

Online Hyper-parameter Learning for Auto-Augmentation Strategy

Chen Lin^{1*}, Minghao Guo^{1*}, Chuming Li¹, Xin Yuan¹, Wei Wu¹, Junjie Yan¹, Dahua Lin², Wanli Ouyang³

¹SenseTime Group Limited

²The Chinese University of Hong Kong

³The University of Sydney, SenseTime Computer Vision Research Group, Australia

{linchen, guominghao, lichuming, yuanxin, wuwei, yanjunjie}@sensetime.com; dhlin@ie.cuhk.edu.hk; wanli.ouyang@sydney.edu.au

Abstract

Data augmentation is critical to the success of modern deep learning techniques. In this paper, we propose Online Hyper-parameter Learning for Auto-Augmentation (OHL-Auto-Aug), an economical solution that learns the augmentation policy distribution along with network training. Unlike previous methods on auto-augmentation that search augmentation strategies in an offline manner, our method formulates the augmentation policy as a parameterized probability distribution, thus allowing its parameters to be optimized jointly with network parameters. Our proposed OHL-Auto-Aug eliminates the need of re-training and dramatically reduces the cost of the overall search process, while establishes significant accuracy improvements over baseline models. On both CIFAR-10 and ImageNet, our method achieves remarkable search accuracy, i.e. $60\times$ faster on CIFAR-10 and $24\times$ faster on ImageNet, while maintaining competitive accuracies.

1. Introduction

Recent years have seen remarkable success of deep neural networks in various computer vision applications. Driven by large amounts of labeled data, the performances of networks have reached an amazing level. A common issue in learning deep networks is that the training process is prone to overfitting, with the sharp contrast between the huge model capacity and the limited dataset. Data augmentation, which applies randomized modifications to the training samples, have been shown to be an effective way to mitigate this problem. Simple image transforms, including random crop, horizontal flipping, rotation and translation, are

*equal contribution

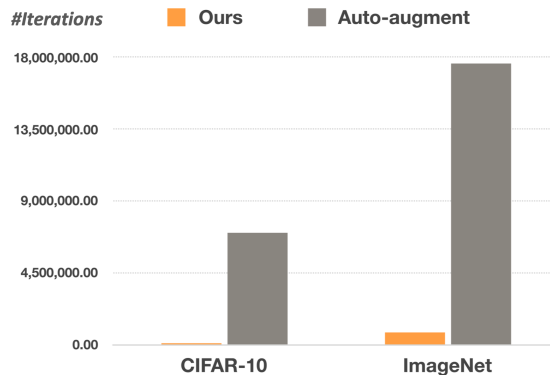


Figure 1. The search cost of Auto-Augment [6] and our OHL-Auto-Aug. ‘#Iterations’ denotes the total training iterations with conversion under a same batch size 1024 (more details in Section 4.2). Our proposed OHL-Auto-Aug achieves a remarkable auto-augmentation search efficiency, which is $60\times$ faster on CIFAR-10 and $24\times$ faster on ImageNet than Auto-Augment [6].

utilized to create label-preserved new training data to promote the network generalization and accuracy performance. Recent efforts such as [8, 34] further develop the transform strategy and boost the model to state-of-the-art accuracy on CIFAR-10 and ImageNet datasets.

However, it is nontrivial to find a good augmentation policy for a given task, as it varies significantly across datasets and tasks. It often requires extensive experience combined with tremendous efforts to devise a reasonable policy. Instinctively, automatically finding the process of data augmentation strategy for a target dataset turns into an alternative. Cubuk *et al.* [6] described a procedure to automatically search augmentation strategy from data. The search process involves sampling hundreds and thousands of poli-

cies, each trained on a child model to measure the performance and further update the augmentation policy distribution represented by a controller. Despite its promising empirical performance, the whole process is computationally expensive and time consuming. Particularly, it takes 15000 policy samples, each trained with 120 epochs. This immense amount of computing limits its applicability in practical environment. Thus [6] subsampled the dataset to 8% and 0.5% for CIFAR-10 and ImageNet respectively. An inherent cause of this computational bottleneck is the fact that augmentation strategy is searched in an offline manner.

In this paper, we present a new approach that considers auto-augmentation as a hyper-parameter optimization problem and drastically improves the efficiency of the autoaugmentation strategy. Specifically, we formulate augmentation policy as a probability distribution. The parameters of the distribution are regarded as hyper-parameters. We further propose a bilevel framework which allows the distribution parameters to be optimized along with network training. In this bilevel setting, the inner objective is the minimization of vanilla train loss w.r.t network parameters, while the outer objective is the maximization of validation accuracy w.r.t augmentation policy distribution parameters utilizing REINFORCE [30]. These two objectives are optimized simultaneously, so that the augmentation distribution parameters are tuned alongside the network parameters in an online manner. As the whole process eliminates the need of retraining, the computational cost is exceedingly reduced. Our proposed OHL-Auto-Aug remarkably improves the search efficiency while achieving significant accuracy improvements over baseline models. This ensures our method can be easily performed on large scale dataset including CIFAR-10 and ImageNet without any data reduction.

Our main contribution can be summarized as follows: **1)** We propose an online hyper-parameter learning approach for auto-augmentation strategy which treats each augmentation policy as a parameterized probability distribution. **2)** We introduce a bilevel framework to train the distribution parameters along with the network parameters, thus eliminating the need of re-training during search. **3)** Our proposed OHL-Auto-Aug dramatically improves the efficiency while achieving significant performance improvements. Compared to the previous state-of-the-art auto-augmentation approach [6], our OHL-Auto-Aug achieves $60\times$ faster on CIFAR-10 and $24\times$ faster on ImageNet with comparable accuracies.

