Deep Learning Optimization - Neural Architecture Search

April 24, 2023

Eunhyeok Park

FBNetV3: Joint Architecture-Recipe Search using Neural Acquisition Function

Limitations of Previous Studies

1. Ignore training hyperparameters
   - Training recipes may drastically change the success or failure of an architecture

2. Support only one-time use
   - Most conventional NAS approaches produce one model for a specific set of resource constraints
   - Requires rerunning NAS once for different resource setting.

- Goal
  - Find the most accurate architecture and training recipe under given constraints
    - $\max_{a,h \in \Omega} acc(A,h)\,, subject\ to\ g_i(A) \leq C_i, for\ i = 1, \ldots, \gamma$

Step 1. Coarse-grained Search

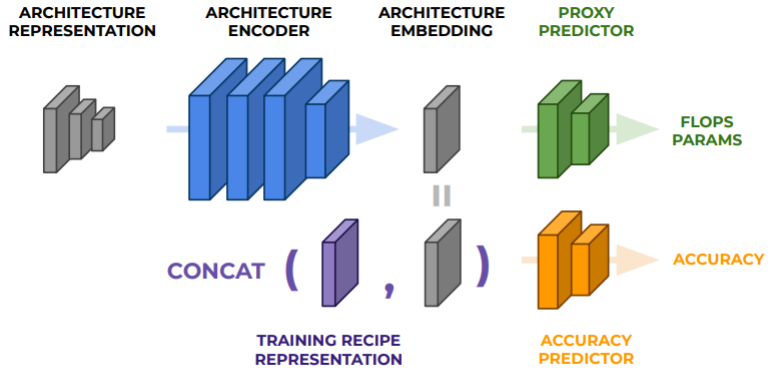


Figure 4: Architecture of the predictor.

- Constrained iterative optimization
  - **Neural Acquisition Function**
  - MLP consists of an architecture encoder and two heads
    - The proxy predictor predicts the statistics of the network (FLOPs, Params)
    - The accuracy predictor predicts the accuracy of the network

    - Step 1. pre-train embedding layer
      - Train a model that takes architecture representation as an input and predicts architecture statistics (FLOPS, Params)
      - The dataset is free, thus this training has negligible cost
    - Step 2. Constrained iterative optimization
      - Select a batch of favorable candidates based on predicted accuracy
      - Train and evaluate the candidates
      - Update the predictor

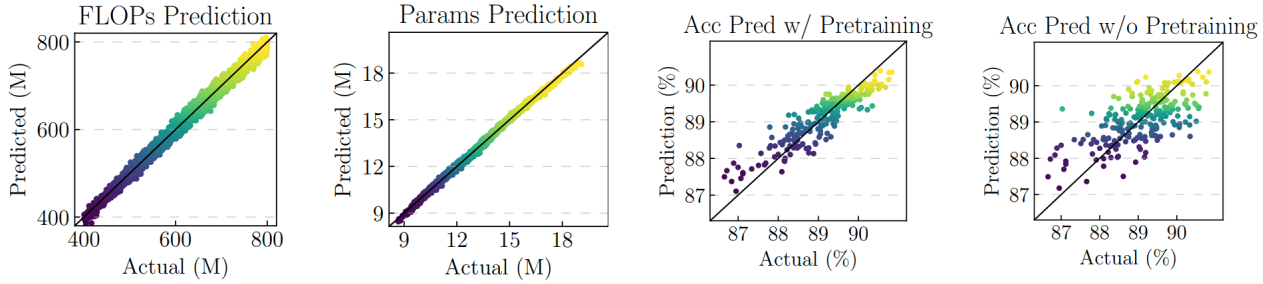


Figure 5: Predictor's performance on both proxy metrics and accuracy.

