Deep Networks with Stochastic Depth

Gao Huang*, Yu Sun*, Zhuang Liu†, Daniel Sedra, Kilian Q. Weinberger

* Authors contribute equally
{gh349, ys646, dms422, kqw4}@cornell.edu, Cornell University
† liuzhuang13@mails.tsinghua.edu.cn, Tsinghua University

Abstract. Very deep convolutional networks with hundreds of layers have led to significant reductions in error on competitive benchmarks. Although the unmatched expressiveness of the many layers can be highly desirable at test time, training very deep networks comes with its own set of challenges. The gradients can vanish, the forward flow often diminishes, and the training time can be painfully slow. To address these problems, we propose *stochastic depth*, a training procedure that enables the seemingly contradictory setup to *train short* networks and *use deep* networks at test time. We start with very deep networks but during training, for each mini-batch, randomly drop a subset of layers and bypass them with the identity function. This simple approach complements the recent success of residual networks. It reduces training time substantially and improves the test error significantly on almost all data sets that we used for evaluation. With stochastic depth we can increase the depth of residual networks even beyond 1200 layers and still yield meaningful improvements in test error (4.91% on CIFAR-10).

1 Introduction

Convolutional Neural Networks (CNNs) were arguably popularized within the vision community in 2009 through AlexNet [1] and its celebrated victory at the ImageNet competition [2]. Since then there has been a notable shift towards CNNs in many areas of computer vision [3, 4, 5, 6, 7, 8]. As this shift unfolds, a second trend emerges; deeper and deeper CNN architectures are being developed and trained. Whereas AlexNet had 5 convolutional layers [1], the VGG network and GoogLeNet in 2014 had 19 and 22 layers respectively [5, 7], and most recently the ResNet architecture featured 152 layers [8].

Network depth is a major determinant of model expressiveness, both in theory [9, 10] and in practice [5, 7, 8]. However, very deep models also introduce new challenges: vanishing gradients in backward propagation, diminishing feature reuse in forward propagation, and long training time.

Vanishing Gradients is a well known nuisance in neural networks with many layers [11]. As the gradient information is back-propagated, repeated multiplication or convolution with small weights renders the gradient information ineffectively small in earlier layers. Several approaches exist to reduce this effect in practice, for example through careful initialization [12], hidden layer supervision [13], or, recently, Batch Normalization [14].

Diminishing feature reuse during forward propagation (also known as loss in information flow [15]) refers to the analogous problem to vanishing gradients in the forward direction. The features of the input instance, or those computed by earlier layers, are "washed out" through repeated multiplication or convolution with (randomly initialized) weight matrices, making it hard for later layers to identify and learn "meaningful" gradient directions. Recently, several new architectures attempt to circumvent this problem through direct identity mappings between layers, which allow the network to pass on features unimpededly from earlier layers to later layers [8, 15].

Long training time is a serious concern as networks become very deep. The forward and backward passes scale linearly with the depth of the network. Even on modern computers with multiple state-of-the-art GPUs, architectures like the 152-layer ResNet require several weeks to converge on the ImageNet dataset [8].

The researcher is faced with an inherent dilemma: shorter networks have the advantage that information flows efficiently forward and backward, and can therefore be trained effectively and within a reasonable amount of time. However, they are not expressive enough to represent the complex concepts that are commonplace in computer vision applications. Very deep networks have much greather model complexity, but are very difficult to train in practice and require a lot of time and patience.

In this paper, we propose *deep networks with stochastic depth*, a novel training algorithm that is based on the seemingly contradictory insight that ideally we would like to have a *deep* network during *testing* but a *short* network during *training*. We resolve this conflict by creating deep Residual Network [8] architectures (with hundreds or even thousands of layers) with sufficient modeling capacity; however, during training we shorten the network significantly by randomly removing a substantial fraction of layers independently for each sample or mini-batch. The effect is a network with a small *expected* depth during training, but a large depth during testing. Although seemingly simple, this approach is surprisingly effective in practice.

In extensive experiments we observe that training with stochastic depth substantially reduces training time and test error (resulting in multiple new records to the best of our knowledge at the time of initial submission to ECCV). The reduction in training time can be attributed to the shorter forward and backward propagation, so the training time no longer scales with the full depth, but the shorter *expected depth* of the network. We attribute the reduction in test error to two factors: 1) shortening the (expected) depth during training reduces the chain of forward propagation steps and gradient computations, which strengthens the gradients especially in earlier layers during backward propagation; 2) networks trained with stochastic depth can be interpreted as an implicit *ensemble* of networks of different depths, mimicking the record breaking ensemble of depth varying ResNets trained by He et al. [8].

We also observe that similar to Dropout [16], training with stochastic depth acts as a regularizer, even in the presence of Batch Normalization [14]. On experiments with CIFAR-10, we increase the depth of a ResNet beyond 1000 layers and still obtain significant improvements in test error.

2 Background

Many attempts have been made to improve the training of very deep networks. Earlier works adopted greedy layer-wise training or better initialization schemes to alleviate the vanishing gradients and diminishing feature reuse problems [12, 17, 18]. A notable recent contribution towards training of very deep networks is Batch Normalization [14], which standardizes the mean and variance of hidden layers with respect to each mini-batch. This approach reduces the vanishing gradients problem and yields a strong regularizing effect.

