

Model compression as constrained optimization, with application to neural nets. Part V: combining compressions

Miguel Á. Carreira-Perpiñán Yerlan Idelbayev
Dept. of Computer Science & Engineering, University of California, Merced
<http://eecs.ucmerced.edu>
<https://github.com/UCMerced-ML/LC-model-compression>

July 9, 2021

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.

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 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?

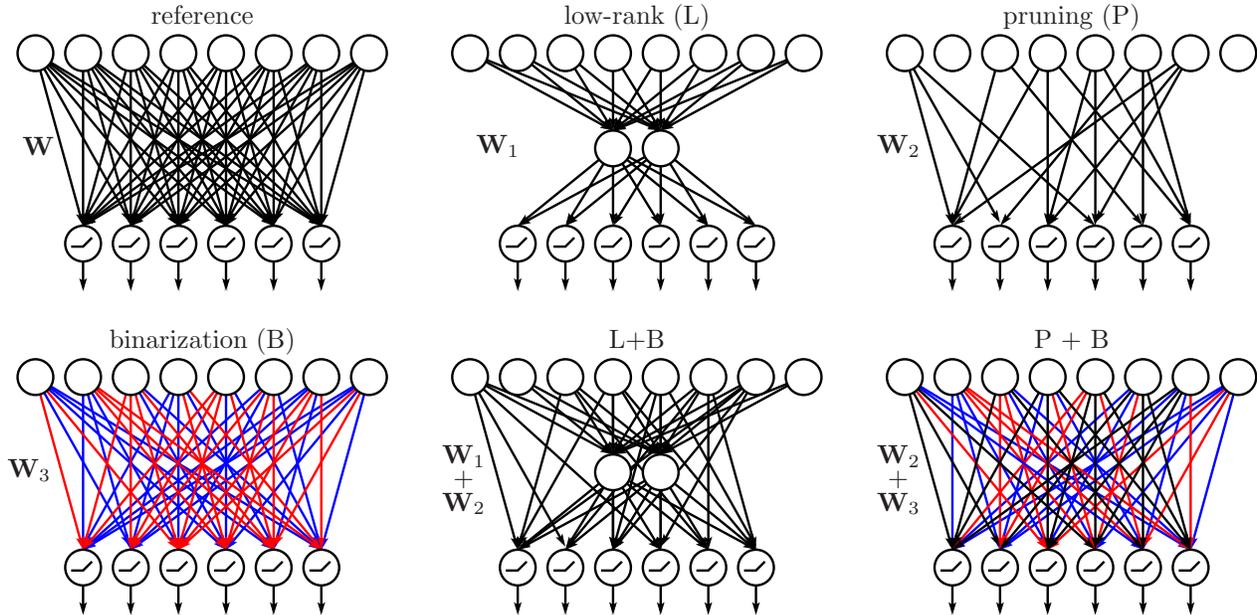


Figure 1: Illustration of compression by additive combination $\mathbf{W} = \mathbf{W}_1 + \mathbf{W}_2 + \mathbf{W}_3$. Black weights are real, red weights are -1 and blue weights are $+1$.

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). A shorter version of this paper appears as [20].

2 Related work

2.1 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 [26, 35]. 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, 33] 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, 24] in one go or over multiple refinements, or by using sparsifying norms [9, 49]. Low-rank approximation is a process of replacing weights with low-rank [19, 22, 41, 46] or tensors-decomposed versions [32].

2.2 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 [11, 13, 13, 39, 47], which may possibly be further compressed via lossless coding algorithms (e.g., Huffman coding). Additive combination of quantizations [3, 45, 53], where weights are the sum of quantized values, as well as low-rank + sparse combination [2, 52] 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 matrix¹. We then write a matrix of weights as $\mathbf{W} = \mathbf{\Delta}_1(\boldsymbol{\theta}_1) + \mathbf{\Delta}_2(\boldsymbol{\theta}_2)$, where $\boldsymbol{\theta}_i$ is the low-dimensional parameters of the i th compression and $\mathbf{\Delta}_i$ is the corresponding decompression mapping. Formally, the $\mathbf{\Delta}$ 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 $\mathbf{W} = \mathbf{\Delta}(\boldsymbol{\theta})$ 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 $\boldsymbol{\theta} = (\mathbf{U}, \mathbf{V})$.

¹The general case of multiple compressions, possibly applied separately to each layer of a net and not necessarily in matrix form, follows in an obvious way. Throughout the paper we use \mathbf{W} or \mathbf{w} or w to notate matrix, vector or scalar weights as appropriate (e.g., \mathbf{W} is more appropriate for low-rank decomposition).

- 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:

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

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 \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 $\|\mathbf{W}_1 + \mathbf{W}_2\|_0 \leq 2\kappa$).

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 appendix C.1.1).

3.1 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 [55], and image processing [6]. This combination enjoys certain theoretical guarantees [7, 10], 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.

3.2 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 Δ , 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 [31], but we implement the augmented Lagrangian one (which works in a similar way but with the introduction of a Lagrange multiplier vector λ of the same dimension as \mathbf{w}). We then optimize the following while driving a penalty parameter $\mu \rightarrow \infty$:

$$Q(\mathbf{w}, \boldsymbol{\theta}; \mu) = L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \Delta_1(\boldsymbol{\theta}_1) - \Delta_2(\boldsymbol{\theta}_2)\|^2 \quad (2)$$

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 $\Delta_1(\boldsymbol{\theta}_1) + \Delta_2(\boldsymbol{\theta}_2)$ 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:

$$\min_{\boldsymbol{\theta}} \|\mathbf{w} - \Delta(\boldsymbol{\theta})\|^2 \Leftrightarrow \min_{\boldsymbol{\theta}_1, \boldsymbol{\theta}_2} \|\mathbf{w} - \Delta_1(\boldsymbol{\theta}_1) - \Delta_2(\boldsymbol{\theta}_2)\|^2. \quad (3)$$

In the original LC algorithm [8], this step (over just a single compression $\Delta(\boldsymbol{\theta})$) 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 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 follows²:

$$\begin{aligned} \boldsymbol{\theta}_1 &= \arg \min_{\boldsymbol{\theta}} \|(\mathbf{w} - \Delta_2(\boldsymbol{\theta}_2)) - \Delta_1(\boldsymbol{\theta})\|^2 \\ \boldsymbol{\theta}_2 &= \arg \min_{\boldsymbol{\theta}} \|(\mathbf{w} - \Delta_1(\boldsymbol{\theta}_1)) - \Delta_2(\boldsymbol{\theta})\|^2 \end{aligned} \quad (4)$$

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 Δ_1 or Δ_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 [55], but not in general, as one would expect since some of the compression

²This form of iterated “fitting” (here, compression) by a “model” (here, Δ_1 or Δ_2) of a “residual” (here, $\mathbf{w} - \Delta_2(\boldsymbol{\theta}_2)$ or $\mathbf{w} - \Delta_1(\boldsymbol{\theta}_1)$) is called backfitting in statistics, and is widely used with additive models [14].

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
 $\theta_1, \theta_2 \leftarrow \arg \min_{\theta_1, \theta_2} \|\mathbf{w} - \Delta_1(\theta_1) - \Delta_2(\theta_2)\|^2$ init
for $\mu = \mu_0 < \mu_1 < \dots < \infty$
 $\mathbf{w} \leftarrow \arg \min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \Delta_1(\theta_1) - \Delta_2(\theta_2)\|^2$ L step
 while alternation does not converge }
 $\theta_1 \leftarrow \arg \min_{\theta_1} \|(\mathbf{w} - \Delta_2(\theta_2)) - \Delta_1(\theta_1)\|^2$ C step
 $\theta_2 \leftarrow \arg \min_{\theta_2} \|(\mathbf{w} - \Delta_1(\theta_1)) - \Delta_2(\theta_2)\|^2$
 if $\|\mathbf{w} - \Delta_1(\theta_1) - \Delta_2(\theta_2)\|$ is small enough **then** exit the loop
return $\mathbf{w}, \theta_1, \theta_2$

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, 42]. For nonconvex functions, convergence results are complex and more restrictive [38]. One simple case where convergence occurs is if the objective in (3) (i.e., each Δ_i is continuously differentiable and it has a unique minimizer over each θ_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.

