More General and Effective Model Compression via an Additive Combination of Compressions

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán

Dept. CSE, University of California, Merced
{yidelbayev,mcarreira-perpinan}@ucmerced.edu

Abstract. Model compression is generally performed by using quantization, low-rank approximation or pruning, for which various algorithms have been researched in recent years. One fundamental question is: what types of compression work better for a given model? Or even better: can we improve by combining compressions in a suitable way? We formulate this generally as a problem of optimizing the loss but where the weights are constrained to equal an additive combination of separately compressed parts; and we give an algorithm to learn the corresponding parts' parameters. Experimentally with deep neural nets, we observe that 1) we can find significantly better models in the error-compression space, indicating that different compression types have complementary benefits, and 2) the best type of combination depends exquisitely on the type of neural net. For example, we can compress ResNets and AlexNet using only 1 bit per weight without error degradation at the cost of adding a few floating point weights. However, VGG nets can be better compressed by combining low-rank with a few floating point weights.

Keywords: Additive combinations · Model compression

1 Introduction

In machine learning, model compression is the problem of taking a neural net or some other model, which has been trained to perform (near)-optimal prediction in a given task and dataset, and transforming it into a model that is smaller (in size, runtime, energy or other factors) while maintaining as good a prediction performance as possible. This problem has recently become important and actively researched because of the large size of state-of-the-art neural nets, trained on large-scale GPU clusters without constraints on computational resources, but which cannot be directly deployed in IoT devices with much more limited capabilities.

The last few years have seen much work on the topic, mostly focusing on specific forms of compression, such as quantization, low-rank matrix approximation and weight pruning, as well as variations of these. These papers typically propose a specific compression technique and a specific algorithm to compress a neural

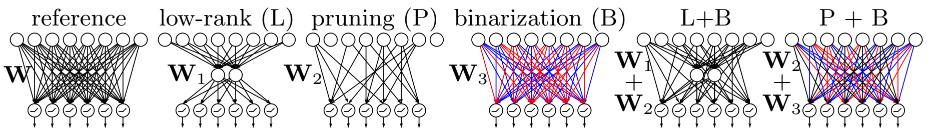


Fig. 1. Illustration of compression by additive combination $\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2 + \mathbf{W}_3$. Black weights are real, red weights are -1 and blue weights are $+1$.

net with it. The performance of these techniques individually varies considerably from case to case, depending on the algorithm (some are better than others) but more importantly on the compression technique. This is to be expected, because (just as happens with image or audio compression) some techniques achieve more compression for certain types of signals.

A basic issue is the representation ability of the compression: given an optimal point in model space (the weight parameters for a neural net), which manifold or subset of this space can be compressed exactly, and is that subset likely to be close to the optimal model for a given machine learning task? For example, for low-rank compression the subset contains all matrices of a given rank or less. Is that a good subset to model weight matrices arising from, say, deep neural nets for object classification?

One way to understand this question is to try many techniques in a given task and gain experience about what works in which case. This is made difficult by the multiplicity of existing algorithms, the heuristics often used to optimize the results experimentally (which are compounded by the engineering aspects involved in training deep nets, to start with), and the lack at present of an apples-to-apples evaluation in the field of model compression.

A second way to approach the question which partly sidesteps this problem is to use a common algorithm that can handle any compression technique. While compressing a deep net in a large dataset will still involve careful selection of optimization parameters (such as SGD learning rates), having a common algorithmic framework should put different compression techniques in the same footing. Yet a third approach, which we propose in this paper, is to *combine* several techniques (rather than try each in isolation) while jointly optimizing over the parameters of each (codebook and assignments for quantization, component matrices for low-rank, subset and value of nonzero weights for pruning, etc.).

There are multiple ways to define a combination of compression techniques. One that is simple to achieve is by applying compression techniques sequentially, such as first pruning the weights, then quantizing the remaining nonzero weights and finally encoding them with Huffman codes [13]. This is suboptimal in that the global problem is solved greedily, one compression at a time. The way we propose here is very different: an *additive combination* of compression techniques. For example, we may want to compress a given weight matrix $\mathbf{W}$ as the sum (or linear combination) $\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2 + \mathbf{W}_3$ of a low-rank matrix $\mathbf{W}_1$, a sparse matrix $\mathbf{W}_2$ and a quantized matrix $\mathbf{W}_3$. This introduces several important advantages. First, it contains as a particular case each technique in isolation (e.g., quantization by making $\mathbf{W}_1 = \mathbf{0}$ a zero-rank matrix and $\mathbf{W}_2 = \mathbf{0}$ a matrix with no nonzeros). Second, and critically, it allows techniques to help each other because of having complementary strengths. For example, pruning can be seen as adding a few elementwise real-valued corrections to a quantized or low-rank weight matrix. This could result (and does in some cases) in using fewer bits, lower rank and fewer nonzeros and a resulting higher compression ratio (in memory or runtime). Third, the additive combination vastly enlarges the subset of parameter space that can be compressed without loss compared to the individual compressions. This can be seen intuitively by noting that a fixed vector times a scalar generates a 1D space, but the additive combination of two such vectors generates a 2D space rather than two 1D spaces).

One more thing remains to make this possible: a formulation and corresponding algorithm of the compression problem that can handle such additive combinations of arbitrary compression techniques. We rely on the previously proposed "learning-compression (LC)" algorithm [8]. This explicitly defines the model weights as a function (called *decompression mapping*) of low-dimensional compression parameters; for example, the low-rank matrix above would be written as $\mathbf{W}_1 = \mathbf{U}\mathbf{V}^T$. It then iteratively optimizes the loss but constraining the weights to take the desired form (an additive combination in our case). This alternates *learning (L)* steps that train a regularized loss over the original model weights with *compression (C)* steps that compress the current weights, in our case according to the additive compression form.

Next, we review related work (section 2), describe our problem formulation (section 3) and corresponding LC algorithm (section 4), and demonstrate the claimed advantages with deep neural nets (sections 5, 6).