Recently, several authors introduced extra skip connections to improve the information flow during forward and backward propagation. Highway Networks [15] allow earlier representations to flow unimpededly to later layers through parameterized skip connections known as "information highways", which can cross several layers at once. The skip connection parameters, learned during training, control the amount of information allowed on these "highways".

Residual networks (ResNets)*[8]* simplify Highway Networks by shortcutting (mostly) with identity functions. This simplification greatly improves training efficiency, and enables more direct feature reuse. ResNets are motivated by the observation that neural networks tend to obtain *higher training error* as the depth increases to very large values. This is counterintuitive, as the network gains more parameters and therefore better function approximation capabilities. The authors conjecture that the networks become *worse* at function approximation because the gradients and training signals vanish when they are propagated through many layers. As a fix, they propose to add *skip connections* to the network. Formally, if H_ℓ denotes the output of the ℓ^{th} layer (or sequence of layers) and $f_\ell(\cdot)$ represents a typical convolutional transformation from layer $\ell-1$ to ℓ, we obtain

$$H_\ell = \texttt{ReLU}(f_\ell(H_{\ell-1}) + \text{id}(H_{\ell-1})), \tag{1}$$

where $\text{id}(\cdot)$ denotes the identity transformation and we assume a `ReLU` transition function [19]. Fig. 1 illustrates an example of a function f_ℓ, which consists of multiple convolutional and Batch Normalization layers. When the output dimensions of f_ℓ do not match those of $H_{\ell-1}$, the authors redefine $\text{id}(\cdot)$ as a linear projection to reduce the dimensions of $\text{id}(H_{\ell-1})$ to match those of $f_\ell(H_{\ell-1})$. The propagation rule in (1) allows the network to pass gradients and features (from the input or those learned in earlier layers) back and forth between the layers via the identity transformation $\text{id}(\cdot)$.

Dropout. Stochastically dropping hidden nodes or connections has been a popular regularization method for neural networks. The most notable example is Dropout [16], which multiplies each hidden activation by an independent

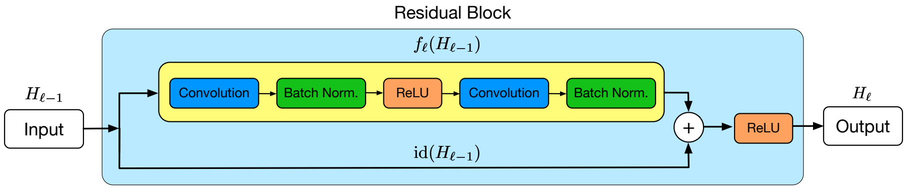


Fig. 1. A close look at the ℓ^{th} ResBlock in a ResNet.

Bernoulli random variable. Intuitively, Dropout reduces the effect known as "co-adaptation" of hidden nodes collaborating in groups instead of independently producing useful features; it also makes an analogy with training an ensemble of exponentially many small networks. Many follow up works have been empirically successful, such as DropConnect [20], Maxout [21] and DropIn [22].

Similar to Dropout, stochastic depth can be interpreted as training an ensemble of networks, but with different depths, possibly achieving higher diversity among ensemble members than ensembling those with the same depth. Different from Dropout, we make the network shorter instead of thinner, and are motivated by a different problem. Anecdotally, Dropout loses effectiveness when used in combination with Batch Normalization [14, 23]. Our own experiments with various Dropout rates (on CIFAR-10) show that Dropout gives practically no improvement when used on 110-layer ResNets with Batch Normalization.

We view all of these previous approaches to be extremely valuable and consider our proposed training with stochastic depth complimentary to these efforts. In fact, in our experiments we show that training with stochastic depth is indeed very effective on ResNets with Batch Normalization.

3 Deep Networks with Stochastic Depth

Learning with stochastic depth is based on a simple intuition. To reduce the *effective* length of a neural network during training, we randomly skip layers entirely. We achieve this by introducing skip connections in the same fashion as ResNets, however the connection pattern is randomly altered for each mini-batch. For each mini-batch we randomly select sets of layers and remove their corresponding transformation functions, only keeping the identity skip connection. Throughout, we use the architecture described by He et al. [8]. Because the architecture already contains skip connections, it is straightforward to modify, and isolates the benefits of stochastic depth from that of the ResNet identity connections. Next we describe this network architecture and then explain the stochastic depth training procedure in detail.

