**Summary of the Paper**

**Deep Residual Learning for Image Recognition**

**Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun**

**Microsoft Research**

## Content Overview

We start with a short summary of the chapters in the paper. The following is an introduction to deep convolutional networks, a mention of the related work used for the paper, an overview about residual learning, the used architecture and implementation and performance experiments on ImageNet 2012 as well as CIFAR-10.

### Introduction

Each layer in a neural network detects a specific detail of the input image, depending on the filter kernel. Creating a deep neural network out of multiple layers stacked together allows the network to understand the correlation between those small details, leading to an abstraction of low-, mid- and high-level features. The paper refers to several sources, stating that a higher network depth is crucial for a better performance on the task of image recognition.

Surprisingly, the accuracy of deeper networks suffers from degradation. Not only on the *testing* data (which might be explained by overfitting), but also on the *training* data. Figure 1 demonstrates this behavior for the CIFAR-10 dataset

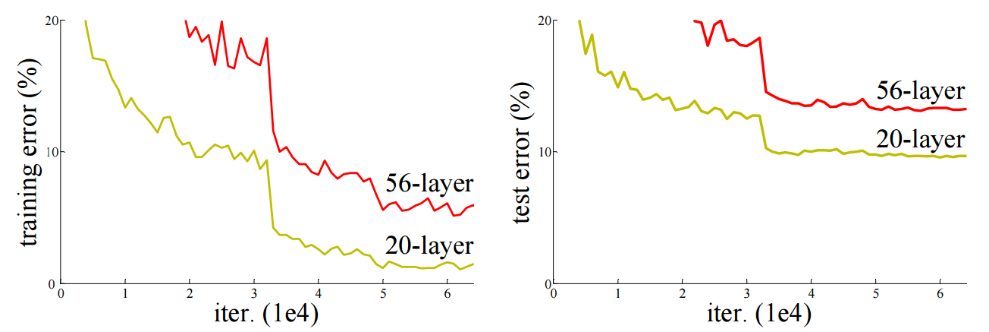


Figure 1. Training (left) and test (right) error for CIFAR-10 with a 56- and a 20-layer network.

Since it is possible to construct a solution for deeper networks, by copying the layers from a shallow network and only adding identity mappings to it, one might assume that the performance should be the same overall. The empirically proven deviation in performance suggests, that optimization of different networks is not similarly easy and that current optimization strategies are unable to find a comparably good solution in feasible time.

To compensate this the paper introduces a *deep residual learning* framework with a mapping F(x) + x. This formulation is achieved by a shortcut connection, that skips two convolutions, performs an identity mapping and is added to the output of the convolutions. As this simple identity shortcut does not add any parameters, the computational complexity is maintained, while leading to much better results. The building block for the deep residual learning network can be seen in figure 2.

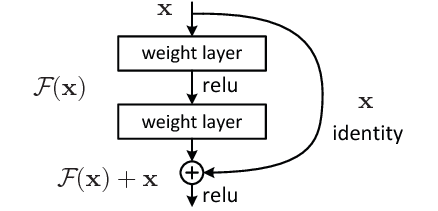


Figure 2. Building block with identity shortcut.

### Related Work

In this chapter the paper mentions the two underlying concepts that lead to the idea of the deep residual learning framework.

*Residual representations* are used in a variety of image recognition approaches. E.g. *VLAD*, which encodes the representation by residual vectors with respect to a dictionary. The *multigrid method*, which is used for solving Partial Differential Equations, splits the system into subproblems at multiple scales, where each subproblem represents the residual solution between a coarser and a finer scale.

*Shortcut connections* have been studied in several different applications. The most notable development was done in parallel to this paper, namely *highway networks* with gated shortcut connections. Such a network controls “the flow” over the shortcuts by additional parameters, even allowing the connection to be fully closed which leads to a non-residual function.

