Constraint-Aware Deep Neural Network Compression

Changan Chen, Frederick Tung, Naveen Vedula, and Greg Mori

School of Computing Science,
Simon Fraser University
{cca278,ftung,nvedula}@sfu.ca, mori@cs.sfu.ca

Abstract. Deep neural network compression has the potential to bring modern resource-hungry deep networks to resource-limited devices. However, in many of the most compelling deployment scenarios of compressed deep networks, the operational constraints matter: for example, a pedestrian detection network on a self-driving car may have to satisfy a latency constraint for safe operation. We propose the first principled treatment of deep network compression under operational constraints. We formulate the compression learning problem from the perspective of constrained Bayesian optimization, and introduce a cooling (annealing) strategy to guide the network compression towards the target constraints. Experiments on ImageNet demonstrate the value of modelling constraints directly in network compression.

1 Introduction

Modern deep neural networks contain millions of parameters over dozens or even hundreds of layers [1, 2]. Standard benchmarks such as ImageNet [3] have incentivized the design of increasingly expensive networks, as the additional expressiveness seems necessary to correctly handle the remaining hard test samples [4]. However, the deployment of deep networks in real-world systems requires consideration of the computation cost. The issue of computation cost has led to a natural surge in interest in deep network compression [5–22].

Constraints matter in many of the most compelling deployment scenarios for compressed deep neural networks. For example, deep neural network compression enables us to deploy powerful networks in systems such as self-driving vehicles with real-time operation requirements. A self-driving vehicle may have latency constraints for executing a scene segmentation routine: if the network cannot return predictions within 50 ms on average, for instance, the safe operation of the vehicle may be compromised. As another example, deep network compression enables us to deploy compact networks on embedded platforms with limited computing power. A drone may have fixed memory constraints and only be able to run a 12 MB network on-board.

Previous work on deep network compression has focused on achieving the highest compression rate while maintaining an acceptable level of accuracy (e.g. within 1-2% of the uncompressed network's accuracy). We refer to this general

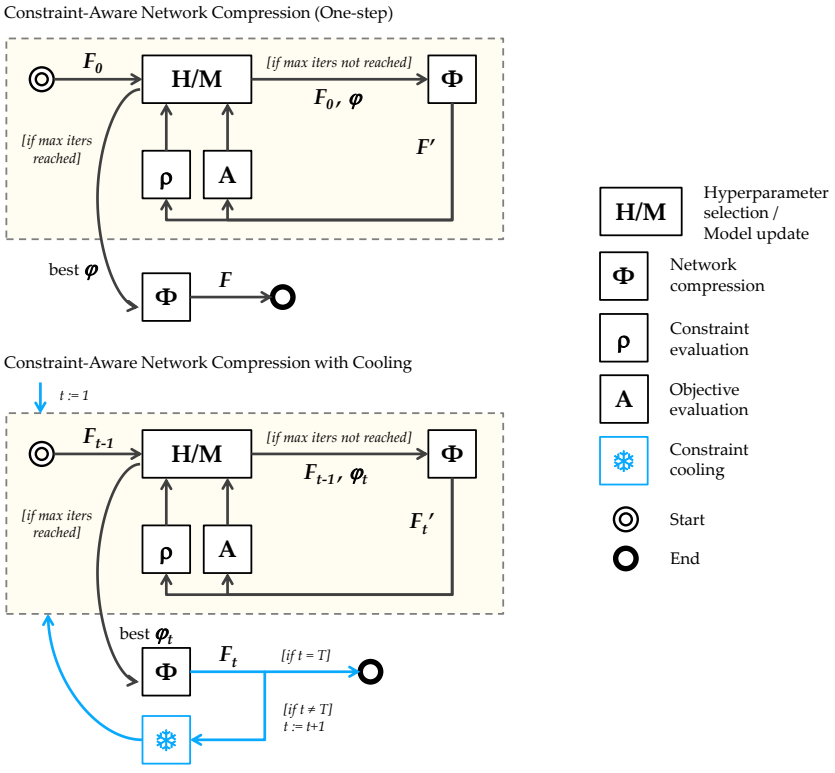


Fig. 1. We propose a framework for deep network compression under operational constraints (e.g. latency). The one-step model takes an uncompressed network F_0 and employs constrained Bayesian optimization to explore the space of compression hyperparameters ϕ such that the task objective (e.g. classification accuracy) is maximized and all constraints are satisfied. F' denotes a "proposed" compressed network. The cooling model guides network compression gradually towards the constraints via a sequence of easier intermediate targets.

approach to network compression as *unconstrained* network compression because operational constraints are not considered in the training of the compressed network. In this paper, we propose *constraint-aware* network compression, in which we incorporate operational constraints directly in the compression process. This framework allows us to ensure that the compressed network satisfies the operational constraints of the system on which the network will be deployed. Fig. 1 shows an overview of our approach.

For some types of operational constraints, such as latency, ensuring that a system always meets *hard* constraints requires verification on domain specific hardware. A wide range of systems design issues often need to be addressed to guarantee correct performance. For example, WCET (worst case execution times) analysis on multi-core processors is notoriously difficult and remains an

open research challenge [23–25]. Verification on domain specific hardware is beyond the scope of this paper. Instead, we focus on ensuring that all constraints are satisfied *in expectation*. In computer vision, this regime is similar to that of budgeted batch classification, in which a fixed time budget is available to classify a set of images and the model is trained to ensure that the time budget is satisfied in expectation [4].

