# Hypercolumn Sparsification for Low-Power Convolutional Neural Networks

PRAVEEN K. PILLY, NIGEL D. STEPP, YANNIS LIAPIS, DAVID W. PAYTON, and NARAYAN SRINIVASA, Center for Autonomy Computing, Information and Systems Sciences Laboratory, HRL Laboratories, LLC, USA

We provide here a novel method, called hypercolumn sparsification, to achieve high recognition performance for convolutional neural networks (CNNs) despite low-precision weights and activities during both training and test phases. This method is applicable to any CNN architecture that operates on signal patterns (e.g., audio, image, video) to extract information such as class membership. It operates on the stack of feature maps in each of the cascading feature matching and pooling layers through the processing hierarchy of the CNN by an explicit competitive process ($k$-WTA, winner take all) that generates a sparse feature vector at each spatial location. This principle is inspired by local brain circuits, where neurons tuned to respond to different patterns in the incoming signals from an upstream region inhibit each other using interneurons, such that only the ones that are maximally activated survive the quenching threshold. We show this process of sparsification is critical for probabilistic learning of low-precision weights and bias terms, thereby making pattern recognition amenable for energy-efficient hardware implementations. Further, we show that hypercolumn sparsification could lead to more data-efficient learning as well as having an emergent property of significantly pruning down the number of connections in the network. A theoretical account and empirical analysis are provided to understand these effects better.

CCS Concepts: • **Computing methodologies** → **Neural networks**;

Additional Key Words and Phrases: Convolutional neural networks, machine learning, low precision, sparse, quantization, low energy, embedded systems, object recognition

**ACM Reference format:**
Praveen K. Pilly, Nigel D. Stepp, Yannis Liapis, David W. Payton, and Narayan Srinivasa. 2019. Hypercolumn Sparsification for Low-Power Convolutional Neural Networks. *J. Emerg. Technol. Comput. Syst.* 15, 2, Article 20 (March 2019), 16 pages.
https://doi.org/10.1145/3304104

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under contract No. HR0011-13-C-0052. The views expressed are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA.

Authors' addresses: P. K. Pilly, N. D. Stepp, and D. W. Payton, Center for Autonomy Computing, Information and Systems Sciences Laboratory, HRL Laboratories, LLC, Malibu, CA, 90265, USA; emails: {pkpilly, ndstepp, dwpayton}@hrl.com; Y. Liapis, Enthought, Inc., Austin, TX 78701; email: yliapis@enthought.com; N. Srinivasa, Eta Compute, Westlake Village, CA 91362; email: physynapse@gmail.com.

1550-4832/2019/03-ART20 $15.00
https://doi.org/10.1145/3304104

## 1 INTRODUCTION

The broad class of deep learning methods, which includes the convolutional neural networks (CNNs) (Sermanet et al. 2013), HMAX model (Serre et al. 2007), and hierarchy of auto-encoders (Masci et al. 2011), has shown good performance on a variety of pattern classification tasks. Still, these methods can have difficulty dealing with the challenge of the 3 Vs (volume, velocity, and variety), where data should be processed at the point of collection to minimize decision latency. The key disadvantage of these methods, with respect to low-energy mobile hardware implementations, is that they require high numerical precision to store and process the great number of weights and cell activities. This is the case because, at low precision, the weight updates in both incremental and batch learning modes are relatively small compared to the interval between the quantization levels for the weights and so are unlikely to be registered. CNNs are leading architectures for pattern recognition but are expensive in terms of the large scale of multiplications that are required. CNNs typically use several alternating layers of convolution and pooling, with the convolution layers extracting increasingly complex features through the hierarchy and the pooling layers creating invariances to position, size, rotation, and so on. Energy per multiplication scales exponentially with the number of bits. So, reducing numerical precision reduces energy consumption, but recognition performance suffers. The minimum number of bits required to adapt the weights online and achieve reasonable performance for CNNs can be prohibitive for meeting low-energy and high-throughput challenges when input size and depth of the pipeline increases. The challenge, then, is to achieve online learning for pattern recognition at low numerical precision for activities and learned weights without compromising accuracy. The approach presented in this article compensates for low precision in CNNs by incorporating the ubiquitous principle of lateral inhibition (Ferster and Miller 2000) seen in several brain circuits across modalities. We call our method *hypercolumn sparsification*, as a hypercolumn in mammalian cortical organization refers to a pool of mutually inhibiting neurons with overlapping receptive fields in a same cortical area that are tuned to detect different features (e.g., orientations of edges, directions of moving stimuli). In our case, it is this inhibition between competing hypercolumn features to allow $k$ winners that cause a unique sparse distributed representation.

While more recent techniques such as the BinaryConnect algorithm (Courbariaux et al. 2014, 2015) and XNOR network (Rastegari et al. 2016) aggressively reduce the cell activities and/or weights to single bit precision, and their extensions quantize the error gradients down to 6 bits (Hubara et al. 2016; Miyashita et al. 2016; Zhou et al. 2016), the weights are only binarized in the feedforward pass. These training methods all require saving a double precision (real-valued) representation of weights to register high-resolution weight updates. Retaining such double precision representations is not conducive for energy-constrained on-chip machine learning.

Techniques utilizing sparse representations for pattern recognition have been explored in Graham (2013), Mutch and Lowe (2006), Mutch and Lowe (2008), and Liu et al. (2015), although each of these approaches differs from our method of $k$-winner-take-all along the hypercolumns of stacked feature maps. Sparse representation often means representing data using a subset of basis vectors from an overcomplete dictionary, where essentially only a partial embedding is used (Mairal et al. 2009). Sparse representations have been shown to be useful for face recognition, image super-resolution, object detection, compressed sensing, image denoising, and so on. Hypercolumn sparsification is a related but different kind of sparse representation to deal with low-precision computations for on-chip learning in CNNs and differs from other applications of sparsity to CNNs such as those operating on inputs (Graham 2014) or kernel weights (Liu et al. 2015).

Methods more closely related to our work can be found in Makhzani and Frey (2014) and Wen et al. (2016). In Wen et al. (2016), sparsity was achieved at the level of filters, channels, and depth by virtue of using group Lasso regularization. This regularization tends to zero out non-essential weights and can achieve a form of sparsity that is similar to the effects seen in our hypercolumn sparsification approach, although the specific number of zeroed out hypercolumns may vary. Even so, this method is aimed at reducing computational cost by zeroing-out various weights. There is no indication that this method would be effective for training and testing in the presence of low-precision weights. In Makhzani and Frey (2014), a method identical to ours is applied to autoencoders. Results show that their $k$-sparse autoencoder method outperforms denoising autoencoders on MNIST and NORB datasets. Here again. though, the effects of low-precision weights are not considered.

A well-known technique to deal with the issue of registering small weight updates with fewer bits in multi-layer networks is the probabilistic rounding method (Hoehfeld and Fahlman 1992). In this method, each weight change is first rectified and scaled by the interval between quantization levels for the weights and then compared with a uniform random number between 0 and 1. If the random number is relatively smaller, then the particular weight is updated to the neighboring quantization level in the direction of the initial weight change. But even this method requires at least 5–10 bits depending on the dataset, allowing for "gradual degradation in performance as precision is reduced to 6 bits" (Hoehfeld and Fahlman 1992). We have validated that hypercolumn sparsification can improve recognition performance compared to a five-layer baseline CNN for weights and activities quantized at <5 bits during both training and test phases.

