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 [34]. 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 [32]). 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

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

initialize(net)
for epoch $= 1, \ldots, K$ **do**
  for batch $= 1, \ldots,$ #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(ℓ)
    update(net, grad)
  end for
end for

useful for efficient training on new hardware in Section 3. In 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].

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.

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

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.

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 $x +$ 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 [4] 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

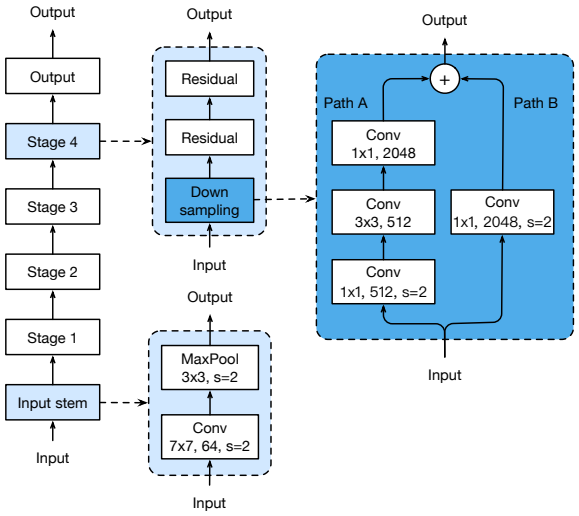


Figure 1: The architecture of ResNet-50. The convolution kernel size, output channel size and stride size (default is 1) are illustrated, similar for pooling layers.

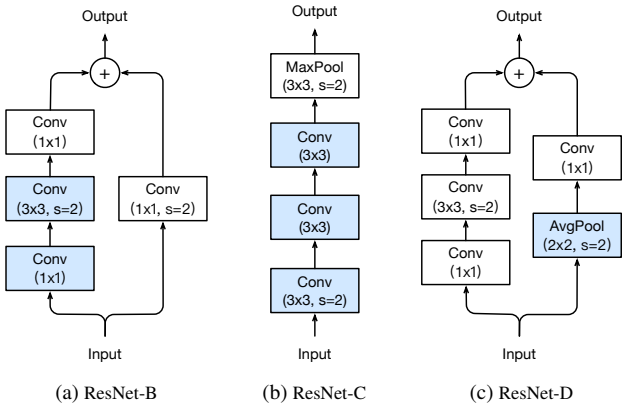


Figure 2: Three ResNet tweaks. ResNet-B modifies the downsampling block of Resnet. ResNet-C further modifies the input stem. On top of that, ResNet-D again modifies the downsampling block.

large-batch training, the model trained with 1024 batch size 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.

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.

of other models, such as SENet [12], PSPNet [31], 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%.

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.

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.

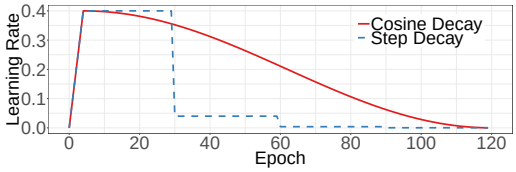
5.1. Cosine Learning Rate Decay

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.

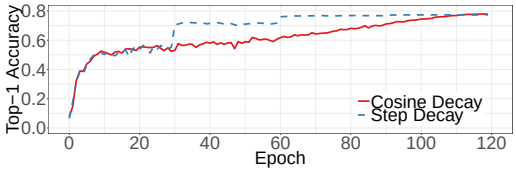
In contrast to it, Loshchilov *et al*. [18] propose a cosine annealing strategy. An simplified version is decreasing the learning rate from the initial value to 0 by following the cosine function. Assume the total number of batches is T (the warmup stage is ignored), then at batch t, the learning rate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



(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.

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 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 $q = \text{softmax}(z)$, 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

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

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 $z_y^* = \inf$ 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.

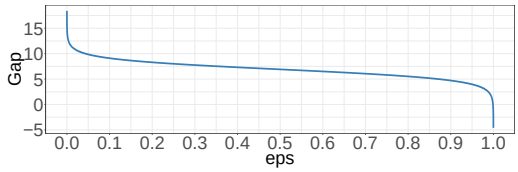
When $\varepsilon = 0$, the gap $\log((K-1)(1-\varepsilon)/\varepsilon)$ will be ∞ and as ε increases, the gap decreases. Specifically when $\varepsilon = (K-1)/K$, all optimal z_i^* will be identical. Figure 4a shows how the gap changes as we move ε, given $K = 1000$ for ImageNet dataset.

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.

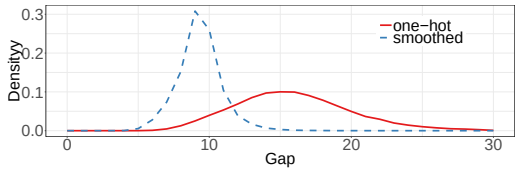
5.3. Knowledge Distillation

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



(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.

negative cross entropy loss $\ell(p, \text{softmax}(z))$ to measure the difference between p and z, here we use the same loss again for the distillation. Therefore, the loss is changed to

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

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 [29]. 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:

$$\hat{x} = \lambda x_i + (1-\lambda) x_j, \tag{7}$$
$$\hat{y} = \lambda y_i + (1-\lambda) y_j, \tag{8}$$

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.

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 [3]. 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

Refinements	ResNet-50-D		Inception-V3		MobileNet	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Efficient	77.16	93.52	77.50	93.60	71.90	90.53
+ cosine decay	77.91	93.81	78.19	94.06	72.83	91.00
+ label smoothing	78.31	94.09	78.40	94.13	72.93	91.14
+ distill w/o mixup	78.67	94.36	78.26	94.01	71.97	90.89
+ mixup w/o distill	79.15	94.58	**78.77**	**94.39**	**73.28**	**91.30**
+ distill w/ mixup	**79.29**	**94.63**	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.