2. Related Work

2.1. Data Augmentation

Data augmentation lies at the heart of all successful applications of deep neural networks. In order to

improve the network generalization, substantial domain knowledge is leveraged to design suitable data transformations. [16] applied various affine transforms, including horizontal and vertical translation, squeezing, scaling, and horizontal shearing to improve their model’s accuracy. [15] exploited principal component analysis on ImageNet to randomly adjust colour and intensity values. [31] further utilized a wide range of colour casting, vignetting, rotation, and lens distortion to train Deep Image on ImageNet.

There also exists attempts to learn data augmentations strategy, including Smart Augmentation [17], which proposed a network that automatically generates augmented data by merging two or more samples from the same class, and [29] which used a Bayesian approach to generate data based on the distribution learned from the training set. Generative adversarial networks have also been used for the purpose of generating additional data [22, 35, 1]. The work most closely related to our proposed method is [6], which formulated the auto-augmentation search as a discrete search problem and exploited a reinforcement learning framework to search the policy consisting of possible augmentation operations. Our proposed method optimizes the distribution on the discrete augmentation operations but is much more computationally economical benefiting from the proposed hyper-parameter optimization formulation.

2.2. Hyper-parameter Optimization

Hyper-parameter optimization was historically accomplished by selecting several values for each hyper-parameter, computing the Cartesian product of all these values, then running a full training for each set of values. [2] showed that performing a random search was more efficient than a grid search by avoiding excessive training with the hyper-parameters that were set to a poor value. [3] later refined this random search process by using quasi random search. [27] further utilized Gaussian processes to model the validation error as a function of the hyper-parameters. Each training further refines this function to minimize the number of sets of hyper-parameters to try. All these methods are “black-box” methods since they assume no knowledge about the internal training procedure to optimize hyper-parameters. Specifically, they do not have access to the gradient of the validation loss w.r.t. the hyper-parameters. To address this issue, [21, 9] explicitly utilized the parameter learning process to obtain such a gradient by proposing a Lagrangian formulation associated with the parameter optimization dynamics. [21] used a reverse-mode differentiation approach, where the dynamics corresponds to stochastic gradient descent with momentum. The theoretical evidence of the hyper-parameter optimization in our auto-augmentation is [9], where the hyper-parameters are changed after a certain number of parameter updates in a forward manner. The forward-mode procedure is suitable

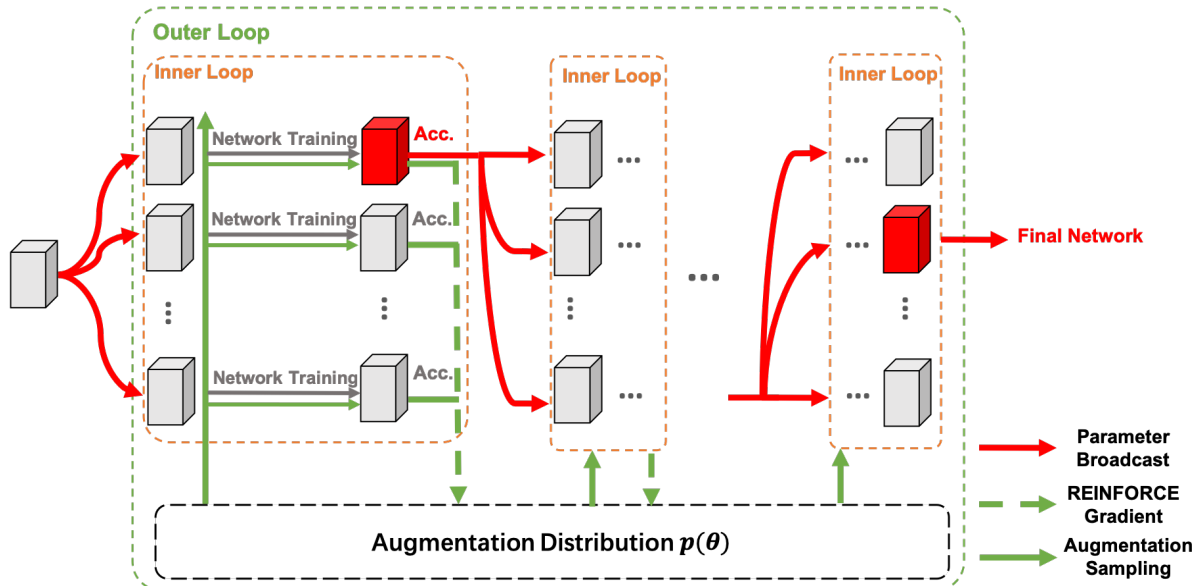


Figure 2. The framework of our OHL-Auto-Aug. We formulate each augmentation policy as a parameterized probability distribution, whose parameters are regarded as hyper-parameters. We propose a bilevel framework which allows the distribution parameters to be optimized simultaneously with the network parameters. In the inner loop, the network parameters are trained using standard SGD with augmentation sampling. In the outer loop, augmentation policy distribution parameters are trained using REINFORCE gradients with trajectory samples. At each time step, the network parameters with the highest accuracy are broadcasted to other trajectory samples.

for the auto-augmentation search process with a drastically reduced cost.

