Bag of Tricks for Image Classification with Convolutional Neural Networks

Tong He Zhi Zhang Hang Zhang Zhongyue Zhang Junyuan Xie Mu Li

Amazon Web Services
{htong,zhiz,hzaws,zhongyue,junyuanx,mli}@amazon.com

Abstract

Much of the recent progress made in image classification research can be credited to training procedure refinements, such as changes in data augmentations and optimization methods. In the literature, however, most refinements are either briefly mentioned as implementation details or only visible in source code. In this paper, we will examine a collection of such refinements and empirically evaluate their impact on the final model accuracy through ablation study. We will show that, by combining these refinements together, we are able to improve various CNN models significantly. For example, we raise ResNet-50's top-1 validation accuracy from 75.3% to 79.29% on ImageNet. We will also demonstrate that improvement on image classification accuracy leads to better transfer learning performance in other application domains such as object detection and semantic segmentation.

1. Introduction

Since the introduction of AlexNet [15] in 2012, deep convolutional neural networks have become the dominating approach for image classification. Various new architectures have been proposed since then, including VGG [24], NiN [16], Inception [1], ResNet [9], DenseNet [13], and NASNet [35]. At the same time, we have seen a steady trend of model accuracy improvement. For example, the top-1 validation accuracy on ImageNet [23] has been raised from 62.5% (AlexNet) to 82.7% (NASNet-A).

However, these advancements did not solely come from improved model architecture. Training procedure refinements, including changes in loss functions, data preprocessing, and optimization methods also played a major role. A large number of such refinements has been proposed in the past years, but has received relatively less attention. In the literature, most were only briefly mentioned as implementation details while others can only be found in source code.

In this paper, we will examine a collection of training procedure and model architecture refinements that improve model accuracy but barely change computational complexity. Many of them are minor "tricks" like modifying the stride size of a particular convolution layer or adjusting learning rate schedule. Collectively, however, they make a big difference. We will evaluate them on multiple network architectures and datasets and report their impact to the final model accuracy.

Model	FLOPs	top-1	top-5
ResNet-50 [9]	3.9 G	75.3	92.2
ResNeXt-50 [27]	4.2 G	77.8	-
SE-ResNet-50 [12]	3.9 G	76.71	93.38
SE-ResNeXt-50 [12]	4.3 G	78.90	94.51
DenseNet-201 [13]	4.3 G	77.42	93.66
ResNet-50 + tricks (ours)	4.3 G	**79.29**	**94.63**

Table 1: **Computational costs and validation accuracy of various models.** ResNet, trained with our "tricks", is able to outperform newer and improved architectures trained with standard pipeline.

Our empirical evaluation shows that several tricks lead to significant accuracy improvement and combining them together can further boost the model accuracy. We compare ResNet-50, after applying all tricks, to other related networks in Table 1. Note that these tricks raises ResNet-50's top-1 validation accuracy from 75.3% to 79.29% on ImageNet. It also outperforms other newer and improved network architectures, such as SE-ResNeXt-50. In addition, we show that our approach can generalize to other networks (Inception V3 [1] and MobileNet [11]) and datasets (Place365 [33]). We further show that models trained with our tricks bring better transfer learning performance in other application domains such as object detection and semantic segmentation.

Paper Outline. We first set up a baseline training procedure in Section 2, and then discuss several tricks that are useful for efficient training on new hardware in Section 3. In

Algorithm 1 Train a neural network with mini-batch stochastic gradient descent.

```
initialize(net)
for epoch $= 1, \ldots, K$ do
    for batch $= 1, \ldots, \#\text{images}/b$ do
        images $\leftarrow$ uniformly random sample $b$ images
        $X, y \leftarrow$ preprocess(images)
        $z \leftarrow$ forward(net, $X$)
        $\ell \leftarrow$ loss($z$, $y$)
        grad $\leftarrow$ backward($\ell$)
        update(net, grad)
    end for
end for
```

Section 4 we review three minor model architecture tweaks for ResNet and propose a new one. Four additional training procedure refinements are then discussed in Section 5. At last, we study if these more accurate models can help transfer learning in Section 6.

Our model implementations and training scripts are publicly available in GluonCV [1] based on MXNet [3] .

2. Training Procedures

The template of training a neural network with mini-batch stochastic gradient descent is shown in Algorithm 1. In each iteration, we randomly sample b images to compute the gradients and then update the network parameters. It stops after K passes through the dataset. All functions and hyper-parameters in Algorithm 1 can be implemented in many different ways. In this section, we first specify a baseline implementation of Algorithm 1.