ResNet architecture. Following He et al. [8], we construct our network as the functional composition of L *residual blocks* (ResBlocks), each encoding the update rule (1). Fig. 1 shows a schematic illustration of the ℓ^{th} ResBlock. In this example, f_ℓ consists of a sequence of layers: `Conv-BN-ReLU-Conv-BN`, where

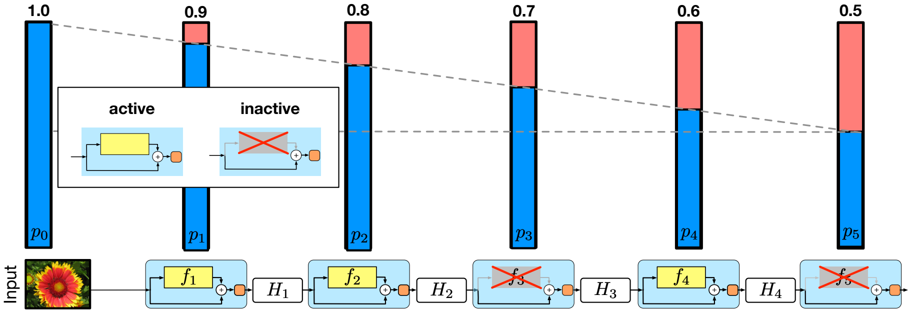


Fig. 2. The linear decay of p_ℓ illustrated on a ResNet with stochastic depth for $p_0 = 1$ and $p_L = 0.5$. Conceptually, we treat the input to the first ResBlock as H_0, which is always active.

`Conv` and `BN` stand for Convolution and Batch Normalization respectively. This construction scheme is adopted in all our experiments except ImageNet, for which we use the bottleneck block detailed in He et al. [8]. Typically, there are 64, 32, or 16 filters in the convolutional layers (see Section 4 for experimental details).

Stochastic depth aims to shrink the depth of a network during training, while keeping it unchanged during testing. We can achieve this goal by randomly dropping entire ResBlocks during training and bypassing their transformations through skip connections. Let $b_\ell \in \{0, 1\}$ denote a Bernoulli random variable, which indicates whether the ℓ^{th} ResBlock is active ($b_\ell = 1$) or inactive ($b_\ell = 0$). Further, let us denote the "survival" probability of ResBlock ℓ as $p_\ell = \Pr(b_\ell = 1)$.

With this definition we can bypass the ℓ^{th} ResBlock by multiplying its function f_ℓ with b_ℓ and we extend the update rule from (1) to

$$H_\ell = \texttt{ReLU}(b_\ell f_\ell(H_{\ell-1}) + \mathrm{id}(H_{\ell-1})). \tag{2}$$

If $b_\ell = 1$, eq. (2) reduces to the original ResNet update (1) and this ResBlock remains unchanged. If $b_\ell = 0$, the ResBlock reduces to the identity function,

$$H_\ell = \mathrm{id}(H_{\ell-1}). \tag{3}$$

This reduction follows from the fact that the input $H_{\ell-1}$ is always non-negative, at least for the architectures we use. For $\ell \geq 2$, it is the output of the previous ResBlock, which is non-negative because of the final `ReLU` transition function (see Fig. 1). For $\ell = 1$, its input is the output of a `Conv-BN-ReLU` sequence that begins the architecture before the first ResBlock. For non-negative inputs the `ReLU` transition function acts as an identity.

The survival probabilities p_ℓ are new hyper-parameters of our training procedure. Intuitively, they should take on similar values for neighboring ResBlocks. One option is to set $p_\ell = p_L$ uniformly for all ℓ to obtain a single hyper-parameter

p_L. Another possibility is to set them according to a smooth function of ℓ. We propose a simple linear decay rule from $p_0 = 1$ for the input, to p_L for the last ResBlock:

$$p_\ell = 1 - \frac{\ell}{L}(1 - p_L). \tag{4}$$

See Fig. 2 for a schematic illustration. The linearly decaying survival probability originates from our intuition that the earlier layers extract low-level features that will be used by later layers and should therefore be more reliably present. In Section 4 we perform a more detailed empirical comparison between the uniform and decaying assignments for p_ℓ. We conclude that the linear decay rule (4) is preferred and, as training with stochastic depth is surprisingly stable with respect to p_L, we set $p_L = 0.5$ throughout (see Fig. 8).

Expected network depth. During the forward-backward pass the transformation f_ℓ is bypassed with probability $(1 - p_\ell)$, leading to a network with reduced depth. With stochastic depth, the number of effective ResBlocks during training, denoted as $\tilde{L}$, becomes a random variable. Its expectation is given by: $E(\tilde{L}) = \sum_{\ell=1}^{L} p_\ell$.

Under the linear decay rule with $p_L = 0.5$, the expected number of ResBlocks during training reduces to $E(\tilde{L}) = (3L - 1)/4$, or $E(\tilde{L}) \approx 3L/4$ when L is large. For the 110-layer network with $L = 54$ commonly used in our experiments, we have $E(\tilde{L}) \approx 40$. In other words, with stochastic depth, we train ResNets with an average number of 40 ResBlocks, but recover a ResNet with 54 blocks at test time. This reduction in depth significantly alleviates the vanishing gradients and the information loss problem in deep ResNets. Note that because the connectivity is random, there will be updates with significantly shorter networks and more direct paths to individual layers. We provide an empirical demonstration of this effect in Section 5.

Training time savings. When a ResBlock is bypassed for a specific iteration, there is no need to perform forward-backward computation or gradient updates. As the forward-backward computation dominates the training time, stochastic depth significantly speeds up the training process. Following the calculations above, approximately 25% of training time could be saved under the linear decay rule with $p_L = 0.5$. The timings in practice using our implementation are consistent with this analysis (see the last paragraph of Section 4). More computational savings can be obtained by switching to a uniform probability for p_ℓ or lowering p_L accordingly. In fact, Fig. 8 shows that with $p_L = 0.2$, the ResNet with stochastic depth obtains the same test error as its constant depth counterpart on CIFAR-10 but gives a 40% speedup.

