Advanced Residual Networks For Enhancing Image Classification

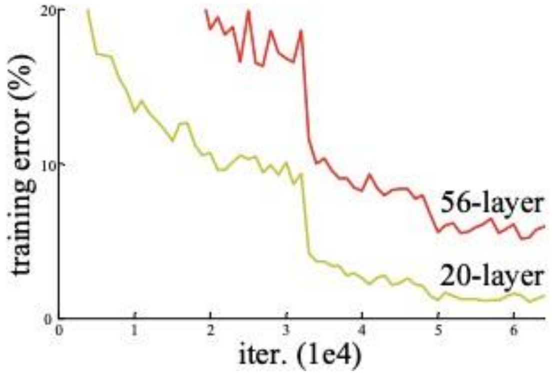
Heta Desai

Abstract

Training deep neural networks becomes increasingly challenging as their depth grows. To address this, we introduce a residual learning framework that simplifies the training of networks significantly deeper than those traditionally used. Rather than having each layer learn a direct mapping, we reformulate the layers to learn residual functions—essentially focusing on the difference from the input, which eases optimization. Our experiments provide strong evidence that these residual networks (ResNets) are easier to train and benefit from increased depth. On the ImageNet dataset, we tested ResNets with up to 152 layers, which is eight times deeper than VGG networks [41], yet they remain less complex in terms of computation. An ensemble of these deep ResNets achieved a 3.57% error rate, securing first place in the ILSVRC 2015 classification task. We also analyzed performance on CIFAR-10, experimenting with networks up to 100 and 1000 layers deep. Our findings show that deeper representations are crucial for visual recognition tasks. Thanks to the extreme depth of our models, we achieved a 28% relative improvement on the COCO object detection dataset. These deep residual networks were also key to our top-ranking entries in the ILSVRC and COCO 2015 competitions, winning first place in multiple tasks including ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

1. Introduction

Deep convolutional neural networks [22, 21] have driven major advances in image classification [21, 50, 40]. These networks effectively combine low-, mid-, and high-level features [50] with classifiers in a unified, multi-layer architecture. Increasing the number of layers—i.e., adding more depth—enhances the richness of the feature representations. Recent studies [41, 44] highlight the critical role of network depth, with top-performing models on the demanding ImageNet dataset [36] relying on very deep architectures [41, 44, 13, 16], ranging from 16 layers [41] up to 30 layers [16]. Beyond image classification, many other complex visual recognition tasks [8, 12, 7, 32, 27] have also shown substantial improvements when utilizing deep neural networks, underscoring the widespread impact of increased depth across various domains.



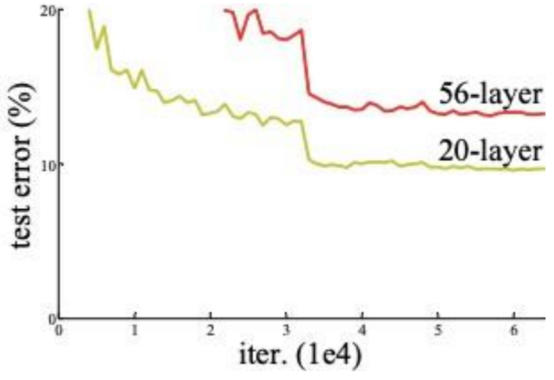


Figure 1 shows the **training error (left)** and **test error (right)** for 20-layer and 56-layer "plain" networks on the **CIFAR-10 dataset**. Interestingly, the deeper 56-layer network exhibits **higher training error**, which also leads to **higher test error**. A similar trend is observed on the **ImageNet dataset**, as illustrated in **Figure 4**.

Given the importance of network depth, a natural question arises: Is building better networks as simple as adding more layers? One major challenge in answering this has been the well-known issue of vanishing or exploding gradients [1, 9], which can prevent proper convergence during training. However, this issue has been largely mitigated through normalized initialization [23, 9, 37, 13] and intermediate normalization layers [16], which allow deep networks with many layers to begin converging effectively using stochastic gradient descent (SGD) and backpropagation [22].

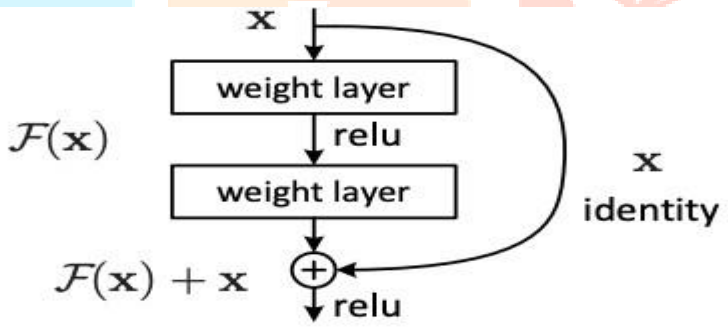

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.

Yet, once convergence becomes feasible in deeper networks, a new challenge emerges: the degradation problem. As depth increases, accuracy initially saturates—an expected outcome—but then unexpectedly declines. Notably, this decline is not due to overfitting. In fact, adding more layers to an already deep network can result in higher training error, as previously reported in [11, 42] and confirmed by our experiments. Figure 1 illustrates this typical behavior.