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 4.1 (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 :*

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

Then the following provides one optimal solution (q^, s^*) : first set $q_i^* = \text{closest}(w_i)$ 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 $q_i^* = \text{closest}(w_i)$. 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 $s_i^* = 0$, $q_i^* = \text{closest}(w_i)$ (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 $q_i^* = \text{closest}(w_i)$ for every i , which is practically more desirable because it leads to a smaller ℓ_1 -norm of s . \square

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

	1bit Q + % P	$\log L$	$E_{\text{test}}(\%)$	ρ_s	ρ_+	ρ_\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
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

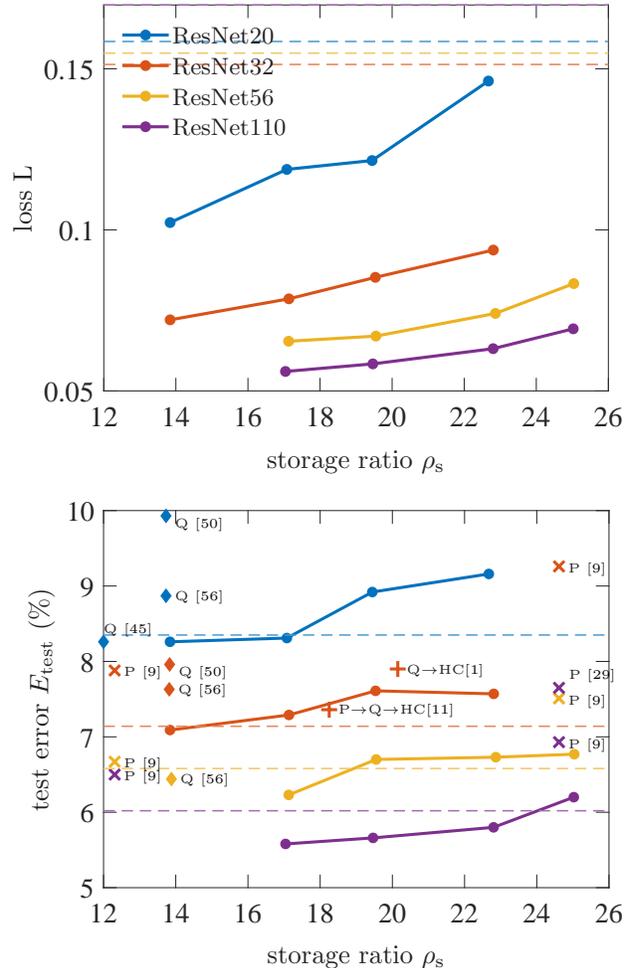


Figure 2: Q+P. *Left*: results of running 1-bit quantization with varying amounts of additive pruning (corrections) on ResNets of different depth on CIFAR-10 (with reference nets denoted **R**). We report training loss (logarithms are base 10), test error E_{test} (%), and ratios of storage ρ_s and floating point additions (ρ_+) and multiplications (ρ_\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 [45]” on the left border is outside the plot ($\rho_s < 12$).

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 [16].

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 [16] (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 [37] to the CIFAR10 dataset (see details in appendix C) and train it using the same data augmentation as for ResNets. The reference VGG16 has 15.2M parameters and

	1bit Q + rank r	$\log L$	E_{test} (%)	ρ_s	ρ_+	ρ_\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

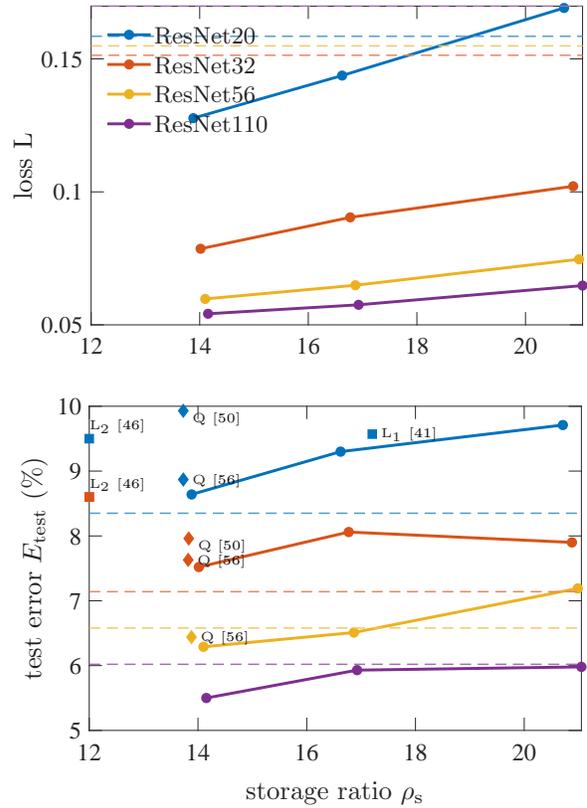


Figure 3: Q+L. *Left*: results of running 1-bit quantization with addition of a low-rank matrix of different rank on ResNets on CIFAR10. The organization is as for fig. 2. In the right-bottom plot we also show results from the literature for single compressions (Q: quantization, L: low-rank). Points “L [46]” on the left border are outside the plot ($\rho_s < 12$).

	Model	E_{test} (%)	ρ_s
	R VGG16	6.45	1.00
sum	rank 2 + 2% P	6.66	60.99
	rank 3 + 2% P	6.65	56.58
	pruning [29]	6.66	≈ 24.53
	filter pruning [27]	6.60	5.55
	quantization [34]	8.00	43.48

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.

achieves a test error of 6.45%.

The optimization protocol of our algorithm is as follows throughout all experiments with minor changes (see appendices C and D). To optimize the L step we use Nesterov’s accelerated gradient method [30] with momentum of 0.9 on minibatches of size 128, with a decayed learning rate schedule of $\eta_0 \cdot a^m$ at the m th 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 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 ρ_\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 appendix B for precise definitions and details of these metrics.

5.1 Q+P: quantization plus pruning

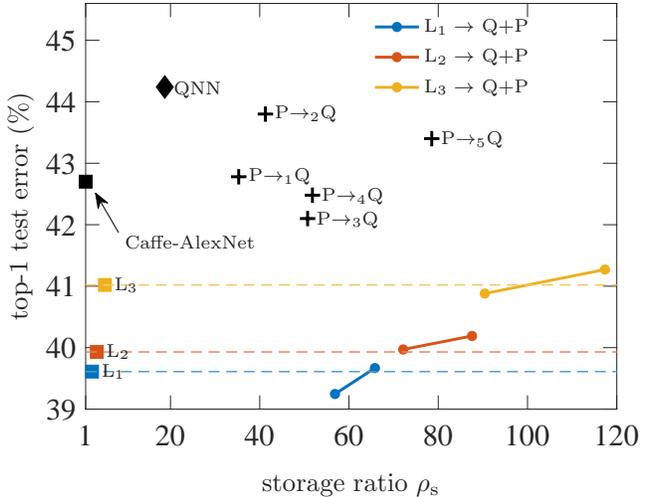
We compress ResNets with a combination of quantization plus pruning. Every layer is quantized separately with a codebook of size 2 (1 bit). For pruning we employ the constrained ℓ_0 formulation [9], which allows us to specify a single number of nonzero weights κ for the entire net (κ is the “% P” value 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 achieve considerable compression ratios ρ_s of up to $20\times$ 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 [50, 56] have $\rho_s \approx 14\times$ and lose up to 1% in test error comparing to the reference; the best unstructured pruning [9, 29] achieves $\rho_s \approx 12\times$ 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 $\rho_s = 13.84\times$. To the best of our knowledge, the highest compression ratio of comparable accuracy using only quantization is $6.22\times$ and has an error of 8.25% [45]. On ResNet110 with 1-bit quantization plus 3% corrections, we achieve 5.58% error while still compressing $17\times$.

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 $\rho_s = 20.15\times$ with 7.9% error, while we achieve $\rho_s = 22.81\times$ 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.

Model	top-1	MBs	MFLOPs
Caffe-AlexNet [21]	42.70	243.5	724
AlexNet-QNN [43]	44.24	13.0	175
P→ ₁ Q [13]	42.78	6.9	724
P→ ₂ Q [11]	43.80	5.9	724
P→ ₃ Q [39]	42.10	4.8	724
P→ ₄ Q [47]	42.48	4.7	724
P→ ₅ Q [47]	43.40	3.1	724
filter pruning [28]	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 (0.25M)	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

Figure 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.

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 convolutional layers are parameterized by low-rank as in Wen et al. [41]. 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 [41, 46]. Assuming storage with 16-bit precision on ResNet20, Wen et al. [41] achieve 17.20× storage compression (with 9.57% error) and Xu et al. [46] respectively 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 appendix D.

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 $117\times$ compression (2.075MB) without degradation in accuracy and $87\times$ compression with more than 2% improvement in the top-1 accuracy when compared to the Caffe-AlexNet. Recently, Yang et al. [47] (essentially using our Learning-Compression framework) reported $118\times$ and $205\times$ compression on AlexNet with none to small reduction of accuracy. However, as can be found by inspecting the code of Yang et al. [47], 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 ($52\times$ and $79\times$), 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 [18].

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.