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.

at 81.33% on VOC, which outperforms the standard model by 4%.

6.2. Semantic Segmentation

Semantic segmentation predicts the category for every pixel from the input images. We use Fully Convolutional Network (FCN) [17] for this task and train models on the ADE20K [33] dataset. Following PSPNet [31] and Zhang *et al*. [30], we replace the base network with various pre-trained models discussed in previous sections and apply dilation network strategy [2, 28] on stage-3 and stage-4. A fully convolutional decoder is built on top of the base network to make the final prediction.

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

[4] B. Ginsburg, I. Gitman, and Y. You. Large batch training of convolutional networks with layer-wise adaptive rate scaling. 2018. 3

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

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

[30] 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

[31] 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

[32] 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

[33] 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

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

CVPR2019 Paper Translation

姓名：余济闻

学号：2017300397

班号：10011707

针对图像分类问题的卷积神经网络训练技巧

Tong He Zhi Zhang Hang Zhang Zhongyue Zhang Junyuan Xie
Mu Li
Amazon Web Services
{htong,zhiz,hzaws,zhongyue,junyuanx,mli}@amazon.com

摘 要

在图像分类问题的研究上，最近的很多进步都归功于训练过程的改进，例如在数据增强方法和优化方法上的改变。然而在文献资料中，大多数改进方法要么只在文中介绍实现细节的部分简短地被提及，要么就只能在源代码中找到。在我们的论文中，我们将测试一系列的改进方法，通过模型简化测试的方法，实验性地评估这些改进方法对最终模型准确率的影响。我们的实验结果表明，通过组合这些改进方法，能够极大地提高各种CNN模型的性能。例如，在ImageNet数据集上验证时，我们的ResNet-50网络的top-1指标从75.3%提高到了79.29%.我们也将展示在其它应用领域，如目标检测和语义分割，提高图像分类准确率所带来的更优秀的迁移学习性能。

1. 前言

自从AlexNet [13]在2012年被使用在图像分类领域之后，深度卷积神经网络已经成为了这一领域的主流方法。在那之后各式各样的网络结构被接连提出，其中包括了VGG [23]，NiN [14]，Inception [2]，ResNet [7]，DenseNet [11]和NASNet [33]。同时，我们也看到模型的准确率呈现了一个稳定提升的趋势。例如，在ImageNet数据集上验证时，top-1指标从AlexNet模型达到的62.5%提升至NASNet-A模型达到的82.7%。

然而，这些提升不仅仅来自于模型结构的改进。对训练过程采取的改进方法也起到了很大的作用，这其中包括了对损失函数、数据预处理和优化算法三个方面上的改进。在过去几年里，大量的改进方法被提出，但是受到的关注却相对较少。在文献中，大多数对训练过程采取的改进往往要么只在介绍实现细节的时候简短地被提及，要么就只能在源代码中找到。

在我们的论文中，我们将测试一系列改进方法，这些方法针对网络训练过程以及网络模型结构进行改良，在仅仅改变计算复杂度的同时提高模型准确率。这些改进方法中有很多仅仅只是小的技巧，例如：对特定的卷积层稍微调整其卷积核的移动步长、调整学习率的设置策略。虽然这些方法并不复杂，但是他们却都可以带来极大的性能提升。我将在不同的网络结构和不同的数据集上评估这些方法，分析他们对最终的模型准确率到底有何影响。

我们的实验评估结果表明：某些训练技巧能明显提高模型准确率，如果将这些方法组合到一起使用，提高的程度也会进一步增加。在表 1中，我们比较了使用所有训练技巧训练的ResNet-50模型与其他没有使用训练技巧的模型的性能差异。从中我们可以发现，在ImageNet数据集上验证时，这些训练技巧使得ResNet-50的top-1指标从75.3%提高到了79.29%。同时，ResNet-50的性能也超过了其他得到过更多改良的新模型，例如SE-ResNeXt-50。之后我们还会说明这些训练技巧也同样适用于其他模型（Inception V3 [2]和MobileNet [9]）和其他数据集（Place365 [31]），而且使用这些训练技巧训练的模型也能在诸如目标检测和语义分割这样的其他应用领域带来更好的迁移学习性能。

论文梗概。我们首先在第 2节规定了一个作为基准的训练流程，之后在第 3节讨论几种在新硬件上有用的训练技巧。在第 4节，我们回顾了三种针对模型结构进行调整的小技巧并提出了一种新的技巧。在第 5节中还会讨

模型	FLOPs	top-1	top-5
ResNet-50	3.9G	75.3	92.2
ResNeXt-50	4.2G	77.8	-
SE-ResNet-50	3.9G	76.71	93.38
SE-ResNeXt-50	4.3G	78.90	94.51
DenseNet-201	4.3G	77.42	93.66
ResNet-50+训练技巧	4.3G	**79.29**	**94.63**

表 1: 不同模型的计算花费以及在验证集上的准确率指标。相比那些得到更多改进但没有使用训练技巧训练的新模型，用我们的训练技巧训练的ResNet模型拥有更优秀的性能。

论四种针对训练流程的改进方法。最后在第 6节，我们会研究这些使用训练技巧训练的模型是否能提高迁移学习的性能。

我们的模型实现以及训练代码是开源的，并且是基于GluonCV[1]的。

2. 训练流程

在算法 1中，我们展示了利用批量学习和随机梯度下降训练一个神经网络的模板。在每一次迭代中，我们随机地挑选b张图片去计算梯度，然后利用梯度信息来更新网络参数。在遍历整个数据集K次之后算法就会停止。算法 1中的所有函数以及超参数都可以用很多不同的方法来实现。在这个小节中，我们首先确定一个基于实现算法 1的基准示例，将这个基准作为之后我们比较的参照。

