Deep Neural Network Compression by In-Parallel Pruning-Quantization

Frederick Tung and Greg Mori

Abstract—Deep neural networks enable state-of-the-art accuracy on visual recognition tasks such as image classification and object detection. However, modern networks contain millions of learned connections, and the current trend is towards deeper and more densely connected architectures. This poses a challenge to the deployment of state-of-the-art networks on resource-constrained systems, such as smartphones or mobile robots. In general, a more efficient utilization of computation resources would assist in deployment scenarios from embedded platforms to computing clusters running ensembles of networks. In this paper, we propose a deep network compression algorithm that performs weight pruning and quantization jointly, and in parallel with fine-tuning. Our approach takes advantage of the complementary nature of pruning and quantization and recovers from premature pruning errors, which is not possible with two-stage approaches. In experiments on ImageNet, CLIP-Q (Compression Learning by In-Parallel Pruning-Quantization) improves the state-of-the-art in network compression on AlexNet, VGGNet, GoogLeNet, and ResNet. We additionally demonstrate that CLIP-Q is complementary to efficient network architecture design by compressing MobileNet and ShuffleNet, and that CLIP-Q generalizes beyond convolutional networks by compressing a memory network for visual question answering.

Index Terms—deep learning, neural network compression, weight pruning, weight quantization, Bayesian optimization

1 Introduction

Deep neural networks have become indispensable tools for a wide range of visual recognition tasks, such as image classification [1], [2], [3], object detection [4], [5], [6], semantic segmentation [7], [8], [9], [10], and visual question answering [11], [12], [13], [14]. The capacity of deep neural networks to act as powerful non-linear function approximators is made possible by their millions of learnable connection weights. In general, there has been a trend towards deeper architectures with increasing numbers of learnable connections [3], [15], [16]. However, many practical applications of computer vision require efficient solutions with low memory and energy footprint. For example, a mobile robot may need to map its surroundings while detecting objects and people, or a smartphone app may need to perform fine-grained classification of animals in the wild. A more efficient utilization of computation resources would assist in a variety of deployment scenarios, from resource-constrained platforms to computing clusters running ensembles of deep networks. An important research question is therefore: how can we make state-of-the-art deep neural networks, with millions of parameters, more compact and efficient?

The focus of this paper is deep network compression, which has the goal of making deep networks more compact [17], [18], [19]. Complementary lines of work focus on accelerating deep network inference at test time [20], [21], [22], [23], or reducing deep network training times [24], [25]. The conventional understanding is that large numbers of connections (weights) are necessary for training deep networks with high predictive accuracy; however, once the network has been trained, there will typically be a high degree of parameter redundancy [26].

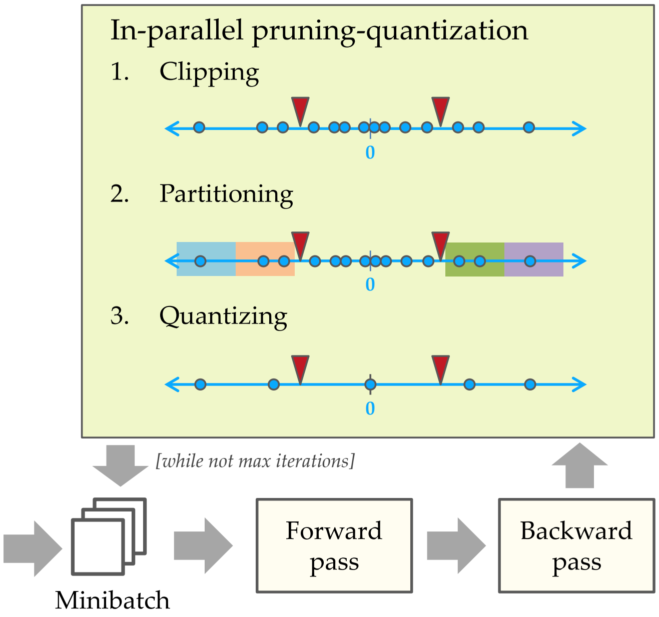


Fig. 1. An overview of our deep network compression approach. CLIP-Q combines weight pruning and quantization in a single learning framework, and performs pruning and quantization in parallel with fine-tuning. The joint pruning-quantization adapts over time with the changing network.

Network pruning is a compression strategy in which network connections are reduced, or sparsified; weight quantization is another strategy in which connection weights are constrained to a set of discrete values, allowing weights

- *F. Tung and G. Mori are with the School of Computing Science, Simon Fraser University, Burnaby, Canada.*
E-mail: ftung@sfu.ca, mori@cs.sfu.ca

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2018.2886192, IEEE Transactions on Pattern Analysis and Machine Intelligence

to be represented using fewer bits. This paper presents CLIP-Q (Compression Learning by In-Parallel Pruning-Quantization), an approach to deep network compression that (1) combines network pruning and weight quantization in a single learning framework that solves for both weight pruning and quantization jointly; (2) makes flexible pruning and quantization decisions that adapt over time as the network structure changes; and (3) performs pruning and quantization in parallel with fine-tuning the full-precision weights (Fig. 1). Experiments on six modern networks for ImageNet classification demonstrate the potential of the proposed approach.

A preliminary version of this paper appeared in [27]. The present paper includes extended related work; additional visualization of pruning-quantization hyperparameter prediction via Bayesian optimization; new compression experiments with VGGNet [16]; new compression experiments with MobileNet [28] and ShuffleNet [29], demonstrating that CLIP-Q is complementary to efficient network architecture design; and new compression experiments with a memory network for visual question answering [14], showing that CLIP-Q generalizes beyond convolutional networks for image classification.

2 Related Work

2.1 Parameter pruning

Network pruning is a common and intuitive approach to deep network compression. Pruning methods remove “unimportant” connections from 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 [30], [31]. Data-free parameter pruning [32] discovers and removes redundant neurons using a data-independent technique. Deep compression [18] 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. It achieves further compression by then quantizing the remaining weights and applying Huffman coding. Our method is different in that we perform pruning and quantization jointly instead of sequentially. This allows us to take advantage of the complementary nature of these tasks and to recover from earlier pruning errors, which is not possible with a two-stage approach.

Determining the importance of a connection is difficult due to the complex interactions among neurons: a connection that initially seems unimportant may become important when other connections are removed. This presents a significant challenge to pruning algorithms as most make hard (permanent) pruning choices during the optimization [17]. Dynamic network surgery [17] additionally performs weight splicing, which restores previously pruned connections. Connections are pruned and spliced based on the magnitude of their weights. In a similar spirit, CLIP-Q makes flexible pruning choices that can adapt to changes to the network over time during training. Different from dynamic network surgery, we incorporate both network pruning and weight quantization in the optimization and solve for the pruning-quantization parameters jointly.

Generalized convolution is often implemented in modern deep learning frameworks by reshaping filter and patch matrices and performing large matrix multiplications using highly optimized BLAS libraries. Structured pruning methods prune entire filters instead of individual connections, eliminating entire rows or columns from those large matrices and reducing inference time [33], [34], [35]. For example, ThiNet [35] prunes filters by greedily determining which channels in the next layer’s input feature map can be removed while minimizing the next layer’s reconstruction error. However, structured sparsity constraints may hinder the compression level that could otherwise be achieved if arbitrary (unstructured) sparsity patterns are allowed [36]. Unstructured pruning can benefit from specialized hardware engines [37] or efficient sparse-with-dense matrix multiplication [36] for practical test-time acceleration.

Besides compression, network pruning has also been used to regularize the training of more accurate full-sized networks [38], reduce over-fitting in transfer learning [39], and produce energy-efficient networks for battery-powered devices [40].

2.2 Parameter quantization and binarization

Weight quantization refers to the process of discretizing the range of weight values so that each weight can be represented using fewer bits. For example, if weights can take on only 16 discrete values (quantization levels), then each weight can be encoded in 4 bits instead of the usual 32 bits for a single-precision value.

Deep compression [18] performs weight quantization separately from pruning in a two-stage approach. Quantization levels are linearly distributed to cover the range of weights. Soft weight-sharing [41] re-trains a network while fitting the weights to a Gaussian mixture model prior. Quantization levels correspond to centers in the Gaussian mixture model, and pruning can be incorporated by enforcing a mixture component at zero. Though theoretically principled, the method is computationally expensive and practical only for small networks [41]. Weighted-entropy-based quantization [42] distributes quantization levels using a weighted entropy measure that encourages fewer quantization levels to be allocated to near-zero and very high-magnitude weights, and a more balanced distribution of quantization levels to be allocated to weights away from these extremes.