Appendices

This appendix contains the following material: a) the pseudocode for augmented Lagrangian version of our algorithm (page 12) b) description of how we define the reduction ratios of storage, additions, and multiplications (page 12) c) description and results of the experiment compressing biases (page 14) d) full details of all reported experiments with extended tables and figures for the CIFAR10 (page 14) and ImageNet (page 23) datasets with description of Jetson Nano measurements (page 24)

A Pseudocode using augmented Lagrangian

In the main paper, we gave pseudocode for our algorithm using Quadratic Penalty formulation, here we give the pseudocode using the augmented Lagrangian version when there are two additive combinations.

```

input: training data, neural net architecture with weights  $\mathbf{w}$ 
 $\mathbf{w} \leftarrow \arg \min_{\mathbf{w}} L(\mathbf{w})$  reference net
 $\theta_1, \theta_2 \leftarrow \arg \min_{\theta_1, \theta_2} \|\mathbf{w} - \Delta_1(\theta_1) - \Delta_2(\theta_2)\|^2$  init as in C step
 $\lambda \leftarrow \mathbf{0}$  Lagrange multipliers
for  $\mu = \mu_0 < \mu_1 < \dots < \infty$ 
   $\mathbf{w} \leftarrow \arg \min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2} \|\mathbf{w} - \Delta_1(\theta_1) - \Delta_2(\theta_2) - \frac{1}{\mu} \lambda\|^2$  L step
  while alternation does not converge
     $\theta_1 \leftarrow \arg \min_{\theta_1} \left\| \left( \mathbf{w} - \Delta_2(\theta_2) - \frac{1}{\mu} \lambda \right) - \Delta_1(\theta_1) \right\|^2$  C step
     $\theta_2 \leftarrow \arg \min_{\theta_2} \left\| \left( \mathbf{w} - \Delta_1(\theta_1) - \frac{1}{\mu} \lambda \right) - \Delta_2(\theta_2) \right\|^2$  C step
   $\lambda \leftarrow \lambda - \mu (\mathbf{W} - \Delta_1(\theta_1) - \Delta_2(\theta_2))$  multipliers step
  if  $\|\mathbf{w} - \Delta_1(\theta_1) - \Delta_2(\theta_2)\|$  is small enough then exit the loop
return  $\mathbf{w}, \theta_1, \theta_2$ 

```

B Metrics

B.1 Compression ratio, storage

We define a storage *compression ratio* (ρ_s) as a ratio between bits required to store a reference model over bits of a compressed model:

$$\rho_s = \frac{\text{bits}(\text{reference})}{\text{bits}(\text{compressed})} \quad (6)$$

In this paper, we assume that the weights of a reference model are stored as 32-bit IEEE floating point numbers, therefore, the total bits for storing the reference model is

$$\text{bits}(\text{reference}) = \text{weights} \times 32, \quad (7)$$

here weights is the total number of parameters in the reference network. We discuss how we compute storage bits of compressed models, $\text{bits}(\text{compressed})$, next.

B.1.1 Quantization

When we quantize the weights with a codebook of size k , we need to store codebook itself and $\lceil \log k \rceil$ bits for every weight to index them from the codebook. In our experimental setup, we define a separate codebook for every layer with the same size k . The total compressed storage then:

$$\text{bits}_Q(\text{compressed}) = \underbrace{\text{layers} \times k \cdot 32}_{\text{codebook bits}} + \underbrace{\text{weights} \times \lceil \log k \rceil}_{\text{index bits}} \quad (8)$$

B.1.2 Sparse corrections (Pruning)

In additive combination sense, un-pruned weights are point-wise corrections. We store such corrections separately for each layer as a list of index-value pairs, where index is the location of a correction in a vector of flattened weights. Instead of storing indexes directly, we adopt storing differences between subsequent indexes, as in [13], using unsigned p -bit integers. If a difference is larger than $2^p - 1$, we add a dummy pair of zeros, i.e., $(0, 0)$; with such scheme storing some corrections would require multiple index-value pairs. We choose to store the values of the corrections as 16-bit IEEE floating point numbers. Then, the compressed storage for a layer is

$$\text{bits}_P(\text{layer}) = \text{diffs} \times (p + 16). \quad (9)$$

B.1.3 Low-rank

Rank r matrix of shape $m \times n$, is stored with two matrices of shapes $m \times r$ and $r \times m$ using 16-bit floating point numbers. The compressed storage for such layer is

$$\text{bits}_L(\text{layer}) = 16 \times r \times (m + n), \quad (10)$$

and the total bits for compressed storage is the sum over all layers.

B.1.4 Uncompressed parts

Since we only compress weights of a layer, some additional structures like biases and batch-normalization parameters would not be compressed. Therefore, we store them in full precision and add their storage cost to the total of compressed bits. In some cases, we can reduce the storage by fusing elements, e.g., biases can be fused with batch-normalization layer preceding it, which we take advantage of.

B.1.5 Additive combination

When compression combined additively, the total compressed storage is the sum of compression parts over all layers, e.g., for quantization with point-wise corrections the total compressed storage is:

$$\text{bits}(\text{compressed}) = \sum_l \left(\text{bits}_P(\text{layer } l) + \text{bits}_Q(\text{layer } l) \right) \quad (11)$$

B.2 Multiplications and additions, efficient implementation

We define reduction ratio of the number of floating point additions (ρ_+) and floating point multiplications (ρ_\times) as a ratio between the number of additions (and respectively multiplications) in reference model over the number of additions (multiplications) in a compressed model:

$$\rho_+ = \frac{\text{\#add in reference}}{\text{\#add in compressed}} \quad (12)$$

$$\rho_\times = \frac{\text{\#mult in reference}}{\text{\#mult in compressed}} \quad (13)$$

B.2.1 FLOPs for uncompressed model

A fully connected layer with weights \mathbf{W} of shape $n \times m$ and biases \mathbf{b} of shape $n \times 1$ requires the following number of multiplications and additions to compute a result of $\mathbf{W}\mathbf{x} + \mathbf{b}$:

$$\begin{aligned} \text{\#add} &= n \times (m - 1) + n = nm \\ \text{\#mult} &= nm. \end{aligned}$$

For a convolutional layer with shape $n \times c \times d \times d$ (here n filters with c channels and spatial resolution $d \times d$) and biases of size n , the total number of multiplications and additions are:

$$\begin{aligned} \text{\#add} &= ncd^2 \times M, \\ \text{\#mult} &= ncd^2 \times M, \end{aligned}$$

where M is the total number of one convolutional filter being applied to the input image.

B.2.2 Quantization

To perform efficient inference with quantized weights, for each neuron with weight vector \mathbf{w} we need to maintain an accumulator corresponding to each value of the codebook $\mathcal{C} = \{c_1 \dots c_k\}$:

$$\mathbf{w}^T \mathbf{x} = \sum_i w_i x_i = \sum_i c(w_i) x_i = \sum_k c_k \underbrace{\sum_{c(w_i)=c_k} x_i}_{\text{accumulate}} \quad (14)$$

Therefore for each neuron, there are k multiplications and the number of additions is equal to the number of the weights connected to the neuron. In the case of fully connected layers with m inputs and n outputs, there are following number of add./mult.:

$$\begin{aligned} \text{\#mult} &= k \times n \\ \text{\#add} &= m \times n \end{aligned}$$

For convolutional layers with n filters of shape $c \times d \times d$, we respectively have:

$$\begin{aligned} \text{\#mult} &= k \times n \times M, \\ \text{\#add} &= ncd^2 \times M, \end{aligned}$$

where M is the total number of the applications of conv. filters to the input.

B.2.3 Pruning

When the weight matrix \mathbf{W} has only p non-zero items, it will require p multiplications and $p-1$ additions for matrix-vector product; for convolutional layer these numbers should be multiplied by number of applications of conv filter, M (see above).

B.2.4 Low-rank

A rank r matrix of shape $n \times m$ can be represented by two matrices of shapes $n \times r$ and $r \times m$, therefore, during the inference of $\mathbf{W}\mathbf{x} + \mathbf{b}$ we have $\text{\#mult} = \text{\#add} = r(n + m)$.

C Experiments on CIFAR10

In this section, we give full details of our experimental setup for the CIFAR10 networks reported in the paper, as well as extended analysis of the results.

C.1 Quantization plus pruning, Q+P

C.1.1 Compressing biases together with weights

We would like to verify whether point-wise corrections are going to recover bad compression decisions. One such decision is to quantize both weights and biases with a single codebook. We report the results of such compression in Table 2. As you can see with few corrections (say, 0.65%) most of the biases are “recovered” from the bad quantization decision. Below we give a full experimental setup.

Reference model We train a multinomial logistic regression classifier on the CIFAR10 dataset (60k color images of 32×32 pixels, 10 classes). We use Nesterov’s SGD with a batch size of 1024 and learning rate of 0.05 which decayed after every epoch by 0.98; we use momentum of 0.9 and run the training for 300 epochs. We preprocess images by standardizing pixel-wise means and variances. The model has 30730 weights (3072×10 weights and 10 biases), and achieves train loss of 1.5253 and test accuracy of 38.79%.

Model	L	test acc, %	corrected biases (out of 10)
R logistic regression	1.5253	38.79	
1-bit quant. + 100 corrections	1.6830	36.06	9
1-bit quant. + 200 corrections	1.6716	37.29	9

Table 2: Results of running 1-bit quantization with corrections on simple multinomial logistic regression model (**R**) trained on the CIFAR10, where both weights and biases are compressed jointly with a single codebook. We report train loss L , test accuracy, and the number of corrections acting on a total of 10 biases. When we allow correcting only 100 (0.16%) values 9 out of 10 biases are already corrected. This indeed confirms that a) biases are important, and compressing them requires higher precision b) point-wise corrections are able to “fix” bad compression decisions.

Compression setup We run our algorithm with quantization and point-wise corrections on the weights and biases jointly. The codebook has $k = 2$ entries and we vary the number of corrections. The algorithm runs for 50 LC steps with $\mu = \times 5 \times 10^{-4} \times 1.1^k$ at k -th step. Each L-step is performed by Nesterov’s SGD with momentum 0.9 and run for 20 epochs with the initial learning rate of 0.05 decayed by 0.98 after each epoch. We do not perform any finetuning. Running the LC algorithm is 3.33 times longer comparing to the training of the reference model. We report results (loss and accuracies) corresponding to the model with the smallest train loss seen during the training, which is usually on the last iteration. For each C-step, we alternate between Q and P compressions 30 times.

