Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing

Article *in* Proceedings of the National Academy of Sciences · March 2016

DOI: 10.1073/pnas.1604850113

Convolutional Networks for Fast, Energy-Efficient Neuromorphic Computing

Steven K. Esser, * Paul A. Merolla, * John V. Arthur, * Andrew S. Cassidy, * Rathinakumar Appuswamy, * Alexander Andreopoulos, * David J. Berg, * Jeffrey L. McKinstry, * Timothy Melano, * Davis R. Barch, * Carmelo di Nolfo, * Pallab Datta, * Arnon Amir, * Brian Taba, * Myron D. Flickner, * and Dharmendra S. Modha *

*IBM Research – Almaden

Abstract

Deep networks are now able to achieve human-level performance on a broad spectrum of recognition tasks. Independently, neuromorphic computing has now demonstrated unprecedented energy-efficiency through a new chip architecture based on spiking neurons, low precision synapses, and a scalable communication network. Here, we demonstrate that neuromorphic computing, despite its novel architectural primitives, can implement deep convolution networks that i) approach state-of-the-art classification accuracy across 8 standard datasets, encompassing vision and speech, ii) perform inference while preserving the hardware's underlying energy-efficiency and high throughput, running on the aforementioned datasets at between 1100 and 2300 frames per second and using between 25 and 325 mW (effectively > 5000 frames / sec / W) and iii) can be specified and trained using backpropagation with the same ease-of-use as contemporary deep learning. For the first time, the algorithmic power of deep learning can be merged with the efficiency of neuromorphic processors, bringing the promise of embedded, intelligent, brain-inspired computing one step closer.

The human brain is capable of remarkable acts of perception while consuming very little energy. The dream of brain-inspired computing is to build machines that do the same, requiring high accuracy algorithms as well as efficient hardware to run those algorithms. On the algorithm front, building on the classic work of backpropagation [1] and the neocognitron [2], deep learning has made great strides in achieving human-level performance on a wide range of recognition tasks [3][4][5]. On the hardware front, building on foundational work on silicon neural systems [6], neuromorphic computing, using novel architectural primitives, has recently demonstrated hardware capable of running 1 million neurons and 256 million synapses for extremely low power (just 70mW at real time operation)[7]. Bringing these approaches together holds the promise of a new generation of embedded, real-time systems, but first requires reconciling key differences in the structure and operation between contemporary algorithms and hardware. Here, we introduce and demonstrate an approach we call Eedn, *Energy-efficient deep networks*, which creates convolutional networks whose connections, neurons, and weights have been adapted to run inference tasks on neuromorphic hardware.

For structure, typical convolutional networks place no constraints on filter sizes, whereas neuromorphic systems are constrained to block-wise connectivity that limits filter sizes – thereby saving energy since weights can now be stored in local on-chip memory within dedicated neural cores. Here, we present a new convolutional network structure that naturally maps to the efficient connection primitives used in contemporary neuromorphic systems. We enforce this connectivity constraint by partitioning filters into multiple groups, and yet maintain network integration by interspersing layers whose filter support region, by using a small topographic size, is able to cover incoming features from many groups [8].

For operation, contemporary convolutional networks typically use high precision ($\geq$ 32-bit) neurons and synapses to provide continuous derivatives and support small incremental changes to network state, both formally required for backpropagation-based gradient learning. In comparison, neuromorphic designs use 1-bit *spikes* to provide event-based computation and communication (consuming energy only when necessary) and use *low-precision synapses* to co-locate memory with computation (keeping data movement local and avoiding off-chip memory bottlenecks). Here, we demonstrate that by introducing two constraints into the learning rule – binary-valued neurons with approximate derivatives and trinary-valued ($\{-1, 0, 1\}$) synapses – it is possible to adapt backpropagation to create networks directly implementable using energy efficient neuromorphic dynamics. This draws inspiration from the spiking neurons and low-precision synapses of the brain [9], and builds upon work showing that deep learning can create networks with low-precision synapses [10] [11], neurons [12][13][14], or both [15] [16]. For input data, we employ a first layer to transform multi-valued, multi-channel input into (up to) 16 binary channels using convolution filters that are learned via backpropagation [12][16] and whose output can be sent on chip in the form of spikes. These binary channels, intuitively akin to independent components [17] learned with supervision, provide a parallel distributed representation to carry out high-fidelity, fault-tolerant computation without the need for high-precision representation.

