CLIP-Q: Deep Network Compression Learning by In-Parallel Pruning-Quantization

Frederick Tung Greg Mori
Simon Fraser University
ftung@sfu.ca, mori@cs.sfu.ca

Abstract

Deep neural networks enable state-of-the-art accuracy on visual recognition tasks such as image classification and object detection. However, modern deep networks contain millions of learned weights; a more efficient utilization of computation resources would assist in a variety of deployment scenarios, from embedded platforms with resource constraints to computing clusters running ensembles of networks. In this paper, we combine network pruning and weight quantization in a single learning framework that performs pruning and quantization jointly, and in parallel with fine-tuning. This allows us to take advantage of the complementary nature of pruning and quantization and to recover from premature pruning errors, which is not possible with current two-stage approaches. Our proposed CLIP-Q method (Compression Learning by In-Parallel Pruning-Quantization) compresses AlexNet by 51-fold, GoogLeNet by 10-fold, and ResNet-50 by 15-fold, while preserving the uncompressed network accuracies on ImageNet.

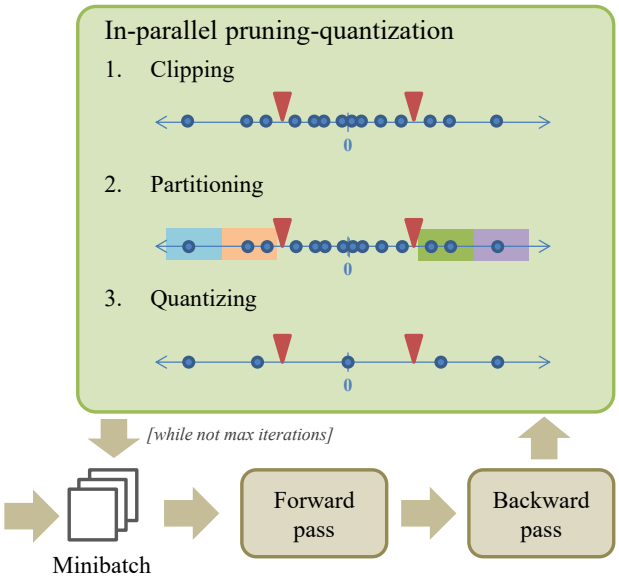

Figure 1. An overview of our deep network compression approach. CLIP-Q combines weight pruning and quantization in a single learning framework, and performs pruning and quantization in parallel with fine-tuning. The joint pruning-quantization adapts over time with the changing network.

1. Introduction

Deep neural networks have become indispensable tools for a wide range of visual recognition tasks, such as image classification [19, 27, 48], object detection [6, 32, 42], semantic segmentation [2, 33, 39, 58], and visual question answering [21, 35, 36, 56]. 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 [19, 23, 45]. 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. More generally, 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?

Our focus in this paper is on deep network compression, which has the goal of making deep networks more compact [13, 16, 49]. Complementary lines of work focus on accelerating deep network inference at test time [9, 10, 11, 30], or reducing deep network training times [1, 24]. 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 [7]. Network pruning, in which the network connections are reduced or sparsified, is one common strategy for compression. Another common strategy is weight quantization, in which connection weights are constrained to a set of discrete values, allowing weights to be represented using fewer bits. However, current approaches perform pruning and quantization separately. This does not exploit the complementary nature of weight pruning and quantization, and errors made in the first stage cannot be corrected in the second stage.

This paper presents CLIP-Q (Compression Learning by In-Parallel Pruning-Quantization), a new 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 AlexNet [27], GoogLeNet [48], and ResNet [19] for ImageNet classification demonstrate the potential of the proposed approach.

2. Related Work

2.1. Network 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 [18, 29]. Data-free parameter pruning [47] discovers and removes redundant neurons using a data-independent technique. Deep compression [16] 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 [13]. Dynamic network surgery [13] 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.

