Population Based Augmentation: Efficient Learning of Augmentation Policy Schedules

Daniel Ho 1 2  Eric Liang 1  Ion Stoica 1  Pieter Abbeel 1 3  Xi Chen 1 3

Abstract

A key challenge in leveraging data augmentation for neural network training is choosing an effective augmentation policy from a large search space of candidate operations. Properly chosen augmentation policies can lead to significant generalization improvements; however, state-of-the-art approaches such as AutoAugment are computationally infeasible to run for the ordinary user. In this paper, we introduce a new data augmentation algorithm, Population Based Augmentation (PBA), which generates nonstationary augmentation policy schedules instead of a fixed augmentation policy. We show that PBA can match the performance of AutoAugment on CIFAR-10, CIFAR-100, and SVHN, with three orders of magnitude less overall compute. On CIFAR-10 we achieve a mean test error of 1.46%, which is a slight improvement upon the current state-of-the-art. The code for PBA is open source and is available at https://github.com/arcelien/pba.

1. Introduction

Data augmentation techniques such as cropping, translation, and horizontal flipping are commonly used to train large neural networks (Lin et al., 2013). Augmentation transforms images to increase the diversity of image data. While deep neural networks can be trained on enormous numbers of data examples to exhibit excellent performance on tasks such as image classification, they contain a likewise enormous number of parameters, which causes overfitting. Data augmentation acts as a regularizer to combat this. However, most approaches used in training state-of-the-art networks only use basic types of augmentation. While neural network architectures have been investigated in depth (Krizhevsky et al., 2012; He et al., 2015; Szegedy et al., 2015; Simonyan & Zisserman, 2014; Zagoruyko & Komodakis, 2016; Huang et al., 2016; Han et al., 2016), less focus has been put into discovering strong types of data augmentation and data augmentation policies that capture data invariances.

A key consideration when applying data augmentation is picking a good set of augmentation functions, since redundant or overly aggressive augmentation can slow down training and introduce biases into the dataset (Graham, 2014). Many recent methods learn augmentation policies to apply different functions to image data. Among these, AutoAugment (Cubuk et al., 2018) stands out with state-of-the-art results in CIFAR-10 (Krizhevsky, 2009), CIFAR-100 (Krizhevsky, 2009), and ImageNet (Deng et al., 2009). Using a method inspired by Neural Architecture Search (Zoph & Le, 2016), Cubuk et al. learn a distilled list of augmentation functions and associated probability-magnitude values, resulting in a distribution of possible augmentations which can be applied to each batch of data. However, the search technique used in the work is very computationally expensive, and code has not been released to reproduce it. In this work, we address these issues with a simple and efficient algorithm for augmentation policy learning.

Figure 1. PBA matches AutoAugment’s classification accuracy across a range of different network models on the CIFAR-10 dataset, while requiring 1,000x less GPU hours to run. For the full set of results, refer to Table 2. Assuming an hourly GPU cost of $1.5, producing a new augmentation policy costs around $7.5 for PBA vs $7,500 with AutoAugment. The same scaling holds for the SVHN dataset as well.
Table 1. Comparison of pre-computation costs and test set error (%) between this paper, AutoAugment (AA), and the previous best published results. Previous results did not pre-compute augmentation policies. AutoAugment reported estimated cost in Tesla P100 GPU hours, while PBA measured cost in Titan XP GPU hours. Besides PBA, all metrics are cited from (Cubuk et al., 2018). For more detail, see Table 2. *CIFAR-100 models are trained with the policies learned on CIFAR-10 data.

| Dataset    | Value     | Previous Best | AA   | PBA  |
|------------|-----------|---------------|------|------|
| CIFAR-10   | GPU Hours | -             | 5000 | 5    |
|            | Test Error| 2.1           | 1.48 | 1.46 |
| CIFAR-100  | GPU Hours | -             | 0    | 0    |
|            | Test Error| 12.2          | 10.7 | 10.9 |
| SVHN       | GPU Hours | -             | 1000 | 1    |
|            | Test Error| 1.3           | 1.0  | 1.1  |

Our formulation of data augmentation policy search, Population Based Augmentation (PBA), reaches similar levels of final performance on a variety of neural network models while utilizing orders of magnitude less compute. We learn a robust augmentation policy on CIFAR-10 data in five hours using one NVIDIA Titan XP GPU, and we visualize its performance in Figure 1. Relative to the several days it takes to train large CIFAR-10 networks to convergence, the cost of running PBA beforehand is marginal and significantly enhances results. These results are summarized in Table 1. PBA leverages the Population Based Training algorithm (Jaderberg et al., 2017) to generate an augmentation schedule that defines the best augmentation policy for each epoch of training. This is in contrast to a fixed augmentation policy that applies the same transformations independent of the current epoch number.