Implicit model ensemble. In addition to the predicted speedups, we also observe significantly lower testing errors in our experiments, in comparison with ResNets of constant depth. One explanation for our performance improvements is that training with stochastic depth can be viewed as training an ensemble of ResNets *implicitly*. Each of the L layers is either active or inactive, resulting

Table 1. Test error (%) of ResNets trained with stochastic depth compared to other most competitive methods previously published (whenever available). A "+" in the name denotes standard data augmentation. ResNet with constant depth refers to our reproduction of the experiments by He et al.

	CIFAR10+	CIFAR100+	SVHN	ImageNet
Maxout [21]	9.38	-	2.47	-
DropConnect [20]	9.32	-	1.94	-
Net in Net [24]	8.81	-	2.35	-
Deeply Supervised [13]	7.97	-	1.92	33.70
Frac. Pool [25]	-	27.62	-	-
All-CNN [6]	7.25	-	-	41.20
Learning Activation [26]	7.51	30.83	-	-
R-CNN [27]	7.09	-	1.77	-
Scalable BO [28]	6.37	27.40	1.77	-
Highway Network [29]	7.60	32.24	-	-
Gen. Pool [30]	6.05	-	1.69	28.02
ResNet with constant depth	6.41	27.76	1.80	21.78
ResNet with stochastic depth	5.25	24.98	1.75	21.98

in 2^L possible network combinations. For each training mini-batch one of the 2^L networks (with shared weights) is sampled and updated. During testing all networks are averaged using the approach in the next paragraph.

Stochastic depth during testing requires small modifications to the network. We keep all functions f_ℓ active throughout testing in order to utilize the full-length network with all its model capacity. However, during training, functions f_ℓ are only active for a fraction p_ℓ of all updates, and the corresponding weights of the next layer are calibrated for this survival probability. We therefore need to re-calibrate the outputs of any given function f_ℓ by the expected number of times it participates in training, p_ℓ. The forward propagation update rule becomes:

$$H_\ell^{\text{Test}} = \texttt{ReLU}(p_\ell f_\ell(H_{\ell-1}^{\text{Test}}; W_\ell) + H_{\ell-1}^{\text{Test}}). \tag{5}$$

From the model ensemble perspective, the update rule (5) can be interpreted as combining all possible networks into a single test architecture, in which each layer is weighted by its survival probability.

4 Results

We empirically demonstrate the effectiveness of stochastic depth on a series of benchmark data sets: CIFAR-10, CIFAR-100 [1], SVHN [31], and ImageNet [2].

Implementation details. For all data sets we compare the results of ResNets with our proposed stochastic depth and the original constant depth, and other most competitive benchmarks. We set p_ℓ with the linear decay rule of $p_0 = 1$ and $p_L = 0.5$ throughout. In all experiments we report the test error from the epoch with the lowest validation error. For best comparisons we use the same

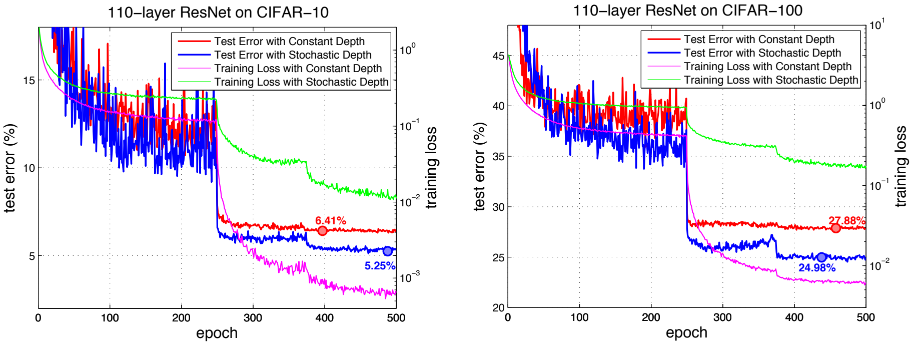


Fig. 3. Test error on CIFAR-10 (*left*) and CIFAR-100 (*right*) during training, with data augmentation, corresponding to results in the first two columns of Table 1.

construction scheme (for constant and stochastic depth) as described by He et al. [8]. In the case of CIFAR-100 we use the same 110-layer ResNet used by He et al. [8] for CIFAR-10, except that the network has a 100-way softmax output. Each model contains three groups of residual blocks that differ in number of filters and feature map size, and each group is a stack of 18 residual blocks. The numbers of filters in the three groups are 16, 32 and 64, respectively. For the transitional residual blocks, i.e. the first residual block in the second and third group, the output dimension is larger than the input dimension. Following He et al. [8], we replace the identity connections in these blocks by an average pooling layer followed by zero paddings to match the dimensions. Our implementations are in Torch 7 [32]. The code to reproduce the results is publicly available on GitHub at `https://github.com/yueatsprograms/Stochastic_Depth`.