2.3. Auto Machine Learning and Neural Architecture Search

Auto Machine Learning (AutoML) aims to free human practitioners and researchers from these menial tasks. Recently many advances focus on automatically searching neural network architectures. One of the first attempts [37, 36, 38] was utilizing reinforcement learning to train a controller that represents a policy to generate a sequence of symbols representing the network architecture. The generation of a neural architecture was formulated as the controller’s action, whose space is identical to the architecture search space. An alternative to reinforcement learning is evolutionary algorithms, that evolved the topology of architectures by mutating the best architectures found so far [24, 32, 25]. There also exists surrogate model based search methods [18] that utilized sequential model-based optimization as a technique for parameter optimization. However, all the above methods require massive computation during the search, particularly thousands of GPU days. Recent efforts such as [19, 20, 23], utilized several techniques trying to reduce the search cost. [19] introduced a real-valued architecture parameter which was jointly trained with weight parameters. [20] embedded architectures into a latent space and performed optimization before decoding. [23] utilized architectural sharing among sampled models instead

of training each of them individually. Several methods attempt to automatically search architectures with fast inference speed, either explicitly took the inference latency as a constrain [28], or implicitly encoded the topology information [11]. Note that [6] utilized a similar controller inspired by [37], whose training is time consuming, to guide the augmentation policy search. Our auto-augmentation strategy is much more efficient and economical compared to these methods.

3. Approach

In this section, we first present the formulation of hyper-parameter optimization for auto-augmentation strategy. Then we introduce the framework of solving the optimization in an online manner. Finally we detail the search space and training pipeline. The pipeline of our OHL-Auto-Aug is shown in Figure 2.

3.1. Problem Formulation

The purpose of auto-augmentation strategy is to automatically find a set of augmentation operations performed on training data to improve the model generalization according to the dataset. In this paper, we formulate the augmentation strategy as p_θ , a probability distribution on augmentation operations. Suppose we have K candidate data augmentation operations $\{O_k(\cdot)\}_{k=1:K}$, each of which is selected under a probability of $p_\theta(O_k)$. Given a network model $\mathcal{F}(\cdot, w)$ parameterized by w , train dataset

$\mathcal{X}_{tr} = \{(x_i, y_i)\}_{i=1}^{N_{tr}}$, and validation dataset $\mathcal{X}_{val} = \{(\hat{x}_i, \hat{y}_i)\}_{i=1}^{N_{val}}$, the purpose of data augmentation is to maximum the validation accuracy with respect to θ , and the weights w associated with the model are obtained by minimizing the training loss:

$$\max_{\theta} \mathcal{J}(\theta) = acc(w^*) \quad (1)$$

$$\text{s.t. } w^* = \arg \min_w \frac{1}{N_{tr}} \sum_{(x,y) \in \mathcal{X}_{tr}} \mathbb{E}_{p_{\theta}(O)} [\mathcal{L}(\mathcal{F}(O(x), w), y)],$$

where $acc(w^*) = \frac{1}{N_{tr}} \sum_{(\hat{x}, \hat{y}) \in \mathcal{X}_{val}} \delta(\mathcal{F}(\hat{x}, w^*), \hat{y})$, \mathcal{L} denotes the loss function and the input training data is augmented by the selected candidate operations.

This refers to a bilevel optimization problem [5], where augmentation distribution parameters θ are regarded as hyper-parameters. At the outer level we look for a augmentation distribution parameter θ , under which we can obtain the best performed model $\mathcal{F}(\cdot, w^*)$, where w^* is the solution of the inner level problem. Previous state-of-the-art methods resort to sampling augmentation strategies with a surrogate model, and then solve the inner optimization problem exactly for each sampled strategy, which raises the time-consuming issue. To tackle this problem, we propose to train the augmentation distribution parameters alongside with the training of network weights, getting rid of training thousands of networks from scratch.

3.2. Online Optimization framework

Inspired by the forward hyper-gradient proposed in [9], we propose an online optimization framework that trains the hyper-parameters and the network weights simultaneously. To clarify, we use $T = 1, 2, \dots, T_{max}$ to denote the update steps of the outer loop, and $i = 1, 2, \dots, I$ to denote the update iterations of the inner loop. This indicates that between two adjacent outer optimization updates, the inner loop updates for I steps. Each steps train the network weights with a batch size of B images. In this section, we focus on the process of the T -th period of the outer loop.

For each image, a augmentation operation is sampled according to the current augmentation policy distribution $p_{\theta_{T-1}}$. We use trajectory $\mathcal{T} = \{O_{k(j)}\}_{j=1:(I \times B)}$ to denote all the augmentation operations sampled in the T -th period. For the I iterations of the inner loop, we have

$$\begin{aligned} w_{T-1}^{(i)} &= w_{T-1}^{(i-1)} - \eta_w \nabla_w [\mathcal{L}_B(\mathcal{T}^{(i)}, w_{T-1}^{(i-1)}, x_B, y_B)], \\ w_{T-1}^{(0)} &= w_{T-1}, w_T = w_{T-1}^{(I)}, \end{aligned} \quad (2)$$

where η_w is learning rate for the network parameters, \mathcal{L}_B denotes the batched loss which takes three inputs: the augmentation operations $\mathcal{T}^{(i)}$ for the i -th batch, the current inner loop parameters $w_{T-1}^{(i-1)}$ and a mini-batched data x_B, y_B , and returns an average loss on the i -th batch.