2.1. Baseline Training Procedure

We follow a widely used implementation [8] of ResNet as our baseline. The preprocessing pipelines between training and validation are different. During training, we perform the following steps one-by-one:

1. Randomly sample an image and decode it into 32-bit floating point raw pixel values in $[0, 255]$.
2. Randomly crop a rectangular region whose aspect ratio is randomly sampled in $[3/4, 4/3]$ and area randomly sampled in $[8\%, 100\%]$, then resize the cropped region into a 224-by-224 square image.
3. Flip horizontally with 0.5 probability.
4. Scale hue, saturation, and brightness with coefficients uniformly drawn from $[0.6, 1.4]$.
5. Add PCA noise with a coefficient sampled from a normal distribution $\mathcal{N}(0, 0.1)$.
6. Normalize RGB channels by subtracting 123.68, 116.779, 103.939 and dividing by 58.393, 57.12, 57.375, respectively.

Model	Baseline		Reference	
	Top-1	Top-5	Top-1	Top-5
ResNet-50 [9]	75.87	92.70	75.3	92.2
Inception-V3 [26]	77.32	93.43	78.8	94.4
MobileNet [11]	69.03	88.71	70.6	-

Table 2: **Validation accuracy of reference implementations and our baseline.** Note that the numbers for Inception V3 are obtained with 299-by-299 input images.

During validation, we resize each image's shorter edge to 256 pixels while keeping its aspect ratio. Next, we crop out the 224-by-224 region in the center and normalize RGB channels similar to training. We do not perform any random augmentations during validation.

The weights of both convolutional and fully-connected layers are initialized with the Xavier algorithm [6]. In particular, we set the parameter to random values uniformly drawn from $[-a, a]$, where $a = \sqrt{6/(d_{in} + d_{out})}$. Here d_{in} and d_{out} are the input and output channel sizes, respectively. All biases are initialized to 0. For batch normalization layers, γ vectors are initialized to 1 and β vectors to 0.

Nesterov Accelerated Gradient (NAG) descent [20] is used for training. Each model is trained for 120 epochs on 8 Nvidia V100 GPUs with a total batch size of 256. The learning rate is initialized to 0.1 and divided by 10 at the 30th, 60th, and 90th epochs.

2.2. Experiment Results

We evaluate three CNNs: ResNet-50 [9], Inception-V3 [1], and MobileNet [11]. For Inception-V3 we resize the input images into 299x299. We use the ISLVRC2012 [23] dataset, which has 1.3 million images for training and 1000 classes. The validation accuracies are shown in Table 2. As can be seen, our ResNet-50 results are slightly better than the reference results, while our baseline Inception-V3 and MobileNet are slightly lower in accuracy due to different training procedure.

3. Efficient Training

Hardware, especially GPUs, has been rapidly evolving in recent years. As a result, the optimal choices for many performance related trade-offs have changed. For example, it is now more efficient to use lower numerical precision and larger batch sizes during training. In this section, we review various techniques that enable low precision and large batch training without sacrificing model accuracy. Some techniques can even improve both accuracy and training speed.

[1] https://github.com/dmlc/gluon-cv

3.1. Large-batch training

Mini-batch SGD groups multiple samples to a mini-batch to increase parallelism and decrease communication costs. Using large batch size, however, may slow down the training progress. For convex problems, convergence rate decreases as batch size increases. Similar empirical results have been reported for neural networks [25]. In other words, for the same number of epochs, training with a large batch size results in a model with degraded validation accuracy compared to the ones trained with smaller batch sizes.

Multiple works [7, 14] have proposed heuristics to solve this issue. In the following paragraphs, we will examine four heuristics that help scale the batch size up for single machine training.

Linear scaling learning rate. In mini-batch SGD, gradient descending is a random process because the examples are randomly selected in each batch. Increasing the batch size does not change the expectation of the stochastic gradient but reduces its variance. In other words, a large batch size reduces the noise in the gradient, so we may increase the learning rate to make a larger progress along the opposite of the gradient direction. Goyal *et al*. [7] reports that linearly increasing the learning rate with the batch size works empirically for ResNet-50 training. In particular, if we follow He *et al*. [9] to choose 0.1 as the initial learning rate for batch size 256, then when changing to a larger batch size b, we will increase the initial learning rate to $0.1 \times b/256$.

Learning rate warmup. At the beginning of the training, all parameters are typically random values and therefore far away from the final solution. Using a too large learning rate may result in numerical instability. In the warmup heuristic, we use a small learning rate at the beginning and then switch back to the initial learning rate when the training process is stable [9]. Goyal *et al*. [7] proposes a gradual warmup strategy that increases the learning rate from 0 to the initial learning rate linearly. In other words, assume we will use the first m batches (e.g. 5 data epochs) to warm up, and the initial learning rate is η, then at batch i, $1 \leq i \leq m$, we will set the learning rate to be $i\eta/m$.

