Faster AutoAugment: Learning Augmentation Strategies Using Backpropagation

Ryuichiro Hataya[1,2], Zdenek Jan[1], Kazuki Yoshizoe[2], Hideki Nakayama[1]

[1] Graduate School of Information Science and Technology, The University of Tokyo
{hataya,jan,nakayama}@nlab.ci.i.u-tokyo.ac.jp
[2] RIKEN Center for Advanced Intelligence Project
kazuki.yoshizoe@riken.jp

Abstract. Data augmentation methods are indispensable heuristics to boost the performance of deep neural networks, especially in image recognition tasks. Recently, several studies have shown that augmentation strategies found by search algorithms outperform hand-made strategies. Such methods employ black-box search algorithms over image transformations with continuous or discrete parameters and require a long time to obtain better strategies. In this paper, we propose a differentiable policy search pipeline for data augmentation, which is much faster than previous methods. We introduce approximate gradients for several transformation operations with discrete parameters as well as a differentiable mechanism for selecting operations. As the objective of training, we minimize the distance between the distributions of augmented and original data, which can be differentiated. We show that our method, Faster AutoAugment, achieves significantly faster searching than prior methods without a performance drop.

1 Introduction

Data augmentation is a powerful technique for machine learning to virtually increase the amount and diversity of data, which improves performance especially in image recognition tasks. Conventional data augmentation methods include geometric transformations such as rotation and color enhancement such as auto-contrast. Similarly to selecting other hyperparameters, the designers of data augmentation strategies usually select transformation operations based on their prior knowledge (e.g., required invariance). For example, horizontal flipping is expected to be effective for general object recognition but probably not for digit recognition. In addition to the selection, the designers need to combine several operations and set their magnitudes (e.g., degree of rotation). Therefore, designing of data augmentation strategies is a complex combinatorial problem.

When designing data augmentation strategies in a data-driven manner, one can regard the problem as searching for optimal hyperparameters in a search space, which becomes prohibitively large as the number of combinations increases. Therefore, efficient methods are required to find optimal strategies. If

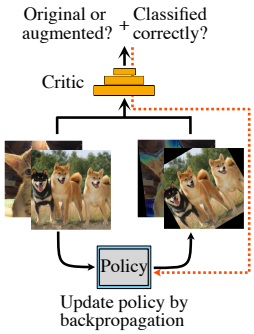

Fig. 1. Overview of our proposed model. We propose to use a **differentiable data augmentation pipeline** to achieve a faster policy search by using adversarial learning.

Table 1. Faster AutoAugment (Faster AA) is much faster than the other methods without a significant performance drop (see Section 5). GPU hours comparison of Faster AA, AutoAugment (AA) [5], PBA [12] and Fast AutoAugment (Fast AA) [18].

Dataset	AA	PBA	Fast AA	Faster AA (ours)
CIFAR-10	5,000	5.0	3.5	**0.23**
SVHN	1,000	1.0	1.5	**0.061**
ImageNet	15,000	-	450	**2.3**

gradient information of these hyperparameters is available, they can be efficiently optimized by gradient descent [20]. However, the gradient information is usually difficult to obtain because some magnitude parameters are discrete and the selection of operations is non-differentiable. Therefore, previous research on automatically designing data augmentation policies has used black-box optimization methods that require no gradient information. For example, AutoAugment [5] used reinforcement learning.

In this paper, we propose to solve the problem by approximating gradient information and thus enabling gradient-based optimization for data augmentation policies. To this end, we approximate the gradients of discrete image operations using a straight-through estimator [3] and make the selection of operations differentiable by incorporating a recent differentiable neural architecture search method [19]. As the objective, we minimize the distance between the distributions of the original and augmented images, because we want the data augmentation pipeline to transform images so that it fills sparsely populated data points in the training data [18] (see Figure 2). To make the transformed images match the distribution of original images, we use adversarial learning (see Figure 1). As a result, the search process becomes end-to-end differentiable and

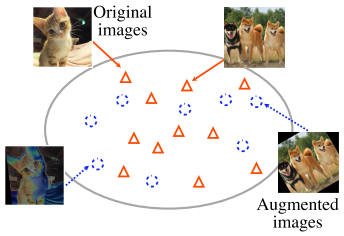

Fig. 2. We regard data augmentation as a process that **fills missing data points of the original training data**; therefore, our objective is to minimize the distance between the distributions of augmented and original data using adversarial learning.

significantly **faster** than prior methods such as AutoAugment, PBA and Fast AutoAugment (see Table 1 [3]).

We empirically show that our method, which we call Faster AutoAugment, enables a much faster policy search while achieving performance comparable to that of prior methods on the standard benchmarks of CIFAR-10, CIFAR-100 [16], SVHN [21] and ImageNet [27].

In summary, our contributions are threefold:

1. We introduce gradient approximations for several non-differentiable data augmentation operations.
2. We make the searching of data augmentation policies end-to-end differentiable by gradient approximations, a differentiable selection of operations and a differentiable objective that measures the distance between the original and augmented image distributions.
3. We show that our proposed method, Faster AutoAugment, significantly reduces the search time compared with prior methods without a crucial performance drop.

