RandAugment: Practical automated data augmentation with a reduced search space

Radek Bartyzal

GLAMI AI

29. 9. 2020

Motivation

Training data augmentation = good.

How to make it better?

- tailor the augmentations to your net + dataset
- $\implies$ training the augmentation transformation

Previous works

AutoAugment [2]

- 16 image transformation functions: f(image, magnitude)
- each has 2 parameters:
    - prob of applying the transformation (discretized 11 values)
    - magnitude of the transformation (discretized 10 values)
- goal = find 5 transformations with proper params
- use Reinforcement Learning (RL) to find them
- RL reward = validation accuracy on a proxy task
- proxy task = smaller net + subset of train dataset
- cca 15000 policies (solutions) were sampled during training

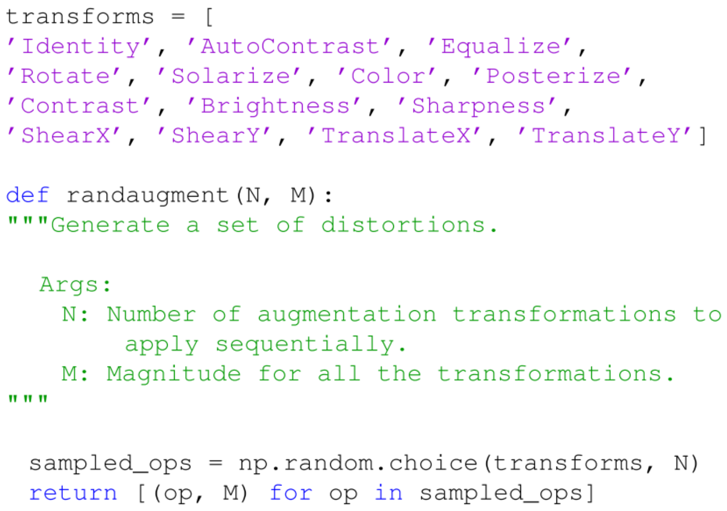
RandAugment

- 14 image transformation functions: f(image, magnitude)
- selects all image transformations with equal probability
- only 2 params:

N:	number of randomly selected transformations
M:	magnitude of all transformations

- $\implies$ can be used as hyper-params during full training
- $\implies$ no proxy task, directly optimize for the final net

RandAugment code



```
transforms = [
'Identity', 'AutoContrast', 'Equalize',
'Rotate', 'Solarize', 'Color', 'Posterize',
'Contrast', 'Brightness', 'Sharpness',
'ShearX', 'ShearY', 'TranslateX', 'TranslateY']

def randaugment(N, M):
"""Generate a set of distortions.

  Args:
    N: Number of augmentation transformations to
        apply sequentially.
    M: Magnitude for all the transformations.
"""

  sampled_ops = np.random.choice(transforms, N)
  return [(op, M) for op in sampled_ops]
```

Figure: Only 2 params: N and M.

RandAugment: example of augmented images

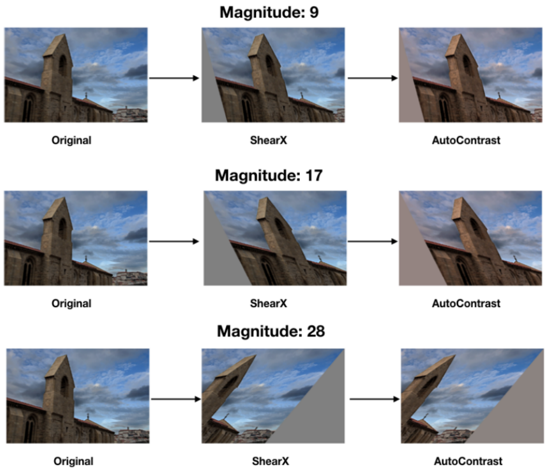


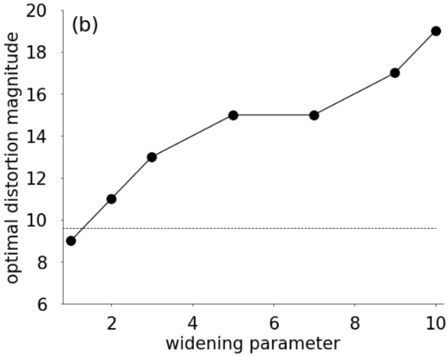
Figure: N=2 and three magnitudes are shown corresponding to the optimal distortion magnitudes for ResNet-50, EfficientNet-B5 and EfficientNet-B7.

RandAugment: justification

Is this going to be better than AutoAugment?

