Recent advances in automatic data augmentation

Nikos Tsilivis

National Technical University of Athens

ntsilivis96@gmail.com

October 15, 2019

Overview

What is Data Augmentation?

Data Augmentation is the process of altering instances of the input of a classifier, in order to increase and enrich a given dataset. It can be used as a tool to make our system "aware" of several invariances of the input. Typical examples include affine transformations, image crops and morphological operations.

Why Automatic Augmentation?

Image datasets vary in shapes and symmetries and successful data augmentation techniques are not necessarily transferred from one dataset to another. That means designing augmentation techniques for different datasets is done every time from scratch, with careful examination of the dataset.

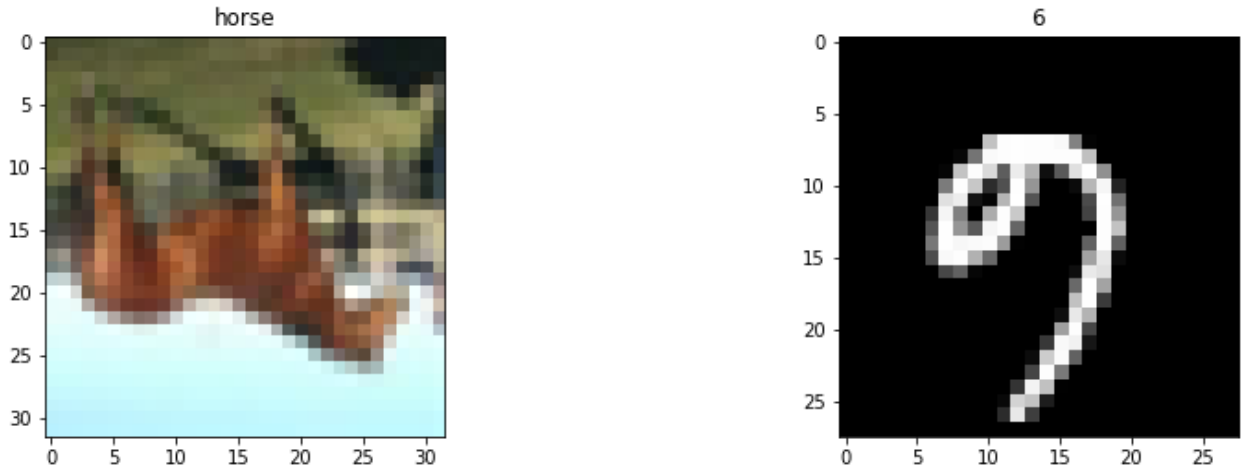


Figure: Horse remains a horse when rotated by 180 degrees, but 9 becomes a 6 (from CIFAR10 and MNIST)

AutoAugment

The proposal from Ekin D. Cubuk, Barret Zoph et al. is an automatic process of learning the best augmentation policies for a given dataset from the data itself. It is formulated as a discrete space search problem in which they try to maximize the validation accuracy of a child network. They success state-of-art results in a variety of datasets and also manage to find transferable augmentation techniques.