Zero γ. A ResNet network consists of multiple residual blocks, each block consists of several convolutional layers. Given input x, assume block(x) is the output for the last layer in the block, this residual block then outputs $x + \text{block}(x)$. Note that the last layer of a block could be a batch normalization (BN) layer. The BN layer first standardizes its input, denoted by $\hat{x}$, and then performs a scale transformation $\gamma\hat{x} + \beta$. Both γ and β are learnable parameters whose elements are initialized to 1s and 0s, respectively. In the zero γ initialization heuristic, we initialize $\gamma = 0$ for all BN layers that sit at the end of a residual block. Therefore, all residual blocks just return their inputs, mimics network that has less number of layers and is easier to train at the initial stage.

No bias decay. The weight decay is often applied to all learnable parameters including both weights and bias. It's equivalent to applying an L2 regularization to all parameters to drive their values towards 0. As pointed out by Jia *et al*. [14], however, it's recommended to only apply the regularization to weights to avoid overfitting. The no bias decay heuristic follows this recommendation, it only applies the weight decay to the weights in convolution and fully-connected layers. Other parameters, including the biases and γ and β in BN layers, are left unregularized.

Note that LARS [28] offers layer-wise adaptive learning rate and is reported to be effective for extremely large batch sizes (beyond 16K). While in this paper we limit ourselves to methods that are sufficient for single machine training, in which case a batch size no more than 2K often leads to good system efficiency.

3.2. Low-precision training

Neural networks are commonly trained with 32-bit floating point (FP32) precision. That is, all numbers are stored in FP32 format and both inputs and outputs of arithmetic operations are FP32 numbers as well. New hardware, however, may have enhanced arithmetic logic unit for lower precision data types. For example, the previously mentioned Nvidia V100 offers 14 TFLOPS in FP32 but over 100 TFLOPS in FP16. As in Table 3, the overall training speed is accelerated by 2 to 3 times after switching from FP32 to FP16 on V100.

Despite the performance benefit, a reduced precision has a narrower range that makes results more likely to be out-of-range and then disturb the training progress. Micikevicius *et al*. [19] proposes to store all parameters and activations in FP16 and use FP16 to compute gradients. At the same time, all parameters have an copy in FP32 for parameter updating. In addition, multiplying a scalar to the loss to better align the range of the gradient into FP16 is also a practical solution.

3.3. Experiment Results

The evaluation results for ResNet-50 are shown in Table 3. Compared to the baseline with batch size 256 and FP32, using a larger 1024 batch size and FP16 reduces the training time for ResNet-50 from 13.3-min per epoch to 4.4-min per epoch. In addition, by stacking all heuristics for large-batch training, the model trained with 1024 batch size

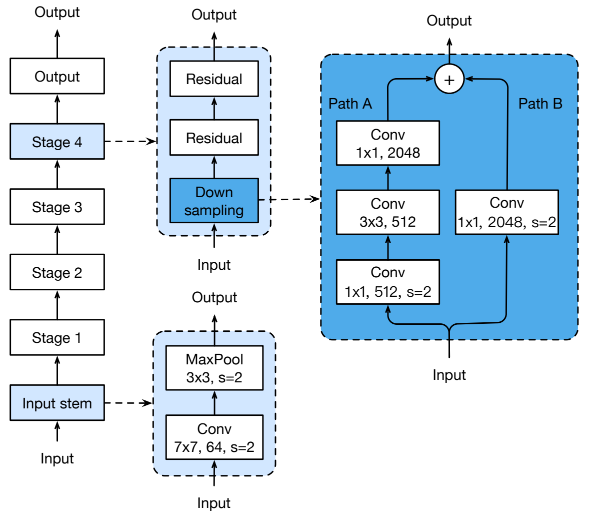


Figure 1: The architecture of ResNet-50. The convolution kernel size, output channel size and stride size (default is 1) are illustrated, similar for pooling layers.

and FP16 even slightly increased 0.5% top-1 accuracy compared to the baseline model.

The ablation study of all heuristics is shown in Table 4. Increasing batch size from 256 to 1024 by linear scaling learning rate alone leads to a 0.9% decrease of the top-1 accuracy while stacking the rest three heuristics bridges the gap. Switching from FP32 to FP16 at the end of training does not affect the accuracy.