### Deep Residual Learning

In order to tackle the degradation problem mentioned in the introduction, a reformulation of the formula F(x) := H(x) – x to F(x)+x is performed. This is possible, because if we can hypothesize that multiple nonlinear layers can asymptotically approximate complicated functions, then we hypothesize that these layers can asymptotically approximate the residual function. This enables solvers to converge the weights of multiple nonlinear layers towards zero in order to approximate identity mappings. However, the possibility of ideal identity mappings in real world examples is very little.

When using residual learning, two equations need to be considered. First there is y = F(x, {Wi}) + x. But this formula only holds if x and F have equal dimensions. To address this a linear projection can be performed on x to match the dimensions. This leads to the equation „y = F(x, {Wi}) + Wsx“.

The plain and residual networks from this paper follow a very similar architecture where the only difference is the usage of a shortcut connection for residual networks. This is visualized in Figure 3. On the right side is the residual network, where the arrows implicate the shortcut connections. The dotted arrow symbolizes that a linear projection is used in order to match the dimensions.

On the left side is the plain network without any shortcuts.

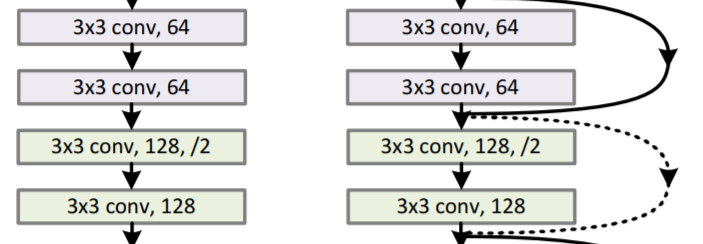


Figure 3. Symbolized plain and residual network architecture

#### Residual architecture for ImageNet

Bevor the actual network architecture is described, the smaller side of the images is randomly set to a value between 256 and 480 pixels in order to perform a scale augmentation. Afterwards a random crop with the dimension of 224x224 or its horizontal flipped image is selected. After performing a convolution or before activation a batch normalization is used. As last point before training the weights are initialized. As an optimizer a stochastic gradient descent with a mini-batch size of 256 is used. During training the learning rate is adapted in order to escape the error plateaus. At 32k and 48k iterations the learning rate, originally 0.1, is multiplied by a factor of 0.1. Additionally, the optimizer uses a weight decay of 0.0001 and a momentum of 0.9. After 60k iterations the training was stopped.

To begin, both networks use a 7x7 convolution with 64 Filters and a stride of 2, followed by a pooling also with a stride of 2. Afterwards Layers of 3x3 convolutions with different features sizes are added. In the end an average pooling is performed and finished by a fully connected layer with a class-size of 1000. In Table 1 the exact number of layers for the different residual nets is displayed:

