# DEEP COMPRESSION: COMPRESSING DEEP NEURAL NETWORKS WITH PRUNING, TRAINED QUANTIZATION AND HUFFMAN CODING

**Song Han**
Stanford University, Stanford, CA 94305, USA
songhan@stanford.edu

**Huizi Mao**
Tsinghua University, Beijing, 100084, China
mhz12@mails.tsinghua.edu.cn

**William J. Dally**
Stanford University, Stanford, CA 94305, USA
NVIDIA, Santa Clara, CA 95050, USA
dally@stanford.edu

arXiv:1510.00149v5 [cs.CV] 15 Feb 2016

## ABSTRACT

Neural networks are both computationally intensive and memory intensive, making them difficult to deploy on embedded systems with limited hardware resources. To address this limitation, we introduce "deep compression", a three stage pipeline: pruning, trained quantization and Huffman coding, that work together to reduce the storage requirement of neural networks by $35\times$ to $49\times$ without affecting their accuracy. Our method first prunes the network by learning only the important connections. Next, we quantize the weights to enforce weight sharing, finally, we apply Huffman coding. After the first two steps we retrain the network to fine tune the remaining connections and the quantized centroids. Pruning, reduces the number of connections by $9\times$ to $13\times$; Quantization then reduces the number of bits that represent each connection from 32 to 5. On the ImageNet dataset, our method reduced the storage required by AlexNet by $\mathbf{35}\times$, from 240MB to 6.9MB, without loss of accuracy. Our method reduced the size of VGG-16 by $\mathbf{49}\times$ from 552MB to 11.3MB, again with no loss of accuracy. This allows fitting the model into on-chip SRAM cache rather than off-chip DRAM memory. Our compression method also facilitates the use of complex neural networks in mobile applications where application size and download bandwidth are constrained. Benchmarked on CPU, GPU and mobile GPU, compressed network has $3\times$ to $4\times$ layerwise speedup and $3\times$ to $7\times$ better energy efficiency.

## 1 INTRODUCTION

Deep neural networks have evolved to the state-of-the-art technique for computer vision tasks (Krizhevsky et al., 2012)(Simonyan & Zisserman, 2014). Though these neural networks are very powerful, the large number of weights consumes considerable storage and memory bandwidth. For example, the AlexNet Caffemodel is over 200MB, and the VGG-16 Caffemodel is over 500MB (BVLC). This makes it difficult to deploy deep neural networks on mobile system.

First, for many mobile-first companies such as Baidu and Facebook, various apps are updated via different app stores, and they are very sensitive to the size of the binary files. For example, App Store has the restriction "apps above 100 MB will not download until you connect to Wi-Fi". As a result, a feature that increases the binary size by 100MB will receive much more scrutiny than one that increases it by 10MB. Although having deep neural networks running on mobile has many great

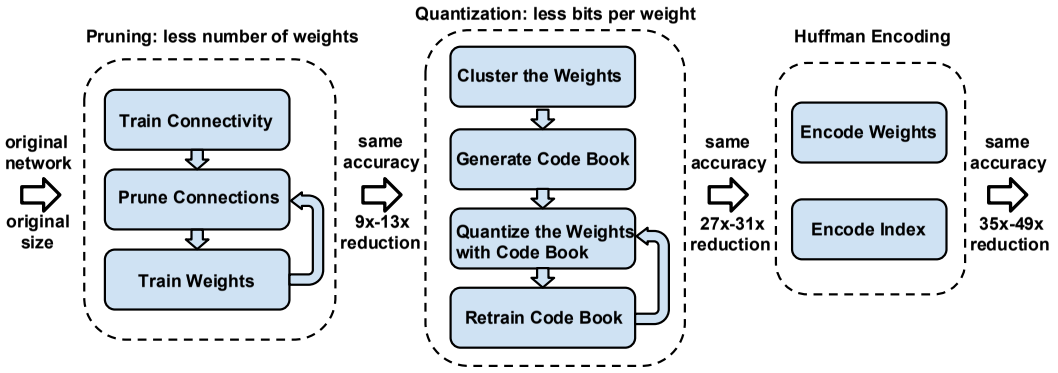


Figure 1: The three stage compression pipeline: pruning, quantization and Huffman coding. Pruning reduces the number of weights by $10\times$, while quantization further improves the compression rate: between $27\times$ and $31\times$. Huffman coding gives more compression: between $35\times$ and $49\times$. The compression rate already included the meta-data for sparse representation. The compression scheme doesn't incur any accuracy loss.

features such as better privacy, less network bandwidth and real time processing, the large storage overhead prevents deep neural networks from being incorporated into mobile apps.