The drop in training accuracy suggests that not all neural network architectures are equally easy to optimize. Let's compare a shallower network with a deeper version that adds extra layers on top. In theory, the deeper network should be at least as good as the shallower one, since we can construct a solution where the additional layers perform identity mappings, and the original layers replicate the trained shallower model. This implies that the deeper network should not have higher training error than the shallow one. However, in practice, current optimization methods often fail to find such good solutions within a reasonable time.

To tackle this degradation issue, this paper proposes a deep residual learning approach. Instead of forcing every group of stacked layers to directly learn the target function, we guide them to learn the *residual* — the difference between the desired function and the input. Mathematically, if the goal is to learn a function H(x)H(x), the network instead learns F(x)=H(x)−xF(x) = H(x) - x, and the final output becomes F(x)+xF(x) + x. We believe it's easier to learn the residual F(x)F(x) than the original mapping H(x)H(x). In cases where the best function is simply the identity, it's easier to learn a residual of zero than to make multiple nonlinear layers replicate the identity function.

This idea is implemented using feedforward neural networks with "shortcut connections" that skip one or more layers (see Fig. 2). In our approach, these shortcut connections perform identity mappings and their outputs are added to the outputs of the stacked layers. These identity shortcuts don't introduce additional parameters or computational overhead. The entire model remains trainable via standard backpropagation and can be implemented using popular deep learning frameworks like Caffe without altering the optimization tools.

We conducted extensive experiments on the ImageNet dataset [36] to demonstrate the degradation issue and assess the effectiveness of our proposed method. Our findings show that:

1. Deep residual networks are much easier to train compared to their "plain" counterparts, which stack layers without residual connections and suffer from higher training errors as depth increases;
2. Residual networks benefit significantly from increased depth, achieving much higher accuracy than previous models.

We observed similar results on the CIFAR-10 dataset [20], indicating that the training challenges and benefits of residual learning are not limited to a specific dataset. On CIFAR-10, we successfully trained networks with more than 100 layers and even experimented with architectures exceeding 1000 layers.

On ImageNet, our extremely deep residual networks delivered outstanding performance. Our 152-layer residual network, the deepest model reported on ImageNet at the time, outperformed VGG networks [41] while being less computationally intensive. Our ensemble achieved a top-5 error rate of just 3.57% on the ImageNet test set, securing 1st place in the ILSVRC 2015 classification challenge.

Additionally, these deep residual models generalized well to other tasks, earning top positions in ImageNet object detection, localization, and the COCO 2015 detection and segmentation challenges. These strong results highlight the broad applicability of the residual learning framework, not only in computer vision but potentially in other domains as well.

2. Related Work

Residual Representations:

In image recognition, representations like VLAD [18] encode information using residual vectors relative to a dictionary, and the Fisher Vector [30] can be viewed as a probabilistic extension of VLAD [18]. Both methods have proven to be effective shallow representations for tasks like image retrieval and classification [4, 48]. In vector quantization, it's been found that encoding the *residual* vectors [17] is more effective than encoding the original feature vectors.

In low-level vision and computer graphics, residual-based techniques are also commonly used for solving Partial Differential Equations (PDEs). The Multigrid method [3] breaks the problem into subproblems across different scales, where each one solves for the *residual* between a coarse and fine level. Similarly, hierarchical basis preconditioning [45, 46] uses variables that capture residuals between scales. These approaches [3, 45, 46] have been shown to converge much faster than traditional methods that do not take residuals into account. These findings emphasize that clever reformulation—especially through residuals—can make optimization significantly easier.

Shortcut Connections:

Shortcut connections, or techniques related to them, have a long history in neural network research [2, 34, 49]. In early multilayer perceptron (MLP) models, it was common to include a direct linear connection from the input to the output [34, 49]. To address issues like vanishing or exploding gradients, works like [44, 24] connected intermediate layers to auxiliary classifiers. Other research [39, 38, 31, 47] proposed methods to center activations, gradients, and error signals using shortcut-like mechanisms. The inception module in [44], for instance, includes both a shortcut and deeper parallel branches.

Around the same time as our work, "highway networks" [42, 43] introduced shortcut paths controlled by gating mechanisms [15]. These gates have learnable parameters and adjust dynamically based on input data. Unlike our identity-based shortcuts, these gates can close off information flow (when near zero), causing the network to learn standard, non-residual functions. In contrast, our method always learns residual functions—our identity shortcuts are always open and pass information directly, with residual mappings added on top. Moreover, highway networks haven't shown improvements in performance when scaled to very deep architectures (e.g., 100+ layers), whereas our residual networks have.

