Theme Article: Machine Learning for Systems

A Programmable Approach to Neural Network Compression

Vinu Joseph and Ganesh L. Gopalakrishnan
University of Utah

Animesh Garg
University of Toronto and Vector Institute

Saurav Muralidharan and Michael Garland
NVIDIA

***Abstract*—Deep neural networks (DNNs) frequently contain far more weights, represented at a higher precision, than are required for the specific task, which they are trained to perform. Consequently, they can often be compressed using techniques such as weight pruning and quantization that reduce both the model size and inference time without appreciable loss in accuracy. However, finding the best compression strategy and corresponding target sparsity for a given DNN, hardware platform, and optimization objective currently requires expensive, frequently manual, trial-and-error experimentation. In this article, we introduce a programmable system for model compression called CONDENSA. Users programmatically compose simple operators, in Python, to build more complex and practically interesting compression strategies. Given a strategy and user-provided objective (such as minimization of running time), CONDENSA uses a novel Bayesian optimization-based algorithm to automatically infer desirable sparsities. Our experiments on four real-world DNNs demonstrate memory footprint and hardware runtime throughput improvements of 188× and 2.59×, respectively, using at most ten samples per search.**

Modern deep neural networks (DNNs) are complex and often contain millions of parameters spanning dozens or even hundreds of layers.[1,2] This complexity translates into substantial memory and runtime costs on hardware platforms at all scales. Recent work has demonstrated that DNNs are often overprovisioned and can be compressed without appreciable loss of accuracy. Model compression can be used to reduce both

Digital Object Identifier 10.1109/MM.2020.3012391
Date of publication 28 July 2020; date of current version 1 September 2020.

model memory footprint and inference latency using techniques such as weight pruning, quantization, and low-rank factorization.[3–5] Unfortunately, the requirements of different *compression contexts*—DNN structure, target hardware platform, and the user's optimization objective—are often in conflict. The recommended compression strategy for reducing inference latency may be different from that required to reduce total memory footprint. For example, in a convolutional neural network (CNN), reducing inference latency may require pruning filters to realize speedups on real hardware,[4] while reducing memory footprint may be accomplished by zeroing out individual weights. Similarly, even for the *same optimization objective*, say reducing inference latency, one may employ filter pruning for a CNN, while pruning 2-D blocks of nonzero weights for a language translation network such as Transformer, since the latter has no convolutional layers. Thus, it is crucial to enable convenient expression of alternative compression schemes; however, none of today's model compression approaches help the designer tailor compression schemes to their needs.

Current approaches to model compression also require manual specification of compression hyperparameters, such as target sparsity—the proportion of zero-valued parameters in the compressed model versus the original. However, with current approaches, finding the best sparsity often becomes a trial-and-error search, with each such trial having a huge cost (often multiple days for large models such as BERT), as each such trial involves training the compressed model to convergence, only to find (in most cases) that the compression objectives are not met. The main difficulty faced by such unguided approaches is that sparsities vary unpredictably with changes in the compression context, making it very difficult to provide users with any guidelines, whatsoever. Therefore, automatic and *sample-efficient* approaches that minimize the number of trials are crucial to support the needs of designers who must adapt a variety of neural networks to a broad spectrum of platforms targeting a wide range of tasks.

To address the above-mentioned problems of flexible expression of compression strategies, automated compression hyperparameter inference, and sample efficiency, we make the following contributions.

1) We present CONDENSA, a new framework for programmable neural network compression. CONDENSA supports the expression of the overall compression strategy in Python using operators provided by its compression library. Since each strategy is a Python function, users are able to programmatically compose elementary schemes to build much more complex and practically interesting schemes.
2) We present a novel sample-efficient algorithm based on Bayesian optimization (B.O.) in CONDENSA for automatically inferring optimal sparsities based on a user-provided objective function. Given CONDENSA's ability to support the expression of meaningful high-level objective functions—for example, the throughput (images per second) of a CNN—users are freed from the burden of having to specify compression hyperparameters manually.
3) We demonstrate the effectiveness of CONDENSA on three image classification and language modeling tasks, resulting in memory footprint reductions of up to $188\times$ and runtime throughput improvements of up to $2.59\times$ using at most ten samples per search.