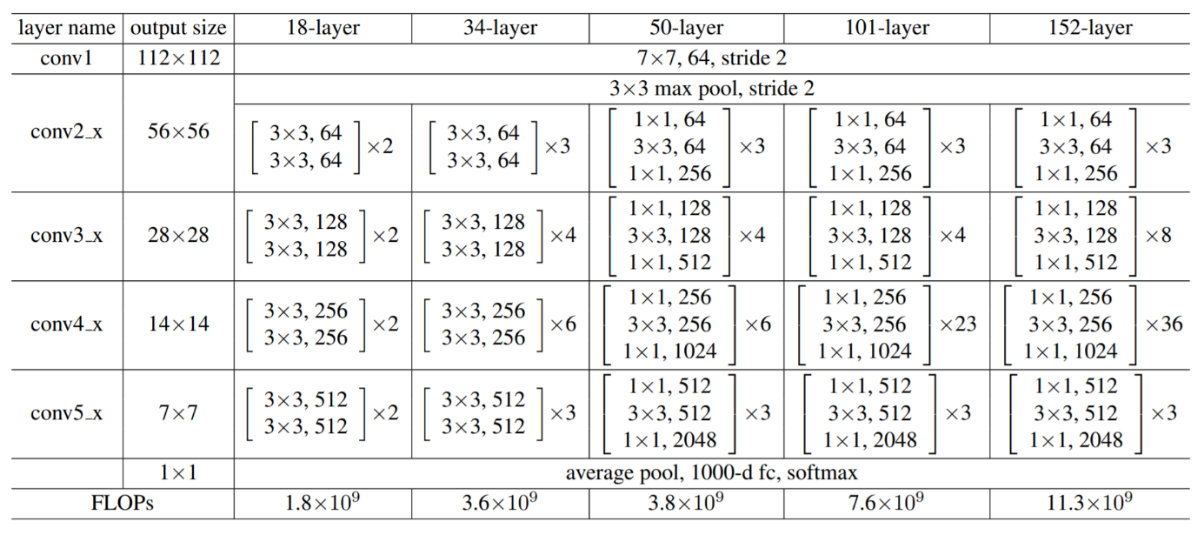


Table 1. Architectures of the different residual networks.

It’s noticeable that the architecture of the single layers changes when switching from a 34-layer network to one with 50-layers. This is done to reduce computation time. The first 1x1 convolution decreases dimensions, to fasten up the 3x3 bottleneck convolution. Afterwards the dimension is restored by another 1x1 convolution. These two different architectures are described as basic- and bottleneck-blocks and are shown in Figure 4.

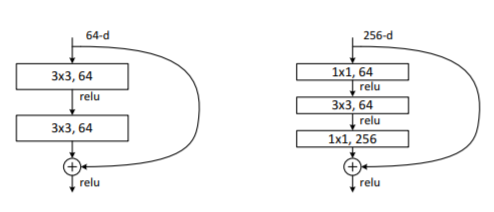


Figure 4. Comparison between Basic- and Bottleneck-block.

#### Residual architecture for CIFAR-10

For CIFAR-10 different residual networks were used, but the architecture is like the ones used for ImageNet. The inputs are images of size 32x32 with the per-pixel mean subtracted, but nothing else is being done. Downsampling is not done in the CIFAR-10 residual networks. Hence the stride of the first 3x3 convolution is 1 and the pooling is removed. Additional only basic-blocks are used to build the network. As last point, the fully-connected-layer is reduced to a class-size of 10.

Totally six networks were configured (ResNet-20,32,44,56,110,1202). Computing the needed number of layers can be computed with Table 2, where n represents (3,5,7,9,18,200) for the stated residual networks.

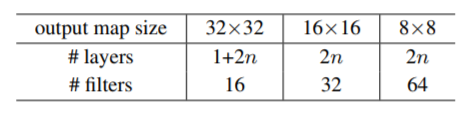


Table 2. Computation of layers in CIFAR-10.

## Experiments

### ImageNet 2012

The presented architecture is evaluated on the ImageNet 2012 classification dataset, including 1.28 million training images over 1000 distinct classes and 100 thousand test images. Training is performed with four different network types: one with 18- and one with 34-layers, each of them with (*residual*) and without (*plain*) the shortcut connection.

The experimental results on the plain networks clearly demonstrate the degradation problem. Although the possible solution space of the deeper 34-layer network is much larger than the one of the 18-layer network, its validation and training error is consistently higher than its counterpart from the shallower network.

Evaluation of the residual network shows that we now achieve a better result with the deeper network. Additionally, the 34-layer network generalizes more leading to a reduced validation error. The residual networks also tend to converge faster compared to their plain counterparts.

The above observations can be seen in figure 3.

