A. Training Settings

For all setups we normalize the images by training set mean and standard deviation after the application of all augmentations, besides a final cutout, if applicable.

A.1. CIFAR

Following previous work we apply the vertical flip and the pad-and-crop augmentations and finally a 16 pixel cutout [3] after TA or generally any augmentation method. We trained Wide-ResNet models [21] in the Wide-ResNet-40-2 and the larger Wide-ResNet-28-10 settings. We trained these models for 200 epochs using SGD with Nesterov Momentum and a learning rate of 0.1, a batch size of 128, a 5e-4 weight decay, cosine learning rate decay [16].

We trained ShakeShake-26-2x96d for 1600 epochs using SGD with Nesterov Momentum, a learning rate of 0.01, a batch size of 128, 1e-3 weight decay and a cosine learning rate decay.

For the augmented batch setups we followed Zhang *et al*. [22]. We used the settings above for the Wide-ResNet-28-10 evaluations. And like Zhang slightly different settings for ShakeShake. We use 600 epochs, with a 0.2 learning rate and a 1e-4 weight decay.

A.2. SVHN

Unlike for CIFAR we do not apply extra augmentations for SVHN, besides a final 16 pixel cutout [3]. For the full dataset we trained for 160 epochs using SGD with Nesterov Momentum of 0.9, a learning rate of 0.005, a batch size of 128, a 1e-3 weight decay and cosine learning rate decay. For SVHN Core we train with the same settings, except that we trained for 200 epochs and used a larger weight decay of 5e-3.

A.3. ImageNet

Like for the other datasets we performed the standard augmentations of the dataset after the learned augmentations. That is we performed a randomly resized crop and scales between 0.08 and 1.0 using bicubic interpolation. We augmented with horizontal flips, applied a color jitter with brightness, contrast and saturation strength set to 0.4 and we applied lighting noise with an alpha of 0.1.

We trained on Imagenet with a ResNet-50 [7] and followed the setup of AA [1]. We train for 270 epochs with a batch size of 2048 distributed among 32 workers. We use image crops of height 224 considered both a 244 width of the images, like RA, and a 224 width, like AA. The initial learning rate of 0.1 is scaled proportional to the batch size divided by 256. As learning rate schedule we apply a stepwise 10-fold reduction after 90, 180 and 240 epochs with a linear warmup of factor 4 over the first 3 epochs. We use Nesterov Momenutm with a momentum parameter of 0.9 and a weight decay of 1e-4.

Unlike [1] we only use 32 instead of 64 workers out of cluster limitations and scale the learning rate accordingly.

PIL operation	range	PIL operation	range
identity	-	auto_contrast	-
equalize	-	rotate	$-30°$ - $+30°$ ($-135°$ - $+135°$)
solarize	0 - 256 (0 - 256)	color	0.1 - 1.9. (0.01 - 2.)
posterize	4 - 8 (2 - 8)	contrast	0.1 - 1.9. (0.01 - 2.)
brightness	0.1 - 1.9. (0.01 - 2.)	sharpness	0.1 - 1.9. (0.01 - 2.)
shear_x	0.0 - 0.3 0.0 - 0.99	shear_y	0.0 - 0.3 0.0 - 0.99
translate_x	0 - 10 (0 - 32)	translate_y	0 - 10 (0 - 32)
cutout	0 - 0.2	invert	-
flip_lr	-	flip_ud	-
sample_pairing	0.0 - 0.4	blur	-
smooth	-		

Table 8: An overview of the augmentation spaces. The unmarked operations are shared by all augmentation spaces and make up the RA augmentation space. The UA augmentation space additionally contains the dash underlined operations and the OHL augmentation space additionally contains the dotted underlined operations. The ranges given here are the ones used for AA and RA with a discretization to thirty values. The UA augmentation space allows translation up to 14 pixels, but inherits all other settings from RA. The Wide augmentation space we use for batch augmentation has the same operations as RA, but uses the strength ranges in parantheses. The OHL augmentation space uses different ranges and a discretization to three values, see [14] for more details. All methods are defined as part of Pillow (`https://github.com/python-pillow/Pillow`), as part of ImageEnhance, ImageOps or as image attribute, besides cutout [3]. We also provide operations with the exact same names in our code.