Quantized convolutional networks [43] express the forward passes of convolutional and fully connected layers as inner products, which can be approximated using product quantization. Our method performs scalar quantization [18], [41], [42] instead of product or vector quantization.

In the limit, weights can be quantized to a single bit to form binary weight networks [44], [45]. Network sketching [46] binarizes pre-trained networks by approximating real-valued filters with weighted combinations of binary filters. Local binary convolutional networks [47] replace convolutional layers with a learnable module inspired by local binary patterns [48].

2.3 Structured matrix

Structured projection methods replace the fully connected layers of convolutional networks, which can be viewed as unstructured projections, with efficient structured projections that can be specified using fewer weights. For example, circulant neural networks [49] replace fully connected layers with learned circulant projections, which can be computed quickly using the Fast Fourier Transform, and deep fried convnets [50] replace fully connected layers with an adaptive version of the Fastfood transform.

2.4 Low-rank factorization

Low-rank factorization methods exploit the redundancy in filters and feature map responses [51], [52], [53]. For example, Jaderberg et al. [52] exploit the low-rank structure of convolutional layers by decomposing the full-rank convolution into a convolution with horizontal basis filters followed by a convolution with vertical basis filters.

2.5 Knowledge distillation

Knowledge distillation [54] trains a compact student network using a weighted combination of ground truth labels and the soft outputs of a larger, more expensive teacher network. FitNets [55] add further regularization to knowledge distillation by guiding the training of one of the student network's intermediate layers using the outputs of an intermediate layer of the teacher network. N2N learning [56] uses reinforcement learning to train policy networks for deriving a student network from a teacher network.

2.6 Efficient architectures

An orthogonal research direction is the design of efficient network architectures. SqueezeNet [57] replaces 3x3 convolutional filters with 1x1 filters, reduces the number of input channels to 3x3 filters, and downsamples later in the network. Xception [58] replaces Inception modules [2] with depthwise separable convolutions. MobileNet [28] targets embedded applications and stacks several layers of depthwise separable convolutions. ShuffleNet [29] employs pointwise group convolutions with channel "shuffling". Multiscale DenseNet [59] uses early-exit classifiers to enable anytime classification and budgeted batch classification. We experimentally show that pruning and quantization are complementary to efficient network architecture design.

2.7 Test-time acceleration

Inference speed is another important consideration when running deep networks on mobile or embedded platforms. Methods dedicated to accelerating deep network inference at test time (without prior compression) focus on reducing the amount of exact computation needed. Perforated convolutional layers [22] perform exact computations in a subset of spatial locations and interpolate the remaining using the nearest spatial neighbors. Motivated by the fact that negative responses are discarded by ReLU activations, low-cost collaborative layers [20] predict which spatial locations will produce negative responses and skips those locations. Spatially adaptive computation time [21] accelerates residual networks by learning to predict halting scores that allow computation to stop early within a block of residual units.

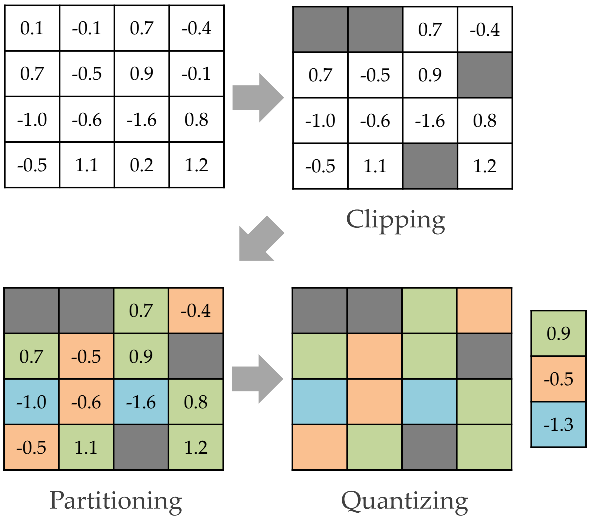
3 METHOD

The goal of our method is to learn a compressed deep network by fine-tuning a pre-trained deep network while (a) removing connections (weights) and (b) reducing the number of bits required to encode the remaining connections. To accomplish this, we combine network pruning and weight quantization in a single operation and learn the pruned network structure and quantized weights together. Our method performs pruning-quantizations that adapt over the course of training as the network structure evolves. Connections may be removed and restored in a later iteration as the network learns more efficient structures. Quantization levels and assignments of connections likewise adapt to the network structure over time.

3.1 In-parallel pruning-quantization

An overview of our approach is presented in Fig. 1. In parallel with network fine-tuning, we perform a pruning-quantization operation on each layer that consists of three steps:

1) **Clipping.** We place two "clips", scalars c^- and c^+, such that $(p \times 100)\%$ of the positive weights in the layer are less than or equal to c^+, and $(p \times 100)\%$ of the negative weights are greater than or equal to c^-. All the weights between c^- and c^+ are set to zero in the next forward pass. This removes the corresponding connections from the network when processing the next minibatch. Note that this pruning decision is impermanent: in the next iteration, we apply the rule again on updated weights, and previously pruned connections can return. While the hyperparameter p is constant, the thresholds c^- and c^+ change in each iteration. Like paper clips, c^- and c^+ can be flexibly "moved" along the 1-D axis of weight values. Clips are represented by red triangles in Fig. 1.
2) **Partitioning.** In the second step, we partition the non-clipped portion of the 1-D axis of weight values into quantization intervals. These intervals are visualized as different colored ranges in Fig. 1. Given a precision budget of b bits per weight, this step produces a partitioning of the 1-D axis into $2^b - 1$ intervals, plus the zero interval between the clips c^- and c^+. We perform linear (uniform) partitioning [18] to the left of c^- and to the right of c^+ for simplicity; other partitioning strategies (e.g. weighted entropy [42]) could also be used for improved accuracy.
3) **Quantizing.** We next update the quantization levels – the discrete values that the weights are permitted to take in the compressed network. Each quantization level is computed by averaging the full-precision weights falling within the corresponding quantization interval (colored ranges in Fig. 1). We then quantize the weights by setting them to the new quantization levels in the next forward pass. Similar to pruning, the quantization applies to the next minibatch and is then re-evaluated. The quantization levels and assignments of weights to levels evolve over time as the learning progresses.



0.1	-0.1	0.7	-0.4
0.7	-0.5	0.9	-0.1
-1.0	-0.6	-1.6	0.8
-0.5	1.1	0.2	1.2

		0.7	-0.4
0.7	-0.5	0.9	
-1.0	-0.6	-1.6	0.8
-0.5	1.1		1.2

		0.7	-0.4
0.7	-0.5	0.9	
-1.0	-0.6	-1.6	0.8
-0.5	1.1		1.2

0.9
-0.5
-1.3

Fig. 2. An example illustrating the three steps of the pruning-quantization operation for a layer with 16 weights, $p = 0.25$ and $b = 2$. Pruning-quantization is performed in parallel with fine-tuning the network's full-precision weights, and updates the pruning statuses, quantization levels, and assignments of weights to quantization levels after each training minibatch.

Fig. 2 illustrates the pruning and quantization operation using a layer with 16 weights. Suppose the pruning rate $p = 0.25$ and the bit budget $b = 2$. We first apply clipping, which sets the two lowest-magnitude negative weights and the two lowest-magnitude positive weights to zero, in effect removing the corresponding connections from the network. We then linearly partition the 1-D axis of weight values. Finally, we compute the quantization levels by averaging the weight values within each partition, and set the weights to these discrete values. These steps are repeated with the next training minibatch using the new full-precision weights.

During training, both the quantized and full-precision weights are tracked. The full-precision weights are used in the pruning-quantization update as well as during back-propagation [42], [45], while the forward pass uses the quantized weights, simulating the output of the compressed network. A summary of the training process is shown below.

```
Input: Full-precision layer weights $\mathbf{w}_1 ... \mathbf{w}_L$, pruning-quantization hyperparameters $\{p_1 ... p_L, b_1 ... b_L\}$ pre-computed using Bayesian optimization (Sect. 3.2), network learning parameters
Output: Quantized weights $\hat{\mathbf{w}}_1 ... \hat{\mathbf{w}}_L$
repeat
    for each layer $i$ = 1 to $L$ do
        Clip using full-precision weights $\mathbf{w}_i$
        Partition full-precision weights
        Quantize full-precision weights to update $\hat{\mathbf{w}}_i$
    end for
    Input minibatch of images
    Forward propagate and compute loss with $\hat{\mathbf{w}}_1 ... \hat{\mathbf{w}}_L$
    Backpropagate to update full-precision weights $\mathbf{w}_1 ... \mathbf{w}_L$
until maximum iterations reached
```

After training is complete, the full-precision weights are discarded and only the quantized weights need to be stored. We store the weights of the compressed network using a standard sparse encoding scheme [18], [41]. In brief, the structure of the sparse weight matrix is encoded using index differences, and each non-pruned weight stores the b-bit identifier of its quantization level. Full implementation details can be found in [18].