We release code to run and evaluate our augmentation search algorithm at https://github.com/arcellien/pba. This allows an ordinary workstation user to easily experiment with the search algorithm and augmentation operations. A particularly interesting use case would be to introduce new augmentation operations, perhaps targeted towards a particular dataset or image modality, and be able to quickly produce a tailored, high performing augmentation schedule. Our code uses the Ray (Moritz et al., 2017) implementation of PBT, which allows for easy parallelization across and within GPUs and CPUs.

This paper is organized as follows: First, we cover relevant background and AutoAugment (Section 2). We then introduce the PBA algorithm (Section 3). We describe the augmentation schedules PBA discovers and its performance on several datasets. Finally, we seek to understand the efficiency gains of PBA through ablation studies and comparison with baseline methods (Section 4).

2. Background

2.1. Related Work

We first review types of data augmentation for image recognition, which improve generalization with limited data by applying transformations to generate additional samples. Common techniques such as random cropping, flipping, rotating, scaling, and translating are used by top performing models for training on MINST, CIFAR-10, and ImageNet datasets (Simard et al., 2003; Ciresan et al., 2012; Wan et al., 2013; Sato et al., 2015; Krizhevsky et al., 2012; Lin et al., 2013; Han et al., 2016). Some additional approaches to generate augmented data include image combining (Inoue, 2018; Okafo et al., 2018), elastic distortions (Wong et al., 2016), and generative adversarial networks (Anoni et al., 2017).

Augmentation has been shown to have a large impact on image modalities where data is scarce or expensive to generate, like medical imaging (Bowles et al., 2018; Frid-Adar et al., 2018) or non-supervised learning approaches (Mundhenk et al., 2017).

Several papers have attempted to automate the generation of data augmentations with data-driven learning. These use methods such as manifold learning (Paschali et al., 2019), Bayesian Optimization (Tran et al., 2017), and generative adversarial networks which generate transformation sequences (Ratner et al., 2017). Additionally, (Lemley et al., 2017) uses a network to combine pairs of images to train a target network, and (DeVries & Taylor, 2017) injects noise and interpolates images in an autoencoder learned feature space. AutoAugment (Cubuk et al., 2018) uses reinforcement learning to optimize for accuracy in a discrete search space of augmentation policies.

Our approach was inspired by work in hyperparameter optimization. There has been much previous work to well-tune hyperparameters, especially in Bayesian Optimization (Srinivas et al., 2009; Bergstra et al., 2011; Snoek et al., 2012; Hutter et al., 2011), which are sequential in nature and expensive computationally. Other methods incorporate parallelization or use non-bayesian techniques (Li et al., 2016; Golovin et al., 2017; Shah & Ghahramani, 2015; Springenberg et al., 2016; González et al., 2016) but still either require multiple rounds of optimization or large amounts of compute. These issues are resolved in Population Based Training (Jaderberg et al., 2017), which builds upon both evolutionary strategies (Clune et al., 2008) and random search (Bergstra & Bengio, 2012) to generate non-stationary, adaptive hyperparameter schedules in a single round of model training.
2.2. AutoAugment

Cubuk et al. shows that using a diverse, stochastic mix of augmentation operations can significantly reduce generalization error. They automate the search over the space of data augmentation policies in a method called AutoAugment, which significantly improves neural network model accuracy on a variety of image datasets. AutoAugment follows an approach similar to work in the neural architecture search area (Zoph et al., 2017; Pham et al., 2018) where a controller RNN network is trained via reinforcement learning to output augmentation policies maximizing for accuracy (Figure 2). However, this approach is expensive in both time and compute, as the signal for the controller has to be generated by training thousands of models to convergence on different augmentation policies and evaluating final validation accuracy.

Cubuk et al. curated an augmentation policy search space based on operations from the PIL python library. These include ShearX/Y, TranslateX/Y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness, Cutout (Devries & Taylor, 2017), and SamplePairing (Inoue, 2018). Each operation has two associated parameters: probability and magnitude. The authors used discrete probability values from 0% to 100%, in increments of 10%. Magnitude can range from 0 to 9 inclusive, but a few operations ignore this value and apply a constant effect.

A policy would then consist of five sub-policies, each consisting of two operations and associated parameters. For every batch of data, one randomly selected sub-policy would be applied. In total, the final policy for AutoAugment concatenated the five best performing policies for a total of 25 sub-policies.