BACKGROUND

For a given task such as image classification, assume we have trained a large *reference* model $\overline{\mathbf{w}} = \text{argmin}_{\mathbf{w}} L(\mathbf{w})$, where $L()$ denotes a *loss function* (e.g., cross-entropy on a given training set), and $\mathbf{w} \in \mathbb{R}^P$. *Model compression* refers to finding a smaller model Θ that can be applied to the same task and ideally achieves the same accuracy as $\overline{\mathbf{w}}$. Model compression can be performed in various ways, and CONDENSA currently supports two commonly used techniques: pruning and quantization. In pruning, nonzero values from $\overline{\mathbf{w}}$ are eliminated or "pruned" to obtain Θ. Pruning is usually performed using some kind of thresholding (for e.g., magnitude-based) and can be unstructured (prune any nonzero value) or structured (prune only *blocks* of nonzeros).[4,3] On the other hand, quantization retains the number of parameters in Θ but assigns parameters in

$\overline{w}$ one of K codebook values, where the codebook may be fixed or adaptive. CONDENSA supports low-precision approximation, which refers to assigning each parameter in $\overline{w}$ a corresponding lower-precision representation (for example, converting from 32-bit to 16-bit floating-point).

Automating model compression involves finding both an optimal compression strategy for a given $\overline{w}$, along with its corresponding compression hyperparameters such as target sparsity with minimal manual intervention. Current state-of-the-art frameworks in this domain include AMC[3] and AutoCompress,[5] which use reinforcement learning and simulated annealing, respectively, to automatically find desirable target sparsities for a fixed pruning strategy. CONDENSA, in contrast, supports the programmable expression of a wide variety of compression strategies (not just pruning). Also, in the context of automated model compression, each sample corresponds to training the compressed model to convergence, and can be extremely expensive to compute; unfortunately, techniques such as reinforcement learning, which is used in AMC,[3] can be highly sample-inefficient.[6] To minimize the number of samples drawn, CONDENSA uses a novel and sample-efficient B.O.-based algorithm for automatically arriving at desirable target sparsities.

CONDENSA FRAMEWORK

In CONDENSA, users specify *compression schemes* to systematically describe how a given reference model $\overline{w}$ is transformed into a compressed version. Schemes are expressed as Python functions and can utilize a rich set of compression operators provided by the CONDENSA library. Integrating with the Python ecosystem makes the expression of common compression patterns more natural. For example, operators can be combined with conditional statements to selectively compress layers based on properties of the input DNN and/or target hardware platform. The CONDENSA Library provides operators and prebuilt schemes for unstructured and structured (filter and block) pruning, and quantization. Listing 1 provides a concrete example of invoking CONDENSA to compress a model.

Listing 1. Example usage of the CONDENSA library.

```
# Construct pre-trained model
criterion = torch.nn.CrossEntropyLoss()
train(model, num_epochs, trainloader, criterion)

# Instantiate compression scheme
prune = condensa.schemes.FilterPrune()
# Define objective function
tput = condensa.objectives.throughput
# Specify optimization operator
obj = condensa.searchops.Maximize(tput)
# Instantiate L-C optimizer
lc = condensa.optimizers.LC(steps=30, lr=0.01)
# Build model compressor instance
compressor = condensa.Compressor(
    model=model, # Trained model
    objective=obj, # Objective
    eps=0.02, # Accuracy threshold
    optimizer=lc, # Accuracy recovery
    scheme=prune, # Compression scheme
    trainloader=trainloader, # Train dataloader
    testloader=testloader, # Test dataloader
    valloader=valloader, # Val dataloader
    criterion=criterion # Loss criterion
)
# Obtain compressed model
wc = compressor.run()
```

CONDENSA System Design

Figure 1 provides a high-level overview of the CONDENSA framework. As shown on the left-hand side of the figure, a user compresses a pretrained model $\overline{w}$ by specifying a compression scheme and an objective function f. Both the compression scheme and objective are specified in Python using operators from the CONDENSA library; alternatively, users may choose from a selection of commonly used built-in schemes and objectives. Apart from the operator library, the core framework, shown in the middle of the figure, consists primarily of two components: 1) the constrained Bayesian optimizer for inferring optimal target sparsities; and 2) the "learning-compression" (L-C) optimizer[7] for accuracy recovery. The remainder of this section describes both the Bayesian and L-C optimizers in more detail.