To summarize, we update pruning statuses, quantization levels, and quantization level assignments with each training minibatch, allowing these to adapt over time as needed. For example, a previously pruned connection may become relevant again once other connections have been pruned, in which case it can be spliced back into the network. In addition, connections can be reassigned quantization levels and the quantization levels themselves evolve over time. The full-precision weights are fine-tuned during training, and discarded after training is complete.

3.2 Pruning-quantization hyperparameter prediction

The pruning-quantization operation is guided by two hyperparameters: the pruning rate p and the bit budget b. We predict a set of pruning-quantization hyperparameters $\theta_i = (p_i, b_i)$ independently for each layer i in the network using Bayesian optimization.

Bayesian optimization provides a general framework for minimizing blackbox objective functions that are typically expensive to evaluate, non-convex, may not be expressed in closed form, and may not be easily differentiable [60]. It enables an efficient search of the joint parameter space by learning from the outcomes of previous exploration. Bayesian optimization iteratively constructs a probabilistic model of the objective function while determining the most promising candidate in parameter space to evaluate next. The result of each evaluation is used to refine the model.

To guide our search for promising pruning-quantization hyperparameters, we optimize

$$\min_{\theta} \varepsilon(\theta) - \lambda \cdot c_i(\theta) \tag{1}$$

for each layer i independently, keeping the other layers uncompressed. We obtain a coarse estimate of the quality of θ by testing the compressed network on a subset of the training data; $\varepsilon(\theta)$ is the resulting top-1 error. For speed we do not perform fine-tuning of the network after applying this prospective pruning and quantization. $c_i(\theta)$ measures the overall compression benefit of applying pruning-quantization hyperparameters θ to layer i, and is computed by

$$c_i(\theta) = \frac{m_i - s_i(\theta)}{\sum_i m_i}, \tag{2}$$

where m_i is the number of bits required to store the weights of layer i in uncompressed form and $s_i(\theta)$ is the number of bits required to store the weights of layer i after pruning-quantization with θ, using the sparse encoding scheme (Section 3.1). Finally, λ is an importance weight that balances accuracy and compression.

We model the objective function as a Gaussian process [61] and select the most promising candidate in pruning-quantization hyperparameter space to evaluate next using the expected improvement criterion, which can be computed in closed form. We briefly sketch the standard solution below and refer the interested reader to more comprehensive treatments in [61], [62], [63].

A Gaussian process is an uncountable set of random variables, any finite subset of which is jointly Gaussian. It is parameterized by a mean function $\mu(\cdot)$ and a covariance kernel $k(\cdot,\cdot)$. Let

$$\begin{aligned} f &\sim \mathcal{GP}(\mu(\cdot), k(\cdot,\cdot)) \,, \\ \mu(\theta) &= \mathbb{E}\left[f(\theta)\right] \,, \\ k(\theta,\theta') &= \mathbb{E}\left[(f(\theta)-\mu(\theta))(f(\theta')-\mu(\theta'))\right] \,. \end{aligned} \tag{3}$$

Given $\Theta = \{\theta_1, \theta_2, ..., \theta_n\}$ and function evaluations $f(\Theta) = \{f(\theta_1), f(\theta_2), ..., f(\theta_n)\}$, the posterior belief of f at a novel candidate $\hat{\theta}$ is given by

$$\begin{aligned} \tilde{f}(\hat{\theta}) &\sim \mathcal{N}(\tilde{\mu}_f(\hat{\theta}), \tilde{\Sigma}_f^2(\hat{\theta}))) \,, \\ \tilde{\mu}_f(\hat{\theta}) &= \mu(\hat{\theta}) + k(\hat{\theta},\Theta)k(\Theta,\Theta)^{-1}(f(\Theta)-\mu(\Theta)), \\ \tilde{\Sigma}_f^2(\hat{\theta}) &= k(\hat{\theta},\hat{\theta}) - k(\hat{\theta},\Theta)k(\Theta,\Theta)^{-1}k(\Theta,\hat{\theta}) \,. \end{aligned} \tag{4}$$

Let θ^+ denote the best candidate evaluated so far. The expected improvement of a candidate $\hat{\theta}$ is defined as

$$\mathrm{EI}(\hat{\theta}) = \mathbb{E}[\max\{0, f(\theta^+) - \tilde{f}(\hat{\theta})\}] \,, \tag{5}$$

and it can be efficiently computed in closed form:

$$\begin{aligned} \mathrm{EI}(\hat{\theta}) &= \tilde{\Sigma}_f(\hat{\theta})(Z\Phi(Z) + \phi(Z)) \,, \\ Z &= \frac{\tilde{\mu}_f(\hat{\theta}) - f(\theta^+)}{\tilde{\Sigma}_f(\hat{\theta})} \,, \end{aligned} \tag{6}$$

where Φ is the standard normal cumulative distribution function and ϕ is the standard normal probability density function.

4 Experiments

We performed compression experiments using AlexNet [1], VGGNet [16], GoogLeNet [2], and ResNet [3] image classification networks on ImageNet (ILSVRC-2012) [64], which contains 1.2M training images and 50K validation images (Sections 4.1 to 4.4). We compared these results with state-of-the-art network compression algorithms (Section 4.5). Next, we validated our hypothesis that CLIP-Q is complementary to efficient network design by performing compression experiments using MobileNet [28] and ShuffleNet [29] (Section 4.6). Finally, we performed experiments on a spatial memory network [14] for visual question answering to investigate whether CLIP-Q generalizes beyond convolutional networks for image classification (Section 4.7).

We implemented CLIP-Q in Caffe and used the public Bayesian optimization libraries of [61], [62].

4.1 AlexNet on ImageNet

We started with Caffe's bundled ImageNet-pretrained AlexNet and trained the compressed network for 900K iterations with a batch size of 256, an initial learning rate of 0.001, and a 1/10 multiplier on the learning rate every 400K iterations. Other network learning parameters (e.g. momentum, weight decay) were kept at Caffe defaults. For Bayesian optimization, we set λ to 40 and the maximum number of iterations (i.e. candidate evaluations) to 50.

Layer	p	b	Original	Compressed	Rate
conv1	0.21	8	140 KB	35 KB	4×
conv2	0.36	6	1.2 MB	204 KB	6×
conv3	0.43	4	3.5 MB	395 KB	9×
conv4	0.32	4	2.7 MB	321 KB	8×
conv5	0.31	3	1.8 MB	174 KB	10×
fc6	0.96	3	151.0 MB	1.80 MB	84×
fc7	0.95	3	67.1 MB	969 KB	69×
fc8	0.74	3	16.4 MB	876 KB	19×
Overall			**243.9 MB**	**4.8 MB**	**51×**

TABLE 1
Layerwise compression statistics for AlexNet on ImageNet (p: pruning rate, b: bits per weight). Original top-1 accuracy: 57.2%. Compressed top-1 accuracy: 57.9%.

Table 1 shows layerwise statistics for pruning rate, quantization in bits, and storage requirements using the sparse encoding scheme [18]. Since Eq. 2 considers the whole-network compression benefit of applying candidate pruning-quantization hyperparameters, CLIP-Q learns to prioritize compressing the fully-connected layers fc6 and fc7, which have the most parameters. CLIP-Q prunes these layers the most aggressively, with close to 95% of the connections removed in both cases, and allocates the smallest number of bits to encode the remaining connections. This enables 84× compression of the fc6 layer and 69× compression of the fc7 layer. Overall, CLIP-Q compresses AlexNet from 243.9 MB to 4.8 MB – a 51× compression rate – while preserving the accuracy of the uncompressed AlexNet on ImageNet.

Fig. 3 visualizes the pruning-quantization hyperparameter prediction using Bayesian optimization. Each row shows the prediction process for one layer in AlexNet over 50 Bayesian optimization iterations. The left plot shows the value of p (pruning rate) selected by Bayesian optimization as the most promising candidate to evaluate next; the middle plot shows the value of b (bits per weight) before rounding; the right plot shows the best objective observed so far. In most cases, Bayesian optimization converges to good pruning-quantization hyperparameters within 20-30 iterations.