2 Related Work

Neural Architecture Search (NAS) NAS aims to automatically design architectures of neural networks to achieve a higher performance than manually designed ones. To this end, NAS algorithms are required to select better combinations of components (e.g., convolution with a 3x3 kernel) from discrete search spaces using search algorithms such as reinforcement learning [38] and evolution strategy [25]. Recently, DARTS [19] achieved a faster search by relaxing the discrete search space to a continuous one which allowed the use of gradient-based optimization. While AutoAugment [5] was inspired by [38], our method is influenced by DARTS [19].

[3] Note that [18] and our study estimated the GPU hours with an NVIDIA V100 GPU while [5] did with an NVIDIA P100 GPU.

Data Augmentation Data augmentation methods improve the performance of learnable models by increasing the virtual size and diversity of training data without collecting additional data samples. Traditionally, geometric and color-enhancing transformations have been used in image recognition tasks. For example, [17,11] randomly applied horizontal flipping and cropping as well as the alternation of image hues. In recent years, other image manipulation methods have been shown to be effective. [37,6] cut out a random patch from an image and replaced it with random noise or a constant value. Another strategy is to mix multiple images of different classes either by convex combinations [36,30] or by creating a patchwork from them [34]. In these studies, the selection of operations, their magnitudes and the probabilities to be applied were carefully manually designed.

Automating Data Augmentation Similar to NAS, it is a natural direction to aim to automate data augmentation. One direction is to search for better combinations of symbolic operations using black-box optimization techniques such as reinforcement learning [5,24], evolution strategy [32], Bayesian optimization [18] and population-based training [12]. As the objective, [5,32,12] directly aimed to minimize the error rate, or equivalently to maximize accuracy, while [24,18] attempted to match the densities of augmented and original images. Another direction is to use generative adversarial networks (GANs) [9]. [31,1] used conditional GANs to generate images that promote the performance of image classifiers. [28,29] used GANs to modify the outputs of simulators to look like real objects. Automating data augmentation can also be applied to representation learning such as semi-supervised learning [4,33] and domain generalization [32].

3 Preliminaries

In this section, we describe the common basis of AutoAugment [5], PBA [12] and Fast AutoAugment [18] (see also Figure 3). Faster AutoAugment also follows this problem setting.

In these methods, input images are augmented by a policy that consists of L different subpolicies $S^{(l)}$ $(l = 1, 2, \ldots, L)$. A randomly selected subpolicy transforms each image $\boldsymbol{X}$. A single subpolicy consists of K consecutive image processing operations $O_1^{(l)}, \ldots, O_K^{(l)}$, which are applied to the image one by one. We refer to the number of consecutive operations K as the operation count. In the rest of this paper, we focus on subpolicies; therefore, we omit the superscript l. Each method first searches for better policies. After the search phase, the obtained policy is used as a data augmentation pipeline to train neural networks.

3.1 Operations

Operations used in each subpolicy include affine transformations such as `shear_x` and color-enhancing operations such as `solarize`. Additionaly, we use `cutout` [6]

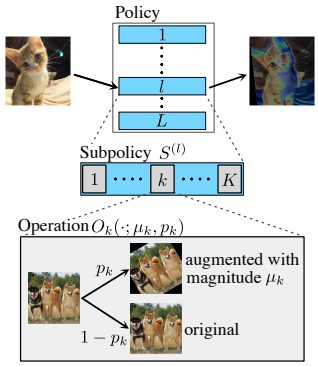

Fig. 3. Schematic view of the problem setting. Each image is augmented by a **subpolicy** randomly selected from the **policy**. A single subpolicy is composed of K consecutive **operations** $(O_1, \ldots, O_K)$, such as `shear_x` and `solarize`. An operation O_k operates a given image with the probability p_k and magnitude μ_k.

and `sample_pairing` [13] following [5,12,18]. We show all 16 operations used in these methods in Table 2. Let the set of operations be $\mathcal{O} = \{\texttt{shear\_x}, \texttt{solarize}, \ldots\}$.

Some operations have magnitude parameters that are free variables, e.g., the angle in `rotate`. On the other hand, some operations, such as `invert`, have no magnitude parameters. For simplicity, we use the following expressions as if every operation has a magnitude parameter $\mu_O (\in [0,1])$. Each operation is applied with the probability $p_O (\in [0,1])$. Therefore, each image $\boldsymbol{X}$ is augmented as

$$\boldsymbol{X} \to \begin{cases} O(\boldsymbol{X}; \mu_O) & \text{(with probability } p_O\text{)} \\ \boldsymbol{X} & \text{(with probability } 1 - p_O\text{).} \end{cases} \tag{1}$$

Rewriting this mapping as $O(\cdot; \mu_O, p_O)$, each subpolicy S consisting of operations $O_1, O_2, \ldots, O_K$ can be written as

$$S(\boldsymbol{X}; \boldsymbol{\mu}_S, \boldsymbol{p}_S) = (O_K \circ \cdots \circ O_1)(\boldsymbol{X}; \boldsymbol{\mu}_S, \boldsymbol{p}_S), \tag{2}$$

where $\boldsymbol{\mu}_S = (\mu_{O_1}, \ldots, \mu_{O_K})$ and $\boldsymbol{p}_S = (p_{O_1}, \ldots, p_{O_K})$. In the rest of this paper, we represent an image operation as O, $O(\cdot; \mu)$ and $O(\cdot; \mu, p)$ interchangeably according to the context.