CIFAR-10. CIFAR-10 [1] is a dataset of 32-by-32 color images, representing 10 classes of natural scene objects. The training set and test set contain 50,000 and 10,000 images, respectively. We hold out 5,000 images as validation set, and use the remaining 45,000 as training samples. Horizontal flipping and translation by 4 pixels are the two standard data augmentation techniques adopted in our experiments, following the common practice [6, 13, 20, 21, 24, 26, 30].

The baseline ResNet is trained with SGD for 500 epochs, with a mini-batch size 128. The initial learning rate is 0.1, and is divided by a factor of 10 after epochs 250 and 375. We use a weight decay of 1e-4, momentum of 0.9, and Nesterov momentum [33] with 0 dampening, as suggested by [34]. For stochastic depth, the network structure and all optimization settings are exactly the same as the baseline. All settings were chosen to match the setup of He et al. [8].

The results are shown in Table 1. ResNets with constant depth result in a competitive 6.41% error on the test set. ResNets trained with stochastic depth yield a further relative improvement of 18% and result in 5.25% test error. To our knowledge this is significantly lower than the best existing single model performance (6.05%) [30] on CIFAR-10 prior to our submission, without resorting

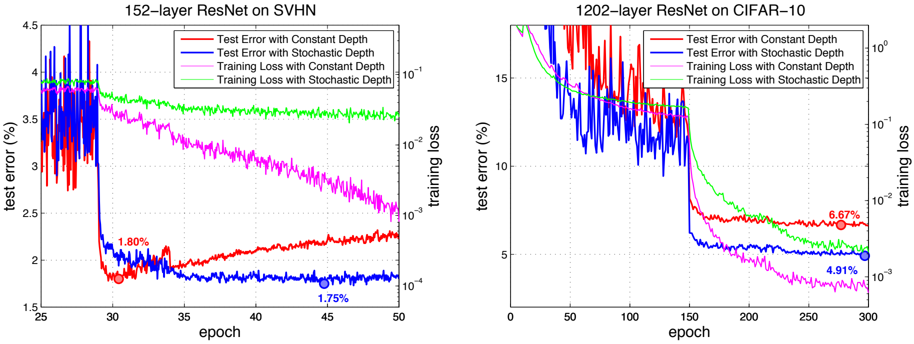


Fig. 4. Left: Test error on SVHN, corresponding to results on column three in Table 1. *right*: Test error on CIFAR-10 using 1202-layer ResNets. The points of lowest validation errors are highlighted in each case.

to massive data augmentation [6, 25].[1] Fig. 3 (*left*) shows the test error as a function of epochs. The point selected by the lowest validation error is circled for both approaches. We observe that ResNets with stochastic depth yield lower test error but also slightly higher fluctuations (presumably due to the random depth during training).

CIFAR-100. Similar to CIFAR-10, CIFAR-100 [1] contains 32-by-32 color images with the same train-test split, but from 100 classes. For both the baseline and our method, the experimental settings are exactly the same as those of CIFAR-10. The constant depth ResNet yields a test error of 27.22%, which is already the state-of-the-art in CIFAR-100 with standard data augmentation. Adding stochastic depth drastically reduces the error to 24.98%, and is again the best published single model performance to our knowledge (see Table 1 and Fig. 3 *right*).

We also experiment with CIFAR-10 and CIFAR-100 without data augmentation. ResNets with constant depth obtain 13.63% and 44.74% on CIFAR-10 and CIFAR-100 respectively. Adding stochastic depth yields consistent improvements of about 15% on both datasets, resulting in test errors of 11.66% and 37.8% respectively.

SVHN. The format of the Street View House Number (SVHN) [31] dataset that we use contains 32-by-32 colored images of cropped out house numbers from Google Street View. The task is to classify the digit at the center. There are 73,257 digits in the training set, 26,032 in the test set and 531,131 easier samples for additional training. Following the common practice, we use all the training samples but do not perform data augmentation. For each of the ten classes, we randomly select 400 samples from the training set and 200 from the additional set, forming a validation set with 6,000 samples in total. We preprocess

[1] The only model that performs even better is the 1202-layer ResNet with stochastic depth, discussed later in this section.

Table 2. Training time comparison on benchmark datasets.

	CIFAR10+	CIFAR100+	SVHN
Constant Depth	20h 42m	20h 51m	33h 43m
Stochastic Depth	15h 7m	15h 20m	25h 33m

the data by subtracting the mean and dividing the standard deviation. Batch size is set to 128, and validation error is calculated every 200 iterations.

Our baseline network has 152 layers. It is trained for 50 epochs with a beginning learning rate of 0.1, divided by 10 after epochs 30 and 35. The depth and learning rate schedule are selected by optimizing for the validation error of the baseline through many trials. This baseline obtains a competitive result of 1.80%. However, as seen in Fig. 4, it starts to overfit at the beginning of the second phase with learning rate 0.01, and continues to overfit until the end of training. With stochastic depth, the error improves to 1.75%, the second-best published result on SVHN to our knowledge after [30].