2.1. 作为基准的训练流程

我们复现了一个被广泛使用的ResNet实现示例[6]作为我们的比较基准。其中在训练部分和在验证部分所使用的预处理流程是不同的。在训练阶段，我们按照以下的步骤一步步地执行：

1. 随机选取一张图片，将初始值在$[0, 255]$范围的像素点解码成32位浮点数表示。

2. 随机裁剪一个矩形区域，区域的横纵比为$[3/4, 4/3]$范围内的随机值，区域的面积占原图的比例为$[8\%, 100\%]$范围内的随机值。之后再将裁减的区域放缩为224×224的正方形图片。

3. 按照0.5的概率对图片进行水平翻转。

4. 改变图片的色度、饱和度、亮度，改变的方法是乘以一个按照均匀分布取自$[0.6, 1.4]$的系数。

5. 加入PCA降噪，使用的系数来自于一个正态分布$\mathcal{N}(0, 0.1)$。

6. 对RGB三通道进行正规化操作，具体的做法是减去123.68、116.779、103.939，然后除以58.393、57.12、57.375。

在验证阶段，我们对图片进行尺寸放缩，在保证横纵比不变的前提下，将较短的一条边放缩到256个像素。接下来，我们从图片中间裁剪出224×224的区域并按照和训练阶段相同的方法对RGB三通道进行正规化。在验证阶段，我们没有使用任何随机的数据增强方法。

我们对卷积层和全连接层的权重都使用了Xavier算法 [4]进行初始化。我们还特意将参数设置成从均匀分布$[-a, a]$里随机挑选的值，其中$a = \sqrt{6/(d_{in} + d_{out})}$。这里的$d_{in}$和$d_{out}$分别是输入和输出的通道数。所有的偏置项都被初始化为0。对于BN层，代表方差的γ向量被初始化为1，代表均值的β向量被初始化为0。

训练阶段使用了NAG [20]（Nesterov Accelerated Gradient）方法。每一个模型都要在8个Nvidia V100 GPU上，用大小为256张图片的batch训练120轮。学习

算法 1 使用批量学习和随机梯度下降训练神经网络。

```
初始化网络
for epoch = 1, ..., K do
  for batch = 1, ..., #images/b do
    训练用的图像 ← 按正态分布随机挑选b张图像
    X, y ← 对图形进行预处理的结果
    z ← 将X通过前向网络得到的结果
    l ← 利用损失函数计算z和y之间的差距
    梯度信息 ← 对l进行反向传播的结果
    按照梯度信息对网络参数进行更新操作
  end for
end for
```

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

模型	基准结果		文献中的结果	
	Top-1	Top-5	Top-1	Top-5
ResNet-50 [7]	75.87	92.70	75.3	92.2
Inception-V3 [25]	77.32	93.43	78.8	94.4
MobileNet [9]	69.03	88.71	70.6	-

表 2: **相关文献和我们的基准测试在验证集上的准确率指标。**需要注意的是Inception-V3的数据来自于尺寸为299 × 299的输入图片。

率被初始化为0.1，并且在30、60、90轮的时候被再除以10。

2.2. 实验结果

我们总共评估了三个卷积神经网络模型的性能，其中包括ResNet-50 [7]，Inception-V3 [2]和MobileNet [9]。对于Inception-V3，我们使用了ISLVRC2012 [22]数据集，其中包含了130万张图片用于训练，总共有1000个类别。验证集上的准确率指标展示在表 2中。从中可以发现，ResNet-50的结果要略微好于文献中的结果，同时Inception-V3和MobileNet的结果由于使用了不同的训练流程而略微低于文献中的结果。

3. 高效的训练

计算机硬件设备，特别是GPU，在近几年一直在飞速的发展着。这样导致的结果就是许多提高训练指标的最优选择发生了改变，如何做出最优的选择往往涉及不同方法权衡。例如，现在在训练阶段使用较低的数值精度和较大的数据批量来训练是更有效率的。在这一小节里面，我们将回顾一些技巧，这些技巧在保证较低精度和较大的数据批量的同时并没有牺牲模型最终的准确率。一些技术甚至可以同时提高准确率和训练速度。

3.1. 大批量学习

基于批量学习的随机梯度下降算法将许多张图片组成一小批数据喂给网络进行训练，这提高了运算的并行性同时也减少了在不同训练阶段之间的通信开销。但是，使用大批量学习也会减慢训练过程。对于凸问题，当一批数据的量增加的时候，收敛速率就会减少。在神经网络上的相似的经验性结果已经被提出了 [24]。换句话说，如果两个网络训练相同的轮数的话，使用较大批量训练的模型相比使用较小批量训练的模型在验证集上的准确率会相对较差。

许多文献 [5, 12]都已经提出了一些具有启发性的方法来解决这个问题。在接下来的段落中，我们将测试四种在单机器上训练时有助于提高数据批量的方法。

线性增大学习率 在基于批量学习的随机梯度下降算法中，由于要随机选取图片来组成一笔数据，所以这是一个随机的过程。增大批量并不会改变随机梯度的期望，而是减小了它的方差。换句话说，大批量学习可以减少梯度中的噪声，因此我们可以增加学习率，以更大的步长沿着梯度方向相反的方向进行学习。Goyal等人 [5]说明了按照数据批量的大小线性地增加学习率对于ResNet-50的训练在经验上是有效果的。而且如果我们按照另一篇文献 [7]中的方法，在数据批量为256的情况下，选择0.1作为初始的学习率，那么当更批量改为更大的b时，我们也会将初始的学习率调整为$0.1 \times b/256$。

