$

Then the gradient of the magnitude w.r.t. our objective L can be calculated as:

$$\frac{\partial L}{\partial m} = \sum_{i,j} \frac{\partial L}{\partial \hat{x}_{i,j}} \frac{\partial \hat{x}_{i,j}}{\partial m} = \sum_{i,j} \frac{\partial L}{\partial \hat{x}_{i,j}}. \tag{11}$$

3.4 RELAX Gradient Estimator

Although the above reparameterization trick make DA parameters differentiable, its gradient estimate is biased [12, 20]. To address this problem, we propose to use RELAX [7] estimator which provides unbiased gradient estimate. We first describe the RELAX estimator for gradient estimation w.r.t distribution parameters. Then, we introduce the use of RELAX estimator to relax the parameters of Categorical $p(c|\alpha)$ and Bernoulli $p(b|\beta)$ distributions to be differentiable.

Gradient Estimation Here, we consider how to estimate the gradient of parameters of distribution $p(q|\theta)$, which can represent either $p(c|\alpha)$ or $p(b|\beta)$ in our algorithm. Unlike the Gumbel-Softmax estimator, the gradient of RELAX cannot simply be achieved by the backward of loss function. Furthermore, the RELAX estimator even does not require the loss function $\mathcal{L}$ to be differentiated w.r.t. the distribution parameters. It means we do not need to apply continuous relaxation the forward stage like Eq. (3) or Eq. (4). Instead, the RELAX estimator requires a differentiable neural network c_ϕ which is a surrogate of loss function, where ϕ are the parameters of neural network c. To make c_ϕ be differentiable w.r.t. the distribution parameters, we need the same Gumbel-Softmax reparameterized distribution $p(z|\theta)$ for $p(q|\theta)$, where θ is the distribution parameter and z is a sampled continuous variable. $p(z|\theta)$ can be either the Gumbel-Softmax reparameterized distributions of $p(c|\alpha)$ or $p(b|\beta)$. Then, in the forward stage, the DA policy can also be sampled from $z \sim p(z|\theta)$ but only forward the policy using deterministic mapping $q = H(z) = b \sim p(b|\theta)$ to guarantee the probability distribution curve is the same. Furthermore, since RELAX applies the control variate at the relaxed input z, we also need a relaxed input conditioned on the discrete variable q, denoted as $\tilde{z} \sim p(z|q, \theta)$. Then the gradient estimated by RELAX can be expressed as:

$$\begin{aligned} \nabla_\theta \mathcal{L} = [\mathcal{L}(q) - c_\phi(\tilde{z})] \nabla_\theta \log p(q|\theta) + \nabla_\theta c_\phi(z) - \nabla_\theta c_\phi(\tilde{z}), \\ q = H(z), z \sim p(z|\theta), \tilde{z} \sim p(z|q, \theta). \end{aligned} \tag{12}$$

To achieve a small variance estimator, the gradient of parameters ϕ can be computed as Eq. (13), which can be jointly optimized with the parameters θ.

$$\nabla_\phi(\text{Variance}(\nabla_\theta \mathcal{L})) = \nabla_\phi(\nabla_\theta \mathcal{L})^2. \tag{13}$$

Sub-Policy Sampling As shown in Eq. (3), the sub-policies can be sampled from the Categorical distribution. To apply the RELAX estimator, we first sample z from the relaxed Categorical distribution RelaxCategorical(α, τ) from Eq. (7), but we only forward the sub-policy with $c = \text{one\_hot}(\text{argmax}(z))$ in Eq. (8). We further sample $\bar{z}$ conditioned on variable c to control the variance:

$$\begin{aligned} \tilde{z}_s = \frac{\tilde{z}_s}{\sum_{s' \in \mathcal{S}} \tilde{z}_{s'}}, \\ \text{where} \quad \hat{z}_i = \begin{cases} -\log(-\log v_i) & c_i = 1 \\ -\log\left(-\frac{\log v_i}{p_i} - \log v_c\right) & c_i = 0 \end{cases}, \ v \sim \text{Uniform}(0, 1). \end{aligned} \tag{14}$$