Training time comparison. We compare the training efficiency of the constant depth and stochastic depth ResNets used to produce the previous results. Table 2 shows the training (clock) time under both settings with the linear decay rule $p_L = 0.5$. Stochastic depth consistently gives a 25% speedup, which confirms our analysis in Section 3. See Fig. 8 and the corresponding section on hyper-parameter sensitivity for more empirical analysis.

Training with a 1202-layer ResNet. He et al. [8] tried to learn CIFAR-10 using an aggressively deep ResNet with 1202 layers. As expected, this extremely deep network overfitted to the training set: it ended up with a test error of 7.93%, worse than their 110-layer network. We repeat their experiment on the same 1202-layer network, with constant and stochastic depth. We train for 300 epochs, and set the learning rate to 0.01 for the first 10 epochs to "warm-up" the network and facilitate initial convergence, then restore it to 0.1, and divide it by 10 at epochs 150 and 225.

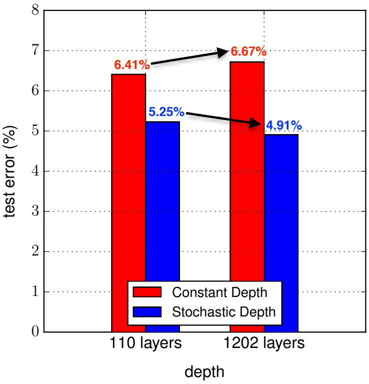


Fig. 5. With stochastic depth, the 1202-layer ResNet still significantly improves over the 110-layer one.

The results are summarized in Fig. 4 (*right*) and Fig. 5. Similar to He et al. [8], the ResNets with constant depth of 1202 layers yields a test error of 6.67%, which is worse than the 110-layer constant depth ResNet. In contrast, if trained with stochastic depth, this extremely deep ResNet performs remarkably well. We want to highlight two trends: 1) Comparing the two 1202-layer nets shows that training with stochastic depth leads to a 27% relative improvement; 2)

Comparing the two networks trained with stochastic depth shows that increasing the architecture from 110 layers to 1202 yields a further improvement on the previous record-low 5.25%, to a 4.91% test error without sign of overfitting, as shown in Fig. 4 (*right*) [2].

To the best of our knowledge, this is the lowest known test error on CIFAR-10 with moderate image augmentation and the first time that a network with more than 1000 layers has been shown to *further reduce* the test error [3]. We consider these findings highly encouraging and hope that training with stochastic depth will enable researchers to leverage extremely deep architectures in the future.

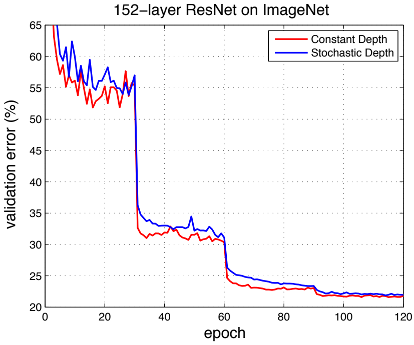


Fig. 6. Validation error on ILSVRC 2012 classification.

ImageNet. The ILSVRC 2012 classification dataset consists of 1000 classes of images, in total 1.2 million for training, 50,000 for validation, and 100,000 for testing. Following the common practice, we only report the validation errors. We follow He et al. [8] to build a 152-layer ResNet with 50 bottleneck residual blocks. When input and output dimensions do not match, the skip connection uses a learned linear projection for the mismatching dimensions, and an identity transformation for the other dimensions. Our implementation is based on the github repository `fb.resnet.torch`[4] [34], and the optimization settings are the same as theirs, except that we use a batch size of 128 instead of 256 because we can only spread a batch among 4 GPUs (instead of 8 as they did).

We train the constant depth baseline for 90 epochs (following He et al. and the default setting in the repository) and obtain a final error of 23.06%. With stochastic depth, we obtain an error of 23.38% at epoch 90, which is slightly higher. We observe from Fig.6 that the downward trend of the validation error with stochastic depth is still strong, and from our previous experience, could benefit from further training. Due to the 25% computational saving, we can add 30 epochs (giving 120 in total, after decreasing the learning rate to 1e-4 at epoch 90), and still finish in almost the same total time as 90 epochs of the baseline. This reaches a final error of 21.98%. We have also kept the baseline running for 30 more epochs. This reaches a final error of 21.78%.

Because ImageNet is a very complicated and large dataset, the model complexity required could potentially be much more than that of the 152-layer

[2] We do not include this result in Table 1 since this architecture was only trained on one of the datasets.

[3] This is, until early March, 2016, when this paper was submitted to ECCV. Many new developments have further decreased the error on CIFAR-10 since then (and some are based on this work).