3. Deep Residual Learning

3.1. Residual Learning

Let’s consider a function H(x)H(x) that a set of stacked layers (not necessarily the entire network) is meant to learn, where xx is the input to the first layer in that stack. If we assume that multiple nonlinear layers are capable of approximating complex functions, then it follows they should also be capable of approximating *residual* functions—specifically, H(x)−xH(x) - x (as long as the input and output dimensions match). Therefore, instead of having the layers directly learn H(x)H(x), we have them learn the residual function F(x)=H(x)−xF(x) = H(x) - x, and the final output becomes F(x)+xF(x) + x. Although both formulations are theoretically capable of approximating the target function, the difficulty of training might differ between them.

This approach is inspired by the surprising issue of degradation (illustrated in Fig. 1, left). As discussed earlier, in theory, if we add layers that perform identity mappings to a network, the deeper model should perform no worse than the shallower one. However, the degradation problem indicates that optimization algorithms struggle to approximate identity mappings using multiple nonlinear layers. With our residual learning formulation, if an identity mapping is ideal, the optimizer can simply reduce the weights in those layers toward zero—making it easier to reach the identity function.

While in practice the ideal function is rarely a pure identity mapping, this residual approach can act as a form of preconditioning. If the target function is closer to the identity than to zero, it's easier for the optimizer to make small adjustments starting from an identity mapping than to learn the whole function from scratch. Our experiments (see Fig. 7) show that the learned residual functions typically have small magnitudes, supporting the idea that identity mappings offer a helpful starting point for optimization.

3.2. Identity Mapping by Shortcuts

We apply residual learning to groups of stacked layers, with each group forming a residual building block (illustrated in Fig. 2). Each block is formally defined as:

y=F(x,{Wi})+x(1)y = F(x, \{W_i\}) + x (1)

Here, xx and yy are the input and output of the block, and F(x,{Wi})F(x, \{W_i\}) represents the residual function to be learned. For instance, in the two-layer example from Fig. 2, the residual function is F=W2σ(W1x)F = W_2\sigma(W_1x), where σ\sigma is the ReLU activation [29], and biases are omitted for simplicity. The result of the residual function is added to the input via a shortcut connection using element-wise addition. The activation function σ\sigma is applied after this addition (i.e., σ(y)\sigma(y); see Fig. 2).

These shortcut connections don't introduce any additional parameters or computational cost, which is beneficial for practical applications. This also ensures that comparisons between residual and plain networks are fair, as both can have the same depth, width, number of parameters, and computational load—apart from the minor cost of the element-wise addition.

For the shortcut addition in Equation (1) to work, xx and FF must have matching dimensions. If they differ (for example, due to changes in the number of channels), a linear projection WsW_s can be applied to xx through the shortcut path, modifying the equation to:

y=F(x,{Wi})+Wsx(2)y = F(x, \{W_i\}) + W_sx (2)

While a square matrix WsW_s can be used in general, our experiments show that using the identity mapping is usually enough to address the degradation issue and is more efficient. We only use projections when a change in dimensionality requires it.

The design of the residual function FF is flexible. Our experiments mainly use functions with two or three layers (see Fig. 5), though deeper configurations are also possible. However, when FF contains only a single layer, the structure y=W1x+xy = W_1x + x behaves like a linear transformation with no observed benefits. Finally, although the notation above assumes fully connected layers for simplicity, the same structure applies to convolutional networks. In this case, F(x,{Wi})F(x, \{W_i\}) represents convolutional operations, and the element-wise addition is applied channel-wise on feature maps.

3.3. Network Architectures

We evaluated several plain and residual networks and observed consistent patterns across all tests. To illustrate our findings, we describe two specific models used for ImageNet experiments.

Plain Network:

Our plain baseline models (shown in Fig. 3, middle) are largely based on the design principles of VGG networks [41] (Fig. 3, left). Most of the convolutional layers use 3×3 filters and follow two main design rules:

1. Layers that produce feature maps of the same size use the same number of filters.
2. When the feature map size is reduced by half (e.g., due to downsampling), the number of filters is doubled. This approach helps maintain a consistent computational cost across layers.

Downsampling is done using convolutional layers with a stride of 2. The network concludes with a global average pooling layer, followed by a 1000-class fully connected layer and a softmax activation. The model shown in Fig. 3 (middle) has a total of 34 weighted layers. It's important to note that our model uses fewer filters and has lower computational complexity compared to VGG networks. For example, our 34-layer plain model requires
3.6 billion FLOPs (floating-point operations), which is just 18% of the 19.6 billion FLOPs used by VGG-19.