To the best of our knowledge, this paper presents the first principled framework for deep network compression under operational constraints. Experiments on Describable Textures [26] and ImageNet [3] demonstrate the value of modelling constraints directly in network compression.

2 Constraint-Aware Network Compression

The problem of deep network compression with constraints can be expressed as follows. Given a pre-trained deep neural network, we would like to obtain a compressed network that satisfies a fixed set of constraints $\mathcal{C}$, while preserving the original task performance of the network as closely as possible.

Suppose we have a network compression algorithm $\Phi(F, \phi)$ that takes as input a deep network F and a set of tunable compression-related hyperparameters ϕ, and outputs a compressed network. For example, ϕ might be a vector specifying magnitude thresholds for pruning each layer in the network [8].

In an unconstrained compression setting, it is difficult to compress the network using Φ while ensuring that the operational constraints $\mathcal{C}$ are satisfied. A straightforward approach would be to repeatedly try different compression configurations until a compressed network is found that satisfies $\mathcal{C}$; however, the configuration space might be very large and each compression attempt may be computationally expensive. Even if the repeated trials are feasible, the final compressed network may not provide satisfactory performance because its training does not directly take $\mathcal{C}$ into account during optimization.

We propose a principled framework for deep network compression under operational constraints. Let $A : F \to \mathbb{R}$ map a network F to a performance metric specific to the network's task; for example, if F is a network for image classification, then A could be the top-1 classification accuracy. Let $\rho_i : F \to \mathbb{R}$ map a network to a measurement of the ith constraint condition, such as latency, energy consumption, or memory, $i = \{1, ..., |\mathcal{C}|\}$. Let $\mathbf{c} \in \mathbb{R}^{|\mathcal{C}|}$ be a vector of constraint values. We define the constraint-aware network compression problem as

$$
\begin{aligned}
F = \arg\max_{F} \; & A(F) \\
\text{subject to} \quad & \forall i \;\; \rho_i(F) \leq \mathbf{c}_i \\
& F = \Phi(F_0, \phi)
\end{aligned}
\tag{1}
$$

where F_0 is the original network to compress. For example, suppose we wish to compress a semantic segmentation network and ensure that the compressed network satisfies a maximum latency constraint of 100 ms at inference time. A

would be a semantic segmentation performance metric such as per-pixel accuracy, $|\mathcal{C}| = 1$, ρ_1 measures the latency of the network at inference time, and $\mathbf{c}_1 = 100$ ms.

To approach this difficult non-convex optimization problem, we employ constrained Bayesian optimization [27–29]. Bayesian optimization provides a general framework for optimizing black-box objective functions that are typically expensive to evaluate, non-convex, may not be expressible in closed form, and may not be easily differentiable [30]. Bayesian optimization iteratively constructs a probabilistic model of the objective function based on the outcomes of evaluating the function at various points in the input parameter space. In each iteration, the candidate point to evaluate next is determined by an acquisition function that trades off exploration (preferring regions of the input space with high model uncertainty) and exploitation (preferring regions of the input space that the model predicts will result in a high objective value). Constrained Bayesian optimization additionally models feasibility with respect to constraints.

In problem (1), we employ constrained Bayesian optimization to obtain compression hyperparameters ϕ that produce a compressed network F satisfying the constraints. We model the objective function as a Gaussian process [31]. A Gaussian process is an uncountable set of random variables, any finite subset of which is jointly Gaussian. Gaussian processes are commonly used in Bayesian optimization as they enable efficient computation of the posterior. For a more comprehensive treatment of Gaussian processes and Bayesian optimization, we refer the interested reader to [27, 29, 31]. In each iteration of our optimization, the next input ϕ is chosen using an expected improvement based acquisition function, the input network F_0 (or F_{t-1} if using a cooling schedule, discussed later) is compressed using Φ with hyperparameters ϕ, and the model is updated with the objective value and whether the constraints are satisfied. Fig. 1 (top) illustrates the basic compression process.

Running the compression algorithm Φ to completion over a large number of Bayesian optimization iterations may be prohibitively expensive. In practice, we substitute Φ with a fast approximation of Φ that skips fine-tuning the network after compression. We also estimate the objective value using a small subset of images. After Bayesian optimization determines the most promising hyperparameters ϕ, we run the full compression algorithm Φ using ϕ to produce the compressed network F.

