An Empirical Comparison of Quantization, Pruning and Low-rank Neural Network Compression using the LC Toolkit

Yerlan Idelbayev
Department of Computer Science and Engineering
University of California, Merced
Merced, CA, USA
yidelbayev@ucmerced.edu

Miguel Á. Carreira-Perpiñán
Department of Computer Science and Engineering
University of California, Merced
Merced, CA, USA
mcarreira-perpinan@ucmerced.edu

***Abstract*—Compression of machine learning models, and of neural networks in particular, has become an essential problem among practitioners. Many different approaches including quantization, pruning, low-rank and tensor decompositions have been proposed in the literature to solve the problem. Despite this, an important unanswered question remains: *what is the best compression scheme for a model?* As a step towards answering this question objectively and fairly, we empirically compare quantization, pruning, and low-rank compressions in the algorithmic footing of the Learning-Compression (LC) framework. This allows us to explore the compression schemes systematically and perform an apples-to-apples comparison along the entire error-compression tradeoff curves. We describe our methodology, the framework, experimental setup, and present our comparisons. Based on our experiments, we conclude that the choice of compression is strongly model-dependent: for example, VGG16 is better compressed with pruning, while quantization is more suitable for the ResNets. This, once again, underlines the need for a common benchmark of compression schemes with fair and objective comparisons of the models of interest.**

I. Introduction

Fueled by the recent progress in the fields of image-, audio/video-, and general signal processing, machine learning models and, in particular, deep neural networks are finding their places in everyday devices like smartphones, wearables, and internet of things (IoT) devices. The general trend enabling these models' success, the ever-increasing sizes and computational requirements backed by high performance hardware like GPUs and TPUs, comes as a challenge when these models are needed to be deployed into intelligent devices with stringent constraints in terms of power, CPU clock speed, or other resource limits. The neverending arms race between larger models and smaller devices poses the question of model compression: how to deploy a trained model into a smaller device while satisfying the device constraints and yet maintaining the accuracy of the model?

In the literature, many solutions have been proposed to the model compression problem. Some of the popular approaches include various forms of quantization [1]–[3] (like binarization [4], ternarization [5], [6], fixed-point quantization [7]–[10]); structured and unstructured pruning [11]–[16]; low-rank and tensor decompositions [17]–[20], [20]–[30]; and different combinations of those (e.g., prune then quantize combination of Han et al. [31] and others [32]). Among these research directions, *the fundamental problem is that in practice, we do not know what type of compression (or their combination) is the best for a model.* As a step towards answering this question, in this paper, we task ourselves with a systematic empirical comparison of common compression mechanisms on popular neural networks. We believe such a comparison is long needed in the community, and our paper fills the gap by establishing a robust yet an easy to perform benchmark.

To perform the comparison, in principle, it may be possible to try different existing (possibly off-the shelf) compression schemes with corresponding algorithms, but practically it is often impossible due to several factors:

1) *Fairness and objectivity of comparison*. Available compression methods have different algorithmic foundations: for instance, while popular quantization algorithms revert to modification of the gradient during forward or backward pass (e.g., [1], [33]), many pruning methods involve adding penalties to the loss function [14], [34]. Differences in the algorithms prevent us from attributing the success of the compression to the mechanism itself, its optimization, or other confounding yet unknown factors.
2) *Availability of the generally applicable codes*. While a significant portion of the research papers share the actual implementations of their methods, these codes are heavily tuned to work with a particular architecture used in the paper itself. Modification of these codes require a substantial effort which prevents researchers from easily applying the method to a new architecture/network.
3) *The choice of hyperparameters*. Often the important hyperparamters of the compression mechanisms (e.g., the amount of thresholding for ℓ_1 pruning) are left undisclosed in the description of the method. Even if the hyperparameters are fully revealed, the underlying reason of why the hyperparameters were chosen in a certain way is not discussed which prevents one from applying it to a new compression task.

To address the aforementioned challenges, for our systematic comparison we use our open source toolkit[1] based on the ideas of Learning Compression (LC) algorithm [35]. The LC framework allows to seamlessly integrate number of compression types under the same algorithmic foundation, including several forms of pruning, quantization and low-rank factorizations while simultaneously achieving state-of-the-art results with these compressions as described in a series of papers [2], [30], [35]–[41]. Such integration is possible due to the general mathematical formulation of the model compression problem employed in our framework which is combined with an alternating optimization solution. This results in an algorithm, which alternates the solution of two well-defined problems. One problem has the form of a standard model training (neural network learning) using the original loss and dataset but with an additional regularization term, (the *learning (L) step*). The other problem has the form of a standard signal processing problem involving the model parameters but not the loss or dataset, which can be solved using an existing algorithm for the chosen form of compression (the *compression (C) step*).

The decoupling of model training (L step) from model compression (C step) allows us to perform highly controlled experiments while exploring different compression mechanisms along their error-compression tradeoff curves: once the L-step solution is available, trying a new compression mechanism is equivalent to combining the L step with a new, different C step. This, in turn, allows us to make the apples-to-apples comparisons: we use single, fixed set of hyperparameters for the L steps, thus, eliminating any confounding effects of the underlying optimization.

The reminder of this paper is organized as follows. In section II we give a detailed overview of Learning-Compression algorithm followed by the overview of its software implementation in section II-A. In section III we fill in the details of compression forms and the corresponding C-step solutions. In section IV we describe our experimental setup and present comparisons of pruning, low-rank, and quantization compressions on multiple networks of interest.