2 Related work

General approaches In the literature of model and particularly neural net compression, various approaches have been studied, including most prominently weight quantization, weight pruning and low-rank matrix or tensor decompositions. There are other approaches as well, which can be potentially used in combination with. We briefly discuss the individual techniques first. Quantization is a process of representing each weight with an item from a codebook. This can be achieved through fixed codebook schemes, i.e., with predetermined codebook values that are not learned (where only the assignments should be optimized). Examples of this compression are binarization, ternarization, low-precision, fixed-point or other number representations [22, 31]. Quantization can also be achieved through adaptive codebook schemes, where the codebook values are learned together with the assignment variables, with algorithms based on soft quantization [1, 29] or hard quantization [13]. Pruning is a process of removal of weights (unstructured) or filters and neurons (structured). It can be achieved by salience ranking [13, 21] in one go or over multiple refinements, or by using sparsifying norms [9, 41]. Low-rank approximation is a process of replacing weights with low-rank [18, 20, 35, 38] or tensors-decomposed versions [28].

Usage of combinations One of the most used combinations is to apply compressions sequentially, most notably first to prune weights and then to quantize the remaining ones [12, 13, 13, 34, 40], which may possibly be further compressed via lossless coding algorithms (e.g., Huffman coding). Additive combination of quantizations [3, 39, 44], where weights are the sum of quantized values, as well as low-rank + sparse combination [2, 43] has been used to compress neural networks. However, these methods rely on optimization algorithms highly specialized to a problem, limiting its application to new combinations (e.g., quantization + low-rank).

3 Compression via an additive combination as constrained optimization

Our basic formulation is that we define the weights as an additive combination of weights, where each term in the sum is individually compressed in some way. Consider for simplicity the case of adding just two compressions for a given matrix[1]. We then write a matrix of weights as $\mathbf{W} = \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) + \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)$, where $\boldsymbol{\theta}_i$ is the low-dimensional parameters of the ith compression and $\boldsymbol{\Delta}_i$ is the corresponding decompression mapping. Formally, the $\boldsymbol{\Delta}$ maps a compressed representation of the weight matrix $\boldsymbol{\theta}$ to the real-valued, uncompressed weight matrix $\mathbf{W}$. Its intent is to represent the space of matrices that can be compressed via a constraint subject to which we optimize the loss of the model in the desired task (e.g., classification). That is, a constraint $\mathbf{W} = \boldsymbol{\Delta}(\boldsymbol{\theta})$ defines a feasible set of compressed models. For example:

- Low-rank: $\mathbf{W} = \mathbf{U}\mathbf{V}^T$ with $\mathbf{U}$ and $\mathbf{V}$ of rank r, so $\boldsymbol{\theta} = (\mathbf{U}, \mathbf{V})$.
- Pruning: $\mathbf{w} = \boldsymbol{\theta}$ s.t. $\|\boldsymbol{\theta}\|_0 \leq \kappa$, so $\boldsymbol{\theta}$ is the indices of its nonzeros and their values.
- Scalar quantization: $w = \sum_{k=1}^K z_k c_k$ with assignment variables $\mathbf{z} \in \{0,1\}^K$, $\mathbf{1}^T\mathbf{z} = 1$ and codebook $\mathcal{C} = \{c_1, \dots, c_K\} \subset \mathbb{R}$, so $\boldsymbol{\theta} = (\mathbf{z}, \mathcal{C})$.
- Binarization: $w \in \{-1, +1\}$ or equivalently a scalar quantization with $\mathcal{C} = \{-1, +1\}$.

Note how the mapping $\boldsymbol{\Delta}(\boldsymbol{\theta})$ and the low-dimensional parameters $\boldsymbol{\theta}$ can take many forms (involving scalars, matrices or other objects of continuous or discrete type) and include constraints on $\boldsymbol{\theta}$. Then, our problem formulation takes the form of *model compression as constrained optimization* [8] and given as:

$$\min_{\mathbf{w}} L(\mathbf{w}) \quad \text{s.t.} \quad \mathbf{w} = \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) + \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2). \tag{1}$$

This expresses in a mathematical way our desire that 1) we want a model with minimum loss on the task at hand ($L(\mathbf{w})$ represents, say, the cross-entropy of the original deep net architecture on a training set); 2) the model parameters

[1] Throughout the paper we use $\mathbf{W}$ or $\mathbf{w}$ or w to notate matrix, vector or scalar weights as appropriate.

$\mathbf{w}$ must take a special form that allows them to be compactly represented in terms of low-dimensional parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$; and 3) the latter takes the form of an additive combination (over two compressions, in the example). Problem (1) has the advantage that it is amenable to modern techniques of numerical optimization, as we show in section 4.

Although the expression "$\mathbf{w} = \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) + \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)$" is an addition, it implicitly is a linear combination because the coefficients can typically be absorbed inside each $\boldsymbol{\Delta}_i$. For example, writing $\alpha\mathbf{U}\mathbf{V}^T$ (for low-rank compression) is the same as, say, $\mathbf{U}'\mathbf{V}^T$ with $\mathbf{U}' = \alpha\mathbf{U}$. In particular, any compression member may be implicitly removed by becoming zero. Some additive combinations are redundant, such as having both $\mathbf{W}_1$ and $\mathbf{W}_2$ be of rank at most r (since $\text{rank}\,(\mathbf{W}_1 + \mathbf{W}_2) \leq 2r$) or having each contain at most κ nonzeros (since $\left\|\mathbf{W}_1 + \mathbf{W}_2\right\|_0 \leq 2\kappa$).