As described so far, the Bayesian optimization attempts to find a compression configuration that immediately satisfies the operational constraints; we will refer to this strategy as *one-step* constraint-aware compression. However, we find that pursuing a gradual trajectory towards the constraints leads to better performance, especially when the constraints are aggressive. This gradual trajectory provides a sequence of easier targets that approach the constraints, and is governed by a cooling schedule. We write the constraint-aware network compression problem with cooling as a sequence of problems indexed by cooling step

Algorithm 1 Constraint-aware network compression with cooling

Input: Network compression algorithm Φ, constraints $\mathcal{C}$ (implemented as measurement functions ρ_i and values $\mathbf{c}_i$ for $i = 1, 2, ..., |\mathcal{C}|$), uncompressed network F_0, number of cooling steps T, cooling function g
Output: Compressed network F

```
F[0] = F_0
for t = 1 to T do
    Update cooled constraint values g_i(t)
    repeat
        Determine most promising compression hyperparameters φ to evaluate next based on expected improvement
        Compress F[t-1] with Φ using hyperparameters φ
        Update Gaussian process model based on objective and constraints evaluation
    until maximum iterations of constrained Bayesian optimization reached
    Compress F[t-1] with best hyperparameters φ discovered to obtain F[t]
    Fine-tune F[t]
end for
F := F[T]
```

$t = 1, 2, ..., T$:

$$F_t = \arg\max_{F_t} A(F_t) \qquad (2)$$
$$\text{subject to} \quad \forall i \ \rho_i(F_t) \leq g_i(t)$$
$$F_t = \Phi(F_{t-1}, \phi_t)$$

where T is the total number of cooling steps, and g_i is a cooling function that depends on T, the ith target constraint value $\mathbf{c}_i$, and $\rho_i(F_0)$, the initial value of the ith constraint variable (for the original uncompressed network). We require $g_i(T) = \mathbf{c}_i$ and return F_T as the final compressed network. Fig. 1 (bottom) illustrates constraint-aware network compression with cooling. In each cooling step $t = 1, 2, ..., T$, constrained Bayesian optimization is used to compress the network while ensuring that the target constraints are satisfied. At the end of a cooling iteration, we have a compressed network that satisfies the target constraints, and the target constraints are updated according to the cooling function. In the final cooling iteration T, the target constraints are equal to the operational constraints $\mathbf{c}_i$. We consider two cooling functions in this paper. For linear cooling, we define

$$g_{i,\text{linear}}(t) = \rho_i(F_0) + t/T \cdot (\mathbf{c}_i - \rho_i(F_0)) \qquad (3)$$

This cooling schedule sets the intermediate targets by linearly interpolating from the value of the uncompressed network $\rho_i(F_0)$ to the constraint $\mathbf{c}_i$. For exponential cooling, we define

$$g_{i,\text{exp}}(t) = \mathbf{c}_i + (\rho_i(F_0) - \mathbf{c}_i) \cdot e^{-\alpha t} + (\mathbf{c}_i - \rho_i(F_0)) \cdot e^{-\alpha T} \qquad (4)$$

The exponential cooling schedule sets more aggressive intermediate targets at first and cools more slowly when approaching the final constraint $\mathbf{c}_i$. The intuition is that the network may be initially easy to compress, but after several iterations of constrained compression it may become more difficult to make further progress. For exponential cooling, there is one parameter α, which controls the degree of cooling in each iteration. The process for constraint-aware network compression with cooling is summarized in Algorithm 1.

3 Experiments

We performed an initial set of experiments on the Describable Textures Dataset (DTD) [26] to explore a range of alternatives for performing network compression under operational constraints; we then performed final experiments on ImageNet (ILSVRC-2012) [3] to show the generalization of our technique to large-scale data. For concreteness, we used inference latency as a typical operational constraint in real-world systems. The performance of the classification networks is measured by top-1 accuracy.

3.1 Implementation details

We used magnitude-based pruning [6, 8, 15, 17] followed by fine-tuning as our compression strategy Φ. Magnitude-based pruning removes the weights in each layer with the lowest absolute value, with the intuition that these will have the least impact on the computation result. The compression hyperparameters ϕ are the pruning percentages for each layer. We used an ImageNet-pretrained CaffeNet (a variation of AlexNet [32]) as the original network. During constrained Bayesian optimization, the accuracy of a compressed network was measured on a subset of the training set. Fine-tuning was performed on the whole training set.

We implemented our networks in the open source library SkimCaffe [15], which can speed up sparse deep neural networks via direct convolution operation and sparse matrix multiplication on CPU. For constrained Bayesian optimization, we used the official Matlab implementation.

Latency is measured as stabilized (after model is loaded into memory) average forwarding time of one batch in SkimCaffe over 100 timing trials. As discussed in the introduction, we focus on satisfying the latency constraint in expectation, i.e. we assume an NHRT (Non Hard Real-Time) system [23, 24] for testing. Since conv1 is memory bandwidth dominant, pruning this layer gives low speedup but makes it harder to preserve the accuracy [15]; by default, we do not prune conv1 in the latency experiments.

3.2 Methods evaluated

We considered four approaches: an unconstrained baseline and three constraint-aware alternatives:

	Latency	Accuracy
Original network	207.3 ± 8.1 ms	60.7%
Unconstrained compression	59.7 ± 0.2 ms	56.3 ± 0.0%
Constraint-aware compression, one-step	57.1 ± 1.6 ms	53.9 ± 6.0%
Constraint-aware compression, linear cooling	58.4 ± 0.6 ms	57.8 ± 1.0%
Constraint-aware compression, exponential cooling	58.2 ± 0.5 ms	59.0 ± 0.4%

Table 1. Compression results for AlexNet on DTD with a latency constraint of 60 ms. Accuracy is top-1 classification accuracy. Results are averaged over five trials.