II. Overview of the Learning Compression Framework

Let us assume we are given a neural network with weights $\mathbf{w}$ that has been trained to minimize some loss function L (e.g., cross-entropy). We will be referring to this well-trained uncompressed neural network as a *reference net*. The Learning Compression framework [35] defines the model compression as a constrained optimization problem in terms of low-dimensional parameters $\boldsymbol{\Theta}$ in the following way:

$$\min_{\mathbf{w},\boldsymbol{\Theta}} L(\mathbf{w}) + \lambda C(\boldsymbol{\Theta}) \quad \text{s.t.} \quad \mathbf{w} = \boldsymbol{\Delta}(\boldsymbol{\Theta}). \tag{1}$$

Here, $\boldsymbol{\Delta}$ is a *decompression mapping* from the low-dimensional (compressed) space of $\boldsymbol{\Theta}$ into the space of weights $\mathbf{w}$ (see sec. III for more details); the term $\lambda C(\boldsymbol{\Theta})$ with $\lambda > 0$ captures the cost of the compressed model in terms of the costs of interest like size, energy consumption, etc. We illustrate this definition pictorially for the case of $\lambda = 0$ in Fig. 1.

[1]https://github.com/UCMerced-ML/LC-model-compression

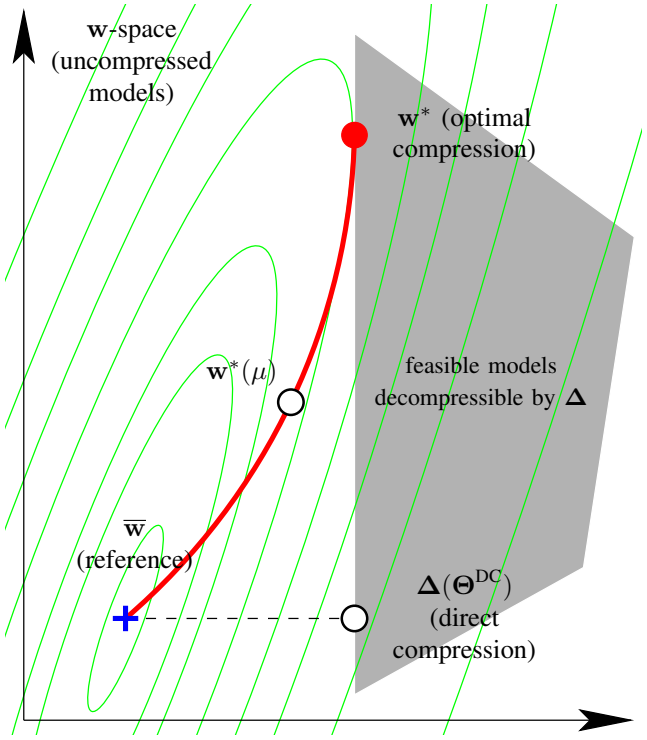


Fig. 1. The illustration of the model compression definition given by problem (1) when $\lambda = 0$. The loss function $L(\mathbf{w})$ is defined over entire $\mathbf{w}$-space, depicted with green contours, and has a minimum at point $\overline{\mathbf{w}}$. The space of decompressible models (given by the form of of $\boldsymbol{\Delta}$) is illustrated in gray. To obtain the constrained minima of the problem (the point $\mathbf{w}^*$), the LC algorithm alternates between L and C steps while driving parameter $\mu \to \infty$ forming path $\mathbf{w}^*(\mu)$.

The problem (1) is a constrained optimization problem involving usually non-convex loss L (due to an underlying neural network) and the problem is potentially non-differentiable due to the compression term $\boldsymbol{\Delta}(\boldsymbol{\Theta})$ (for instance when we use binarization as a form of compression). To efficiently handle this problem we bring the equality constraints into the objective using a penalty method [42, ch. 17] and optimize the following while driving $\mu \to \infty$:

$$\min_{\mathbf{w},\boldsymbol{\Theta}} L(\mathbf{w}) + \lambda C(\boldsymbol{\Theta}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|. \tag{2}$$

The penalty formulation (2) now admits alternating optimization solution if minimized separately over $\mathbf{w}$ and $\boldsymbol{\Theta}$, revealing L and C steps:

- **L (learning) step:** $\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$. This is a regular learning problem of the uncompressed model but with a quadratic regularization term. *The L step is independent from the form of chosen compression.*
- **C (compression) step:** $\min_{\boldsymbol{\Theta}} \|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2 + \lambda C(\boldsymbol{\Theta})$. When $\lambda = 0$ it means finding the best lossy compression of the current uncompressed model weights $\mathbf{w}$ in the ℓ_2 sense: as we show next, many compressions of interest

input training data and model with parameters $\mathbf{w}$

$\mathbf{w} \leftarrow \overline{\mathbf{w}} = \arg\min_{\mathbf{w}} L(\mathbf{w})$ pretrained model

$\boldsymbol{\Theta} \leftarrow \boldsymbol{\Theta}^{\mathrm{DC}} = \boldsymbol{\Pi}(\overline{\mathbf{w}})$ init compression

$\boldsymbol{\beta} \leftarrow \mathbf{0}$

for $\mu = \mu_0 < \mu_1 < \cdots < \infty$

  $\mathbf{w} \leftarrow \arg\min_{\mathbf{w}} L(\mathbf{w}) + \frac{\mu}{2}\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta}) - \frac{1}{\mu}\boldsymbol{\beta}\|^2$ L step

  $\boldsymbol{\Theta} \leftarrow \arg\min_{\boldsymbol{\Theta}} \|\mathbf{w} - \frac{1}{\mu}\boldsymbol{\beta} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2 + \lambda C(\boldsymbol{\Theta})$ C step

  $\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \mu(\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta}))$ multipliers step

  if $\|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|$ is small enough **then** exit the loop

return $\mathbf{w}$, $\boldsymbol{\Theta}$

Fig. 2. Full pseudocode of Learning Compression algorithm using the augmented Lagrangian.

appear as well-formulated and well-studied problems in the C step. The C-step solution *only* depends on the actual form of of the compression scheme $\boldsymbol{\Delta}(\boldsymbol{\Theta})$ and the structure of the cost $C(\boldsymbol{\Theta})$.