The additive combination formulation has some interesting consequences. First, *an additive combination of compression forms can be equivalently seen as a new, learned deep net architecture.* For example (see fig. 1), low-rank plus pruning can be seen as a layer with a linear bottleneck and some skip connections which are learned (i.e., which connections to have and their weight value). It is possible that such architectures may be of independent interest in deep learning beyond compression. Second, while *pruning in isolation means (as is usually understood) the removal of weights from the model, pruning in an additive combination means the addition of a few elementwise real-valued corrections.* This can potentially bring large benefits. As an extreme case, consider binarizing both the multiplicative and additive (bias) weights in a deep net. It is known that the model's loss is far more sensitive to binarizing the biases, and indeed compression approaches generally do not compress the biases (which also account for a small proportion of weights in total). In binarization plus pruning, all weights are quantized but we learn which ones need a real-valued correction for an optimal loss. Indeed, our algorithm is able to learn that the biases need such corrections more than other weights (see corresponding experiment in suppl. mat. [10]).

Well known combinations Our motivation is to combine generically existing compressions in the context of model compression. However, some of the combinations are well known and extensively studied. Particularly, low-rank + sparse combination has been used in its own right in the fields of compressed sensing [7], matrix decomposition [45], and image processing [6]. This combination enjoys certain theoretical guarantees [7, 11], yet it is unclear whether similar results can be stated over more general additive combinations (e.g., with non-differentiable scheme like quantization) or when applied to non-convex models as deep nets.

Hardware implementation The goal of model compression is to implement in practice the compressed model based on the $\boldsymbol{\theta}$ parameters, not the original weights $\mathbf{W}$. With an additive combination, the implementation is straightforward and efficient by applying the individual compressions sequentially and cumulatively. For example, say $\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2$ is a weight matrix in a layer of a deep net and we want to compute the layer's output activations $\sigma(\mathbf{W}\mathbf{x})$ for a given input vector of activations $\mathbf{x}$ (where $\sigma(\cdot)$ is a nonlinearity, such as a ReLU).

By the properties of linearity, $\mathbf{W}\mathbf{x} = \mathbf{W}_1\mathbf{x} + \mathbf{W}_2\mathbf{x}$, so we first compute $\mathbf{y} = \mathbf{W}_1\mathbf{x}$ according to an efficient implementation of the first compression, and then we accumulate $\mathbf{y} = \mathbf{y} + \mathbf{W}_2\mathbf{x}$ computed according to an efficient implementation of the second compression. This is particularly beneficial because some compression techniques are less hardware-friendly than others. For example, quantization is very efficient and cache-friendly, since it can store the codebook in registers, access the individual weights with high memory locality, use mostly floating-point additions (and nearly no multiplications), and process rows of $\mathbf{W}_1$ in parallel. However, pruning has a complex, nonlocal pattern of nonzeros whose locations must be stored. Combining quantization plus pruning not only can achieve higher compression ratios than either just quantization or just pruning, as seen in our experiments; it can also reduce the number of bits per weight and (drastically) the number of nonzeros, thus resulting in fewer memory accesses and hence lower runtime and energy consumption.

4 Optimization via a learning-compression algorithm

Although optimizing (1) may be done in different ways for specific forms of the loss L or the decompression mapping constraint $\boldsymbol{\Delta}$, it is critical to be able to do this in as generic way as possible, so it applies to any combination of forms of the compressions, loss and model. Following Carreira-Perpinan [8], we apply a penalty method and then alternating optimization. We give the algorithm for the quadratic-penalty method [27], but we implement the augmented Lagrangian one (which works in a similar way but with the introduction of a Lagrange multiplier vector $\boldsymbol{\lambda}$ of the same dimension as $\mathbf{w}$). We then optimize the following while driving a penalty parameter $\mu \to \infty$:

$$Q(\mathbf{w}, \boldsymbol{\theta}; \mu) = L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)\|^2 \tag{2}$$

by using alternating optimization over $\mathbf{w}$ and $\boldsymbol{\theta}$. The step over $\mathbf{w}$ ("learning (L)" step) has the form of a standard loss minimization but with a quadratic regularizer on $\mathbf{w}$ (since $\boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) + \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)$ is fixed), and can be done using a standard algorithm to optimize the loss, e.g., SGD with deep nets. The step over $\boldsymbol{\theta}$ ("compression (C)" step) has the following form:

$$\min_{\boldsymbol{\theta}} \|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\theta})\|^2 \Leftrightarrow \min_{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2} \|\mathbf{w} - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)\|^2. \tag{3}$$

In the original LC algorithm [8], this step (over just a single compression $\boldsymbol{\Delta}(\boldsymbol{\theta})$) typically corresponds to a well-known compression problem in signal processing and can be solved with existing algorithms. This gives the LC algorithm a major advantage: in order to change the compression form, we simply call the corresponding subroutine in this step (regardless of the form of the loss and model). For example, for low-rank compression the solution is given by a truncated SVD, for pruning by thresholding the largest weights, and for quantization by k-means. It is critical to preserve that advantage here so that we can handle

Algorithm 1 Pseudocode (quadratic-penalty version)

```
input training data, neural net architecture with weights $\mathbf{w}$
$\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} L(\mathbf{w})$                                      reference net
$\boldsymbol{\theta}_1, \boldsymbol{\theta}_2 \leftarrow \arg\min_{\boldsymbol{\theta}_1,\boldsymbol{\theta}_2} \|\mathbf{w} - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)\|^2$    init
for $\mu = \mu_0 < \mu_1 < \dots < \infty$
  $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)\|^2$    L step
  while alternation does not converge                                     }
    $\boldsymbol{\theta}_1 \leftarrow \arg\min_{\boldsymbol{\theta}_1} \|(\mathbf{w} - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)) - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1)\|^2$    } C step
    $\boldsymbol{\theta}_2 \leftarrow \arg\min_{\boldsymbol{\theta}_2} \|(\mathbf{w} - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1)) - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)\|^2$    }
  if $\|\mathbf{w} - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1) - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)\|$ is small enough then exit the loop
return $\mathbf{w}, \boldsymbol{\theta}_1, \boldsymbol{\theta}_2$
```

in a generic way an arbitrary additive combination of compressions. Fortunately, we can achieve this by applying alternating optimization again but now to (3) over $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, as follows[2]:

$$\begin{aligned} \boldsymbol{\theta}_1 &= \arg\min_{\boldsymbol{\theta}} \|(\mathbf{w} - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)) - \boldsymbol{\Delta}_1(\boldsymbol{\theta})\|^2 \\ \boldsymbol{\theta}_2 &= \arg\min_{\boldsymbol{\theta}} \|(\mathbf{w} - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1)) - \boldsymbol{\Delta}_2(\boldsymbol{\theta})\|^2 \end{aligned} \tag{4}$$

Each problem in (4) now does have the standard compression form of the original LC algorithm and can again be solved by an existing algorithm to compress optimally according to $\boldsymbol{\Delta}_1$ or $\boldsymbol{\Delta}_2$. At the beginning of each C step, we initialize $\boldsymbol{\theta}$ from the previous C step's result (see Alg. 1).