1. Unconstrained compression. This baseline repeatedly tries different compression configurations and returns the best compressed network found that satisfies the operational constraints. Starting from a compression rate of 50%, we discard half the weights in each layer, until the operational constraints are satisfied; we then iteratively increase and decrease the compression rate by binary search to satisfy the constraints as closely as possible, until the binary search interval is smaller than 0.001. We then prune the model using the found compression rate and fine-tune the whole model.
2. Constraint-aware compression with one step. This constraint-aware variation is illustrated in Fig. 1 (top), in which we do not use a cooling schedule and attempt to meet the operational constraint directly in a single step ($T = 1$).
3. Constraint-aware compression with linear cooling. This method is Fig. 1 (bottom) with a linear cooling schedule to gradually and uniformly approach the operational constraint.
4. Constraint-aware compression with exponential cooling. This method is Fig. 1 (bottom) with an exponential cooling schedule to gradually approach the operational constraint with more aggressive targets at the beginning.

3.3 Exploratory Experiments (DTD)

Our initial experiments for exploring alternatives were performed on the first train-validation-test split of the Describable Textures Dataset [26]. We performed multiple trials on a single split instead of single trials on multiple splits so that any variation in outcomes would be due to the stochastic elements of the algorithm instead of differences in the data split. The hardware platform for latency measurement was Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz. At a batch size of 32 and 4 threads, the inference latency of the original uncompressed network was 207 ms. We set the learning rate to 0.001, the number of constrained Bayesian optimization iterations to 100, the number of fine-tuning iterations in each cooling step to 1500, the number of cooling steps T to 5, and the exponential coefficient for exponential cooling to 0.5. The unconstrained and one-step baselines were fine-tuned to 3000 iterations, which was enough for convergence.

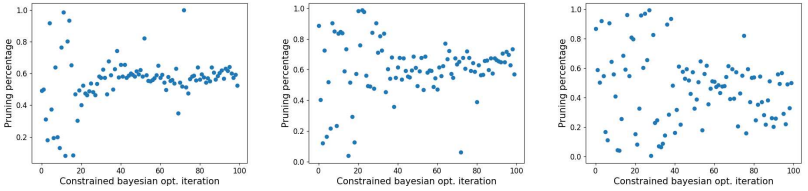
Method comparisons. Table 1 shows the final latency and top-1 accuracy results under a latency constraint of 60 ms for all methods. Means and standard

	Pruning rate	Latency before	Latency after
conv2	95.06 ± 1.85 %	52.9 ± 2.7 ms	7.8 ± 1.8 ms
conv3	92.19 ± 4.25 %	30.2 ± 1.6 ms	4.9 ± 2.3 ms
conv4	85.29 ± 4.26 %	25.6 ± 1.4 ms	6.5 ± 1.7 ms
conv5	77.61 ± 4.02 %	19.1 ± 1.2 ms	8.3 ± 1.0 ms
fc6	96.03 ± 2.33 %	15.6 ± 0.3 ms	2.4 ± 0.7 ms
fc7	90.05 ± 6.42 %	6.3 ± 0.2 ms	1.8 ± 0.8 ms
fc8	82.07 ± 8.35 %	0.1 ± 0.01 ms	0.07 ± 0.02 ms

Table 2. Layer-wise compression results for the constraint-aware compression with exponential cooling method in Table 1.

deviations were computed over five independent runs to account for stochastic elements of the algorithm. For a given run, the latency of the compressed network was obtained via 100 timing measurements to account for variance from the CPU environment; the standard deviation of the timing measurements was 0.3 ms on average. Compared to the unconstrained compression baseline, constraint-aware compression with cooling obtained 2.7% higher accuracy. The exponential cooling schedule led to a higher final accuracy than the linear cooling schedule, suggesting that rapid initial cooling followed by a more conservative final approach was an effective strategy in this case.

Why does cooling provide better performance? One might ask whether a better hyperparameter sweep would suffice for the unconstrained or one-step baselines to match the performance of linear or exponential cooling. Why is cooling valuable? The cooling schedule induces a series of easier compression problems and allows the compression to adapt to the network structure as it changes over time [16]. Since the network is fine-tuned at the end of each cooling step, each round of Bayesian optimization starts from an initial network F_{t-1} with structure that is closer to the final compressed network. A one-step or unconstrained approach does not benefit from these intermediate network structures. We performed an additional experiment in which we ran the one-step baseline using the compression hyperparameters found in the final cooling step of exponential cooling. This resulted in an accuracy of 52.6 ± 2.4%, which is worse than the result of the normal exponential cooling method, with compression performed over multiple steps. Interestingly, if instead of transferring only the compression hyperparameters from exponential cooling to the one-step baseline, we also transferred the network *structure* (i.e. the sparsity structure), then the accuracy improved to 58.3 ± 1.2%. This suggests that the exponential cooling approach does not perform better simply because of a better hyperparameter search, but that the setting of progressive targets and the intermediate fine-tuning are helpful in evolving the network to the highest performing compressed structure.



(a) conv2 (b) conv3 (c) fc7

Fig. 2. Visualization of the pruning rates proposed by constrained Bayesian optimization for the first cooling step of a single trial in Table 2.