针对学习率的热身策略 在训练刚开始的时候，所有的参数都是随机的数值，因此也离最终的训练目标有远的距离。使用太大的学习率可能会造成数值变动不稳定。在一种受热身运动启发的方法中，我们在一开始使用较小的学习率，然后当训练过程稳定下来之后就切换回原先设置的初始学习率 [7]。Goyal等人 [5]提出了了一种渐变的热身策略，将学习率从0开始线性地增加到初始值。换句话说，假设我们将使用开始的m笔数据进行热身，我们的初始学习率是η，在学习第i笔数据的时候，$1 \leq i \leq m$，我们将会把学习率设置为$i\eta/m$。

“γ零初始化” 一个ResNet网络包含多个残差模块，每一个模块又包含了若干卷积层。设输入是x，block(x)是最后一个层的输出，那么这个残差模块总的输出就是$x +$ block(x)。注意一个残差模块的最后一层可能是BN层。BN层首先要标准化它的输入，结果记为$\hat{x}$，然后对其进行尺度变换，记为$\gamma\hat{x} + \beta$。其中，γ和β都是网络可以学习的参数，这些参数会在一开始的时候被分别初始化为1和0。在“γ零初始化”这个方

法中，我们将所有位于残差模块最后的BN层的γ向量设置为0。因此，所有的残差模块在开始训练的时候就像是在模拟一个层数很少的网络一样，仅仅返回自己的输入值。在初始阶段，这样的处理是有利于我们训练网络的。

不对偏置量进行衰减操作 权重衰减经常会在所有的可学习参数上被使用，包括了网络中的权重和偏置量。对所有的参数使用L2正则化来驱使他们的值接近0也是同样的道理。但是正如Jia等人[12]提出的，只在权重上使用正则化来避免过拟合才是推荐的做法。“不对偏置量进行衰减操作”的方法正是考虑了这一建议，只在卷积层和全连接层的权重上面使用权重衰减。除此之外的其他的参数，包括偏置量、BN层中的γ和β，都不会进行正则化。

需要说明的是，层级自适应比例缩放（Layer-wise Adaptive Rate Scaling, LARS） [3]提出了逐层调整学习率的方法并被证明在学习极端巨大的批量数据时（每个数据批次的数据量超过了16000）是比较有效的。在我们的论文中，我们将讨论的范围限制在那些对于单机器训练来说已经足够的方法里，在这种情况下，不超过2000的数据批量已经能带来较好的训练效率。

3.2. 低精度训练

神经网络的训练精度一般是32为浮点数（FP32）。也就是说，所有存储的数据都是按照FP32保存的，也包括了所有算术运算的输入数据与输出数据。然而在新的硬件上面，运算单元在低精度数据类型上的计算被大大优化了。例如，在我们之前提到的Nvidia V100上，提供给FP32的浮点计算速度是14TFLOPS，相比之下，提供给FP16的浮点计算速度可以达到100TFLOPS。正如表 3所展示的，将数据类型由FP32转换到FP16之后，整体的训练速度将会被加速2到3倍。

尽管上述的方法能够带来训练速度的提升，减少数据的精度也必然会导致更窄的数据表示范围，从而使得结果更加容易越界以至于妨碍了训练过程的正常进行。Micikevicius等人 [19]提出将所有的参数和激活函数的结果都以FP16的格式存储下来用以计算梯度。同时，所有的参数也保存一份FP32格式的数据用以参数更新。而且对损失函数的值乘以一个实数，使得求得的梯度的数值范围与FP16匹配，这也是一种实用的解决办法。

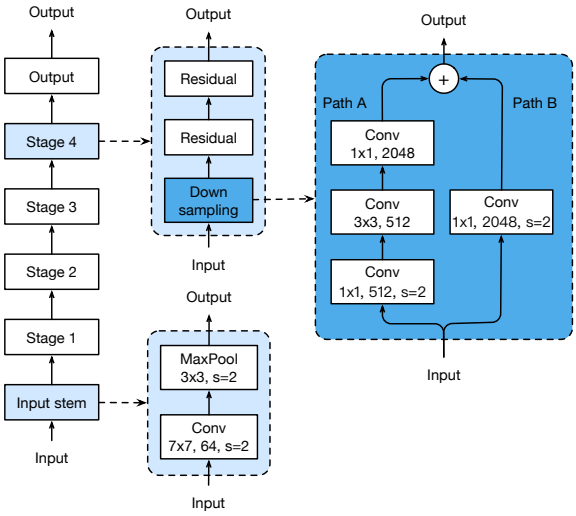


图 1: ResNet-50的网络结构。卷积层和池化层的卷积核尺寸、输出通道数以及卷积核移动步长（默认是1）都展示在图中。

3.3. 实验结果

表 3展示了ResNet-50的实验结果。相比于基准测试（数据批量为256，数据类型为FP32），使用了更大数据批量（达到1024）以及设置数据类型为FP16的优化方法将每一轮的训练时间从13.3分钟减少到了4.4分钟。而且，通过将这些优化方法叠加使用，新训练的模型甚至在top-1准确率指标相比基准测试，上取得了0.5%的微小进步。

针对所有方法的模型简化测试结果展示在表 4中。在使用了线性增大学习率的方法之后，将数据批量由256增加到1024导致了在top-1准确率上0.9%的下降。在逐个加入其他方法之后，这种差距得到了缓和。最后将数据类型由FP32转换为FP16并没有影响最终的准确率指标。