Here and throughout the paper we use the quadratic penalty method as our penalty function. However, in practice we implement the augmented Lagrangian version which has an additional vector of multipliers, $\boldsymbol{\beta}$, of the same dimension as the weights $\mathbf{w}$. The full pseudocode of LC algorithm is given in Fig 2.

Overall, the LC algorithm works by alternating between L and C steps while slowly driving the penalty parameter $\mu \to \infty$: this traces a solution path $\mathbf{w}^*(\mu)$ as depicted in Fig. 1. The convergence of this path to the stationary point of (1) were given under standard assumptions in [35]. Empirically, we observed that it is sufficient to follow this path approximately, using an exponential schedule on μ-values of $\mu_k = a \times b^t$ for the tth step.

A. LC Toolkit

The LC Toolkit [43] is our in-house implementation of the LC algorithm available under the open-source BSD 3-clause license. It is written in Python using PyTorch and contains all necessary building blocks to compress any neural network. In the toolkit, the implementation of the L step is handed off to the user through Python's anonymous functions. A typical implementation of the L step would look as follows:

```
def my_l_step(model, lc_penalty, args**):
    # some code is skipped
    loss = model.loss(out_, target_) + lc_penalty()
    loss.backward()
    optimizer.step()
    # some code is skipped
```

Here we skipped some housekeeping code for brevity. Notice that implementation of L step is no different from a regular learning of the neural network. Once the implementation of the L step is provided, *user can choose any compression (C step) available in the toolkit in a mix-and-match way*: the toolkit can handle multiple compression per layer as well as multiple layers per a single compression. We refer the reader to the full description of the toolkit's functionality in [43].

III. Compression Forms

In this section we briefly describe the forms of the compression we are using in our comparison (which are conveniently available in the LC toolkit as well). Fig. 3 pictorially shows different compression schemes.

A. Quantization

Quantization is the process of assigning the weights to a smaller set of codebook entries $\mathcal{C}$: such allocation allows to store weights more compactly. We consider the general case of quantization when the weights of the model are compressed with a learned codebook $\mathcal{C} = \{c_1, c_2, \ldots, c_K\}$ of size K. By using a binary assignment variables $\mathbf{z}_i$ (for each $w_i \in \mathbf{w}$) of the form $\sum_k z_{ik} = 1$ and $z_{ik} \in \{0, 1\}$ we can equivalently reformulate the model quantization problem as:

$$\min_{\mathbf{w},\mathcal{C},\mathbf{z}_1,\ldots\mathbf{z}_N} L(\mathbf{w}) \quad \text{s.t.} \quad w_i = \sum_{k=1}^{K} z_{ik} c_k, \quad \forall i = 1, \ldots, N.$$

This formulation now can be optimized using the he Learning-Compression algorithm. To see the equivalence to eq. (1) assume $\boldsymbol{\Theta} = (\mathcal{C}, \mathbf{z}_1, \ldots \mathbf{z}_N)$. The corresponding C-step problem of $\min_{\boldsymbol{\Theta}} \|\mathbf{w} - \boldsymbol{\Delta}(\boldsymbol{\Theta})\|^2$ has the form of:

$$\min_{\mathcal{C},\mathbf{z}_1,\ldots\mathbf{z}_P} \sum_{i=1}^{N}\sum_{k=1}^{K} z_{ik}(w_i - c_k)^2, \tag{3}$$

which has been thoroughly studied in signal compression and unsupervised clustering literature, and is known as the k-means clustering problem. Therefore, we call a standard k-means routing as the solution of the C-step for the quantization problem.

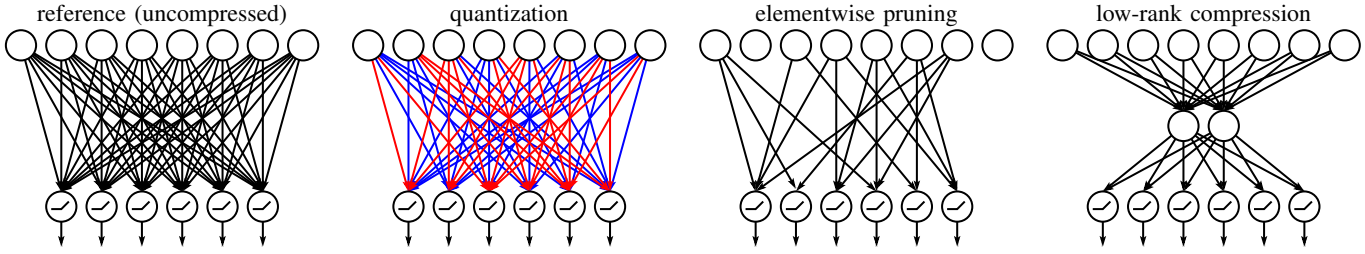


Fig. 3. Illustration of compressions forms we are exploring in this paper. The weights of a reference layer (left) can be represented in low-rank, pruned, or quantized forms. Here, black colored lines represent real valued weights; colored weights (red, blue) represent assignment to the same codebook values. The LC algorithm and the toolkit can handle all this compression schemes out of the box.