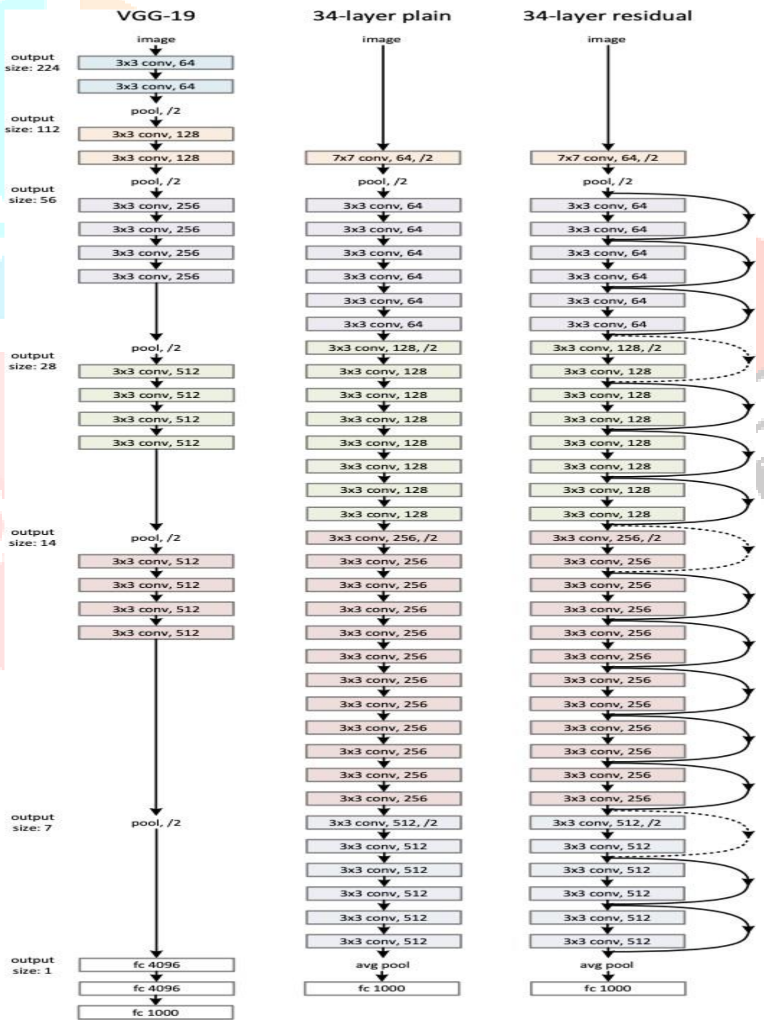


Figure 3 shows example network architectures for ImageNet. On the left is the VGG-19 model [41], which has 19.6 billion FLOPs and is used as a reference. In the middle is a plain network with 34 parameter layers, requiring 3.6 billion FLOPs. On the right is a residual network with the same number of parameter layers (34), also with 3.6 billion FLOPs. The dotted shortcut connections in the residual network increase the dimensionality. Additional details and other variants are provided in Table 1.

Residual Network: Starting with the plain network described earlier, we introduce shortcut connections (as shown in Fig. 3, right), transforming it into a residual network. When the input and output dimensions are the same, we can directly use identity shortcuts (Eqn. (1)), represented by solid line shortcuts in Fig. 3. However, when the dimensions change (as indicated by the dotted line shortcuts in Fig. 3), there are two options:

- **Option A:** The shortcut still performs an identity mapping, with extra zero-padding added to accommodate the increased dimensions. This option does not introduce any additional parameters.
- **Option B:** The shortcut uses a projection (Eqn. (2)) to match the dimensions, typically implemented with 1×1 convolutions.

In both options, when the shortcuts span across feature maps of different sizes, they are applied with a stride of 2.

3.4. Implementation

Our ImageNet implementation follows the methods outlined in [21, 41]. We begin by resizing the image, randomly selecting its shorter side from the range [256, 480] for scale augmentation [41]. A 224×224 crop is then randomly sampled from the image, including its horizontal flip, with the per-pixel mean subtracted [21]. The standard color augmentation described in [21] is also applied. Batch normalization (BN) [16] is applied immediately after each convolution layer and before activation, as recommended in [16]. We initialize the weights as in [13] and train both plain and residual networks from scratch. Stochastic Gradient Descent (SGD) is used with a mini-batch size of 256. The learning rate starts at 0.1 and is reduced by a factor of 10 when the error stops improving. The models are trained for up to 60×10^4 iterations. A weight decay of 0.0001 and momentum of 0.9 are used, and we do not apply dropout [14], in line with the approach in [16]. During testing, for comparison, we use the standard 10-crop testing method [21]. For optimal results, we employ the fully-convolutional architecture from [41, 13], averaging the scores across multiple image scales (resizing the shorter side to values in {224, 256, 384, 480, 640}).

4. Experiments

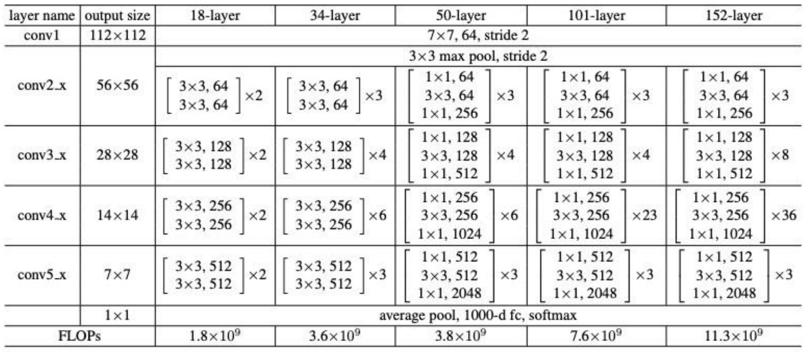
4.1. ImageNet Classification

We evaluate our method on the ImageNet 2012 classification dataset [36], which contains 1000 classes. The models are trained using 1.28 million training images and tested on 50,000 validation images, with the final results reported on 100,000 test images by the test server. Both top-1 and top-5 error rates are used for evaluation.