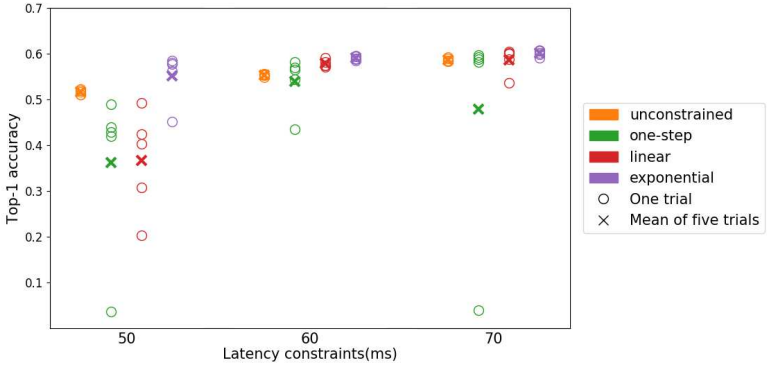


Fig. 3. Compression results for AlexNet on DTD for latency constraints of 50 ms, 60 ms, and 70 ms. Constraint-aware compression with exponential cooling provides the best overall performance for all three tested constraint values.

Layer-wise results. Table 2 shows the average latency and pruning rate for each layer obtained by the constraint-aware compression with exponential cooling method in Table 1. Fig. 2 shows the evolution of the individual pruning rates proposed by Bayesian optimization for the conv2, conv3, and fc7 layers, for the first cooling step of a single trial with exponential cooling. Bayesian optimization quickly clusters around an effective pruning rate range for conv2 and conv3, while the rates proposed for fc7 are more scattered, even at the maximum number of iterations. This suggests that the quality of the compressed solutions is more dependent on how conv2 and conv3 are pruned than on how fc7 is pruned, which is expected since convolution layers contribute more to the overall latency than fully connected layers. Likewise, we can observe from Table 2 that the variance in pruning rate is higher for fc7 and fc8 than for the convolution layers.

Generalization to different constraint values. Fig. 3 shows compression performance over a range of latency constraint values. Our observations about

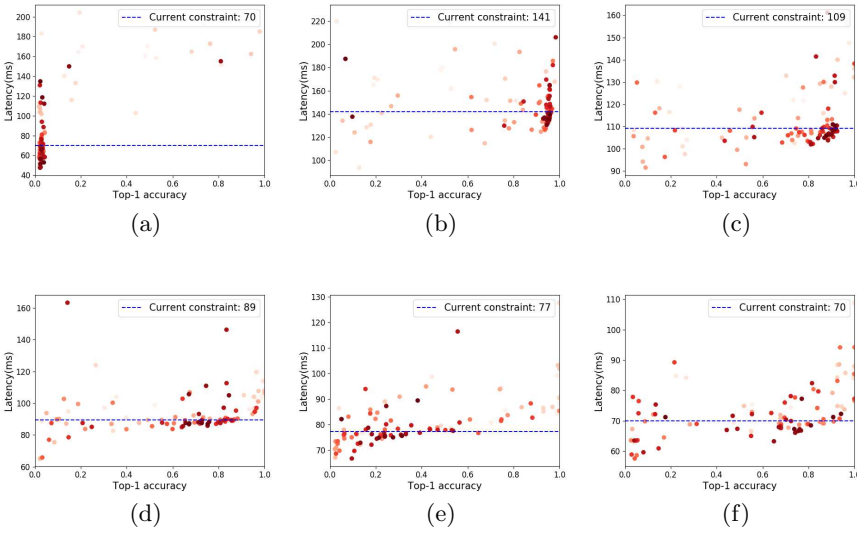


Fig. 4. A visualization of the compression solutions proposed by constrained Bayesian optimization for the 70 ms experiment in Fig. 3. Deeper saturation corresponds to a later iteration in the Bayesian optimization. (a) Single-step compression failure case. (b-f) Compression with exponential cooling. See text for discussion.

the importance of cooling generalize to other latency targets. The large variance in the one-step baseline in both the 50 ms and 70 ms experiments is due to failures in a single trial in the respective experiments. To explain these failures, Fig. 4(a) visualizes the compressed networks proposed by constrained Bayesian optimization for the 70 ms trial in which the single-step baseline failed. The saturation of the data points indicates the Bayesian optimization iteration in which that compressed network was proposed; deeper saturation corresponds to a later iteration. We can see that the feasible solutions proposed by Bayesian optimization (data points under the dotted line) are almost equally poor in terms of accuracy. In this case, the model is unable to distinguish between compressed networks that can be improved with sufficient fine-tuning and networks that cannot be improved with any amount of fine-tuning. Fig. 4(b-f) visualize the compressed networks proposed by constrained Bayesian optimization for a trial of the exponential cooling method. In the case of exponential cooling, since compression is performed gradually with intermediate targets over several iterations, constrained Bayesian optimization is able to consistently converge to high-accuracy solutions that respect the operational constraint.