B. Elementwise pruning

Pruning is the process of removing (sparsifying) the weights of the model by settings some of them to be equal to 0. In this paper we use the ℓ_0-norm constrained pruning defined as:

$$\min_{\mathbf{w}} L(\mathbf{w}) \quad \text{s.t.} \quad \|\mathbf{w}\|_0 \leq \kappa. \tag{4}$$

Since the ℓ_0-norm is the count of non-zero items in the vector, the formulation of (4) allows us to precisely specify the number of remaining weights after the compression.

We bring (4) into the Learning-Compression form (1) by introducing a copy parameter $\boldsymbol{\theta}$ in the following way:

$$\min_{\mathbf{w}} L(\mathbf{w}) \quad \text{s.t.} \quad \mathbf{w} = \boldsymbol{\theta}, \quad \|\boldsymbol{\theta}\|_0 \leq \kappa,$$

for which the C step is given by solving:

$$\min_{\boldsymbol{\theta}} \|\mathbf{w} - \boldsymbol{\theta}\|^2 \quad \text{s.t.} \quad \|\boldsymbol{\theta}\|_0 \leq \kappa. \tag{5}$$

The solution of (5) can be obtained by selecting all but top-κ weights (in magnitude) of $\mathbf{w}$ and zeroing the remaining weights.

C. Low-rank compression with rank selection

Low-rank compression reparametrizes the weight matrix $\mathbf{W}$ of a layer as product of $\mathbf{U}\mathbf{V}^T$: this reduces both size and computational demands (for an appropriately small rank). However, such a compression requires knowing the right choice of the rank for each layer in the network. To address the issue, we use the automatic rank selection which is formulated as a suitable optimization problem.

Let us assume we have a model with M layers and the weight matrices $\mathbf{W}_l$ for layer $l = 1, \dots, M$. We collectively refer to this matrices as $\mathbf{w} = \{\mathbf{W}_1, \dots, \mathbf{W}_M\}$. We define the following model selection problem over the set of possible low-rank models:

$$\begin{aligned} \min_{\mathbf{w}} \quad & L(\mathbf{w}) + \lambda\, C(\mathbf{w}) \\ \text{s.t.} \quad & \operatorname{rank}(\mathbf{W}_m) = r_m \leq R_m, \quad \forall\, m = 1, \dots, M \end{aligned}$$

Here R_l is the maximum possible rank for matrix $\mathbf{W}_l$ and the compression cost $C(\mathbf{w})$ is defined in terms of the ranks of the matrices in the following way:

$$C(\mathbf{w}) = C(r_1) + C(r_2) + \dots + C(r_L). \tag{6}$$

Here, $C(r_l)$ measures the total parameters required for an r_l-rank matrix of a layer l. To put this rank-selection problem into Learning-Compression form (eq. 1), we introduce the parameter $\boldsymbol{\Theta}_l$ for each layer. The corresponding C-step problem then separates over the layers into M independent subproblems of:

$$\begin{aligned} \min_{\boldsymbol{\Theta}_l, r_l} \quad & \lambda\, C_l(r_l) + \frac{\mu}{2}\|\mathbf{W}_l - \boldsymbol{\Theta}_l\|^2 \\ \text{s.t.} \quad & \operatorname{rank}(\boldsymbol{\Theta}_l) = r_l \leq R_l. \end{aligned}$$

The solution of this C step was given in [36], and involves an SVD and enumeration over the ranks for each layer's weight matrix.

One remaining question is *how to handle the layers with non-matrix weights?* If the weights are stored in a tensor form (e.g., convolutional layers) they must be reshaped into a matrix. For convolutional layers, two reshaping schemes known as scheme 1 [20], [22], [24]–[26] and scheme 2 [21], [23], [26], [44] have been studied in the literature. Both schemes can be efficiently implemented as a sequence of two convolutional layers.

TABLE I
DETAILS OF THE NETWORKS WE ARE USING IN OUR EXPERIMENTS. FULL TRAINING DETAILS ARE GIVEN IN THE TEXT.

Network	Dataset	Parameters	Size, MiB	Test error, %
LeNet300	MNSIT	0.27M	1.02	1.79
LeNet5	MNIST	0.40M	1.64	0.57
ResNet20	CIFAR10	0.27M	2.07	8.35
ResNet32	CIFAR10	0.46M	3.56	7.14
VGG16	CIFAR10	15.2M	58.22	6.46

IV. EXPERIMENTS

We perform our comparison on representative networks trained on MNIST and CIFAR10 datasets. We choose both fully-connected (LeNet300) and convolutional networks with few (LeNet5, ResNet20, ResNet32) and many (VGG16) parameters. All experiments are initialized from reasonably well-trained reference networks.

For our comparison experiments we compress all layers of the networks, only excluding biases (few weights per layer) and batch-normalization layers. For quantization, we use separate codebooks per each layer of the same size k. For a single model we try different values of $k = 2, 4, \dots$ and generate a corresponding tradeoff curve. For pruning, we choose the amount of non-zero weights left in the network, the parameter κ in (4), to be a certain percent from the total number of weights. The pruning is applied jointly to all layers. The tradeoff curve is obtained by varying the amount of $\kappa = 1\%, 2\%, \dots$, etc. For low-rank compression we use the number of parameters as our target cost C in (6). The weights of convolutional layers are reshaped using scheme-2. To obtain the tradeoff curve we use range of values for λ. We give all compression parameters for each network in sections IV-B-IV-C.

For a particular model we use same settings of the L-step optimization (epochs, learning rates, etc.) across all compressions. Once compression is finished, we finetune the models wrt compressed parameters using SGD: for quantization this involves optimizing wrt codebook values, for pruning we finetune wrt non-zero weights, and for low-rank compression we finetune wrt decomposed low-rank matrices. The amount and duration of finetuning (epochs, learning rate, etc.) is the same for all compression of a given model.