Sample-Efficient Bayesian Optimization

It is intuitive to split the problem of finding optimal target sparsities into two stages: 1) find the highest target sparsity that loses at most ϵ accuracy w.r.t the original uncompressed model $\overline{w}$; and 2) in a constrained sparsity regime obtained from stage 1), optimize a user-provided objective function f (e.g., throughput, or

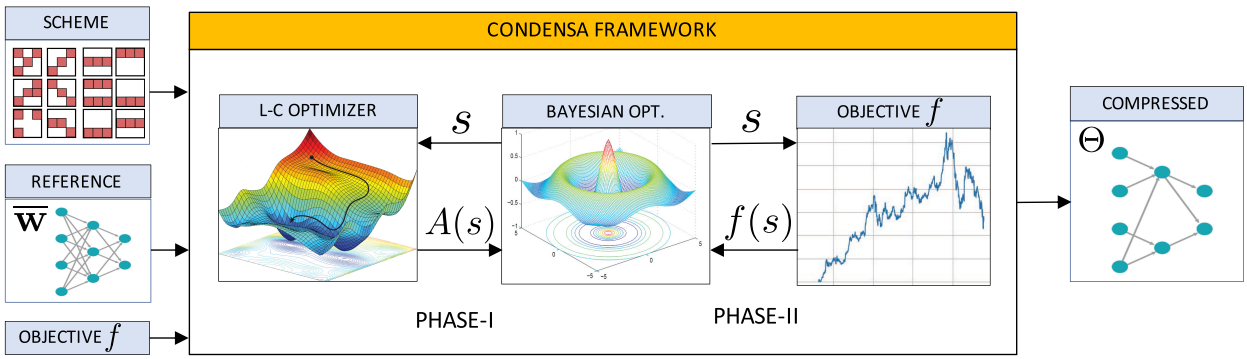

Figure 1. CONDENSA framework overview. The user provides the pretrained model ($\overline{\mathbf{w}}$), a compression scheme, and an objective function f. CONDENSA uses the Bayesian and L-C optimizers to infer an optimal target sparsity s^* and corresponding compressed model Θ.

memory footprint) and return the solution as the final sparsity. For both stages, CONDENSA utilizes B.O., as shown in Figure 1.

B.O. is an optimization framework based on continually updating a *probabilistic model* with measurements of a function to be optimized.[8] Given a set of parameters to be optimized, B.O. makes black-box calls to the objective, updates the probabilistic model with the new information, and selects the next point to evaluate using an *acquisition function* that combines information about the expectation and uncertainty of a function value under the probabilistic model. CONDENSA employs a Gaussian Process (G.P.) model for B.O. due to its favorable statistical and computational characteristics.[9] It is worth highlighting that B.O. leverages principled Bayesian inference to tradeoff exploration and exploitation and is sample-efficient for nonconvex black-box functions such as the ones optimized by CONDENSA.[8]

In CONDENSA's two-stage optimization pipeline, we first find a sparsity ratio s_{acc} that constrains the model accuracy function A to the provided ϵ. We then constrain the *sparsity search space* to $(0, s_{\text{acc}})$ while optimizing the user-provided objective function f. Note that we assume that A decreases monotonically w.r.t. sparsity in the region $(0, s_{\text{acc}})$. For each stage, CONDENSA uses a distinct acquisition function to guide the next best point for function evaluation.

Stage 1: Solving Accuracy Constraints: Recall that in the first stage of the sparsity inference process, we aim to find the highest sparsity s_{acc} that loses at most ϵ accuracy w.r.t. the original reference model $\overline{\mathbf{w}}$. To this end, we first define a *Level-Set* L that represents $\text{Acc}(\overline{\mathbf{w}}) - \epsilon$ and aim to find the point on the accuracy curve of the compressed model that intersects with L; the sparsity corresponding to this solution will be s_{acc}. We propose a novel acquisition function to find s_{acc} named domain-restricted upper confidence bound (DR-UCB).