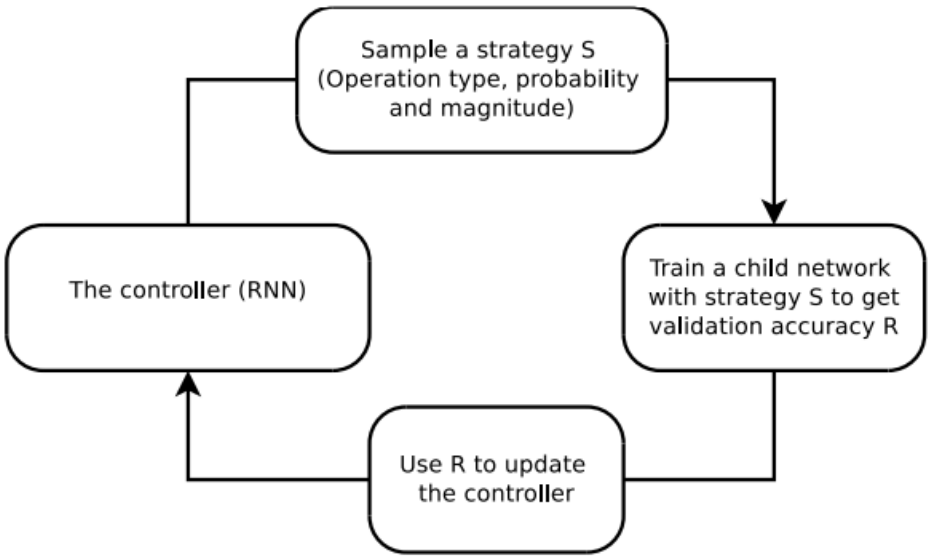


Figure: AutoAugment framework

AutoAugment: Search Space

Each state (policy) consists of 5 sub-policies, which in turn consist of two operations, each operation being an image processing function and the probabilities and magnitudes with which the functions are applied.

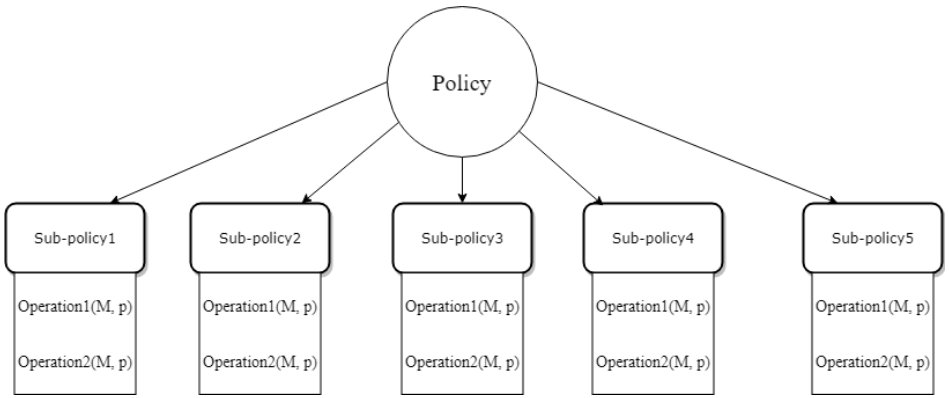


Figure: State

AutoAugment: Augmentations operations

Operation Name	Description	Range of magnitudes
ShearX(Y)	Shear the image along the horizontal (vertical) axis with rate *magnitude*.	[-0.3,0.3]
TranslateX(Y)	Translate the image in the horizontal (vertical) direction by *magnitude* number of pixels.	[-150,150]
Rotate	Rotate the image *magnitude* degrees.	[-30,30]
AutoContrast	Maximize the the image contrast, by making the darkest pixel black and lightest pixel white.	
Invert	Invert the pixels of the image.	
Equalize	Equalize the image histogram.	
Solarize	Invert all pixels above a threshold value of *magnitude*.	[0,256]
Posterize	Reduce the number of bits for each pixel to *magnitude* bits.	[4,8]
Contrast	Control the contrast of the image. A *magnitude*=0 gives a gray image, whereas *magnitude*=1 gives the original image.	[0.1,1.9]
Color	Adjust the color balance of the image, in a manner similar to the controls on a colour TV set. A *magnitude*=0 gives a black & white image, whereas *magnitude*=1 gives the original image.	[0.1,1.9]
Brightness	Adjust the brightness of the image. A *magnitude*=0 gives a black image, whereas *magnitude*=1 gives the original image.	[0.1,1.9]
Sharpness	Adjust the sharpness of the image. A *magnitude*=0 gives a blurred image, whereas *magnitude*=1 gives the original image.	[0.1,1.9]
Cutout [12, 69]	Set a random square patch of side-length *magnitude* pixels to gray.	[0,60]
Sample Pairing [24, 68]	Linearly add the image with another image (selected at random from the same mini-batch) with weight *magnitude*, without changing the label.	[0, 0.4]

Table 6. List of all image transformations that the controller could choose from during the search. Additionally, the values of magnitude that can be predicted by the controller during the search for each operation at shown in the third column (for image size 331x331). Some transformations do not use the magnitude information (e.g. Invert and Equalize).