Besides compression, network pruning has also been used to regularize the training of more accurate full-sized networks [17], reduce over-fitting in transfer learning [31], and produce energy-efficient networks for battery-powered devices [55].

2.2. Weight quantization

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 [16] 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 [50] 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 [50]. Weighted-entropy-based quantization [38] 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 [53] 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 [16, 38, 50] instead of product or vector quantization.

In the limit, weights can be quantized to a single bit to form binary weight networks [5, 41]. Network sketching [14] binarizes pre-trained networks by approximating real-valued filters with weighted combinations of binary filters. Local binary convolutional networks [54] replace convolutional layers with a learnable module inspired by local binary patterns [37].

2.3. Distillation, low-rank approximation, and structured projections

In addition to network pruning and weight quantization, several other directions have been successfully applied to train or fine-tune compact deep networks. Knowledge dis-

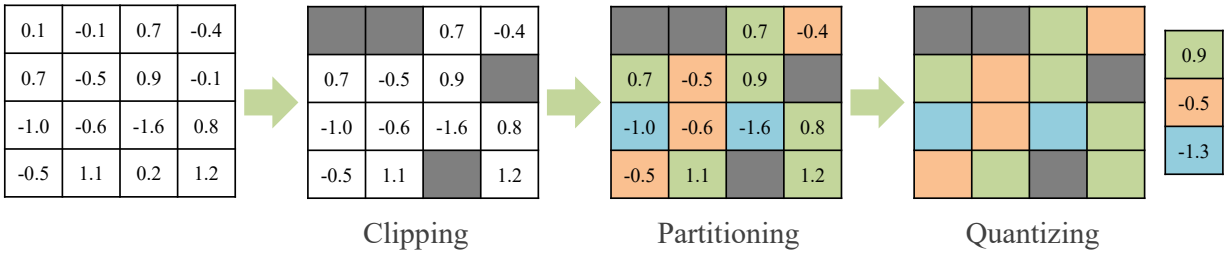

Figure 2. An example illustrating the three steps of the pruning-quantization operation for a layer with 16 weights, $p = 0.25$ and $b = 2$. Pruning-quantization is performed in parallel with fine-tuning the network's full-precision weights, and updates the pruning statuses, quantization levels, and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

tillation [20, 43] trains a compact student network using a weighted combination of ground truth labels and the soft outputs of a larger, more expensive teacher network. Low-rank approximation methods exploit the redundancy in filters and feature map responses [8, 26, 59]. Structured projection methods [3, 57] 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.

2.4. Efficient architectures

An orthogonal direction is the design of efficient network architectures. SqueezeNet [25] replaces 3x3 convolutional filters with 1x1 filters, reduces the number of input channels to 3x3 filters, and downsamples later in the network. Xception [4] replaces Inception modules [48] with depthwise separable convolutions. Multi-scale DenseNet [22] uses early-exit classifiers to enable anytime classification and budgeted batch classification. Pruning and quantization are complementary to these architectural innovations; for example, deep compression [16] is applied to compress SqueezeNet by 10-fold in [25].

2.5. 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 [11] 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 [9] predict which spatial locations will produce negative responses and skips those locations. Spatially adaptive computation time [10] accelerates residual networks by learning to predict halting scores that allow computation to stop early within a block of residual units.

3. CLIP-Q

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 [16] to the left of c^- and to the right of c^+ for simplicity; other partitioning strategies (e.g. weighted entropy [38]) 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.

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 backpropagation [38, 41], while the forward pass uses the quantized weights, simulating the output of the compressed network. A summary of the training process is shown in Algorithm 1. 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 [16, 50]. 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 [16].

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.

Algorithm 1 Training the compressed network

Input: Full-precision layer weights $\mathbf{w}_1 ... \mathbf{w}_L$, pruning-quantization hyperparameters $\{p_1 ... p_L, b_1 ... b_L\}$ precomputed 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

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 $\theta_i = (p_i, b_i)$ 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 [51]. 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. 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 using 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 [40] 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 [12, 40, 46].