Our method operates on deep learning architectures, which comprise multiple feature channels, to sparsify feature vectors of hypercolumns at each layer in the hierarchy (see Figure 2). As mentioned above, CNNs have several cascading stages of feature matching and pooling layers to generate a high-level multi-channel representation of the input data for pattern recognition. Cells in each feature matching layer infer the degree of match between different learned patterns (based on feature channels) and cell activities in the upstream layer within their localized receptive fields. Cells in each pooling layer either compute the mean or take the maximum of cell activities in the upstream layer within their localized receptive fields. Hypercolumn sparsification, which is applied during both training and testing, introduces explicit competition throughout the pipeline within each of the various sets of cells (hypercolumns) across the feature channels that share a spatial receptive field. Within each such set of cells with a same spatial receptive field (namely, location), this operation ensures that only a given fraction of cells with maximal activities (e.g., 10%) are able to propagate their signals to the next layer in the deep learning network. Output activities of non-selected cells are quenched to zero. Figure 1 is an illustration of how this method works. When sparsification is applied in each hypercolumn across space and at each layer in a deep learning network, sparse distributed representations are created. For a visual stimulus, this is in line with the premise that at each spatial location there are at most a handful features that can be present unambiguously; i.e., the various feature detectors for each location compete among themselves such that a suitable stimulus representation is achieved across space.

The remainder of the article is organized into the Methods, Results, Discussion, and Conclusion sections. The Methods section provides details of our simulation experiments, including the dataset and the metric used. The Results section not only provides but also interprets the key results from the simulation experiments to demonstrate the various effects of hypercolumn sparsification for low-precision CNNs. The Discussion section summarizes the practical benefits of our method and provides ideas for future work.

## 2 METHODS

We performed a series of simulation experiments to understand the contribution of hypercolumn sparsification to CNNs for achieving high recognition performance despite low numerical

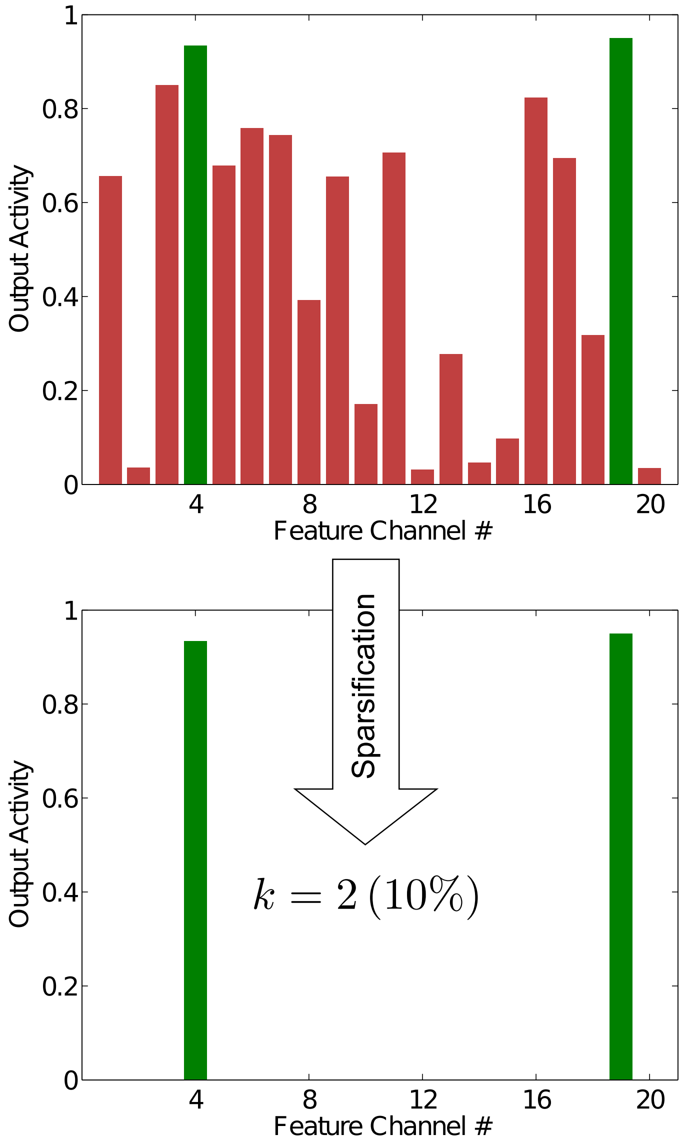

Fig. 1. Illustration of the hypercolumn sparsification process that operates in each hypercolumn of the stacked feature maps in each layer of a CNN, by which a subset (10% in this case) of the cells with the highest activity survive the cut while others are quenched to zero.

precision of stored parameters (<6 bits). Our baseline CNN (called "Conventional") is a double precision implementation comprised an input layer (IL) of size $64 \times 64$, which registers the grayscale image of an image patch; two cascading stages of alternating feature matching (S1, S2) and pooling (C1, C2) layers with 20 feature channels each; and a classification layer (CL) with six output cells (see Figure 2). The first matching layer (S1) consists of twenty $60 \times 60$ maps, the first pooling layer (C1) of twenty $20 \times 20$ maps, the second matching layer (S2) of twenty $16 \times 16$ maps, and the second pooling layer (C2) of twenty $6 \times 6$ maps. Each map in S1 is formed by convolving IL with

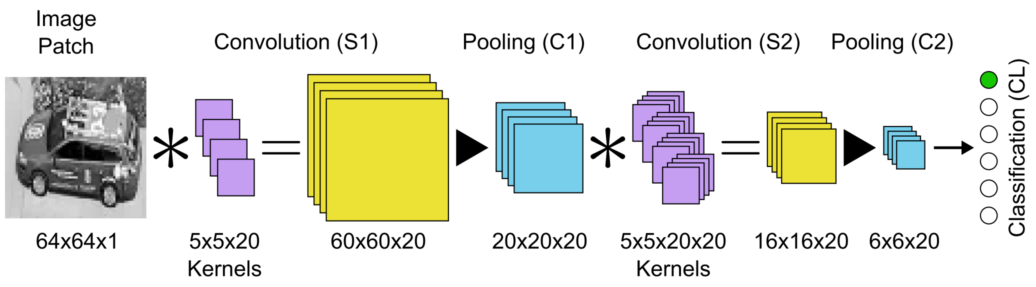

Fig. 2. Block diagram of an illustrative five-layer convolution neural network–(CNN) based recognition system, from the image patch to the classification layer (CL). This conventional CNN has two cascading stages of feature matching and pooling layers.

one of twenty $5 \times 5$ kernels, and each map in S2 filters inputs from all feature channels in C1. Both pooling layers (C1, C2) subsample their input matching layers by calculating mean values over $3 \times 3$ non-overlapping spatial windows in each of the 20 maps. Each map in feature matching layers S1 and S2 and each cell in the classification layer also had bias terms, and the logistic activation function $(1/(1 + e^{-x}))$ was used on outputs of S1, S2, and CL to suppress noise and also place bounds on the corresponding cell activities ($[0, 1]$).