4.2 VGGNet on ImageNet

VGGNet stacks 3x3 convolution layers as its basic building block. We performed experiments on the ImageNet-pretrained 16-layer version (VGG-16). We reduced the batch size to 32 and kept the other learning parameters from AlexNet. For Bayesian optimization, we set λ to 80 and the maximum number of iterations (i.e. candidate evaluations) to 50.

Table 2 shows layerwise statistics for pruning rate, quantization in bits, and storage requirements using the sparse encoding scheme [18]. Similar to AlexNet, CLIP-Q learns to most aggressively compress the fully-connected layers fc6 and fc7, which have the most parameters, removing 99% and 97% of the connections, respectively. After pruning and quantization, the fc6 and fc7 layers are both compressed by two orders of magnitude. Overall, CLIP-Q compresses

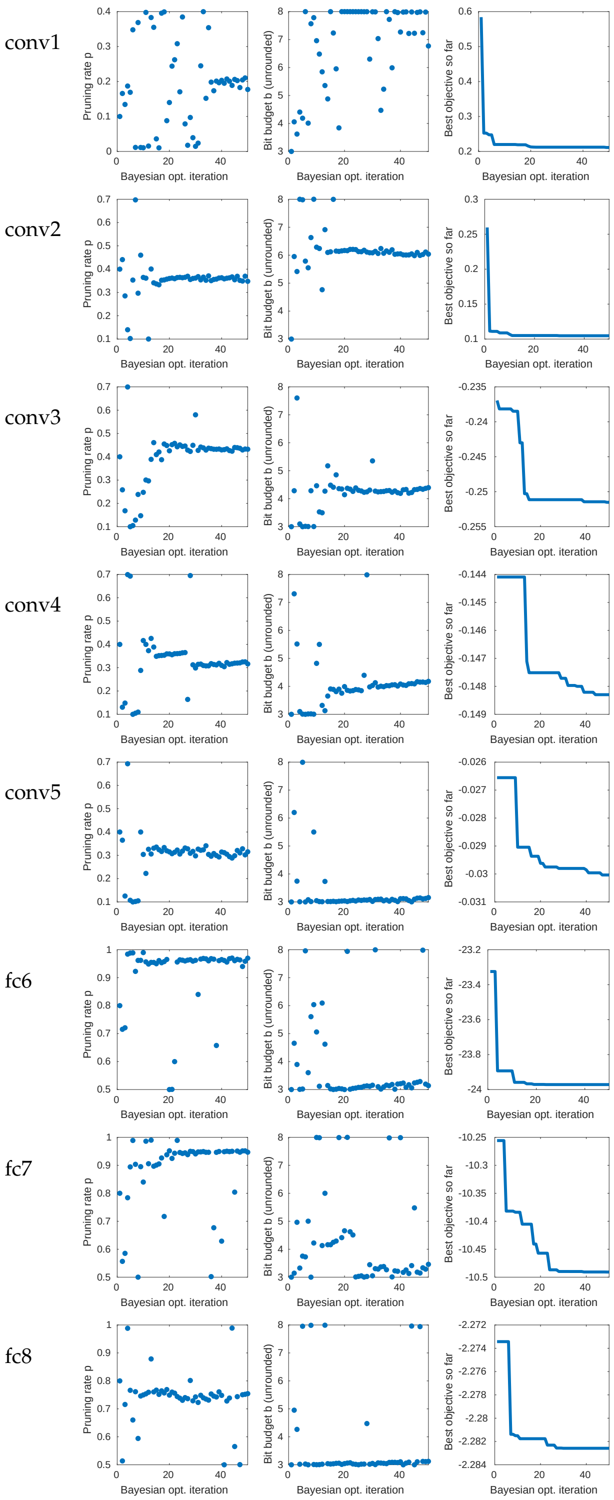


Fig. 3. Pruning-quantization hyperparameter prediction for AlexNet on ImageNet.

Layer	p	b	Original	Compressed	Rate
conv1_1	0.14	8	7 KB	3.0 KB	2×
conv1_2	0.33	6	148 KB	25.6 KB	6×
conv2_1	0.23	5	295 KB	46.7 KB	6×
conv2_2	0.24	4	590 KB	76.3 KB	8×
conv3_1	0.44	4	1.2 MB	133 KB	9×
conv3_2	0.62	3	2.4 MB	175 KB	13×
conv3_3	0.53	3	2.4 MB	197 KB	12×
conv4_1	0.62	3	4.7 MB	345 KB	14×
conv4_2	0.73	3	9.4 MB	550 KB	17×
conv4_3	0.58	3	9.4 MB	735 KB	13×
conv5_1	0.65	3	9.4 MB	663 KB	14×
conv5_2	0.61	3	9.4 MB	710 KB	13×
conv5_3	0.65	3	9.4 MB	689 KB	14×
fc6	0.99	3	411.1 MB	1.80 MB	228×
fc7	0.97	3	67.1 MB	655 KB	102×
fc8	0.76	3	16.4 MB	840 KB	20×
Overall			**553.4 MB**	**7.6 MB**	**72×**

TABLE 2
Layerwise compression statistics for VGG-16 on ImageNet (p: pruning rate, b: bits per weight). Original top-1 accuracy: 68.5%. Compressed top-1 accuracy: 69.2%.

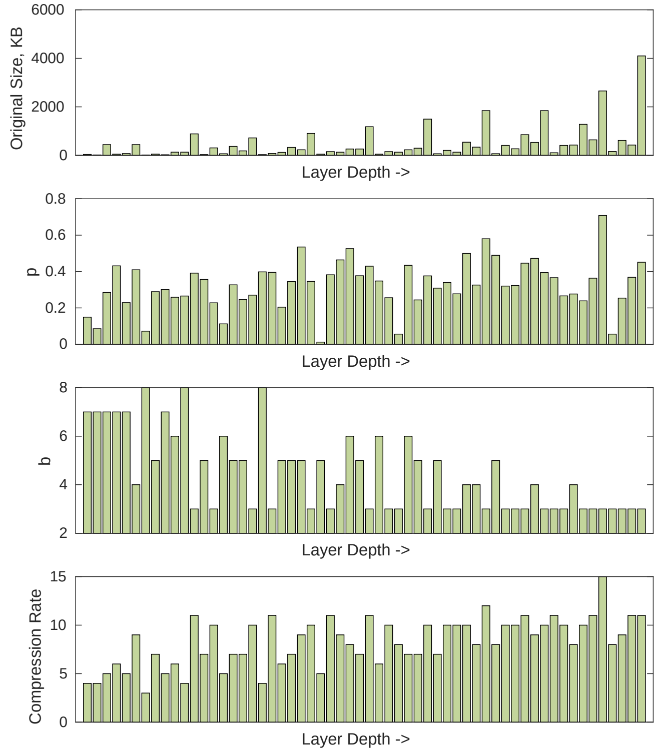


Fig. 4. Layerwise compression statistics for GoogLeNet on ImageNet. Overall compression from 28.0 MB to 2.8 MB (10× compression). Original top-1 accuracy: 68.9%. Compressed top-1 accuracy: 68.9%.

VGG-16 from 553.4 MB to 7.6 MB while maintaining the accuracy of the uncompressed network.

4.3 GoogLeNet on ImageNet

We compressed GoogLeNet using the default network learning parameters of the quick-solver bundled with Caffe's distribution of GoogLeNet, except that we initialized the learning rate to 1e-5. For Bayesian optimization, we set λ to 5 and the cutoff number of iterations (i.e. number of candidate evaluations) to 50. We excluded the auxiliary branches of the network from pruning-quantization since these are discarded after training.