A Gaussian process 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 [27], GoogLeNet [48], and ResNet [19] image classification networks on ImageNet (ILSVRC-2012) [44], which contains 1.2M training images and 50K validation images. We implemented CLIP-Q in Caffe and used the public Bayesian optimization libraries of [12, 40].

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%.

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.

Table 1 shows layerwise statistics for pruning rate, quantization in bits, and storage requirements using the sparse encoding scheme [16]. 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.

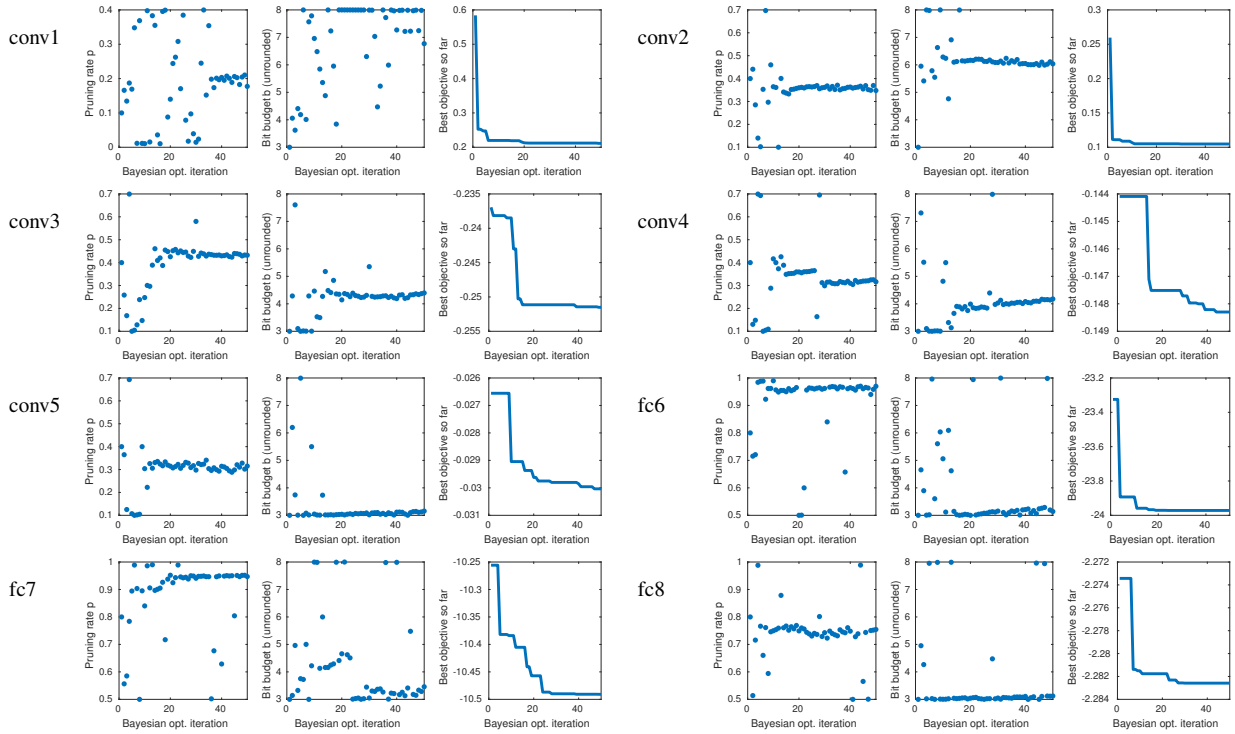

Figure 3. Pruning-quantization hyperparameter prediction for AlexNet on ImageNet.

4.2. 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.

Table 2 shows the full layerwise statistics for pruning rate, quantization in bits, and storage requirements using the sparse encoding scheme [16]. Similar to AlexNet, CLIP-Q learns to prioritize the later layers 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.

Similar to AlexNet, Bayesian optimization usually converges to good pruning-quantization hyperparameters within 20-30 iterations; we omit the visualization due to space constraints.