To learn an augmentation policy, 15,000 sampled policies were evaluated on a Wide-ResNet-40-2 (40 layers, widening factor of 2) child model (Zagoruyko & Komodakis, 2016) by taking the validation accuracy after training for 120 epochs on a “reduced” dataset. For CIFAR-10, this consists of 4,000 images from the training set, and for SVHN, 1,000 images. CIFAR-100 is trained with a transferred augmentation policy from CIFAR-10.

3. Population Based Augmentation

In this section we introduce the design and implementation of the PBA algorithm.

3.1. Why Augmentation Schedules?

The end goal of PBA is to learn a schedule of augmentation policies as opposed to a fixed policy. As we will see, this choice is responsible for much of the efficiency gains of PBA (Section 4). Though the search space for schedules over training epochs is larger than that of fixed policies \( f \in F \) by a factor of \( |F|^\text{epochs} \), counter-intuitively, PBA shows that it is far more efficient to search for a good schedule than a
fixed policy. Several factors contribute to this.

First, estimating the final test error of a fixed augmentation policy is difficult without running the training of a child model to completion. This is true in particular because the choice of regularizing hyperparameters (e.g., data augmentation functions) primarily impacts the tail end of training. Therefore, estimating the final performance of a given fixed augmentation policy requires training a model almost to completion. In contrast, it is straightforward to reuse prior computations to estimate the performance of two variants of a schedule that share a prefix.

Second, there is reason to believe that it is easier to find a good augmentation policy when searching in the space of schedules. An augmentation function that can reduce generalization error at the end of training is not necessarily a good function at initial phases. Such functions would be selected out when holding the augmentation function fixed for the entirety of training. And though the space of schedules is large, most good schedules are necessarily smooth and hence easily discoverable through evolutionary search algorithms such as PBT.

3.2. Learning a Schedule

In PBA we consider the augmentation policy search problem as a special case of hyperparameter schedule learning. Thus, we leverage Population Based Training (PBT) (Jaderberg et al., 2017): a hyperparameter search algorithm which optimizes the parameters of a network jointly with their hyperparameters to maximize performance. The output of PBT is not an optimal hyperparameter configuration but rather a trained model and schedule of hyperparameters. In PBA, we are only interested in the learned schedule and discard the child model result (similar to AutoAugment). This learned augmentation schedule can then be used to improve the training of different (i.e., larger and costlier to train) models on the same dataset.

PBT executes as follows. To start, a fixed population of models are randomly initialized and trained in parallel. At certain intervals, an “exploit-and-explore” procedure is applied to the worse performing population members, where the model clones the weights of a better performing model (i.e., exploitation) and then perturbs the hyperparameters of the cloned model to search in the hyperparameter space (i.e., exploration). Because the weights of the models are cloned and never reinitialized, the total computation required is the computation to train a single model times the population size.

The Ray framework (Moritz et al., 2017) includes a parallelized implementation of PBT (https://ray.readthedocs.io/en/latest/tune.html) which handles the exploit-and-explore process in the backend. This implementation allows a user to deploy multiple trials on the same GPU, provided there is enough GPU memory. When the models only require a fraction of the computation resources and memory of an entire GPU, as in this work, training is sped up by fully utilizing the GPU.

3.3. Policy Search Space

In Algorithm 1, we describe the augmentation policy function used in PBA and the optimization strategy we adapt from PBT. The challenge here is defining a smooth parameterization of the augmentation policy so that PBT can incrementally adopt good augmentations, while still allowing good coverage of the search space within a limited number of perturbations.

To make PBA more directly comparable with AutoAugment, we attempt to preserve the qualities of the AutoAugment formulation when possible, using the same augmentation functions, a similar number of total augmentation functions in the policy, and the same set of magnitude variants per function as applicable. Our augmentation policy search space consists of the augmentation operations from AutoAugment, less SamplePairing (Inoue, 2018), for a total of 15 operations. We use the same code and magnitude options derived from PIL operations to ensure a fair comparison based on search algorithm performance.

We define a set of hyperparameters consisting of two magnitude and probability values for each operation, with discrete possibilities for each. This gives us 30 operation-magnitude-probability tuples for a total of 60 hyperparameters. Like AutoAugment, we have 10 possibilities for magnitude and

---

**Algorithm 1** The PBA augmentation policy template, the parameters of which are optimized by PBT. The parameter vector is a vector of \((op, prob, mag)\) tuples. There are two instances of each \(op\) in the vector, and this parameter cannot be changed. PBT learns a schedule for the \(prob\) and \(mag\) parameters during the course of training a population of child models.