DR-UCB builds upon an existing level-set black-box optimization technique named ILS-UCB,[10] which is characterized by two properties. 1) It prioritizes searching in the region where the level set intersects the accuracy curve. 2) It does not seek to precisely learn the shape of the entire accuracy curve. However, in CONDENSA, since accuracy values can be safely assumed to decrease monotonically with increasing sparsity, we notice that it is also possible to progressively restrict the search domain of sparsities based on whether the currently sampled point meets the level-set constraints. In DR-UCB, we exploit this property to greatly improve sample efficiency over ILS-UCB. Mathematically, we define $\mathbf{s}_t$, the sparsity value sampled at iteration t using DR-UCB, as follows:

$$\begin{aligned} \mathbf{s}_t &= \underset{\mathbf{s}}{\text{argmax}}\ (1-\gamma)\sigma(\mathbf{s}) - \gamma|\mu(\mathbf{s}) - L| \\ \text{s.t.}\quad & \mathbf{s}_t > \mathbf{s}_i \quad \forall i \in [0, t-1], \quad \mathcal{B}_f(\mathbf{s}_t) \geq L. \end{aligned} \tag{1}$$

Here, $\mathcal{B}_f$ represents the L-C accuracy function, and $\mathbf{s}_t$ is 1) greater than all the previous sparsities $\mathbf{s}_i$; and 2) satisfies the level set constraint $\mathcal{B}_f(\mathbf{s}_t) \geq L$. We achieve this by minimizing the difference between the GP's mean curve $\mu(\mathbf{s})$ and the level set using the term $|\mu(\mathbf{s}) - L|$ in (1); the parameter γ controls the tradeoff between exploitation and exploration. Algorithm 1 illustrates how DR-UCB is employed to efficiently find s_{acc}.

Algorithm 1. Bayesian Sparsity Inference with Domain Restriction

procedure $\text{BO}_{\text{DR-UCB}}$ $(\mathcal{B}_f, L, T)$
  ▷ $\mathcal{B}_f$: Function to optimize
  ▷ L: Level set
  ▷ T: # Iterations
  GP ← GP-Regressor.initialize()
  $s_0 \leftarrow 0$; $D \leftarrow (0, 1)$; $\mathbf{X} \leftarrow \emptyset$
  for $t \leftarrow 1, 2, \ldots . T-1$ **do**
    $s_t \leftarrow \texttt{argmax}_D \text{DR-UCB}(s|\mathbf{X}_{0:t-1})$
    $y_t \leftarrow \mathcal{B}_f(s_t)$
    if $s_t > s_{t-1}$ **and** $y_t \geq L$ **then**
      $D \leftarrow (s_t, 1)$
    end if
    $\mathbf{X}_{0:t} \leftarrow \{\mathbf{X}_{0:t-1}, (s_t, y_t)\}$
    GP.Update($\mathbf{X}_{0:t}$)
  end for
  return s_{T-1}
end procedure

Stage 2: Optimizing the User-Defined Objective: Once we find a sparsity s_{acc} that satisfies the user-provided accuracy constraints in stage 1, our next objective is to find the final sparsity s^* that optimizes the user-defined objective function f in the constrained sparsity domain $(0, s_{\text{acc}})$. For this, we employ the upper and lower confidence bound (UCB/LCB) acquisition functions for function maximization and minimization, respectively.[9]

Accuracy Recovery Using L-C

As described earlier in this section, given a reference model $\overline{\mathbf{w}}$, compression scheme, and target sparsity (obtained automatically by the Bayesian optimizer), CONDENSA tries to recover any accuracy lost due to compression. In this article, we use the recently proposed L-C algorithm[7] for accuracy recovery, which formulates model compression as a constrained optimization problem. L-C naturally supports all the compression operators supported by CONDENSA while providing optimality guarantees whenever possible. Due to space restrictions, we refer the reader to the article by Carreira-Perpinán and Idelbayev[7] for a more detailed description of the L-C algorithm.