4.3. ResNet on ImageNet

Finally, we performed experiments with ResNet-50. 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.

Table 3 shows the full layerwise statistics. Similar to the AlexNet and GoogLeNet experiments, CLIP-Q learns to prioritize the later layers containing more parameters. Overall, CLIP-Q compresses 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.

4.4. Comparison to state-of-the-art methods

Table 4 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 [16], 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 re-

Layer	p	b	Original	Compressed	Rate
conv1/7x7_s2	0.15	7	38 KB	9.1 KB	4×
conv2/3x3_reduce	0.09	7	17 KB	4.5 KB	4×
conv2/3x3	0.28	7	443 KB	88 KB	5×
inception_3a/1x1	0.43	7	49 KB	8.9 KB	6×
inception_3a/3x3_reduce	0.23	7	74 KB	16 KB	5×
inception_3a/3x3	0.41	4	443 KB	50 KB	9×
inception_3a/5x5_reduce	0.07	8	12 KB	4.3 KB	3×
inception_3a/5x5	0.29	5	51 KB	7.8 KB	7×
inception_3a/pool_proj	0.30	7	25 KB	5.4 KB	5×
inception_3b/1x1	0.26	6	132 KB	24 KB	6×
inception_3b/3x3_reduce	0.27	8	132 KB	31 KB	4×
inception_3b/3x3	0.39	3	886 KB	81 KB	11×
inception_3b/5x5_reduce	0.36	5	33 KB	4.8 KB	7×
inception_3b/5x5	0.23	3	308 KB	32 KB	10×
inception_3b/pool_proj	0.11	6	66 KB	13 KB	5×
inception_4a/1x1	0.33	5	369 KB	53 KB	7×
inception_4a/3x3_reduce	0.25	5	185 KB	28 KB	7×
inception_4a/3x3	0.27	3	720 KB	72 KB	10×
inception_4a/5x5_reduce	0.40	8	31 KB	7.0 KB	4×
inception_4a/5x5	0.40	3	77 KB	7.3 KB	11×
inception_4a/pool_proj	0.20	5	123 KB	20 KB	6×
inception_4b/1x1	0.34	5	328 KB	47 KB	7×
inception_4b/3x3_reduce	0.53	5	230 KB	26 KB	9×
inception_4b/3x3	0.35	3	904 KB	86 KB	10×
inception_4b/5x5_reduce	0.01	5	49 KB	9.3 KB	5×
inception_4b/5x5	0.38	3	154 KB	15 KB	11×
inception_4b/pool_proj	0.46	4	131 KB	14 KB	9×
inception_4c/1x1	0.53	6	263 KB	35 KB	8×
inception_4c/3x3_reduce	0.38	5	263 KB	36 KB	7×
inception_4c/3x3	0.43	3	1.2 MB	107 KB	11×
inception_4c/5x5_reduce	0.35	6	49 KB	8.4 KB	6×
inception_4c/5x5	0.26	3	154 KB	16 KB	10×
inception_4c/pool_proj	0.06	3	131 KB	16 KB	8×
inception_4d/1x1	0.43	6	230 KB	35 KB	7×
inception_4d/3x3_reduce	0.24	5	295 KB	45 KB	7×
inception_4d/3x3	0.38	3	1.5 MB	142 KB	10×
inception_4d/5x5_reduce	0.31	5	66 KB	9.7 KB	7×
inception_4d/5x5	0.34	3	205 KB	20 KB	10×
inception_4d/pool_proj	0.28	3	131 KB	13 KB	10×
inception_4e/1x1	0.50	4	542 KB	55 KB	10×
inception_4e/3x3_reduce	0.33	4	339 KB	41 KB	8×
inception_4e/3x3	0.58	3	1.8 MB	150 KB	12×
inception_4e/5x5_reduce	0.49	5	68 KB	8.3 KB	8×
inception_4e/5x5	0.32	3	410 KB	41 KB	10×
inception_4e/pool_proj	0.32	3	271 KB	26 KB	10×
inception_5a/1x1	0.45	3	853 KB	75 KB	11×
inception_5a/3x3_reduce	0.47	4	533 KB	56 KB	9×
inception_5a/3x3	0.39	3	1.8 MB	177 KB	10×
inception_5a/5x5_reduce	0.37	3	107 KB	9.9 KB	11×
inception_5a/5x5	0.27	3	410 KB	42 KB	10×
inception_5a/pool_proj	0.28	4	426 KB	53 KB	8×
inception_5b/1x1	0.24	3	1.3 MB	131 KB	10×
inception_5b/3x3_reduce	0.36	3	640 KB	59 KB	11×
inception_5b/3x3	0.71	3	2.7 MB	178 KB	15×
inception_5b/5x5_reduce	0.06	3	160 KB	19 KB	8×
inception_5b/5x5	0.25	3	615 KB	65 KB	9×
inception_5b/pool_proj	0.37	3	426 KB	40 KB	11×
loss3/classifier	0.45	3	4.1 MB	357 KB	11×
Overall			**28.0 MB**	**2.8 MB**	**10×**