The compressions we are applying in our study do operate with hard-to-compare measurements. For example, in compression literature, for quantized models usually only the size of the codebook in bits (e.g., 2 bit codebook) is reported. In contrast, for the elementwise pruning a typically reported

quantity is sparsity (i.e., number of non-zero items). Since our focus is in fair and objective comparison, we will be reporting the final size and corresponding compression ratio of the compressed model when saved to the disk. In section IV-A we give full details of this process.

A. Storage of the compressed models

We report the compression ratio of a particular compressed model as the ratio of

$$\text{compression ratio} = \frac{\text{uncompressed size}}{\text{compressed size}} \tag{7}$$

which is computed wrt to the number of bytes required to store the corresponding compressed and uncompressed models. All uncompressed parameters (biases, batch-normalization weights, etc.) are stored along the model as is, without specific handling, and included into calculation of the *compressed size*. Next, we give the details of how we are storing the compressed models assuming, for brevity, that there are no uncompressed parts.

Reference model We assume that reference model is stored using IEEE 32 bit floating point numbers. Thus uncompressed size of the model is $N \times 32$ bits where N is the total number of weights in the model.

Quantized model When we quantize the model, weights are shared across the network. Therefore, instead of storing the weight values themselves, for each weight we store its index in the codebook. If the codebook has K entries then total storage required is $N \times \lceil \log K \rceil + M \times K \times 32$. The latter term accounts for the storage of K-sized codebooks for each of the M layers in the network.

Pruned model When we prune the model, we are left with exactly κ non-zero weights. However, to store such model we need to allocate $\kappa \times 32$ bits for the weights and *additional amount for the sparse index*: the locations of the non-zero weights within the model. Instead of storing the spare index directly, we adopt storing the differences between the subsequent indexes (when counted in a flattened array) as suggested by Han et al. [45]. Each difference is encoded using p-bit integer where p is chosen optimally to minimize the total storage of $\kappa \times 32 + \kappa \times p$.

Low-rank model When we apply low-rank compression to the neural network, we can equivalently parametrize weight matrices as a low-rank product of $\mathbf{U}\mathbf{V}^T$. Thus, the total storage required to low-rank model is the total number of parameters in the low-rank versions of the layers multiplied by 32 bits. Note: occasionally, some layer's selected rank do not correspond to the storage savings (for example when algorithm assigns full rank to this particular layer). In such cases, we save the layer in its original form using a single full-rank matrix.

B. MNIST experiments

On the MNIST dataset [46] of 60K grayscale images of 28×28 we train and subsequently compress LeNet300 and Caffe version[2] of LeNet5. We normalize the pixel grayscale values of training and test images to the range of $[0, 1]$ and then subtract the mean image (computed over the training dataset). Reference networks are trained using the cross-entropy loss with additional ℓ_2 regularization on the weights (10^{-5} for LeNet300 and 10^{-6} for LeNet5). We use stochastic gradient descent (SGD) accelerated by Nesterov's scheme on minibatches of 256 images and momentum of 0.9. Both reference networks are trained for 300 epochs with the initial learning rate of 0.1 decayed by 0.99 after ever epoch. Reference LeNet300 has a test error of 1.67% and LeNet5 has a test error of 0.57% (see Tab. I)

LC algorithm settings We use same L step settings for all experiments on a given model. For LeNet300, the L step is performed using Nesterov's SGD on minibatches of 256 images with momentum of 0.9, learning rate of 0.1×0.98^t at the tth L step. For LeNet5, the L step is performed using Nesterov's SGD on minibatches of 256 images with momentum of 0.9, learning rate of 0.05×0.98^t at the tth L step. For both networks each L step is run for 10 epochs. In total, we have 30 L and C step alternations. We use multiplicative schedule on μ-values of $\mu_t = 10^{-5} \times 1.4^t$ at the tth LC step.

Finetuning We finetune compressed networks for 100 epochs using the learning rate of 0.02×0.99^t at epoch t (all other settings are same as in L step)

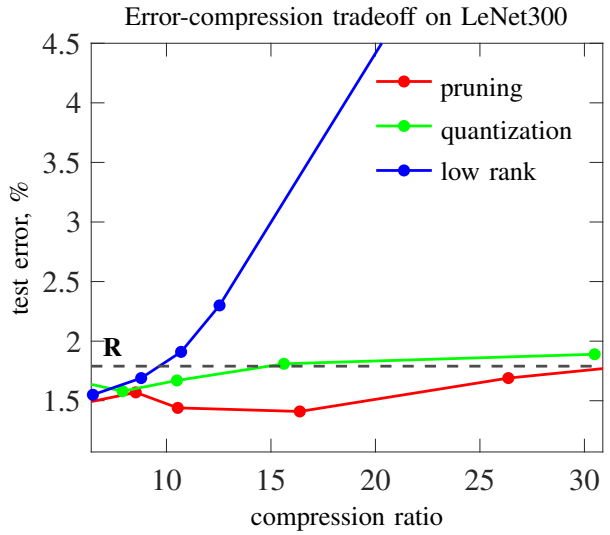
Results We present our results as error-compression tradeoff curves on Fig 4. For both networks, we drive the compression ratio to a considerable values (beyond $30\times$), however, to make plots comparable and easily readable we show the tradeoff curves for the compression ratios of 7–31. For LeNet300, pruning and quantization results are comparable along the entire tradeoff curve, with slightly favorable results towards the pruning. Interestingly, some of the compression points showcase even a better test error indicating the regularization capabilities of compression mechanisms. The performance of the low-rank compression on the LeNet300 quickly deteriorates and becomes drastically worse wrt pruning/quantization. In contrast, on the convolutional LeNet5 all schemes seem to deteriorate in a similar manner when the compression ratio increases. Yet again, the pruning results are marginally better across the tradoff region. We conclude that elementwise pruning on is the best choice among considered compressions for the LeNet300 and LeNet5 networks.