**Input:** \(data x\), \(parameters p\), \([list of (op, prob, mag)]\)

Shuffle parameters

Set \(count = [0, 1, 2]\) with probability \([0.2, 0.3, 0.5]\)

for \((op, prob, mag)\) in \(p\) do

\(\text{if } count = 0 \text{ then}\)

break

\(\text{end if}\)

\(\text{if } \text{random}(0, 1) < prob \text{ then}\)

\(x = count - 1\)

\(count = op(x, mag)\)

\(\text{end if}\)

\(\text{end for}\)

Return \(x\)
Algorithm 2 The PBA explore function. Probability parameters have possible values from 0% to 100% in increments of 10%, and magnitude parameters have values from 0 to 9 inclusive.

Input: Params $p$, list of augmentation hyperparameters

for param in $p$ do
    if random(0, 1) < 0.2 then
        Resample param uniformly from domain
    else
        amt = [0,1,2,3] uniformly at random
        if random(0, 1) < 0.5 then
            param = param + amt
        else
            param = param - amt
        end if
        Clip param to stay in domain
    end if
end for

4. Experiments and Analysis

In this section, we describe experiments we ran to better understand the performance and characteristics of the PBA algorithm. We seek to answer the following questions:

1. How does classification accuracy and computational cost of PBA compare to state-of-the-art and random search baselines?

2. Where does the performance gain of PBA come from – does having a schedule of augmentations really matter, or is a stationary distribution sufficient?

3. How does PBA performance scale with the amount of computation used?

4. How sensitive is PBA to the hyperparameters of the optimization procedure – did we just move part of the optimization process into hyperparameter selection?

4.1. Comparison with Baselines

Accuracy (CIFAR-10, CIFAR-100, SVHN) We first compare PBA to other state-of-the-art methods on the CIFAR-10 (Krizhevsky, 2009) and SVHN (Netzer et al., 2011) datasets. Following (Cubuk et al., 2018), we search over a “reduced” dataset of 4,000 and 1,000 training images for CIFAR-10 and SVHN respectively. Comparatively, CIFAR-10 has a total of 50,000 training images and SVHN has 73,257 training images with an additional 531,131 “extra” training images. PBA is run with 16 total trials on the Wide-ResNet-40-2 model to generate augmentation schedules.

For the augmentation policy, we initialize all magnitude and probability values to 0, as we hypothesize that less augmentation is required early in training when the validation accuracy is close to training accuracy. However, since training error decreases faster than validation error as training progresses, more regularization should be required, so we expect the probability and magnitude values to increase as training progresses. This would counteract overfitting as we introduce the model to more diverse data.

We then train models on the full training datasets, using the highest performing augmentation schedules discovered on the reduced datasets. The schedule learned on reduced CIFAR-10 is used to train final models on reduced CIFAR-10, CIFAR-10, and CIFAR-100. The schedule learned on reduced SVHN is used to train final models on reduced SVHN and SVHN. We report results in Table 2. Each model is evaluated five times with different random initializations, and we report both the mean and standard deviation test set error in %.

The models we evaluate on include: Wide-ResNet-28-10 (Zagoruyko & Komodakis, 2016), Shake-Shake (26 2x32d)
Table 2. Test set error (%) on CIFAR-10, CIFAR-100, and SVHN. Lower is better. The baseline applies regular random crop and horizontal flip operations. Cutout is applied on top of the baseline, and PBA/AutoAugment are applied on top of Cutout. We report the mean final test error of 5 random model initializations. We used the models: Wide-ResNet-28-10 (Zagoruyko & Komodakis, 2016), Shake-Shake (26 2x32d) (Gastaldi, 2017), Shake-Shake (26 2x96d) (Gastaldi, 2017), Shake-Shake (26 2x112d) (Gastaldi, 2017), and PyramidNet with ShakeDrop (Han et al., 2016; Yamada et al., 2018). Code for AA eval on SVHN was not released, so differences between our implementations could impact results. Thus, we report AA* from our re-evaluation.

