# Deep Residual Learning for Image Recognition

## 1 Citation

He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

https://arxiv.org/pdf/1512.03385.pdf

## 2 Abstract

To make it easier to train deep neural networks, we force each layer to learn a residual function of its previous layer. We then trained an ensemble of networks that set state of the art on ImageNet 2015 classification, localization and detection. We also won COCO detection and segmentation.

## 3 Introduction

Vanishing and exploding gradients were seen as a challenge to training deep neural networks, but normalized initialization and batch normalization has addressed the challenge. However, if you train a deeper model in practice, it can sometimes get higher training error than a shallower model. We call this the degradation problem, and it's not due to overfitting/underfitting, but due to a limitation of our optimization algorithms.

To address degradation, we take the following approach. Suppose we want to learn $H(x)$. We make the layer learn $F(x) = H(x) - x$. Then, we use a shortcut connection from the input layer (i.e. $x$) with a sum to compute $F(x) + x$. The advantage to this approach is that, if the layer needs to learn the identity function, all it must do is push $F(x)$ to zero.

We demonstrate that the degradation problem exists and we train deep residual networks to win ImageNet and COCO.

## 4 Related Work

VLAD and Fisher Vectors are a kind of residual representation, and they are powerful shallow techniques. When solving PDEs, residual modeling works well.

Inception, Highway networks, and many others use shortcut connections. In our case, we always use identity shortcuts, which are simple.

## 5 Deep Residual Learning

Suppose we want to learn $H(x)$ with some stacked layers. Let's learn $F(x) = H(x) - x$ instead and just use a shortcut connection to compute $F(x) + x$. This alleviates the degradation problem because it's easier to learn the zero function than to learn the identity function. In practice, we find that occasionally layers do need to learn the identity (or something similar to the identity), so making them learn zero (or something near zero) helps.

The residual building block takes the form:

$$\mathbf{y} = F(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}$$

where the $W_s$ is a linear projection to make $\mathbf{x}$ match the dimension of $F$. If they already match, we can just omit $W_s$.

Our baseline is a 34-layer network that resembles VGG-19. Basically, it's a bunch of $3 \times 3$ convolutional layers that ends with average pooling, a fully connected layer, and softmax. We have a 34-layer residual network that is similar, but every two or three convolutional layers is treated as a residual block. Our baseline uses 3.6B multiply-adds, while VGG-19 uses 19.6B. For shortcut connections where the dimensions must increase, we tried both the projection matrix $W_s$ and just plain-old zero-padding.

## 6 Experiments

We train a 18-layer plain network and 34-layer plain network. The latter has higher training error, but has backward gradients with healthy norms. This shows that the degradation problem exists.

The 34-layer ResNet beats the 18-layer ResNet. This shows the degradation problem is addressed. The 34-layer ResNet also performs better than the plain networks and 18-layer ResNet, which shows that ResNets perform well. The 18-layer ResNet converges faster than the 18-layer plain net.

We found that using zero-padding for increased dimensions is fine, but that projection matrix $W_s$ is a little better. In practice, you may want to avoid using the projection matrix because it uses a lot more computation. In our case, we use the projection matrix when we increase dimension.

Next, we tried to train a deeper ResNet. To avoid blowing up computation, we made the layer stack inside a residual block consist of a $1 \times 1$ conv layer, a $3 \times 3$ conv layer, and another $1 \times 1$ conv layer. The first layer reduces the dimensionality, the second does convolution, and the third increases dimensionality.

We made a 50, 101, and 152-layer ResNet (the 152 layer ResNet is still less compute-intensive than VGG, with 11.3B ops). The 152-layer ResNet beats every single ensemble ever run on ImageNet. Making an ensemble of ResNets gives 3.57% top-5 error.

We studied CIFAR-10 to see how deep we could make our models. The plain net takes $32 \times 32$ images with per-pixel mean subtracted. Then we stack a bunch of $3 \times 3$ conv layers and end with global average pooling, a fully connected layer, and softmax. For the ResNets, we use identity shortcuts everywhere to ensure we have the same depth and same number of parameters. We train with SGD + momentum with scheduled learning rate drops and weight decay on two GPUs. For data augmentation, we pad the image and take crops.

The degradation problem occurs again but the ResNets overcome it. ResNets have small responses in some layers, indicating that the hypothesis of learning the zero function is true.

We trained a 1202 layer network which we were able to optimize, but which did worse on the testing set (probably overfitted). We did not use dropout/maxout but you might get better results if you use them.

For COCO, we use Faster R-CNN but replace VGG-16 with ResNet-101. We won.

**Note: The appendices discuss how ResNets were used for object detection**