Since GoogLeNet contains a large number of layers, for easier readability we summarize the layerwise statistics in Fig. 4, which visualizes (from top to bottom) the original layer size in KB, the pruning rate p, the quantization in bits b, and the achieved compression rate. Similar to AlexNet and VGGNet, CLIP-Q learns to prioritize the later layers

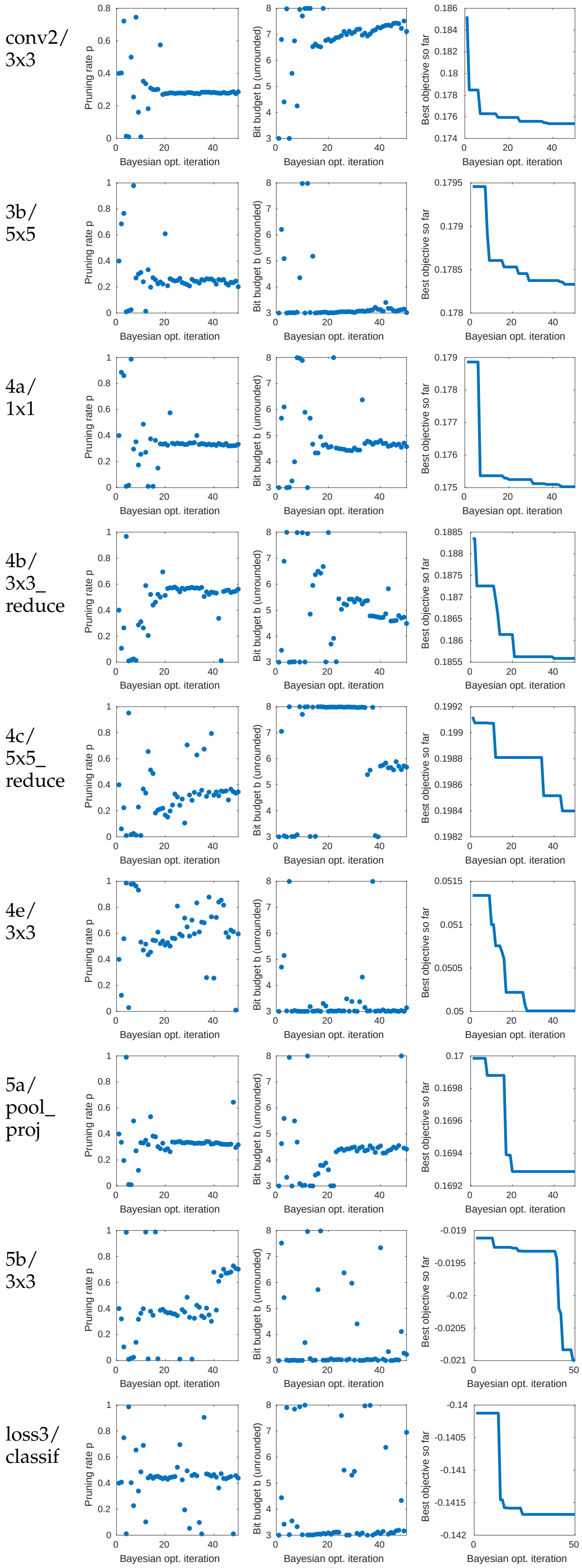


Fig. 5. Pruning-quantization hyperparameter prediction for a sample of GoogleNet layers.

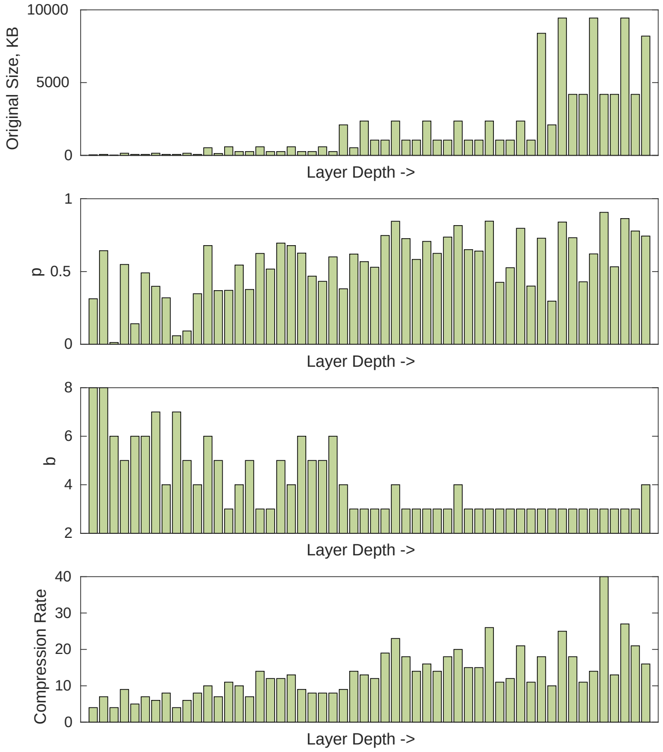


Fig. 6. Layerwise compression statistics for ResNet-50 on ImageNet. Overall compression from 102.5 MB to 6.7 MB (15× compression). Original top-1 accuracy: 73.1%. Compressed top-1 accuracy: 73.7%.

as these contain more parameters. The most aggressively compressed layer is inception_5b/3x3, in which 71% of the connections are pruned and the remaining weights are quantized to 3 bits, resulting in a compression rate of 15×. Overall, CLIP-Q compresses GoogLeNet from 28.0 MB to 2.8 MB, or 10×, while matching the accuracy of the uncompressed network. Efficient use of weights built into the GoogLeNet architecture (e.g. inception modules with low-dimensional embeddings) may explain the more modest compression result compared to AlexNet and VGGNet.

Fig. 5 visualizes the pruning-quantization hyperparameter prediction for a small sample of GoogLeNet layers. Similar to AlexNet and VGGNet, Bayesian optimization usually converges to good pruning-quantization hyperparameters within 20-30 iterations; occasionally, such as for inception_5b/3x3, the best solution is obtained closer to the 50-iteration cutoff.

4.4 ResNet on ImageNet

We performed experiments with ResNet-50 on ImageNet. We trained the compressed network for 250K iterations with a batch size of 24, initial learning rate of 1e-4, and a 1/10 multiplier on the learning rate after 100K iterations. For Bayesian optimization, we set λ to 10 and the maximum number of iterations (i.e. candidate evaluations) to 50.

Fig. 6 summarizes the layerwise statistics. Similar to the preceding experiments, CLIP-Q learns to prioritize the later layers containing more parameters, assigning these higher pruning rates and fewer bits. Overall, CLIP-Q compresses

	Δ Accuracy	Network Size
AlexNet on ImageNet		
Uncompressed	–	243.9 MB
Data-Free Pruning [32] (CaffeNet)	-2.2%	159 MB
Deep Fried Convnets [50]	-0.3%	68 MB
Less Is More [65]	-0.6%	57 MB
Dynamic Network Surgery [17]	-0.2% [1]	13.8 MB
Circulant CNN [49]	-0.4%	12.7 MB
Quantized CNN [43]	-1.4%	12.6 MB
Binary-Weight-Networks [45]	+0.1%	7.6 MB
Deep Compression [18]	+0.0%	8.9 (6.9) MB
[18] + Weighted-Entropy Quantization [42]	-0.8%	8.3 (6.5) MB
CLIP-Q	+0.7%	**4.8 MB**
VGG-16 on ImageNet		
Uncompressed	–	553.4 MB
ThiNet-GAP [35]	-1.0%	33.3 MB
Quantized CNN [43]	-1.4%	27.2 MB
Deep Compression [18]	-0.3%	17.7 (11.3) MB
CLIP-Q	+0.7%	**7.6 MB**
GoogLeNet on ImageNet		
Uncompressed	–	28.0 MB
Weighted-Entropy Quantization [42]	+0.2%	4.4 MB
CLIP-Q	+0.0%	**2.8 MB**
ResNet-50 on ImageNet		
Uncompressed	–	102.5 MB
ThiNet [35]	-1.9%	49.5 MB
Weighted-Entropy Quantization [42]	-1.8%	16.0 MB
CLIP-Q	+0.6%	**6.7 MB**

TABLE 3
Network compression performance compared with state-of-the-art algorithms for AlexNet, VGG-16, GoogLeNet, and ResNet-50 on ImageNet

ResNet-50 from 102.5 MB to 6.7 MB, or a 15× compression rate, while preserving the accuracy of the uncompressed network on ImageNet.

4.5 Comparison to state-of-the-art methods

Table 3 shows a comparison of CLIP-Q with state-of-the-art network compression algorithms. "Δ Accuracy" refers to the change in network accuracy after compression. We report deltas to fairly treat small variations in training across papers that can produce different uncompressed networks as starting points. For deep compression [18], the bracketed numbers are after post-processing by Huffman coding, a lossless data compression technique.