As illustrated in Equation 2, w_T is affect by the trajectory \mathcal{T} . Since w_T is only influenced by the operations in trajectory \mathcal{T} , w_T should be regarded as a random variable, with probability $p(w_T) = p(\mathcal{T})$, where $p(w_T)$ denotes the probability of \mathcal{T} sampled by $p_{\theta_{T-1}}$, calculated as $p(\mathcal{T}) = \prod_{j=1}^{I \times B} p_{\theta_{T-1}}(O_{k(j)})$. We update the augmentation distribution parameters θ_T by taking one step of optimization for the outer objective,

$$\begin{aligned} \theta_T &= \theta_{T-1} + \eta_{\theta} \nabla_{\theta} \mathcal{J}(\theta_{T-1}) \\ &= \theta_{T-1} + \eta_{\theta} \nabla_{\theta} \mathbb{E}_{w_T} [acc(w_T)], \end{aligned} \quad (3)$$

where η_{θ} is learning rate for the augmentation distribution parameters. As the outer objective is a maximization, here the update is ‘gradient ascent’ instead of ‘gradient descent’. The above process iteratively continues until the network model is converged.

The motivation of this online framework is that we approximate $w^*(\theta)$ using only I steps of training, instead of completely solving the inner optimization in Equation 1 until convergence. At the outer level, we would like to find the augmentation distribution that can train a network to a high validation accuracy given the I -step optimized network weights. A similar approach has been exploited in meta-learning for neural architecture search [19]. Although the convergence are not guaranteed, practically the optimization is able to reach a fixed point as our experiments have demonstrated, as well in [19].

It is a tricky problem to calculate the gradient of validation accuracy with respect to θ in Equation 3. This is mainly due to two facts: 1. the validation accuracy is non-differentiable with respect to θ . 2. Analytically calculating the integral over w_T is intractable. To address these two issues, we utilize REINFORCE [30] to approximate the gradient in Equation 3 by Monte-Carlo sampling. This is achieved by training N networks in parallel for the inner loop, which are treated as N sampled trajectories. We calculate the average gradient of them based on the REINFORCE algorithm. We have

$$\begin{aligned} \nabla_{\theta} \mathcal{J}(\theta) &= \nabla_{\theta} \mathbb{E}_{w_T} [acc(w_T)] \approx \frac{1}{N} \sum_{n=1}^N \nabla_{\theta} \log(p(w_{T,n})) acc(w_{T,n}) \\ &= \frac{1}{N} \sum_{n=1}^N \nabla_{\theta} \log(p(\mathcal{T}_n)) acc(w_{T,n}), \end{aligned} \quad (4)$$

where \mathcal{T}_n is the n -th trajectory. By substituting $p(\mathcal{T}_n)$ to Equation 4,

$$\begin{aligned} \nabla_{\theta} \mathcal{J}(\theta) &\approx \frac{1}{N} \sum_{n=1}^N \nabla_{\theta} \log(p(\mathcal{T}_n)) acc(w_{T,n}) \\ &= \frac{1}{N} \sum_{n=1}^N \sum_{j=1}^{I \times B} \nabla_{\theta} \log(p_{\theta_{T-1}}(O_{k(j),n})) acc(w_{T,n}). \end{aligned} \quad (5)$$

Table 1. List of Candidate Augmentation Elements

Elements Name	range of magnitude
<i>HorizontalShear</i>	{0.1, 0.2, 0.3}
<i>VerticalShear</i>	{0.1, 0.2, 0.3}
<i>HorizontalTranslate</i>	{0.15, 0.3, 0.45}
<i>VerticalTranslate</i>	{0.15, 0.3, 0.45}
<i>Rotate</i>	{10, 20, 30}
<i>ColorAdjust</i>	{0.3, 0.6, 0.9}
<i>Posterize</i>	{4.4, 5.6, 6.8}
<i>Solarize</i>	{26, 102, 179}
<i>Contrast</i>	{1.3, 1.6, 1.9}
<i>Sharpness</i>	{1.3, 1.6, 1.9}
<i>Brightness</i>	{1.3, 1.6, 1.9}
<i>AutoContrast</i>	None
<i>Equalize</i>	None
<i>Invert</i>	None

In practice, we also utilize the baseline trick [26] to reduce the variance of the gradient estimation,

$$\nabla_{\theta} \mathcal{J}(\theta) \approx \frac{1}{N} \sum_{n=1}^N \sum_{j=1}^{I \times B} \nabla_{\theta} \log(p_{\theta_{T-1}}(O_{k(j),n})) \tilde{acc}(w_{T,n}), \quad (6)$$

where $\tilde{acc}(w_{T,n})$ denotes the function that normalizes the accuracy returned by each sample trajectory to zero mean.¹

Since we have N trajectories, each of which outputs a $w_{T,n}$ from Equation 2, we need to synchronize these $w_{T,n}$ to get a same start point for the next training step. We simply select the $w_{T,n}$ with the best validation accuracy as the final output of Equation 2.

3.3. Search Space and Training Pipeline

In this paper, we formulate the auto-augmentation strategy as distribution optimization. For each input train image, we sample an augmentation operation from the search space and apply to it. Each augmentation operation consists of two augmentation elements. The candidate augmentation elements are listed in Table 1.