[4] `https://github.com/facebook/fb.resnet.torch`

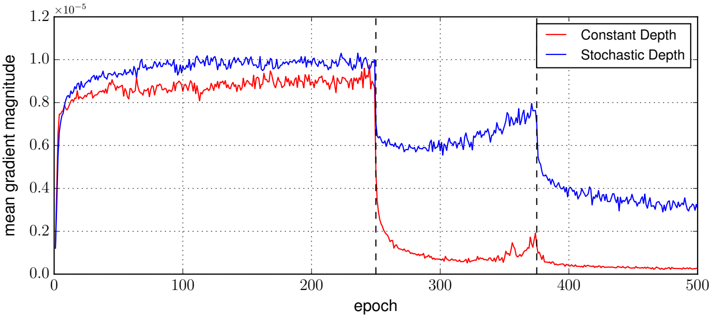


Fig. 7. The first convolutional layer's mean gradient magnitude for each epoch during training. The vertical dotted lines indicate scheduled reductions in learning rate by a factor of 10, which cause gradients to shrink.

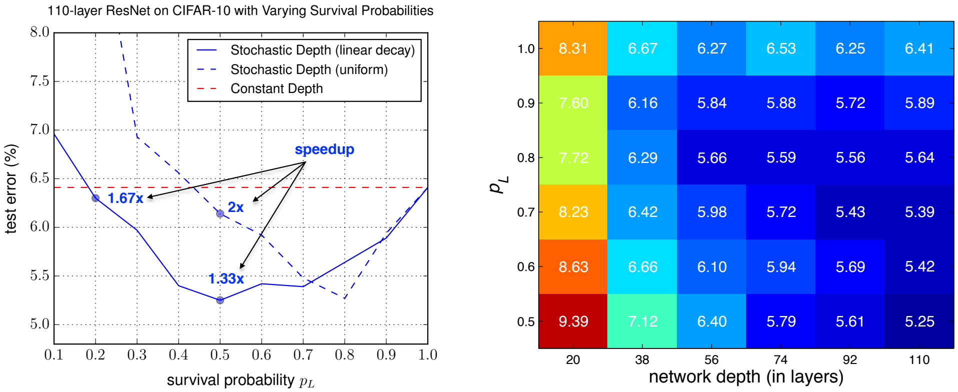
ResNet [35]. In the words of an anonymous reviewer, the current generation of models for ImageNet are still in a different regime from those of CIFAR. Although there seems to be no immediate benefit from applying stochastic depth on this particular architecture, it is possible that stochastic depth will lead to improvements on ImageNet with larger models, which the community might soon be able to train as GPU capacities increase.

5 Analytic Experiments

In this section, we provide more insights into stochastic depth by presenting a series of analytical results. We perform experiments to support the hypothesis that stochastic depth effectively addresses the problem of vanishing gradients in backward propagation. Moreover, we demonstrate the robustness of stochastic depth with respect to its hyper-parameter.

Improved gradient strength. Stochastically dropping layers during training reduces the effective depth on which gradient back-propagation is performed, while keeping the test-time model depth unmodified. As a result we expect training with stochastic depth to reduce the vanishing gradient problem in the backward step. To empirically support this, we compare the magnitude of gradients to the first convolutional layer of the first ResBlock ($\ell = 1$) with and without stochastic depth on the CIFAR-10 data set.

Fig. 7 shows the mean absolute values of the gradients. The two large drops indicated by vertical dotted lines are due to scheduled learning rate division. It can be observed that the magnitude of gradients in the network trained with stochastic depth is always larger, especially after the learning rate drops. This seems to support out claim that stochastic depth indeed significantly reduces the vanishing gradient problem, and enables the network to be trained more effectively. Another indication of the effect is in the left panel of Fig. 3, where one can observe that the test error of the ResNets with constant depth approximately plateaus after the first drop of learning rate, while stochastic depth still



p_L	20	38	56	74	92	110
1.0	8.31	6.67	6.27	6.53	6.25	6.41
0.9	7.60	6.16	5.84	5.88	5.72	5.89
0.8	7.72	6.29	5.66	5.59	5.56	5.64
0.7	8.23	6.42	5.98	5.72	5.43	5.39
0.6	8.63	6.66	6.10	5.94	5.69	5.42
0.5	9.39	7.12	6.40	5.79	5.61	5.25

Fig. 8. Left: Test error (%) on CIFAR-10 with respect to the p_L with uniform and decaying assignments of p_ℓ. Right: Test error (%) heatmap on CIFAR-10 varyied over p_L and network depth.

improves the performance even after the learning rate drops for the second time. This further supports that stochastic depth combines the benefits of shortened network during training with those of deep models at test time.

Hyper-parameter sensitivity. The survival probability p_L is the only hyper-parameter of our method. Although we used $p_L = 0.5$ throughout all our experiments, it is still worth investigating the sensitivity of stochastic depth with respect to its hyper-parameter. To this end, we compare the test error of the 110-layer ResNet under varying values of p_L ($L = 54$) for both linear decay and uniform assignment rules on the CIFAR-10 data set in Fig. 8 (*left*). We make the following observations: 1) both assignment rules yield better results than the baseline when p_L is set properly; 2) the linear decay rule outperforms the uniform rule consistently; 3) the linear decay rule is relatively robust to fluctuations in p_L and obtains competitive results when p_L ranges from 0.4 to 0.8; 4) even with a rather small survival probability e.g. $p_L = 0.2$, stochastic depth with linear decay still performs well, while giving a 40% reduction in training time. This shows that stochastic depth can save training time substantially without compromising accuracy.