Critically, we demonstrate that bringing the above innovations together allows us to create networks that approach state-of-the-art accuracy performing inference on 8 standard datasets, running on a neuromorphic chip at between 1100 and 2300 frames per second (FPS), using between 25 and 325 mW. We further explore how our approach scales by simulating multi-chip configurations. Ease-of-use is achieved using training tools built from existing, optimized deep learning frameworks [18], with learned parameters mapped to hardware using a high level deployment language [19]. While we choose the IBM TrueNorth chip [7] for our example deployment platform, the essence of our constructions can apply to other emerging neuromorphic approaches [20][21][22][23] and may lead to new architectures that incorporate deep learning and efficient hardware primitives from the ground up.

Approach

Here, we provide a description of the relevant elements of deep convolutional networks and the TrueNorth neuromorphic chip, and describe how the essence of the former can be realized on the latter.

Deep Convolutional Networks. A deep convolutional network is a multilayer feedforward neural network, whose input is typically image-like and whose layers are neurons that collectively perform a convolutional filtering of the input or a prior layer (Figure 1). Neurons within a layer are arranged in two spatial dimensions, corresponding to shifts in the convolution filter, and one feature dimension, corresponding to different filters.

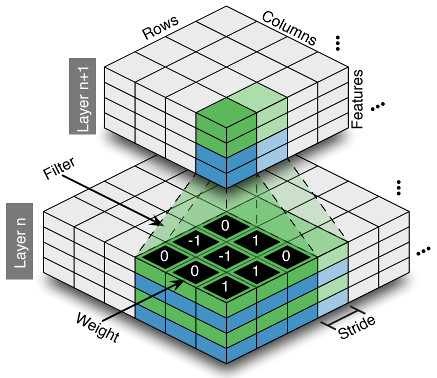

Fig. 1. Two layers of a convolutional network, where each layer is a rows × columns × features collection of outputs from filters applied to the prior layer. Each output neuron has a topographically aligned filter support region in its source layer. Adjacent features have their receptive field shifted by the stride in the source layer. A layer can be divided into multiple groups along the feature dimension, where each group has a filter support region that covers a different set of features in the source layer. Two groups are highlighted (green, blue).

Each neuron computes a summed weighted input, s, as

$$s = \sum_{i,j} \sum_{f} x_{i,j,f} w_{i,j,f},$$

where $\mathbf{x} = \{x_{i,j,f}\}$ are the neuron's input pixels or neurons, $\mathbf{w} = \{w_{i,j,f}\}$ are the filter weights, i, j are over the topographic dimensions, and f is over the feature dimension or input channels. Batch normalization [24] can be used to zero center s and normalize its standard deviation to 1, following

$$r = \frac{s - \mu}{\sigma + \epsilon} + b \qquad [1]$$

where r is the filter response, b is a bias term, $\epsilon = 10^{-4}$ provides numerical stability, and μ and σ are the mean and standard deviation of s computed per filter using all topographic locations and examples in a data batch during training, or using the entire training set during inference. Final neuron output is computed by applying a non-linear activation function to the filter response, typically a rectified linear unit that sets negative values to 0 [25]. In a common scheme, features in the last layer are each assigned a label – such as prediction class – and vote to formulate network output [8].

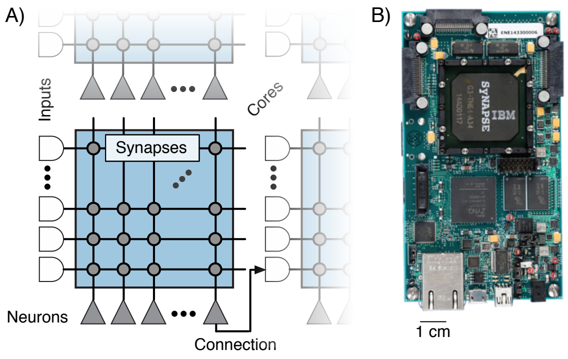

Fig. 2. A) The TrueNorth architecture, a multicore array where each core consists of 256 input lines, 256 neurons, and a 256 × 256 synaptic crossbar array. Each neuron can connect to 1 input on 1 core through a spike router, and can connect to any neuron on the target core through the crossbar, thereby resulting in block-wise connectivity. B) IBM's NS1e board, with 1 TrueNorth chip, comprising 4096 cores, with a total of 1 Million neurons and 256 Million synapses.