It is possible that a better algorithm exists for a specific form of additive combination compression (3). In such case we can employ specialized version during the C step. But our proposed alternating optimization (4) provides a generic, efficient solution as long as we have a good algorithm for each individual compression.

Convergence of the alternating steps (4) to a global optimum of (3) over $(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2)$ can be proven in some cases, e.g., low-rank + sparse [45], but not in general, as one would expect since some of the compression problems involve discrete and continuous variables and can be NP-hard (such as quantization with an adaptive codebook). Convergence can be established quite generally for convex functions [4, 36]. For nonconvex functions, convergence results are complex and more restrictive [33]. One simple case where convergence occurs is if the objective in (3) (i.e., each $\boldsymbol{\Delta}_i$) is continuously differentiable and it has a unique minimizer over each $\boldsymbol{\theta}_i$ [5, Proposition 2.7.1]. However, in certain cases the optimization can be solved exactly without any alternation. We give a specific result next.

[2] This form of iterated "fitting" (here, compression) by a "model" (here, $\boldsymbol{\Delta}_1$ or $\boldsymbol{\Delta}_2$) of a "residual" (here, $\mathbf{w} - \boldsymbol{\Delta}_2(\boldsymbol{\theta}_2)$ or $\mathbf{w} - \boldsymbol{\Delta}_1(\boldsymbol{\theta}_1)$) is called backfitting in statistics, and is widely used with additive models [14].

4.1 Exactly solvable C step

Solution of the C step (eq. 3) does not need to be an alternating optimization. Below we give an exact algorithm for the additive combination of fixed codebook quantization (e.g., $\{-1,+1\}$, $\{-1,0,+1\}$, etc.) and sparse corrections.

Theorem 1 (Exactly solvable C step for combination of fixed codebook quantization + sparse corrections). *Given a fixed codebook $\mathcal{C}$ consider compression of the weights w_i with an additive combinations of quantized values $q_i \in \mathcal{C}$ and sparse corrections s_i:*

$$\min_{q,s} \sum_i (w_i - (q_i + s_i))^2 \quad s.t. \quad \|s\|_0 \leq \kappa, \tag{5}$$

Then the following provides one optimal solution (q^*, s^*): first set $q_i^* = closest(w_i)$ in codebook for each i, then solve for s: $\min_s \sum_i (w_i - q_i^* - s_i))^2$ s.t. $\|s\|_0 \leq \kappa$.

Proof. Imagine we know the optimal set of nonzeros of the vector s, which we denote as $\mathcal{N}$. Then, for the elements not in $\mathcal{N}$, the optimal solution is $s_i^* = 0$ and $q_i^* = closest(w_i)$. For the elements in $\mathcal{N}$, we can find their optimal solution by solving independently for each i:

$$\min_{q_i,s_i} (w_i - (q_i + s_i))^2 \quad \text{s.t.} \quad q_i \in \mathcal{C}.$$

The solution is $s_i^* = w_i - q_i$ for arbitrary chosen $q_i \in \mathcal{C}$. Using this, we can rewrite the eq. 5 as $\sum_{i \notin \mathcal{N}} (w_i - q_i^*)^2$.

This is minimized by taking as set $\mathcal{N}$ the κ largest in magnitude elements of $w_i - q_i^*$ (indexed over i). Hence, the final solution is: 1) Set the elements of $\mathcal{N}$ to be the κ largest in magnitude elements of $w_i - q_i^*$ (there may be multiple such sets, any one is valid). 2) For each i in $\mathcal{N}$: set $s_i^* = w_i - q_i^*$, and $q_i^* =$ any element in $\mathcal{C}$. For each i not in $\mathcal{N}$: set $s_i^* = 0$, $q_i^* = closest(w_i)$ (there may be 2 closest values, any one is valid). This contains multiple solutions. One particular one is as given in the theorem statement, where we set $q_i^* = closest(w_i)$ for every i, which is practically more desirable because it leads to a smaller ℓ_1-norm of s.

5 Experiments on CIFAR10

We evaluate the effectiveness of additively combining compressions on deep nets of different sizes on the CIFAR10 (VGG16 and ResNets). We systematically study each combination of two or three compressions out of quantization, low-rank and pruning. We demonstrate that the additive combination improves over any single compression contained in the combination (as expected), and is comparable or better than sequentially engineered combinations such as first pruning some weights and then quantizing the rest. We sometimes achieve models that not only compress the reference significantly but also reduce its error. Notably, this happens with ResNet110 (using quantization plus either pruning or low-rank), even though our reference ResNets were already well trained and achieved a lower test error than in the original paper [15].