| Dataset          | Model                      | Baseline | Cutout | AA     | AA*    | PBA    |
|------------------|----------------------------|----------|--------|--------|--------|--------|
| CIFAR-10         | Wide-ResNet-28-10          | 3.87     | 3.08   | 2.68   | 2.58 ± 0.062 |
|                  | Shake-Shake (26 2x32d)     | 3.55     | 3.02   | 2.47   | 2.54 ± 0.10 |
|                  | Shake-Shake (26 2x96d)     | 2.86     | 2.56   | 1.99   | 2.03 ± 0.11 |
|                  | Shake-Shake (26 2x112d)    | 2.82     | 2.57   | 1.89   | 2.03 ± 0.080 |
|                  | PyramidNet+ShakeDrop       | 2.67     | 2.31   | 1.48   | 1.46 ± 0.077 |
| Reduced CIFAR-10 | Wide-ResNet-28-10          | 18.84    | 17.05  | 14.13  | 12.82 ± 0.26 |
|                  | Shake-Shake (26 2x96d)     | 17.05    | 13.40  | 10.04  | 10.64 ± 0.22 |
| CIFAR-100        | Wide-ResNet-28-10          | 18.8     | 18.41  | 17.09  | 16.73 ± 0.15 |
|                  | Shake-Shake (26 2x96d)     | 17.05    | 16.00  | 14.28  | 15.31 ± 0.28 |
|                  | PyramidNet+ShakeDrop       | 13.99    | 12.19  | 10.67  | 10.94 ± 0.094 |
| SVHN             | Wide-ResNet-28-10          | 1.50     | 1.40   | 1.07   | 1.13 ± 0.024 |
|                  | Shake-Shake (26 2x96d)     | 1.40     | 1.20   | 1.02   | 1.10 ± 0.032 |
|                  | PyramidNet+ShakeDrop       | 13.21    | 12.52  | 8.15   | 7.83 ± 0.22 |
| Reduced SVHN     | Wide-ResNet-28-10          | 13.22    | 24.22  | 5.92   | 6.46 ± 0.13 |
|                  | Shake-Shake (26 2x96d)     | 13.32    | 32.5   | 8.15   | 7.83 ± 0.22 |

While model accuracy on Reduced CIFAR-10 would have likely been improved with hyperparameter tuning for the reduced dataset size and smaller Wide-ResNet-40-2 model, our result shows that no hyperparameter tuning is required for high performance.

Overall, the PBA learned schedule leads AutoAugment slightly on PyramidNet and Wide-ResNet-28-10, and performs comparably on Shake-Shake models, showing that the learned schedule is competitive with state-of-the-art.

We visualize the discovered schedule used in training our final CIFAR models in Figure 4. For the AutoContrast, Equalize, and Invert augmentations, magnitude values were ignored. From the probability values, our schedule seems to contain all augmentations to at least a moderate degree at some point, which is reasonable given our random perturb exploration method. However, there is emphasis on Cutout, Posterize, Invert, Equalize, and AutoContrast throughout the schedule.

(Cubuk et al., 2018) suggests that color-based transformations are more useful on CIFAR compared to geometric ones, and our results also indicate this. However, they also found that the Invert transformation is almost never used, while it was very common in our schedule. A possible explanation may be that a model is able to better adapt to Invert when using a nonstationary policy. PBA may be exploring systematically different parts of the design space than AutoAugment. Alternatively, it may be that by the randomness in PBA, Cutout was introduced and impacted performance. It may be fruitful to explore combinations....
Population Based Augmentation

(a) Operation magnitudes increase rapidly in the initial phase of training, eventually reaching a steady state around epoch 130.

(b) Normalized plot of operation probability parameters over time. The distribution flattens out towards the end of training.

Figure 4. Plots showing the evolution of PBA operation parameters in the discovered schedule for CIFAR-10. Note that each operation actually appears in the parameter list twice; we take the mean parameter value for each operation in this visualization.

of PBA and AutoAugment to design nonstationary policies with more precision from a RNN Controller.

CIFAR-100 We additionally evaluate on CIFAR-100 using the same augmentation schedule discovered using Reduced CIFAR-10. We find that these results are also competitive with AutoAugment and significantly better than Baseline or only applying Cutout.

SVHN We ran PBA for 160 epochs on a 1,000 image Reduced SVHN dataset to discover an augmentation policy schedule without tuning any parameters of the algorithm. See the appendix for a visualization of an example PBA policy on the SVHN dataset.

We then trained models on both the Reduced SVHN and SVHN Full (core training data with extra data), using the discovered schedule. Except for the Wide-ResNet-28-10 model on Reduced SVHN, training was done without tuning.

Figure 5. Plot of the expected best child test accuracy after a given number of random trials on Wide-ResNet-40-2. Random policy schedules were generated by randomly selecting intervals of length between 1 and 40, and then selecting a random policy for the interval. All values were selected uniformly from the domain.

using the hyperparamters from AutoAugment. We were able to obtain a policy comparable with AutoAugment. This demonstrates the robustness of the PBA algorithm across datasets.

Examining the learned policy schedule, we observe that Cutout, Translate Y, Shear X, and Invert stand out as being present with high probability across all epochs. This fits with the findings of (Cubuk et al., 2018), indicating that Invert and geometric transformations are successful in SVHN because it is important to learn invariances to these augmentations. From another perspective, all of the augmentations appear with reasonable probability at some point in the schedule, which suggests that using a preliminary strategy like AutoAugment to filter out poor performing augmentations would be an interesting direction to explore.