3.2 Search Space

The goal of searching is to find the best operation combination $O_1, \ldots, O_K$ and parameter sets $(\boldsymbol{\mu}_S, \boldsymbol{p}_S)$ for L subpolicies. Therefore, the size of the total search space is roughly $(\#\mathcal{O} \times [0,1] \times [0,1])^{KL}$. Using multiple subpolicies results in a prohibitively large search space for brute-force searching. [18] used Bayesian optimization in this search space. [5,12] discretized the continuous part $[0,1]$

Table 2. Operations used in AutoAugment, PBA, Fast AutoAugment and Faster AutoAugment. Some operations have discrete magnitude parameters μ, while others have no or continuous magnitude parameters. Different from previous methods, we approximate gradients of operations w.r.t. discrete magnitude μ, which we describe in Section 4.1.

	Operation	Magnitude μ
Affine transformation	`shear_x`	continuous
	`shear_y`	continuous
	`translate_x`	continuous
	`translate_y`	continuous
	`rotate`	continuous
	`flip`	none
Color enhancing operations	`solarize`	discrete
	`posterize`	discrete
	`invert`	none
	`contrast`	continuous
	`color`	continuous
	`brightness`	continuous
	`sharpness`	none
	`auto_contrast`	none
	`equalize`	none
Other operations	`cutout`	discrete
	`sample_pairing`	continuous

into 10 or 11 values and searched the space using reinforcement learning and population-based training. Nevertheless, the problem is still difficult to solve naively even after discretizing the search space. For instance, if the number of subpolicies L is 10 with $K = 2$ consecutive operations, the discretized space size becomes $(16 \times 10 \times 11)^{2\times 10} \approx 8.1 \times 10^{64}$.

Previous methods [5,12,18] used black-box optimization. Therefore, they needed to train CNNs with candidate policies and obtain their validation accuracy. The repetition of this process requires a long time. In contrast, Faster AutoAugment achieves a faster search with gradient-based optimization to avoid repetitive evaluations, even though the search space is the same as that in Fast AutoAugment. We describe the details of Faster AutoAugment in the next section.

4 Faster AutoAugment

Faster AutoAugment explores the search space to find better policies in a gradient-based manner, which distinguishes our method. In Section 4.1, we describe the details of the gradient approximation for policy searching. To accomplish gradient-based training, we adopt distance minimization between the distributions of the augmented and original images as the learning objective, which we present in Section 4.2.

4.1 Differentiable Data Augmentation Pipeline

Previous search methods [5,12,18] have used image processing libraries (e.g., Pillow) that do not support backpropagation through the operations in Table 2. Contrary to previous methods, we modify these operations to be differentiable with respect to the probability p and magnitude μ. Thanks to this modification, the search problem becomes an optimization problem. The sequence of operations in each subpolicy also needs to be optimized in the same manner.

Probability parameter p First, we regard equation 1 as $bO(\boldsymbol{X};\mu)+(1-b)\boldsymbol{X}$, where $b \in \{0,1\}$ is sampled from the Bernoulli distribution $\mathrm{Bern}(b;p)$, i.e., $b=1$ with the probability p. Since this distribution is non-differentiable, we instead use the relaxed Bernoulli distribution [14]

$$\mathrm{ReBern}(b;p,\lambda) = \varsigma(\frac{1}{\lambda}\{\log\frac{p}{1-p} + \log\frac{u}{1-u}\}). \tag{3}$$

Here, $\varsigma(x) = \dfrac{1}{1+\exp(-x)}$ is a sigmoid function that keeps the range of the function in $(0,1)$ and u is a value sampled from a uniform distribution on $[0,1]$. At a low temperature λ, this relaxed distribution behaves similarly to a Bernoulli distribution. Using this reparameterization, each operation $O(\cdot;\mu_O,p_O)$ can be differentiable w.r.t. its probability parameter p.