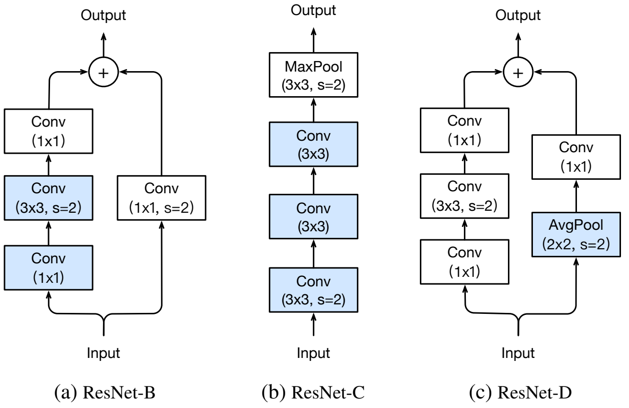
4. Model Tweaks

A model tweak is a minor adjustment to the network architecture, such as changing the stride of a particular convolution layer. Such a tweak often barely changes the computational complexity but might have a non-negligible effect on the model accuracy. In this section, we will use ResNet as an example to investigate the effects of model tweaks.

4.1. ResNet Architecture

We will briefly present the ResNet architecture, especially its modules related to the model tweaks. For detailed information please refer to He *et al*. [9]. A ResNet network consists of an input stem, four subsequent stages and a final output layer, which is illustrated in Figure 1. The input stem has a 7×7 convolution with an output channel of 64 and a stride of 2, followed by a 3×3 max pooling layer also with a stride of 2. The input stem reduces the input width and height by 4 times and increases its channel size to 64.

Starting from stage 2, each stage begins with a downsampling block, which is then followed by several residual blocks. In the downsampling block, there are path A and path B. Path A has three convolutions, whose kernel sizes are 1×1, 3×3 and 1×1, respectively. The first convolution has a stride of 2 to halve the input width and height, and the last convolution's output channel is 4 times larger than the previous two, which is called the bottleneck structure. Path B uses a 1×1 convolution with a stride of 2 to transform the input shape to be the output shape of path A, so we can sum outputs of both paths to obtain the output of the downsampling block. A residual block is similar to a downsampling block except for only using convolutions with a stride of 1.



(a) ResNet-B (b) ResNet-C (c) ResNet-D

Figure 2: Three ResNet tweaks. ResNet-B modifies the downsampling block of Resnet. ResNet-C further modifies the input stem. On top of that, ResNet-D again modifies the downsampling block.

One can vary the number of residual blocks in each stage to obtain different ResNet models, such as ResNet-50 and ResNet-152, where the number presents the number of convolutional layers in the network.

4.2. ResNet Tweaks

Next, we revisit two popular ResNet tweaks, we call them ResNet-B and ResNet-C, respectively. We propose a new model tweak ResNet-D afterwards.

ResNet-B. This tweak first appeared in a Torch implementation of ResNet [8] and then adopted by multiple works [7, 12, 27]. It changes the downsampling block of ResNet. The observation is that the convolution in path A ignores three-quarters of the input feature map because it uses a kernel size 1×1 with a stride of 2. ResNet-B switches the strides size of the first two convolutions in path A, as shown in Figure 2a, so no information is ignored. Because the second convolution has a kernel size 3×3, the output shape of path A remains unchanged.

ResNet-C. This tweak was proposed in Inception-v2 [26] originally, and it can be found on the implementations

Model	Efficient			Baseline		
	Time/epoch	Top-1	Top-5	Time/epoch	Top-1	Top-5
ResNet-50	**4.4 min**	**76.21**	**92.97**	13.3 min	75.87	92.70
Inception-V3	**8 min**	**77.50**	**93.60**	19.8 min	77.32	93.43
MobileNet	**3.7 min**	**71.90**	**90.47**	6.2 min	69.03	88.71

Table 3: Comparison of the training time and validation accuracy for ResNet-50 between the baseline (BS=256 with FP32) and a more hardware efficient setting (BS=1024 with FP16).

Heuristic	BS=256		BS=1024	
	Top-1	Top-5	Top-1	Top-5
Linear scaling	75.87	92.70	75.17	92.54
+ LR warmup	76.03	92.81	75.93	92.84
+ Zero γ	76.19	93.03	76.37	92.96
+ No bias decay	76.16	92.97	76.03	92.86
+ FP16	76.15	93.09	76.21	92.97

Table 4: The breakdown effect for each effective training heuristic on ResNet-50.

Model	#params	FLOPs	Top-1	Top-5
ResNet-50	25 M	**3.8 G**	76.21	92.97
ResNet-50-B	25 M	4.1 G	76.66	93.28
ResNet-50-C	25 M	4.3 G	76.87	93.48
ResNet-50-D	25 M	4.3 G	**77.16**	**93.52**

Table 5: Compare ResNet-50 with three model tweaks on model size, FLOPs and ImageNet validation accuracy.

of other models, such as SENet [12], PSPNet [32], DeepLabV3 [1], and ShuffleNetV2 [21]. The observation is that the computational cost of a convolution is quadratic to the kernel width or height. A 7×7 convolution is 5.4 times more expensive than a 3×3 convolution. So this tweak replacing the 7×7 convolution in the input stem with three conservative 3×3 convolutions, which is shown in Figure 2b, with the first and second convolutions have their output channel of 32 and a stride of 2, while the last convolution uses a 64 output channel.