Plain Networks:
We begin by evaluating 18-layer and 34-layer plain networks. The 34-layer plain network is shown in Fig. 3 (middle). Throughout the training process, the 34-layer plain network consistently exhibits higher training error, even though the solution space for the 18-layer network is a subset of the 34-layer network's solution space.

We believe that this optimization challenge is not due to vanishing gradients, as these networks are trained with batch normalization (BN) [16], which ensures that
forward-propagated signals maintain non-zero variance. We also confirm that the

gradients during backpropagation have healthy norms with BN, so neither the forward nor backward signals



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Table 1. Architectures for ImageNet. Building blocks are shown in brackets (see also Fig. 5), with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2.

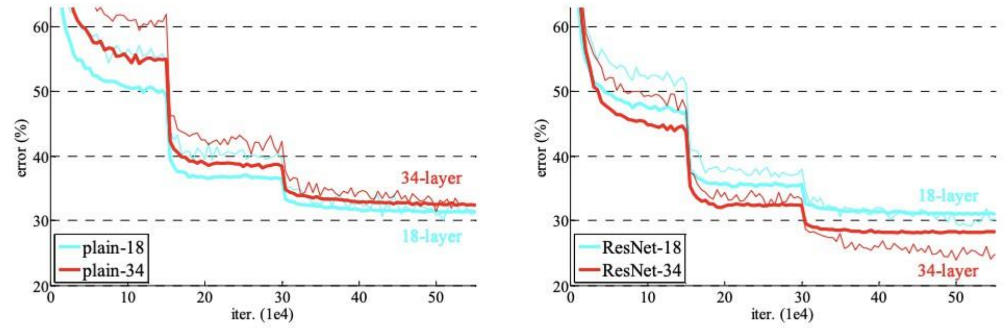


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

vanish. Although the 34-layer plain network still achieves competitive accuracy (as shown in Table 3), we suspect that the deep plain networks may have exponentially slow convergence rates, which could also affect the 18-layer network, as it follows a similar structure. Detailed architectures are provided in Table 1.

The results in Table 2 show that the deeper 34-layer plain network has a higher validation error than the shallower 18-layer plain network. To better understand this, we compare their training and validation errors during the training process in Fig. 4 (left).

We observe the degradation problem, where the training error reduces. Further investigation into the causes of these optimization challenges will be conducted in future work.

We then assess the performance of 18-layer and 34-layer residual networks (ResNets). These models share the same baseline architecture as the previously described plain networks, with the key difference being the addition of shortcut connections after each pair of 3×3 convolutional layers, as shown in Fig. 3 (right). In this initial comparison (refer to Table 2 and Fig. 4, right), we use identity mappings for all shortcuts and apply zero-padding to handle changes in dimensions (option A), meaning no additional parameters are introduced compared to the plain networks.

From Table 2 and Fig. 4, we draw three key insights. First, residual learning reverses the previous trend — the 34-layer ResNet outperforms the 18-layer ResNet by 2.8%. More significantly, the deeper ResNet has a much lower training error and maintains strong generalization to validation data. This suggests that residual learning effectively tackles the degradation problem and enables performance improvements with increased depth.

Second, when compared to the plain 34-layer network, the 34-layer ResNet achieves a 3.5% reduction in top-1 error (as shown in Table 2), which is directly linked to the drop in training error (see Fig. 4, right vs. left). This clearly demonstrates the power of residual learning in training deeper architectures successfully.

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	$8.43^{\dagger}$
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

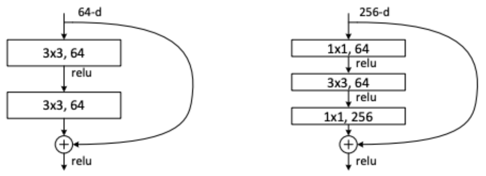


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

Finally, it's worth mentioning that while the 18-layer plain and residual networks achieve similar accuracy levels (as shown in Table 2), the 18-layer ResNet trains faster (refer to Fig. 4, right versus left). When the network isn't very deep—like the 18-layer case—the standard SGD optimizer is still effective for training the plain version. In such scenarios, the ResNet mainly improves the optimization by speeding up early-stage convergence.

Identity vs. Projection Shortcuts:

Earlier results show that identity shortcuts, which introduce no additional parameters, are helpful for training. To explore this further, we analyze the use of projection shortcuts (as defined in Eqn. (2)). Table 3 presents three configurations:

- (A) using zero-padding for increasing dimensions while keeping all shortcuts parameter-free (as in Table 2 and Fig. 4, right);
- (B) applying projection shortcuts only where dimensions increase, while other shortcuts remain as identity mappings;
- (C) using projection shortcuts for all connections.