	1bit Q + % P	log L	$E_{\text{test}}(\%)$	ρ_{s}	ρ_{+}	$\rho_{\times}$
ResNet20	**R**	-0.80	8.35	1.00	**1.00**	1.00
	Q + 1.0% P	-0.84	9.16	**22.67**	0.97	**30.74**
	Q + 2.0% P	-0.92	8.92	19.44	0.96	19.74
	Q + 3.0% P	-0.93	8.31	17.08	0.94	15.80
	Q + 5.0% P	-0.99	**8.26**	13.84	0.92	11.54
ResNet32	**R**	-0.82	7.14	1.00	**1.00**	1.00
	Q + 1.0% P	-1.03	7.57	**22.81**	0.97	**30.52**
	Q + 2.0% P	-1.07	7.61	19.54	0.96	19.85
	Q + 3.0% P	-1.10	7.29	17.14	0.94	15.80
	Q + 5.0% P	-1.14	**7.09**	13.84	0.92	11.56
ResNet56	**R**	-0.81	6.58	1.00	**1.00**	1.00
	Q + 0.5% P	-1.08	6.77	**25.04**	0.98	**49.79**
	Q + 1.0% P	-1.13	6.73	22.87	0.97	32.04
	Q + 2.0% P	-1.17	6.70	19.55	0.96	20.46
	Q + 3.0% P	-1.18	**6.23**	17.11	0.94	15.98
ResNet110	**R**	-0.77	6.02	1.00	**1.00**	1.00
	Q + 0.5% P	-1.16	6.20	**25.03**	0.99	**55.63**
	Q + 1.0% P	-1.20	5.80	22.80	0.98	35.94
	Q + 2.0% P	-1.23	5.66	19.47	0.96	27.27
	Q + 3.0% P	-1.25	**5.58**	17.04	0.95	17.84

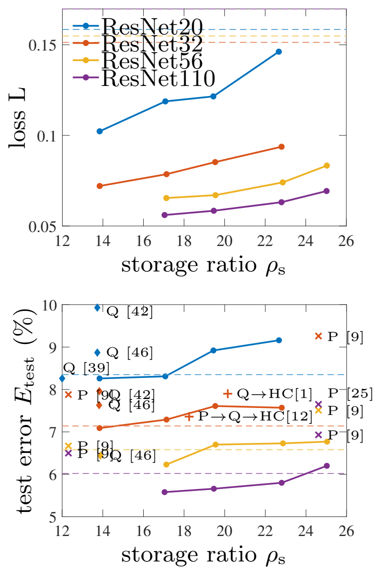


Fig. 2. Q+P. *Left*: results of running 1-bit quantization with varying amounts of additive pruning (corrections) on ResNets of different depth on CIFAR-10 (with reference nets denoted **R**). We report training loss (logarithms are base 10), test error E_{test} (%), and ratios of storage ρ_{s} and floating point additions (ρ_{+}) and multiplications ($\rho_{\times}$). Boldfaced results are the best for each ResNet depth. *Right*: training loss (*top*) and test error (*bottom*) as a function of the storage ratio. For each net, we give our algorithm's compression over several values of P, thus tracing a line in the error-compression space (reference nets: horizontal dashed lines). We also report results from the literature as isolated markers with a citation: quantization Q, pruning P, Huffman coding HC, and their sequential combination using arrows (e.g., Q→HC). Point "Q [39]" on the left border is outside the plot ($\rho_{\text{s}} < 12$).

We initialize our experiments from reasonably well-trained reference models. We train reference ResNets of depth 20, 32, 56, and 110 following the procedure of the original paper [15] (although we achieve lower errors). The models have 0.26M, 0.46M, 0.85M, and 1.7M parameters and test errors of 8.35%, 7.14%, 6.58% and 6.02%, respectively. We adapt VGG16 [32] to the CIFAR10 dataset (see details in suppl. mat. [10]) and train it using the same data augmentation as for ResNets. The reference VGG16 has 15.2M parameters and achieves a test error of 6.45%.

The optimization protocol of our algorithm is as follows throughout all experiments with minor changes (see suppl. mat. [10]). To optimize the L step we use Nesterov's accelerated gradient method [26] with momentum of 0.9 on minibatches of size 128, with a decayed learning rate schedule of $\eta_0 \cdot a^m$ at the mth epoch. The initial learning rate η_0 is one of {0.0007,0.007,0.01}, and the learning rate decay one of {0.94,0.98}. Each L step is run for 20 epochs. Our LC

	1bit Q + rank r	$\log L$	E_{test} (%)	ρ_s	ρ_+	$\rho_\times$
ResNet20	**R**	-0.80	**8.35**	1.00	**1.00**	1.00
	Q + rank 1	-0.77	9.71	**20.71**	0.96	**21.45**
	Q + rank 2	-0.84	9.30	16.62	0.92	11.26
	Q + rank 3	-0.89	8.64	13.88	0.89	7.64
ResNet32	**R**	-0.82	**7.14**	1.00	**1.00**	1.00
	Q + rank 1	-0.99	7.90	**20.94**	0.97	**21.89**
	Q + rank 2	-1.04	8.06	16.81	0.92	11.47
	Q + rank 3	-1.10	7.52	14.04	0.89	7.77
ResNet56	**R**	-0.81	6.58	1.00	**1.00**	1.00
	Q + rank 1	-1.13	7.19	**21.04**	0.96	**22.19**
	Q + rank 2	-1.19	6.51	16.91	0.92	11.61
	Q + rank 3	-1.22	**6.29**	14.10	0.89	7.87
ResNet110	**R**	-0.77	6.02	1.00	**1.00**	1.00
	Q + rank 1	-1.19	5.98	**21.11**	0.96	**22.38**
	Q + rank 2	-1.24	5.93	16.96	0.92	11.70
	Q + rank 3	-1.27	**5.50**	14.18	0.89	7.92

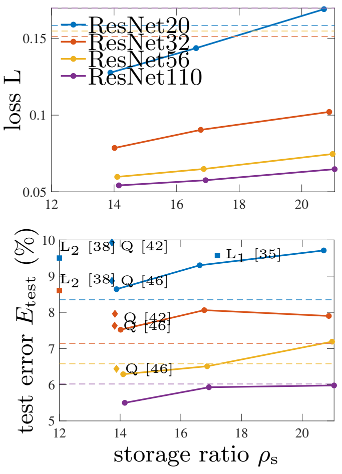


Fig. 3. Q+L. *Left*: results of running 1-bit quantization with addition of a low-rank matrix of different rank on ResNets on CIFAR10. The organization is as for fig. 2. In the right-bottom plot we also show results from the literature for single compressions (Q: quantization, L: low-rank). Points "L [38]" on the left border are outside the plot ($\rho_s < 12$).

algorithm (we use augmented Lagrangian version) runs for j steps where $j \leq 50$, and has a penalty parameter schedule $\mu_j = \mu_0 \cdot 1.1^j$; we choose μ_0 to be one of $\{5 \cdot 10^{-4}, 10^{-3}\}$. The solution of the C step requires alternating optimization over individual compressions, which we perform 30 times per each step.