EVALUATION

We conduct extensive experiments and fully analyze CONDENSA on three tasks: 1) image classification on the CIFAR-10 data set[11] using the VGG-19[2] and ResNet56[1] neural networks; 2) image classification on the ILSVRC (ImageNet) task[12] using the VGG-16 neural network[2]; and 3) language modeling on WikiText-2[13] using a two-layer LSTM network described by Yu *et al.*[14] We optimize the networks in each task for two distinct objectives described below.

Objective 1: Minimize Memory Footprint: The memory footprint of a model is defined as the number of bytes consumed by the model's *nonzero* parameters. Reducing the footprint below a threshold value is desirable, especially for memory-constrained devices such as mobile phones, and can be accomplished through either pruning or quantization, or both. For reducing footprint, we define a compression scheme that performs unstructured pruning of each learnable layer (except batch normalization layers), and then quantizes it to half-precision floating-point, yielding an additional $2\times$ reduction. We denote this scheme by P+Q and implement it using the CONDENSA library as follows:

```
from schemes import Compose, Prune, Quantize
scheme = Compose([Prune(), Quantize(float16)])
```

Objective 2: Maximize Throughput: Inference throughput is defined as the number of input samples processed by a model per second and is commonly used for measuring real-world performance. For CIFAR-10 and ImageNet, we measure hardware inference throughput of the compressed model in the objective function. We use an NVIDIA Titan V GPU with the TensorRT 5 framework to obtain throughput data. For WikiText-2, due to the lack of optimized block-sparse kernels for PyTorch, we measure the floating-point operations (FLOPs) of the compressed model instead as a proxy for inference performance. To improve throughput, we focus on removing entire blocks of nonzeros, such as convolutional filters, since they have been shown to improve performance on real-world hardware.[4] For CIFAR-10 and ImageNet, we use filter pruning, since all the networks we consider are CNNs. In WikiText-2, we employ block pruning with a block size of 5.

Bayesian Optimizer Settings: We use a Gaussian Processes prior with the Matern kernel ($\nu = 2.5$), length scale of 1.0, and α value of 0.1 with normalization of the predictions. For the DR-UCB acquisition function, we use a γ value of 0.95 for all our

Table 1. CONDENSA performance results on CIFAR-10, ImageNet, and WikiText-2.

Method	Dataset	Network	s^*	Accuracy	r_c	Throughput
Baseline	CIFAR-10	VGG19-BN		92.98%	1×	1×
CONDENSA P+Q	CIFAR-10	VGG19-BN	0.99	93.26%	188.23×	N/A
CONDENSA Filter	CIFAR-10	VGG19-BN	0.79	93.34%	1.35×	2.59×
Baseline	CIFAR-10	ResNet56		92.75%	1×	1×
AMC[3]	CIFAR-10	ResNet56	N/A	90.1%	N/A	$s_F = 2\times$
CONDENSA P+Q	CIFAR-10	ResNet56	0.95	91.42%	31.14×	N/A
CONDENSA Filter	CIFAR-10	ResNet56	0.63	93.18%	1.14×	1.17×
Baseline	ImageNet	VGG16-BN		91.50%	1×	1×
Filter Pruning[4]	ImageNet	VGG16-BN		89.80%	≈ 4×	N/A
AutoCompress[5]	ImageNet	VGG16-BN	N/A	90.90%	6.4×	N/A
AMC[3]	ImageNet	VGG16-BN	N/A	90.1%	N/A	$s_F = 1.25\times$
CONDENSA P+Q	ImageNet	VGG16-BN	0.93	89.89%	29.29	N/A
CONDENSA Filter	ImageNet	VGG16-BN	0.12	90.25%	1×	1.16×
Baseline	WikiText-2	LSTM		Log-Perplexity: 4.70	1×	1×
Lottery Ticket[14]	WikiText-2	LSTM	N/A	Log-Perplexity: 4.70	≈ 10×	N/A
CONDENSA P+Q	WikiText-2	LSTM	0.92	Log-Perplexity: 4.75	4.2×	N/A
CONDENSA Block	WikiText-2	LSTM	0.60	Log-Perplexity: 4.62	1.1×	$s_F = 2.14$