ResNet-D. Inspired by ResNet-B, we note that the 1×1 convolution in the path B of the downsampling block also ignores 3/4 of input feature maps, we would like to modify it so no information will be ignored. Empirically, we found adding a 2×2 average pooling layer with a stride of 2 before the convolution, whose stride is changed to 1, works well in practice and impacts the computational cost little. This tweak is illustrated in Figure 2c.

4.3. Experiment Results

We evaluate ResNet-50 with the three tweaks and settings described in Section 3, namely the batch size is 1024 and precision is FP16. The results are shown in Table 5.

Suggested by the results, ResNet-B receives more information in path A of the downsampling blocks and improves validation accuracy by around 0.5% compared to ResNet-50. Replacing the 7×7 convolution with three 3×3 ones gives another 0.2% improvement. Taking more information in path B of the downsampling blocks improves the validation accuracy by another 0.3%. In total, ResNet-50-D improves ResNet-50 by 1%.

On the other hand, these four models have the same model size. ResNet-D has the largest computational cost, but its difference compared to ResNet-50 is within 15% in terms of floating point operations. In practice, we observed ResNet-50-D is only 3% slower in training throughput compared to ResNet-50.

5. Training Refinements

In this section, we will describe four training refinements that aim to further improve the model accuracy.

5.1. Cosine Learning Rate Decay

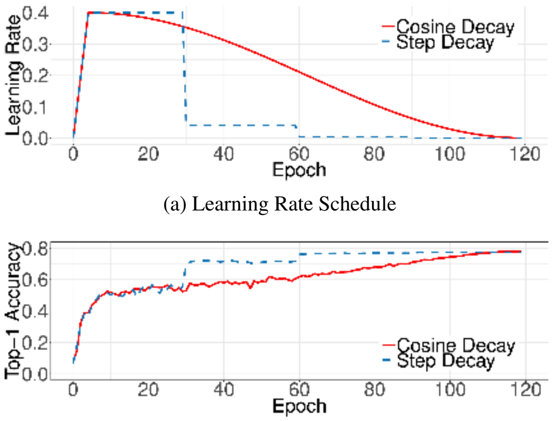
Learning rate adjustment is crucial to the training. After the learning rate warmup described in Section 3.1, we typically steadily decrease the value from the initial learning rate. The widely used strategy is exponentially decaying the learning rate. He *et al.* [9] decreases rate at 0.1 for every 30 epochs, we call it "step decay". Szegedy *et al.* [26] decreases rate at 0.94 for every two epochs.

In contrast to it, Loshchilov *et al.* [18] propose a cosine annealing strategy. An simplified version is decreasing the learning rate from the initial value to 0 by following the cosine function. Assume the total number of batches is T (the warmup stage is ignored), then at batch t, the learning rate η_t is computed as:

$$\eta_t = \frac{1}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)\eta, \tag{1}$$

where η is the initial learning rate. We call this scheduling as "cosine" decay.

The comparison between step decay and cosine decay are illustrated in Figure 3a. As can be seen, the cosine decay decreases the learning rate slowly at the beginning, and then becomes almost linear decreasing in the middle, and slows



(a) Learning Rate Schedule

(b) Validation Accuracy

Figure 3: Visualization of learning rate schedules with warm-up. Top: cosine and step schedules for batch size 1024. Bottom: Top-1 validation accuracy curve with regard to the two schedules.

down again at the end. Compared to the step decay, the cosine decay starts to decay the learning since the beginning but remains large until step decay reduces the learning rate by 10x, which potentially improves the training progress.

5.2. Label Smoothing

The last layer of a image classification network is often a fully-connected layer with a hidden size being equal to the number of labels, denote by K, to output the predicted confidence scores. Given an image, denote by z_i the predicted score for class i. These scores can be normalized by the softmax operator to obtain predicted probabilities. Denote by q the output of the softmax operator $q = \mathrm{softmax}(z)$, the probability for class i, q_i, can be computed by:

$$q_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)}. \tag{2}$$

It's easy to see $q_i > 0$ and $\sum_{i=1}^{K} q_i = 1$, so q is a valid probability distribution.