Deep networks are trained using the backpropagation learning rule [1]. This procedure involves iteratively i) computing the network's response to a batch of training examples in a *forward pass*, ii) computing the error between the network's output and the desired output, iii) using the chain rule to compute the error gradient at each synapse in a *backward pass*, and iv) making a small change to each weight along this gradient so as to reduce error.

TrueNorth. A TrueNorth chip consists of a network of neurosynaptic cores with programmable connectivity, synapses, and neuron parameters (Figure 2). Connectivity between neurons follows a block-wise scheme: each neuron can connect to an input line of any one core in the system, and from there to any neuron on that core through its local synapses. All communication to-, from-, and within- chip is performed using spikes.

TrueNorth neurons use a variant of an integrate-and-fire model with 23 configurable parameters [26] where a neuron's state variable, $V(t)$, updates each tick, t – typically at 1000 ticks per second, though higher rates are possible – according to

$$V(t+1) = V(t) + \sum_i \hat{x}_i(t) w_i + L, \qquad [2]$$

where $\hat{\boldsymbol{x}}(t) = \{\hat{x}_i\}$ are the neuron's spiking inputs, $\mathbf{w} = \{w_i\}$ are its corresponding weights, L is its leak chosen from $\{-255, -254, ..., 255\}$, and i is over its inputs. If $V(t)$ is greater than or equal to a threshold θ, the neuron emits a spike and resets using one of several reset modes, including resetting to 0. If $V(t)$ is below a lower bound, it can be configured to snap to that bound.

Synapses have individually configurable on/off states and have a strength assigned by look-up table. Specifically, each neuron has a 4-entry table parameterized with values in the range $\{-255, -254, ..., 255\}$, each input line to a core is assigned an input type of 1, 2, 3 or 4, and each synapse then determines its strength by using the input type on its source side to index into the table of the neuron on its target side.[1] In this work, we only use 2 input types, corresponding to synapse strengths of -1 and 1, described in the next section.

Mapping Deep Convolutional Networks to TrueNorth. By appropriately designing the structure, neurons, network input, and weights of convolutional networks during training, it is possible to efficiently map those networks to neuromorphic hardware.

Structure

Network structure is mapped by partitioning each layer into 1 or more equally sized groups along the feature dimension,[2] where each group applies its filters to a different, non-overlapping, equally sized subset of layer input features. Layers are designed such that the total filter size (rows × columns × features) of each group is less than or equal to the number of input lines available per core, and the number of output features is less than or equal to the number of neurons per core. This arrangement allows 1 group's features, filters, and filter support region to be implemented using 1 core's neurons, synapses, and input lines, respectively (Figure 3A).

[1]It should be noted that our approach can easily be adapted to hardware with other synaptic representation schemes.

[2]Feature groups were originally used by AlexNet[25], which split the network to run on 2 parallel GPUs during training. The use of grouping is expanded upon considerably in this work.

[3]Schemes that use higher precision are possible, such as using the number of spikes generated in a given time window to represent data (a rate code). However, we observed the best accuracy for a given energy budget by using the binary scheme described here.