Table 3 indicates that all three methods outperform the plain network by a significant margin. Option B performs slightly better than A, likely because the zero-padded dimensions in A don't benefit from residual learning. Option C shows a slight edge over, possibly due to the extra parameters from its thirteen projection shortcuts. However, the relatively small differences between A, B, and C suggest that using projection shortcuts everywhere isn't necessary to resolve the degradation issue. Therefore, to keep the models more efficient in terms of memory and computational cost, the rest of the paper favors identity shortcuts—especially when working with bottleneck-style architectures introduced later.

Deeper Bottleneck Architectures :

To build deeper neural networks for ImageNet while keeping training time manageable, the standard residual block is redesigned into a **bottleneck structure**. Instead of the traditional two-layer configuration, each residual function FF now includes **three layers**: a 1×1 convolution to reduce dimensions, a 3×3 convolution for the main computation, and another 1×1 convolution to restore the original dimensions. This approach keeps the computational cost similar to the simpler block while making the middle layer the actual "bottleneck." **Identity shortcuts**, which don't add parameters, are especially valuable in this architecture. If these are replaced with projection shortcuts (as shown in Fig. 5, right), the time and memory costs **double** because the projections span high-dimensional layers. Thus, identity shortcuts allow for a more efficient and lightweight design.For example, by replacing each 2-layer block in the original 34-layer network with a 3-layer bottleneck block, they create a **50-layer ResNet** (refer to Table 1), using projection shortcuts (option B) only where dimensions increase. Despite being deeper, this model has only **3.8 billion FLOPs**, maintaining efficiency.

They also build even **deeper models**: a **101-layer** and a **152-layer** ResNet, each by adding more bottleneck blocks (Table 1). Impressively, the **152-layer ResNet**, with **11.3 billion FLOPs**, is still computationally lighter than VGG-16 and VGG-19 (15.3 and 19.6 billion FLOPs, respectively). These deeper ResNets outperform the 34-layer model by a significant margin (as seen in Tables 3 and 4), and no signs of the **degradation problem** are observed. Instead, increasing depth results in substantial accuracy improvements across all evaluation metrics.

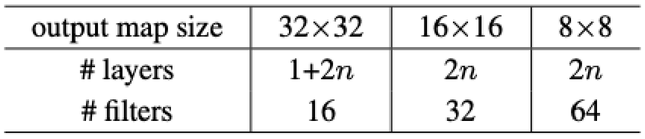
When compared with other top methods, even the baseline **34-layer ResNet** shows very competitive performance. The **152-layer ResNet** achieves a top-5 validation error of **4.49%**—beating previous **ensemble-based methods** (Table 5). By ensembling six models of various depths (including two 152-layer models), they achieve a **3.57% top-5 test error**, winning **1st place in ILSVRC 2015**.

4.2. CIFAR-10 and Analysis

We carried out additional experiments using the **CIFAR-10 dataset** [20], which contains 50,000 training images and 10,000 test images across 10 classes. The goal of these experiments wasn't to beat state-of-the-art performance, but rather to explore how **very deep networks behave**. To keep things simple and focused, we deliberately used **basic network structures**.

Both the plain and residual models are based on the architectures shown in **Fig. 3 (middle and right)**. Input images are **32×32 pixels**, and each has its **mean pixel value subtracted** beforehand. The network starts with a **3×3 convolutional layer**. This is followed by a sequence of **6n convolutional layers** (each using 3×3 filters), applied over feature maps of size **32, 16, and 8**, with **2n layers for each resolution**. The number of filters used for each stage is **16, 32, and 64**, respectively. Downsampling is done using **convolutions with a stride of 2**.

The network ends with **global average pooling**, followed by a **fully connected layer with 10 outputs**, and a **softmax layer** for classification. In total, each model has **6n + 2 learnable layers**. A table that follows provides a summary of the complete architecture.



output map size	32×32	16×16	8×8
# layers	1+2n	2n	2n
# filters	16	32	64

When using shortcut connections in our models, they are applied to every pair of 3×3 convolutional layers—resulting in a total of 3n shortcut paths. For all experiments on this dataset, we opt for identity shortcuts only (option A). As a result, the residual networks have identical depth, width, and parameter count as their plain counterparts.

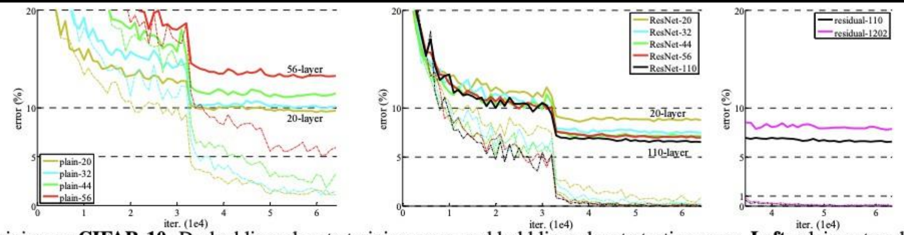


Figure 6. Training on **CIFAR-10**. Dashed lines denote training error, and bold lines denote testing error. **Left**: plain networks. The error of plain-110 is higher than 60% and not displayed. **Middle**: ResNets. **Right**: ResNets with 110 and 1202 layers.