Computational Cost AutoAugment samples and evaluates \( \sim 15,000 \) distinct augmentation policies on child models, which requires about \( 15000 \times 120 = 1.8m \) epochs of training. In comparison, PBA leverages PBT to learn a schedule with a population of 16 child models. PBA uses 200 epochs of training per child model, for a total of 3200 epochs, or over 500x less compute than AutoAugment.

As a second baseline, we also train 250 child models with randomly selected augmentation policies, and 250 child models with randomly selected augmentation schedules. In Figure 5, we use this data to plot the expected maximum child model test accuracy after a given number of random trials. As shown, it takes over 250 trials for the expected child accuracy, which is strongly correlated with final accuracy, to approach that reached by a single 16-trial PBA run. Hence, PBA still provides over an order of magnitude speedup here.
Table 3. Ablation study: We evaluate models on CIFAR-10 using a fixed policy (the last policy of the PBA schedule learned on Reduced CIFAR-10), shuffled schedule order, and a fully collapsed schedule, comparing to results with the original PBA schedule. See Section 4.2 for further explanation. We evaluate each model once, and some combinations were not evaluated due to cost considerations.

| Model                  | Cutout | Fixed Policy | Order-shuffled | Fully-shuffled | PBA       |
|------------------------|--------|--------------|----------------|----------------|-----------|
| Wide-ResNet-28-10      | 3.08   | 2.76         | 2.66           | 2.89           | 2.576 ± 0.062 |
| Shake-Shake (26 2x32d)| 3.02   | 2.73         | -              | -              | 2.54 ± 0.10  |
| Shake-Shake (26 2x96d)| 2.56   | 2.33         | -              | -              | 2.03 ± 0.11  |
| Shake-Shake (26 2x112d)| 2.57  | 2.09         | -              | -              | 2.03 ± 0.080 |
| PyramidNet+ShakeDrop   | 2.31   | 1.55         | -              | -              | 1.46 ± 0.077 |

**Real-time Overhead** Since PBT trains all members of its population simultaneously, the minimal real-time overhead is just the time it takes to train one child model. In practice, there is a slight overhead from the mutation procedures triggered by PBT, but the overall search time is still small compared to the time to train the primary model. In contrast, AutoAugment leverages reinforcement-learning based techniques, in which a Recurrent Neural Network (RNN) controller is trained with the reinforcement learning algorithm Proximal Policy Optimization (PPO) (Schulman et al., 2017). Using this strategy, new augmentation policies can only be sampled and trained after the previous batch of samples has completed, so parallelization is limited to the batch size of the PPO update.

**4.2. Does having a schedule matter?**

PBA distinguishes itself from AutoAugment by learning a augmentation policy schedule, where the distribution of augmentation functions can vary as a function of the training epoch. To check whether a schedule contributes to performance, we try training the model using (1) the last augmentation policy of the PBA schedule as a fixed policy, (2) the augmentation schedule with the order of policies shuffled but the duration of each policy fixed, and (3) the augmentation schedule collapsed into a time-independent stationary distribution of augmentations (i.e., a policy is sampled independently for each batch of data, where each policy is weighted by its duration).

In Table 3, we see that training with the PBA Fixed Policy degrades accuracy by ~10% percent on average, which is significantly worse than training with the full schedule. Compared to using Cutout, the fixed policy gives up ~50% of gains on Wide-ResNet-28-10, Shake-Shake 32, and Shake-Shake 96, and ~10% of gains on Shake-Shake 112 and PyramidNet. This shows that the augmentation schedule improves accuracy over a fixed policy, especially on smaller models.

Similarly, when we evaluated the shuffled schedules (only on Wide-ResNet-28-10), accuracy is also significantly lower, showing that a stationary distribution derived from the schedule does not emulate the schedule. We hypothesize that schedule improves training by allowing "easy" augmentations in the initial phase of training while still allowing "harder" augmentations to be added later on.

**4.3. Hyperparameter Tuning and Sensitivity**

We did not tune the discrete space for magnitude or probability options to keep our policy easy to compare to AutoAugment. We have two copies of each operation, as the AutoAugment sub-policy is able to contain two copies of the same operation as well.

For the search algorithm, we lightly tuned the explore function and the distribution for count in Algorithm 1, the maximum number of augmentation functions to apply for each batch of data. While we keep the maximum value of count at 2 in line with AutoAugment’s length 2 subpolicy, there may be room for performance improvement by carefully tuning the distribution.

We tried perturbation intervals of 2 and 4 once, but did not find this value to be sensitive. We also tried to run PBT for 100 epochs, but found this to slightly decrease performance when evaluated on models for 200 epochs.