Here, s^* represents the target sparsity obtained by CONDENSA, r_c is the memory footprint reduction, and s_F the FLOP reduction. The level-set, represented by ϵ, is set to 2% below baseline in all experiments.

experiments with a bias toward sampling more in the area of level set, with the intention that the Bayesian optimizer results in a favorable sparsity level in as few samples as possible. We implemented DR-UCB using the fmfn/BO package.*

Results

We present the memory footprint reductions and inference throughput improvements obtained by CONDENSA for each of the three tasks we evaluate in Table 1. For each task, we list the target sparsity obtained by the CONDENSA Bayesian optimizer (s^* in the table), its corresponding accuracy/perplexity (top-1 accuracy, top-5 accuracy, and log perplexity for CIFAR-10, ImageNet, and WikiText-2, respectively), memory footprint reductions using pruning and quantization (column labeled r_c), and inference throughput/FLOP improvements using filter/block pruning.

*https://github.com/fmfn/BayesianOptimization

We also compare our approach with recent work on automated model compression. For CIFAR-10 and ImageNet, we compare our results with AMC[3] and AutoCompress,[5] and for WikiText-2, we compare with Yu *et al.*[14] Since AMC[3] and Yu *et al.*[14] do not report actual runtime numbers on hardware, we report the corresponding FLOP improvements instead (values marked s_F). We also use FLOP reduction as a metric for LSTM block pruning, as described above.

Using the P+Q scheme designed to minimize memory footprint, CONDENSA is able to obtain compression ratios up to 188 ×, which surpasses those of frameworks such as AutoCompress. While AMC and AutoCompress only report theoretical FLOP improvements on CIFAR-10 and ImageNet, the filter pruning strategy implemented using CONDENSA yields real-world runtime improvements of up to 2.59× on an NVIDIA Titan V GPU. Since AMC and AutoCompress do not report the number of samples evaluated to arrive at their