B. Comparison of Different Methods on the Same Augmentation Space

While in the above experiments we used the augmentation space corresponding to each method in the evaluations, in this section, we probe the impact of these differences. We follow the setup of section 4.1.2 and compare our reproduced results of each method to TA on the exact same augmentation space as in the paper introducing the respective method. Table 9 shows that TA's improvements generalize across augmentation spaces and methods.

Dataset	Setup	AA	FAA	RA	UA
CIFAR-10	Method	97.31 ± .22	97.43 ± .09	97.12 ± .14	**97.46** ± .14
	TA	**97.55** ± .06	**97.51** ± .09	**97.46** ± .09	**97.42** ± .07
CIFAR-100	Method	82.91 ± .41	**83.27** ± .13	83.1 ± .32	83.08 ± .27
	TA	**83.34** ± .10	**83.36** ± .15	**83.54** ± .12	**83.33** ± .14
SVHN	Method	97.99 ± .06	-	**98.06** ± .04	**98.05** ± .04
Core	TA	**98.04** ± .02	97.84 ± .03	**98.05** ± .02	**98.06** ± .04

Table 9: Comparisons of various methods (in our reimplementation) to TA, using the exact same augmentation space. E.g., for CIFAR-10 on the AA space, AA reached 97.31 ± .22 and TA reached 97.55 ± .06. No policy is published for FAA on SVHN Core, since this setup was not part of the FAA paper. Therefore, we do not reproduce FAA on SVHN Core.

C. Evaluation on Special Datasets

To further show that this method generalizes to more particular image classification datasets without fine-tuning, we considered two more datasets, following the settings of Section 4.1.2. (i) Since we are not aware of an image recognition dataset that contains occlusions, we created an occlusion variant of CIFAR-10 (Occ. CIFAR-10), where a 14x14 square is occluded by a black box, in each image including the test images; we evaluate a WRN-28-10 on Occ. CIFAR-10. We follow the settings for CIFAR-10 closely for this experiment. (ii) Additionally, we evaluate an RN-50 on the Stanford Cars dataset, which is a dataset in which visual details are important to distinguish car models. We train for 1000 epochs. Table 10 shows that TA continues to perform well in these settings, outperforming even brute-force tuned RA.

Method	Baseline	Transfer-RA	BF-RA	TA (RA)
Occ. CIFAR-10	94.99 ± .11	95.2 ± .08	95.52 ± .18	**95.72** ± .09
Stanford Cars	90.21 ± .16	92.47 ± .17	-	**92.77** ± .12

Table 10: A comparison on non-standard datasets. The RA setting of BruteForce-RA is searched in the same large set as in Section 4.1.2. Transfer-RA is transferred from Wide-ResNet-28-10 on CIFAR-10 and ResNet-50 on ImageNet, respectively. BF-RA for S. Cars was not feasible in the given time.

D. Evaluation on EfficientNet-B1

While we tried to evaluatate on as relevant setups as possible, we also had to make sure that we can compare with previous work for the main evaluation in Section 4.1.1. Here, we add an Evaluation of an EfficientNet-B1 following the ImageNet setup described in the original paper [18] closely. None of the methods we compare to compares on this task, thus we re-implemented UA and RA as baselines. For RA we performed a search over 3 settings for m, namely 8, 14 and 21, and fixed the number of augmentations n to 2 following the EfficientNet evaluations in [2].

	RA	UA	TA (Wide)
EfficientNet-B1	78.75 ± .16	78.83 ± .23	78.99 ± .12

Table 11: The average performance of an EfficientNet-B1 across 5 re-runs on ImageNet with different augmentation methods.

E. Approximation of the Compute Costs for Different Methods

In this section, we discuss the data underlying our performance per compute comparison. To fairly compare methods, we do not rely on published GPU times as much as possible, but instead calculate all costs for a RTX 2080 Ti for which we know many training times. Therefore, we can only compare methods for which we ran the models. That means for CIFAR-100 we consider only, the consider the Wide-ResNets as well as Shake-Shake-26-2x96d.