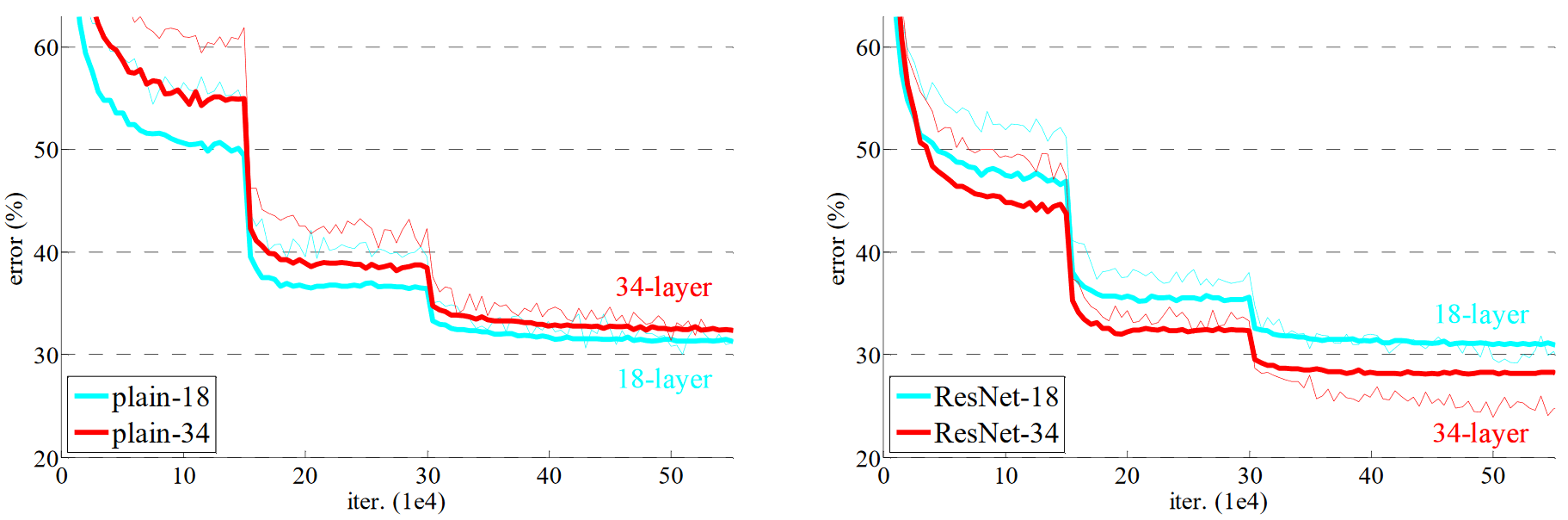


Figure 3. Training on ImageNet dataset. Thin curves show training- and bold curves show validation error. Plain (left) and residual (right) networks have same number of parameters.

### CIFAR-10

Similar results can be observed for the CIFAR-10 dataset, suggesting that the optimization behavior is independent from a particular dataset. This dataset consists of 50k training and 10k test images equally spread across 10 classes. Analysis of multiple training runs with a different network depth shows a behavior analogous to the ImageNet dataset.