On the other hand, assume the true label of this image is y, we can construct a truth probability distribution to be $p_i = 1$ if $i = y$ and 0 otherwise. During training, we minimize the negative cross entropy loss

$$\ell(p, q) = -\sum_{i=1}^{K} p_i \log q_i \tag{3}$$

to update model parameters to make these two probability distributions similar to each other. In particular, by the way how p is constructed, we know $\ell(p, q) = -\log p_y = -z_y + \log\left(\sum_{i=1}^{K} \exp(z_i)\right)$. The optimal solution is $z_y^* = \mathrm{inf}$ while keeping others small enough. In other words, it encourages the output scores dramatically distinctive which potentially leads to overfitting.

The idea of label smoothing was first proposed to train Inception-v2 [26]. It changes the construction of the true probability to

$$q_i = \begin{cases} 1 - \varepsilon & \text{if } i = y, \\ \varepsilon/(K-1) & \text{otherwise}, \end{cases} \tag{4}$$

where ε is a small constant. Now the optimal solution becomes

$$z_i^* = \begin{cases} \log((K-1)(1-\varepsilon)/\varepsilon) + \alpha & \text{if } i = y, \\ \alpha & \text{otherwise}, \end{cases} \tag{5}$$

where α can be an arbitrary real number. This encourages a finite output from the fully-connected layer and can generalize better.

When $\varepsilon = 0$, the gap $\log((K-1)(1-\varepsilon)/\varepsilon)$ will be ∞ and as ε increases, the gap decreases. Specifically when $\varepsilon = (K-1)/K$, all optimal z_i^* will be identical. Figure 4a shows how the gap changes as we move ε, given $K = 1000$ for ImageNet dataset.

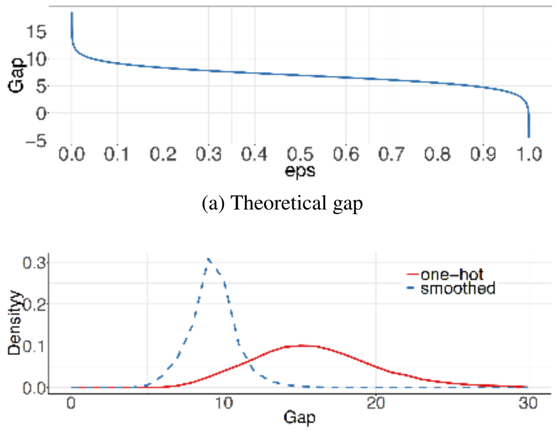
We empirically compare the output value from two ResNet-50-D models that are trained with and without label smoothing respectively and calculate the gap between the maximum prediction value and the average of the rest. Under $\varepsilon = 0.1$ and $K = 1000$, the theoretical gap is around 9.1. Figure 4b demonstrate the gap distributions from the two models predicting over the validation set of ImageNet. It is clear that with label smoothing the distribution centers at the theoretical value and has fewer extreme values.

5.3. Knowledge Distillation

In knowledge distillation [10], we use a teacher model to help train the current model, which is called the student model. The teacher model is often a pre-trained model with higher accuracy, so by imitation, the student model is able to improve its own accuracy while keeping the model complexity the same. One example is using a ResNet-152 as the teacher model to help training ResNet-50.

During training, we add a distillation loss to penalize the difference between the softmax outputs from the teacher model and the learner model. Given an input, assume p is the true probability distribution, and z and r are outputs of the last fully-connected layer of the student model and the teacher model, respectively. Remember previously we use a negative cross entropy loss $\ell(p, \mathrm{softmax}(z))$ to measure the difference between p and z, here we use the same loss again for the distillation. Therefore, the loss is changed to

$$\ell(p, \mathrm{softmax}(z)) + T^2 \ell(\mathrm{softmax}(r/T), \mathrm{softmax}(z/T)), \tag{6}$$



(a) Theoretical gap

(b) Empirical gap from ImageNet validation set

Figure 4: Visualization of the effectiveness of label smoothing on ImageNet. Top: theoretical gap between z_p^* and others decreases when increasing ε. Bottom: The empirical distributions of the gap between the maximum prediction and the average of the rest.

where T is the temperature hyper-parameter to make the softmax outputs smoother thus distill the knowledge of label distribution from teacher's prediction.

5.4. Mixup Training

In Section 2.1 we described how images are augmented before training. Here we consider another augmentation method called mixup [30]. In mixup, each time we randomly sample two examples (x_i, y_i) and (x_j, y_j). Then we form a new example by a weighted linear interpolation of these two examples:

$$\hat{x} = \lambda x_i + (1-\lambda)x_j, \quad (7)$$

$$\hat{y} = \lambda y_i + (1-\lambda)y_j, \quad (8)$$

where $\lambda \in [0, 1]$ is a random number drawn from the $\mathbf{Beta}(\alpha, \alpha)$ distribution. In mixup training, we only use the new example $(\hat{x}, \hat{y})$.