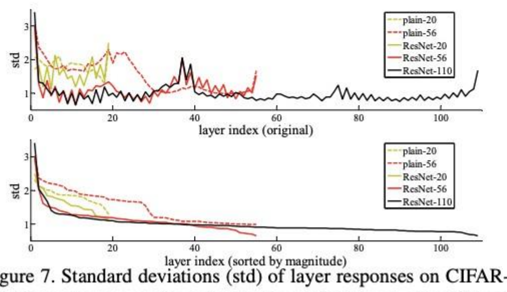


Figure 7. Standard deviations (std) of layer responses on CIFAR-10. The responses are the outputs of each 3×3 layer, after BN and before nonlinearity. **Top**: the layers are shown in their original order. **Bottom**: the responses are ranked in descending order.

training data	07+12	07++12
test data	VOC 07 test	VOC 12 test
VGG-16	73.2	70.4
ResNet-101	**76.4**	**73.8**

Table 7. Object detection mAP (%) on the PASCAL VOC 2007/2012 test sets using **baseline** Faster R-CNN. See also Table 10 and 11 for better results.

metric	mAP@.5	mAP@[.5, .95]
VGG-16	41.5	21.2
ResNet-101	**48.4**	**27.2**

Table 8. Object detection mAP (%) on the COCO validation set using **baseline** Faster R-CNN. See also Table 9 for better results.

method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Table 6. Classification error on the **CIFAR-10** test set. All methods are with data augmentation. For ResNet-110, we run it 5 times and show "best (mean±std)" as in [43].

We train these models with weight decay set to 0.0001 and momentum of 0.9, using the initialization strategy from [13] and batch normalization [16], but no dropout. The training is done using a mini-batch size of 128 across two GPUs. Training begins with a learning rate of 0.1, which is reduced by a factor of 10 at 32k and 48k iterations, and training stops at 64k iterations, based on performance from a 45k/5k train/validation split.

For **data augmentation**, we apply a simple approach from [24]: pad 4 pixels on each side of the image, then take a **random 32×32 crop** from the padded image or its **horizontal flip**. During testing, only the **original, unaltered 32×32 image** is evaluated (no augmentation).

We experiment with different values of n = {3, 5, 7, 9}, which correspond to networks with 20, 32, 44, and 56 layers. As shown in Fig. 6 (left), deeper plain networks perform worse—they experience increased training error as depth increases. This mirrors the optimization difficulties observed on ImageNet (Fig. 4, left) and MNIST [42], indicating that deeper plain nets constantly struggle with training. In contrast, Fig. 6 (middle) shows that ResNets handle increasing depth effectively, just like in the ImageNet experiments (Fig. 4, right). They successfully overcome optimization challenges and achieve better accuracy as they go deeper.

We also push further by setting **n = 18**, building a **110-layer ResNet**. Initially, the default learning rate of 0.1 proved too high to kick off proper training, so we **started with a** smaller rate of 0.01 until the training error dropped below 80% (around 400 iterations). After that, we resumed with 0.1 and followed the usual training schedule. This 110-layer model converged well (see Fig. 6, middle), and despite having fewer parameters than other deep-but-thin networks like FitNet [35] and Highway [42], it achieved competitive performance (6.43% error, Table 6), placing it among the top results.

Layer Response Analysis:
Figure 7 presents the standard deviations of outputs from each 3×3 convolutional layer—these are measured after batch normalization and before applying non-linear operations like ReLU or addition. In the case of ResNets, this provides insight into the strength of the residual mappings themselves. The results show that residual networks tend to produce **smaller response magnitudes** compared to plain networks. This supports the hypothesis from Section 3.1 that residual functions are typically closer to zero than non-residual ones. Furthermore, deeper ResNets display even smaller response magnitudes. When comparing networks like ResNet-20, ResNet-56, and ResNet-110 in Figure 7, it's evident that **the more layers a ResNet has, the less each individual layer alters the signal**—suggesting subtler transformations at deeper levels.

Pushing Beyond 1000 Layers:
We also tested extremely deep models, building a **1202-layer ResNet** by setting n =
200. This model followed the same training strategy as earlier ones. Impressively, it showed **no optimization challenges**, successfully reaching **under 0.1% training error**
(Figure 6, right). Its test accuracy was also solid, yielding a **7.93% error rate** (Table 6).

However, some concerns arise with this level of depth. The test performance of the 1202-layer model was actually **worse than that of the 110-layer network**, even though both achieved very low training error. We suspect this is due to **overfitting**—the
1202-layer model (with 19.4 million parameters) may be **excessive for a relatively small dataset** like CIFAR-10.