Operation Sampling As shown in Eq. (4), the parameters b can be sampled from Bernoulli distribution. To adapt to the RELAX gradient estimator, we also utilize the Gumbel-Softmax reparameterization trick for $p(z|\beta)$ like Eq. (9), but only forward the operation with $b = \mathbb{I}(z > 0.5)$, where $\mathbb{I}$ is the indicator function. To control the variance, we sample $\tilde{z}$ conditioned as the sampled parameters b:

$$\tilde{z} = \sigma((\log \frac{\beta}{1-\beta} + \log \frac{v'}{1-v'})/\tau),$$
$$\text{where} \quad v' = \begin{cases} v \cdot (1-p), b = 0 \\ v \cdot p + (1-p), b = 1 \end{cases}, \; v \sim \text{Uniform}(0,1). \tag{15}$$

As for the gradient of magnitudes, we approximate them following Eq. (10). Since we need to estimate the gradient of the parameters of joint distribution $p(c, b|\alpha, \beta)$, we feed c_ϕ with the relaxed variables c and b in the same time for the surrogate of loss function.

3.5 Bi-Level Optimization

We have discussed the differentiable optimization of the DA policy in Section 3.1-3.4. We now discuss the joint optimization of DA policy and neural network. Clearly, it is very slow to sequentially iterate 1) the optimization of DA policy, 2) neural network training and performance evaluation until converge. Ideally, we conduct 1) and 2) by training neural network once, i.e. *one-pass* optimization. To achieve this, we propose the following bi-level optimization strategy.

Let $\mathcal{L}_{\text{train}}$ and $\mathcal{L}_{\text{val}}$ denote the training and validation loss, respectively. The optimization objective is to find the optimal parameters $d^* = \{\alpha^*, \beta^*, m^*, \phi^*\}$ which minimizes the validation loss $\mathcal{L}_{\text{val}}(w^*)$. The weights w^* are obtained by minimizing the expectation of training loss $w^* = \text{argmin}_w \, \mathbb{E}_{\bar{d}\sim p(\bar{d}|d)}[\mathcal{L}_{\text{train}}(w, \bar{d})]$ (m and ϕ are just the original parameters without sampling). For simplicity, we drop the sampling notation of the training loss as $\mathbb{E}[\mathcal{L}_{\text{train}}(w, d)]$. Therefore, our joint optimization objective can be represented as a bi-level problem:

$$\begin{aligned} &\min \;\; \mathcal{L}_{\text{val}}(w^*(d)), \\ &\text{s.t.} \quad w^*(d) = \text{argmin}_w \; \mathbb{E}[\mathcal{L}_{\text{train}}(w, d)]. \end{aligned} \tag{16}$$

Directly solving the above bi-level optimization problem would require repeatedly computing the model weights $w^*(d)$ as policy parameters d are changed. To avoid that, we optimize w and d alternately through gradient descent. At step k, give the current data augmentation parameters d_{k-1}, we obtain w_k by gradient descent w.r.t. expectation of training loss $\mathbb{E}[\mathcal{L}_{\text{train}}(w_{k-1}, d_{k-1})]$. Then, we approximate the above bi-level objective through a single virtual gradient step:

$$\mathcal{L}_{\text{val}}(w_k - \zeta \nabla_w \mathbb{E}[\mathcal{L}_{\text{train}}(w_k, d_{k-1})]). \tag{17}$$

Then, the gradient of Eq. (17) w.r.t. d is (with the step index k removed for simplicity):

$$-\zeta \nabla^2_{d,w} \mathbb{E}[\mathcal{L}_{\text{train}}(w, d)] \nabla_{w'} \mathcal{L}_{\text{val}}(w'), \tag{18}$$

where $w' = w - \zeta \nabla_w \mathbb{E}[\mathcal{L}_{\text{train}}(w, d)]$. The gradient is expensive to compute, therefore we use the finite difference approximation. Let ϵ be a small scalar, $w^+ = w + \epsilon \nabla_{w'} \mathcal{L}_{\text{val}}(w')$ and $w^- = w - \epsilon \nabla_{w'} \mathcal{L}_{\text{val}}(w')$. Then the gradient can be computed as:

$$-\zeta \frac{\nabla_d \mathbb{E}[\mathcal{L}_{\text{train}}(w^+, d)] - \nabla_d \mathbb{E}[\mathcal{L}_{\text{train}}(w^-, d)]}{2\epsilon}. \tag{19}$$

As for the gradients $\nabla_d \mathbb{E}[\mathcal{L}_{\text{train}}(w^+, d)]$ and $\nabla_d \mathbb{E}[\mathcal{L}_{\text{train}}(w^-, d)]$, they can be estimated by the techniques mentioned in Section 3.3 and Section 3.4. Specifically, we compute those two gradient with the same sampling sub-policy to make the difference of the two gradients more reliable. For the hyper-parameters ζ and ϵ, we follow the settings of another bi-level optimization DARTS [19], where $\zeta = \{\text{learning rate of } w\}$ and $\epsilon = 0.01/\|\nabla_{w'} \mathcal{L}_{\text{val}}(w')\|_2$.