AutoAugment: Search Space Size

- 16 operations
- Discretization of magnitude: 10 values
- Discretization of probability of operation: 11 values (uniform spacing of $[0, 1]$)

In total, the search space with 5 sub-policies has

$$(16 \times 10 \times 11)^{10} = 2.9 \times 10^{32}$$

possibilities. On each dataset, the controller samples about 15,000 policies.

AutoAugment: Search Algorithm

The authors employ a similar search algorithm as in *Learning transferable architectures for scalable image recognition* [3] (Google paper as well). It consists of 2 parts:

- A RNN controller (production of policies)
- A training algorithm (update of RNN parameters)

AutoAugment: RNN Controller

- Aim: Produce a state in the defined search space (that is, 5 sub-policies)
- Architecture: 1-layer LSTM with 100 hidden units, followed by 30 softmax outputs (5 sub-policies$\times$6 parameters per sub-policy). The output probability is fed back to the network, in order to be used in the next time step.

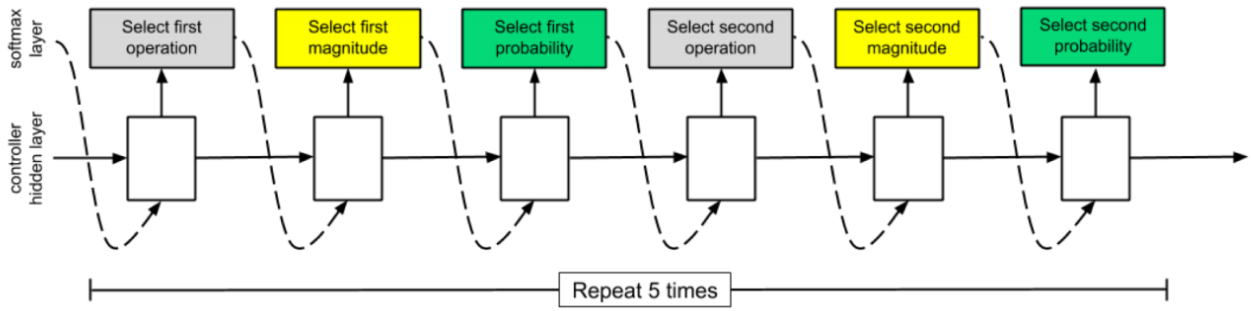


Figure: RNN Controller

disclaimer: The authors haven't published full details of the controller and the training algorithm, so we assume similar structure as in [3].

AutoAugment: Training algorithm

The training procedure seeks to maximize $\mathbb{E}_{\pi \sim p_\theta}[R(\pi)]$, where:

- π is the policy the RNN produces
- θ are the RNN's parameters
- $R(\pi)$ is the validation accuracy of the child network, measured on data augmented by policy π

Maximization $\rightarrow$ Gradient of $\mathbb{E}_{\pi \sim p_\theta}[R(\pi)]$ ✗ (R is not differentiable)
Solution: policy gradient methods ✓ (Proximal Policy Optimization, in particular)

disclaimer: The authors haven't published full details of the controller and the training algorithm, so we assume similar structure as in [3].

AutoAugment: Proximal Policy Optimization

As in every policy gradient method, we construct an objective function, whose gradient is the policy gradient estimator. The objective function to be maximized in PPO is:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{R}_t(\pi), clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{R}_t(\pi)],$$

where

- $\hat{\mathbb{E}}_t$ indicates the empirical average over a finite batch of samples at timestep t
- $r_t(\theta) = \frac{\pi_\theta}{\pi_{\theta_{old}}}$
- clip() is the clip function

The proposed algorithm to update policy parameters is the following:

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

AutoAugment: Training algorithm

At the end of the search, the 5 best policies for each dataset are saved. The final models are being trained with these 25 sub-policies.

Comment

As the authors claim, the RL algorithm is chosen out of convenience (due to the previous works of Google). It is not pivotal to the success of the proposed framework.