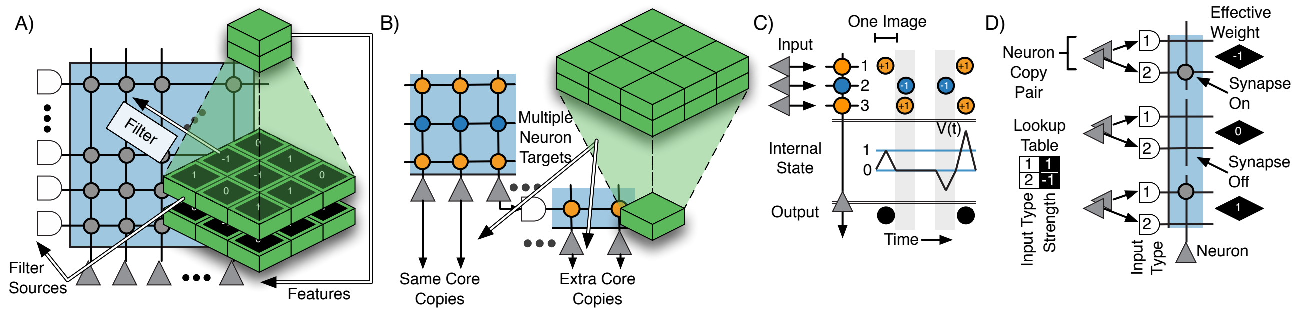

Fig. 3. Mapping of a convolutional network to TrueNorth. A) Convolutional network features for 1 group at 1 topographic location are implemented using neurons on the same TrueNorth core, with their corresponding filter support region implemented using the core's input lines, and filter weights implemented using the core's synaptic array. B) For a neuron to target multiple core inputs, its output must be replicated by neuron copies, recruited from other neurons on the same core, or on extra cores if needed. C) In each tick, the internal state variable, V(t), of a TrueNorth neuron increases in response to positive weighted inputs (light orange circles) and decreases in response to negative weighted inputs (dark blue circles). Following integration of all inputs for a tick, if V(t) is greater than or equal to the threshold $\theta = 1$, a spike is emitted and V(t) is reset to 0, while if V(t) is below 0, V(t) is reset to 0 (without producing a spike). D) Convolutional network filter weights (numbers in black diamonds) implemented using TrueNorth. The TrueNorth architecture supports weights with individually configured on/off state and strength assigned using a lookup table. In our scheme, each feature is represented with pairs of neuron copies. Each pair connects to 2 inputs on the same target core, with the inputs assigned types 1 and 2, which via the look up table assign strengths of $+1$ or -1 to synapses on the corresponding input lines. By turning on the appropriate synapses, each synapse pair can be used to represent -1, 0, or $+1$.

Total filter size was further limited to 128 here, to support trinary synapses, described below. For efficiency, multiple topographic locations for the same group can be implemented on the same core. For example, by delivering a $4 \times 4 \times 8$ region of the input space to a single core, that core can be used to implement overlapping filters of size $3 \times 3 \times 8$ for 4 topographic locations.

Where filters implemented on different cores are applied to overlapping regions of the input space, the corresponding input neurons must target multiple cores, which is not explicitly supported by TrueNorth. In such instances, multiple neurons on the same core are configured with identical synapses and parameters (and thus will have matching output), allowing distribution of the same data to multiple targets. If insufficient neurons are available on the same core, a feature can be "split" by connecting it to a core with multiple neurons configured to spike whenever they receive an input spike from that feature. Neurons used in either duplication scheme are referred to here as *copies* (Figure 3B).

Neurons

To match the use of spikes in hardware, we employ a binary representation scheme for data throughout the network.[3] Neurons in the convolutional network use the activation function

$$y = \begin{cases} 1 & \text{if } r \geq 0, \\ 0 & \text{otherwise.} \end{cases} \qquad [3]$$

where y is neuron output and r is the neuron filter response (Equation 1). By configuring TrueNorth neurons such that i) $L = \lceil b(\sigma + \epsilon) - \mu \rceil$, where L is the leak from Equation 2 and the remaining variables are the normalization terms from Equation 1, which are computed from training data offline, ii) threshold (θ in Equation 2) is 1, iii) reset is to 0 after spiking, and iv) the lower bound on the membrane potential is 0, their behavior exactly matches that in Equation 3 (Figure 3C). Conditions iii and iv ensure that $V(t)$ is 0 at the beginning of each image presentation, allowing for 1 classification per tick using pipelining.

Network input

Network inputs are typically represented with multi-bit channels (for example, 8-bit RGB channels). Directly converting the state of each bit into a spike would result in an unnatural neural encoding since each bit represents a different value (for example, the most-significant-bit spike would carry a weight of 128 in an 8-bit scheme). Here, we avoid this awkward encoding altogether by converting the high precision input into a spiking representation using convolution filters with the binary output activation function described in Equation 3. This process is akin to the transduction that takes place in biological sensory organs, such as the conversion of brightness levels into single spikes representing spatial luminance gradients in the retina.