The total number of the elements is 36. We will sample augmentation choices twice, The augmentation distribution is a multinomial distribution which has K possible outcomes. The probability of the k th operation is a normalized sigmoid function of θ_k . A single operation is defined as the combination of two elements, resulting in $K = 36^2$ possible combinations (the same augmentation choice can repeat). The augmentation distribution is a multinomial distribution which has K possible outcomes. The probability

¹To meet $p(w_T) = p(\mathcal{T})$, we use the same sampled mini-batches for all trajectories, so there is no randomness caused by mini-batch sampling.

Algorithm 1 Online Optimization for Auto-Augmentation Strategy

Initialize θ_0 , initialize the same w_0 for N models;
while $T \leq T_{max}$ **do**
 for all n such that $1 \leq n \leq N$ **do**
 for all i such that $0 \leq i \leq I$ **do**
 Compute $\{w_{T,n}^i\}$ in Equation 2;
 end for
 return $w_{T,n}$
 end for
 return $\{w_{T,n}\}_{n=1:N}$
 Fix $\{w_{T,n}\}_{n=1:N}$, calculate $\nabla_{\theta} \mathcal{J}(\theta)$ in Equation 6;
 Update θ_T according to Equation 3;
 Select w_T from $\{w_{T,n}\}_{n=1:N}$ with the best validation accuracy;
 Broadcast w_T to all the N models;
end while
return $w_{T_{max}}, \theta_{T_{max}}$;

of the k th operation is a normalized sigmoid function of θ_k ,

$$p_{\theta}(O_k) = \frac{\left(\frac{1}{1+e^{-\theta_k}}\right)}{\sum_{i=1}^K \left(\frac{1}{1+e^{-\theta_i}}\right)}, \quad (7)$$

where $\theta \in \mathbb{R}^K$.

The whole training pipeline is described in Algorithm 1.

4. Experiments

4.1. Implementation Details

We perform our OHL-Auto-Aug on two classification datasets: CIFAR-10 and ImageNet. Here we describe the experimental details that we used to search the augmentation policies.

CIFAR-10 CIFAR-10 dataset [14] consists of natural images with resolution 32×32 . There are totally 60,000 images in 10 classes, with 6000 images per class. The train and test sets contain 50,000 and 10,000 images respectively. For our OHL-Auto-Aug on CIFAR-10, we use a validation set of 5,000 images, which is randomly splitted from the training set, to calculate the validation accuracy during the training for the augmentation distribution parameters.

In the training phase, the basic pre-processing follows the convention for state-of-the-art CIFAR-10 models: standardizing the data, random horizontal flips with 50% probability, zero-padding and random crops, and finally Cutout [8] with 16×16 pixels. Our OHL-Auto-Aug strategy is applied in addition to the basic pre-processing. For each training input data, we first perform basic pre-processing, then our OHL-Auto-Aug strategy, and finally Cutout.

All the networks are trained from scartch on CIFAR-10. For the training of network parameters, we use a mini-batch size of 256 and a standard SGD optimizer. The momentum rate is set to 0.9 and weight decay is set to 0.0005. The

Table 2. Test error rates (%) on CIFAR-10. The number in brackets refers to the results of our implementation. We compare our OHL-Auto-Aug with standard augmentation (Baseline), standard augmentation with Cutout (Cutout), augmentation strategy discovered by [6] (Auto-Augment). Compared to Baseline, our OHL-Auto-Aug achieves about 30% reduction of error rate.

Model	Baseline	Cutout [8]	Auto-Augment [6]	OHL-Auto-Aug	Error Reduce (Baseline/Cutout)
ResNet-18 [12]	4.66	3.62	3.46	3.29	1.37/0.33
WideResNet-28-10 [33]	3.87	3.08	2.68	2.61	1.26/0.47
DualPathNet-92 [4]	4.55	3.71	3.16	2.75	1.8/0.96
AmoebaNet-B(6, 128) [6] (our impl.)	2.98 (3.4)	2.13 (2.9)	1.75	1.89	1.09/0.24 (1.51/1.01)

cosine learning rate scheme is utilized with the initial learning rate of 0.2. The total training epochs is set to 300. For the AmoebaNet-B [24], there are several hyper-parameters modifications. The initial learning rate is set to 0.024. We also use an additional learning rate warmup stage [10] before training. The learning rate is linearly increased from 0.005 to the initial learning rate 0.024 in 40 epochs.

ImageNet ImageNet dataset [7] contains 1.28 million training images and 50,000 validation images from 1000 classes. For our experiments on CIFAR-10, we set aside an additional validation set of 50,000 images splitted from the training dataset.

For basic data pre-processing, we follow the standard practice and perform the random size crop to 224×224 and random horizontal flipping. The practical mean channel subtraction is adopted to normalize the input images for both training and testing. Our OHL-Auto-Aug operations are performed following this basic data pre-processing.

For the training on ImageNet, we use synchronous SGD with a Nesterov momentum of 0.9 and $1e-4$ weight decay. The mini-batch size is set to 2048 and the base learning rate is set to 0.8. The cosine learning rate scheme is utilized with a warmup stage of 2 epochs. The total training epoch is 150.

Augmentation Distribution Parameters For the training of augmentation policy parameters, we use Adam optimizer with a learning rate of $\eta_\theta = 0.05$, $\beta_1 = 0.5$ and $\beta_2 = 0.999$. On CIFAR-10 dataset, the number of trajectory sample is set to 8. This number is reduced to 4 for ImageNet, due to the large computation cost. Having finished the outer update for the distribution parameters, we broadcast the network parameters with the best validation accuracy to other sampled trajectories using multiprocessing utensils.