Step 2. Fine-grained Search

- Predictor-based evolutionary search
  - An iterative process based on adaptive genetic algorithms
    - The best architecture-recipe pairs from the first stage are inherited as the first generation candidates
    - In each iteration, we introduce mutations to the candidates and generate a set of children
    - Evaluate the score s for each child with the pre-trained accuracy predictor u, and select top K highest-scoring candidates for the next generation
    - Compute the gain of the highest score after each iteration, and terminate the loop when the improvement saturates
  - With the accuracy predictor, the evolution incurs a negligible cost

Search Space Design

- Architecture Configuration
  - Based on inverted residual block
    - Input resolution
    - Kernel size
    - Expansion
    - Number of channels per layer
    - Depth
- Training recipes
  - Optimizer type
  - Initial learning rate
  - Weight decay
  - Mixup ratio (augmentation)
  - Dropout ratio
  - Stochastic depth drop ratio
  - Whether or not to use model exponential moving-average

Algorithm Overview

Algorithm 1: Efficient Two-stage Constraint-aware Architecture Search

Input:
Ω: the designed search space;
n: the size of the pool Λ;
m: the number of DNN candidates ($\mathcal{X}$) to train in each iteration;
T: the number of batches for constrained iterative optimization;
Stage 1: Constrained Iterative Optimization:
Initialize $\mathcal{D}_0$ as $\emptyset$; Generate a pool Λ_t with QMC sampling subject to constraints $g_i(A) \leqslant C_i$;
for $t = 1, 2, ..., T$ **do**
    Find a batch of DNN candidates $\mathcal{X} \subset \Lambda_t$ based on predicted scores, $u(x), x \in \Lambda_t$;
    Evaluate all $x \in \mathcal{X}$ by training in parallel;
    if $t = 1$: Determine early stopping criteria;
    Update the dataset: $\mathcal{D}_t = \mathcal{D}_{t-1} \cup \{(x_1, acc(x_1)), (x_2, acc(x_2)), ...\}$;
    Retrain the accuracy predictor u on $\mathcal{D}_t$;
end
Stage 2: Predictor-based Evolutionary Search:
Initialize $\mathcal{D}^\star$ with p best-performing and q randomly generated candidates $\mathcal{X}^\star$ paired with scores predicted by u; Initialize $s^\star$ with the best score in $\mathcal{D}^\star$; set $s_0^\star = 0$; $\epsilon = 0.000001$;
while $(s^\star - s_0^\star) > \epsilon$ **do**
    for $x \in \mathcal{X}^\star$ **do**
        Generate a set of children $\mathcal{C} \subset \Omega$ subject to constraints $g_i(x) \leqslant C_i$, by the adaptive genetic algorithm [6], where $c \in \mathcal{C}$;
    end
    Augment $\mathcal{D}^\star$ with $\mathcal{C}$ paired with scores predicted by u;
    Select top K candidates from the augmented set to update $\mathcal{D}^\star$;
    Update the previous best ranking score by $s_0^\star = s^\star$;
    Update the current best ranking score $s^\star$ by the best predicted score in $\mathcal{D}^\star$.
end
Result: $\mathcal{D}^\star$, i.e., top K best samples $\mathcal{X}^\star$ with their predicted scores.