Magnitude parameter μ For some operations, such as `rotate` or `translate_x`, their gradients w.r.t. their magnitude parameter μ can be obtained easily. However, some operations such as `posterize` and `solarize` discretize magnitude values. In such cases, gradients w.r.t. μ cannot backpropagate through these operations. Thus, we approximate their gradient in a similar manner to the straight-through estimator [3,22]. More precisely, we approximate the (i,j)th element of an augmented image by an operator O as

$$\tilde{O}(\boldsymbol{X};\mu)_{i,j} = \mathrm{StopGrad}(O(\boldsymbol{X};\mu)_{i,j} - \mu) + \mu, \tag{4}$$

where StopGrad is a stop gradient operation, which treats its operand as a constant. During the forward computation, the augmentation is exactly operated: $\tilde{O}(\boldsymbol{X};\mu)_{i,j} = O(\boldsymbol{X};\mu)_{i,j}$. However, during the backward computation, the first term of the right-hand side of equation 4 is ignored because it is constant, and then we obtain an approximated gradient:

$$\frac{\partial O(\boldsymbol{X})_{i,j}}{\partial \mu} \approx \frac{\partial \tilde{O}(\boldsymbol{X})_{i,j}}{\partial \mu} = 1. \tag{5}$$

Despite its simplicity, we find that this method works well in our experiments. Using this approximation, each operation $O(\cdot;\mu_O,p_O)$ can be differentiable w.r.t. its magnitude parameter μ.

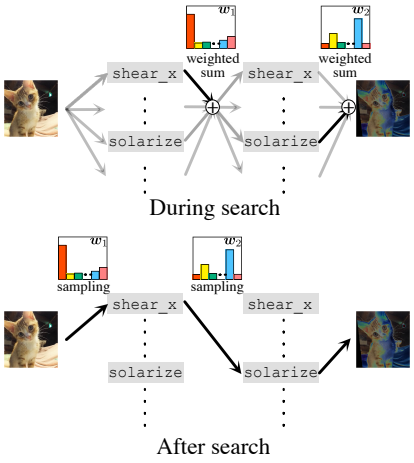

Fig. 4. Schematic view of the selection of operations in a single subpolicy when $K = 2$. During the search, we apply all operations to an image and take the weighted sum of the results as an augmented image. The weights $\boldsymbol{w}_1$ and $\boldsymbol{w}_2$ are also updated as other parameters. After the search, we sample operations according to the trained weights.

Selecting operations in subpolicies Each subpolicy S consists of K consecutive operations. To select the appropriate operation O_k where $k \in \{1, 2, \ldots, K\}$, we use a strategy similar to the one used in NAS [19] (see also Algorithm 1 and Figure 4 for details). To be specific, selecting the nth operation in $\mathcal{O}$ is equal to applying all operations $O_k^{(1)}, O_k^{(2)}, \ldots, O_k^{(\#\mathcal{O})}$ and selecting a result by multiplying a one-hot vector whose nth element is 1. We approximate this onehot vector by weighted sum of the outputs of all operations:

$$O_k(\boldsymbol{X}; \mu_k, p_k) \approx \sum_{m=1}^{\#\mathcal{O}} [\sigma_\eta(\boldsymbol{w}_k)]_m O_k^{(m)}(\boldsymbol{X}; \mu_k^{(m)}, p_k^{(m)}). \tag{6}$$

Here, $O_k^{(m)}$ is an operation in $\mathcal{O}$, and $O_k^{(m)}$ and $O_k^{(m')}$ are different operations if $m \neq m'$. $\boldsymbol{w}_k$ is a learnable parameter and σ_η is the softmax function $\sigma_\eta(\boldsymbol{z}) = \dfrac{\exp(\boldsymbol{z}/\eta)}{\sum_j \exp(z_i/\eta)}$ with a temperature parameter $\eta > 0$. At a low temperature η, $\sigma_\eta(\boldsymbol{w}_k)$ becomes a onehot-like vector. During inference, we sample the kth operation according to the categorical distribution $\mathrm{Cat}(\sigma_k(\boldsymbol{w}_k))$.

4.2 Data Augmentation as Density Matching

Using the techniques described above, we can backpropagate through the data augmentation process. In this section, we describe the objective of policy learning.

Algorithm 1 Selection of operations in a single subpolicy during a search. Refer to Figure 4 for the case of $K = 2$.

$\boldsymbol{X}$: input image, $\{\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K\}$: learnable weights,
σ_η: softmax function with temperature η
for k in $\{1, 2, \ldots, K\}$:
  Augment $\boldsymbol{X}$ by the kth stage operations: $\boldsymbol{X} \leftarrow \sum_{n=1}^{\#\mathcal{O}} [\sigma_\eta(\boldsymbol{w}_k)]_n O_k^{(n)}(\boldsymbol{X}; \mu_k^{(n)}, p_k^{(n)})$
return $\boldsymbol{X}$

One possible objective is the minimization of the validation loss as in DARTS [19]. Such formulation requires a nested optimization with an inner loop for parameter optimization and an outer loop for hyperparameter or architecture optimization. Unfortunately, this optimization takes a long time and requires a large memory footprint [7], which makes it impossible to apply it to training on large-scale datasets such as ImageNet. Moreover, data augmentation is not applied during the outer loop, i.e., validation, which differs from NAS that uses a searched architecture during the outer loop. Thus, we adopt a different of adversarial learning to avoid the nested loop.