Weights

While TrueNorth does not directly support trinary weights, they can be simulated by using neuron copies such that a feature's output is delivered in pairs to its target cores. One member of the pair is assigned input type 1, which corresponds to a $+1$ in every neuron's lookup table, and the second input type 2, which corresponds to a -1. By turning on neither, one, or the other of the corresponding synaptic connections, a weight of 0, $+1$ or -1 can be created (Figure 3D). To allow us to map into this representation, we restrict synaptic weights in the convolutional network to these same trinary values.

Training

Constraints on receptive field size and features per group, the use of binary neurons and use of trinary weights are all employed during training. As the binary-valued neuron used here has a derivative of ∞ at 0, and 0 everywhere else, which is not amenable to backpropagation, we instead approximate its derivative as being 1 at 0 and linearly decaying to 0 in the positive and negative direction according to

$$\frac{\partial y}{\partial r} \approx \max(0, 1 - |r|),$$

[4] This is rule is similar to the recent results from BinaryNet [16], but was developed independently here in this work. Our specific neuron derivative and use of hysteresis are unique.

where r is the filter response and y is the neuron output. Weight updates are applied to a high precision hidden value, w_h, which is bounded in the range -1 to 1 by clipping, and mapped to the trinary value used for the forward and backward pass by rounding with hysteresis according to

$$w(t) = \begin{cases} -1 & \text{if } w_h(t) \leq -0.5 - h, \\ 0 & \text{if } w_h(t) \geq -0.5 + h \vee w_h(t) \leq 0.5 - h, \\ 1 & \text{if } w_h(t) \geq 0.5 + h, \\ w(t-1) & \text{otherwise,} \end{cases}$$

where h is a hysteresis parameter set to 0.1 here.[4] The hidden weights allows each synapse to flip between discrete states based on subtle differences in the relative amplitude of error gradients measured across multiple training batches.

We employ standard heuristics for training, including weight decay (10^{-6}), momentum (0.9), and decreasing learning rate (dropping by $10\times$ twice during training). Dropout layers [27] were used during learning, but not deployment. Training was performed offline on conventional GPUs, using a library of custom training layers built upon functions from the MatConvNet toolbox [18]. Network specification and training complexity using these layers is on par with standard deep learning.

Deployment

The parameters learned through training are mapped to hardware using reusable, composable hardware description functions called *corelets* [19]. The corelets created for this work automatically compile the learned network parameters, which are independent of any neuromorphic platform, into an platform-specific hardware configuration file that can directly program TrueNorth chips.

Results

We applied our approach to 8 image and audio benchmarks (Table 2 and Figure 4) by creating 4 template networks tuned to maximize accuracy on CIFAR10 using 0.5, 1, 4 or 8 TrueNorth chips (Table 1), and then applied to the remaining datasets.[5] Testing was performed at 1 classification per hardware tick.

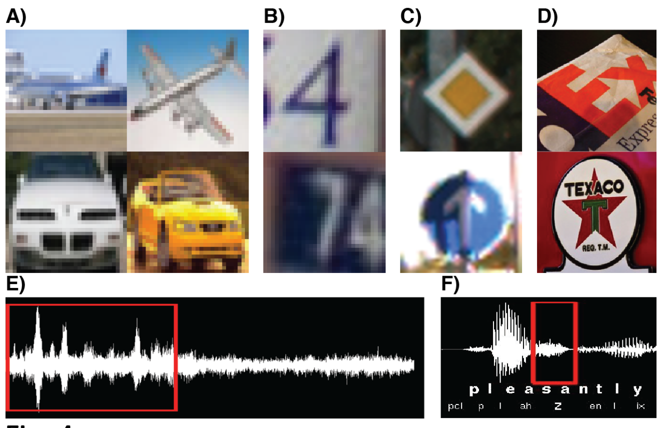

Fig. 4. Dataset samples. A) CIFAR10 examples of airplane and automobile. B) SVHN examples of the digits '4' and '7'. C) GTSRB examples of the German traffic signs for 'priority road' and 'ahead only'. D) Flickr-Logos32 examples of corporate logos for 'FedEx' and 'Texaco'. E) VAD example showing voice activity (red box) and no voice activity at 0 dB SNR. F) TIMIT examples of the phonemes 'pcl', 'p', 'l', 'ah', 'z' (red box), 'en', 'l', 'ix'.