4.2. Results

CIFAR-10 Results: On CIFAR-10, we perform our OHL-Auto-Aug on three popular network architectures: ResNet [12], WideResNet [33], DualPathNet [4] and AmoebaNet-B [24]. For ResNet, a 18-layer ResNet with 11.17M parameters is used. For WideResNet, we use a 28 layer WideResNet with a widening factor of 10 which has

36.48M parameters. For Dual Path Network, we choose DualPathNet-92 with 34.23M parameters for CIFAR-10 classification. AmoebaNet-B is an auto-searched architecture by a regularized evolution approach proposed in [24]. We use the AmoebaNet-B (6,128) setting for fair comparison with [6], whose number of parameters is 33.4M. The WideResNet-28, DualPathNet-92 and AmoebaNet-B are relatively heavy architectures for CIFAR-10 classification.

We illustrate the test set error rates of our OHL-Auto-Aug for different network architectures in Table 2. All the experiments follow the same setting described in Section 4.1. We compare the empirical results with standard augmentation (Baseline), standard augmentation with Cutout (Cutout), an augmentation strategy proposed in [6] (Auto-Augment). For ResNet-18 and DualPathNet-92, our OHL-Auto-Aug achieves a significant improvement than Cutout and Auto-Augment. Compare to the Baseline model, the error rate is reduced about 30% ResNet-18, about 40% for DualPathNet-92 and about 30% for WideResNet-28-10. For AmoebaNet-B, although we have not reproduced the accuracy reported in [6], our OHL-Auto-Aug still achieves a 1.09% error rate reduction compared to Baseline model. Compared to the baseline we implemented, our OHL-Auto-Aug achieves an error rate of 1.90% which is about 45% error rate reduction. For all the models, our OHL-Auto-Aug exhibits a significantly error rate drop compared with Cutout.

We also find that the results of our OHL-Auto-Aug outperform those of Auto-Augment on all the networks except AmoebaNet-B. We think the main reason is that we could not train the Baseline AmoebaNet-B to a comparable accuracy with the original paper. By applying our OHL-Auto-Aug, the accuracy gap can be narrowed from 0.42% (Baseline accuracy comparing [6] with our implementation) to 0.14% (comparing OHL-Auto-Aug with Auto-Augment), which could demonstrate the effectiveness of our approach.

ImageNet Results: On ImageNet dataset, we perform our method on two networks: ResNet-50 [12] and SE-ResNeXt101 [13] to show the effectiveness of our augmentation strategy search. ResNet-50 contains 25.58M parameters and SE-ResNeXt101 contains 48.96M parameters. We regard ResNet-50 as a medium architecture and SE-

ResNeXt101 as a heavy architecture to illustrate the generalization of our OHL-Auto-Aug.

Table 3. Top-1 and Top-5 error rates (%) on ImageNet. We compare our OHL-Auto-Aug with standard augmentation (Baseline), standard augmentation with mixup [34] (mixup), augmentation strategy discovered by [6] (Auto-Augment). For both the ResNet-50 and SE-ResNeXt-101, our OHL-Auto-Aug improves the performance significantly.

Method	ResNet-50 [12]	SE-ResNeXt-101 [13]
Baseline	24.70/7.8	20.70/5.01
mixup $\alpha = 0.2$ [34]	23.3/6.6	–
Auto-Augment [6]	22.37/6.18	20.03/5.23 (our impl.)
OHL-Auto-Aug	21.07/5.68	19.30/4.57

In Table 3, we show the Top-1 and Top-5 error rates on validation set. We notice there exists another augmentation strategy mixup [34] which trains network on convex combinations of pairs of examples and their labels and achieves performance improvement on ImageNet dataset. We report the results of mixup together with Auto-Augment for comparison. All the experiments are conducted following the same setting described in Section 4.1. Since the original paper of [6] does not perform experiments on SE-ResNeXt101, we train it with the searched augmentation policy provided in [6]. As can be seen from the table, OHL-Auto-Aug achieves state-of-the-art accuracy on both models. For the medium model ResNet-50, our OHL-Auto-Aug increases by 3.7% points over the Baseline, 1.37% points over Auto-Augment, which is a significant improvement. For the heavy model SE-ResNeXt101, our OHL-Auto-Aug still boosts the performance by 1.4% even with a very high baseline. Both of the experiments demonstrate the effectiveness of our OHL-Auto-Aug.

Search Cost: The principle motivation of our OHL-Auto-Aug is to improve the efficiency of the search of auto-augmentation strategy. To demonstrate the efficiency of our OHL-Auto-Aug, we illustrate the search cost of our method together with Auto-Augment in Table 4. For fair comparison, we compute the total training iterations with conversion under a same batch size 1024 and denote as ‘#Iterations’. For example, Auto-Augment performs searching on CIFAR-10 by sampling 15,000 child models, each trained on a ‘reduced CIFAR-10’ with 4,000 randomly chosen images for 120 epochs. So the #Iterations of Auto-Augment on CIFAR-10 could be calculated as $15,000 \times 4,000 \times 120/1024 = 7.03 \times 10^6$. Similarly, Auto-Augment samples 15,000 child models and trains them each on a ‘reduced ImageNet’ of 6,000 images for 200 epochs, which equals to 1.76×10^7 #Iterations. Our OHL-Auto-Aug trains on the whole CIFAR-10 for 300 epochs with 8 trajectory samples, which equals to 1.17×10^5 #Iterations. For ImageNet, the #Iterations is $1.28 \times 10^6 \times 4 \times 150/1024 = 7.5 \times 10^5$.