Our Conventional CNN was incrementally trained with backpropagation-based gradient descent to minimize the sum of squared errors with a learning rate $\eta = 0.1$ for one epoch. The epoch presented 100,000 labeled examples that were sampled randomly from the Training sequences of the Stanford Tower dataset (http://ilab.usc.edu/neo2/dataset/). The initial values for all the weights and bias terms were randomly sampled from a uniform distribution from 0 to 1. All bounding boxes were converted to gray-scale and resized to $64 \times 64$ using bilinear interpolation. Six classes ("Car," "Truck," "Bus," "Person," "Cyclist," and "Background") were present in the Training set with the following base rates: 11.15%, 0.14%, 0.44%, 19.34%, 8.93%, and 60%, respectively. The trained Conventional CNN was evaluated on a representative subset of 10,000 boxes that were sampled randomly from the Test sequences of the Stanford Tower dataset, which maintained the base rates of the classes used for training. Test performance was evaluated using a metric adapted from Kasturi et al. (2014) and Wong et al. (2017) to assess recognition of multiple classes with varying base rates, which is computed as follows:

(a) A normalized accuracy score was first computed for each of the five object classes ("Car," "Truck," "Bus," "Person," "Cyclist") across all the bounding boxes in the Test set:

$$P = 1 - \frac{c_m M + c_{fa} F}{G}, \tag{1}$$

where $M$, $F$, and $G$ are the number of misses, false alarms, and ground-truth objects, respectively. $P$ penalizes misses and false alarms using the associated costs $c_m$ and $c_{fa}$ (each set to a value of 1), which are normalized by the number of ground-truth instances of the class. The $P$ scores range from $-\infty$ to 1. They are 0 when the system misses all objects of a given class and has no false alarms. An object misclassification is considered a miss for the ground-truth class but not a false alarm for the predicted class. However, a "Background" input image that is misclassified as one of the five object classes is counted as a false alarm.

(b) A single performance score $S$ was then calculated by a weighted average of the $P$ scores across the five object classes using their normalized frequencies $f_i$ (between 0 and 1) in

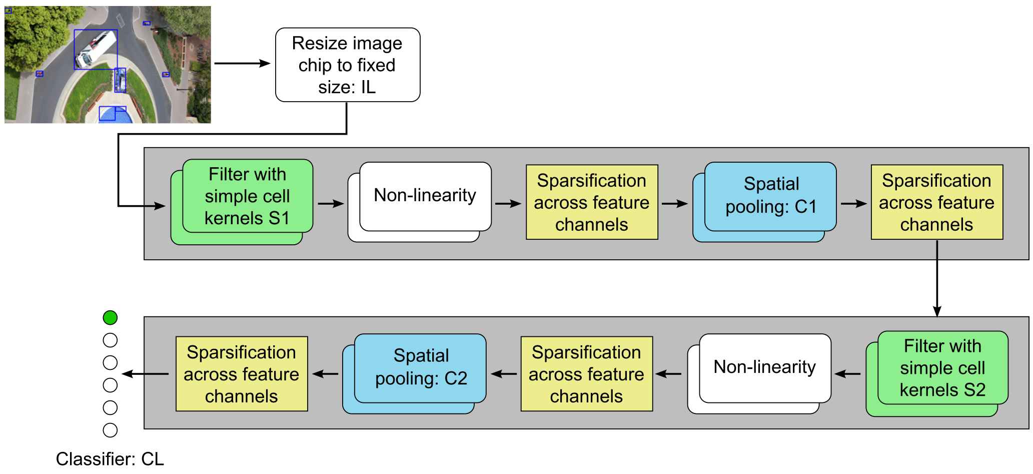
Resize image chip to fixed size: IL
Filter with simple cell kernels S1
Non-linearity
Sparsification across feature channels
Spatial pooling: C1
Sparsification across feature channels
Filter with simple cell kernels S2
Non-linearity
Sparsification across feature channels
Spatial pooling: C2
Sparsification across feature channels
Classifier: CL

Fig. 3. Application of hypercolumn sparsification to each layer (other than input and output layers) of a conventional CNN, whose schematic is shown in Figure 2. Here each image patch (or bounding box) detected in a video frame is processed through the classification pipeline to arrive at a class label.

the Test set:

$$S = \sum f_i \cdot P_i. \tag{2}$$

We next applied hypercolumn sparsification to a Conventional CNN (see Figure 3) and performed the same training as above with a parameter of $k = 10\%$ for sparsification in each of feature matching and pooling layers (namely, S1, C1, S2, C2). For this version of CNN (called "Sparse"), the weights were still learned with double precision as with the Conventional CNN. Cell activities were also represented at double precision. For both Conventional and Sparse CNNs, we analyzed test performance as a function of the number of training data examples (in steps of 5,000) to assess any potential differences in data efficiency. We ran four separate runs for each architecture with the Training set of 100,000 randomly ordered between runs.

For investigating low-precision CNNs, we employed two learning methods. For the first method, we implemented four variations of quantized CNNs, two of which used hypercolumn sparsification and two of which did not. For the CNNs without sparsification, the kernel weights and bias terms in the feature matching layers S1 and S2 were taken from the pre-trained Conventional CNN and hard-coded with quantization at 4 bits. And for the CNNs with sparsification, the kernel weights and bias terms in the feature matching layers S1 and S2 were taken from the pre-trained Sparse CNN and hard-coded with quantization at 4 bits. Cell activities throughout these new pipelines, including the input and output layers, were quantized at 3 bits on the range 0–1. And training comprised learning only the weights of connections from the final pooling layer (C2) to the classification layer (CL), and the bias terms at CL, at much lower than double precision with the number of bits systematically varied from 3 to 12 in steps of one. The quantization ranges for these weights and bias terms were obtained from the corresponding overall ranges of weights and bias terms for CL from the respective pre-trained CNNs. This training was done with and without probabilistic rounding, thus accounting for the four quantized CNNs. The quantized CNN that used hypercolumn sparsification, but not probabilistic rounding, is called "Gold" CNN. The other variants are referred to as follows: Gold CNN with probabilistic rounding, Non-Sparse Gold CNN, and Non-Sparse Gold CNN with probabilistic rounding.