Table 1. Structure of convolution networks used in this work. Each layer is described as "type-features(groups)", where types are indicated with "S" for spatial filter layers with filter size 3×3 and stride 1, "N" for network-in-network layers with filter size 1×1 and stride 1, "P" for convolutional pooling layer with filter size 2×2 and stride 2, and "P4" for convolutional pooling layer with filter size 4×4 and stride 2, and "D" for dropout layers. The number of output features assigned to each of the 10 CIFAR10 classes is indicated below the final layer as "(features/class)"; this ratio varies for datasets with a different class count.

1/2 Chip	1 Chip	4 Chip	8 Chip
S-12	S-12	S-16	S-16
P4-128(4)	P4-252(2)	S-256(2)	S-512(4)
D	N-256(2)	N-256(2)	N-512(4)
S-256(16)	P-256(8)	N-128(2)	N-256(4)
N-256(2)	D	P-128(4)	D
P-512(16)	S-512(32)	S-512(16)	P-256(8)
S-1020(4)	N-512(4)	N-512(4)	S-256(32)
(102/class)	N-512(4)	N-256(4)	S-1024(32)
	N-512(4)	D	N-1024(8)
	P-512(16)	P-256(8)	N-512(8)
	D	S-1024(32)	D
	S-1024(64)	N-1024(8)	P-512(16)
	N-1024(8)	N-512(8)	S-512(64)
	P-1024(32)	P-512(16)	S-2048(64)
	N-2040(8)	S-2048(64)	N-2048(16)
	(204/class)	N-2048(64)	N-512(16)
		N-2048(16)	D
		N-1024(32)	S-2048(64)
		D	N-2048(16)
		S-2048(128)	D
		N-2048(16)	S-4080(16)
		N-2048(16)	(408/class)
		N-4080(16)	
		(408/class)	

Networks. Three layer configurations were especially useful in this work, though our approach supports a variety of other parameterizations. First, spatial filter layers employ patch size $3 \times 3 \times 8$ and stride 1, allowing placement of 4 topographic locations per core. Second, network-in-network layers (see [8]) employ patch size $1 \times 1 \times 128$ and stride of 1, allowing each filter to span a large portion of the incoming feature space, thereby helping to maintain network integration. Finally, pooling layers employ standard convolution layers [28] with patch size $2 \times 2 \times 32$ and stride 2, thereby resulting in non-overlapping patches that reduce the need for neuron copies.

Networks were adjusted for dataset only for input size or to remove dropout layers as dictated by performance. We found that using 12 channels for the transduction layer (Figure 5) gave good performance at a low bandwidth. For multi-chip networks we used 16 channels in the transduction layer.

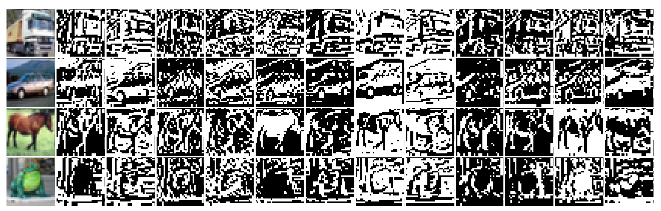

Fig. 5. Each row shows an example image from CIFAR10 (column 1) and the corresponding output of 12 transduction filters (columns $2-13$).

[5] Additional network sizes for the audio datasets (VAD, TIMIT classification, TIMIT frames) were created by adjusting features per layer or removing layers.

[6] Active energy per classification does not change as the chip's tick runs faster or slower as long as the voltage is the same (as in the experiments here) because the same number of transistors switch independent of the tick duration.

Table 2. Summary of datasets. GTSRB and Flickr-Logos32 are cropped and/or downsampled from larger images. VAD and TIMIT datasets have Mel-frequency cepstral coefficients (MFCC) computed from 16kHz audio data.