C. CIFAR10 experiments

We train ResNet models with 20 and 32 layers and the adapted[3] version of VGG16 [47] on the CIFAR10 dataset (10 classes, 60K colored images of 32×32). All networks use batch normalization [48] between the layers; we used same data-augmentation across the networks: normalize the images, randomly flip image over horizontal axis, zero pad the

[2]https://github.com/BVLC/caffe/blob/master/examples/mnist/lenet.prototxt

[3]With outputs of fully connected layers of dimensions 512, 512, and 10.



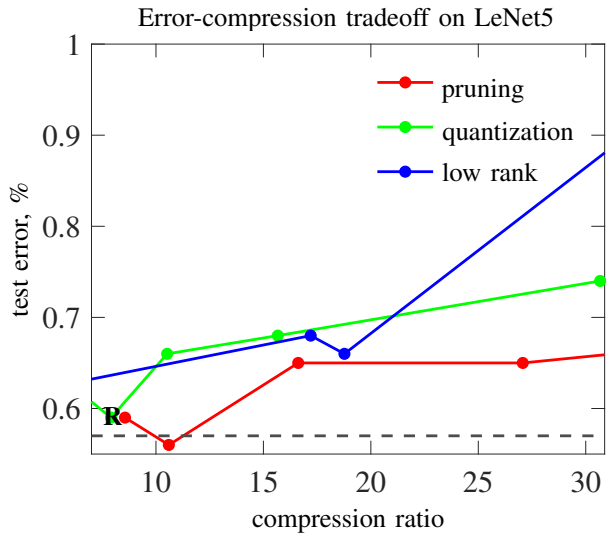
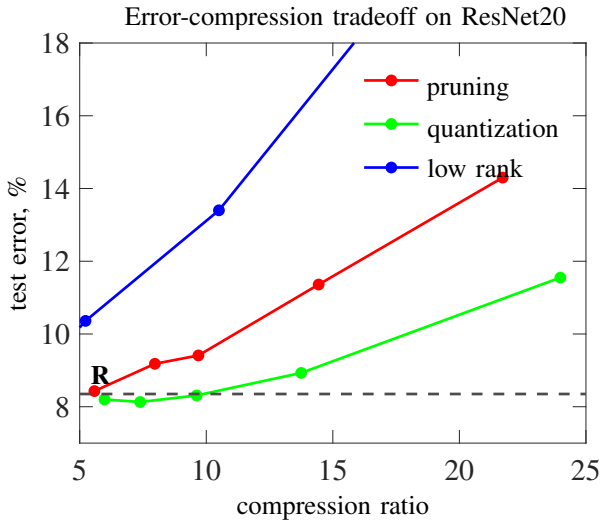
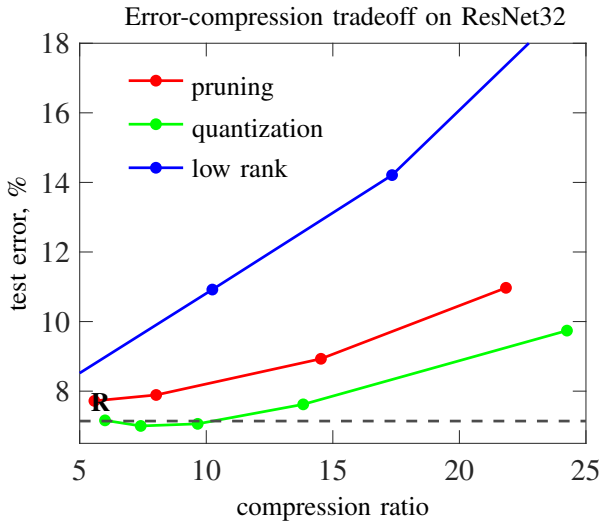


Fig. 4. Exploration of the tradeoff curves for quantization, low-rank, and pruning on the LeNet300 and LeNet5. The reference models have a test error of 1.79% and 0.56% respectively (horizontal dashed line marked by **R**). For these particular models, the pruning based compression is a better choice along the entire tradeoff curve.

image (4 pixels on each side) and randomly crop a 32×32 piece. During the testing phase we use normalized images without augmentation. Reference networks are trained using Nesterov's SGD with momentum of 0.9 on the minibatches of 128 images; the loss is the cross-entropy with additive ℓ_2 regularization of 5×10^{-4}. We train ResNets for a total of 200 epochs with the learning rate of 0.1 which is decreased by 0.1 after 100 and 125 epochs. We train VGG16 for a total of 300 epochs with the initial learning rate of 0.05 which is decayed by 0.977 after every epoch. Reference networks have the following test errors: ResNet20 of 8.35%, ResNet32 of 7.14%, and VGG15 of 6.46% (see Tab. I)

LC algorithm settings We use the same L-step settings for all experiments of a particular model. For ResNet20 and





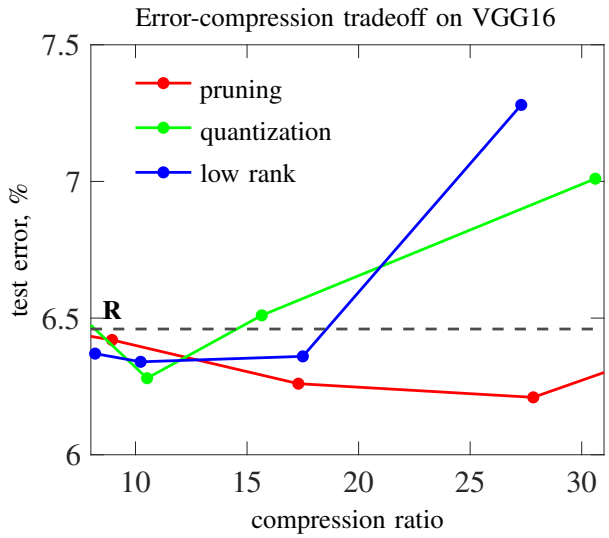


Fig. 5. Exploration of the tradeoff curves for quantization, low-rank, and pruning on the ResNet20, ResNet32, and VGG16. The reference models have a test error of 8.35%, 7.14%, and 6.46% respectively (horizontal dashed line marked by **R**). For ResNets quantization performs the best along the entire tradeoff curve; for VGG16 pruning is more suitable.