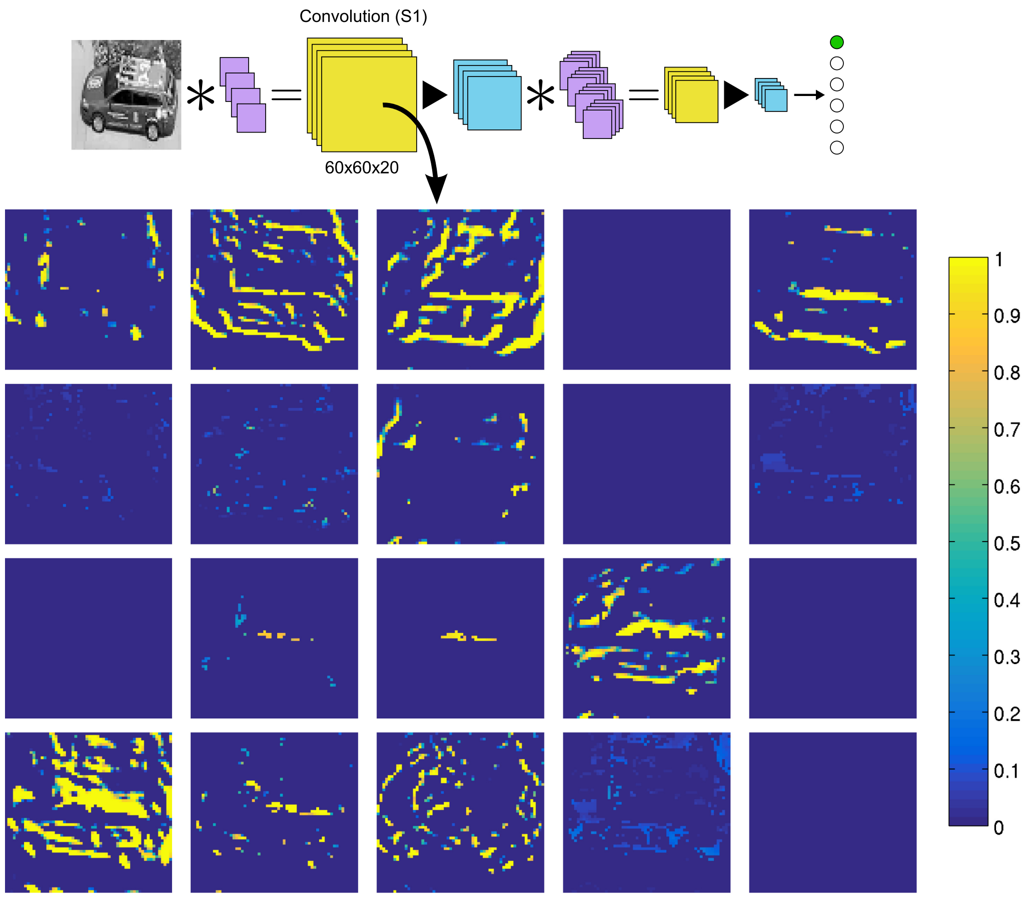

Fig. 4. The application of hypercolumn sparsification, through regular supervised training, automatically down-selects the number of useful feature channels in each layer of the CNN depicted at the top. The effects of sparsification on feature maps at the first matching layer (S1) after one epoch of training are shown here.

For the second method, we implemented the same four variations of quantized CNNs but all the weights and bias terms throughout the network were learned from training (with numerical precision systematically varied from 3 to 17 bits in steps of one). Cell activities throughout these new pipelines were quantized at the same precision as the weights and bias terms on the range 0–1. The quantization ranges for the weights and bias terms in S1, S2, and CL were obtained from the ranges of all weights and bias terms for the corresponding layers from the respective pre-trained CNNs, as in the first method.

## 3 RESULTS

While all 20 feature channels in each layer were employed for the Conventional CNN, the application of hypercolumn sparsification for Sparse CNN gradually self-selected a subset of the channels in each layer through the training. Figure 4 highlights this novel property for the first feature matching layer (S1). In particular, we found that sparsification led to about 15× reduction in network size without compromising performance.

Furthermore, Figure 5 shows that Sparse CNN learned faster with fewer training examples to reach a particular Test performance level than Conventional CNN. A paired-sample *t*-test shows

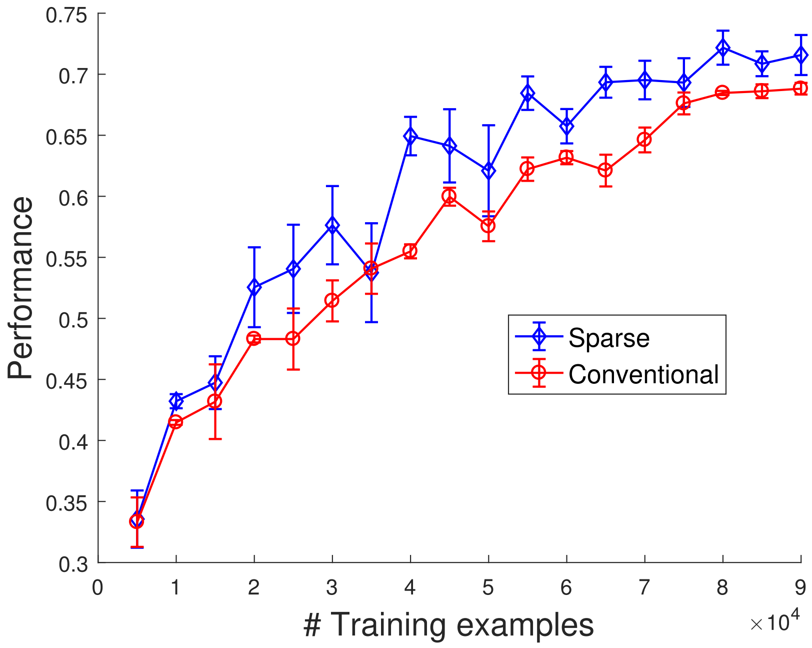

Fig. 5. Tracking performance during training shows that Sparse CNN tends to require fewer training examples to reach a particular performance level than Conventional CNN. Error bars show the standard error of the mean (SEM) over four runs.

that the performance of Sparse CNN significantly outperforms that of Conventional CNN through the entire training period ($t(53) = -5.106, p < 1e-5$).

Figure 6 shows the Test performance for the four variations of quantized CNNs implemented using the first learning method as numerical precision was systematically varied for CL weights and bias terms. Several observations can be made. First, the probabilistic rounding method improved performance at low precision (<6 bits) for both Gold CNN and Non-Sparse Gold CNN (compare green curve with red, and blue curve with cyan). Second, hypercolumn sparsification also improved performance with probabilistic rounding (≥3 bits) and without (>4 bits); compare the green curve with the blue and the red curve with the cyan. Third, at lower precision (≤4 bits) the best performance is obtained using a combination of hypercolumn sparsification and probabilistic rounding. Indeed, Gold CNN with probabilistic rounding outperforms Non-Sparse Gold CNN without probabilistic rounding (which is equivalent to a regular quantized CNN) at 3 bits by about 30% and at 4 bits by about 45%. The probabilistic rounding method, by default, allows weights to jump at most to neighboring quantization levels. As numerical precision for CL weights increases, the interval between quantization levels decreases, causing the weights to register increasingly smaller changes during training. This effectively reduces the learning rate, allowing Gold CNN to begin outperforming Gold CNN with probabilistic rounding after 5 bits. Similarly, probabilistic rounding would impair the performance of Non-Sparse Gold CNN as numerical precision is increased.