4 Experiments

We compare our method with the effective baseline data augmentation method, Cutout [4], and the augmentation policy learners: AutoAugment [3] (AA), Population Based Augmentation [10] (PBA), Fast AutoAugment [15] (Fast AA), and OHL Auto-Aug [16] (OHL AA) on the CIFAR-10 [13], CIFAR-100 [13], SVHN [22] and ImageNet [24] datasets.

4.1 Settings

Search Space For the search space, we follow AA for a fair comparison. Specifically, our search space contains the same 15 data augmentation operations as PBA [10], which is also the same as AA and Fast AA except that the SamplePairing [11] is removed. Following AA, our sub-policy consists of two DA operations ($k = 2$), and the policy consists of 25 sub-polices.

Policy Search Following [3, 10, 15], we search the DA policies on the reduced datasets and evaluate on the full datasets. Furthermore, we split half of the reduced datasets as training set, and the remaining half as validation set for the data augmentation search. In the search stage, we search the policy parameters for 20 epochs on the reduced datasets. We use the Adam optimizer for the policy parameters $d = \{\alpha, \beta, m, \phi\}$ optimization with learning rate $\eta_d = 5 \times 10^{-3}$, momentum $\beta = (0.5, 0.999)$ and weight decay 0. For the optimization of neural network parameters, we use momentum SGD as optimizer, with the same hyper-parameters as evaluation stage except the batch size and the initial learning rate. In the search stage, we only apply the data augmentation policy to training examples, and set the batch size to 128 for CIFAR-10 and 32 for other datasets. The initial learning rate is set according to the batch size by the linear rule. We set τ to 0.5 and use a two layer fully connected neural network with 100 hidden units for c_ϕ. Parameters α are initialized to 10^{-3}. Parameters β and magnitudes m are all initialized to 0.5.

Table 1: GPU hours spent on DA policy search and corresponding test error (%). We use Wide-ResNet-28-10 model for CIFAR-10, CIFAR-100 and SVHN, and ResNet-50 for ImageNet. We use the *Titan XP* to estimate the search cost as PBA. AA and Fast AA reported the search cost on *Tesla P100* and *Tesla V100* GPU respectively. * : estimated.

(a) GPU hours

Dataset	AA [3]	PBA [10]	Fast AA [15]	OHL AA [16]	**DADA**
CIFAR-10	5000	5	3.5	83.4*	0.1
CIFAR-100	-	-	-	-	0.2
SVHN	1000	1	1.5	-	0.1
ImageNet	15000	-	450	625*	1.3

(b) Test set error rate (%)

Dataset	AA [3]	PBA [10]	Fast AA [15]	OHL AA [16]	**DADA**
CIFAR-10	2.6	2.6	2.7	2.6	2.7
CIFAR-100	17.1	16.7	17.3	-	17.5
SVHN	1.1	1.2	1.1	-	1.2
ImageNet	22.4	-	22.4	21.1	22.5

Table 2: CIFAR-10 and CIFAR-100 test error rates (%).

Dataset	Model	Baseline	Cutout [4]	AA [3]	PBA [10]	Fast AA [15]	**DADA**
CIFAR-10	Wide-ResNet-40-2	5.3	4.1	3.7	-	3.6	3.6
CIFAR-10	Wide-ResNet-28-10	3.9	3.1	2.6	2.6	2.7	2.7
CIFAR-10	Shake-Shake(26 2x32d)	3.6	3.0	2.5	2.5	2.7	2.7
CIFAR-10	Shake-Shake(26 2x96d)	2.9	2.6	2.0	2.0	2.0	2.0
CIFAR-10	Shake-Shake(26 2x112d)	2.8	2.6	1.9	2.0	2.0	2.0
CIFAR-10	PyramidNet+ShakeDrop	2.7	2.3	1.5	1.5	1.8	1.7
CIFAR-100	Wide-ResNet-40-2	26.0	25.2	20.7	-	20.7	20.9
CIFAR-100	Wide-ResNet-28-10	18.8	18.4	17.1	16.7	17.3	17.5
CIFAR-100	Shake-Shake(26 2x96d)	17.1	16.0	14.3	15.3	14.9	15.3
CIFAR-100	PyramidNet+ShakeDrop	14.0	12.2	10.7	10.9	11.9	11.2