We report the training loss and test error as measures of the model classification performance; and the ratios of storage (memory occupied by the parameters) ρ_s, number of multiplications $\rho_\times$ and number of additions ρ_+ as measures of model compression. Although the number of multiplications and additions is about the same in a deep net's inference pass, we report them separately because different compression techniques (if efficiently implemented) can affect quite differently their costs. We store low-rank matrices and sparse correction values using 16-bit precision floating point values. See our suppl. mat. [10] for precise definitions and details of these metrics.

5.1 Q+P: quantization plus pruning

We compress ResNets with a combination of quantization plus pruning. Every layer is quantized separately with a codebook of size 2 (1 bit). For pruning we employ the constrained ℓ_0 formulation [9], which allows us to specify a single number of nonzero weights κ for the entire net (κ is the "% P" value in fig. 2). The C step of eq. (3) for this combination alternates between running k-means (for quantization) and a closed-form solution based on thresholding (for pruning).

Fig. 2 shows our results; published results from the literature are at the bottom-right part, which shows the error-compression space. We are able to

Table 1. L+P. Compressing VGG16 with low-rank and pruning using our algorithm (top) and by recent works on structured and unstructured pruning. Metrics as in Fig. 2.

	Model	E_{test} (%)	ρ_{s}
	R VGG16	**6.45**	1.00
ours	rank 2 + 2% P	6.66	**60.99**
ours	rank 3 + 2% P	6.65	56.58
	pruning [25]	6.66	$\approx$ 24.53
	filter pruning [23]	6.60	5.55
	quantization [30]	8.00	43.48

achieve considerable compression ratios ρ_{s} of up to 20× without any degradation in accuracy, and even higher ratios with minor degradation. These results beat single quantization or pruning schemes reported in the literature for these models. The best 2-bit quantization approaches for ResNets we know about [42, 46] have $\rho_{\text{s}} \approx 14\times$ and lose up to 1% in test error comparing to the reference; the best unstructured pruning [9, 25] achieves $\rho_{\text{s}} \approx 12\times$ and loses 0.8%.

ResNet20 is the smallest and hardest to compress out of all ResNets. With 1-bit quantization plus 3% pruning corrections we achieve an error of 8.26% with $\rho_{\text{s}} = 13.84\times$. To the best of our knowledge, the highest compression ratio of comparable accuracy using only quantization is 6.22× and has an error of 8.25% [39]. On ResNet110 with 1-bit quantization plus 3% corrections, we achieve 5.58% error while still compressing 17×.

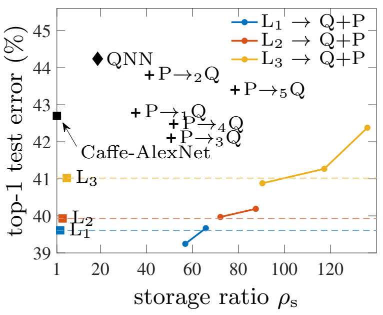
Our results are comparable or better than published results where multiple compressions are applied sequentially (Q→HC and P→Q→HC in fig. 2). For example, quantizing and then applying Huffman coding to ResNet32 [1] achieves $\rho_{\text{s}} = 20.15\times$ with 7.9% error, while we achieve $\rho_{\text{s}} = 22.81\times$ with 7.57% error. We re-emphasize that unlike the "prune then quantize" schemes, our additive combination is different: we quantize all weights and apply a pointwise correction.

5.2 Q+L: quantization plus low-rank

We compress the ResNets with the additive combination of 1-bit quantized weights (as in section 5.1) and rank-r matrices, where the rank is fixed and has the same value for all layers. The convoloutional layers are parameterized by low-rank as in Wen et al. [35]. The solution of the C step (4) for this combination is an alternation between k-means and truncated SVD.

Fig. 3 shows our results and at bottom-right of Fig. 3 we see that our additive combination (lines traced by different values of the rank r in the error-compression space) consistently improve over individual compression techniques reported in the literature (quantization or low-rank, shown by markers Q or L). Notably, the low-rank approximation is not a popular choice for compression of ResNets: fig. 3 shows only two markers, for the only two papers we know [35, 38]. Assuming storage with 16-bit precision on ResNet20, Wen et al. [35] achieve 17.20× storage compression (with 9.57% error) and Xu et al. [38] respec-

	Model	top-1	MBs	MFLOPs
	Caffe-AlexNet [19]	42.70	243.5	724
	AlexNet-QNN [37]	44.24	13.0	175
	P→$_1$Q [13]	42.78	6.9	724
	P→$_2$Q [12]	43.80	5.9	724
	P→$_3$Q [34]	42.10	4.8	724
	P→$_4$Q [40]	42.48	4.7	724
	P→$_5$Q [40]	43.40	3.1	724
	filter pruning [24]	43.17	232.0	334
	R Low-rank AlexNet (L$_1$)	39.61	100.5	227
ours	L$_1$ → Q + P (0.25M)	39.67	3.7	227
ours	L$_1$ → Q + P (0.50M)	**39.25**	4.3	227
	R Low-rank AlexNet (L$_2$)	39.93	69.8	185
ours	L$_2$ → Q + P (0.25M)	40.19	2.8	185
ours	L$_2$ → Q + P (0.50M)	39.97	3.4	185
	R Low-rank AlexNet (L$_3$)	41.02	45.9	**152**
ours	L$_3$ → Q + P (0.125M)	42.38	**1.8**	**152**
ours	L$_3$ → Q + P (0.25M)	41.27	2.1	**152**
ours	L$_3$ → Q + P (0.50M)	40.88	2.7	**152**