Table 2. Layerwise compression statistics for GoogLeNet on ImageNet (p: pruning rate, b: bits per weight). Original top-1 accuracy: 68.9%. Compressed top-1 accuracy: 68.9%.

Layer	p	b	Original	Compressed	Rate
conv1	0.31	8	38 KB	9.7 KB	4×
res2a_branch1	0.64	8	66 KB	9.6 KB	7×
res2a_branch2a	0.01	6	16 KB	3.8 KB	4×
res2a_branch2b	0.55	5	147 KB	17.0 KB	9×
res2a_branch2c	0.14	6	66 KB	13.0 KB	5×
res2b_branch2a	0.49	6	66 KB	9.1 KB	7×
res2b_branch2b	0.40	7	147 KB	26.6 KB	6×
res2b_branch2c	0.32	4	66 KB	8.1 KB	8×
res2c_branch2a	0.06	7	66 KB	16.1 KB	4×
res2c_branch2b	0.09	5	147 KB	25.6 KB	6×
res2c_branch2c	0.35	4	66 KB	7.9 KB	8×
res3a_branch1	0.68	6	524 KB	50.9 KB	10×
res3a_branch2a	0.37	5	131 KB	18.1 KB	7×
res3a_branch2b	0.37	3	590 KB	54.4 KB	11×
res3a_branch2c	0.54	4	262 KB	25.2 KB	10×
res3b_branch2a	0.38	5	262 KB	35.9 KB	7×
res3b_branch2b	0.62	3	590 KB	42.1 KB	14×
res3b_branch2c	0.52	3	262 KB	21.9 KB	12×
res3c_branch2a	0.70	5	262 KB	21.4 KB	12×
res3c_branch2b	0.68	4	590 KB	45.3 KB	13×
res3c_branch2c	0.63	6	262 KB	29.9 KB	9×
res3d_branch2a	0.47	5	262 KB	32.3 KB	8×
res3d_branch2b	0.43	5	590 KB	77.8 KB	8×
res3d_branch2c	0.60	6	262 KB	31.3 KB	8×
res4a_branch1	0.38	4	2.1 MB	238 KB	9×
res4a_branch2a	0.62	3	524 KB	36.7 KB	14×
res4a_branch2b	0.57	3	2.4 MB	183 KB	13×
res4a_branch2c	0.53	3	1.0 MB	84.2 KB	12×
res4b_branch2a	0.75	3	1.0 MB	55.3 KB	19×
res4b_branch2b	0.85	4	2.4 MB	104 KB	23×
res4b_branch2c	0.73	3	1.0 MB	59.1 KB	18×
res4c_branch2a	0.58	3	1.0 MB	77.4 KB	14×
res4c_branch2b	0.71	3	2.4 MB	143 KB	16×
res4c_branch2c	0.62	3	1.0 MB	73.0 KB	14×
res4d_branch2a	0.74	3	1.0 MB	56.8 KB	18×
res4d_branch2b	0.82	4	2.4 MB	117 KB	20×
res4d_branch2c	0.65	3	1.0 MB	71.3 KB	15×
res4e_branch2a	0.64	3	1.0 MB	71.0 KB	15×
res4e_branch2b	0.85	3	2.4 MB	91.4 KB	26×
res4e_branch2c	0.43	3	1.0 MB	92.5 KB	11×
res4f_branch2a	0.53	3	1.0 MB	84.2 KB	12×
res4f_branch2b	0.80	3	2.4 MB	114 KB	21×
res4f_branch2c	0.40	3	1.0 MB	94.1 KB	11×
res5a_branch1	0.73	3	8.4 MB	466 KB	18×
res5a_branch2a	0.30	3	2.1 MB	202 KB	10×
res5a_branch2b	0.84	3	9.4 MB	372 KB	25×
res5a_branch2c	0.73	3	4.2 MB	231 KB	18×
res5b_branch2a	0.43	3	4.2 MB	367 KB	11×
res5b_branch2b	0.62	3	9.4 MB	694 KB	14×
res5b_branch2c	0.91	3	4.2 MB	104 KB	40×
res5c_branch2a	0.53	3	4.2 MB	333 KB	13×
res5c_branch2b	0.86	3	9.4 MB	344 KB	27×
res5c_branch2c	0.78	3	4.2 MB	203 KB	21×
fc1000	0.74	4	8.2 MB	513 KB	16×
Overall			**102.5 MB**	**6.7 MB**	**15×**