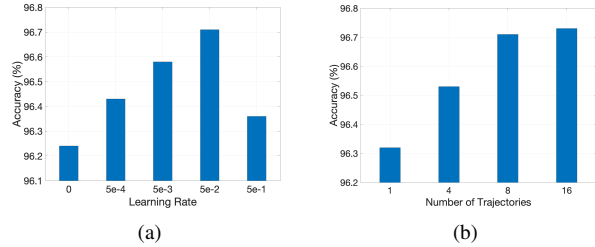


Figure 3. (a) Comparisons with different augmentation distribution learning rates. All the experiments are conducted with ResNet-18 on CIFAR-10 following the same setting as described in Section 4.1 except η_θ . As can be observed, too large η_θ will make the distribution parameters difficult to converge, while too small η_θ will slow down the convergence process, both harming the final performance. The chosen of $\eta_\theta = 5 \times 10^{-2}$ is a trade-off between convergence speed and performance. (b) Comparisons with different number of trajectory samples. All the experiments use ResNet-18 on CIFAR-10 following the same setting as described in Section 4.1 except N . As can be observed, increasing the number of trajectories from 1 to 8 steadily improves the performance of the network. When the number comes to ≥ 8 , the accuracy improvement is minor. We select $N = 8$ on CIFAR-10 as a trade-off between computation cost and performance.

As illustrated in Table 4, even training on the whole dataset, our OHL-Auto-Aug achieves $60\times$ faster on CIFAR-10 and $24\times$ faster on ImageNet than Auto-Augment. Instead of training on a reduced dataset, the high efficiency enables our OHL-Auto-Aug to train on the whole dataset, a guarantee for better performance. This also ensures our OHL-Auto-Aug algorithm could be easily performed on large dataset such as ImageNet without any bias on input data. Our OHL-Auto-Aug also gets rid of the need of retraining from scratch, further saving computation resources.

4.3. Analysis

In this section, we perform several analysis to illustrate the effectiveness of our proposed OHL-Auto-Aug.

Comparisons with different augmentation distribution learning rates: We conduct our OHL-Auto-Aug with different augmentation distribution learning rates: $\eta_\theta = 0, 5 \times 10^{-4}, 5 \times 10^{-3}, 5 \times 10^{-2}, 5 \times 10^{-1}$. All the experiments use ResNet-18 on CIFAR-10 following the same setting as described in Section 4.1 except η_θ . The final accuracies are illustrated in Figure 3(a). Note that our OHL-Auto-Aug chooses $\eta_\theta = 5 \times 10^{-2}$. Comparing the result of $\eta_\theta = 0$ with other results, we find that updating the augmentation distribution could produce better performed network than uniformly sampling from the whole search space. As can be observed, too large η_θ will make the distribution parameters difficult to converge, while too small η_θ will slow down the convergence process. The chosen of $\eta_\theta = 5 \times 10^{-2}$ is a trade-off for convergence speed and performance.

Table 4. Search cost of our method compared with Auto-Augment [6]. For fair comparison, we compute the total training iterations with conversion under a same batch size 1024 and denote as ‘#Iterations’. Specific computing implementation is detailed in Section 4.2. Our OHL-Auto-Aug achieves 60× faster on CIFAR-10 and 24× faster on ImageNet than Auto-Augment. Our OHL-Auto-Aug also gets rid of the need of retraining from scratch, further saving computation resources.

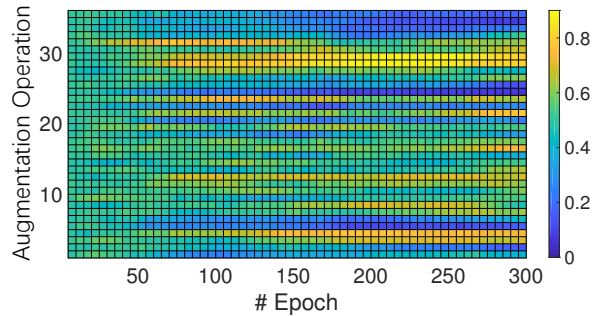
Dataset	Auto-Augment [6]		OHL-Auto-Aug	
	#Iterations	Usage of Dataset (%)	#Iterations	Usage of Dataset (%)
CIFAR-10	7.03×10^6	8%	1.17×10^5	100%
ImageNet	1.76×10^7	0.5%	7.5×10^5	100%
No Need to Retrain	×		✓	

Comparisons with different number of trajectory samples: We further conduct an analysis on the sensitivity of the number of trajectory samples. We produce experiments with ResNet-18 on CIFAR-10 with different number of trajectories: $N = 1, 4, 8, 16$. All the experiments follows the same setting as described in Section 4.1 except N . The final accuracies are illustrated in Figure 3(b). Principly, a larger number of trajectories will benefit the training for the augmentation distribution parameters due to the more accurate gradient calculated by Equation 4. Meanwhile, larger number of trajectories increases the memory requisition and the computation cost. As observed in Figure 3(b), increasing the number of trajectories from 1 to 8 steadily improves the performance of the network. When the number comes to ≥ 8 , the accuracy improvement is minor. We select $N = 8$ as a trade-off between computation cost and performance.