Without any post-processing, CLIP-Q produces the smallest compressed AlexNet model at 4.8 MB. This result extends the previous best compressed AlexNet result, Deep Compression + Weighted-Entropy Quantization, by 1.7 MB while obtaining 1.5% higher accuracy; this represents an improvement on the previous best compression rate (37.5×) by 35% relative. On VGG-16, CLIP-Q obtains a state-of-the-art compressed network size of 7.6 MB, a 48% relative improvement with respect to the previous best reported compression result. On GoogLeNet, CLIP-Q obtains a state-of-the-art compressed network size of 2.8 MB, improving on the previous best compression rate by 57% relative. On ResNet-50, CLIP-Q produces a compressed model size of 6.7 MB, or a 139% relative improvement in the state-of-the-art compression rate, while obtaining 2.4% higher accuracy.

4.6 Efficient Architectures on ImageNet

We next experimentally verify that pruning and quantization are complementary to efficient architecture design by compressing the highly efficient MobileNet [28] and ShuffleNet [29] architectures. MobileNet is designed for embedded applications and stacks several layers of depthwise separable convolutions. A depthwise separable convolution consists of depthwise convolution – spatial convolution filters applied to single input channels – followed by pointwise convolution – 1x1 filters applied across all input channels. This factorization significantly reduces the computational cost compared to conventional convolution at only a small cost in accuracy [28]. ShuffleNet employs depthwise convolutions and pointwise group convolutions – group convolutions on 1x1 filters – and introduces a channel "shuffle" operation.

To reuse our existing implementation, we compressed the pointwise convolution filters, regular convolution filters, and fully connected layers, and omitted the depthwise convolution filters; the depthwise filters contribute 1% of the network parameters in both MobileNet and ShuffleNet. We compressed MobileNet for 100K iterations with a batch size of 64, initial learning rate of 1e-5, and a 1/10 multiplier on the learning rate after 40K iterations. We used RMSProp following [28]. For Bayesian optimization, we set λ to 5 and the maximum number of iterations (candidate evaluations) to 50. For ShuffleNet, we followed the training parameters in [29] except that we reduced the initial learning rate to 1e-4 and used 5% of the total epochs; we used a batch size of 64. For Bayesian optimization, we set λ to 2 and the maximum number of iterations (candidate evaluations) to 50.

Fig. 7 summarizes the layerwise statistics for MobileNet. CLIP-Q compresses the conv6/step and fc7 layers, which have the most parameters, the most aggressively. Overall, CLIP-Q compresses the already efficient MobileNet from 17.0 MB to 2.2 MB, or a 7.6× compression rate, while preserving the accuracy of the uncompressed MobileNet on ImageNet. Fig. 8 summarizes the layerwise statistics for ShuffleNet. CLIP-Q compresses ShuffleNet from 7.4 MB to 1.1 MB, or a 6.6× compression rate, while preserving the accuracy of the uncompressed network on ImageNet. These results demonstrate that CLIP-Q is complementary to advances in efficient network design, and can be used in coordination with efficient design to achieve even more compact networks that achieve the same accuracy.

Fig. 9 visualizes the pruning-quantization hyperparameter prediction for sample MobileNet layers. Bayesian optimization converges to good hyperparameters within 20-30 iterations in most cases, similar to AlexNet, VGG-16, GoogLeNet, and ResNet-50.

Table 4 provides a comparison with state-of-the-art network compression algorithms. On MobileNet, CLIP-Q achieves a substantially higher compression rate compared with [66] and [67], and a 9% relative improvement with respect to weighted-entropy quantization [42]. On ShuffleNet,

[1] estimated using Caffe AlexNet model accuracy

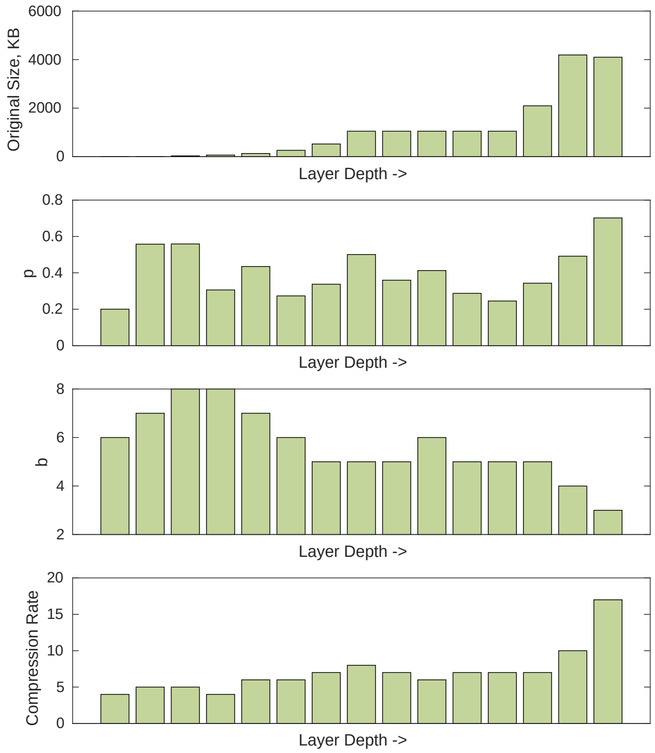


Fig. 7. Layerwise compression statistics for MobileNet on ImageNet. Overall compression from 17.0 MB to 2.2 MB (7.6× compression). Original top-1 accuracy: 70.3%. Compressed top-1 accuracy: 70.3%.

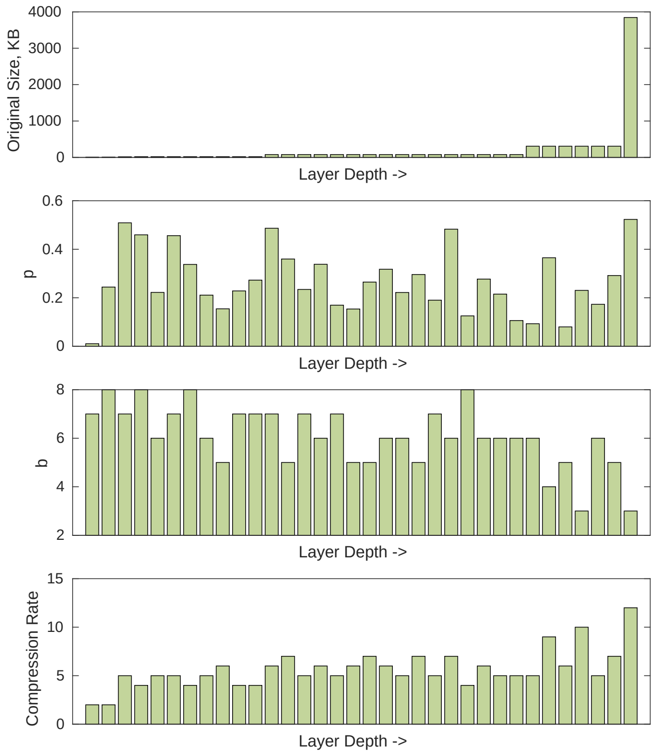


Fig. 8. Layerwise compression statistics for ShuffleNet on ImageNet. Overall compression from 7.4 MB to 1.1 MB (6.6× compression). Original top-1 accuracy: 65.3%. Compressed top-1 accuracy: 65.4%.