Data augmentation can be seen as a process that fills missing data points in training data [18,24,31]. Therefore, we minimize the distance between distributions of the original and augmented images. This goal can be achieved by minimizing the Wasserstein distance between these distributions $d_{\boldsymbol{\theta}}$ using a stable Wasserstein GAN with a gradient penalty [2,10]. Here, $\boldsymbol{\theta}$ is the parameters of its critic. Unlike typical GANs for image modification, our model does not have a typical generator that learns to transform images using conventional neural network layers. Instead, a policy — explained in previous sections — is trained and transforms images using predefined operations. Following prior work [5,12,18], we use WideResNet-40-2 [35] (for CIFAR-10, CIFAR-100 and SVHN) or ResNet-50 [11] (for ImageNet) and add a two-layer perceptron that serves as a critic alongside the original classifier. The classification loss is used to prevent images of a certain class from being transformed into images of another class. Algorithm 2 depicts the detailed procedure. Importantly, we match different minibatches $\mathcal{B}$ and $\mathcal{B}'$ and randomly initialize $\boldsymbol{M}, \boldsymbol{P}, \boldsymbol{W}$, which are expected to prevent policies from being trapped into "no-op" solutions, as shown in Figure 5.

5 Experiments and Results

In this section, we show the empirical results of our approach on CIFAR-10, CIFAR-100 [16], SVHN [21] and ImageNet [27] datasets and compare the results with those of AutoAugment [5], PBA [12] and Fast AutoAugment [18]. Except for ImageNet, we run all experiments three times and report the average results. Details of the datasets are presented in Table 3.

Algorithm 2 Training of Faster AutoAugment

$\boldsymbol{M}, \boldsymbol{P}, \boldsymbol{W}$: learnable parameters of a subpolicy corresponding to $\boldsymbol{\mu}, \boldsymbol{p}, \boldsymbol{w}$, respectively
$d_{\boldsymbol{\theta}}(\cdot, \cdot)$: distance between two densities with learnable parameters $\boldsymbol{\theta}$, f: image classifier, $\mathcal{L}$: cross entropy loss, ϵ: coefficient of classification loss, $\mathcal{D}$: training set
while not converge :
  Sample a pair of batches $\mathcal{B}, \mathcal{B}'$ from $\mathcal{D}$
  Augment data $\mathcal{A} = \{S(\boldsymbol{X}; \boldsymbol{M}, \boldsymbol{P}, \boldsymbol{W}); (\boldsymbol{X}, \cdot) \in \mathcal{B}\}$
  Measure distance $d = d_{\boldsymbol{\theta}}(\mathcal{A}, \mathcal{B}')$
  Obtain classification loss $l = \mathbb{E}_{(\boldsymbol{X}, y) \sim \mathcal{A}} \mathcal{L}(f(\boldsymbol{X}), y) + \mathbb{E}_{(\boldsymbol{X}', y') \sim \mathcal{B}'} \mathcal{L}(f(\boldsymbol{X}'), y')$
  Update parameters $\boldsymbol{M}, \boldsymbol{P}, \boldsymbol{W}, \boldsymbol{\theta}$ to minimize $d + \epsilon l$ using SGD (e.g., Adam)

5.1 Experimental Details

Implementation Details Prior methods [5,12,18] employed Python's Pillow [4] as the image processing library. We transplant the operations described in Section 3.1 to PyTorch [23], a tensor computation library with automatic differentiation. For geometric operations, we extend functions in kornia [26]. We implement color-enhancing operations, sample pairing [13] and cutout [6] using PyTorch. Operations with discrete magnitude parameters are implemented as described in Section 4.1 with additional CUDA kernels.

We use CNN models and baseline preprocessing procedures available from Fast AutoAugment's repository [5] and follow their settings and hyperparameters for CNN training such as the initial learning rate and learning rate scheduling.

Experimental Settings To compare our results with those of previous studies [5,18,12], we follow their experimental settings on each dataset. We train the policy on randomly selected subsets of each dataset presented in Table 3. In the evaluation phase, we train CNN models from scratch on each dataset with learned Faster AutoAugment policies. For SVHN, we use both training and additional datasets.

Similar to Fast AutoAugment [18], our policies are composed of 10 subpolicies, each of which has operation count $K = 2$ as described in Section 3.2. We train the policies for 20 epochs using ResNet-50 for ImageNet and WideResNet-40-2 for other datasets. In all experiments, we set the temperature parameters λ and η to 0.05. We use Adam optimizer [15] with a learning rate of 1.0^{-3}, coefficients for running averages (betas) of $(0, 0.999)$, the coefficient for the classification loss ϵ of 0.1 and the coefficient for the gradient penalty of 10. Because GPUs are optimized for batched tensor computation, we apply subpolicies to chunks of images. The number of chunks determines the balance between speed and diversity. We set the chunk size to 16 for ImageNet and 8 for the other datasets during search. For evaluation, we use chunk sizes of 32 for ImageNet and 16 for other datasets.

[4] `https://python-pillow.org/`
[5] `https://github.com/kakaobrain/fast-autoaugment/tree/master/FastAutoAugment/networks`

Table 3. Summary of datasets used in the experiments. For the policy training on ImageNet, we use only 6,000 images from the 120 selected classes following [5,18].