C.1.2 ResNet experiments

We train ResNets [16] of depth 20, 32, 56, and 110 layers (0.27M, 0.46M, 0.85M, and 1.7M parameters, respectively) on the CIFAR10 dataset using the same augmentation setup as in [16]. Images in the dataset are normalized to have channel-wise zero mean, variance 1. For training, we use a random horizontal flip, zero pad the image with 4 pixels on each side and randomly crop 32×32 image out of it. For test, we use normalized images without augmentation. We report results obtained at the end of the training. The loss is average cross entropy with weight decay (as in the original paper).

Training reference nets The models are trained with Nesterov’s SGD [30] with a momentum of 0.9 on the minibatches of size 128. The loss is average cross entropy with weight decay of 10^{-4} ; weights initialized following [15]. Each reference network is trained for 200 epochs with a learning rate of 0.1 which is divided by 10 after 100 and 150 epochs.

Training compressed models We run our algorithm for 50 LC iterations on the ResNet-20/32, and for 45 iterations on the ResNet-56/110, with $\mu = \times 10^{-3} \times 1.1^k$ at k -th step. Each L-step is performed by Nesterov’s SGD with a momentum of 0.9 and runs for 20 epochs with a learning rate of 0.01 at the beginning of the step and decayed by 0.94 after each epoch. We do not perform any finetuning. Running the LC algorithm is 5 times longer comparing to the training of the reference network. We report results (train loss and test error) corresponding to the networks with the smallest train loss seen during the training, which is usually the last iteration. For each C-step, we alternate between Q and P compressions 30 times.

In our compression setup, each layer is quantized with its own codebook of size 2, however, corrections are applied throughout, to all weights with a predefined κ . We quantize only weights, and not the biases. For ResNet 20 and 32, we chose κ values to be 1, 2, 3, 5 % of the total number of parameters. For ResNet 56 and 110, we chose κ values to be 0.5, 1, 2, 3 % accordingly. The results are given in Table 3, and we compare our results to others in Fig.5. We additionally plot how pointwise corrections affect the weight of each layer in Fig. 6.

C.2 Quantization plus low-rank, Q+L

We run quantization with low-rank corrections on ResNet-s of depth 20, 32, 56, and 110. Each layer is quantized with its own codebook of size 2 applied to weights and corrected with r -rank matrices. We set r

	Model	$\log L$	$E_{\text{test}}, \%$	ρ_s	ρ_+	ρ_\times
ResNet20	R	-0.80	8.35	1.00	1.00	1.00
	1-bit quant. + 1.0% correction	-0.84	9.16	22.67	0.97	30.74
	1-bit quant. + 2.0% correction	-0.92	8.92	19.44	0.96	19.74
	1-bit quant. + 3.0% correction	-0.93	8.31	17.08	0.94	15.80
	1-bit quant. + 5.0% correction	-0.99	8.26	13.84	0.92	11.54
ResNet32	R	-0.82	7.14	1.00	1.00	1.00
	1-bit quant. + 1.0% correction	-1.03	7.57	22.81	0.97	30.52
	1-bit quant. + 2.0% correction	-1.07	7.61	19.54	0.96	19.85
	1-bit quant. + 3.0% correction	-1.10	7.29	17.14	0.94	15.80
	1-bit quant. + 5.0% correction	-1.14	7.09	13.84	0.92	11.56
ResNet56	R	-0.81	6.58	1.00	1.00	1.00
	1-bit quant. + 0.5% correction	-1.08	6.77	25.04	0.98	49.79
	1-bit quant. + 1.0% correction	-1.13	6.73	22.87	0.97	32.04
	1-bit quant. + 2.0% correction	-1.17	6.70	19.55	0.96	20.46
	1-bit quant. + 3.0% correction	-1.18	6.23	17.11	0.94	15.98
ResNet110	R	-0.77	6.02	1.00	1.00	1.00
	1-bit quant. + 0.5% correction	-1.16	6.20	25.03	0.99	55.63
	1-bit quant. + 1.0% correction	-1.20	5.80	22.80	0.98	33.94
	1-bit quant. + 2.0% correction	-1.23	5.66	19.47	0.96	23.27
	1-bit quant. + 3.0% correction	-1.25	5.58	17.04	0.95	17.84

Table 3: Results of running 1-bit quantization with corrections on the ResNets (**R**-s) of different depth. We report test error, the reduction ratio of storage(ρ_s), floating point additions (ρ_+), and multiplications (ρ_\times), under assumption of efficient implementation (section B.2); logarithms are base 10. We trade-off additions to multiplications, e.g., for ResNet110 with 3% corrections, the number of multiplications is decreased by factor of 17 with only a 5% increase in the number of additions. Also, notice that with a higher percentage of the corrections the compressed model achieves better than reference test error.

to have the same value throughout all layers.

We follow the procedure for the reference and compressed net training as described in section C.1.2 with minor changes. We use 45 LC iterations for all networks, however, learning rates are different: for ResNet20 we set learning rate at the beginning of each step to be 0.01 decayed by 0.94 after every epoch, and for other ResNet-s the initial learning rate for every L-step is 0.007 decayed by 0.94 after every epoch. The results are given in Table 4. We describe the low-rank parameterization of every convolutional layer next.

C.2.1 Low-rank parametrization of convolution

A convolutional layer having n filters of shape $c \times d \times d$ (here c is a number of channels and $d \times d$ is spatial resolution), can be seen as a linear layer with shape $n \times cd^2$ applied to the reshaped volumes of input. Its rank- r parametrization will have two linear mappings with weights $n \times r$ and $r \times cd^2$, which can be efficiently implemented as a sequence of two convolutional layers: the first with r filters of shape $c \times d \times d$ and the second with n filters of shape $r \times 1 \times 1$.

C.3 Low-rank plus pruning, L+P

We compress using a low-rank plus pruning combination the ResNet-s on CIFAR10 of depth 20, 32, 56, and 110; and the VGG16 adapted for the CIFAR10. Each layer is compressed with r -rank matrices and point-wise corrections. We set r to be the same throughout all layers; we set an amount of point-wise corrections (pruning) for the entire network in terms of percentage of a total number of weights. The results are in Table 6.

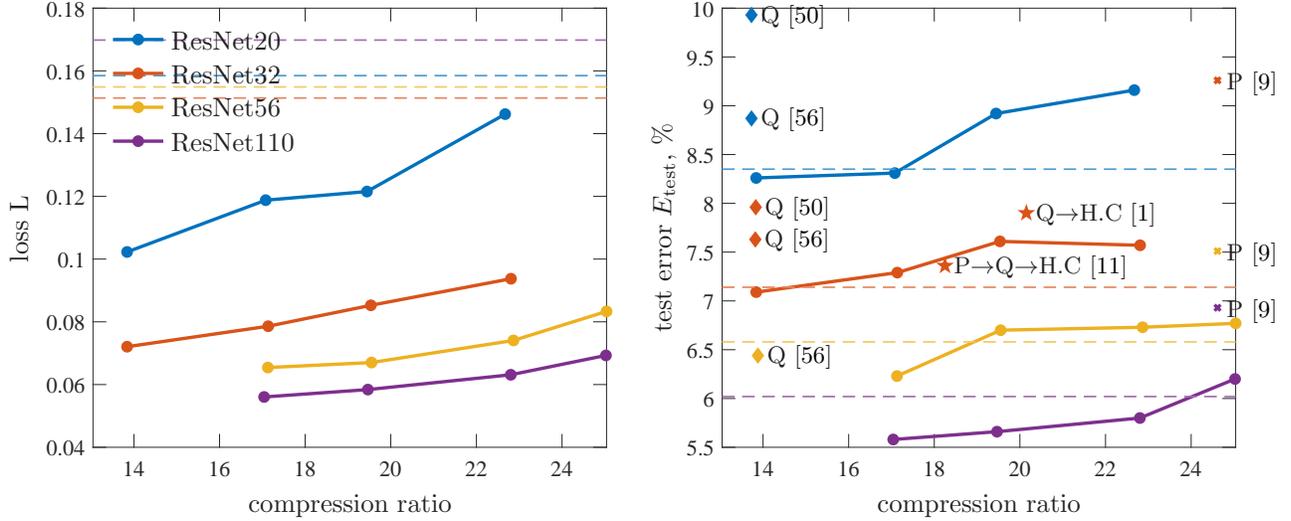


Figure 5: Results of running 1-bit quantization with corrections on ResNets of different depth, we plot train loss and test error as a function of compression (See Table 3 for more detail). Here, thick lines — our results, horizontal thin lines — reference nets. We also additionally indicate (via markers) results from the literature: Q — quantization, P—pruning, H.C — Huffman coding, right arrow indicates nesting the compression via “add” combination.

C.3.1 ResNets

We follow the procedure for reference and compressed net training as described in section C.1.2 with minor changes. We use 45 LC iterations for all networks, with $\mu_k = 5 \times 10^{-4} \times 1.1^k$ at k -th iteration. The learning rate at the beginning of each L-step is 0.01 and decayed by 0.94 after every epoch; we run each L-step for 20 epochs.