It may be interesting to consider training a larger child model (e.g. Shake-Shake) for 1,800 epochs to generate a schedule over the full training duration and eliminate the need to stretch the schedule. In a similar vein, an experiment to use PBT directly on the full CIFAR-10 dataset or Wide-ResNet-28-10 model may lead to better performance, and is computationally feasible with PBA.

**5. Conclusion**

This paper introduces PBA, a novel formulation of data augmentation search which quickly and efficiently learns state-of-the-art augmentation policy schedules. PBA is simple to implement within any PBT framework, and we release the code for PBA as open source.
Acknowledgements

We thank Richard Liaw, Dogus Cubuk, Quoc Le, and the ICML reviewers for helpful discussion.

References

Antoniou, A., Storkey, A. J., and Edwards, H. A. Data augmentation generative adversarial networks. CoRR, abs/1711.04340, 2017.

Bergstra, J. and Bengio, Y. Random search for hyper-parameter optimization. Journal of Machine Learning Research, 13:281–305, 2012.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. Algorithms for hyper-parameter optimization. In NIPS, 2011.

Bowles, C., Chen, L., Guerrero, R., Bentley, P., Gunn, R. N., Hammers, A., Dickie, D. A., del C. Valdés Hernández, M., Wardlaw, J. M., and Rueckert, D. Gan augmentation: Augmenting training data using generative adversarial networks. CoRR, abs/1810.10863, 2018.

Ciresan, D., Meier, U., and Schmidhuber, J. Multicolumn deep neural networks for image classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition, Jun 2012. doi: 10.1109/cvpr.2012.6248110. URL http://dx.doi.org/10.1109/CVPR.2012.6248110.

Clune, J., Misevic, D., Ofria, C., Lenski, R. E., Elena, S. F., and Sanjuán, R. Natural selection fails to optimize mutation rates for long-term adaptation on rugged fitness landscapes. PLoS Computational Biology, 4:1145 – 1146, 2008.

Cubuk, E. D., Zoph, B., Mané, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation policies from data. CoRR, abs/1805.09501, 2018. URL http://arxiv.org/abs/1805.09501.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In 2009 IEEE Conference on Computer Vision and Pattern Recognition, pp. 248–255, June 2009. doi: 10.1109/CVPR.2009.5206848.

Devries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. CoRR, abs/1708.04552, 2017. URL http://arxiv.org/abs/1708.04552.

DeVries, T. and Taylor, G. W. Dataset augmentation in feature space. CoRR, abs/1702.05538, 2017. URL https://arxiv.org/abs/1702.05538.

Frid-Adar, M., Klang, E., Amitai, M., Goldberger, J., and Greenspan, H. Synthetic data augmentation using gan for improved liver lesion classification. 2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018), pp. 289–293, 2018.

Gastaldi, X. Shake-shake regularization. CoRR, abs/1705.07485, 2017. URL http://arxiv.org/abs/1705.07485.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J. E., and Sculley, D. (eds.). Google Vizier: A Service for Black-Box Optimization, 2017. URL https://research.google.com/pubs/archive/46180.pdf.

González, J., Dai, Z., Hennig, P., and Lawrence, N. D. Batch bayesian optimization via local penalization. In AISTATS, 2016.

Graham, B. Fractional max-pooling. CoRR, abs/1412.6071, 2014. URL http://arxiv.org/abs/1412.6071.

Han, D., Kim, J., and Kim, J. Deep pyramidal residual networks. CoRR, abs/1610.02915, 2016. URL http://arxiv.org/abs/1610.02915.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. CoRR, abs/1603.05027, 2016. URL http://arxiv.org/abs/1603.05027.

Huang, G., Liu, Z., and Weinberger, K. Q. Densely connected convolutional networks. CoRR, abs/1608.06993, 2016. URL http://arxiv.org/abs/1608.06993.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. Sequential model-based optimization for general algorithm configuration. In LION, 2011.

Inoue, H. Data augmentation by pairing samples for images classification. CoRR, abs/1801.02929, 2018. URL http://arxiv.org/abs/1801.02929.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. Population based training of neural networks. CoRR, abs/1711.09846, 2017. URL http://arxiv.org/abs/1711.09846.

Krizhevsky, A. Learning multiple layers of features from tiny images. Tech Report, 2009.
when to warp? *CoRR*, abs/1609.08764, 2016. URL http://arxiv.org/abs/1609.08764.