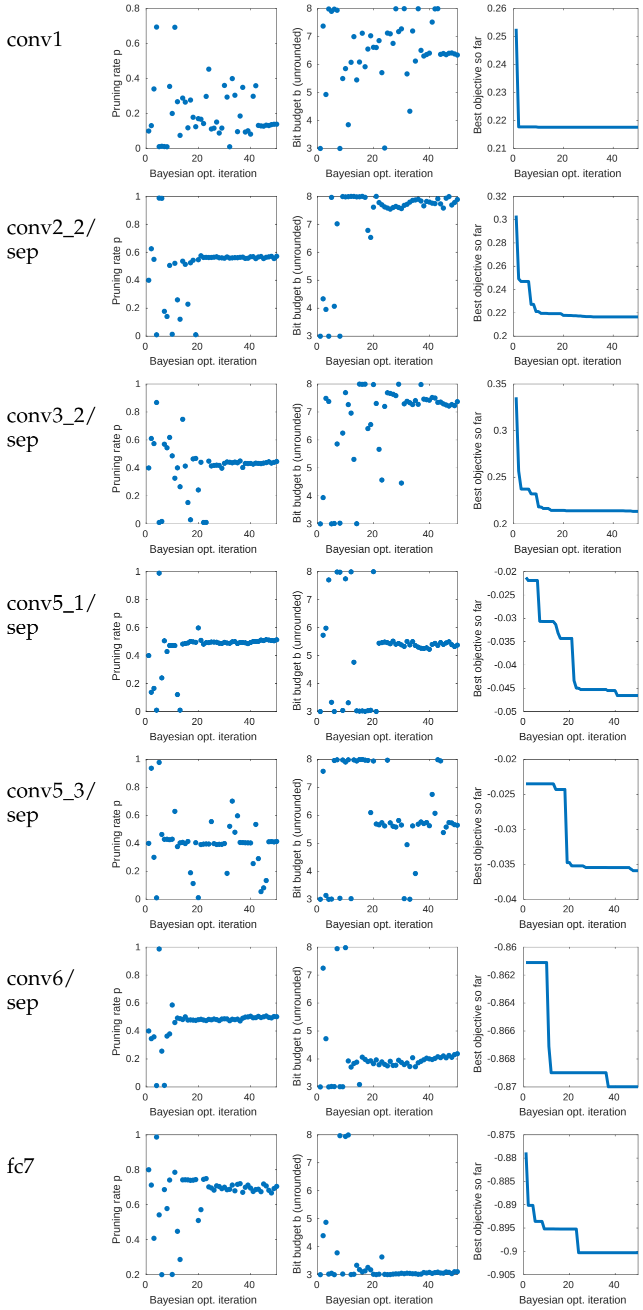


Fig. 9. Pruning-quantization hyperparameter prediction for a sample of MobileNet layers.

CLIP-Q obtains a 23% relative improvement in compression rate with respect to weighted-entropy quantization, with 1% higher accuracy.

Fig. 10 illustrates the network size-accuracy tradeoff obtained by varying λ for MobileNet and ShuffleNet. The other compression results reported in this paper are included for reference. The figure indicates that the compressed MobileNet and ShuffleNet series, as well as compressed ResNet-50, offer the best size-accuracy tradeoff among the tested networks. The MobileNet series was generated by $\lambda = \{0.5, 1, 5, 10, 20\}$ and the ShuffleNet series by $\lambda = \{1, 2, 5, 10, 20\}$.

	Δ Accuracy	Network Size
MobileNet on ImageNet		
Uncompressed	–	17.0 MB
To Prune or Not To Prune [66]	-1.1%	8.5 MB
Ristretto [67]	-0.2%	4.5 MB
Weighted-Entropy Quantization [42]	-0.3%	2.4 MB
CLIP-Q	+0.0%	**2.2 MB**
ShuffleNet on ImageNet		
Uncompressed	–	7.4 MB
Weighted-Entropy Quantization [42]	-0.9%	1.4 MB
CLIP-Q	+0.1%	**1.1 MB**

TABLE 4
Network compression performance compared with state-of-the-art algorithms for MobileNet and ShuffleNet on ImageNet

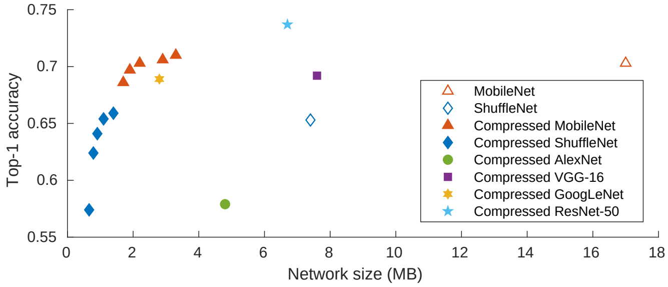


Fig. 10. Network size-accuracy tradeoff for MobileNet and ShuffleNet. The original networks as well as the other compression results reported in this paper are included for reference.

4.7 Spatial Memory Network on VQA

To demonstrate that CLIP-Q generalizes beyond convolutional networks for image classification, we performed experiments using the Spatial Memory Network [14] for visual question answering (VQA). Given an image and a natural language question about the image, the goal of a VQA system is to produce an appropriate natural language answer. An example is shown in Fig. 11. The task integrates computer vision, natural language processing, and logical reasoning, and is sometimes considered to be "AI-complete" or a proxy to evaluating a system for deep image understanding [68], [69]. The SMem-VQA network employs recurrent units that allow the network to sequentially attend to spatial regions that are relevant in answering the question.

We performed compression experiments on the VQA dataset [68] using the public evaluation server. The VQA dataset includes 614,163 questions for 204,721 images from the Microsoft COCO (Common Objects in Context) dataset [70], and follows MSCOCO's training, validation, and testing splits. We adopted SMem-VQA's default learning parameters, except that we reduced the initial learning rate and iterations by a factor of 10. For Bayesian optimization, we set λ to 1 and the cutoff number of iterations to 30. In addition to convolution and fully-connected layers, the SMem-VQA network includes embedding layers, which are not found in the image classification networks previously studied. We pruned and quantized these novel layers in the same way as the convolution and fully-connected layers.

What brand of peanut butter is in the photo?
Is the fruit ripe?

Fig. 11. In visual question answering, the system is required to answer natural-language questions about an image. Source: VQA dataset [68]

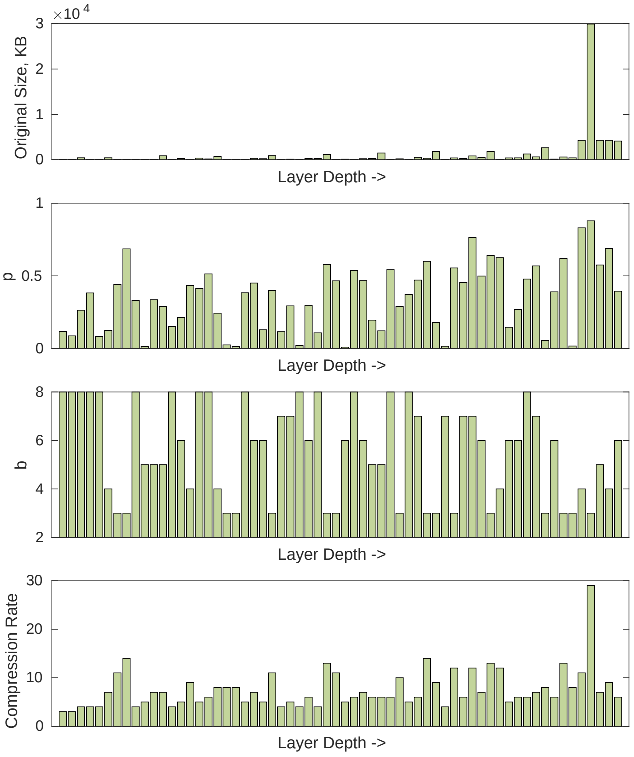


Fig. 12. Layerwise compression statistics for SMem-VQA on VQA dataset, test-dev split. Original top-1 accuracy: 57.99%. Compressed top-1 accuracy: 57.04%.

Fig. 12 summarizes the layerwise statistics. The largest embedding layer, containing close to 30 MB in parameters, is compressed the most aggressively, at a 29× compression rate. Overall, CLIP-Q compresses SMem-VQA from 70.8 MB to 6.4 MB, achieving a 11× compression rate, with a 0.95% drop in answer accuracy. For comparison, weighted-entropy quantization [42] compresses SMem-VQA to 9.4 MB, with a 1.02% drop in answer accuracy.

5 Conclusion

We have presented a new method for deep network compression called CLIP-Q that combines weight pruning and quantization in a single learning framework, makes flexible pruning and quantization decisions that adapt to the changing network structure over time, and performs pruning and quantization in parallel with network fine-tuning. CLIP-Q obtains state-of-the-art compression rates of 51× for AlexNet, 72× for VGG-16, 10× for GoogLeNet, and 15× for ResNet-50, improving upon previously reported compression rates by 35%, 48%, 57%, and 139% relative, respectively.

We have also shown that CLIP-Q is complementary to efficient network architecture design by compresing MobileNet by nearly 8× and ShuffleNet by nearly 7×, in both cases without loss in accuracy. Finally, CLIP-Q generalizes beyond convolutional networks and obtains a 11× compression rate for a spatial memory network for visual question answering with less than 1% drop in accuracy.

Limitations and future work. We have focused on compression performance in this work because practical test-time speed-up is dependent on the software implementation of basic network operations and on hardware details. For example, the way generalized convolution is implemented in many modern deep learning frameworks has inspired acceleration methods based on interpolation at test time or structured pruning at training time (see Section 2.1). Practical acceleration on mobile or embedded devices will require careful consideration of these implementation details.