Dataset	Training set size	Subset size for policy training
CIFAR-10 [16]	50,000	4,000
CIFAR-100 [16]	50,000	4,000
SVHN [21]	603,000	1,000
ImageNet [27]	1,200,000	6,000

Table 4. Faster AutoAugment yields performance comparable to prior methods. Test error rates on CIFAR-10, CIFAR-100 and SVHN. We report average rates over three runs. For CIFAR-100, we report results obtained with policies trained on CIFAR-10 / CIFAR-100.

Dataset	Model	Baseline	Cutout [6]	AA [5]	PBA [12]	Fast AA [18]	Faster AA (ours)
CIFAR-10	WideResNet-40-2 [35]	5.3	4.1	3.7	-	3.6	3.7
	WideResNet-28-10 [35]	3.9	3.1	2.6	2.6	2.7	2.6
	Shake-Shake (26 2×32d) [8]	3.6	3.0	2.5	2.5	2.7	2.7
	Shake-Shake (26 2×96d) [8]	2.9	2.6	1.9	2.0	2.0	2.0
	Shake-Shake (26 2×112d) [8]	2.8	2.6	1.9	2.0	2.0	2.0
CIFAR-100	WideResNet-40-2 [35]	26.0	25.2	20.7	-	20.7	22.1 / 21.4
	WideResNet-28-10 [35]	18.8	18.4	17.1	16.7	17.3	17.8 / 17.3
	Shake-Shake (26 2×96d) [8]	17.1	16.0	14.3	15.3	14.9	15.6 / 15.0
SVHN	WideResNet-28-10 [35]	1.5	1.3	1.1	1.2	1.1	1.2

5.2 Results

CIFAR-10 and CIFAR-100 In Table 4, we show test error rates on CIFAR-10 and CIFAR-100 with various CNN models: WideResNet-40-2, WideResNet-28-10 [35] and Shake-Shake (26 $2 \times \{32, 96, 112\}$d) [8]. We train WideResNets for 200 epochs and Shake-Shakes for 1,800 epochs as in [5], and report averaged values over three runs for Faster AutoAugment. The results of the baseline and Cutout are from [5,18]. Faster AutoAugment not only shows competitive results with prior methods, but is also significantly faster to train (see Table 1). For CIFAR-100, we report results with policies trained on reduced CIFAR-10 following [5] as well as policies trained on reduced CIFAR-100. The latter results are better than the former ones, which suggests the importance of training policy on the target dataset.

We also show several examples of augmented images in Figure 5. The policy seems often to use color-enhancing operations as reported in AutoAugment [5].

In Table 5[6], we report error rates on reduced CIFAR-10 to show the effect of Faster AutoAugment in the low-resource scenario. In this experiment, we randomly sample 4,000 images from the training dataset. We train the policy using the subset and evaluate the policy with WideResNet-28-10 on the same subset for 200 epochs. As can be seen, Faster AutoAugment improves the performance 7.7% over Cutout and achieves a similar error rate to AutoAugment. This result implies that data augmentation can moderately reduce the difficulty in learning from small data.

[6] [5] reported better baseline and Cutout performance than us (18.8% and 16.5% respectively), but we could not reproduce the results in [5]

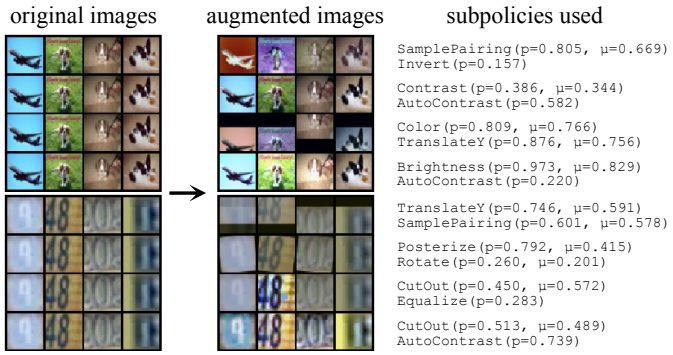

Fig. 5. Original and augmented images of CIFAR-10 (upper) and SVHN (lower). As can been seen, Faster AutoAugment can transform original images into diverse augmented images with subpolicies as shown on the right-hand side.

Table 5. Test error rates with models trained on reduced CIFAR-10, which consists of 4,000 images randomly sampled from the training set. We show that the policy obtained by Faster AutoAugment is useful for a low-resource scenario.

Baseline	Cutout [6]	AA [5]	Faster AA (ours)
24.3	22.5	14.1	14.8

SVHN In Table 4, we show test error rates on SVHN with WideResNet-28-10 trained for 200 epochs. For Faster AutoAugment, we report the average value of three runs. Faster AutoAugment achieves an error rate of 1.2%, which is a 0.1% improvement over Cutout and on a par with PBA. The augmented images are shown in Figure 5. We also show the augmented images in Figure 5 with the obtained subpolicies, which seem to select more geometric transformations than CIFAR-10's policy as reported in [5].