Yamada, Y., Iwamura, M., and Kise, K. Shakedrop regularization. *CoRR*, abs/1802.02375, 2018. URL http://arxiv.org/abs/1802.02375.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016. URL http://arxiv.org/abs/1611.01578.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. URL http://arxiv.org/abs/1707.07012.
A. PBA Scalability with Compute

In Figure 6 we look at how large the model and PBT population size is necessary to learn an effective schedule. The population size determines how much of the search space is explored during training, and also the computational overhead of PBA. Our results indicate that a population size of 16 WRN-40-2 models performs the best. Having more than 16 trials seems not to help, and having less than 16 seems to lead to decreased performance. However, we found that results could fluctuate significantly between runs of PBT, most likely due to exploring a very limited search space with a noisy exploration strategy.

Besides WRN-40-2, we also tried to use a ResNet-20 (He et al., 2016) model for PBT population, which required about half the compute. Empirical results (in Table 4 and Figure 6) suggest that the ResNet-20 population does not achieve as high of a test accuracy as with WRN-40-2, but results were relatively close. Because a ResNet-20 model has much less parameters, training accuracy plateaus faster than WRN-40-2, which may change the effects of augmentation.

B. Model Hyperparameters

The hyperparameters used to train WideResNet-40-2 to discover augmentation schedules, and also the ones used to train final models, are displayed in Table 5. For full details on the hyperparameters and implementation, see the open source code.
Population Based Augmentation

Table 4. Test error during PBT search and policy schedule evaluated afterwards, for varying population sizes and models. PBA Search with variation of model and compute, on Reduced CIFAR-10 dataset. ResNet-20 (Res) took approximately half the compute of WideResNet-40-2 (WRN). Number in title is the population size, and speedup is relative to AutoAugment. Note that models with larger population sizes, while scoring high during the search, don’t actually perform better when re-evaluated.

| Model                  | 8-Res | 16-Res | 32-Res | 16-WRN | 32-WRN | 64-WRN |
|------------------------|-------|--------|--------|--------|--------|--------|
| WRN-40-2 during search | -     | -      |        | 0.8484 | 0.8446 | 0.8523 |
| WRN-40-2               | -     | -      | -      | 0.8452 | 0.8445 | 0.8446 |
| ResNet-20 during search| 0.7484| 0.7657 | 0.7619 | -      | -      | -      |
| ResNet-20              | 0.7457| 0.7545 | 0.7534 | -      | -      | -      |
| WRN-28-10              | 0.9711| 0.9721 | 0.9740 | 0.9740 | 0.9736 | 0.9703 |

Relative Speedup: 2250x 1125x 562.5x 562.5x 281.25x 140.625x

Table 5. Hyperparameters used for evaluation on CIFAR-10, CIFAR-100, and (R)educed-CIFAR-10. Besides Wide-ResNet-28-10 and Wide-ResNet-40-2 on Reduced SVHN, no hyperparameter tuning was done. Instead, all hyperparameters are the same as those used in AutoAugment.

| Dataset    | Model                  | Learning Rate | Weight Decay | Batch Size |
|------------|------------------------|---------------|--------------|------------|
| CIFAR-10   | Wide-ResNet-40-2       | 0.1           | 0.0005       | 128        |
| CIFAR-10   | Wide-ResNet-28-10      | 0.1           | 0.0005       | 128        |
| CIFAR-10   | Shake-Shake (26 2x32d) | 0.01          | 0.001        | 128        |
| CIFAR-10   | Shake-Shake (26 2x96d) | 0.01          | 0.001        | 128        |
| CIFAR-10   | Shake-Shake (26 2x112d)| 0.01          | 0.001        | 128        |
| CIFAR-10   | PyramidNet+ShakeDrop   | 0.05          | 0.00005      | 64         |
| CIFAR-100  | Wide-ResNet-28-10      | 0.1           | 0.0005       | 128        |
| CIFAR-100  | Shake-Shake (26 2x96d) | 0.01          | 0.0025       | 128        |
| CIFAR-100  | PyramidNet+ShakeDrop   | 0.025         | 0.0005       | 64         |
| R-CIFAR-10 | Wide-ResNet-28-10      | 0.05          | 0.005        | 128        |
| R-CIFAR-10 | Shake-Shake (26 2x96d) | 0.025         | 0.0025       | 128        |
| SVHN       | Wide-ResNet-40-2       | 0.05          | 0.005        | 128        |
| SVHN       | Wide-ResNet-28-10      | 0.005         | 0.001        | 128        |
| SVHN       | Shake-Shake (26 2x96d) | 0.01          | 0.00015      | 128        |
| R-SVHN     | Wide-ResNet-28-10      | 0.05          | 0.01         | 128        |
| R-SVHN     | Shake-Shake (26 2x96d) | 0.025         | 0.005        | 128        |
Figure 8. Plots showing the evolution of PBA operation parameters in a schedule learned on Reduced SVHN. Note that each operation actually appears in the parameter list twice; we take the mean parameter value for each operation in this visualization.