Figure 7 shows the Test performance for the four variations of quantized CNNs implemented using the second learning method, where the entire pipeline is trained with quantized weights and activities as a function of the numerical precision. Similarly to Figure 6, the probabilistic rounding method yielded better performance at low precision for both Gold CNN (<7 bits) and Non-Sparse Gold CNN (between 6 to 11 bits); compare the green curve with the red and the blue curve with the

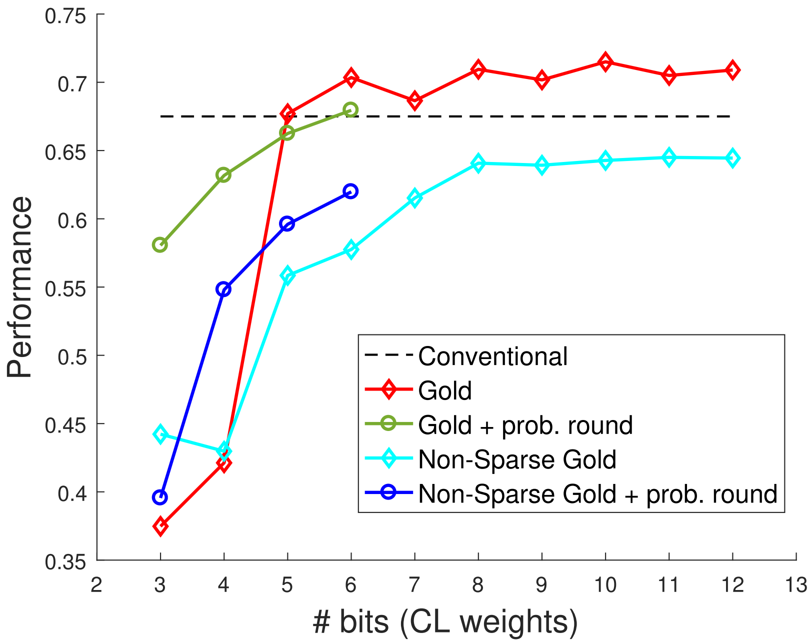

Fig. 6. Performances of Gold and Non-Sparse Gold CNNs with and without probabilistic rounding are shown as a function of numerical precision (from 3 to 12 bits in steps of one) that was used to learn and store the weights and bias terms for the last layer (namely, CL). For comparison, the performance of Conventional CNN, which does not have any restrictions on weights and cell activities, is shown as a black line.

cyan. Further, hypercolumn sparsification yielded better performance with probabilistic rounding (<10 bits) and without (between 6 to 16 bits); compare the green curve with the blue and the red curve with the cyan. Moreover, at lower precision (≤7 bits) the best performance is obtained by Gold CNN with probabilistic rounding. The combined benefits of hypercolumn sparsification and probabilistic rounding seen in Figure 6 are amplified here in the more challenging circumstance of learning all the weights and bias terms throughout the pipeline at reduced precision. However, Non-Sparse Gold CNN without probabilistic rounding (which is equivalent to a regular quantized CNN) began to perform only at 16 bits.

As described under Methods, the results shown in Figures 6 and 7 are produced using the logistic activation function. Many recent implementations of CNN and other deep learning networks, however, use the rectified linear unit (ReLU) activation function (Krizhevsky et al. 2012). Figure 8 shows the effects of sparsity, with and without probabilistic rounding, on the network with ReLU activations in the hidden layers. To show these effects at low precision, the number of bits used for weights was swept from 4 to 8, while keeping the precision of activities constant at 3 bits. As for the results shown in Figure 7, each of the four variations of quantized CNNs was initialized with layer-specific quantization ranges from a trained Sparse CNN with ReLU activations before training on one epoch of 100,000 labeled examples. Figure 8 shows the Test performances following training with quantized weights and activities. Low-precision performances using ReLU activations appear to be relatively impaired (compared to those using the logistic function), with the best results for Gold CNN with probabilistic rounding. In other words, the combined benefits of hypercolumn sparsification and probabilistic rounding at lower precision for weights and activities during both training and testing are qualitatively replicated also with the ReLU activation function.

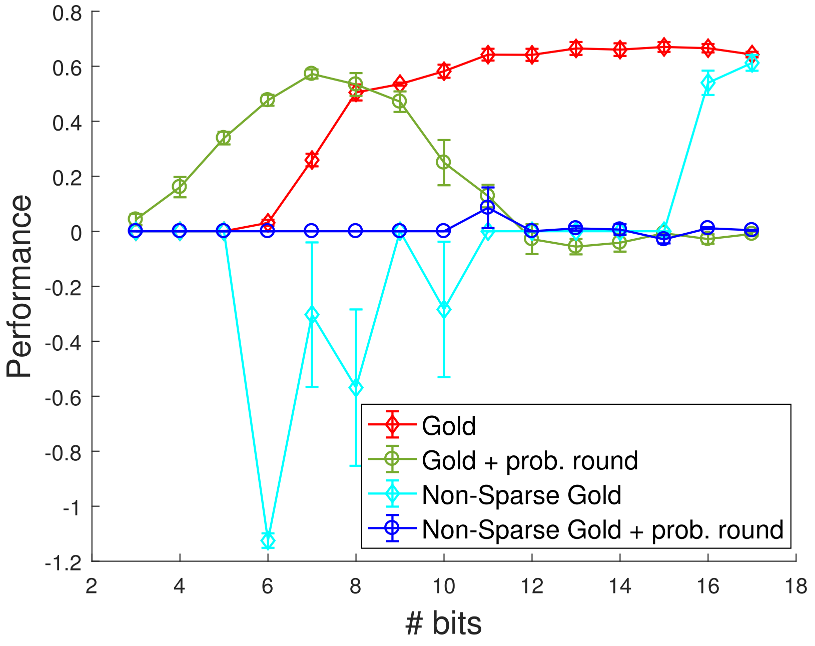

Fig. 7. Performances of Gold and Non-Sparse Gold CNN with and without probabilistic rounding are shown as a function of numerical precision (from 3 to 17 bits in steps of one) that was used to learn and store the weights and bias terms throughout the entire network (namely, S1, S2, CL), as well as to represent all cell activities. Error bars show the standard error of the mean (SEM) over four runs.

## 3.1 Explaining the Benefits of Hypercolumn Sparsification

The intuitive explanation for why hypercolumn sparsification leads to faster learning is that since only strong activities are kept, only meaningful changes to kernel weights are made. This insight comes from examining the process of backpropagation itself. Starting with an error vector at CL, which is the difference between actual and desired output activities, errors (or deltas) are computed sequentially down the hierarchy using the backpropagation algorithm using current-layer activities and previous-layer errors as follows:

$$\Delta^L = X^L \left(1 - X^L\right) T\{\Delta^{L+1}\}, \tag{3}$$

where $X^L$ is the activity matrix for layer $L$, $T$ is the de-convolution operation (Duda et al. 1973), and $\Delta^{L+1}$ is the error matrix from layer $L + 1$. In turn, these deltas multiplied by subsequent-layer activities and scaled by learning rate become kernel weight updates. The important aspect to note is that if an activity is zero, in this case due to sparsification, then the corresponding delta term will also be zero, and that particular cell will not contribute to changes in kernel weights.

This intuitive argument may be checked through the analysis of deltas during learning and their correspondence to output classifications. Specifically, we calculated two information theoretic measures (Cover and Thomas 2012): conditional entropy, $H(X|Y)$, and mutual information, $I(X;Y)$. Conditional entropy measures uncertainty of one variable provided you know the value of another. Mutual information is a related measure with a slightly different interpretation, which quantifies how much information is shared between two variables.