Our estimates for the cost of one epoch on the full CIFAR-10 dataset with 50,000 examples for each model:

- Wide-ResNet-28-2: 16s
- Wide-ResNet-40-2: 40s
- Wide-ResNet-28-10: 101s
- Shake-Shake-26-2x96d: 83s

AA In the AA paper [1] the policy is trained over 15'000 evaluations of a Wide-ResNet-40-2 on 120 epochs of 4000 examples from CIFAR-10 for all models. We therefore estimate the search cost of AA as $15000 \cdot 4000/50000 \cdot 120 \cdot 40s = 1600h$. Additionally we add the standard time for 200 epochs of standard training for each mode. Wide-ResNet-40-2: $1600h + 40s \cdot 200/60/60$, Wide-ResNet-28-10: $1600h + 101s \cdot 200/60/60$, Shake96: $1600h + 83s \cdot 1800/60/60$.

Fast AA For Fast AA [13], we estimate, based on the GPU times in the paper that the search costs more than one

full training. We therefore estimate the compute cost as one training.

UA and TA No search costs. Therefore the total cost simply is the cost of a single training. This is $\#\text{epochs} \cdot \text{costperepoch}$.

Adv. AA and TA x8 We assume for both setups no costs, even though this of course is only a lower bound on the compute requirements of Adv. AA. We simply multiply the number of epochs with the cost per epoch and 8, the number of workers.

RA The authors of RA use a search space of 5 settings each is evaluated on 90% of the full dataset with the same number of epochs and model. So we have a factor of $5 \cdot 9/10$ with which we multiply the standard costs to get the search costs.

OHL OHL uses 300 epochs for the Wide-ResNets and trains with 8 parallel workers. We therefore have a factor of $8 \cdot 300/200$ compared to standard costs for search and training combined.

AWS For AWS the data is not completely clear. First, we have an earlier version of the paper that says it evaluates 800 policies, but in a later version this was corrected down to 500. We therefore assume only 500 policy evaluations to be conservative. They used a Wide-ResNet-28-10 for the augmentation search CIFAR-100 experiments. During augmentation search they train on 80% of the training set for 200 epochs first, and then for 10 epochs for each policy evaluation. This yields $0.8 \cdot (200 + 500 \cdot 10) \cdot 101 = 117h$.

For AWS's x8 setting (8-times augmented batches), we assume the same search costs as above and 8-times the training costs.

F. Recommendations for the Application of Automatic Augmentation Methods

Based on our intense study of automatic augmentation methods for different image classification tasks using different models we recommend the following steps when applying automatic augmentation methods. In the application of an automatic augmentation method it is of course important to know, whether a method is easy to reimplement. We thus put together Table 12 for easy guidance.

Standard Model and Dataset If the model and dataset combination you are using is part of automatic augmentation literature, we recommend to simply use the best published method for your setup with published code and policies.

Novel model or Novel dataset If you are using a setup not evaluated in the automatic augmentation literature, it is a good approach to try both the best performing model on a similar task as well as a parameter-free baseline. The parameter-free baseline, like UniformAugment or TrivialAugment, especially can be expected to generalize to the new task, since they generalized to all standard automatic evluation benchmarks without any tuning. If you have tuning budget, you can of course tune something like PBA to your particular task. This likely is a good idea if your images are very dissimilar to the automatic augmentation benchmarks.

Method	Policies for Training	Code for training	Code for meta-training
Cheap Search			
TA	✓	✓	-
UA	✓	✗	-
Fast AA	✓[p]	✓	✓
Expensive Search ($> 2\times$)			
RA	✓	✗	✗
Adv. AA	✗	✗	✗
OHL	✗	✗	✗[p]
Very Expensive Search ($> 10\times$)			
PBA	✓[s]	✓[i]	✓[i]
AWS	✗	✗	✗[p]
AA	✓	✓[i]	✗

Table 12: In this table we compare the reproducibility of different methods in three categories. (i) Whether the augmentation policies used for model trainings are available, (ii) whether the authors provide code for training a model with the policies on which they report their performance and (iii) whether there is code available to run the search for training policies, code for a meta-training. We mark entries with - if it is not an applicable category for the given augmentation method and additionally use the following symbols. ✓[s]: Only available for subset of experiments, ✓[i]: Not available for ImageNet trainings, which for PBA was also not considered in the paper, ✗[p]: there is publicly work in progress.