Results

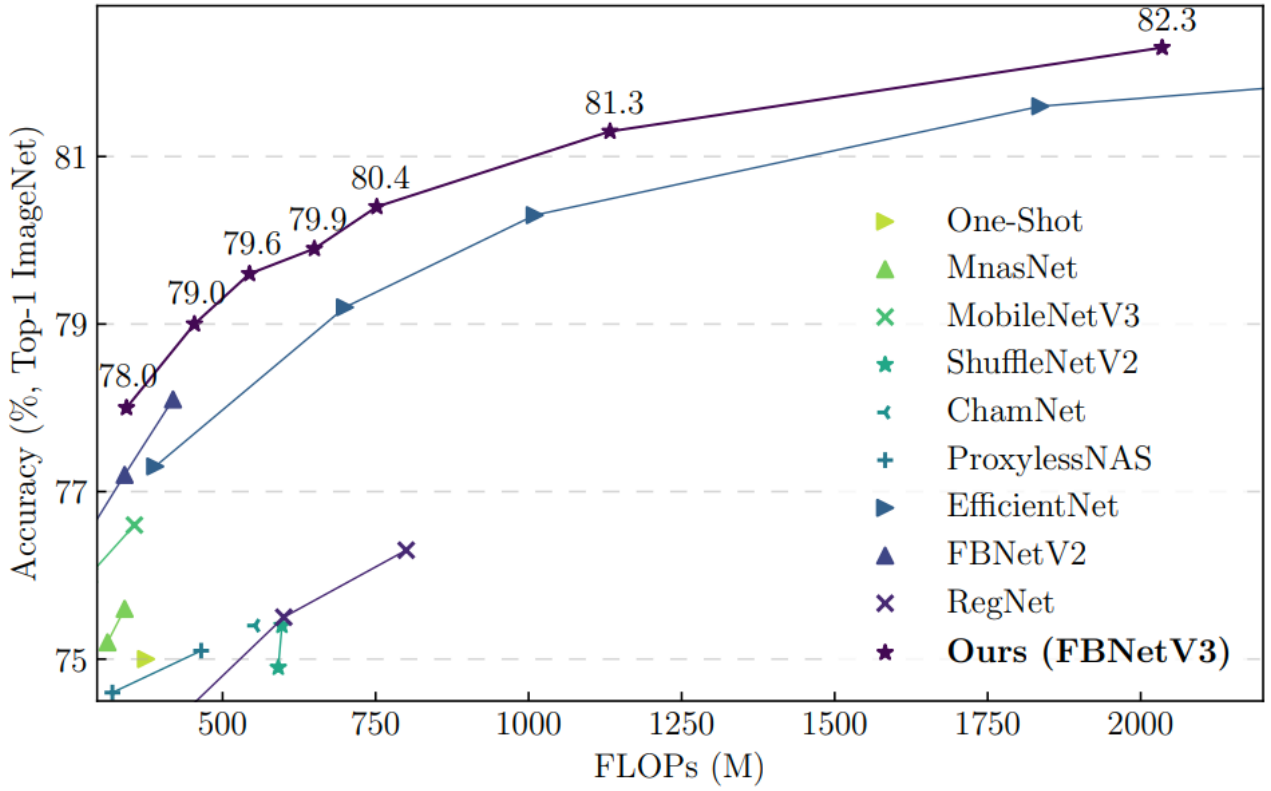


Figure 1: Comparison of FBNetV3 with other efficient Convolutional neural networks.

Model	Top-1 Accuracy (%)		
	Original	AutoTrain	Δ
FBNetV2-L3 [36]	79.1	79.9	+0.8
AlexNet [19]	56.6	62.3	+5.7
ResNet34 [11]	73.3	76.3	+3.0
ResNet50 [11]	76.1	79.2	+3.1
ResNet101 [11]	77.4	81.2	+3.8
ResNet152 [11]	78.3	81.9	+3.6
DenseNet201 [15]	77.2	80.2	+3.0
ResNeXt101 [42]	79.3	82.6	+3.3

Table 2: Accuracy improvements with the searched training recipes. Above, ResNeXt101 refers to the 32x8d variant.

Results

Model	Search method	Search space	FLOPs	Accuracy (%, Top-1 ImageNet)
FBNet [38]	gradient	arch	375M	74.9
ProxylessNAS [3]	RL/gradient	arch	465M	75.1
ChamNet [6]	predictor	arch	553M	75.4
RegNet [29]	pop. param.[1]	arch	600M	75.5
MobileNetV3-1.25x [13]	RL/NetAdapt	arch	356M	76.6
EfficientNetB0 [34]	RL/scaling	arch	390M	77.3
AtomNAS [27]	gradient	arch	363M	77.6
FBNetV2-L2 [36]	gradient	arch	423M	78.1
FBNetV3-A	**JointNAS**	**arch/training**	**343M**	**78.0**
ResNet152 [11]	manual	-	11G	78.3
EfficientNetB1 [34]	RL/scaling	arch	734M	79.2
ResNeXt101-32x8d [42]	manual	-	7.8G	79.3
FBNetV3-C	**JointNAS**	**arch/training**	**544M**	**79.6**
EfficientNetB2 [34]	RL/scaling	arch	1.0G	80.3
FBNetV3-E	**JointNAS**	**arch/training**	**752M**	**80.4**
EfficientNetB3 [34]	RL/scaling	arch	1.8G	81.7
ResNeSt-101 [48]	manual	-	10.2G	82.3
FBNetV3-G	**JointNAS**	**arch/training**	**2.0G**	**82.3**

Table 3: Comparisons of different architecture search methods. For baselines, we cite statistics on ImageNet from the original papers. Our results are bolded. 1: population parameterization.