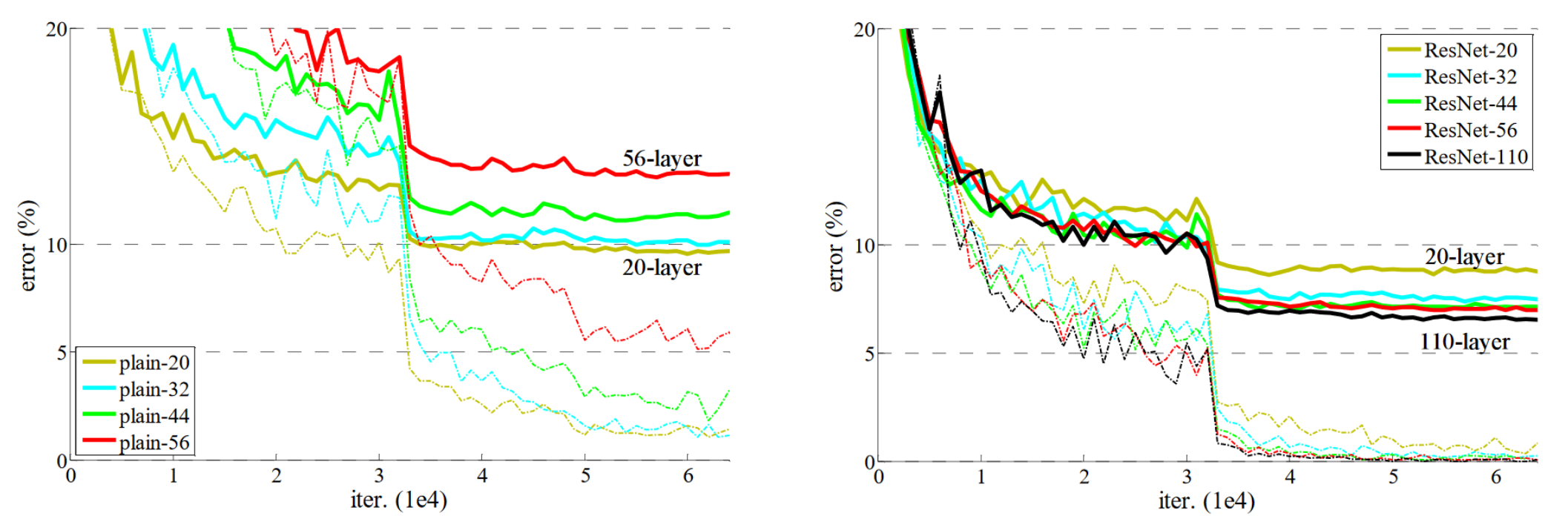


Figure 4. Training on CIFAR-10. Thin curves show training- and bold curves show testing error.

## Ease of Implementation

As a part of our assignment we implemented ResNet20 and ResNet32 from this paper and compared our results to the one in this paper.

### Implementation

At first implementing the stated ResNets was more complicated than we assumed. Especially because ResNet20 and ResNet32 are used for the CIFAR-10 Image-Set which uses a slightly different approach, then the residual nets described the main implementation part of the paper. It took a bit to figure out, which block-types were used, and which components were obsolete in the new residual nets. But once the first residual net was finished, it was to quite easy to add a different residual net.

A different problem was the runtime. In the paper 64 \*10^4 Iterations were performed. Even though we had a fast System available this was out of our reach, since the computation time, especially for deeper networks increased to multiple hours. Hence, we decided to compare the results of the first 30\*10^4 Iterations, which are shown in #TODO(Add Picture-reference). This however maid the exact use of the learning-rate-scheduler impossible, since it was used at 32\*10^4 and 48\*10^4 iterations. We decided to try a smaller approach and reduced the learning rate by factor of 0.1 at 16\*10^4 and 24\*10^4 iterations. Assuming we would get a better result short-term. Test showed this #TODO (ergebnis eintragen.)

### Success of Implementation

Overall the implementation of the two residual networks was very successful. Our results were almost identical with the ones from the paper. Additional we were able to show that increasing layers in residual networks also increase accuracy, while the opposite occurs in plain convolutional networks, where more layers lead to a decrease in accuracy.

One point we could not achieve was the overall amount of iterations because the computation took too long. We also stopped with two working neural networks, since additional layers also meant additional computation time.

## Comparison to PyTorch Implementation

To have an architectural comparison for the residual networks trained on ImageNet we decided to compare the implementation from PyTorch, since both sources implanted the same ResNets (18, 34, 50, 101, 152). After a quick comparison we noticed the architectures were almost identical. Both implementations used the same preprocessing methods, the same basic- and bottleneck-block architecture, as well as the same number of convolutions in each network. They also make the same switch from basic- to bottleneck-block from ResNet32 to ResNet50.[[1]](#footnote-1)

The only difference we could spot was that PyTorch offers the possibility to zero-initialize the last batch norms in each residual branch. As a result, the residual branches start with zeroes and each residual block behaves like and identity, which should slightly increase the accuracy of the model[[2]](#footnote-2). This technique however was release after our paper and hence cannot be counted as an actual difference.[[3]](#footnote-3)1

1. <https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py> [↑](#footnote-ref-1)
2. https://arxiv.org/abs/1706.02677 [↑](#footnote-ref-2)
3. [↑](#footnote-ref-3)