ResNet32, the L step is performed using Nesterov's SGD on minibatches of 128 images with a momentum of 0.9; the learning rate of 0.005 is decayed by 0.9 (restarted on every L step). For VGG16 we have the learning rate of 0.0007 with all other parameters unchanged. Each L step for all networks is run for 15 epochs. The ResNet experiments have a total of 50 LC alternations with the $\mu_t = 10^{-4} \times 1.3^t$ at the tth step. For VGG16 we have 60 LC alternations with $\mu_t = 2 \cdot 10^{-5} \times 1.2^t$ at the tth step.

Finetuning We finetune compressed networks for 100 epochs. For ResNets we use the learning rate of 0.0005×0.99^t at tth epoch; and we use the learning rate schedule of 0.00008×0.99^t for VGG16 (all other settings are same as in the L step)

Results In Fig. 5 we present our compression results as tradeoff curves for every combination of model and compression. In contrast to the MNIST experiments, here we see a different picture in terms of the best choice of a compression for the networks. For ResNet models, quantization has a significantly better error-compression tradeoff when compared to pruning and low-rank. We can compress already small ResNets by $10\times$ without any loss of accuracy. We observe that the pruning does not perform as good as we have seen in other experiments. This is not surprising: unlike quantization, a weight is pruned its contribution is completely eliminated. If there are not many weights in the network to start with (here we are compressing very lean ResNets), at some point the removal of the weights becomes irreversible. This interplay, of course, depends on the task, architecture, and the performance of the compressions scheme. For instance, pruning on the other networks, and in particular on large VGG16, outperforms other compression strategies with a significant margin.

D. Discussion

Our observations supports the motivation of the paper: without an actual empirical comparison the question of the *best compression for a model* will be ill-posed. As we have demonstrated, while some compressions (for instance, pruning) showcase favorable results in many cases there is no *one-fits-all* recipe for an arbitrary combination of a task and a model. Yet, having such a rule of thumb about effectiveness of a certain compression for the layers of a certain type would be a great addition. We hope our empirical study would pave the way to collect and combine many future compression schemes and their results in a single, easily comparable benchmark.

V. Conclusion

In this paper, we argued and experimentally validated the necessity of fair and objective empirical comparison of different compression mechanisms available in the literature. The tool that allowed us to perform this comparison is the Learning Compression framework and its open-source implementation—the LC Toolkit. The mathematical concept behind the framework is the separation of model learning (L step) from model compression (C step). We have empirically observed that there is no single compression that universally outperforms all other schemes. Despite this, for some networks and datasets, empirical evidence suggests that pruning is a safe starting choice. To the best of our knowledge, this is the first work comparing diverse compression mechanisms in apples-to-apples way, thus filling very needed gap in the literature.

Acknowledgments We thank NVIDIA Corporation for multiple GPU donations.

References

[1] M. Courbariaux, Y. Bengio, and J.-P. David, "BinaryConnect: Training deep neural networks with binary weights during propagations," in *Advances in Neural Information Processing Systems (NIPS)*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. MIT Press, Cambridge, MA, 2015, pp. 3105–3113.

[2] M. Á. Carreira-Perpiñán and Y. Idelbayev, "Model compression as constrained optimization, with application to neural nets. Part II: Quantization," Jul. 13 2017, arXiv:1707.04319.

[3] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in *Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17)*, Honolulu, HI, Jul. 21–26 2017, pp. 5456–5464.

[4] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: ImageNet classification using binary convolutional neural networks," in *Proc. 14th European Conf. Computer Vision (ECCV'16)*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Amsterdam, The Netherlands, Oct. 11–14 2016, pp. 525–542.

[5] 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," Jul. 17 2016, arXiv:1606.06160.

[6] 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.

[7] K. Hwang and W. Sung, "Fixed-point feedforward deep neural network design using weights $+1$, 0, and -1," in *2014 IEEE Workshop on Signal Processing Systems (SiPS)*, Belfast, UK, Oct. 20–22 2014, pp. 1–6.

[8] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing deep convolutional networks using vector quantization," in *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.

[9] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'15)*, Brisbane, Australia, Apr. 19–24 2015, pp. 1131–1135.

[10] Y. Idelbayev, P. Molchanov, M. Shen, H. Yin, M. Á. Carreira-Perpiñán, and J. M. Alvarez, "Optimal quantization using scaled codebook," in *Proc. of the 2021 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'21)*, Virtual, Jun. 19–25 2021.

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

[12] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems (NIPS)*, S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds., vol. 5. Morgan Kaufmann, San Mateo, CA, 1993, pp. 164–171.

[13] B. Liu, M. Wan, H. Foroosh, M. Tappen, and M. Pensky, "Sparse convolutional neural networks," in *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, Boston, MA, Jun. 7–12 2015, pp. 806–814.

[14] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. MIT Press, Cambridge, MA, 2016, pp. 2074–2082.