AutoAugment: Specific Dataset Results (small datasets)

1. CIFAR10
   - Augmentation search process on reduced dataset (8%)
   - Child model: small Wide-ResNet-40-2
   - Found: Mostly color-based transformations (Equalize, Autocontrast, Brightness)
   - Testing: Standard baseline preprocessing (standardizing the data, horizontal flips with 50% probability, zero-padding, random crops, and finally Cutout with 16x16 pxls). Learned policies are applied just before the final Cutout.
   - SOTA result with PyramidNet (deep CNN network performing a type of multiscale analysis) + ShakeDrop (residual regularization technique)
2. Reduced CIFAR10 (typically used in semi-supervised learning)
   - Significant improvement on accuracy in absence of data
3. CIFAR100
   - Experiments with the policies found from CIFAR10
   - SOTA results with PyramidNet+ShakeDrop

AutoAugment: Specific Dataset Results (big datasets)

1. SVHN (street view home numbers)
   - 600k training examples in total : 1k used in the augmentation search process
   - Same learning procedure/architecture as in CIFAR, but no final Cutout is applied.
   - Found: Different techniques than in CIFAR (Invert, ShearX/Y, Rotate). Intuitevely appealing
   - SOTA results
2. ImageNet
   - 6k samples used in augmentation search
   - Found: similar policies to CIFAR (color-based transformations)
   - Testing: Inception preprocessing followed by learned augmentation policies
   - SOTA Top 1/Top 5 accuracy with AmoebaNet-C (deep network discovered from evolutionary algorithms)

AutoAugment: Specific Dataset Results

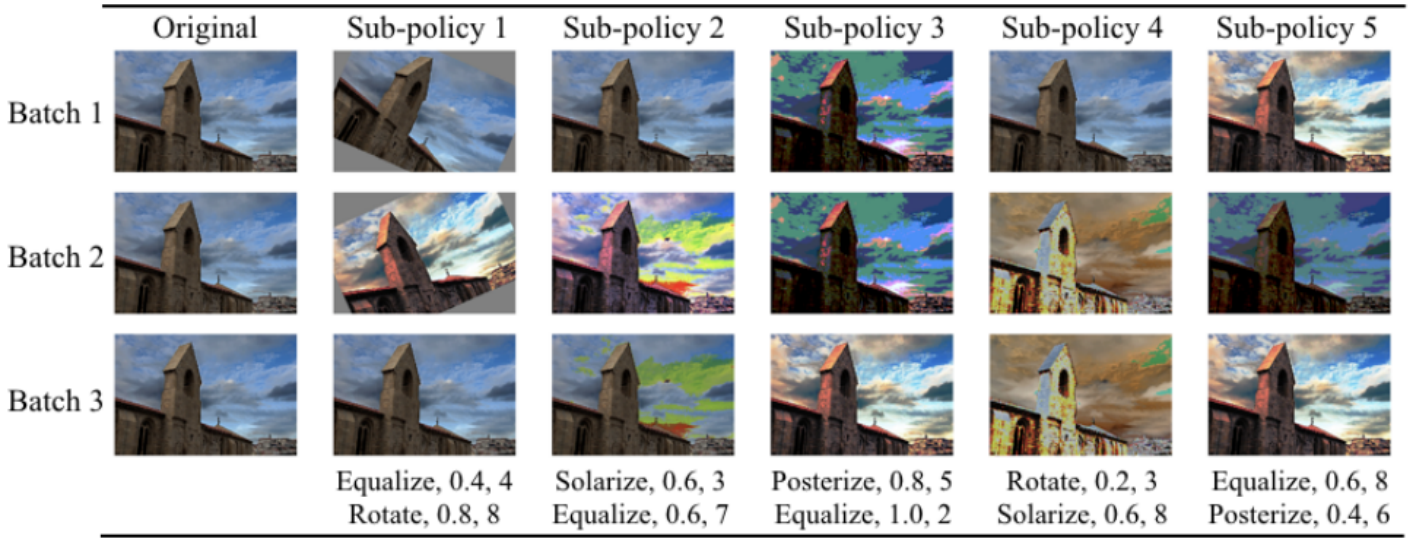


Figure 3. One of the successful policies on ImageNet. As described in the text, most of the policies found on ImageNet used color-based transformations.