5.5. Experiment Results

Now we evaluate the four training refinements. We set $\varepsilon = 0.1$ for label smoothing by following Szegedy *et al.* [26]. For the model distillation we use $T = 20$, specifically a pretrained ResNet-152-D model with both cosine decay and label smoothing applied is used as the teacher. In the mixup training, we choose $\alpha = 0.2$ in the Beta distribution and increase the number of epochs from 120 to 200 because the mixed examples ask for a longer training progress to converge better. When combining the mixup training with distillation, we train the teacher model with mixup as well.

We demonstrate that the refinements are not only limited to ResNet architecture or the ImageNet dataset. First, we train ResNet-50-D, Inception-V3 and MobileNet on ImageNet dataset with refinements. The validation accuracies for applying these training refinements one-by-one are shown in Table 6. By stacking cosine decay, label smoothing and mixup, we have steadily improving ResNet, InceptionV3 and MobileNet models. Distillation works well on ResNet, however, it does not work well on Inception-V3 and MobileNet. Our interpretation is that the teacher model is not from the same family of the student, therefore has different distribution in the prediction, and brings negative impact to the model. In addition, we experiemented training without mixup for 200 epochs and that only results in an increment around 0.15% increment on the Top-1 accuracy, therefore the effect of mixup training is still significant.

To support our tricks is transferable to other dataset, we train a ResNet-50-D model on MIT Places365 dataset with and without the refinements. Results are reported in Table 7. We see the refinements improve the top-5 accuracy consistently on both the validation and test set.

6. Transfer Learning

Transfer learning is one major down-streaming use case of trained image classification models. In this section, we will investigate if these improvements discussed so far can benefit transfer learning. In particular, we pick two important computer vision tasks, object detection and semantic segmentation, and evaluate their performance by varying base models.

6.1. Object Detection

The goal of object detection is to locate bounding boxes of objects in an image. We evaluate performance using PASCAL VOC [4]. Similar to Ren *et al.* [22], we use union set of VOC 2007 *trainval* and VOC 2012 *trainval* for training, and VOC 2007 test for evaluation, respectively. We train Faster-RCNN [22] on this dataset, with refinements from Detectron [5] such as linear warmup and long training schedule. The VGG-19 base model in Faster-RCNN is replaced with various pretrained models in the previous discussion. We keep other settings the same so the gain is solely from the base models.

Mean average precision (mAP) results are reported in Table 8. We can observe that a base model with a higher validation accuracy leads to a higher mAP for Faster-RNN in a consistent manner. In particular, the best base model with accuracy 79.29% on ImageNet leads to the best mAP at 81.33% on VOC, which outperforms the standard model by 4%.

Refinements	ResNet-50-D		Inception-V3		MobileNet	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Efficient	77.26 (± 0.07)	93.53 (± 0.05)	77.50	93.60	71.90	90.53
+ cosine decay	77.73 (± 0.13)	93.82 (± 0.08)	78.19	94.06	72.83	91.00
+ label smoothing	78.37 (± 0.09)	94.10 (± 0.03)	78.40	94.13	72.93	91.14
+ distill w/o mixup	78.63 (± 0.04)	94.38 (± 0.03)	78.26	94.01	71.97	90.89
+ mixup w/o distill	79.01 (± 0.09)	94.57 (± 0.02)	**78.77**	**94.39**	**73.28**	**91.30**
+ distill w/ mixup	**79.33** (± 0.07)	**94.66** (± 0.03)	78.34	94.16	72.51	91.02

Table 6: The validation accuracies on ImageNet for stacking training refinements one by one. The baseline models are obtained from Section 3. We repeat each refinement on ResNet-50-D for 4 times with different initialization, and report the mean and standard deviation in the table.

Model	Val Top-1 Acc	Val Top-5 Acc	Test Top-1 Acc	Test Top-5 Acc
ResNet-50-D Efficient	56.34	86.87	57.18	87.28
ResNet-50-D Best	**56.70**	**87.33**	**57.63**	**87.82**

Table 7: Results on both the validation set and the test set of MIT Places 365 dataset. Prediction are generated as stated in Section 2.1. ResNet-50-D Efficient refers to ResNet-50-D trained with settings from Section 3, and ResNet-50-D Best further incorporate cosine scheduling, label smoothing and mixup.