One way to mitigate this failure mode for the one-step baseline is to perform a look ahead step during constrained Bayesian optimization: for every compressed network proposed by Bayesian optimization, we partially fine-tune the proposed network (e.g. we look ahead one epoch) to obtain a better estimate of the final

	Latency	Accuracy
Original network	237.0 ± 2.9 ms	57.41%
Unconstrained compression	69.2 ± 1.3 ms	50.62 ± 0.50%
Constraint-aware compression, one-step	62.2 ± 4.3 ms	47.27 ± 3.31%
Constraint-aware compression, exponential cooling	69.7 ± 0.7 ms	53.70 ± 0.15%

Table 3. Compression results for AlexNet on ImageNet with a latency constraint of 70 ms. Accuracy is top-1 classification accuracy. Results are averaged over three trials.

accuracy. We implemented this variation and found that looking ahead improves the average accuracy from 53.9% to 55.9% and reduces the standard deviation from 6.0% to 1.1%. However, fine-tuning each proposed network increases the computation overhead of Bayesian optimization and may not be suitable for large-scale datasets such as ImageNet.

Limitations. In our experiments, we have assumed that the operational constraint value is feasible given the selected network F and compression method Φ. If the constraint cannot be satisfied even if the network is maximally compressed using Φ (e.g. in our case, if the layers are pruned to the extent of leaving a single non-zero weight in each layer), then our framework cannot propose a feasible solution. The lower bound on the operational constraint depends on Φ: for instance, in knowledge distillation [9, 14] the architecture of the compressed (student) network is typically different from that of the original (teacher) network, so it may be able to achieve more extreme latency targets than a method that keeps the original network architecture.

3.4 Large-Scale Experiments (ImageNet)

To demonstrate that our framework generalizes to large-scale data, we performed experiments on ImageNet (ILSVRC-2012). We performed three independent trials on the standard training and validation split. The hardware platform for latency measurement was Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz. We set the learning rate to 0.001, the number of constrained Bayesian optimization iterations to 200, the number of fine-tuning iterations in each cooling step to 10K, the number of cooling steps T to 10, and the exponential coefficient for exponential cooling to 0.5. We matched the total number of fine-tuning iterations to be 150K for all baselines.

Method comparisons. Table 3 shows the averaged results for unconstrained compression, constraint-aware compression in one-step, and constraint-aware compression with exponential cooling, under an operational constraint of 70 ms latency. Compared to unconstrained compression, constraint-aware compression with exponential cooling obtains 3.1% higher top-1 accuracy while satisfying the operational constraint.

	Pruning rate	Storage before	Storage after
conv1	27.37 %	0.13 MB	0.10 MB
conv2	35.39 %	1.17 MB	0.76 MB
conv3	50.12 %	3.38 MB	1.68 MB
conv4	55.85 %	2.53 MB	1.12 MB
conv5	55.41 %	1.69 MB	0.75 MB
fc6	98.29 %	144.02 MB	2.46 MB
fc7	96.79 %	64.02 MB	2.05 MB
fc8	84.61 %	15.63 MB	2.41 MB
overall	95.13 %	232.56 MB	11.33 MB

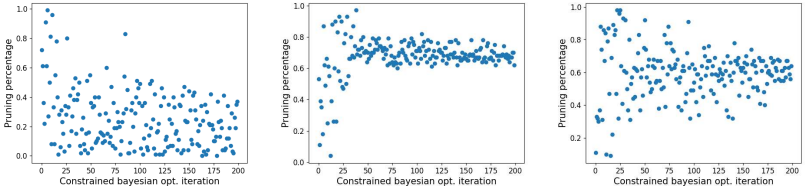
Table 4. Layer-wise compression results for constraint-aware compression with a storage constraint of 5% of the original storage costs, AlexNet on ImageNet.

Layer-wise results. Detailed layer-wise results can be found in the supplementary material. Similar to the DTD experiments, Bayesian optimization quickly clusters around an effective pruning rate range for conv2 and conv3, while the rates proposed for fc7 are more scattered, even at the maximum number of iterations. The highest variances in pruning rate are for fc7 and fc8. Similar to DTD, the quality of the compressed solutions depends more on how the convolution layers are pruned than on how the fc7 and fc8 layers are pruned, which is expected given the constraint on inference latency.

Time requirements. In one step of the cooling schedule, constrained Bayesian optimization is first used to search for the hyperparameters and then fine-tuning is performed. The constrained Bayesian optimization takes 1.5 hours for 200 iterations. One iteration is roughly 30 s, and consists of 17 s of accuracy measurement, 10 s of latency test, and 3 s of Bayesian optimization calculation. We perform 150K fine-tuning iterations, which requires 30 hours on a 1080Ti.

Generalization to other constraint types. To demonstrate that our framework generalizes to different types of operational constraints, we performed a preliminary experiment on ImageNet with a constraint on the storage requirements of the network. We set the maximum storage cost to be 5% of the original cost and ran a single trial of constraint-aware compression with exponential cooling. We set the learning rate to 0.001, the number of constrained Bayesian optimization iterations to 200, the number of fine-tuning iterations in each cooling step to 10K, the number of cooling steps T to 10, and the exponential coefficient to 0.4. After the final cooling step, we fine-tuned for an additional 350K iterations in which we started with a learning rate of 0.001 and reduced the learning rate by a factor of 10 after every 150K iterations.