Policy Evaluation We use the official publicly available code of Fast AA to evaluate the searched DA policies for a fair comparison. Following Fast AA [15] and AA [3], we use the same hyper-parameters for policy evaluation: weight decay, learning rate, batch size, training epoch.

4.2 Results

CIFAR-10 and CIFAR-100 Both CIFAR-10 and CIFAR-100 have $50,000$ training examples. Following [3, 10, 15], we conduct DA optimization on the reduced CIFAR-10 dataset ($4,000$ randomly selected examples) and evaluate the trained policies on the full CIFAR-10 test set. [3,10,15] use the discovered policies from CIFAR-10 and evaluate these policies on CIFAR-100. Since our DADA is much more efficient than other methods, thus, we conduct both the search (using a reduced dataset of $4,000$ randomly selected examples) and evaluation on CIFAR-100. Following [3,10,15], we search the DA policy using a Wide-ResNet-40-2 network [29] and evaluate the searched policy using Wide-ResNet-40-2 [29], Wide-ResNet-28-10 [29], Shake-Shake [6] and PyramidNet+ShakeDrop [27].

From the results in Table 1 and 2, we can see that DADA requires significantly less computation than the competitors, while providing comparable error rate.

Table 3: SVHN test error rates (%).

Model	Baseline	Cutout [4]	AA [3]	PBA [10]	Fast AA [15]	**DADA**
Wide-ResNet-28-10	1.5	1.3	1.1	1.2	1.1	1.2
Shake-Shake(26 2x96d)	1.4	1.2	1.0	1.1	-	1.1

Table 4: Validation set Top-1 / Top-5 error rate (%) on ImageNet using ResNet-50 and DA policy search time (h). *: Estimated.

	Baseline	AA [3]	Fast AA [15]	OHL AA [16]	**DADA**
Error Rate (%)	23.7 / 6.9	22.4 / 6.2	22.4 / 6.3	21.1 / 5.7	22.5 / 6.5
Search Time (h)	0	15000	450	625*	1.3

For example, we require only 0.1 GPU hours for policy search on reduced CIFAR-10, which is at least one order of magnitude faster than AA ($50,000\times$) and Fast AA ($35\times$). Similar to CIFAR-10, we achieve very competitive performance on CIFAR-100 yet with much less searching cost. Despite the lower error rates of OHL AA, OHL AA is not directly comparable to other methods since it uses a larger and dynamic search space.

SVHN To verify the generalization ability of our search algorithm for different datasets, we further conduct experiments with a larger dataset: SVHN. SVHN dataset has $73,257$ training examples ('core training set'), $531,131$ additional training examples, and $26,032$ testing examples. Following AA and Fast AA, we also search the DA policy with the reduced SVHN dataset, which has $1,000$ randomly selected training samples from the core training set. Following AA, PBA and Fast AA, we evaluate the learned DA policy performance with the full SVHN training data. Unlike AA, we use the Wide-ResNet-28-10 [29] architecture in the search stage and evaluate the policy on the Wide-ResNet-28-10 [29] and Shake-Shake (26 2x96d) [6]. Our results are shown in Table 3. As shown in Table 3, our DADA achieves similar error rate to PBA, slightly worse than AA and Fast AA. However, we only use 0.1 GPU hours in the search stage.

ImageNet Finally, we evaluate our algorithm on the ImageNet dataset. We train the policy on the same ImageNet subset as Fast AA, which consists of $6,000$ examples from a randomly selected 120 classes. We use ResNet-50 [9] for policy optimization and report the performance trained with the full ImageNet dataset. As shown in Table 1 and Table 4, we achieve very competitive error rate against AA and Fast AA while requiring only 1.3 GPU hours in the search stage – compared to 15000 and 450 hours respectively for these alternatives. Again, OHL AA [16] uses a larger and dynamics search space, thus, it is not comparable to other methods.

Table 5: Object detection bounding box (bb) and mask AP on COCO test-dev.