Refinement	Top-1	mAP
B-standard	76.14	77.54
D-efficient	77.16	78.30
+ cosine	77.91	79.23
+ smooth	78.34	80.71
+ distill w/o mixup	78.67	80.96
+ mixup w/o distill	79.16	81.10
+ distill w/ mixup	79.29	**81.33**

Table 8: Faster-RCNN performance with various pre-trained base networks evaluated on Pascal VOC.

Refinement	Top-1	PixAcc	mIoU
B-standard	76.14	78.08	37.05
D-efficient	77.16	78.88	38.88
+ cosine	77.91	**79.25**	**39.33**
+ smooth	78.34	78.64	38.75
+ distill w/o mixup	78.67	78.97	38.90
+ mixup w/o distill	79.16	78.47	37.99
+ mixup w/ distill	79.29	78.72	38.40

Table 9: FCN performance with various base networks evaluated on ADE20K. Note that the refinement is only applied on the base network training.

6.2. Semantic Segmentation

In this section, we study how these improvement on base network would be transferable to semantic segmentation. We use Fully Convolutional Network (FCN) [17] as the baseline approach and evaluate it on the ADE20K [34] dataset. Following PSPNet [32] and Zhang *et al*. [31], we replace the base network with various pre-trained models discussed in previous sections and apply dilation network strategy [2, 29] on stage-3 and stage-4. A fully convolutional decoder is built on top of the base network to make the final prediction. We follow the practice in Zhang [31] to choose training hyper-parameters.

Both pixel accuracy (pixAcc) and mean intersection over union (mIoU) are reported in Table 9. In contradiction to our results on object detection, the base network trained with cosine learning rate schedule effectively improves the performance of the FCN, while other refinements provide inferior results. A potential explanation to the phenomenon is that semantic segmentation provides dense prediction in the pixel level. While models trained with label smoothing, distillation and mixup favor soften labels, blurred pixel-level information and degrade overall pixel-level accuracy.

7. Conclusion

In this paper, we survey a dozen tricks to train deep convolutional neural networks to improve model accuracy. These tricks introduce minor modifications to the model architecture, data preprocessing, loss function, and learning rate schedule. Our empirical results on ResNet-50, Inception-V3 and MobileNet indicate that these tricks improve model accuracy consistently. More excitingly, stacking all of them together leads to a significantly higher accuracy. In addition, these improved pre-trained models show strong advantages in transfer learning, which improve both object detection and semantic segmentation. We believe the benefits can extend to broader domains where classification base models are favored.

References

[1] L. Chen, G. Papandreou, F. Schroff, and H. Adam. Rethinking atrous convolution for semantic image segmentation. *CoRR*, abs/1706.05587, 2017. 1, 2, 5

[2] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018. 8

[3] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015. 2

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. 7

[5] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He. Detectron. https://github.com/facebookresearch/detectron, 2018. 7

[6] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010. 2

[7] P. Goyal, P. Dollár, R. B. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, and K. He. Accurate, large minibatch SGD: training imagenet in 1 hour. *CoRR*, abs/1706.02677, 2017. 3, 4

[8] S. Gross and M. Wilber. Training and investigating residual nets. http://torch.ch/blog/2016/02/04/resnets.html. 2, 4

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2, 3, 4, 5

[10] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 6

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1, 2

[12] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 7, 2017. 1, 4, 5

[13] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269. IEEE, 2017. 1

[14] X. Jia, S. Song, W. He, Y. Wang, H. Rong, F. Zhou, L. Xie, Z. Guo, Y. Yang, L. Yu, et al. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *arXiv preprint arXiv:1807.11205*, 2018. 3

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1

[16] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 1

[17] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015. 8

[18] I. Loshchilov and F. Hutter. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. 5

[19] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaev, G. Venkatesh, et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017. 3

[20] Y. E. Nesterov. A method for solving the convex programming problem with convergence rate o (1/kˆ 2). In *Dokl. Akad. Nauk SSSR*, volume 269, pages 543–547, 1983. 2

[21] H.-T. Z. Ningning Ma, Xiangyu Zhang and J. Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 2018. 5

[22] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015. 7

[23] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 1, 2

[24] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 1

[25] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017. 3

[26] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. *CoRR*, abs/1512.00567, 2015. 2, 4, 5, 6, 7

[27] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017. 1, 4

[28] Y. You, I. Gitman, and B. Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017. 3

[29] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015. 8

[30] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017. 7

[31] H. Zhang, K. Dana, J. Shi, Z. Zhang, X. Wang, A. Tyagi, and A. Agrawal. Context encoding for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018. 8

[32] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 6230–6239. IEEE, 2017. 5, 8

[33] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba. Places: A 10 million image database for scene recognition. *IEEE transactions on pattern analysis and machine intelligence*, 2017. 1

[34] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba. Scene parsing through ade20k dataset. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 8

[35] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. 1