AutoAugment:
Learning Augmentation Policies from Data

Motivation

- Data Augmentation?
  - Increase the effective diversity of data based on the random distortion

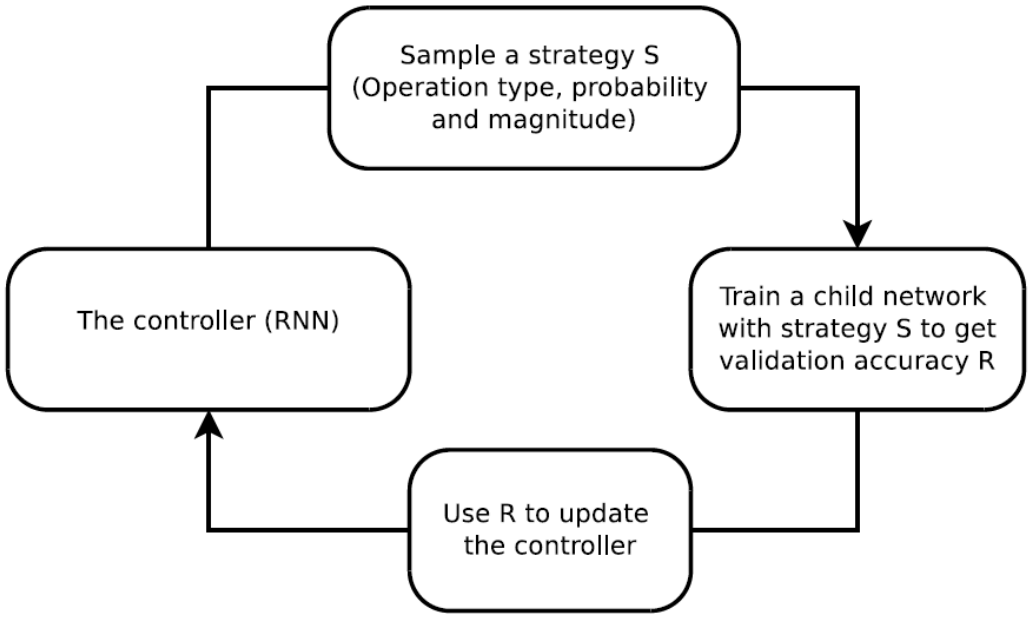
- Commonly, hand-crafted augmentation is utilized
  - The best augmentation strategies are dataset-specific.

- When augmentation improvements have been found for a particular dataset, they often do not transfer to other datasets as effectively.

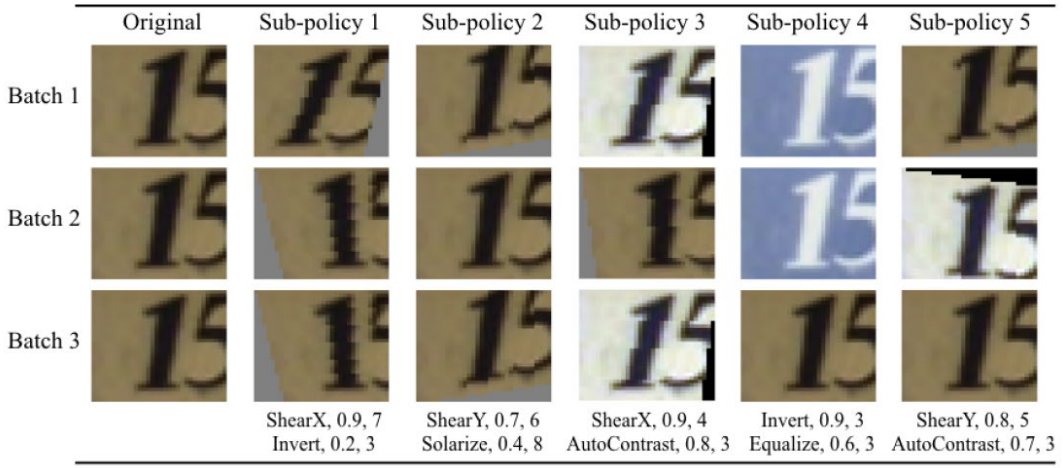
⇒An automated approach to find data augmentation policies from data.

Framework

- The search algorithm that they used in experiment uses Reinforcement Learning.
  1) A controller RNN predicts an augmentation policy from the search space.
  2) A child network with a fixed architecture is trained with 5 sub-policies
  3) The reward R is used with the policy gradient method to update the controller.