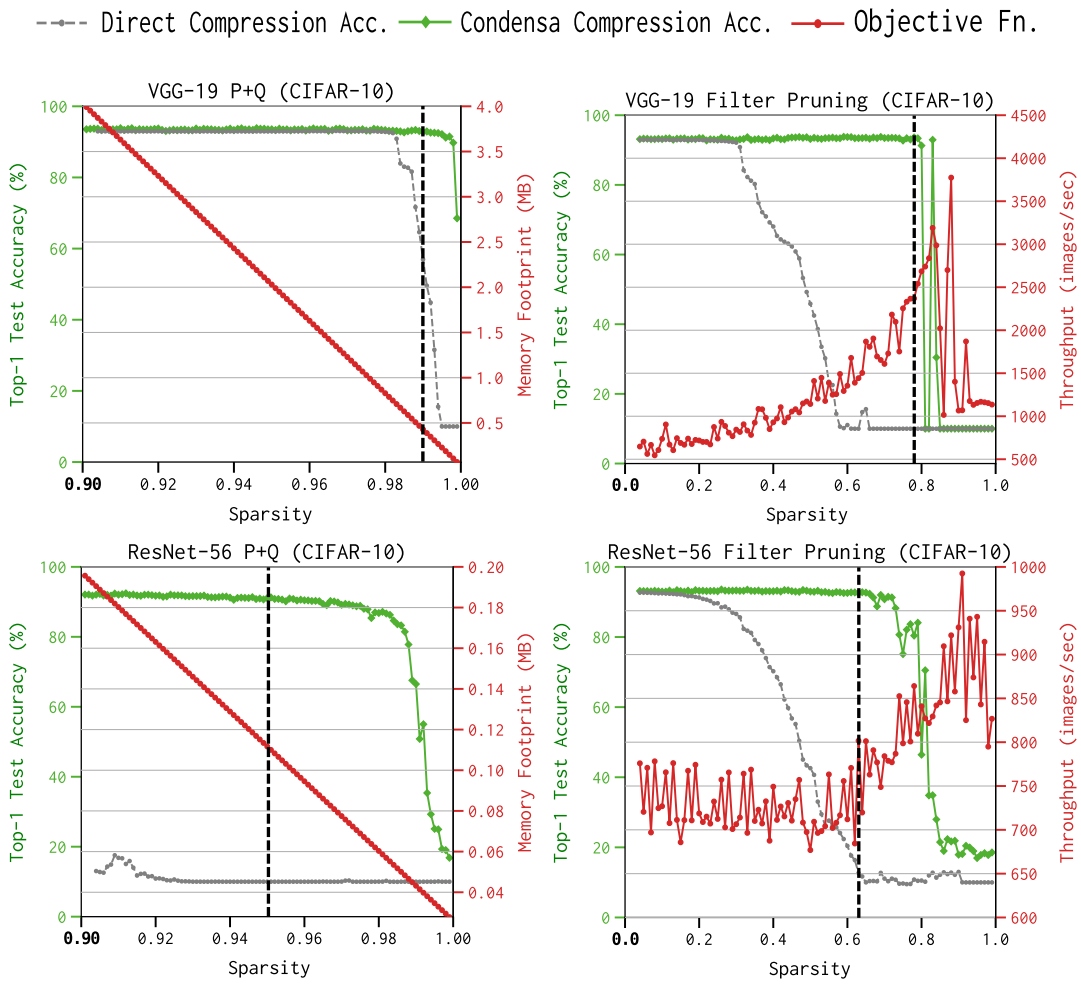

Figure 2. CONDENSA sparsity profiles for VGG19-BN and ResNet56 for CIFAR-10. Column 1 shows the problem of the form "minimize *memory footprint* with a lower bound on accuracy," while Column 2 illustrates "maximize *throughput* with a lower bound on accuracy." The dc line (gray) shows accuracy values if no accuracy recovery with L-C is performed. Note that the *x*-axis ranges are different: the plots on the left have sparsities ranging from 0.9 to 1.0 while those on the right have values ranging from 0 to 1.

solutions, we are unable to directly compare sample efficiencies with these frameworks; however, we notice that CONDENSA obtains desirable model sparsities using a fixed ten iterations per search in all experiments. Finally, while we set the level set to be 2% below the accuracy of the reference model in all our experiments, we notice that CONDENSA-compressed models often exceed baseline accuracy.

Sparsity Profile Analysis

Figure 2 illustrates how a compressed model's accuracy, inference performance, and memory footprint varies w.r.t. sparsity for the CIFAR-10 task. All three of these functions *are assumed to be unknown* in our problem formulation, but we compute them explicitly here to better understand the quality of solutions produced by CONDENSA. For each figure, compression accuracies (shown in green) are obtained by running the L-C algorithm to convergence for 100 sparsities ranging from 0.9 to 1.0 (for pruning + quantization), and from 0 to 1 for the filter and block pruning schemes; collecting each such point requires between 30 min and 8 h of time on a single NVIDIA Tesla V100 GPU. Inference throughput, FLOPs, and memory footprint data are collected for each compressed model and depicted by red lines in the figures (right-hand-side *y*-axis). We also show direct compression (D-C) accuracies in gray for comparison;

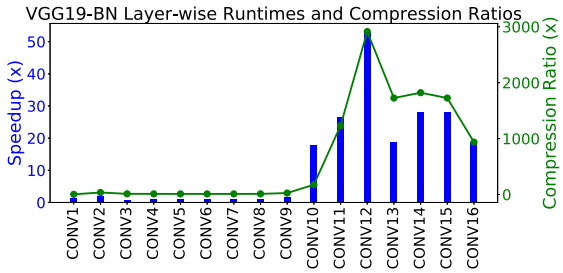

Figure 3. TensorRT runtimes and compression ratios of convolutional layers in VGG19-BN (filter pruning).

direct compression refers to applying a compression scheme to a model without any subsequent attempt at accuracy recovery. In each figure, the sparsity found by CONDENSA is shown as a black vertical dashed line.

We notice three important trends in Figure 2.

1) CONDENSA consistently finds solutions near the "knee" of the L-C accuracy curves, signifying the effectiveness of the DR-UCB acquisition function.
2) Local minima/maxima is avoided while optimizing the objective function, demonstrating that the UCB acquisition function for objective function optimization is working as expected.
3) The knees of the D-C accuracy curves occur at significantly lower sparsities; the L-C optimizer, on the other hand is able to recover accuracies up to much higher target sparsities.