ImageNet In Table 6, we compare the top-1 and top-5 validation error rates on ImageNet with those of [5,18]. To align our results with [5], we also train ResNet-50 for 200 epochs. [5,18] reported top-1/top-5 error rates of 23.1%/6.5%. Faster AutoAugment achieves a 0.6% improvement over the baseline for top-1 error rate. This gain verifies that Faster AutoAugment has an effect comparable to prior methods on a large and complex dataset. The performance is slightly worse than AutoAugment and Fast AutoAugment, which may be attributed to the limited number of different subpolicies for each minibatch because of the number of chunks, which we describe in Section 5.1.

6 Analysis

Changing the Number of subpolicies The number of subpolicies L is arbitrary. Figure 6 shows the relationship between the number of subpolicies and

Table 6. Top-1/Top-5 validation error rates on ImageNet [27] with ResNet-50 [11]. Faster AutoAugment achieves comparable performance to AA and Fast AA.

Baseline	AA [5]	Fast AA [18]	Faster AA (ours)
23.7/6.9	22.4/6.2	22.4/6.3	23.1/6.5

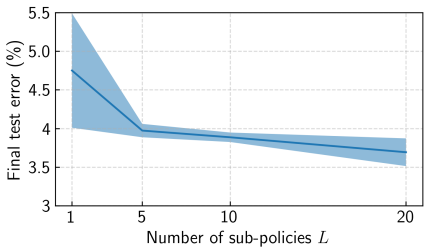

Fig. 6. The performance increases with the number of sub-policies. Relationship between the number of subpolicies and the test error rate (CIFAR-10 with WideResNet-40-2). We plot test error rates and their standard deviations averaged over three runs.

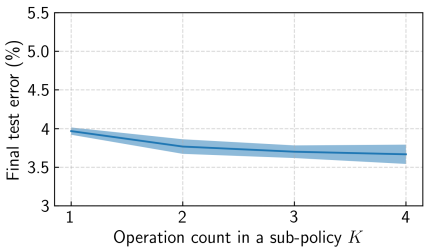

Fig. 7. The performance increases with the operation count. Relationship between the operation count of each subpolicy and the average test error rate of three runs (CIFAR-10 with WideResNet-40-2).

the final test error on CIFAR-10 dataset with WideResNet-40-2. As can be seen, the more subpolicies we have, the lower the error rate is. This phenomenon is straightforward because the number of subpolicies determines the diversity of augmented images. Importantly, an increase in the number of subpolicies results in the exponential growth of the search space, which is prohibitive for standard search methods.

Changing the Operation Count The operation count K of each subpolicy is also arbitrary. Similarly to the number of subpolicies L, increasing the operation count of a subpolicy K also exponentially increases the search space. We change K from 1 to 4 on CIFAR-10 dataset with WideResNet-40-2. We present the resulting error rates in Figure 7. As can be seen, as the operation count in each subpolicy grows, the performance increases, i.e., the error rate decreases. These results show that Faster AutoAugment is scalable to a large search space.

Changing the Data Size In the main experiments in Section 5, we used a subset of CIFAR-10 of 4,000 images for policy training. To validate the effect of this sampling, we train a policy on the full CIFAR-10 of 50,000 images as in [18] and evaluate the obtained policy with WideResNet-40-2. We find that the increase of data size causes a significant performance drop (from 3.7% to 4.1%) with subpolicies of $L = 10$. We hypothesize that this drop is because of lower capability of the policy when $L = 10$. Therefore, we train a policy with

Table 7. Test error rates on CIFAR-10 using policies trained on reduced CIFAR-10 (4,000 images) and full CIFAR-10 (50,000 images) with WideResNet-40-2.

Data size	Fast AA [18]	Faster AA (ours)
4,000	3.6	3.7
50,000	3.7	3.8

$L = 80$ subpolicies and randomly sample 10 subpolicies to evaluate the policy, which results in comparable error rates (3.8% and 3.7%). We present the results in comparison with those of Fast AutoAugment [18] in Table 7, which shows the effectiveness of using subsets for Fast AutoAugment and Faster AutoAugment.

Effect of Policy Training To confirm that trained policies are more effective than randomly initialized policies, we compare test error rates on CIFAR-10 with and without policy training, as performed in AutoAugment [5]. Using WideResNet-28-10, trained policies achieve an error rate of 2.6%, while randomly initialized policies have a slightly higher error rate of 2.7% (both error rates are an average of three runs). These results imply that data augmentation policy searching is a meaningful research direction, but still has much room to improve.

7 Conclusion

In this paper, we proposed Faster AutoAugment, which achieves faster policy searching for data augmentation than previous methods [5,12,18]. To achieve this, we introduced gradient approximation for several non-differentiable image operations and made the policy search process end-to-end differentiable. We verified our method on several standard benchmarks and showed that Faster AutoAugment could achieve competitive performance with other methods for automatic data augmentation. Moreover, our additional experiments suggest that gradient-based policy optimization can be scaled to more complex scenarios.

We believe that faster policy searching will be beneficial for research on representation learning such as semi-supervised learning [4,33] and domain generalization [32]. Additionally, learning from limited data using learnable policies might be an interesting future direction.