[15] H. Yang, W. Wen, and H. Li, "DeepHoyer: Learning sparser neural network with differentiable scale-invariant sparsity measures," in *Proc. of the 8th Int. Conf. Learning Representations (ICLR 2020)*, Addis Ababa, Ethiopia, Apr. 26–30 2020.

[16] Y. Li, S. Gu, C. Mayer, L. V. Gool, and R. Timofte, "Group sparsity: The hinge between filter pruning and decomposition for network compression," in *Proc. of the 2020 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'20)*, Seattle, WA, Jun. 14–19 2020, pp. 8018–8027.

[17] T. N. Sainath, B. Kingsbury, V. Sindhwani, E. Arısoy, and B. Ramabhadran, "Low-rank matrix factorization for deep neural network training with high-dimensional output targets," in *Proc. of the IEEE Int. Conf. Acoustics, Speech and Sig. Proc. (ICASSP'13)*, Vancouver, Canada, Mar. 26–30 2013, pp. 6655–6659.

[18] J. Xue, J. Li, and Y. Gong, "Restructuring of deep neural network acoustic models with singular value decomposition," in *Proc. of Interspeech'13*, F. Bimbot, C. Cerisara, C. Fougeron, G. Gravier, L. Lamel, F. Pellegrino, and P. Perrier, Eds., Lyon, France, Aug. 25–29 2013, pp. 2365–2369.

[19] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems (NIPS)*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., vol. 26. MIT Press, Cambridge, MA, 2013, pp. 2148–2156.

[20] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in *Advances in Neural Information Processing Systems (NIPS)*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds., vol. 27. MIT Press, Cambridge, MA, 2014, pp. 1269–1277.

[21] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *Proc. of the 25th British Machine Vision Conference (BMVC 2014)*, M. Valstar, A. French, and T. Pridmore, Eds., Nottingham, UK, Sep. 1–5 2014.

[22] X. Zhang, J. Zou, K. He, and J. Sun, "Accelerating very deep convolutional networks for classification and detection," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 1943–1955, Oct. 2016.

[23] C. Tai, T. Xiao, Y. Zhang, X. Wang, and W. E, "Convolutional neural networks with low-rank regularization," in *Proc. of the 4th Int. Conf. Learning Representations (ICLR 2016)*, San Juan, Puerto Rico, May 2–4 2016.

[24] 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.

[25] C. Li and C. J. R. Shi, "Constrained optimization based low-rank approximation of deep neural networks," in *Proc. 15th European Conf. Computer Vision (ECCV'18)*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Munich, Germany, Sep. 8–14 2018, pp. 746–761.

[26] X. Xu, Y. Ding, S. X. Hu, M. Niemier, J. Cong, Y. Hu, and Y. Shi, "Scaling for edge inference of deep neural networks," *Nature Electronics*, vol. 1, pp. 216–222, Apr. 17 2018.

[27] V. Lebedev, Y. Ganin, M. Rakhuba, I. Oseledets, and V. Lempitsky, "Speeding-up convolutional neural networks using fine-tuned CP-decomposition," in *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*, San Diego, CA, May 7–9 2015.

[28] A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov, "Tensorizing neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. MIT Press, Cambridge, MA, 2015, pp. 442–450.

[29] T. Garipov, D. Podoprikhin, A. Novikov, and D. Vetrov, "Ultimate tensorization: Compressing convolutional and FC layers alike," Nov. 10 2016, arXiv:1611.03214.

[30] 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)*, Seattle, WA, Jun. 14–19 2020, pp. 8046–8056.

[31] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems (NIPS)*, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28. MIT Press, Cambridge, MA, 2015, pp. 1135–1143.

[32] S. Gui, H. N. Wang, H. Yang, C. Yu, Z. Wang, and J. Liu, "Model compression with adversarial robustness: A unified optimization framework," in *Advances in Neural Information Processing Systems (NEURIPS)*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds., vol. 32. MIT Press, Cambridge, MA, 2019, pp. 1285–1296.

[33] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proc. of the 2018 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'18)*, Salt Lake City, UT, Jun. 18–22 2018, pp. 2704–2713.

[34] J. M. Alvarez and M. Salzmann, "Compression-aware training of deep networks," in *Advances in Neural Information Processing Systems (NIPS)*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. MIT Press, Cambridge, MA, 2017, pp. 856–867.

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

[36] 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)*, Salt Lake City, UT, Jun. 18–22 2018, pp. 8532–8541.

[37] M. Á. Carreira-Perpiñán and A. Zharmagambetov, "Fast model compression," in *Bay Area Machine Learning Symposium (BayLearn 2018)*, Facebook, Menlo Park, CA, 2018.

[38] Y. Idelbayev and M. Á. Carreira-Perpiñán, "More general and effective model compression via an additive combination of compressions," 2020, submitted.

[39] ——, "Neural network compression via additive combination of reshaped, low-rank matrices," in *Proc. Data Compression Conference (DCC 2021)*, Mar. 23–26 2021, pp. 243–252.

[40] ——, "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)*, Toronto, Canada, Jun. 6–11 2021.

[41] ——, "Beyond FLOPs in low-rank compression of neural networks: Optimizing device-specific inference runtime," 2021, submitted.

[42] J. Nocedal and S. J. Wright, *Numerical Optimization*, 2nd ed., ser. Springer Series in Operations Research and Financial Engineering. New York: Springer-Verlag, 2006.

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

[44] 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)*, Long Beach, CA, Jun. 16–20 2019, pp. 12 569–12 577.

[45] 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.

[46] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[47] 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.

[48] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. of the 32nd Int. Conf. Machine Learning (ICML 2015)*, F. Bach and D. Blei, Eds., Lille, France, Jul. 6–11 2015, pp. 448–456.