4. 模型的调整

模型微调是指对网络的结构进行微小的调整，例如改变某一特定卷积层的卷积核的移动步长。这样的微调操作经常在几乎不改变模型的计算复杂度的情况下却可能带来不可忽视的模型准确率的提升。在这一

模型	优化结果			基准结果		
	每一轮用时	Top-1	Top-5	每一轮用时	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

表 3: 对于ResNet-50模型在基准条件（数据批量大小是256，数据类型是32位浮点数）和硬件优化条件下（数据批量大小是1024，数据类型是16位浮点数）训练效果的比较。比较的指标有每一轮的训练时间和和在验证集上的准确率。

方法	数据批量为256		数据批量为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

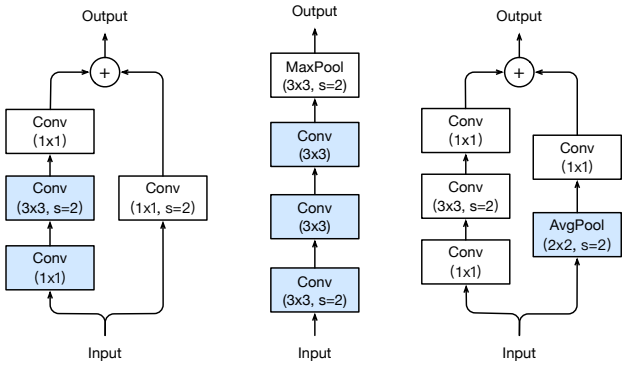
表 4: 每一种方法在ResNet-50上的训练效果。

小节中，我们将以ResNet模型为例，评估模型微调的效果。

4.1. ResNet网络结构

我们将要简短的说明一下Resnet网络结构，特别是要介绍一下与模型微调方法相关的部分。关于详细的介绍请参见He等人的论文 [7]。正如图 1中所展示的，一个ResNet的网络都包含一个处理输入的主干部分（input stem），之后会包含有四个连续的层次（stage）和最后的输出层（output layer）。在主干部分，首先是一个卷积层（输出通道数是64、卷积核移动步长是2），紧接着是一个最大池化层（移动步长也是2）。整个主干部分对输入的长宽减少四倍，将通道数增加到64。

从stage 2开始，每一个stage再开都会有一个下取样的模块，之后再紧跟若干残差模块。在下取样模块中一共由A和B两路。A路一共有三个卷积层，他们的卷积核尺寸分别是1×1、3×3和1×1。第一个卷积层将输入的长宽减半，同时最后一个卷积层的输出通道数相比前两个卷积层要多出四倍。这整个结构叫做瓶颈结构（bottleneck structure）。B路使用了一个1×1的



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

图 2: 三种针对ResNet的模型微调。其中，ResNet-B调整了网络中的下取样模块；ResNet-C调整了主干部分（input stem）；除此之外，ResNet-D进一步调整了下取样模块。

卷积层（卷积核步长为2）来保证输出的尺寸和A路的输出尺寸相同。因此，我们可以将两路的输出合并作为下取样模块的总的输出。一个残差模块与下取样模块是类似的，只是其中只使用了步长为1的卷积层。

我们可以改变每一层次的残差模块的个数，以此来得到不同的ResNet网络结构，比如ResNet-50和ResNet-152（他们名字里的数字表明了网络结构中卷积层的数目）。

4.2. ResNet网络结构的调整

接下来，我们将回顾两种很受欢迎的针对ResNet模型的微调，我们分别称之为ResNet-B和ResNet-C。然后门将提出一种新的模型微调方法，叫做ResNet-D。

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**

表 5: 比较了使用三种不同微调方法的ResNet-50网络的模型尺寸，浮点运算速度以及在ImageNet数据集上的准确率。

ResNet-B. 这种微调方法最早出现在Torch实现ResNet网络 [6]时，之后被很多研究者在工作中采用 [5, 10, 26]。它修改了ResNet中的下取样模块。我们知道在原先的A路中，由于在卷积层使用的是步长为2的1 × 1的卷积层，所以会忽略四分之三的输入特征。在ResNet-B中，将A路中的前两个卷积层的步长进行调整。如图 2a展示的那样，调整之后就没有信息被忽略掉了又因为第二个卷积层的卷积核尺寸是3 × 3，所以A路的输出尺寸也能够保持不变。

ResNet-C. 这种微调方法最早在Inception-v2的研究中 [25]被提出，现在也可以在其他的模型的实现中找到，例如SENet [10]，PSPNet [30]，DeepLabV3 [2]，和ShuffleNetV2 [18]。我们知道卷积层的计算消耗和其卷积核的尺寸的平方是正比关系。一个7 × 7的卷积层的计算消耗是一个3 × 3的卷积层的5.4倍。所以这种微调方法想到可以在主干部分用3个3 × 3的卷积层来代替7 × 7的卷积层（如图 2b所示）。其中，前两个卷积层的输出通道时是32，卷积核的移动步长是2。最后一个卷积层的输出通道数则是64。

ResNet-D. 收到ResNet-B的启发，我们注意到在下取样模块中，B路的1 × 1卷积层也同样忽略了3/4的输入特征。根据经验，我们发现将卷积层的步长调整为1之后，在其之前加入一层2 × 2的平均池化层，在基本不影响计算开销的情况下有很好的训练效果。在图 2c中展示了这种方法的详细内容。