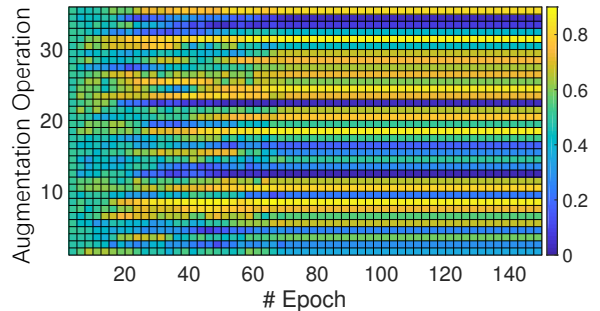
Analysis of augmentation distribution parameters: We visualize the augmentation distribution parameters of the whole training stage. We calculate the marginal distribution parameters of the first element in our augmentation operation. Specifically, we convert θ to a 36×36 matrix and sum up each row of the matrix following normalization. The results are illustrated in Figure 4(a) and Figure 4(b) for CIFAR-10 and ImageNet respectively. On CIFAR-10, we choose the network of ResNet-18 and show the results for every 5 epochs. On ImageNet we choose ResNet-50 and show the results for every 2 epochs. For both the two settings, we find that during training, our augmentation distribution parameters has converged. Some augmentation operations have been discarded, while the probabilities of some others have increased. We also notice that for different dataset, the distributions of augmentation operations are different. For instance, on CIFAR-10, the 36th operation, which is the ‘Color Invert’, is discarded, while on ImageNet, this operation is preserved with high probability. This further demonstrates our OHL-Auto-Aug could be easily performed on different datasets.

5. Conclusion

In this paper, we have proposed an online hyperparameter learning method for auto-augmentation strategy,



(a) Visualization of augmentation distribution parameters of ResNet-18 training on CIFAR-10.



(b) Visualization of augmentation distribution parameters of ResNet-50 training on ImageNet.

Figure 4. Analysis of augmentation distribution parameters. For both settings, we find that during training, our augmentation distribution parameters has converged. Some augmentation operations have been discarded, while the probabilities of some others have increased.

which formulates the augmentation policy as a parameterized probability distribution. Benefitting from the proposed bilevel framework, our OHL-Auto-Aug is able to optimize the distribution parameters iteratively with the network parameters in an online manner. Experimental results illustrate that our proposed OHL-Auto-Aug achieves remarkable auto-augmentation search efficiency, while establishes significantly accuracy improvements over baseline models.

References

- [1] Antreas Antoniou, Amos Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *arXiv preprint arXiv:1711.04340*, 2017.
- [2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(Feb):281–305, 2012.
- [3] Olivier Bousquet, Sylvain Gelly, Karol Kurach, Olivier Teytaud, and Damien Vincent. Critical hyper-parameters: No random, no cry. *arXiv preprint arXiv:1706.03200*, 2017.
- [4] Yunpeng Chen, Jianan Li, Huaxin Xiao, Xiaojie Jin, Shuicheng Yan, and Jiashi Feng. Dual path networks. In *NIPS*, pages 4467–4475, 2017.
- [5] Benoît Colson, Patrice Marcotte, and Gilles Savard. An overview of bilevel optimization. *Annals of operations research*, 153(1):235–256, 2007.
- [6] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018.
- [7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.
- [8] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.
- [9] Luca Franceschi, Michele Donini, Paolo Frasconi, and Massimiliano Pontil. Forward and reverse gradient-based hyperparameter optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1165–1173. JMLR. org, 2017.
- [10] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.
- [11] Minghao Guo, Zhao Zhong, Wei Wu, Dahua Lin, and Junjie Yan. Irlas: Inverse reinforcement learning for architecture search. *arXiv preprint arXiv:1812.05285*, 2018.
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.
- [13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.
- [14] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: <http://www.cs.toronto.edu/kriz/cifar.html>*, 2014.
- [15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [16] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [17] Joseph Lemley, Shabab Bazrafkan, and Peter Corcoran. Smart augmentation learning an optimal data augmentation strategy. *IEEE Access*, 5:5858–5869, 2017.
- [18] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. In *ECCV*, September 2018.
- [19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.
- [20] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. In *Advances in Neural Information Processing Systems*, pages 7827–7838, 2018.
- [21] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.
- [22] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.
- [23] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.
- [24] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018.
- [25] Shreyas Saxena and Jakob Verbeek. Convolutional neural fabrics. In *NIPS*, pages 4053–4061, 2016.
- [26] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.
- [27] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in neural information processing systems*, pages 2951–2959, 2012.
- [28] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.
- [29] Toan Tran, Trung Pham, Gustavo Carneiro, Lyle Palmer, and Ian Reid. A bayesian data augmentation approach for learning deep models. In *Advances in Neural Information Processing Systems*, pages 2797–2806, 2017.
- [30] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.
- [31] Ren Wu, Shengen Yan, Yi Shan, Qingqing Dang, and Gang Sun. Deep image: Scaling up image recognition. *arXiv preprint arXiv:1501.02876*, 2015.
- [32] Lingxi Xie and Alan L Yuille. Genetic cnn. In *ICCV*, pages 1388–1397, 2017.
- [33] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.
- [34] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
- [35] Xinyue Zhu, Yifan Liu, Zengchang Qin, and Jiahong Li. Data augmentation in emotion classification using genera-

tive adversarial networks. *arXiv preprint arXiv:1711.00648*, 2017.

- [36] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.
- [37] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6), 2017.
- [38] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. In *CVPR*, 2018.