C.3.2 VGG-16

We train the VGG16 [37] adapted for the CIFAR10 dataset. We employ batch normalization after every layer except the last, and dropout after fully connected layers (see Table 5 for the full details). Images in the dataset are normalized channel-wise to have zero mean and variance one. For training, we use simple augmentation (random horizontal flip, zero pad with 4 pixels on each side and randomly crop 32×32 image). For test we use normalized images without augmentation. We report results corresponding to a model with the smallest loss. The loss is average cross entropy with ℓ_2 weight decay. The resulting net has 15M parameters.

Training the reference net The model is trained with Nesterov’s accelerated SGD [30] with momentum 0.9 on minibatches of size 128. The loss is average cross entropy with weight decay of 5×10^{-4} . The network is trained for 300 epochs with an initial learning rate of 0.05 decayed by 0.97716 after every epoch. The resulting test error is 6.45%.

Training compressed models Our algorithm is run for 50 LC iterations, with $\mu = 5 \times 10^{-4} \times 1.1^k$ at k -th iteration. Each L-step is performed by Nesterov’s SGD with momentum of 0.9 and runs for 20 epochs with a learning rate of 0.0007 at the beginning of the step; and decayed by 0.98 after each epoch. We do not perform any finetuning. Running the LC algorithm with finetuning is 3.4 times longer comparing to the training of the reference network. The results are presented in Table 6.

C.4 Quantization plus low-rank plus pruning, Q+L+P

In order to verify the complementarity benefits of additive compressions, we run experiments on ResNet-s to see whether adding another compression is going to help. As we saw in Table 4, the quantization with additive

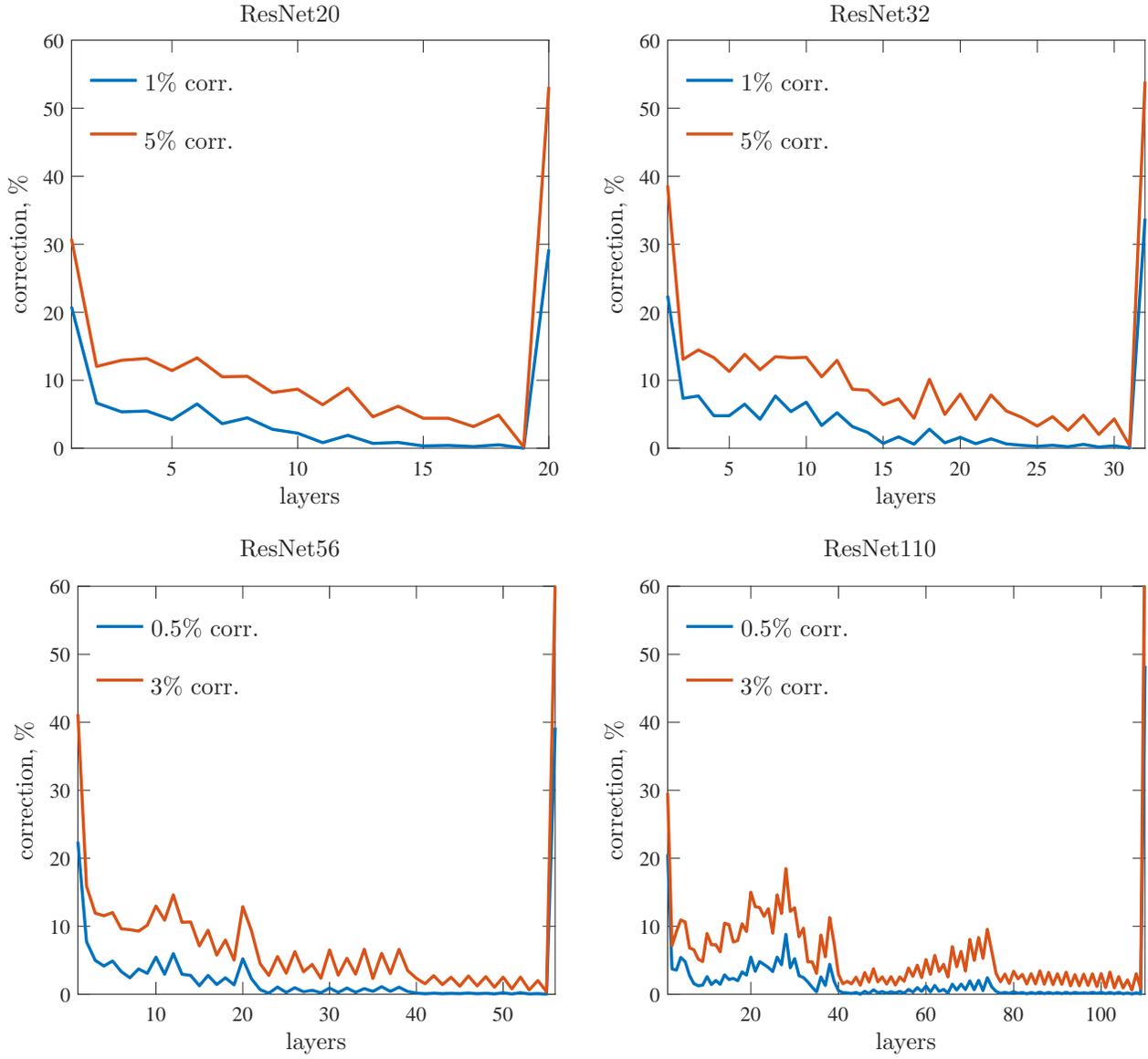


Figure 6: For every ResNet compressed with Q+P, we plot the proportion of weight being corrected at every level when the different amount of corrections are allowed. We see that the most corrected layers are the first and the last ones, which coincide with heuristics in the literature of not compressing the first and last layers.

	Model	$\log L$	$E_{\text{test}}, \%$	ρ_s	ρ_+	ρ_\times
ResNet20	R	-0.80	8.35	1.00	1.00	1.00
	1-bit quant. + rank = 1	-0.77	9.71	20.71	0.96	21.45
	1-bit quant. + rank = 2	-0.84	9.30	16.62	0.92	11.26
	1-bit quant. + rank = 3	-0.89	8.64	13.88	0.89	7.64
ResNet32	R	-0.82	7.14	1.00	1.00	1.00
	1-bit quant. + rank = 1	-0.99	7.90	20.94	0.97	21.89
	1-bit quant. + rank = 2	-1.04	8.06	16.81	0.92	11.47
	1-bit quant. + rank = 3	-1.10	7.52	14.04	0.89	7.44
ResNet56	R	-0.81	6.58	1.00	1.00	1.00
	1-bit quant. + rank = 1	-1.13	7.19	21.04	0.96	22.19
	1-bit quant. + rank = 2	-1.19	6.51	16.91	0.92	11.61
	1-bit quant. + rank = 3	-1.22	6.29	14.10	0.89	7.87
ResNet110	R	-0.77	6.02	1.00	1.00	1.00
	1-bit quant. + rank = 1	-1.19	5.98	21.11	0.96	22.38
	1-bit quant. + rank = 2	-1.24	5.93	16.96	0.92	11.70
	1-bit quant. + rank = 3	-1.27	5.50	14.18	0.89	7.92

Table 4: Results of running 1-bit quantization plus low-rank on ResNet-s (**R**-s) of different depth. We report test error, the reduction ratio of storage (ρ_s), floating point additions (ρ_+) and multiplications (ρ_\times), under assumption of efficient implementation (section B.2); logarithms are base 10. Notice how on the 110-layers ResNet, all compressed models achieve better than the reference test error.

low-rank (Q+L) achieves good performance on the ResNet110, having compressed models outperforming the reference. However, for smaller depth ResNets the Q+L scheme does not perform as good. We choose to compress these ResNets of depth 20, 32, 56 with Q+L scheme with a small number additional point-wise corrections (pruning), turning it to Q+L+P scheme. Table 7 shows the results.

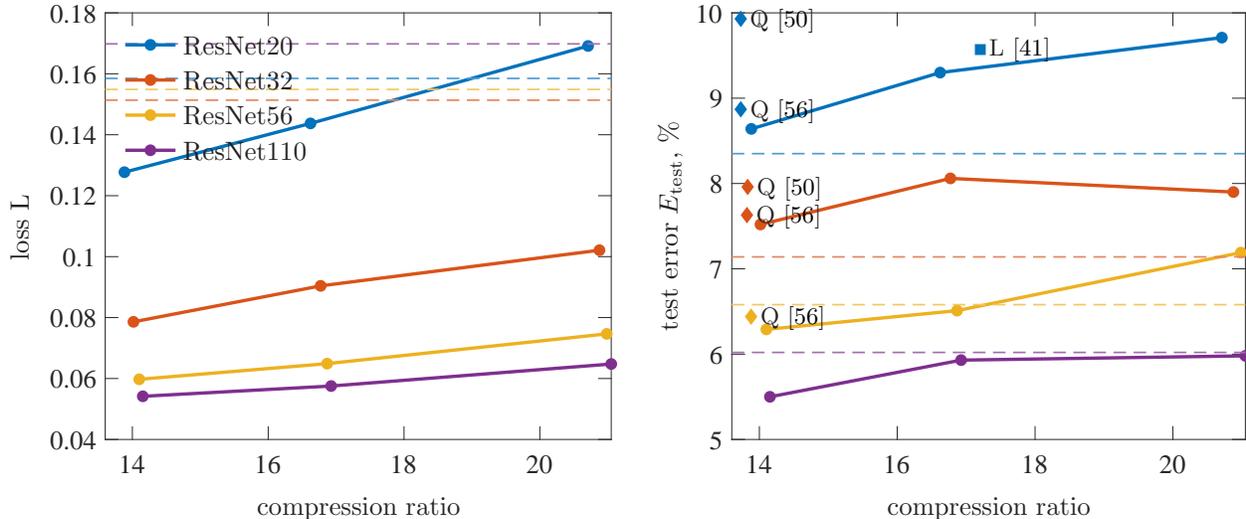


Figure 7: Results of running 1-bit quantization with corrections on ResNets of different depth, we plot train loss and test error as a function of compression (See Table 3 for more detail). Here, thick lines are our results, and horizontal thin lines are reference nets. We also additionally indicate (via markers) results from the literature involving quantization (Q) and low-rank (L).