Inference time and speed-up using 1-bit Q + 0.25M P

Model	time, ms	speed-up
Caffe-AlexNet	23.27	1.00
L$_1$ → Q + P	11.32	2.06
L$_2$ → Q + P	8.75	2.66
L$_3$ → Q + P	6.72	3.46

Fig. 4. Q+P scheme is powerful enough to further compress already downsized models, here, it is used to further compress the low-rank AlexNets [17]. In all our experiments reported here, we use 1-bit quantization with varying amount of pruning. *Left:* We report top-1 validation error, size of the final model in MB when saved to disk, and resulting FLOPs. P—pruning, Q— quantization, L—low-rank. *Top right:* same as the table on the left, but in graphical form. Our compressed models are given as solid connected lines. *Bottom right:* The delay (in milliseconds) and corresponding speed-ups of our compressed models on Jetson Nano Edge GPU.

tively 5.39× (with 9.5% error), while our combination of 1-bit quantization plus rank-2 achieves 16.62× (9.3% error).

5.3 L+P: low-rank plus pruning

We compress VGG16 trained on CIFAR10 using the additive combination of low-rank matrices and pruned weights. The reference model has 15.2M parameters, uses 58.17 MB of storage and achieves 6.45% test error. When compressing with L+P scheme of rank 2 and 3% point-wise corrections (Table 1), we achieve a compression ratio of to 60.99× (0.95 MB storage), and the test error of 6.66%.

6 Experiments on ImageNet

To demonstrate the power and complementary benefits of additive combinations, we proceed by applying the Q+P combination to *already downsized* models trained on the ILSVRC2012 dataset. We obtain low-rank AlexNets following the work of [17], and compress them further with Q+P scheme. The hyperparameters of the experiments are almost identical to CIFAR10 experiments (sec. 5) with minor changes, see suppl. mat. [10].

In Fig. 4 (left) we report our results: the achieved top-1 error, the size in megabytes when a compressed model is saved to disk (we use the sparse index compression procedure of Han et al. [13]), and floating point operations required to perform the forward pass through a network. Additionally, we include prominent results from the literature to put our models in perspective. Our Q+P models achieve *significant compression* of AlexNet: we get 136× compression (1.789 MB) without degradation in accuracy and 87× compression with more than 2% improvement in the top-1 accuracy when compared to the Caffe-AlexNet. Recently, Yang et al. [40] (essentially using our Learning-Compression framework) reported 118× and 205× compression on AlexNet with none to small reduction of accuracy. However, as can be found by inspecting the code of Yang et al. [40], these numbers are artificially inflated in that they do not account for the storage of the element indices (for a sparse matrix), for the storage of the codebook, and use a fractional number of bits per element instead of rounding it up to an integer. If these are taken into account, the actual compression ratios become much smaller (52× and 79×), with models of sizes 4.7MB and 3.1MB respectively (see left of Fig. 4). Our models outperform those and other results not only in terms of size, but also in terms of inference speed. We provide the runtime evaluation (when processing a single image) of our compressed models on a small edge device (NVIDIA's Jetson Nano) on the right bottom of Fig. 4.

7 Conclusion

We have argued for and experimentally demonstrated the benefits of applying multiple compressions as an additive combination. We achieve this via a general, intuitive formulation of the optimization problem via constraints characterizing the additive combination, and an algorithm that can handle *any choice of compression combination* as long as each individual compression can be solved on its own. In this context, pruning takes the meaning of adding a few elementwise corrections where they are needed most. This can not only complement existing compressions such as quantization or low-rank, but also be an interesting way to learn skip connections in deep net architectures. With deep neural nets, we observe that we can find significantly better models in the error-compression space, indicating that *different compression types have complementary benefits*, and that the best type of combination depends exquisitely on the type of neural net. The resulting compressed nets may also make better use of the available hardware. Our codes and models are available at https://github.com/UCMerced-ML/LC-model-compression as part of LC Toolkit [16].

Our work opens up possibly new and interesting mathematical problems regarding the best approximation of a matrix by X, such as when X is the sum of a quantized matrix and a sparse matrix. Also, we do not claim that additive combination is the only or the best way to combine compressions, and future work may explore other ways.

Acknowledgments We thank NVIDIA Corporation for multiple GPU donations.

Bibliography

[1] Agustsson, E., Mentzer, F., Tschannen, M., Cavigelli, L., Timofte, R., Benini, L., Van Gool, L.: Soft-to-hard vector quantization for end-to-end learning compressible representations. In: Advances in Neural Information Processing Systems (NIPS). vol. 30, pp. 1141–1151.

[2] Alvarez, J.M., Salzmann, M.: Compression-aware training of deep networks. In: Advances in Neural Information Processing Systems (NIPS). vol. 30.

[3] Babenko, A., Lempitsky, V.: Additive quantization for extreme vector compression. In: Proc. of the 2014 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'14). pp. 931–938.

[4] Beck, A., Tetruashvili, L.: On the convergence of block coordinate descent type methods. SIAM J. Optimization **23**(4), 2037–2060 (2013)

[5] Bertsekas, D.P.: Nonlinear Programming. Athena Scientific, Nashua, NH, second edn. (1999)

[6] Bouwmans, T., hadi Zahzah, E.: Robust principal component analysis via decomposition into low-rank and sparse matrices: An overview. In: Handbook of Robust Low-Rank and Sparse Matrix Decomposition. Applications in Image and Video Processing, chap. 1, pp. 1.1–1.61. CRC Publishers (2016)

[7] Candès, E.J., Li, X., Ma, Y., Wright, J.: Robust principal component analysis? Journal of the ACM **58**(3), 11 (May 2011)

[8] Carreira-Perpiñán, M.Á.: Model compression as constrained optimization, with application to neural nets. Part I: General framework (Jul 5 2017), arXiv:1707.01209

[9] Carreira-Perpiñán, M.Á., Idelbayev, Y.: "Learning-compression" algorithms for neural net pruning. In: Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18). pp. 8532–8541.