Layerwise Compression Analysis

In this section, we analyze how improving throughput using compression translates to execution time improvements for each layer on actual hardware. For this experiment, we focus on VGG-19 on CIFAR-10, since it has a relatively simple structure and is easy to analyze on a layer-by-layer basis. We use filter pruning with a target sparsity of 0.79 (found by the Bayesian optimizer, as decribed in Table 1) for this experiment. Figure 3 shows layer-by-layer mean runtimes collected over 100 runs using TensorRT (blue bars, left *y*-axis), and compression ratios (green line, right *y*-axis) for filter pruning. We only show data for convolutional layers as they dominate computation time for this network. We make two key observations: 1) runtime speedups on real hardware are largely correlated with compression ratios, but may be affected by hardware and implementation details (e.g., compare conv13 with conv14 in the Figure); and 2) higher compression ratios and corresponding speedups for the later layers of the network, which indicates that distributing a given global sparsity evenly across network layers may not always be optimal, and algorithms such as L-C are essential to automatically finding desirable distributions of sparsity across layers.

CONCLUSIONS

This article has presented CONDENSA, which is a flexible programming system for DNN compression and corresponding hyperparameter optimization. We have demonstrated CONDENSA's effectiveness and ease-of-use on a range of state-of-the-art DNNs for image classification and language modeling, and achieved memory footprint reductions of up to $188\times$ and runtime throughput improvements of up to $2.59\times$ using at most ten samples per search.

This article has presented CONDENSA, which is a flexible programming system for DNN compression and corresponding hyperparameter optimization.

ACKNOWLEDGMENTS

This work was supported in part by DARPA under Contract HR0011-18-3-0007; in part by National Science Foundation (NSF) under Award CCF-1704715; and in part by a CIFAR AI Chair award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

REFERENCES

1. K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.

2. K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Representations*, 2015.
3. Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. 10th Eur. Conf. Comput. Vis.*, 2018, pp. 784–800.
4. Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, vol. 2, pp. 1398–1406.
5. N. Liu, X. Ma, Z. Xu, Y. Wang, J. Tang, and J. Ye, "AutoCompress: An automatic DNN structured pruning framework for ultra-high compression rates," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 4876–4883.
6. V. Mnih *et al.*, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
7. M. A. Carreira-Perpinán and Y. Idelbayev, ""Learning-compression" algorithms for neural net pruning," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8532–8541.
8. D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Global Optim.*, vol. 13, no. 4, pp. 455–492, 1998.
9. N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," in *Proc. 27th Int. Conf. Mach. Learn.*, 2010.
10. A. Garg *et al.*, "Tumor localization using automated palpation with Gaussian process adaptive sampling," in *Proc. IEEE Int. Conf. Autom. Sci. Eng.*, 2016, pp. 194–200.
11. A. Krizhevsky, V. Nair, and G. Hinton, "The CIFAR-10 dataset," [Online]. Avaliable: http://www. cs. toronto. edu/kriz/cifar. html, vol. 55, 2014.
12. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2009, pp. 248–255.
13. S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models," 2016, *arXiv:1609.07843*.
14. H. Yu, S. Edunov, Y. Tian, and A. S. Morcos, "Playing the lottery with rewards and multiple languages: Lottery tickets in RL and NLP," 2019, *arXiv:1906.02768*.

Vinu Joseph is currently working toward the Ph.D. degree in computer science at the University of Utah. Contact him at vinu@cs.utah.edu.

Ganesh L. Gopalakrishnan is a Professor of computer science with the School of Computing, University of Utah. Contact him at ganesh@cs.utah.edu.

Saurav Muralidharan is a Senior Research Scientist in the Programming Systems & Applications research group with NVIDIA. Contact him at sauravm@nvidia.com.

Michael Garland is the Senior Director for Programming Systems & Applications research with NVIDIA. Contact him at mgarland@nvidia.com.

Animesh Garg is an Assistant Professor of computer science with the University of Toronto, and a Faculty Member with the Vector Institute, Canada. Contact him at garg@cs.toronto.edu.