Acknowledgement

The research results were achieved as a part of the “Research and Development of Deep Learning Technology for Advanced Multilingual Speech Translation”, the Commissioned Research of the National Institute of Information and Communications Technology, JAPAN. This work was also supported by JSPS KAKENHI Grant Numbers JP19H04166, JP19K22861 and JP20H04251. We used the RAIDEN system for the experiments.

References

1. Antoniou, A., Storkey, A., Edwards, H.: Data Augmentation Generative Adversarial Networks. In: ICLR (2018)
2. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN. In: ICML (2017)
3. Bengio, Y., Léonard, N., Courville, A.: Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. arXiv (2013)
4. Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., Raffel, C.: MixMatch: A Holistic Approach to Semi-Supervised Learning. In: NeurIPS (2019)
5. Cubuk, E.D., Zoph, B., Mane, D., Vasudevan, V., Le, Q.V.: AutoAugment: Learning Augmentation Policies from Data. In: CVPR (2019)
6. DeVries, T., Taylor, G.W.: Improved Regularization of Convolutional Neural Networks with Cutout. arXiv (2017)
7. Finn, C., Abbeel, P., Levine, S.: Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In: ICML (2017)
8. Gastaldi, X.: Shake-Shake regularization of 3-branch residual networks. In: ICLR (2017)
9. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, J.: Generative Adversarial Networks. In: NIPS (2014)
10. Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., Courville, A.: Improved Training of Wasserstein GANs. In: NIPS (2017)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep Residual Learning for Image Recognition. In: CVPR (2016)
12. Ho, D., Liang, E., Stoica, I., Abbeel, P., Chen, X.: Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules. In: ICML (2019)
13. Inoue, H.: Data Augmentation by Pairing Samples for Images Classification. arXiv (2018)
14. Jang, E., Gu, S., Poole, B.: Categorical Reparameterization with Gumbel-Softmax. In: ICLR (2017)
15. Kingma, D.P., Ba, J.L.: Adam: a Method for Stochastic Optimization. In: ICLR (2015)
16. Krizhevsky, A.: Learning Multiple Layers of Features from Tiny Images. Tech. rep. (2009)
17. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet Classification with Deep Convolutional Neural Networks. In: NIPS (2012)
18. Lim, S., Kim, I., Kim, T., Kim, C., Kim, S.: Fast AutoAugment. In: NeurIPS (2019)
19. Liu, H., Simonyan, K., Yang, Y.: DARTS: Differentiable Architecture Search. In: ICLR (2018)
20. Maclaurin, D., Duvenaud, D., Adams, R.: Gradient-based hyperparameter optimization through reversible learning. In: Bach, F., Blei, D. (eds.) ICML (2015)
21. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading Digits in Natural Images with Unsupervised Feature Learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning (2011)
22. van den Oord, A., Vinyals, O., Kavukcuoglu, K.: Neural Discrete Representation Learning. In: NIPS (2017)
23. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: Pytorch: An imperative style, high-performance deep learning library. In: NeurIPS (2019)

24. Ratner, A.J., Ehrenberg, H.R., Hussain, Z., Dunnmon, J., Ré, C.: Learning to compose domain-specific transformations for data augmentation. In: NIPS (2017)
25. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized Evolution for Image Classifier Architecture Search. In: AAAI (2019)
26. Riba, E., Mishkin, D., Ponsa, D., Rublee, E., Bradski, G.: Kornia: an Open Source Differentiable Computer Vision Library for PyTorch. In: WACV (2019)
27. 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. IJCV (2015)
28. Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R.: Learning from simulated and unsupervised images through adversarial training. In: CVPR (2017)
29. Sixt, L., Wild, B., Landgraf, T.: RenderGAN: Generating realistic labeled data. In: Frontiers Robotics AI (2018)
30. Tokozume, Y., Ushiku, Y., Harada, T.: Between-class Learning for Image Classification. In: CVPR (2018)
31. Tran, T., Pham, T., Carneiro, G., Palmer, L., Reid, I.: A Bayesian Data Augmentation Approach for Learning Deep Models. In: NIPS (2017)
32. Volpi, R., Murino, V.: Addressing Model Vulnerability to Distributional Shifts over Image Transformation Sets. In: ICCV (2019)
33. Xie, Q., Dai, Z., Hovy, E., Luong, M.T., Le, Q.V.: Unsupervised Data Augmentation. arXiv (2019)
34. 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)
35. Zagoruyko, S., Komodakis, N.: Wide Residual Networks. In: BMVC (2016)
36. Zhang, H., Cisse, M., Dauphin, Y.N., Lopez-Paz, D.: mixup: Beyond Empirical Risk Minimization. In: ICLR (2018)
37. Zhong, Z., Zheng, L., Kang, G., Li, S., Yang, Y.: Random erasing data augmentation. arXiv (2017)
38. Zoph, B., Le, Q.V.: Neural Architecture Search with Reinforcement Learning. In: ICLR (2017)