In contrast to other top-performing models on CIFAR-10, which often rely on stronger regularization techniques like maxout [10] or dropout [14], we deliberately chose not to use these. Instead, we relied on designing deep but narrow architectures as a form of implicit regularization to keep the focus on solving optimization challenges. That said, integrating more advanced regularization methods could further improve performance—this is a direction we plan to investigate in future work.

4.3. Object Detection on PASCAL and MS COCO

Our approach also shows strong generalization across different recognition tasks. As seen in Tables 7 and 8, we report baseline object detection results on PASCAL VOC 2007 and 2012 [5], as well as COCO [26], using Faster R-CNN [32] as the detection framework. In these experiments, we focus on the performance gain achieved by replacing VGG-16 [41] with ResNet-101. Since the detection pipeline remains identical for both models (details in the appendix), any improvements observed are purely due to the superior feature representations of ResNet.

Notably, on the more challenging COCO dataset, this switch results in a 6.0% absolute increase in the COCO standard metric (mAP@[.5, .95]), which corresponds to a 28% relative improvement—entirely attributable to the effectiveness of deep residual learning. Leveraging ResNet architectures, we secured first place in multiple categories at the ILSVRC & COCO 2015 competitions, including ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation. Further details are available in the appendix.

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2):157–166, 1994.
[2] C. M. Bishop. Neural networks for pattern recognition. Oxford university press, 1995.
[3] W. L. Briggs, S. F. McCormick, et al. A Multigrid Tutorial. Siam, 2000.
[4] K. Chatfield, V. Lempitsky, A. Vedaldi, and A. Zisserman. The devil is in the details: an evaluation of recent feature encoding methods. In BMVC, 2011.
[5] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. IJCV, pages 303–338, 2010.
[6] S. Gidaris and N. Komodakis. Object detection via a multi-region & semantic segmentation-aware cnn model. In ICCV, 2015.
[7] R. Girshick. Fast R-CNN. In ICCV, 2015.
[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In CVPR, 2014.
[9] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In AISTATS, 2010.
[10] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. arXiv:1302.4389, 2013.
[11] K. He and J. Sun. Convolutional neural networks at constrained time cost. In CVPR, 2015.
[12] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In ECCV, 2014.
[13] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In ICCV, 2015.
[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by preventing coadaptation of feature detectors. arXiv:1207.0580, 2012.
[15] S. Hochreiter and J. Schmidhuber. Long short-term memory. Neural computation, 9(8):1735–1780, 1997.
[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In ICML, 2015.
[17] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. TPAMI, 33, 2011.
[18] H. Jegou, F. Perronnin, M. Douze, J. Sanchez, P. Perez, and C. Schmid. Aggregating local image descriptors into compact codes. TPAMI, 2012.
[19] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093, 2014.
[20] A. Krizhevsky. Learning multiple layers of features from tiny images. Tech Report, 2009.
[21] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.
[22] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L.
D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural computation, 1989.
[23] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Muller. Efficient backprop. ¨ In Neural Networks: Tricks of the Trade, pages 9–50. Springer, 1998.
[24] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu. Deeplysupervised nets. arXiv:1409.5185, 2014.
[25] M. Lin, Q. Chen, and S. Yan. Network in network. arXiv:1312.4400, 2013.
[26] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C.
L. Zitnick. Microsoft COCO: Common objects in ´ context. In ECCV. 2014.
[27] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In CVPR, 2015.
[28] G. Montufar, R. Pascanu, K. Cho, and Y. Bengio. On the number of ´ linear regions of deep neural

networks. In NIPS, 2014.
[29] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In ICML, 2010.
[30] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In CVPR, 2007.
[31] T. Raiko, H. Valpola, and Y. LeCun. Deep learning made easier by linear transformations in perceptrons. In AISTATS, 2012.
[32] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In NIPS, 2015.
[33] S. Ren, K. He, R. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. arXiv:1504.06066, 2015.
[34] B. D. Ripley. Pattern recognition and neural networks. Cambridge university press, 1996.
[35] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. In ICLR, 2015.
[36] 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. arXiv:1409.0575, 2014.
[37] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv:1312.6120, 2013.
[38] N. N. Schraudolph. Accelerated gradient descent by factor-centering decomposition. Technical report, 1998.
[39] N. N. Schraudolph. Centering neural network gradient factors. In Neural Networks: Tricks of the Trade, pages 207–226. Springer, 1998.
[40] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. In ICLR, 2014
[41] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.
[42] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. arXiv:1505.00387, 2015.
[43] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. 1507.06228, 2015.
[44] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR, 2015.
[45] R. Szeliski. Fast surface interpolation using hierarchical basis functions. TPAMI, 1990.
[46] R. Szeliski. Locally adapted hierarchical basis preconditioning. In SIGGRAPH, 2006.
[47] T. Vatanen, T. Raiko, H. Valpola, and Y. LeCun. Pushing stochastic gradient towards second-order methods–backpropagation learning with transformations in nonlinearities. In Neural Information Processing, 2013.
[48] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms, 2008.
[49] W. Venables and B. Ripley. Modern applied statistics with s-plus. 1999.
[50] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional neural networks. In ECCV, 2014.

IJCRT