Smaller network models are not necessarily more energy efficient because energy consumption is also determined by the pattern of memory accesses [40]. Making deep networks more accessible to low-power devices will require us to consider the energy efficiency of network structures. It will be interesting to see whether our purning-quantization hyperparameter prediction framework, which currently considers compression rate and accuracy, can incorporate an estimate of energy consumption using hardware models.

ACKNOWLEDGMENTS

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

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[4] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object detection via region-based fully convolutional networks," in *Advances in Neural Information Processing Systems*, 2016.

[5] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," in *European Conference on Computer Vision*, 2016.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[7] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, , and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFs," in *International Conference on Learning Representations*, 2015.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[9] T. Pohlen, A. Hermans, and M. Mathias, "Full-resolution residual networks for semantic segmentation in street scenes," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[10] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," in *International Conference on Learning Representations*, 2016.

[11] R. Hu, J. Andreas, M. Rohrbach, T. Darrell, and K. Saenko, "Learning to reason: End-to-end module networks for visual question answering," in *IEEE International Conference on Computer Vision*, 2017.

[12] M. Malinowski, M. Rohrbach, and M. Fritz, "Ask your neurons: A neural-based approach to answering questions about images," in *IEEE International Conference on Computer Vision*, 2015.

[13] H. Noh, P. H. Seo, and B. Han, "Image question answering using convolutional neural network with dynamic parameter prediction," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[14] H. Xu and K. Saenko, "Ask, attend and answer: Exploring question-guided spatial attention for visual question answering," in *European Conference on Computer Vision*, 2016.

[15] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *International Conference on Learning Representations*, 2015.

[17] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Advances in Neural Information Processing Systems*, 2016.

[18] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding," in *International Conference on Learning Representations*, 2016.

[19] F. Tung, S. Muralidharan, and G. Mori, "Fine-pruning: Joint fine-tuning and compression of a convolutional network with Bayesian optimization," in *British Machine Vision Conference*, 2017.

[20] X. Dong, J. Huang, Y. Yang, and S. Yan, "More is less: a more complicated network with less inference complexity," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[21] M. Figurnov, M. D. Collins, Y. Zhu, L. Zhang, J. Huang, D. Vetrov, and R. Salakhutdinov, "Spatially adaptive computation time for residual networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[22] M. Figurnov, A. Ibraimova, D. Vetrov, and P. Kohli, "Perforated-CNNs: acceleration through elimination of redundant convolutions," in *Advances in Neural Information Processing Systems*, 2016.

[23] X. Li, Z. Liu, P. Luo, C. C. Loy, and X. Tang, "Not all pixels are equal: Difficulty-aware semantic segmentation via deep layer cascade," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[24] B. Chang, L. Meng, E. Haber, F. Tung, and D. Begert, "Multi-level residual networks from dynamical systems view," in *International Conference on Learning Representations*, 2018.

[25] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *European Conference on Computer Vision*, 2016.

[26] M. Denil, B. Shakibi, L. Dinh, M. Ranzato, and N. de Freitas, "Predicting parameters in deep learning," in *Advances in Neural Information Processing Systems*, 2013.

[27] F. Tung and G. Mori, "CLIP-Q: Deep network compression learning by in-parallel pruning-quantization," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv:1704.04861, 2017.

[29] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," arXiv:1707.01083, 2017.

[30] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 1992.

[31] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 1990.

[32] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in *British Machine Vision Conference*, 2015.

[33] V. Lebedev and V. Lempitsky, "Fast ConvNets using group-wise brain damage," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[34] 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*, 2016.

[35] J.-H. Luo, J. Wu, and W. Lin, "ThiNet: A filter level pruning method for deep neural network compression," in *IEEE International Conference on Computer Vision*, 2017.

[36] J. Park, S. Li, W. Wen, P. Tang, H. Li, Y. Chen, and P. Dubey, "Faster CNNs with direct sparse convolutions and guided pruning," in *International Conference on Learning Representations*, 2017.

[37] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: Efficient inference engine on compressed deep neural network," in *ACM/IEEE International Symposium on Computer Architecture*, 2016.

[38] S. Han, H. Mao, E. Gong, S. Tang, W. J. Dally, J. Pool, J. Tran, B. Catanzaro, S. Narang, E. Elsen, P. Vajda, and M. Paluri, "DSD: Dense-sparse-dense training for deep neural networks," in *International Conference on Learning Representations*, 2017.

[39] J. Liu, Y. Wang, and Y. Qiao, "Sparse deep transfer learning for convolutional neural network," in *AAAI Conference on Artificial Intelligence*, 2017.

[40] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[41] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in *International Conference on Learning Representations*, 2017.

[42] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[43] J. Wu, C. Leng, Y. Wang, Q. Hu, and J. Cheng, "Quantized convolutional neural networks for mobile devices," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[44] 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*, 2015.

[45] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, 2016.

[46] Y. Guo, A. Yao, H. Zhao, and Y. Chen, "Network sketching: exploiting binary structure in deep CNNs," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[47] F. Xu, V. N. Boddeti, and M. Savvides, "Local binary convolutional neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[48] T. Ojala, M. Pietikainen, and D. Harwood, "A comparative study of texture measures with classification based on feature distributions," *Pattern Recognition*, vol. 29, no. 1, pp. 51–59, 1996.

[49] Y. Cheng, F. X. Yu, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang, "An exploration of parameter redundancy in deep networks with circulant projections," in *IEEE International Conference on Computer Vision*, 2015.

[50] Z. Yang, M. Moczulski, M. Denil, N. de Freitas, A. Smola, L. Song, and Z. Wang, "Deep fried convnets," in *IEEE International Conference on Computer Vision*, 2015.

[51] E. 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*, 2014.

[52] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank expansions," in *British Machine Vision Conference*, 2014.

[53] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun, "Efficient and accurate approximations of nonlinear convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015.

[54] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," arXiv:1503.02531, 2015.

[55] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "FitNets: hints for thin deep nets," in *International Conference on Learning Representations*, 2015.

[56] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2N learning: Network to network compression via policy gradient reinforcement learning," in *International Conference on Learning Representations*, 2018.

[57] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5mb model size," arXiv:1602.07360, 2016.

[58] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[59] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Weinberger, "Multi-scale dense networks for resource efficient image classification," in *International Conference on Learning Representations*, 2018.

[60] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas, "Bayesian optimization in high dimensions via random embeddings," in *International Joint Conference on Artificial Intelligence*, 2013.

[61] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[62] J. R. Gardner, M. J. Kusner, Z. Xu, K. Q. Weinberger, and J. P. Cunningham, "Bayesian optimization with inequality constraints," in *International Conference on Machine Learning*, 2014.

[63] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems*, 2012.

[64] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," arXiv:1409.0575, 2014.

[65] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: towards compact CNNs," in *European Conference on Computer Vision*, 2016.

[66] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression," in *International Conference on Learning Representations Workshop*, 2018.

[67] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, in press.

[68] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh, "VQA: Visual question answering," in *IEEE International Conference on Computer Vision*, 2015.

[69] Y. Zhu, O. Groth, M. Bernstein, and L. Fei-Fei, "Visual7W: Grounded question answering in images," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[70] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context," in *European Conference on Computer Vision (ECCV)*, 2014.

Frederick Tung received the Ph.D. degree in computer science from the University of British Columbia in 2017 and is currently a postdoctoral fellow at Simon Fraser University. His research interests are in computer vision and machine learning, with a focus on developing algorithms for training resource-efficient deep neural networks.

Greg Mori was born in Vancouver and grew up in Richmond, BC. He received the Ph.D. degree in Computer Science from the University of California, Berkeley in 2004. He received an Hon. B.Sc. in Computer Science and Mathematics with High Distinction from the University of Toronto in 1999. He spent one year (1997-1998) as an intern at Advanced Telecommunications Research (ATR) in Kyoto, Japan. After graduating from Berkeley, he returned home to Vancouver and is currently a professor in the School of Computing Science at Simon Fraser University. Dr. Mori's research interests are in computer vision, and include object recognition, human activity recognition, human body pose estimation. He serves on the program committee of major computer vision conferences (CVPR, ECCV, ICCV), and was the program co-chair of the Canadian Conference on Computer and Robot Vision (CRV) in 2006 and 2007. Dr. Mori received the Excellence in Undergraduate Teaching Award from the SFU Computing Science Student Society in 2006. Dr. Mori received the Canadian Image Processing and Pattern Recognition Society (CIPPRS) Award for Research Excellence and Service in 2008. Dr. Mori received NSERC Discovery Accelerator Supplement awards in 2008 and 2016.