Using deltas $\Delta$ in Equation (3) as one variable of interest, we choose the other to be the detrended cumulative sum of correct classifications. Given that output classifications are either correct or not,

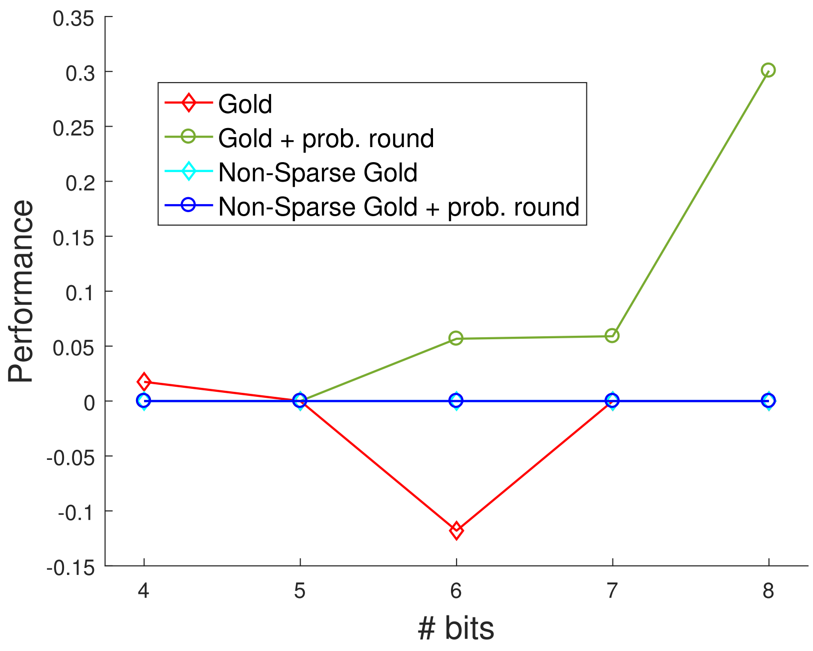

Fig. 8. Effects of ReLU activation function on the performances of Gold and Non-Sparse Gold CNN with and without probabilistic rounding are shown as a function of numerical precision (from 4 to 8 bits in steps of one) that was used to learn and store the weights and bias terms throughout the entire network (namely, S1, S2, CL).

we may think of this history of classifications as a binary-valued discrete time-series, $x_n$, where $x_n = 1$ if classification of example $n$ was correct and zero otherwise. Changes in performance may be captured using a cumulative sum $y$, such that

$$y_n = \sum_{i=0}^{n} x_i. \tag{4}$$

Last, since this cumulative sum has an upward trend that is not strictly related to the deltas, a linear trend $\hat{y}_n$ is removed, resulting in a relatively stationary time series, the shape of which details the history of classification performance,

$$z_n = y_n - \hat{y}_n. \tag{5}$$

With these two variables of interest, we examined conditional entropy $H(z|\Delta)$ and mutual information $I(\Delta; z)$. If sparsification helps to produce more focused and efficacious deltas, then we would expect sparse deltas to be more informative about classification performance than conventional deltas. That is, we expect conditional entropy to be less and mutual information to be greater.

We performed an empirical study comparing these measures by training both Conventional and Sparse CNNs using 20 different random seeds and 10,000 training examples. For each random seed run, the delta matrices in the first feature map in S2 are flattened in row-major order so that $H$ and $I$ may be computed for each element across the training examples. The measures were then averaged across the elements to get a summary value for each random seed. Results of this study are summarized in Figures 9 and 10. Conditional entropy was lower in the sparse case, judged by

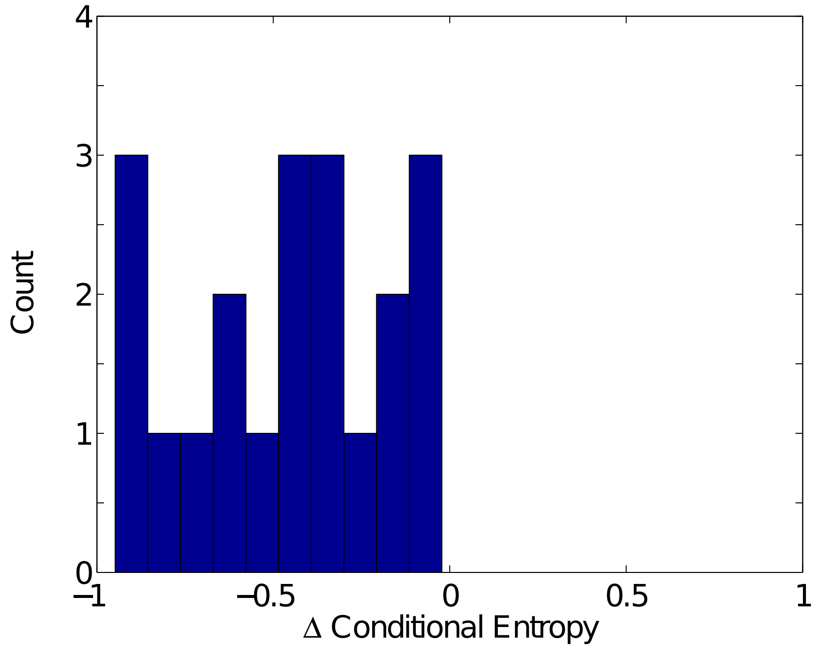

Fig. 9. Histogram of conditional entropy difference scores (Sparse–Conventional), showing that sparse deltas tend to reduce uncertainty about classification performance more than conventional deltas.

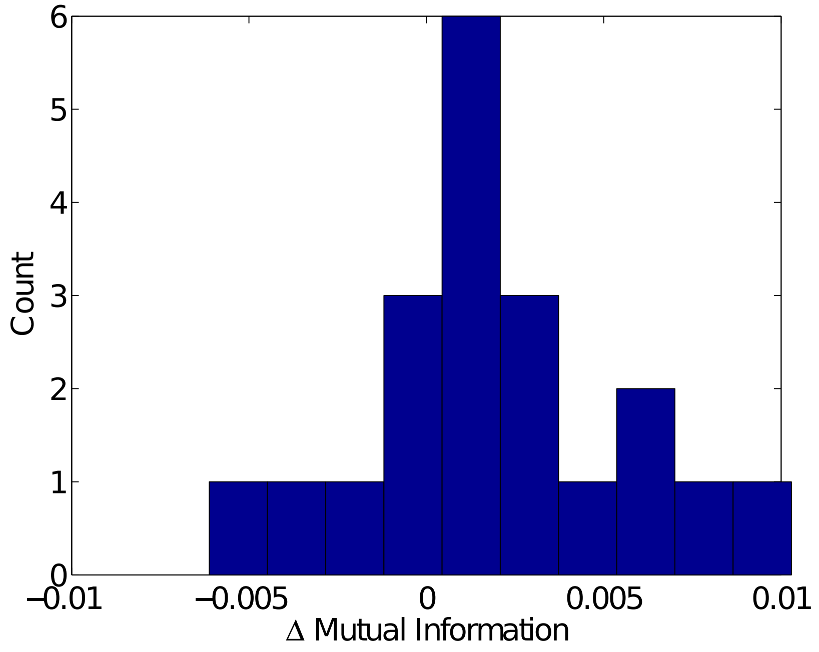

Fig. 10. Histogram of mutual information difference scores (Sparse–Conventional), showing that sparse deltas tend to share more information with classification performance than conventional deltas.