Layer	Connectivity
Input	32×32 image
1	convolutional, 64 3×3 filters (stride=1), followed by BN and ReLU
2	convolutional, 64 3×3 filters (stride=1), followed by BN and ReLU max pool, 2×2 window (stride=2)
3	convolutional, 128 3×3 filters (stride=1), followed by BN and ReLU
4	convolutional, 128 3×3 filters (stride=1), followed by BN and ReLU max pool, 2×2 window (stride=2)
5	convolutional, 256 3×3 filters (stride=1), followed by BN and ReLU
6	convolutional, 256 3×3 filters (stride=1), followed by BN and ReLU
7	convolutional, 256 3×3 filters (stride=1), followed by BN and ReLU max pool, 2×2 window (stride=2)
8	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
9	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
10	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU max pool, 2×2 window (stride=2)
11	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
12	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU
13	convolutional, 512 3×3 filters (stride=1), followed by BN and ReLU max pool, 2×2 window (stride=2)
14	fully connected, 512 neurons and dropout with $p = 0.5$, followed by ReLU
15	fully connected, 512 neurons and dropout with $p = 0.5$, followed by ReLU
16	fully connected, 10 neurons, followed by softmax (output)
14981952 weights, 8970 biases, 8448 running means/variances for BN	

Table 5: Structure of the adapted VGG16 network for CIFAR10 dataset. BN–Batch Normalization, ReLU – rectified linear units.

	Model	$\log L$	$E_{\text{test}}, \%$	ρ_s	ρ_+	ρ_\times
RN-20	R	-0.80	8.35	1.00	1.00	1.00
	3% correction + rank = 3	-0.51	12.02	15.84	5.72	5.72
	3% correction + rank = 4	-0.55	11.44	13.34	4.70	4.70
RN-32	R	-0.82	7.14	1.00	1.00	1.00
	3% correction + rank = 3	-0.70	9.55	15.98	5.92	5.92
	3% correction + rank = 4	-0.79	9.14	13.47	4.85	4.85
RN-56	R	-0.81	6.58	1.00	1.00	1.00
	3% correction + rank = 3	-0.90	8.38	16.06	5.99	5.99
	3% correction + rank = 4	-0.98	8.02	13.54	4.90	4.90
RN-110	R	-0.77	6.02	1.00	1.00	1.00
	3% correction + rank = 3	-1.11	6.63	16.03	6.38	6.38
	3% correction + rank = 4	-1.14	6.36	13.53	5.17	5.17
VGG16	R	-0.89	6.45	1.00	1.00	1.00
	3% correction + rank = 2	-1.04	6.66	60.99	8.20	8.20
	3% correction + rank = 3	-1.05	6.65	56.58	7.81	7.81

Table 6: Results of running low-rank with pointwise corrections (L+P) on ResNets of different depth and VGG16. We report test error, the reduction ratio of storage (ρ_s), floating point additions (ρ_+) and multiplications (ρ_\times), under the assumption of efficient implementation (section B.2). RS stands for ResNet, and **R** for reference models; logarithms are base 10. In general, this type of compression performs much better on VGG16 than on ResNets.

	Model	$\log L$	$E_{\text{test}}, \%$	ρ_s	improvement Δ
RN20	R	-0.80	8.35	1.00	
	1-bit quant. + rank = 1 + 0.37% correction	-0.87	9.55	19.55	+0.16
	1-bit quant. + rank = 2 + 0.37% correction	-0.93	9.14	15.85	+0.16
RN32	R	-0.81	7.14	1.00	
	1-bit quant. + rank = 1 + 0.32% correction	-1.08	7.47	19.82	+0.43
	1-bit quant. + rank = 2 + 0.32% correction	-1.12	7.27	16.81	+0.79
RN56	R	-0.77	6.58	1.00	
	1-bit quant. + rank = 1 + 0.35% correction	-1.20	6.64	19.82	+0.55
	1-bit quant. + rank = 2 + 0.35% correction	-1.23	6.47	16.10	+0.04

Table 7: Results of running our algorithm with three compressions combined: quantization + low-rank corrections + point-wise corrections (Q+L+P). We choose to add a few pointwise compression to some of the Q+L experiments in Table 4. We report loss L , test error E_{test} , storage compression ratio (ρ_s), and improvement Δ of test error over Q+L scheme (higher is better). **R** stands for the reference network, and RS is shorthand for ResNet. As you can see, adding few point-wise corrections improve train loss and test error in all compression schemes.

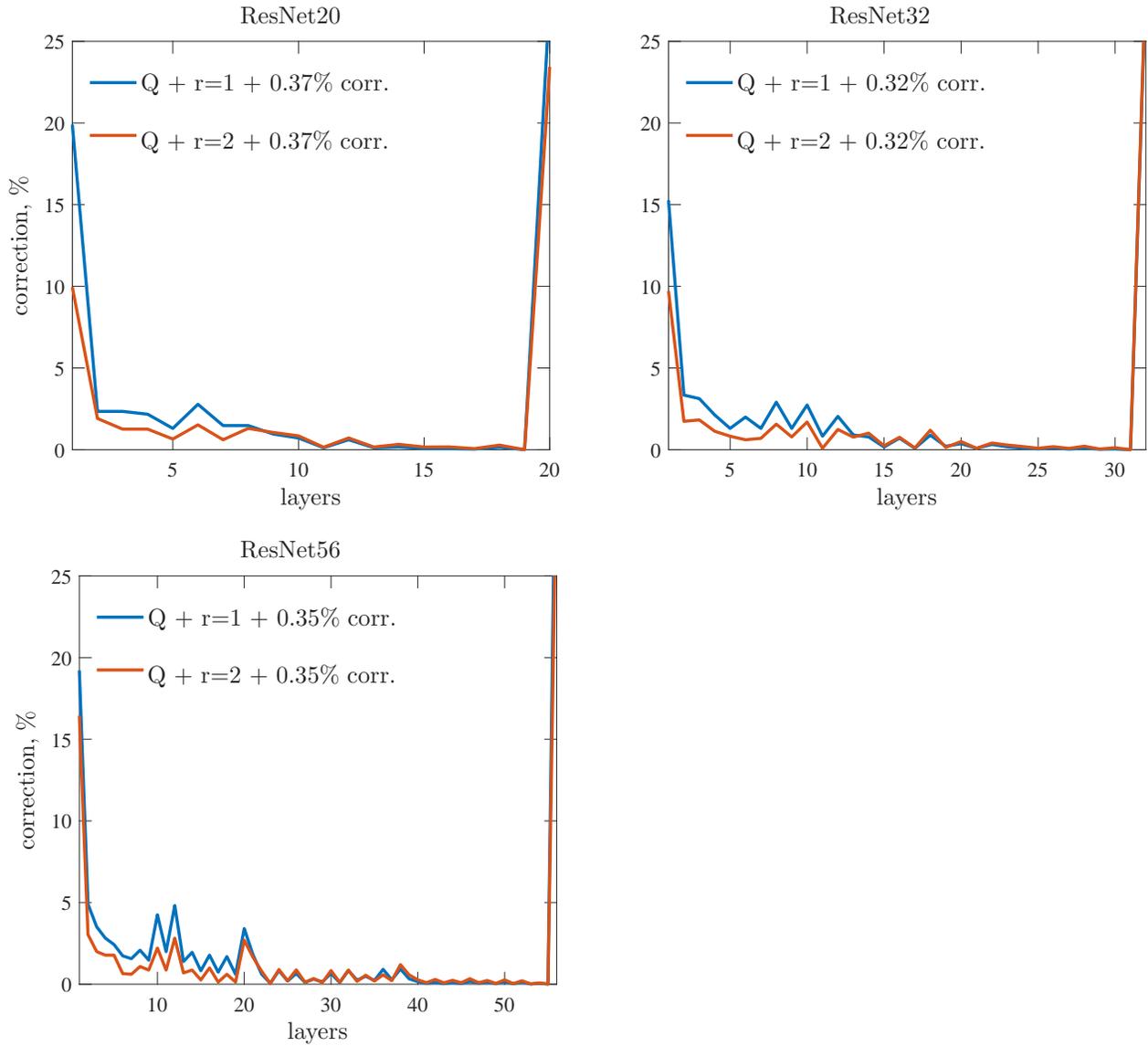


Figure 8: We plot here for every ResNet compressed with Q+L+P scheme, the proportion of weight being corrected at every layer when a different amount of rank corrections is applied. As rank increases, the corrections move to other places where it is needed most.