Table 3. Layerwise compression statistics for ResNet-50 on ImageNet (p: pruning rate, b: bits per weight). Original top-1 accuracy: 73.1%. Compressed top-1 accuracy: 73.7%.

	Δ Accuracy	Network Size
AlexNet on ImageNet		
Uncompressed	–	243.9 MB
Data-Free Pruning [47] (CaffeNet)	-2.2%	159 MB
Deep Fried Convnets [57]	-0.3%	68 MB
Less Is More [60]	-0.6%	57 MB
Dynamic Network Surgery [13]	-0.2% [1]	13.8 MB
Circulant CNN [3]	-0.4%	12.7 MB
Quantized CNN [53]	-1.4%	12.6 MB
Binary-Weight-Networks [41]	+0.1%	7.6 MB
Deep Compression [16]	+0.0%	8.9 (6.9) MB
[16] + Weighted-Entropy Quantization [38]	-0.8%	8.3 (6.5) MB
CLIP-Q	+0.7%	**4.8 MB**
GoogLeNet on ImageNet		
Uncompressed	–	28.0 MB
Weighted-Entropy Quantization [38]	+0.2%	4.4 MB
CLIP-Q	+0.0%	**2.8 MB**
ResNet-50 on ImageNet		
Uncompressed	–	102.5 MB
ThiNet [34]	-1.9%	49.5 MB
Weighted-Entropy Quantization [38]	-1.8%	16.0 MB
CLIP-Q	+0.6%	**6.7 MB**

Table 4. Network compression performance compared with state-of-the-art algorithms

sult extends the previous best compressed AlexNet result, Deep Compression + Weighted-Entropy Quantization, by 1.7 MB while obtaining 1.5% higher accuracy. In terms of compression rate, CLIP-Q improves on the previous best compression rate (37.5×) by 35% relative. On GoogLeNet, CLIP-Q obtains a state-of-the-art compressed network size of 2.8 MB, improving on the previous best compression rate (6.4×) by 57% relative. On ResNet-50, CLIP-Q obtains a 139% relative improvement in the state-of-the-art compression rate while obtaining 2.4% higher accuracy.

5. Conclusion

We have presented a new method for deep network compression 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, 10× for GoogLeNet, and 15× for ResNet-50, improving upon previously reported compression rates by 35%, 57%, and 139% relative, respectively.