Dataset	Classes	Input	Description
CIFAR10 [30]	10	32 row × 32 column × 3 RGB	Natural and manufactured objects in their environment.
CIFAR100 [30]	100	32 row × 32 column × 3 RGB	Natural and manufactured objects in their environment.
SVHN [31]	10	32 row × 32 column × 3 RGB	Single digits of house addresses from Google's Street View.
GTSRB [32]	43	32 row × 32 column × 3 RGB	German traffic signs in multiple environments.
Flickr-Logos32 [33]	32	32 row × 32 column × 3 RGB	Localized Corporate logos in their environment.
VAD [34][35]	2	16 sample × 26 MFCC	Voice activity present or absent, with noise (TIMIT + NOISEX).
TIMIT Class. [34]	39	32 sample × 16 MFCC × 3 delta	Phonemes from English speakers, with phoneme boundaries.
TIMIT Frame [34]	39	16 sample × 39 MFCC	Phonemes from English speakers, without phoneme boundaries.

Table 3. Summary of results. TrueNorth network sizes refer to chip count of network running CIFAR10. Individual networks may vary according to data size. Power is measured in milliWatts (mW). FPS = Frames/Sec. FPS/W = Frames/Sec/Watt. CNN = Convolutional Neural Network. MLP = Multilayer Perceptron. HGMM = Hierarchical Gaussian Mixture Model. BLSTM = Bidirectional Long Short-Term Memory.

Dataset	State of the Art		TrueNorth Best Accuracy		TrueNorth 1 Chip				
	Approach	Accuracy	Accuracy	#cores	Accuracy	#cores	FPS	mW	FPS/W
CIFAR10	CNN[11]	91.73%	**87.50%**	31872	**82.50%**	3978	1191	226.4	5261.9
CIFAR100	CNN[36]	65.43%	**63.05%**	31843	**52.73%**	3978	1679	255.9	6560.0
SVHN	CNN[36]	98.08%	**97.17%**	31843	**96.36%**	3978	2128	323.0	6587.3
GTSRB	CNN[37]	99.46%	**96.21%**	32867	**96.87%**	3978	1357	241.5	5620.7
LOGO32	CNN*	93.70%	**87.51%**	3220	**87.51%**	3220	1830	224.9	8134.7
VAD	MLP[38]	95.00%	**97.00%**	1758	**95.42%**	423	1539	26.1	59010.7
TIMIT Class.	HGMM[39]	83.30%	**81.49%**	33542	**78.39%**	2179	2233	183.3	12180.0
TIMIT Frames	BLSTM[40]	72.10%	**71.93%**	19168	**67.75%**	2444	1989	183.5	10838.7

*Unpublished internal implementation.

Hardware. To characterize performance, all networks that fit on a single chip were run in TrueNorth hardware. Multi-chip networks were run in simulation [29], pending forthcoming infrastructure for interconnecting chips. Single-chip classification accuracy and throughput were measured on the NS1e development board (Figure 2B), but power was measured on a separate NS1t test and characterization board (not shown) – using the same supply voltage of 1.0V on both boards – since the current NS1e board is not instrumented to measure power and the NS1t board is not designed for high throughput. Total TrueNorth power is the sum of i) *leakage power*, computed by measuring idle power on NS1t and scaling by the fraction of the chip's cores used by the network, and ii) *active power*, computed by measuring total power during classification on NS1t, subtracting idle power, and scaling by the classification throughput (FPS) measured on NS1e.[6] For hardware measurement, our focus was to characterize operation on the TrueNorth chip as a component in a future embedded system. Such a system will also need to consider capabilities and energy requirements of sensors, transduction, and off-chip communication, which requires hardware choices that are application specific and are not considered here.

Performance. Table 3 and Figure 6 show our results for all 8 datasets and a comparison with state-of-the-art approaches, with measured power and classifications per energy (Frames/Sec/Watt) reported for single-chip networks. Our experiments show that for almost all of the benchmarks, a single-chip network is sufficient to come within a few percent of state-of-the-art accuracy. Increasing to up to 8 chips improved accuracy by several percentage points, and in the case of the VAD dataset surpassed state-of-the-art performance.