	Model	E top-1, %	E top-5, %	ρ_s
ours	R BN-AlexNet	40.43	17.55	1.00
	1-bit quant. + 0.5M corrections (0.8%)	39.09	16.84	25.96
	1-bit quant. + 1.0M corrections (1.6%)	38.94	16.69	22.42
	1-bit (DoReFa) [54]	46.10	23.70	10.35
	1-bit (BWN) [35]	43.20	20.59	10.35
	1-bit (ADMM) [25]	43.00	20.29	10.35
	1-bit (Quantization Net) [48]	41.20	18.30	10.35
	multi-bit AlexNet-QNN [43]	44.24	20.92	18.76

Table 8: Results of running 1-bit quantization with 0.5M and 1M corrections on AlexNet and comparison to the quantization methods in the literature. We report top-1 and top-5 errors on the validation set, the compression ratio of parameter storage (ρ_s). Notice that although results in the literature use 1-bit quantization, the compression ratio is rather small due to not compressing first and last layers, keeping them full precision. Also, notice that our compressed model achieves better than the reference validation error.

D Experiments on ImageNet

Due to the large number of possible combinations and scale of each experiment, we limit our attention to quantization with additive point-wise corrections combination (Q+P). Below we give details on training reference and compressed networks.

Training the reference net We train Batch Normalized AlexNet (having 1140 FLOPs) network using data-augmentation of the original paper [23] on ImageNet ILSVRC2012 dataset [36]. During training the images are resized to have the shortest side of 256 pixels length, and random 227×227 part of the image is cropped. Then pixel-wise color mean subtracted and image normalized to have standard variance. During test, we use a central 227×227 crop. We train with SGD on minibatches of size 256, with weight decay of 0.5×10^{-4} and an initial learning rate of 0.05, which is reduced $10\times$ after every 20 epochs. The trained model achieves Top-1 validation accuracy of 59.57% and Top-5 validation accuracy of 82.45%. This is slightly better than the the original paper [23] and Caffe re-implementation [21]. *Training time* on the NVIDIA Titan V GPU is 17 hours.

Training compressed models We train Q+P scheme with 1-bit quantization throughout (every layer having its own codebook) and total number of pointwise correction κ to be 0.5M and 1M. These models are trained for 35 LC steps, with $\mu = 5 \times 10^{-4} \times 1.12^k$ at k -th iteration. Each L-step is performed by Nesterov’s SGD with momentum 0.9 and runs for 10 epochs (with 256 images in a minibatch) with a learning rate of 0.001 at the beginning of the step and decayed by 0.94 after each epoch. Finally, we perform fine-tuning for 10 epochs with a learning rate of 0.001 decayed by 0.9 after every epoch, on minibatches of size 512. Running the LC algorithm with finetuning is 3.6 times longer comparing to the training of the reference network.

The resulting compressed models are quantized and have sparse corrections. To efficiently save them to disk, we utilized the sparse index compression technique described in Han et al. [13], and used the standard compressed save option of numpy library.

Table 8 summarizes our results and compares to existing quantization methods. We would like to note that we apply quantization throughout all layers, and allow compression mechanism (sparse additions) to self-correct at necessary positions. This is in contrast to quantization results on AlexNet where first and last layers are excluded from compression to keep overall performance from degradation. Thus, we achieve $25.96\times$ compression in parameter storage size without any accuracy loss, while best 1-bit quantization methods achieve only $10.35\times$ compression.

Although the Q+P scheme is quite powerful in representing the weights, its compression ratio with scalar quantization is limited by at most $32\times$, as the minimal amount of bits required to represent one weight is at least 1bit. If we would like to drive it further, compressions need to be nested, and in the following set

	Model	top-1 %	top-5 %	size MB	compression ratio, ρ_{size}	MFLOPs
	Caffe-AlexNet[13]	42.70	19.80	243.5	1	724
	Caffe-AlexNet-QNN [43]	44.24	20.92	13	18.7	175*
	P→Q [13]	42.78	19.70	6.9	35.2	724
	P→Q [11]	43.80	–	5.9	41.2	724
	P→Q [39]	42.10	–	4.8	50.7	724
	P→Q [47]	42.48	–	4.7	51.8	724
	P→Q [47]	43.40	–	3.1	78.5	724
	filter pruning [40]	–	21.63	–	–	231
	filter pruning [51]	44.13	–	–	–	232
	filter pruning [28]	43.17	–	232	1.04	334
	filter pruning [12]	43.83	20.47	–	–	492
	weight pruning [44]	44.10	–	13	18.7	–
	R Low-rank AlexNet (L ₁)	39.61	17.40	100.9	2.4	227
	L ₁ → Q (1-bit) + P (0.25M)	39.67	17.36	3.7	65.8	227
our	L ₁ → Q (1-bit) + P (0.50M)	39.25	16.97	4.3	56.9	227
	R Low-rank AlexNet (L ₂)	39.61	17.40	72.4	3.6	185
	L ₂ → Q (1-bit) + P (0.25M)	40.19	17.50	2.8	87.6	185
our	L ₂ → Q (1-bit) + P (0.50M)	39.97	17.35	3.4	72.15	185
	R Low-rank AlexNet (L ₃)	41.02	18.22	49.9	4.8	152
	L ₃ → Q (1-bit) + P (0.25M)	41.27	18.44	2.1	117.3	151
our	L ₃ → Q (1-bit) + P (0.50M)	40.88	18.29	2.7	90.4	151

Table 9: Q+P scheme is powerful enough to further compress already downsized models, here (bottom of the table) obtained by low-rank compression (L). We report top-1 validation error, size of the final model in MB, and resulting FLOPs. Shorthands are as follows: P stands for pruning, Q for quantization, L for low-rank, and H.C. for Huffman Coding. Numbers with * assumes efficient software/hardware implementation.

of experiments we show that we can capitalize on the power of the Q+P scheme and drive the compression further. To demonstrate it, we decided to further compress using the Q+P scheme the low-rank versions of AlexNet obtained from [17], see Table 9. We choose multiple low-rank networks achieving $3.20\times$ to $4.78\times$ reduction in FLOPs and applied our algorithm with varying amount of corrections (P) and fixed 1-bit quantization (Q).

D.1 Runtime measurements on Jetson Nano

The NVIDIA’s Jetson Nano³ is a small yet powerful edge inference device with quad core CPU and 128 core GPU requiring only 10 watts. We use its publicly available developer kit⁴ to run our experiments. The full characteristics of our setup is given on Figure 9. Since device runs the Ubuntu operating system, most of the software is readily available. However, the deep-learning libraries require specific configuration and re-compilation to run on the available GPU. The pre-configured libraries are made available by NVIDIA’s software teams.

D.1.1 Runtime evaluation

We evaluated the models on both CPU and GPU. We run the inference task (the forward pass) of tested neural networks using the batch size of 1, i.e., a single image, using 32 bit floating point values. For the CPU inference we converted all models into the ONNX inter-operable neural network format and used highly

³<https://developer.nvidia.com/embedded/jetson-nano>

⁴<https://developer.nvidia.com/embedded/jetson-nano-developer-kit>

characteristics		photo of the actual board
CPU	Quad-core ARM Cortex-A57 at 1479 MHz	
GPU	128 CUDA cores at 922 MHz	
RAM	4 GB 64-bit LPDDR4 at 1600MHz	
OS	Ubuntu 18.04.5 LTS	
Kernel	GNU/Linux 4.9.140-tegra	
Storage	128 GB microSDXC memory card (class UHS-I)	
Software	PyTorch v1.6.0, TensorFlow v2.2.0, ONNXRuntime v1.4.0, TensorRT v7.1.3.0	

Figure 9: The characteristics of the Jetson Nano developer kit we have used and a photo of the actual experimental setup.

Model	Top-1 err, %	Edge CPU		Edge GPU	
		time, ms	speed-up	time, ms	speed-up
Caffe-AlexNet	42.70	328.69	1.00	23.27	1.00
L ₁ → Q (1-bit) + P (0.25M)	39.67	83.62	3.93	11.32	2.06
L ₂ → Q (1-bit) + P (0.25M)	40.19	60.49	5.43	8.74	2.66
L ₃ → Q (1-bit) + P (0.25M)	41.27	44.80	7.33	6.72	3.46

Table 10: Runtime measurements of forward pass on Jetson Nano and corresponding speed-up when compared to Caffe-Alexnet.

optimized ONNXRuntime. For the GPU inference we used the TensorRT library to convert the ONNX models into TensorRuntime models. To account for various operating systems scheduling issues and context switches, we repeated the measurements N times and dropped p -proportion of the largest measurements. For the CPU measurements we used $N = 100$ and for the GPU we used $N = 1000$. We set $p = 0.1$, i.e., the highest 10% of the measurements will not be used in calculating the average delay. To avoid the influence of the thermal throttling, we allowed a cooldown period of at least 30 seconds between the measurements of different neural networks.

References

- [1] E. Agustsson, F. Mentzer, M. Tschannen, L. Cavigelli, R. Timofte, L. Benini, and L. Van Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In I. Guyon, U. v. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 1141–1151. MIT Press, Cambridge, MA, 2017.
- [2] J. M. Alvarez and M. Salzmann. Compression-aware training of deep networks. In I. Guyon, U. v. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 30, pages 856–867. MIT Press, Cambridge, MA, 2017.
- [3] A. Babenko and V. Lempitsky. Additive quantization for extreme vector compression. In *Proc. of the 2014 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR’14)*, pages 931–938, Columbus, OH, June 23–28 2014.
- [4] A. Beck and L. Tetruashvili. On the convergence of block coordinate descent type methods. *SIAM J. Optimization*, 23(4):2037–2060, 2013.
- [5] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Nashua, NH, second edition, 1999.