- obviously faster than running 15000 trainings on proxy task
- results from proxy task do not translate that well to the real task
- both of these change optimal augmentation:
    - dataset size
    - model size

Model and dataset size affect best magnitude



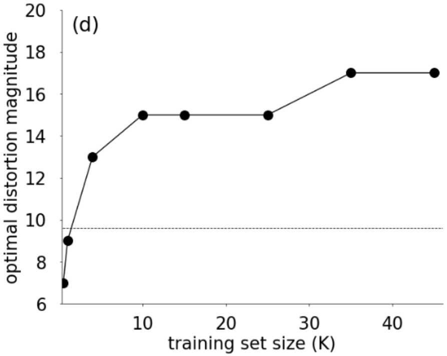
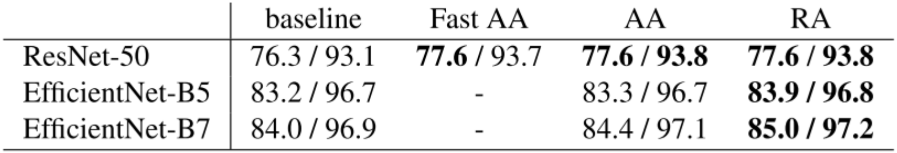


Figure: Uses CIFAR-10 validation accuracy for Wide-ResNet architectures averaged over 20 random initializations, N = 1. Dashed line = properly scaled M found by AutoAugment on proxy task.

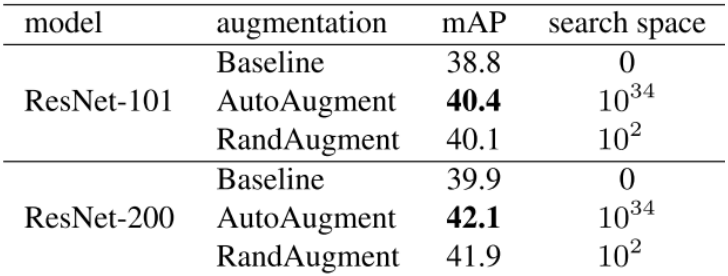
Results: ImageNet



	baseline	Fast AA	AA	RA
ResNet-50	76.3 / 93.1	**77.6** / 93.7	**77.6 / 93.8**	**77.6 / 93.8**
EfficientNet-B5	83.2 / 96.7	-	83.3 / 96.7	**83.9 / 96.8**
EfficientNet-B7	84.0 / 96.9	-	84.4 / 97.1	**85.0 / 97.2**

Figure: Top-1 and Top-5 accuracies (%) on ImageNet. Baseline, AutoAugment (AA), Fast AutoAugment (Fast AA) taken from sources (see paper).

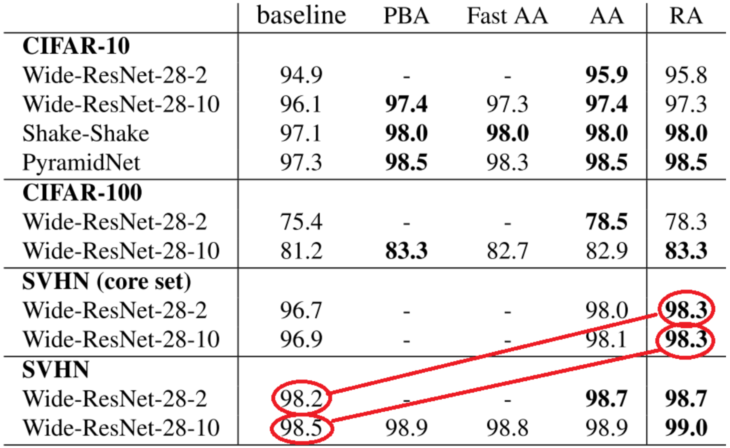
Results: COCO detection task



model	augmentation	mAP	search space
ResNet-101	Baseline	38.8	0
	AutoAugment	**40.4**	10^{34}
	RandAugment	40.1	10^2
ResNet-200	Baseline	39.9	0
	AutoAugment	**42.1**	10^{34}
	RandAugment	41.9	10^2

Figure: Models are trained for 300 epochs from random initialization. AutoAugment used extra transformations to augment the localized bounding box. AutoAugment used 15K GPU hours, where as RandAugment was tuned on 6 values.

Results: CIFAR, SVHN



	baseline	PBA	Fast AA	AA	RA
CIFAR-10					
Wide-ResNet-28-2	94.9	-	-	**95.9**	95.8
Wide-ResNet-28-10	96.1	**97.4**	97.3	**97.4**	97.3
Shake-Shake	97.1	**98.0**	**98.0**	**98.0**	**98.0**
PyramidNet	97.3	**98.5**	98.3	**98.5**	**98.5**
CIFAR-100					
Wide-ResNet-28-2	75.4	-	-	**78.5**	78.3
Wide-ResNet-28-10	81.2	**83.3**	82.7	82.9	**83.3**
SVHN (core set)					
Wide-ResNet-28-2	96.7	-	-	98.0	**98.3**
Wide-ResNet-28-10	96.9	-	-	98.1	**98.3**
SVHN					
Wide-ResNet-28-2	98.2	-	-	**98.7**	**98.7**
Wide-ResNet-28-10	98.5	98.9	98.8	98.9	**99.0**

Figure: Test accuracy (%). AA, RA results averaged over 10 independent runs. SVHN core set consists of 73K examples. Full SVHN has extra 531K easier examples to help training but RA has same acc.