Figure: ImageNet augmentation techniques

AutoAugment: Specific Dataset Results

Dataset	Model	Baseline	Cutout [12]	AutoAugment
CIFAR-10	Wide-ResNet-28-10 [67]	3.9	3.1	2.6±0.1
	Shake-Shake (26 2x32d) [17]	3.6	3.0	2.5±0.1
	Shake-Shake (26 2x96d) [17]	2.9	2.6	2.0±0.1
	Shake-Shake (26 2x112d) [17]	2.8	2.6	1.9±0.1
	AmoebaNet-B (6,128) [48]	3.0	2.1	1.8±0.1
	PyramidNet+ShakeDrop [65]	2.7	2.3	**1.5 ± 0.1**
Reduced CIFAR-10	Wide-ResNet-28-10 [67]	18.8	16.5	14.1±0.3
	Shake-Shake (26 2x96d) [17]	17.1	13.4	**10.0 ± 0.2**
CIFAR-100	Wide-ResNet-28-10 [67]	18.8	18.4	17.1±0.3
	Shake-Shake (26 2x96d) [17]	17.1	16.0	14.3±0.2
	PyramidNet+ShakeDrop [65]	14.0	12.2	**10.7 ± 0.2**
SVHN	Wide-ResNet-28-10 [67]	1.5	1.3	1.1
	Shake-Shake (26 2x96d) [17]	1.4	1.2	**1.0**
Reduced SVHN	Wide-ResNet-28-10 [67]	13.2	32.5	8.2
	Shake-Shake (26 2x96d) [17]	12.3	24.2	**5.9**

Table 2. Test set error rates (%) on CIFAR-10, CIFAR-100, and SVHN datasets. Lower is better. All the results of the baseline models, and baseline models with Cutout are replicated in our experiments and match the previously reported results [67, 17, 65, 12]. Two exceptions are Shake-Shake (26 2x112d), which has more filters than the biggest model in [17] – 112 vs 96, and Shake-Shake models trained on SVHN, these results were not previously reported. See text for more details.

Figure: CIFAR, SVHN

AutoAugment: Specific Dataset Results

Model	Inception Pre-processing [59]	AutoAugment ours
ResNet-50	76.3 / 93.1	77.6 / 93.8
ResNet-200	78.5 / 94.2	80.0 / 95.0
AmoebaNet-B (6,190)	82.2 / 96.0	82.8 / 96.2
AmoebaNet-C (6,228)	83.1 / 96.1	**83.5 / 96.5**

Table 3. Validation set Top-1 / Top-5 accuracy (%) on ImageNet. Higher is better. ResNet-50 with baseline augmentation result is taken from [20]. AmoebaNet-B,C results with Inception-style pre-processing are replicated in our experiments and match the previously reported result by [48]. There exists a better result of 85.4% Top-1 error rate [37] but their method makes use of a large amount of weakly labeled extra data. Ref. [68] reports an improvement of 1.5% for a ResNet-50 model.

Figure: ImageNet

AutoAugment: Transferability of Learned Policies

Policies found on ImageNet are being used in 5 different datasets (with similar image dimensions). It is shown that the models trained are benefited from the ImageNet policies.

Dataset	Train Size	Classes	Baseline	AutoAugment-transfer
Oxford 102 Flowers [43]	2,040	102	6.7	**4.6**
Caltech-101 [15]	3,060	102	19.4	**13.1**
Oxford-IIIT Pets [14]	3,680	37	13.5	**11.0**
FGVC Aircraft [38]	6,667	100	9.1	**7.3**
Stanford Cars [27]	8,144	196	6.4	**5.2**

Table 4. Test set Top-1 error rates (%) on FGVC datasets for Inception v4 models trained from scratch with and without AutoAugment-transfer. Lower rates are better. AutoAugment-transfer results use the policy found on ImageNet. Baseline models used Inception pre-processing.