4.3. 实验结果

我们在第 3节讨论的设定下（即数据批量为1024，数据类型为FP16），评估了ResNet-50在三种微调方法上的性能。结果展示在表 5中。

结果表明，ResNet-B在下取样模块的A路上取得了更多的信息，相比ResNet-50，在验证集上的准确率提高了大约0.5%。将7 × 7的卷积层替换为3个3 × 3的卷积层也进一步得到了0.2%的提高。在下取样模块的B路试图获取更多的信息也使得在验证集上的准确率又得到了0.3%的提高。总共ResNet-50-D在ResNet-50的基础上性能指标提高了1%。

另一方面，这四种不同的模型的模型尺寸是相同的，ResNet-D的计算消耗最大，相比ResNet-50，在浮点计算方面也只有15%的差距。在实际运行中，我们发现，相比ResNet-50，ResNet-50-D在吞吐量上仅仅只慢3%。

5. 训练的改进

在这一节中，我们将介绍四种针对训练的改进方法，用来进一步提高模型的准确率。

5.1. 余弦式衰减学习率

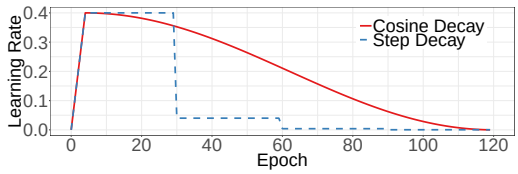
学习率的调整对于训练来说至关重要。在第 3.1节中我们讨论了针对学习率的热身策略，在这种方法里，我们一般将学习率从初始值稳定地减小。另外一种使用更广泛的策略是按照指数下降的规律来减少学习率。而在He等人的工作中 [7] ，他们每30轮就将学习率减少0.1，这被称作台阶式的衰减学习率。在Szegedy等人的工作中 [25]，他们每2轮就将学习率减少0.94。

相比之下，Loshchilov等人 [16]提出了一种余弦式退火策略。这种策略的一个简化的版本是：按照余弦函数从设定的初值开始减少学习率直到0。忽略热身阶段的话，假设数据一共有T批次，那么在第t批次的时候，学习率η_t的计算公式为：

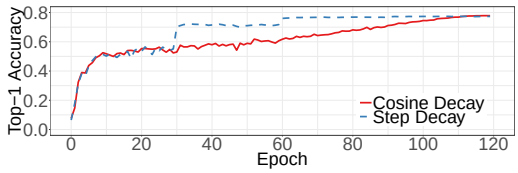
$$\eta_t = \frac{1}{2}\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)\eta, \tag{1}$$

其中η是设定的学习率初值。我们把这种调整学习率的方法叫做“余弦式衰减”。

在图 3a中展示了阶梯式衰减策略和余弦式衰减策略的比较结果。可以看出，余弦式衰减在开始阶段衰减的比较慢，在中间阶段又以线性的速度衰减，在最后又开始减慢衰减的速度。与阶梯式策略相比，余弦式衰减虽然也是在一开始就在衰减学习率，但是却可以在阶梯式策略的学习率已经减少了十倍的时候仍然



(a) Learning Rate Schedule



(b) Validation Accuracy

图 3: 使用了热身策略的两种学习衰减策略的可视化图像。上面的图中展示的是在数据批量为1024的情况下，不同学习率衰减策略对应的学习率变化曲线。下面的图片中展示的是两种策略对应的在验证集上的Top-1准确率曲线。

保持比较大的学习率，这一点潜在地提高了训练的效果。

5.2. 标签平滑

在解决图像分类问题的网络结构中，最后一层经常是一个全连接层。这一层的隐藏层的长度是等于分类问题中类别的数量的（记为K），这是为了输出代表预测结果的置信度分数。每给定一张图像，可以得到对应每一个类别的一个预测分数（第i类的分数记为z_i）。这些分数通过softmax层的正规化处理之后就可以得到预测的概率分布。得到的概率分布的公式可以记为$q = \text{softmax}(z)$，其中q代表了输出的概率的分布，对于每第i个类别的概率q_i，可以通过以下的公式进行计算：

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

容易看出$q_i > 0$和$\sum_{i=1}^{K} q_i = 1$，所以q是一个合法的概率分布。

另一方面，假设图像正确的类别是y，我们可以构造一个真实的概率分布为：当$i = y$的时候，$p_i = 1$；否则，$p_i = 0$。在训练过程中，我们最小化负交叉熵损失函数

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

来更新模型的参数，最终使得两个概率分布越来越接近。我们知道$\ell(p, q) = -\log q_y = -z_y + \log\left(\sum_{i=1}^{K} \exp(z_i)\right)$。最小化这个函数的最优解是$z_y^* = \inf$并且其他所有类别的预测分数都要足够的小。换句话说，这种损失函数的设置方法会促使输出的不同类别的分数极大的不同，从而导致潜在的过拟合结果。

标签平滑（label smoothing）的方法最早在训练Inception-v2的实验 [25]中被提出。这种方法修改了构造真实概率分布的公式为

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

其中ε是一个较小的常数。再修改之后，问题的最优解就变为

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

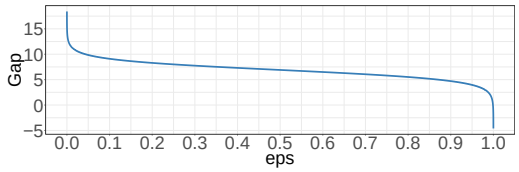
其中α可以是任意常数。这种方法导致全连接层的输出值可以是有限值，从而也就保证了训练的模型的泛化性能。

当$\varepsilon = 0$的时候，正确类别与其他错误类别之间的差距$\log((K-1)(1-\varepsilon)/\varepsilon)$将会是无限大。当$\varepsilon$增加的时候，这种差距会随之减少。当$\varepsilon = (K-1)/K$的时候，最优解中所有的$z_i^*$都将会是相同的值。图 4a展示了当移动$\varepsilon$的时候（实验在ImageNet数据集上进行，总的类别$K = 1000$），正确类别和其他错误类别之间的差距的变化情况。