The storage cost of the original network is 232.56 MB. Constraint-aware compression with exponential cooling produces a compressed network with storage cost 11.33 MB (5% of the original cost) and top-1 accuracy of 54.84%. Table 4



(a) conv2 (b) fc6 (c) fc7

Fig. 5. Visualization of the pruning rates proposed by constrained Bayesian optimization for the first cooling step of the storage-constraint experiment in Table 4.

shows the storage cost and pruning rate for each layer, and Fig. 5 visualizes the individual pruning rates proposed by Bayesian optimization for the conv2, fc6, and fc7 layers. In contrast to the previous experiments with latency constraints, given a storage constraint, our framework learns a policy that prioritizes the fully connected layers: Bayesian optimization quickly converges for fc6 and fc7, while the proposed pruning rates for conv2 are more scattered, even at the maximum number of iterations; in Table 4, the fully connected layers are pruned more aggressively than the convolution layers. This contrasting behavior is expected because fully connected layers contribute more to the storage costs (they have more weights to store) than convolution layers. We can see that the compression behavior of constraint-aware compression automatically adapts to the type of operational constraint that the system is required to satisfy.

3.5 Comparison to Guided Sparsity Learning

The timing performance of compression algorithms is dependent on the hardware platform and software libraries used to implement key network operations such as convolution. We draw a comparison with Guided Sparsity Learning (GSL) [12] in SkimCaffe as our implementation is in SkimCaffe. GSL is specifically optimized for inference speed: compression is guided by a performance model that predicts the speedup potential of each layer, tuned to hardware characteristics (e.g. compute capability in FLOP/s, memory bandwidth). On DTD, GSL achieves a latency of 74.2 ms with a top-1 accuracy of 60.9%. This result motivated us to set successively harder latency targets of 70 ms, 60 ms, and 50 ms in our DTD experiments. Despite being hardware agnostic, and not specifically optimized for speed, our method obtains competitive performance at these aggressive targets: 60.1% at 70 ms and 59.0% at 60 ms. We also performed an additional comparison on ImageNet. On ImageNet, GSL achieves a latency of 78.2 ms with a top-1 accuracy of 57.5%. For a direct comparison, we set a latency target of 78.2 ms for constraint-aware compression. At 78.2 ms, constraint-aware compression achieves a top-1 accuracy of 57.4%. The results show that, in a fair comparison with the same hardware and software platforms, our method obtains compara-

ble latency performance to optimized GSL, while requiring no hardware-specific tuning and providing generality to other constraint types besides latency.

4 Related Work

Network pruning methods sparsify the connections in a pre-trained network and then fine-tune the sparsified network to restore accuracy. The earliest pruning methods removed connections based on the second-order derivatives of the network loss [33, 34]. A recent common pruning strategy removes the connections with the lowest magnitude weights, with the intuition that low-magnitude weights are likely to have less impact on the computation result if set to zero [6, 8, 15, 17]. Structured pruning methods remove entire filters instead of invididual connections [11, 19, 35]; this produces a speed-up in deep learning frameworks that implement convolutions as large matrix multiplications, at the possible cost of lower compression rates [15].

Weight quantization methods represent connections using a small set of permitted weight values, reducing the number of bits required to store each connection [12, 18]. For example, if 64 unique values are permitted, then each connection can be represented using 6 bits. At the extreme, weights can be quantized to a single bit [7, 13, 20]. Weight quantization and pruning are complementary and can be combined sequentially or jointly [8, 17].

Knowledge distillation uses a more expensive teacher network to guide the training of a smaller student network [9, 14]. Low-rank approximation methods exploit the redundancy in filters and feature maps [5, 10].

Any of these methods can in principle be plugged into our constraint-aware compression framework as the module Φ (see Fig. 1), provided that it exposes a set of tunable compression hyperparameters, accepts an uncompressed or partially compressed deep network as input, and outputs a compressed deep network. To the best of our knowledge, our study is the first principled treatment of deep network compression under operational constraints.

5 Conclusion

Advances in deep neural network compression have the potential to bring powerful networks to limited-compute platforms such as drones, mobile robots, and self-driving vehicles. We argue that our network compression algorithms should be constraint-aware, because in the real world, computation is not free and the operational constraints matter. In this paper, we have presented a general framework for training compressed networks that satisfy operational constraints in expectation. Our framework is complementary to specific compression techniques (e.g. distillation, pruning, quantization) and can accommodate any of these as its compression module Φ. In future, we plan to study whether the constraint cooling schedule can be learned, for example by using reinforcement learning.

Acknowledgements. This work was supported by the Natural Sciences and Engineering Research Council of Canada.