Search Space Design



- A policy consists of 5 sub-policies
  - sub-polices consists of two image operations
    - They use 16 operations from PIL
    - Each operation is associated with two hyperparameters
      - The probability of applying the operation - 11 values
      - The magnitude of the operation - 10 values

- The search space with 5 sub-policies has roughly $(16 \times 10 \times 11)^{10} \approx 2.9 \times 10^{32}$ possilities.

Search Space Design

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).

Search algorithm

- The search algorithm has two components:
  - Controller – a recurrent neural network
  - Training algorithm – Proximal Policy Optimization algorithm

At each step,

- The controller predicts a decision produced by a softmax.
  - The prediction is fed into the next step as an embedding.
  - Controller has 30 softmax predictions in order to predict 5 sub-polices.
    - 5 sub-polices * 2 operations * (type + magnitude + probability) = 30

At the end of the search,

- Concatenate the sub-policies from the best 5 policies into a single policy (with 25 sub-policies).
  - This final policy with 25 sub-policies is used to train the models for each dataset.

Experiments

1) They benchmark AutoAugment with direct search for best augmentation polices on highly competitive datasets
   - CIFAR-10, CIFAR-100, SVHN, and ImageNet datasets

2) For the transferability of augmentation policies between datasets.
   - They transfer the best augmentation policies found on ImageNet to fine-grained classification datasets
     - Oxford 102 Flowers, Caltech-101, Oxford-IIT Pets, FGVC Aircraft, Stanford Cars

Result

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.

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.

Result

- To evaluate the transferability of the policy found on ImageNet, they use the same policy that is learned on ImageNet on five FGVC datasets with image size similar to ImageNet.

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.

Conclusion

- They introduce an automated approach to find data augmentation policies from data.
  - Their method achieve state-of-the-art accuracy on CIFAR, SVHN, and ImageNet.
  - Augmentation policies they found are transferable between datasets.

- Searching augmentation policies directly on target datasets is better than transferring augmentation policies to other datasets.
  - Policies learned on data distributions closest to the target yield the best performance.

RandAugment: Practical automated data augmentation with a reduced search space

Challenges of AutoAugment

- Separate optimization procedure is required for AutoAugment
  - Increase the computation cost
  - Increase the complexity of training a machine-learning model

- The policy is found on the proxy task and transferred to the target task
  - There is a strong assumption that the proxy task provides a predictive indication of the larger task
  - Is it valid?

RandAugment Setup

- Simplifying search space, and use a grid-search to observe the effect of augmentation strength to the final accuracy
  - N: $the\ number\ of\ transformation\ for\ a\ training\ image$
  - K: $transformation\ strength$

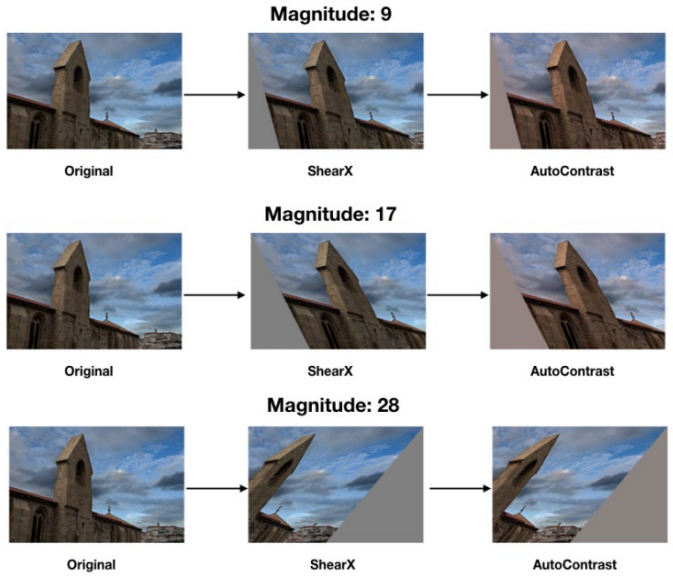


Figure 1. **Example images augmented by RandAugment.** In these examples N=2 and three magnitudes are shown corresponding to the optimal distortion magnitudes for ResNet-50, EfficientNet-B5 and EfficientNet-B7, respectively. As the distortion magnitude increases, the strength of the augmentation increases.

```
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 2. Python code for RandAugment based on numpy.

Results

	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**

	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**

ImageNet

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}

COCO Object Detection

Observation

- Network/dataset size vs augmentation strength