[10] Carreira-Perpiñán, M.Á., Idelbayev, Y.: Model compression as constrained optimization, with application to neural nets. Part V: Combining compressions (2021), arXiv:2107.04380

[11] Chandrasekaran, V., Sanghavi, S., Parrilo, P.A., Willsky, A.S.: Rank-sparsity incoherence for matrix decomposition. SIAM J. Optimization **21**(2), 572–596 (2010)

[12] Choi, Y., El-Khamy, M., Lee, J.: Towards the limit of network quantization. In: Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017).

[13] Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In: Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016).

[14] Hastie, T.J., Tibshirani, R.J.: Generalized Additive Models. No. 43 in Monographs on Statistics and Applied Probability, Chapman & Hall, London, New York (1990)

[15] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16). pp. 770–778.

[16] Idelbayev, Y., Carreira-Perpiñán, M.Á.: A flexible, extensible software framework for model compression based on the LC algorithm (May 15 2020), arXiv:2005.07786

[17] Idelbayev, Y., Carreira-Perpiñán, M.Á.: Low-rank compression of neural nets: Learning the rank of each layer. In: Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20).

[18] Idelbayev, Y., Carreira-Perpiñán, M.Á.: Optimal selection of matrix shape and decomposition scheme for neural network compression. In: Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'21).

[19] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., Darrell, T.: Caffe: Convolutional architecture for fast feature embedding (Jun 20 2014), arXiv:1408.5093

[20] Kim, H., Khan, M.U.K., Kyung, C.M.: Efficient neural network compression. In: Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19). pp. 12569–12577.

[21] LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Touretzky, D.S. (ed.) Advances in Neural Information Processing Systems (NIPS). vol. 2, pp. 598–605. Morgan Kaufmann, San Mateo, CA (1990)

[22] Li, F., Zhang, B., Liu, B.: Ternary weight networks (Nov 19 2016), arXiv:1605.04711

[23] Li, H., Kadav, A., Durdanovic, I., Graf, H.P.: Pruning filters for efficient ConvNets. In: Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017).

[24] Li, J., Qi, Q., Wang, J., Ge, C., Li, Y., Yue, Z., Sun, H.: OICSR: Out-in-channel sparsity regularization for compact deep neural networks. In: Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19). pp. 7046–7055.

[25] Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T.: Rethinking the value of network pruning. In: Proc. of the 7th Int. Conf. Learning Representations (ICLR 2019).

[26] Nesterov, Y.: A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. Soviet Math. Dokl. **27**(2), 372–376 (1983)

[27] Nocedal, J., Wright, S.J.: Numerical Optimization. Springer Series in Operations Research and Financial Engineering, Springer-Verlag, New York, second edn. (2006)

[28] Novikov, A., Podoprikhin, D., Osokin, A., Vetrov, D.P.: Tensorizing neural networks. In: Advances in Neural Information Processing Systems (NIPS). vol. 28, pp. 442–450. MIT Press, Cambridge, MA (2015)

[29] Nowlan, S.J., Hinton, G.E.: Simplifying neural networks by soft weight-sharing. Neural Computation **4**(4), 473–493 (Jul 1992)

[30] Qu, Z., Zhou, Z., Cheng, Y., Thiele, L.: Adaptive loss-aware quantization for multi-bit networks. In: Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20).

[31] Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-net: ImageNet classification using binary convolutional neural networks. In: Proc. 14th European Conf. Computer Vision (ECCV'16). pp. 525–542.

[32] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015).

[33] Tseng, P.: Convergence of a block coordinate descent method for nondifferentiable minimization. J. Optimization Theory and Applications **109**(3), 475–494 (Jun 2001)

[34] Tung, F., Mori, G.: CLIP-Q: Deep network compression learning by in-parallel pruning-quantization. In: Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18).

[35] Wen, W., Xu, C., Wu, C., Wang, Y., Chen, Y., Li, H.: Coordinating filters for faster deep neural networks. In: Proc. 17th Int. Conf. Computer Vision (ICCV'17).

[36] Wright, S.J.: Coordinate descent algorithms. Math. Prog. **151**(1), 3–34 (Jun 2016)

[37] Wu, J., Leng, C., Wang, Y., Hu, Q., Cheng, J.: Quantized convolutional neural networks for mobile devices. In: Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16).

[38] Xu, Y., Li, Y., Zhang, S., Wen, W., Wang, B., Qi, Y., Chen, Y., Lin, W., Xiong, H.: TRP: Trained rank pruning for efficient deep neural networks. In: Proc. of the 29th Int. Joint Conf. Artificial Intelligence (IJCAI'20).

[39] Xu, Y., Wang, Y., Zhou, A., Lin, W., Xiong, H.: Deep neural network compression with single and multiple level quantization. In: Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018).

[40] Yang, H., Gui, S., Zhu, Y., Liu, J.: Automatic neural network compression by sparsity-quantization joint learning: A constrained optimization-based approach. In: Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20). pp. 2175–2185.

[41] Ye, J., Lu, X., Lin, Z., Wang, J.: Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In: Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018).

[42] Yin, P., Zhang, S., Lyu, J., Osher, S., Qi, Y., Xin, J.: BinaryRelax: A relaxation approach for training deep neural networks with quantized weights. SIAM J. Imaging Sciences **11**(4), 2205–2223 (2018), arXiv:1801.06313

[43] Yu, X., Liu, T., Wang, X., Tao, D.: On compressing deep models by low rank and sparse decomposition. In: Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17). pp. 67–76.

[44] Zhou, A., Yao, A., Guo, Y., Xu, L., Chen, Y.: Incremental network quantization: Towards lossless CNNs with low-precision weights. In: Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017).

[45] Zhou, T., Tao, D.: GoDec: Randomized low-rank & sparse matrix decomposition in noisy case. In: Proc. of the 28th Int. Conf. Machine Learning (ICML 2011). pp. 33–40.

[46] Zhu, C., Han, S., Mao, H., Dally, W.J.: Trained ternary quantization. In: Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017).