References

1. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition. (2016)
2. Huang, G., Liu, Z., van der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: IEEE Conf. on Computer Vision and Pattern Recognition. (2017)
3. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet Large Scale Visual Recognition Challenge. arXiv:1409.0575 (2014)
4. Huang, G., Chen, D., Li, T., Wu, F., van der Maaten, L., Weinberger, K.: Multi-scale dense networks for resource efficient image classification. In: International Conference on Learning Representations. (2018)
5. Denton, E., Zaremba, W., Bruna, J., LeCun, Y., Fergus, R.: Exploiting linear structure within convolutional networks for efficient evaluation. In: Advances in Neural Information Processing Systems. (2014)
6. Guo, Y., Yao, A., Chen, Y.: Dynamic network surgery for efficient DNNs. In: Advances in Neural Information Processing Systems. (2016)
7. Guo, Y., Yao, A., Zhao, H., Chen, Y.: Network sketching: exploiting binary structure in deep CNNs. In: IEEE Conference on Computer Vision and Pattern Recognition. (2017)
8. Han, S., Mao, H., Dally, W.J.: Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. In: International Conference on Learning Representations. (2016)
9. Hinton, G., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network. arXiv:1503.02531 (2015)
10. Jaderberg, M., Vedaldi, A., Zisserman, A.: Speeding up convolutional neural networks with low rank expansions. In: British Machine Vision Conference. (2014)
11. Lebedev, V., Lempitsky, V.: Fast ConvNets using group-wise brain damage. In: IEEE Conference on Computer Vision and Pattern Recognition. (2016)
12. Park, E., Ahn, J., Yoo, S.: Weighted-entropy-based quantization for deep neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition. (2017)
13. Rastegari, M., Ordonez, V., Redmon, J., Farhadi, A.: XNOR-Net: ImageNet classification using binary convolutional neural networks. In: European Conference on Computer Vision. (2016)
14. Romero, A., Ballas, N., Kahou, S.E., Chassang, A., Gatta, C., Bengio, Y.: FitNets: hints for thin deep nets. In: International Conference on Learning Representations. (2015)
15. Park, J., Li, S., Wen, W., Tang, P., Li, H., Chen, Y., Dubey, P.: Faster CNNs with direct sparse convolutions and guided pruning. In: International Conference on Learning Representations. (2017)
16. Tung, F., Muralidharan, S., Mori, G.: Fine-pruning: Joint fine-tuning and compression of a convolutional network with Bayesian optimization. In: British Machine Vision Conference. (2017)
17. Tung, F., Mori, G.: CLIP-Q: Deep network compression learning by in-parallel pruning-quantization. In: IEEE Conference on Computer Vision and Pattern Recognition. (2018)
18. Ullrich, K., Meeds, E., Welling, M.: Soft weight-sharing for neural network compression. In: International Conference on Learning Representations. (2017)

19. Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H.: Learning structured sparsity in deep neural networks. In: Advances in Neural Information Processing Systems. (2016)
20. Xu, F., Boddeti, V.N., Savvides, M.: Local binary convolutional neural networks. In: IEEE Conference on Computer Vision and Pattern Recognition. (2017)
21. Yang, T.J., Chen, Y.H., Sze, V.: Designing energy-efficient convolutional neural networks using energy-aware pruning. In: IEEE Conference on Computer Vision and Pattern Recognition. (2017)
22. Zhou, H., Alvarez, J.M., Porikli, F.: Less is more: towards compact CNNs. In: European Conference on Computer Vision. (2016)
23. Paolieri, M., Quiñones, E., Cazorla, F.J., Bernat, G., Valero, M.: Hardware support for WCET analysis of hard real-time multicore systems. In: International Symposium on Computer Architecture. (2009)
24. Ungerer, T., et al.: Merasa: Multicore execution of hard real-time applications supporting analyzability. IEEE Micro **30**(5) (2010) 66–75
25. Abella, J., et al.: WCET analysis methods: Pitfalls and challenges on their trustworthiness. In: IEEE International Symposium on Industrial Embedded Systems. (2015)
26. Cimpoi, M., Maji, S., Kokkinos, I., Mohamed, S., Vedaldi, A.: Describing textures in the wild. In: IEEE Conference on Computer Vision and Pattern Recognition. (2014)
27. Snoek, J., Larochelle, H., Adams, R.P.: Practical Bayesian optimization of machine learning algorithms. In: Advances in Neural Information Processing Systems. (2012)
28. Gelbart, M.A., Snoek, J., Adams, R.P.: Bayesian optimization with unknown constraints. In: Conference on Uncertainty in Artificial Intelligence. (2014)
29. Gardner, J.R., Kusner, M.J., Xu, Z., Weinberger, K.Q., Cunningham, J.P.: Bayesian optimization with inequality constraints. In: International Conference on Machine Learning. (2014)
30. Wang, Z., Zoghi, M., Hutter, F., Matheson, D., de Freitas, N.: Bayesian optimization in high dimensions via random embeddings. In: International Joint Conference on Artificial Intelligence. (2013)
31. Rasmussen, C.E., Williams, C.K.I.: Gaussian Processes for Machine Learning. MIT Press (2006)
32. Krizhevsky, A., Sutskever, I., Hinton, G.E.: ImageNet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems. (2012)
33. Hassibi, B., Stork, D.G.: Second order derivatives for network pruning: optimal brain surgeon. In: Advances in Neural Information Processing Systems. (1992)
34. LeCun, Y., Denker, J.S., Solla, S.A.: Optimal brain damage. In: Advances in Neural Information Processing Systems. (1990)
35. Luo, J.H., Wu, J., Lin, W.: ThiNet: A filter level pruning method for deep neural network compression. In: IEEE International Conference on Computer Vision. (2017)