Figure: Transferability

AutoAugment: The whole picture

Dataset	GPU hours	Best published results	Our results
CIFAR-10	5000	2.1	1.5
CIFAR-100	0	12.2	10.7
SVHN	1000	1.3	1.0
Stanford Cars	0	5.9	5.2
ImageNet	15000	3.9	3.5

Table 1. Error rates (%) from this paper compared to the best results so far on five datasets (Top-5 for ImageNet, Top-1 for the others). Previous best result on Stanford Cars fine-tuned weights originally trained on a larger dataset [66], whereas we use a randomly initialized network. Previous best results on other datasets only include models that were not trained on additional data, for a single evaluation (without ensembling). See Tables 2,3, and 4 for more detailed comparison. GPU hours are estimated for an NVIDIA Tesla P100.

Figure: AutoAugment: error rates & GPU hours

AutoAugment achieves state of the art results in a range of datasets and seems to solve the problem of automatically learned data-augmentation, but it requires huge computational resources (time/hardware). A less expensive algorithm would benefit more computer vision datasets and tasks and would not rely on the transferability of policies.

Population Based Augmentation: Main Idea

Instead of augmentation policies, search for augmentation **schedule** of policies $\rightarrow$ A special case of hyperparameter optimization. Use an evolutionary strategy to avoid training thousands of child networks.

Population Based Augmentation: Search Space

The authors try to preserve the core structure of the AutoAugment search space, in order to make the 2 frameworks directly comparable.

- 15 operations (AutoAugment's minus Sample Pairing)
- Same discretization of magnitude/probability values In total

$$(10 \times 11)^{2 \times 15} = 1.75 \times 10^{61}$$

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.

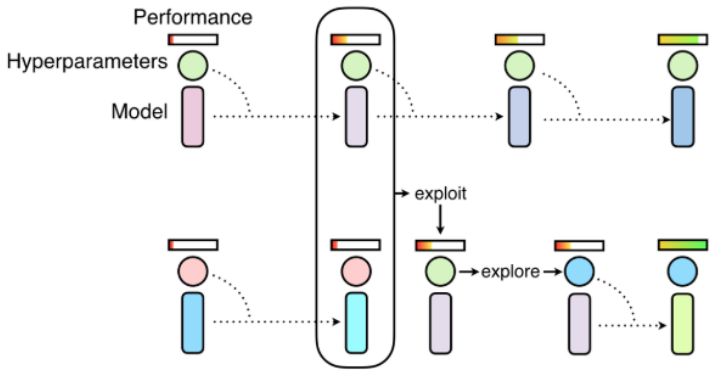
Input: data x, **parameters** p, [list of $(op, prob, mag)$]
Shuffle parameters
Set $count$ = [0, 1, 2] with probability [0.2, 0.3, 0.5]
for $(op, prob, mag)$ in p **do**
  if $count = 0$ **then**
    break
  end if
  if random(0, 1) $< prob$ **then**
    $count = count - 1$
    $x = op(x, mag)$
  end if
end for
Return x

Figure: Policy template of PBA

Population Based Augmentation: Search Algorithm

In order to find the best hyperparameters, the authors employ the recently proposed Population Based Training procedure, which is a general framework for hyperparameters tuning.

- Several workers train in parallel and in fixed intervals an evolutionary procedure happens; The worst models are being substituted by the best (elitism) and their parameters may be perturbed a little (exploration)



Overview of Population Based Training, which discovers hyperparameter schedules by training a population of neural networks. It combines random search (explore) with the copying of model weights from high performing workers (exploit). Source

Figure: PBT

Population Based Augmentation: Search Algorithm

- **Init**: Initialize 16 Wide-ResNet-40-2 models. For each model do:
- **Step**: Run an epoch of gradient descent
- **Eval**: Evaluate on validation set
- **Ready**: A model is ready to go through the exploit-and-explore phase once 3 steps/epochs have elapsed.
- **Exploit**: If model belongs to the 25% worst performing models, it clones the weights and hyperparameters of a model in the top 25% (Truncation selection)
- **Explore**: Either resample hyperparameter values from initial distribution or slightly perturb them

Computational time: $16 \times 200 = 3200 epochs \ll 15,000 \times 120$ *of AutoAugment*