The heatmap on the right shows the test error varied over both p_L and network depth. Not surprisingly, deeper networks (at least in the range of our experiments) do better with a $p_L = 0.5$. The ”valley” of the heatmap is along the diagonal. A deep enough model is necessary for stochastic depth to significantly outperform the baseline (an observation we also make with the ImageNet data set), although shorter networks can still benefit from less aggressive skipping.

6 Conclusion

In this paper we introduced deep networks with *stochastic depth*, a procedure to train very deep neural networks effectively and efficiently. Stochastic depth reduces the network depth during training *in expectation* while maintaining the full depth at testing time. Training with stochastic depth allows one to increase the depth of a network well beyond 1000 layers, and still obtain a reduction in test error. Because of its simplicity and practicality we hope that training with stochastic depth may become a new tool in the deep learning "toolbox", and will help researchers scale their models to previously unattainable depths and capabilities.

Acknowledgements. We thank the anonymous reviewers for their kind suggestions. Kilian Weinberger is supported by NFS grants IIS-1550179, IIS-1525919 and EFRI-1137211. Gao Huang is supported by the International Postdoctoral Exchange Fellowship Program of China Postdoctoral Council (No.20150015). Yu Sun is supported by the Cornell University Office of Undergraduate Research. We also thank our lab mates, Matthew Kusner and Shuang Li for useful and interesting discussions.

Bibliography

[1] Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images (2009)

[2] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE (2009) 248–255

[3] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. (2012) 1097–1105

[4] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., LeCun, Y.: Overfeat: Integrated recognition, localization and detection using convolutional networks. arXiv preprint arXiv:1312.6229 (2013)

[5] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)

[6] Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv preprint arXiv:1412.6806 (2014)

[7] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 1–9

[8] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. arXiv preprint arXiv:1512.03385 (2015)

[9] Håstad, J., Goldmann, M.: On the power of small-depth threshold circuits. Computational Complexity **1**(2) (1991) 113–129

[10] Håstad, J.: Computational limitations of small-depth circuits. (1987)

[11] Bengio, Y., Simard, P., Frasconi, P.: Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on **5**(2) (1994) 157–166

[12] Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: International conference on artificial intelligence and statistics. (2010) 249–256

[13] Lee, C.Y., Xie, S., Gallagher, P., Zhang, Z., Tu, Z.: Deeply-supervised nets. arXiv preprint arXiv:1409.5185 (2014)

[14] Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167 (2015)

[15] Srivastava, R.K., Greff, K., Schmidhuber, J.: Highway networks. arXiv preprint arXiv:1505.00387 (2015)

[16] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research **15**(1) (2014) 1929–1958

[17] Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. (1989)

[18] Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? The Journal of Machine Learning Research **11** (2010) 625–660

[19] Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: Proceedings of the 27th International Conference on Machine Learning (ICML-10). (2010) 807–814

[20] Wan, L., Zeiler, M., Zhang, S., Cun, Y.L., Fergus, R.: Regularization of neural networks using dropconnect. In Dasgupta, S., Mcallester, D., eds.: Proceedings of the 30th International Conference on Machine Learning (ICML-13). Volume 28., JMLR Workshop and Conference Proceedings (May 2013) 1058–1066

[21] Goodfellow, I.J., Warde-Farley, D., Mirza, M., Courville, A., Bengio, Y.: Maxout networks. arXiv preprint arXiv:1302.4389 (2013)

[22] Smith, L.N., Hand, E.M., Doster, T.: Gradual dropin of layers to train very deep neural networks. CVPR (2016)

[23] Zagoruyko, S.: 92.45% on cifar-10 in torch (2015)

[24] Lin, M., Chen, Q., Yan, S.: Network in network. arXiv preprint arXiv:1312.4400 (2013)

[25] Graham, B.: Fractional max-pooling. arXiv preprint arXiv:1412.6071 (2014)

[26] Agostinelli, F., Hoffman, M., Sadowski, P., Baldi, P.: Learning activation functions to improve deep neural networks. arXiv preprint arXiv:1412.6830 (2014)

[27] Liang, M., Hu, X.: Recurrent convolutional neural network for object recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. (2015) 3367–3375

[28] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Ali, M., Adams, R.P., et al.: Scalable bayesian optimization using deep neural networks. arXiv preprint arXiv:1502.05700 (2015)

[29] Srivastava, R.K., Greff, K., Schmidhuber, J.: Training very deep networks. In: Advances in Neural Information Processing Systems. (2015) 2368–2376

[30] Lee, C.Y., Gallagher, P.W., Tu, Z.: Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. arXiv preprint arXiv:1509.08985 (2015)

[31] Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning. In: NIPS workshop on deep learning and unsupervised feature learning. Volume 2011., Granada, Spain (2011) 4

[32] Collobert, R., Kavukcuoglu, K., Farabet, C.: Torch7: A matlab-like environment for machine learning. In: BigLearn, NIPS Workshop. (2011)

[33] Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: Proceedings of the 30th international conference on machine learning (ICML-13). (2013) 1139–1147

[34] Gross, S., Wilber, M.: Training and investigating residual nets (2016)

[35] He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. arXiv preprint arXiv:1603.05027 (2016)