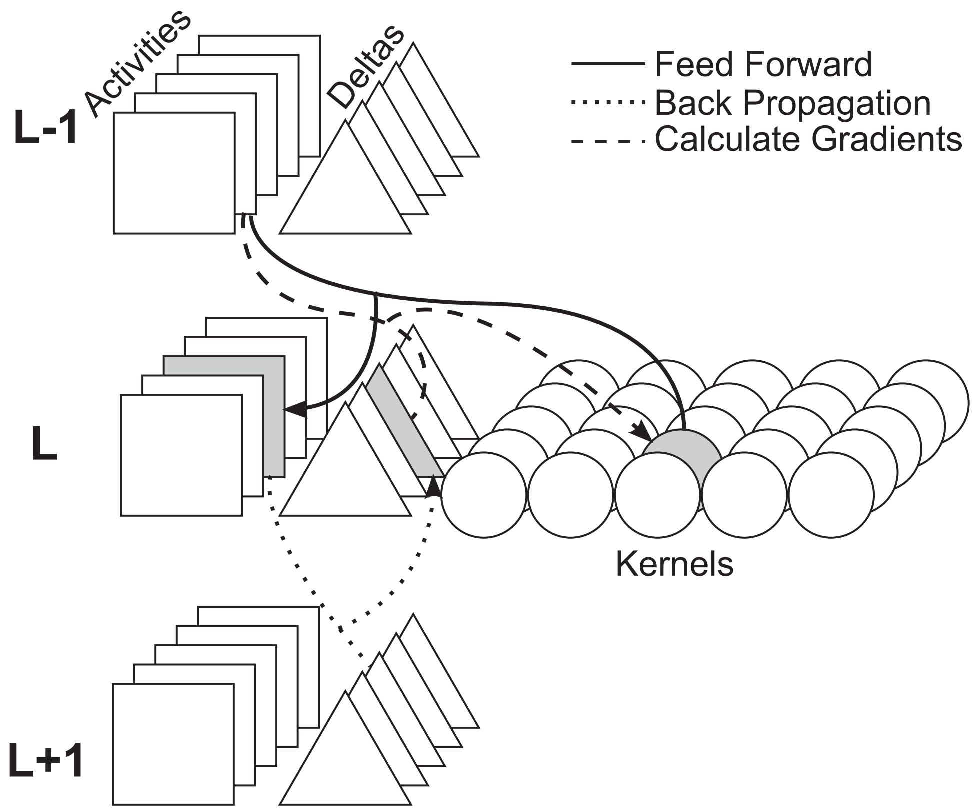

Fig. 11. Graphical depiction of the backpropagation cycle. In the feedforward step, a kernel and activity are combined resulting in a new activity in the next layer. In the backpropagation step, an activity and post-layer delta are combined resulting in a new delta. In the parameter update step, the delta and previous layer activity are combined resulting in a new kernel. These steps describe a cycle from kernel, to activity, to delta, back to kernel.

a left-tailed $t$-test of difference scores ($t(19) = -7.09, p < 0.001$). Mutual information was higher in the sparse case, judged by a right-tailed $t$-test of difference scores ($t(19) = 2.27, p < 0.05$).

### 3.2 Sparse Activities Lead to Sparse Weights

Assume that kernel element $k^L_{mnab}$, where $m$ indexes one of $M$ input maps, $n$ indexes one of $N$ output maps, $L$ specifies the CNN layer, and $(a, b)$ indexes the specific element of the kernel matrix, was increased as a result of a backpropagation calculation. Then, in the next feedforward pass, activity $X^L_{nij}$ is given by

$$X^L_{nij} = f\left(\sum_m \sum_k X^{L-1}_{mkl} k^L_{mn(i-k)(j-l)} + b^L_n\right), \tag{6}$$

where $f()$ is the logistic activation function and $b^L_n$ is the bias term for feature map $n$ in layer $L$. The relationship between these components in the feedforward and backpropagation steps is depicted in Figure 11.

In a Sparse CNN, there are two cases for the value of $X^{L-1}_{mkl}$ when $a = i - k, b = j - l$. Either $X^{L-1}_{mkl} = 0$, if its value was below the sparsification threshold, or $X^{L-1}_{mkl} = x$, where $x$ is a value greater than the sparsification threshold. In the latter case, the increase of $k^L_{mnab}$ leads to an increase of $X^L_{nij}$, *ceteris paribus*, making it more likely that $X^L_{nij}$ will itself be above the sparsification threshold. If, however, $X^{L-1}_{mkl} = 0$ or $k^L_{mnab}$ is small, then $X^L_{nij}$ will also be small or vanish, making it less likely to be above the sparsification threshold.

Following the backpropagation process, $\Delta^L_n$ is given by Equation (3) above, leading to a change in kernel weights given by the convolution

$$\delta^L_{mn} = \text{rev}\left\{X^{L-1}_m\right\} * \Delta^L_n * \eta, \tag{7}$$

where rev is a two-dimensional reversing operation, and $\eta$ is the learning rate. At the element-wise level, where $Z^{L-1}_m = \text{rev}\{X^{L-1}_m\}$, or $Z^{L-1}_{mij} = X^{L-1}_{m(K-i)(K-j)}$ when $X$ is a $K \times K$ matrix, this convolution is implemented as

$$\delta^L_{mnab} = \eta * \sum_k \sum_l Z^{L-1}_{mkl} \Delta^L_{n(a-k)(b-l)}. \tag{8}$$

If we assume, as above, that $X^{L-1}_{mkl} = x$ and that $X^L_{nij}$ has increased above the sparsification threshold, then $Z^{L-1}_{m(K-k)(K-l)} = x$ and

$$\Delta^L_{nij} = X^L_{nij}\left(1 - X^L_{nij}\right) T\left\{\Delta^{L+1}_n\right\} \tag{9}$$

will be non-zero depending on the value of $T\{\Delta^{L+1}_n\}_{ij}$, as will $\delta^L_{mnab}$.

The preceding shows that a probabilistic positive feedback loop exists. If an element in kernel $k^L_{mn}$ is increased, then it increases the likelihood that an activity in $X^L_n$ rises above the sparsification threshold. Likewise, if an element of $X^L_n$ is increased above the sparsification threshold, then it increases the probability that an element in kernel $k^L_{mn}$ is increased. Specifically, if $\delta^L_{mnab} > 0$, then one of the terms of the sum in Equation (6) contains $k^L_{mnab} + \delta^L_{mnab}$, and $X^L_n$ will have an element that is increased by $X^{L-1}_{mij}\delta^L_{mnab}$. If the sparsification threshold for that feature is $\theta$, then $P(X^L_{nij} + X^{L-1}_{mij}\delta^L_{mnab} > \theta) > P(X^L_{nij} > \theta)$. In the other direction, if $X^L_{nij} < \theta$, then $P(\delta^L_{mnab} > 0) = 0$ but can be non-zero otherwise.