Method	Model	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}	AP^{bb}_{S}	AP^{bb}_{M}	AP^{bb}_{L}	AP^{mask}
RetinaNet	ResNet-50 (baseline)	35.9	55.8	38.4	19.9	38.8	45.0	-
RetinaNet	ResNet-50 (DADA)	**36.6**$^{+0.7}$	56.8	39.2	20.2	39.7	46.0	-
Faster R-CNN	ResNet-50 (baseline)	36.6	58.8	39.6	21.6	39.8	45.0	-
Faster R-CNN	ResNet-50 (DADA)	**37.2**$^{+0.6}$	59.1	40.2	22.2	40.2	45.7	-
Mask R-CNN	ResNet-50 (baseline)	37.4	59.3	40.7	22.0	40.6	46.3	34.2
Mask R-CNN	ResNet-50 (DADA)	**37.8**$^{+0.4}$	59.6	41.1	22.4	40.9	46.6	**34.5**$^{+0.3}$

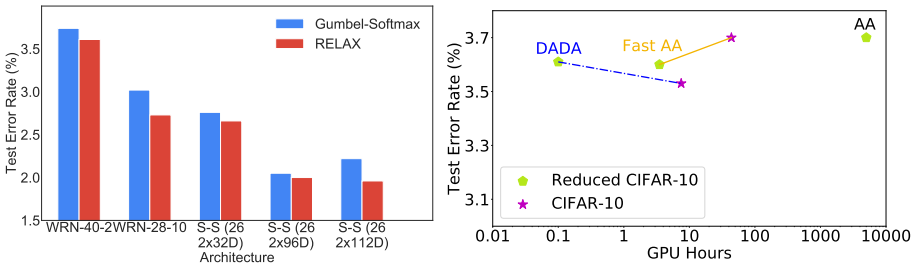
4.3 DADA for Object Detection

The use of ImageNet pre-training backbone networks is a common technique for object detection [8, 17, 23]. To improve the performance of object detection, people usually focus on designing a better detection pipeline, while paying less attention to improving the ImageNet pre-training backbones. It is interesting to investigate whether the backbones trained using our DADA can improve detection performance. With our DADA algorithm, we have reduced the Top-1 error rate of ResNet-50 on ImageNet from 23.7% to 22.5%. In this section, we further conduct experiments on object detection dataset MS COCO [18] with the better ResNet-50 model. We adopt three mainstream detectors RetinaNet [17], Faster R-CNN [23] and Mask R-CNN [8] in our experiments. For the same detector, we use the same setting as [2] except that the ResNet-50 model is trained with or without DADA policy. From Table 5, the performance of ResNet-50 trained with DADA policy is consistently better than the ResNet-50 trained without DADA policy. The results show that our learned DA policy also improves generalisation performance of downstream deep models that leverage the pre-trained feature.

4.4 Further Analysis

Comparison with the Gumbel-Softmax Estimator One technical contribution of this paper is the derivation of a RELAX estimator for DA policy search, which removes the bias of the conventional Gumbel-Softmax estimator. To evaluate the significance of this contribution, we conduct the search experiments for both estimators with the same hyper-parameters on the CIFAR-10 dataset. As we can see from Fig. 3a, the policy found using our RELAX estimator achieves better performance on CIFAR-10 compared with Gumbel-Softmax estimator.

Search on the Full Dataset We further evaluate the performance when we train the DA policy on the full dataset rather than on the reduced one, noting that this is feasible for the first time with DADA, due to its dramatically increased efficiency compared to alternatives. We conduct DA policy search on both the reduced and full CIFAR-10 dataset with Wide-ResNet-40-2. As we can



(a) Comparison with Gumbel-Softmax.. (b) Error rate vs policy search cost.

Fig. 3: Additional Analysis on DADA.

Table 6: The test error rate (%) with DA policy learned on different training set.

Dataset	Model	Reduced CIFAR-10	Full CIFAR-10
CIFAR-10	Wide-ResNet-40-2	3.61	3.53
CIFAR-10	Wide-ResNet-28-10	2.73	2.64

see from Table 6, the policy searched on the full dataset works better than that on the reduced one evaluated on CIFAR-10.

We finally bring together some of these results and compare the speed accuracy trade-off provided by DADA, Fast AA and AA in Fig. 3b for CIFAR-10 on WRN-40-2 architecture. Fast AA does not benefit from DA policy on the full CIFAR-10 dataset. However, DADA provides an excellent tradeoff, especially at low resource operating points.