The second issue is energy consumption. Running large neural networks require a lot of memory bandwidth to fetch the weights and a lot of computation to do dot products— which in turn consumes considerable energy. Mobile devices are battery constrained, making power hungry applications such as deep neural networks hard to deploy.

Energy consumption is dominated by memory access. Under 45nm CMOS technology, a 32 bit floating point add consumes $\mathbf{0.9pJ}$, a 32bit SRAM cache access takes $\mathbf{5pJ}$, while a 32bit DRAM memory access takes $\mathbf{640pJ}$, which is 3 orders of magnitude of an add operation. Large networks do not fit in on-chip storage and hence require the more costly DRAM accesses. Running a 1 billion connection neural network, for example, at 20fps would require $(20Hz)(1G)(640pJ) = 12.8W$ just for DRAM access - well beyond the power envelope of a typical mobile device.

Our goal is to reduce the storage and energy required to run inference on such large networks so they can be deployed on mobile devices. To achieve this goal, we present "deep compression": a three-stage pipeline (Figure 1) to reduce the storage required by neural network in a manner that preserves the original accuracy. First, we prune the networking by removing the redundant connections, keeping only the most informative connections. Next, the weights are quantized so that multiple connections share the same weight, thus only the codebook (effective weights) and the indices need to be stored. Finally, we apply Huffman coding to take advantage of the biased distribution of effective weights.

Our main insight is that, pruning and trained quantization are able to compress the network without interfering each other, thus lead to surprisingly high compression rate. It makes the required storage so small (a few megabytes) that all weights can be cached on chip instead of going to off-chip DRAM which is energy consuming. Based on "deep compression", the EIE hardware accelerator Han et al. (2016) was later proposed that works on the compressed model, achieving significant speedup and energy efficiency improvement.

## 2 Network Pruning

Network pruning has been widely studied to compress CNN models. In early work, network pruning proved to be a valid way to reduce the network complexity and over-fitting (LeCun et al., 1989; Hanson & Pratt, 1989; Hassibi et al., 1993; Ström, 1997). Recently Han et al. (2015) pruned state-of-the-art CNN models with no loss of accuracy. We build on top of that approach. As shown on the left side of Figure 1, we start by learning the connectivity via normal network training. Next, we prune the small-weight connections: all connections with weights below a threshold are removed from the network. Finally, we retrain the network to learn the final weights for the remaining sparse connections. Pruning reduced the number of parameters by $9\times$ and $13\times$ for AlexNet and VGG-16 model.

To compress further, we store the index difference
difference in 8 bits for conv layer and 5 bits for f
than the bound, we the zero padding solution show
8, the largest 3-bit (as an example) unsigned numb

## 3 TRAINED QUANTIZATION AND WEI

Network quantization and weight sharing further
number of bits required to represent each weight. W
store by having multiple connections share the san

Weight sharing is illustrated in Figure 3. Suppos
output neurons, the weight is a $4 \times 4$ matrix. On
bottom left is the $4 \times 4$ gradient matrix. The weig
all the weights in the same bin share the same valu
a small index into a table of shared weights. Durin
and summed together, multiplied by the learning r
last iteration. For pruned AlexNet, we are able to
CONV layers, and 5-bits (32 shared weights) for e

To calculate the compression rate, given $k$ clusters,
general, for a network with $n$ connections and each
the connections to have only $k$ shared weights wil

$$r = \frac{}{nlog_2}$$

For example, Figure 3 shows the weights of a sing
four output units. There are $4 \times 4 = 16$ weights ori
weights are grouped together to share the same va