- [6] T. Bouwmans and E. hadi Zahzah. Robust principal component analysis via decomposition into low-rank and sparse matrices: An overview. In T. Bouwmans, N. S. Aybat, and E. hadi Zahzah, editors, *Handbook of Robust Low-Rank and Sparse Matrix Decomposition. Applications in Image and Video Processing*, chapter 1, pages 1.1–1.61. CRC Publishers, 2016.
- [7] E. J. Candès, X. Li, Y. Ma, and J. Wright. Robust principal component analysis? *Journal of the ACM*, 58(3):11, May 2011.
- [8] M. Á. Carreira-Perpiñán. Model compression as constrained optimization, with application to neural nets. Part I: General framework. arXiv:1707.01209, July 5 2017.
- [9] M. Á. Carreira-Perpiñán and Y. Idelbayev. “Learning-compression” algorithms for neural net pruning. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR’18)*, pages 8532–8541, Salt Lake City, UT, June 18–22 2018.
- [10] V. Chandrasekaran, S. Sanghavi, P. A. Parrilo, and A. S. Willsky. Rank-sparsity incoherence for matrix decomposition. *SIAM J. Optimization*, 21(2):572–596, 2010.
- [11] Y. Choi, M. El-Khamy, and J. Lee. Towards the limit of network quantization. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017.
- [12] X. Ding, G. Ding, Y. Guo, J. Han, and C. Yan. Approximated oracle filter pruning for destructive CNN width optimization. In K. Chaudhuri and R. Salakhutdinov, editors, *Proc. of the 36th Int. Conf. Machine Learning (ICML 2019)*, pages 1607–1616, Long Beach, CA, June 9–15 2019.
- [13] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016.
- [14] T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*. Number 43 in Monographs on Statistics and Applied Probability. Chapman & Hall, London, New York, 1990.
- [15] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In *Proc. 15th Int. Conf. Computer Vision (ICCV’15)*, pages 1026–1034, Santiago, Chile, Dec. 11–18 2015.
- [16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR’16)*, pages 770–778, Las Vegas, NV, June 26 – July 1 2016.
- [17] Y. Idelbayev and M. Á. Carreira-Perpiñán. 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)*, pages 8046–8056, Seattle, WA, June 14–19 2020.
- [18] Y. Idelbayev and M. Á. Carreira-Perpiñán. A flexible, extensible software framework for model compression based on the LC algorithm. arXiv:2005.07786, May 15 2020.
- [19] Y. Idelbayev and M. Á. Carreira-Perpiñán. 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)*, pages 3250–3254, Toronto, Canada, June 6–11 2021.
- [20] Y. Idelbayev and M. Á. Carreira-Perpiñán. More general and effective model compression via an additive combination of compressions. In *Proc. of the 32nd European Conf. Machine Learning (ECML–21)*, Bilbao, Spain, Sept. 13–17 2021.
- [21] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. arXiv:1408.5093 [cs.CV], June 20 2014.

- [22] H. Kim, M. U. K. Khan, and C.-M. Kyung. Efficient neural network compression. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 12569–12577, Long Beach, CA, June 16–20 2019.
- [23] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 25, pages 1106–1114. MIT Press, Cambridge, MA, 2012.
- [24] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems (NIPS)*, volume 2, pages 598–605. Morgan Kaufmann, San Mateo, CA, 1990.
- [25] C. Leng, H. Li, S. Zhu, and R. Jin. Extremely low bit neural network: Squeeze the last bit out with ADMM. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, pages 3466–3473, New Orleans, LA, Feb. 2–7 2018.
- [26] F. Li, B. Zhang, and B. Liu. Ternary weight networks. arXiv:1605.04711, Nov. 19 2016.
- [27] H. Li, A. Kadav, I. Durdanovic, and H. P. Graf. Pruning filters for efficient ConvNets. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017.
- [28] J. Li, Q. Qi, J. Wang, C. Ge, Y. Li, Z. Yue, and H. Sun. 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)*, pages 7046–7055, Long Beach, CA, June 16–20 2019.
- [29] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell. Rethinking the value of network pruning. In *Proc. of the 7th Int. Conf. Learning Representations (ICLR 2019)*, New Orleans, LA, May 6–9 2019.
- [30] Y. Nesterov. A method of solving a convex programming problem with convergence rate $\mathcal{O}(1/k^2)$. *Soviet Math. Dokl.*, 27(2):372–376, 1983.
- [31] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.
- [32] A. Novikov, D. Podoprikin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 28, pages 442–450. MIT Press, Cambridge, MA, 2015.
- [33] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, July 1992.
- [34] Z. Qu, Z. Zhou, Y. Cheng, and L. Thiele. Adaptive loss-aware quantization for multi-bit networks. In *Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20)*, pages 7988–7997, Seattle, WA, June 14–19 2020.
- [35] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. XNOR-net: ImageNet classification using binary convolutional neural networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, *Proc. 14th European Conf. Computer Vision (ECCV'16)*, pages 525–542, Amsterdam, The Netherlands, Oct. 11–14 2016.
- [36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *Int. J. Computer Vision*, 115(3):211–252, Dec. 2015.
- [37] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.
- [38] P. Tseng. Convergence of a block coordinate descent method for nondifferentiable minimization. *J. Optimization Theory and Applications*, 109(3):475–494, June 2001.

- [39] F. Tung and G. Mori. 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)*, pages 7873–7882, Salt Lake City, UT, June 18–22 2018.
- [40] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 29, pages 2074–2082. MIT Press, Cambridge, MA, 2016.
- [41] W. Wen, C. Xu, C. Wu, Y. Wang, Y. Chen, and H. Li. Coordinating filters for faster deep neural networks. In *Proc. 16th Int. Conf. Computer Vision (ICCV'17)*, Venice, Italy, Dec. 11–18 2017.
- [42] S. J. Wright. Coordinate descent algorithms. *Math. Prog.*, 151(1):3–34, June 2016.
- [43] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng. Quantized convolutional neural networks for mobile devices. In *Proc. of the 2016 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'16)*, pages 4020–4028, Las Vegas, NV, June 26 – July 1 2016.
- [44] X. Xiao, Z. Wang, and S. Rajasekaran. AutoPrune: Automatic network pruning by regularizing auxiliary parameters. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems (NEURIPS)*, volume 32, pages 13681–13691. MIT Press, Cambridge, MA, 2019.
- [45] Y. Xu, Y. Wang, A. Zhou, W. Lin, and H. Xiong. Deep neural network compression with single and multiple level quantization. In *Proc. of the 32nd AAAI Conference on Artificial Intelligence (AAAI 2018)*, pages 4335–4342, New Orleans, LA, Feb. 2–7 2018.
- [46] Y. Xu, Y. Li, S. Zhang, W. Wen, B. Wang, Y. Qi, Y. Chen, W. Lin, and H. Xiong. TRP: Trained rank pruning for efficient deep neural networks. In *Proc. of the 29th Int. Joint Conf. Artificial Intelligence (IJCAI'20)*, pages 977–983, Yokohama, Japan, Jan. 21–15 2020.
- [47] H. Yang, S. Gui, Y. Zhu, and J. Liu. 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)*, pages 2175–2185, Seattle, WA, June 14–19 2020.
- [48] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, and X.-S. Hua. Quantization networks. In *Proc. of the 2019 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'19)*, pages 7308–7316, Long Beach, CA, June 16–20 2019.
- [49] J. Ye, X. Lu, Z. Lin, and J. Wang. Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers. In *Proc. of the 6th Int. Conf. Learning Representations (ICLR 2018)*, Vancouver, Canada, Apr. 30 – May 3 2018.
- [50] P. Yin, S. Zhang, J. Lyu, S. Osher, Y. Qi, and J. Xin. BinaryRelax: A relaxation approach for training deep neural networks with quantized weights. *SIAM J. Imaging Sciences*, 11(4):2205–2223, 2018.
- [51] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis. NISP: Pruning networks using neuron importance score propagation. In *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, pages 9194–9203, Salt Lake City, UT, June 18–22 2018.
- [52] X. Yu, T. Liu, X. Wang, and D. Tao. 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)*, pages 67–76, Honolulu, HI, July 21–26 2017.
- [53] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen. Incremental network quantization: Towards lossless CNNs with low-precision weights. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017.

- [54] S. Zhou, Z. Ni, X. Zhou, H. Wen, Y. Wu, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv:1606.06160, July 17 2016.
- [55] T. Zhou and D. Tao. GoDec: Randomized low-rank & sparse matrix decomposition in noisy case. In L. Getoor and T. Scheffer, editors, *Proc. of the 28th Int. Conf. Machine Learning (ICML 2011)*, pages 33–40, Bellevue, WA, June 28 – July 2 2011.
- [56] C. Zhu, S. Han, H. Mao, and W. J. Dally. Trained ternary quantization. In *Proc. of the 5th Int. Conf. Learning Representations (ICLR 2017)*, Toulon, France, Apr. 24–26 2017.