Population Based Augmentation: Results

- Run PBA for a small number of epochs (~ 200) on reduced data
- Apply a linearly scaled version of the found schedule on the training process of a big network with the complete dataset

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		$\mathbf{1.46 \pm 0.077}$
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	1.18 ± 0.022
	Shake-Shake (26 2x96d)	1.40	1.20	**1.02**	1.10 ± 0.032	1.13 ± 0.029
Reduced SVHN	Wide-ResNet-28-10	13.21	32.5	8.15		7.83 ± 0.22
	Shake-Shake (26 2x96d)	13.32	24.22	**5.92**		6.46 ± 0.13

Figure: Error rate on a variety of datasets

Similar results with 500x less computational cost than AA!

AA vs PBA

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

Figure: GPU times and error rates

RandAugment

According to a recent paper from the authors of AutoAugment, even **random search** among different augmentation techniques seems to improve generalization of deep networks, which almost eliminates the computational cost of the method.

	CIFAR-10 PyramidNet		**SVHN** Wide-ResNet		**ImageNet** ResNet-50		**ImageNet** EfficientNet-B7	
	cost	acc.	cost	acc.	cost	acc.	cost	acc.
Baseline	0	97.3	0	98.5	0	76.3	0	84.0
AA	5K	98.5	1K	98.9	15K	77.6	15K	84.4
Fast AA	3.5	98.3	1.5	98.8	450	77.6	-	-
PBA	5.0	98.5	1.0	98.9	-	-	-	-
RA (ours)	0	98.5	0	99.0	0	77.6	0	85.0

Figure: Comparison of different automatic augmentation algorithms

to be continued..

References

[1] 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.

[2] Daniel Ho, Eric Liang, Ion Stoica, Pieter Abbeel, and Xi Chen.

Population based augmentation: Efficient learning of augmentation policy schedules.

arXiv preprint arXiv:1905.05393, 2019

[3] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le.

Learning transferable architectures for scalable image recognition.

In Proceedings of IEEE CVPR, 2017

[4] Jaderberg, M. et al.

Population based training of neural networks.

arXiv preprint arXiv: 1711.09846, 2017

[5] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, Quoc V. Le

RandAugment: Practical data augmentation with no separate search

arXiv preprint arXiv: 1909.13719, 2019

The End

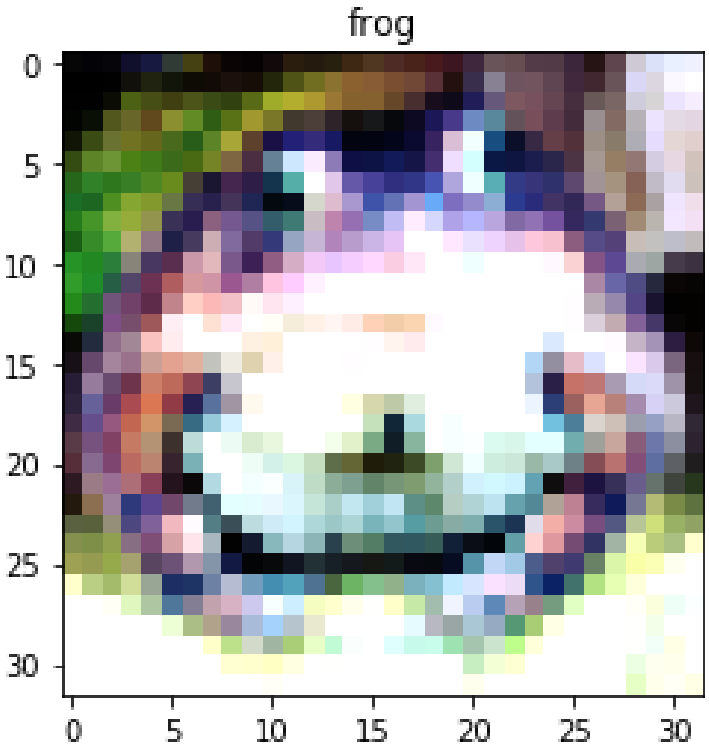


Figure: An augmented frog :)