5 Conclusion

In this work, we proposed Differentiable Automatic Data Augmentation (DADA) for data augmentation policy learning. DADA relaxes the discrete policy selection process to be differentiable using Gumbel-Softmax. To achieve efficient and accurate optimization, we propose a *one-pass* optimization strategy. In our differentiable optimization framework, we introduce an unbiased gradient estimator RELAX to achieve an accurate gradient estimation. Experimental results show that DADA achieves comparable image classification accuracy to state-of-the-art with at least one order of magnitude less search cost. DADA's greater efficiency makes it the first practical Auto-DA tool of choice that practitioners can use to optimize DA pipelines for diverse applications on desktop-grade GPUs.

Acknowledgment

This work is supported by National Natural Science Foundation of China under Grant 61673029.

References

1. Bengio, Y., Léonard, N., Courville, A.C.: Estimating or propagating gradients through stochastic neurons for conditional computation. CoRR **abs/1308.3432** (2013)
2. Chen, K., Wang, J., Pang, J., Cao, Y., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Xu, J., Zhang, Z., Cheng, D., Zhu, C., Cheng, T., Zhao, Q., Li, B., Lu, X., Zhu, R., Wu, Y., Dai, J., Wang, J., Shi, J., Ouyang, W., Loy, C.C., Lin, D.: Mmdetection: Open mmlab detection toolbox and benchmark. CoRR **abs/1906.07155** (2019)
3. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: Autoaugment: Learning augmentation strategies from data. In: CVPR (2019)
4. Devries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout. CoRR **abs/1708.04552** (2017)
5. Dong, X., Yang, Y.: Searching for a robust neural architecture in four gpu hours. In: CVPR (2019)
6. Gastaldi, X.: Shake-shake regularization of 3-branch residual networks. In: ICLR (2017)
7. Grathwohl, W., Choi, D., Wu, Y., Roeder, G., Duvenaud, D.: Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In: ICLR (2018)
8. He, K., Gkioxari, G., Dollár, P., Girshick, R.B.: Mask R-CNN. In: ICCV (2017)
9. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR (2016)
10. Ho, D., Liang, E., Chen, X., Stoica, I., Abbeel, P.: Population based augmentation: Efficient learning of augmentation policy schedules. In: ICML (2019)
11. Inoue, H.: Data augmentation by pairing samples for images classification. CoRR **abs/1801.02929** (2018)
12. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. In: ICLR (2017)
13. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
14. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)
15. Lim, S., Kim, I., Kim, T., Kim, C., Kim, S.: Fast autoaugment. In: NeurIPS (2019)
16. Lin, C., Guo, M., Li, C., Yuan, X., Wu, W., Yan, J., Lin, D., Ouyang, W.: Online hyper-parameter learning for auto-augmentation strategy. In: ICCV (2019)
17. Lin, T., Goyal, P., Girshick, R.B., He, K., Dollár, P.: Focal loss for dense object detection. In: ICCV (2017)
18. Lin, T., Maire, M., Belongie, S.J., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: common objects in context. In: ECCV (2014)
19. Liu, H., Simonyan, K., Yang, Y.: DARTS: differentiable architecture search. In: ICLR (2019)
20. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. In: ICLR (2017)
21. Mohamed, S., Rosca, M., Figurnov, M., Mnih, A.: Monte carlo gradient estimation in machine learning. CoRR **abs/1906.10652** (2019)
22. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning (2011)
23. Ren, S., He, K., Girshick, R.B., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. In: NeurIPS (2015)

24. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) **115**(3), 211–252 (2015)
25. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
26. Xie, S., Zheng, H., Liu, C., Lin, L.: SNAS: stochastic neural architecture search. In: ICLR (2019)
27. Yamada, Y., Iwamura, M., Akiba, T., Kise, K.: Shakedrop regularization for deep residual learning. IEEE Access **7**, 186126–186136 (2019)
28. Yun, S., Han, D., Oh, S.J., Chun, S., Choe, J., Yoo, Y.: Cutmix: Regularization strategy to train strong classifiers with localizable features. In: ICCV (2019)
29. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: BMVC (2016)
30. Zhang, H., Cissé, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond empirical risk minimization. In: ICLR (2018)