Comments on Resnet
David Levinthal
Microsoft Cloud Services Infrastructure
Performance

V0.95

The Residual Network image processing algorithm (https://arxiv.org/abs/1512.03385) has become a standard for evaluation hardware performance on CNNs in recent times. Its accuracy and flexibility for retraining make it particularly useful. One of the insights in the design of the Resnet convolutional network is the realization that sometimes less is more. This is to say that the networks sub sections should be able to produce an identity output for cases where more depth is not adding to the quality of the result. From the paper this is shown diagrammatically as follows:

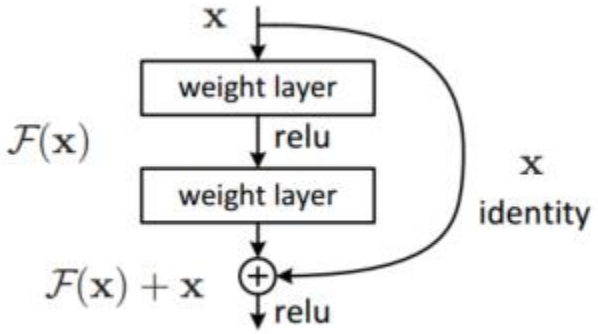


Figure 2. Residual learning: a building block.

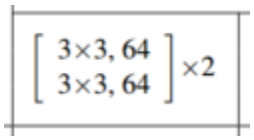
The identity term is also referred to as a shortcut.

The algorithm takes a standard 224 X 224 X 3 array of pixels as input, the 3 layers giving the RGB values. There are a standardized set of layers that are used in the algorithm referred to as resnet18 through resnet 152. They are characterized in the table (from the paper) shown below:

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.

The table lists the convolution layers in blocks of either 2 or 3 convolutions followed by the number of times a block is repeatedly executed. Thus, the top block of the resnet18 network:



Signifies a block of a 3X3 kernel used in 64 filters invoked 2 times and then that in turn is invoked 2 times. The total number of layers called out in the table header can be easily computed from the column. For example, resnet18 gets to 18 layers from the 4 blocks of 4 convolutions plus 1 for the top convolution and then 1 more for the fully connected layer at the bottom (or the pooling layer at the top)

The input dimension decreases by 2X as one progresses through the table while the filter depths increase by 2X. This results in the number of FP operations per block remaining approximately constant. The number of FP operations in a convolution can be computed as
FP ops = (2*Filter_width*Filter_width+1)*Input_depth*Output_w*output_w*Output_depth
using this and calculating the number of FP operations indicated in the table rows for conv2_x through conv5_x, then adding the number for the first 7 X 7 convolution I get values approximately 2X larger than shown in the table below. Also included in the table are the weight counts only for those layers described in “table 1” of the original paper computed using the expression (for square convolutions)
Weight count = input_depth*filter_width*filter_width*filter_depth

	FP operations	Weights
resnet 18	3.8 X 10**9	11 X 10**6
resnet 34	7.7 X 10**9	13.9 X 10**6
resnet 50	8.9 X 10**9	20.7 X 10**6
resnet 101	18.3 X 10**9	39.6 X 10**6
resnet 152	27.7 X 10**9	55.2 X 10**6

These computations of FP operations are in excellent agreement with what is observed using the tensorflow profiler on the tf_cnn_banchmark suite running resnet 50. (https://github.com/tensorflow/benchmarks/tree/master/scripts/tf_cnn_benchmarks). The profiler yielded a value of 7.7 billion FP operations for a forward pass through resnet50.
Using the Nvidia NVProf tool one gets a still slightly higher value of 8.4 billion FP operations. Clearly, determining the count of FP operations is not entirely simple.

The network is actually somewhat more complicated than just the elements outlined in the table as there are batch normalization layers and scaling layers and the 1 X 1 conv kernels applied to the shortcut to match the sizes for the recombination. A picture of the first block for resnet50 generated from the cafffe prototxt file is shown below, where “bn” stands for batch normalization and the terms conv, relu, pool and scale are self-explanatory.

ResNet-50


data
conv1
bn_conv1
scale_conv1
conv1_relu
pool1
res2a_branch2a
bn2a_branch2a
scale2a_branch2a
res2a_branch2a_relu
res2a_branch2b
bn2a_branch2b
scale2a_branch2b
res2a_branch2b_relu
res2a_branch1
bn2a_branch1
scale2a_branch1
res2a_branch2c
bn2a_branch2c
scale2a_branch2c
res2a
res2a_relu
res2b_branch2a
bn2b_branch2a

This brings up another small matter that caused me some confusion. The usual expression for calculating the output size of a convolution (or pooling) is:

O = 1 + (W – K +2P)/S

Where W is the width of the input image, K is the width of the kernel, P is the width of the padding applied to both sides and S is the stride.
Looking at this one would think (and many texts on machine learning say this) that (W – K + 2P) must be an even multiple of S. If this is applied to any of the convolutions or poolings in the table one cannot find a value of P which makes this work as the widths are even integers and the kernel widths are odd integers. The answer to this quandary was found in
https://stackoverflow.com/questions/42924324/tensorflows-asymmetric-padding-assumptions

The discussion in that stackoverflow entry and the picture from it reproduced below suggests that ML frameworks use asymmetric padding to solve the issue. However, it appears that different frameworks may handle this differently. This in turn may result in numerical differences between identical networks run on different frameworks.