## 4 DISCUSSION

Our article shows that hypercolumn sparsification is critical when probabilistic rounding is applied to learn weights at low numerical precision. Sparsification improves learning, because it restricts the weight updates to only those projections whose input and output neurons have "signal" activities. In the case without sparsification, weights do not stabilize toward minimizing the loss at the final classification layer because of "noisy" jumps from one quantization level to the other in almost all connections. In summary, our method not only is useful for reducing the energy consumption of any CNN but also is critical for any learning to happen in the first place when weights are to be learned and stored at low precision.

Our sparsification method serves a twofold purpose: (a) identify a subset of feature channels that are sufficient and necessary to process a given dataset for pattern recognition and (b) ensure optimal recognition performance for the situations in which the weights of connections between cells in the network and the cell activities themselves can only be represented and processed at low precision. It also allows for faster, more data-efficient learning. These goals play a critical role for practical realizations of state-of-the-art deep learning architectures. As recognition problems grow in complexity, the number of layers required grows as well, leading to ever-increasing processing and memory requirements. For instance, the well-known OverFeat architecture (Sermanet et al. 2013) uses 11 layers (8 feature matching and 3 MAX pooling), with the number of channels ranging from 96 to 1,024 at different layers. It recognizes objects from among 1,000 classes in response to input images sized at 231 × 231. The need for numerical precision leads to more size, weight, area, and power requirements, which are prohibitive for practical real-world deployment of these state-of-the-art deep learning networks.

Through the use of low-precision weights and cell activities in conjunction with our sparsification method, we can greatly reduce both memory and processing loads while maintaining, or even enhancing, learning, and performance of conventional CNNs. Moreover, as we seek to perform online training of these architectures without the benefit of the traditional memory-intensive and power hungry floating-point hardware, the advantages sparsity provides for training low-precision hardware will become evermore important. As the world continues to move toward smaller, lightweight, more portable applications for cell phones, autonomous cars, UAVs, or even stationary battery-powered surveillance cameras, the option for high numerical precision will become prohibitive. In these cases, our proposed sparsification method may provide a viable alternative for achieving ultra-low-power and high-throughput recognition systems.

There are several directions for future work. First, we can validate the benefits of our sparsification method on learning low-precision versions of state-of-the-art deeper CNNs such as AlexNet (Krizhevsky et al. 2012) and ResNet (He et al. 2016) and using challenging datasets such as ImageNet (Russakovsky et al. 2014) with 1,000 classes. A relevant research question is whether the depth of the network can by itself compensate for learning at low precision. Second, we can investigate the benefits of sparsification for learning binarized CNNs such as the XNOR network (Rastegari et al. 2016) with quantized gradients (Hubara et al. 2016; Miyashita et al. 2016; Zhou et al. 2016) and without the need for storing a double precision representation of weights, making them amenable for on-chip machine learning.

## 5 CONCLUSION

We presented a novel method to incorporate sparse distributed representations across feature channels throughout the hierarchical pipeline in deep learning networks. This method proactively reduces the number of non-zero cell activities throughout the pipeline, leading to significantly reduced energy consumption in transmitting signals between layers and significantly reduced storage requirements for the weights and bias terms at each layer of the deep network. In particular, this method facilitates probabilistic learning of weights and bias terms throughout the network at low numerical precision without compromising recognition performance. Further, this method facilitates more data-efficient learning and identifies the sufficient and necessary subset of feature channels at each layer in the network.

## REFERENCES

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2014. Training deep neural networks with low precision multiplications. (2014). arXiv:arXiv:1412.7024

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. 2015. BinaryConnect: Training deep neural networks with binary weights during propagations. (2015). arXiv:arXiv:1511.00363

Thomas M. Cover and Joy A. Thomas. 2012. *Elements of Information Theory.* John Wiley & Sons, New York, NY.

Richard O. Duda, Peter E. Hart, and David G. Stork. 1973. *Pattern Classification*, Vol. 2. Wiley Press, New York, NY.

David Ferster and Kenneth D. Miller. 2000. Neural mechanisms of orientation selectivity in the visual cortex. *Annu. Rev. Neurosci.* 23, 1 (2000), 441–471.

Benjamin Graham. 2013. Sparse arrays of signatures for online character recognition. (2013). arXiv:arXiv:1308.0371

Benjamin Graham. 2014. Spatially-sparse convolutional neural networks. (2014). arXiv:arXiv:1409.6070

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), 770–778.

Markus Hoehfeld and Scott E Fahlman. 1992. Learning with limited numerical precision using the cascade-correlation algorithm. *IEEE Trans. Neur. Netw.* 3, 4 (1992), 602–611.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Quantized neural networks: Training neural networks with low precision weights and activations. (2016). arXiv:arXiv:1609.07061

Rangachar Kasturi, Dmitry B. Goldgof, Rajmadhan Ekambaram, Gill Pratt, Eric Krotkov, Douglas D. Hackett, Yang Ran, Qinfen Zheng, Rajeev Sharma, Mark Anderson, et al. 2014. Performance evaluation of neuromorphic-vision object recognition algorithms. In *Proceedings of the International Conference on Pattern Recognition* (2014). 2401–2406.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems* (2012). 1097–1105.

Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. 2015. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2015). 806–814.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. 2009. Online dictionary learning for sparse coding. In *Proceedings of the International Conference on Machine Learning* (2009). 689–696.

Alireza Makhzani and Brendan Frey. 2014. $k$-sparse autoencoders. (2014). arXiv:arXiv:1312.5663

Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. 2011. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the International Conference on Artificial Neural Networks* (2011), 52–59.

Daisuke Miyashita, Edward H Lee, and Boris Murmann. 2016. Convolutional neural networks using logarithmic data representation. (2016). arXiv:arXiv:1603.01025

Jim Mutch and David G. Lowe. 2006. Multiclass object recognition with sparse, localized features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2006), 11–18.

Jim Mutch and David G. Lowe. 2008. Object class recognition and localization using sparse features with limited receptive fields. *Int. J. Comput. Vis.* 80, 1 (2008), 45–57.

Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. 2016. XNOR-Net: ImageNet classification using binary convolutional neural networks. (2016). arXiv:arXiv:1603.05279

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, and Michael Bernstein. 2014. ImageNet large scale visual recognition challenge. (2014). arXiv: arXiv:1409.0575

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. (2013). arXiv:arXiv:1312.6229

Thomas Serre, Aude Oliva, and Tomaso Poggio. 2007. A feedforward architecture accounts for rapid categorization. In *Proc. Natl. Acad. Sci. U.S.A.* 104, 15 (2007), 6424–6429.

Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. 2016. Learning structured sparsity in deep neural networks. In *Advances in Neural Information Processing Systems* (2016), 2074–2082.

Sebastien C. Wong, Victor Stamatescu, Adam Gatt, David Kearney, Ivan Lee, and Mark D. McDonnell. 2017. Track everything: Limiting prior knowledge in online multi-object recognition. (2017). arXiv:arXiv:1704.06415

Shuchang Zhou, Zekun Ni, Xinyu Zhou, He Wen, Yuxin Wu, and Yuheng Zou. 2016. Training low bitwidth convolutional neural networks with low bitwidth gradients. (2016). arXiv:arXiv:1606.06160

Received July 2018; revised November 2018; accepted January 2019