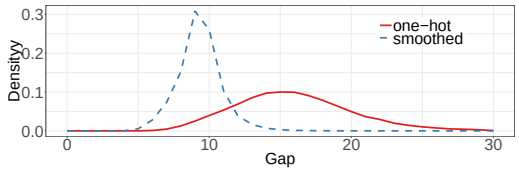
我们经验性地比较了使用和未使用标签平滑的两个ResNet-50-D模型的输出值，计算了最大预测分数和其余分数的平均值之间的差距（gap）。在$\varepsilon = 0.1$和$K = 1000$的情况下，理论的gap大约是9.1。图 4b中展示了两个模型上的gap的分布（实验在ImageNet验证集上完成）。通过结果，我们可以很明显的看出使用了标签平滑的模型的gap更加集中在理论值附近，而较少出现有极端值的情况。

改进方法	ResNet-50-D		Inception-V3		MobileNet	
	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
Efficient	77.16	93.52	77.50	93.60	71.90	90.53
+ cosine decay	77.91	93.81	78.19	94.06	72.83	91.00
+ label smoothing	78.31	94.09	78.40	94.13	72.93	91.14
+ distill w/o mixup	78.67	94.36	78.26	94.01	71.97	90.89
+ mixup w/o distill	79.15	94.58	**78.77**	**94.39**	**73.28**	**91.30**
+ distill w/ mixup	**79.29**	**94.63**	78.34	94.16	72.51	91.02

表 6: 一个接一个叠加使用改进方法的实验结果，表中性能指标是训练集上的准确率。作为基准的模型是在第 3节获得的。



(a) Theoretical gap



(b) Empirical gap from ImageNet validation set

图 4: 可视化了在imageNet数据上标签平滑方法的效果。上面的图片：z_p^*和其他类别的预测分数之间的差距随着ε的增加而逐渐减小。下面的图片：一个关于最大预测分数和其他预测分数的平均值之间的gap的概率分布。

5.3. 知识蒸馏

在知识蒸馏中 [8]，我们使用一个“教师模型”来帮助训练当前的模型，这个模型叫做“学生模型”。“教师模型”一般是一个与训练过的模型，拥有较高的准确率，所以通过模仿教师模型，学生模型能够在保持自己的模型复杂度的同时，提高它自己的准确率。一个典型例子就是可以利用ResNet-152来帮助训练ResNet-50。

在训练过程中，我们引入一个蒸馏损失函数来惩罚教师模型和学生模型之间在softmax层输出上的差异。给定一个输入，假设p是真实的概率分布，z和r分别是学生模型和教师模型最后的全连接层的输出。之前我们使用了一个负交叉熵损失函数$\ell(p, \text{softmax}(z))$来评估$p$和$z$之间的差距，在这里对蒸馏过程我们使用相同的损失函数。因此，这个损失函数应该修改为

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

其中T是一个温度超参数，用来使得softmax层的输出更加光滑，以此蒸馏出来自教师模型预测的结果的相关知识。

5.4. 数据增强方法Mixup

在第 2.1节中，我们描述了图片在训练之前是如何进行数据增强的。在这里我们将考虑另一种数据增强的方法，这种方法叫做mixup [28]。在这种方法中，每一次我们都会随机地选取两笔数据，表示为(x_i, y_i)和(x_j, y_j)。然后我们会使用下面的公式组合出一个新的数据：

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

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

其中$\lambda \in [0,1]$是一个选自$\mathbf{Beta}(\alpha, \alpha)$分布的随机数。在mixup方法中，我们只使用新的数据$(\hat{x}, \hat{y})$。

5.5. 实验结果

现在我们要评估以上介绍的四种针对训练过程的改进方法。根据Szegedy等人的论文 [25]，我们将标签

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**

表 7: 在MIT Places 365数据集上的训练结果（包括验证集上的结果和测试集上的结果）。预测结果按照第 2.1节的流程得出。ResNet-50-D Efficient一栏指的是按照第 3节设置训练的ResNet-50-D模型。ResNet-50-D Best一栏指的是进一步整合了余弦式学习率衰减、标签平滑和mixup方法训练的ResNet-50-D模型。

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**

表 8: Faster-RCNN模型在不同预训练模型上的性能（实验选用Pascal VOC数据集）。

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

表 9: ADE20K数据集上不同基础模型的全卷积网络的性能。

平滑方法中的ε设置为0.1。对于只是蒸馏方法，我们设置$T = 20$，使用一个预训练过的ResNet-152-D模型（使用了余弦式学习率衰减和标签平滑）作为教师模型。在mixup数据增强方法中，我们选择$\alpha = 0.2$作为Beta分布的参数，并提高训练的轮数在120到200轮的区间内（这样做的原因是使用mixup方法得到的数据进行训练需要更长的训练过程来达到更好的效果）。当我们将mixup方法和只是蒸馏方法组合到一起使用时，我们使用了mixup得到的数据来训练教师模型。

我们证明了这些改进方法并不仅仅只限于训练ResNet系列的模型或者在ImageNet数据上进行训练。一个接一个使用这些方法来提高验证集上准确率的实验结果展示在表 6中。通过叠加余弦式学习率衰减、标签平滑和mixup三个方法，我们可以稳定地提高ResNet模型, InceptionV3模型和MobileNet模型的性能。虽然知识蒸馏在ResNet上有很好的效果，但在Inception-V3和MobileNet上却效果不佳。我们对此的解释是这几组实验中的教师模型并不和学生模型来自同一个模型系列，因此教师模型的预测结果也来自不同的分布，这反而给模型训练带来了负面的影响。