Adding possible transformations improves accuracy

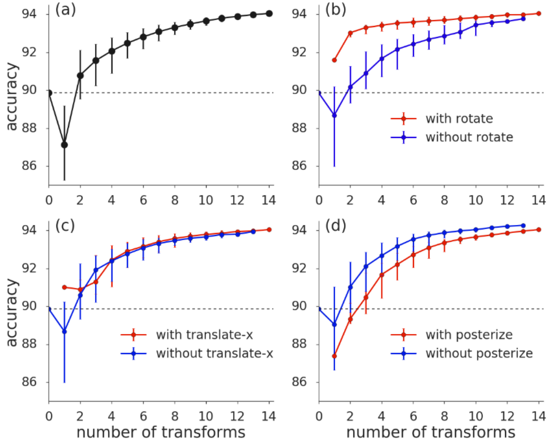


Figure: Median CIFAR-10 validation accuracy for Wide-ResNet-28-2 model architectures trained with RandAugment (N = 3, M = 4) using randomly sampled subsets of transformations. No other data augmentation is included in training. Dashed line = no augmentations. Posterize hurts accuracy.

Posterize

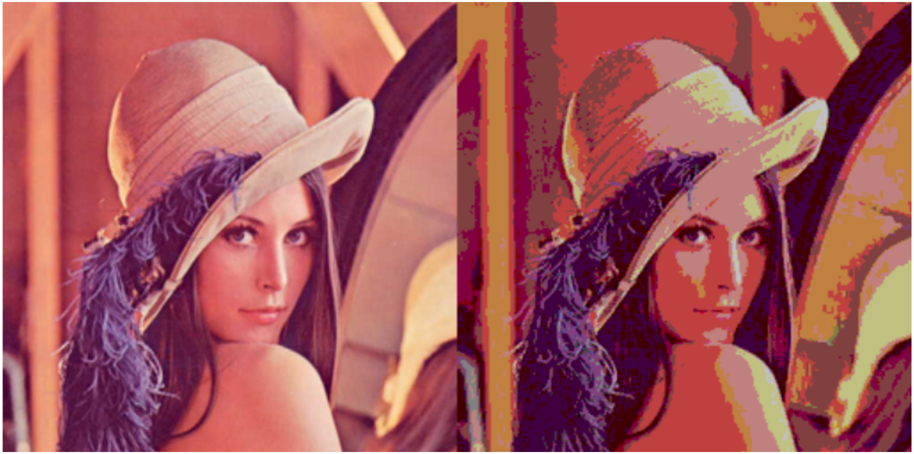
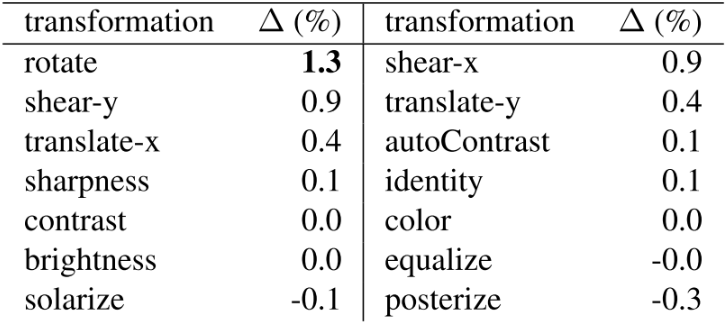


Figure: Posterize reduces number of bits per channel. Hurst model accuracy.

Individual effects of transformations



transformation	Δ (%)	transformation	Δ (%)
rotate	**1.3**	shear-x	0.9
shear-y	0.9	translate-y	0.4
translate-x	0.4	autoContrast	0.1
sharpness	0.1	identity	0.1
contrast	0.0	color	0.0
brightness	0.0	equalize	-0.0
solarize	-0.1	posterize	-0.3

Figure: Average improvement due to each transformation. Average difference in validation accuracy (%) when a particular transformation is added to a randomly sampled set of transformations. Wide-ResNet-28-2 trained on CIFAR-10 using RandAugment ($N = 3$, $M = 4$) with the randomly sampled set of transformations, with no other data augmentation.

Conclusion

Main:

- slightly better accuracy for 2 hyperparams, try:
  - $N \in \{1, 2, 3\}$
  - $M \in \{6, 12, 18, 24\}$ on uniform scale 1 - 30
- try stronger augmentation when enlarging datasets and models

Other:

- linearly increasing magnitude during training does not help $=$ magnitude $=$ M $=$ constant during training
- SOTA $+0.6\%$ on ImageNet Top 1 when used on EfficientNet-B7 [3]
- all transformations available in PIL package [4]

Sources

1. Cubuk, Ekin D., et al. "Randaugment: Practical automated data augmentation with a reduced search space." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. 2020. https://arxiv.org/abs/1909.13719

2. Cubuk, Ekin D., et al. "Autoaugment: Learning augmentation policies from data." arXiv preprint arXiv:1805.09501 (2018). https://arxiv.org/abs/1805.09501

3. SOTA ImageNet checkpoints. https://github.com/tensorflow/tpu/tree/master/models/official/efficientnet

4. PIL. https://pypi.org/project/Pillow/