Limitations and future work. We have focused on compression performance in this work because practical test-time speedups are often dependent on the software implementation of basic network operations and on hardware details. For example, 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. This design has inspired acceleration methods that reduce the size of those large matrices by eliminating entire rows and columns via interpolation at test time or structured pruning at training time [11, 28, 52]. Unstructured pruning can benefit from specialized hardware engines for practical test-time acceleration [15]. Practical speed-ups on mobile or embedded devices will require careful consideration of these details.

In addition, recent work has shown that smaller network models are not necessarily more energy efficient because energy consumption is also determined by the pattern of memory accesses [55]. 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.

Acknowledgements

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

[1] estimated using Caffe AlexNet model accuracy

References

[1] 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. 1

[2] 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. 1

[3] 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. 3, 8

[4] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 3

[5] 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. 2

[6] 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. 1

[7] 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. 2

[8] 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. 3

[9] 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. 1, 3

[10] 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. 1, 3

[11] M. Figurnov, A. Ibraimova, D. Vetrov, and P. Kohli. PerforatedCNNs: acceleration through elimination of redundant convolutions. In *Advances in Neural Information Processing Systems*, 2016. 1, 3, 8

[12] 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. 5

[13] Y. Guo, A. Yao, and Y. Chen. Dynamic network surgery for efficient DNNs. In *Advances in Neural Information Processing Systems*, 2016. 1, 2, 8

[14] 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. 2

[15] 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. 8

[16] 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. 1, 2, 3, 4, 5, 6, 8

[17] 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. 2

[18] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: optimal brain surgeon. In *Advances in Neural Information Processing Systems*, 1992. 2

[19] 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. 1, 2, 5

[20] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. arXiv:1503.02531, 2015. 3

[21] 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. 1

[22] 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. 3

[23] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *IEEE Conf. on Computer Vision and Pattern Recognition*, 2017. 1

[24] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *European Conference on Computer Vision*, 2016. 1

[25] 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. 3

[26] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference*, 2014. 3

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2012. 1, 2, 5

[28] V. Lebedev and V. Lempitsky. Fast ConvNets using group-wise brain damage. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 8

[29] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, 1990. 2

[30] 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. 1

[31] J. Liu, Y. Wang, and Y. Qiao. Sparse deep transfer learning for convolutional neural network. In *AAAI Conference on Artificial Intelligence*, 2017. 2

[32] 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. 1

[33] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2015. 1

[34] 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. 8

[35] 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. 1

[36] 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. 1

[37] T. Ojala, M. Pietikainen, and D. Harwood. A comparative study of texture measures with classification based on feature distributions. *Pattern Recognition*, 29(1):51–59, 1996. 2

[38] 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. 2, 4, 8

[39] 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. 1

[40] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006. 5

[41] 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. 2, 4, 8

[42] 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. 1

[43] 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. 3

[44] 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. 5

[45] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 1

[46] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, 2012. 5

[47] S. Srinivas and R. V. Babu. Data-free parameter pruning for deep neural networks. In *British Machine Vision Conference*, 2015. 2, 8

[48] 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. 1, 2, 3, 5

[49] 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. 1

[50] K. Ullrich, E. Meeds, and M. Welling. Soft weight-sharing for neural network compression. In *International Conference on Learning Representations*, 2017. 2, 4

[51] 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. 4

[52] 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. 8

[53] 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. 2, 8

[54] F. Xu, V. N. Boddeti, and M. Savvides. Local binary convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 2

[55] 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. 2, 8

[56] Z. Yang, X. He, J. Gao, L. Deng, and A. Smola. Stacked attention networks for image question answering. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2016. 1

[57] 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. 3, 8

[58] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *International Conference on Learning Representations*, 2016. 1

[59] 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. 3

[60] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: towards compact CNNs. In *European Conference on Computer Vision*, 2016. 8