为了证明我们的技巧是可以迁移到其他数据集上的，我们在MIT Places365数据集上训练了一个ResNet-50-D的模型（使用改进方法和不使用改进方法，这两种情况都有做）。结果展示在表 7中，我们可以看出这些改进方法确实稳定提高了在验证集和测试集上的top-5准确率。

6. 迁移学习

迁移学习是已训练过的图像分类模型的一个主要使用方向。在这一小节中，我们将研究之前讨论过的这些方法是否可以应用到迁移学习中。具体来说，我们挑选了两类重要的计算机视觉任务，目标检测和语义分割。在这两个任务的基础上，更换不同的模型来评估性能的优劣。

6.1. 目标检测

目标检测任务的目的是在图像中定位目标所在的边框。我们评估性能所使用的数据集是PASCAL VOC数据集 [17]。与Ren等人的工作 [21]类似，我们整

合了VOC 2007 *trainval*和VOC 2012 *trainval*作为训练集，将VOC 2007 test作为验证集。我们在这个数据集上使用了来自Detectron [17]一些改进方法（如热身策略和增长训练流程）训练了Faster-RCNN [21]模型。Faster-RCNN中的VGG-19模型也被替换成了其他各种之前讨论过的预训练模型。我们保持其他的设定一致，所以我们结果中性能的提高都只来自于模型更换造成的影响。

平均精度均值（Mean average precision，mAP）指标备战实在表 8中。我们可以发现一个拥有更高验证集准确率指标的模型，在Faster-RNN模型上也拥有更高的mAP指标。在ImageNet数据集上拥有最好准确率（79.29%）的模型，在VOC数据上也拥有最好的mAP指标（81.33%）。这个指标超出标准模型4个百分点。

6.2. 语义分割

语义分割任务的目标是预测图像中每一个像素的所属类别。我们在ADE20K数据上 [15]使用了全卷积网络（Fully Convolutional Network，FCN） [32]。根据PSPNet [30]和Zhang等人的工作 [29]，我们用之前讨论过的不同预训练模型替换了基础网络模型，然后在stage-3和stage-4使用了扩张网络策略（dilation network strategy） [1, 27]。一个全卷积解码器，在基础网络的基础上被构建起来用来进行最后的预测工作。

像素精度（pixel accuracy，pixAcc）和平均交除并（mean intersection over union，mIoU）两个指标都展示在表 9中。

与我们在目标检测中的结果不同的是，余弦式学习率衰减极为有效的提高了全卷积网络的准确率，其他方法带来的性能提高都比其要略低。对这个现象的一个可能的解释是语义分割在像素一级进行预测，当在使用标签平滑、知识蒸馏和mixup方法训练模型时，标签被软化，模糊的像素级信息可能会被进一步模糊，然后对整体的像素级精度造成负面影响。

7. 结论

为了提高模型的精度，本文研究了十几种训练深层卷积神经网络的技巧。这些技巧引入了对模型结构、数据预处理、损失函数和学习率策略的小调整。我们在ResNet-50、Inception-V3和MobileNet上的经验结果表明，这些技巧一致地提高了模型的准确率。更令人兴奋的是，将所有这些方法叠加在一起可以大大提高准确率。此外，这些改进了的预训练模型在迁移学习方面具有很强的优势，既能提高目标检测任务的结果，也能提高语义分割任务的结果。我们相信，这些好处可以扩展到更广泛的领域，在这些领域中，分类模型将会受到更多青睐。

参考文献

[1] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:834–848, 2016.

[2] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

[3] Boris Ginsburg, Igor Gitman, and Yang You. Large batch training of convolutional networks with layer-wise adaptive rate scaling. 2018.

[4] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[5] Priya Goyal, Piotr Dollár, Ross B. Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *ArXiv*, abs/1706.02677, 2017.

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

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[8] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[10] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2017.

[11] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks.

In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[12] Xianyan Jia, Shutao Song, Wei He, Yangzihao Wang, Haidong Rong, Feihu Zhou, Liqiang Xie, Zhenyu Guo, Yuanzhou Yang, Liwei Yu, Tiegang Chen, Guangxiao Hu, Shaohuai Shi, and Xiaowen Chu. Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes. *ArXiv*, abs/1807.11205, 2018.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[14] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network, 2013.

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015.

[16] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with restarts. *ArXiv*, abs/1608.03983, 2016.

[17] C. K. I. Williams J. Winn M. Everingham, L. Van Gool and A. Zisserman. The pascal visual object classes challenge 2007 (voc2007) results. http://www.pascalnetwork.org/challenges/VOC/voc2007/workshop/index.html.

[18] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design, 2018.

[19] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2017.

[20] Yu. E. Nesterov. A method for solving the convex programming problem with convergence rate $o(1/k^2)$. 1983.

[21] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39:1137–1149, 2015.

[22] Olga Russakovsky, Jun Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115:211–252, 2014.

[23] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[24] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V. Le. Don't decay the learning rate, increase the batch size. *ArXiv*, abs/1711.00489, 2017.

[25] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2015.

[26] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5987–5995, 2016.

[27] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions, 2015.

[28] Hongyi Zhang, Moustapha Cissé, Yann Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *ArXiv*, abs/1710.09412, 2017.

[29] Hang Zhang, Kristin J. Dana, Jianping Shi, Zhongyue Zhang, Xiaogang Wang, Ambrish Tyagi, and Amit Agrawal. Context encoding for semantic segmentation. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7151–7160, 2018.

[30] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6230–6239, 2016.

[31] 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*, 40(6):1452–1464, June 2018.

[32] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5122–5130, 2017.

[33] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.