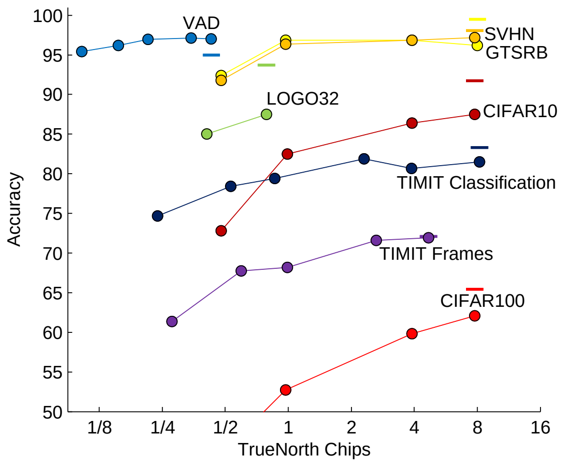

Fig. 6. Accuracy of different sized networks running on one or more TrueNorth chips to perform inference on 8 datasets. For comparison, accuracy of state-of-the-art unconstrained approaches are shown as bold horizontal lines (hardware resources used for these networks are not indicated).

Discussion

Our work demonstrates that the structural and operational differences between neuromorphic computing and deep learning are not fundamental, and points to the richness of neural network constructs and the adaptability of backpropagation.

This marks an important step towards a new generation of applications based on embedded neural networks.

These results help to validate the neuromorphic approach, which is to provide an efficient yet flexible substrate for spiking neural networks, instead of targeting a single application or network structure. Indeed, the specification for TrueNorth and a prototype chip [41] were developed in 2011, before the recent resurgence of convolutional networks in 2012 [25]. Not only is TrueNorth capable of implementing these convolutional networks, which it was not originally designed for, but it also supports a variety of connectivity patterns (feedback and lateral, as well as feedforward) and can simultaneously implement a wide range of other algorithms (see [7][13][15][42][43][44]). We envision running multiple networks on the same TrueNorth chip, enabling composition of end-to-end systems encompassing saliency, classification, and working memory.

We see several avenues of potentially fruitful exploration for future work. Several recent innovations in unconstrained deep learning that may be of value for the neuromorphic domain include deeply supervised networks [36], and modified gradient optimization rules. The approach used here applies hardware constraints from the beginning of training, that is, *constrain-then-train*, but innovation may also come from *constrain-while-train* approaches, where training initially begins in an unconstrained space, but constraints are gradually introduced during training [12]. Finally, co-design between algorithms and future neuromorphic architectures promises even better accuracy and efficiency.

ACKNOWLEDGMENTS. This research was sponsored by the Defense Advanced Research Projects Agency under contract No. FA9453-15-C-0055. The views, opinions, and/or findings contained in this paper are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government. We acknowledge Rodrigo Alvarez-Icaza for support with hardware infrastructure.

1. D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, pp. 533–536, 1986.
2. K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," Biological Cybernetics, vol. 36, no. 4, pp. 193–202, 1980.
3. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.
4. S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in Advances in Neural Information Processing Systems, 2015, pp. 91–99.
5. D. Cireşan, A. Giusti, L. M. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images," in Advances in neural information processing systems, 2012, pp. 2843–2851.
6. C. Mead, "Neuromorphic electronic systems," Proceedings of the IEEE, vol. 78, no. 10, pp. 1629–1636, 1990.
7. P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura et al., "A million spiking-neuron integrated circuit with a scalable communication network and interface," Science, vol. 345, no. 6197, pp. 668–673, 2014.
8. M. Lin, Q. Chen, and S. Yan, "Network in network," In ICLR, 2014. [Online]. Available: http://arxiv.org/abs/1312.4400
9. T. M. Bartol, C. Bromer, J. Kinney, M. A. Chirillo, J. N. Bourne, K. M. Harris, and T. J. Sejnowski, "Nanoconnectomic upper bound on the variability of synaptic plasticity," eLife, vol. 4, p. e10778, 2016.
10. E. Stromatias, D. Neil, M. Pfeiffer, F. Galluppi, S. B. Furber, and S.-C. Liu, "Robustness of spiking deep belief networks to noise and reduced bit precision of neuro-inspired hardware platforms," Frontiers in neuroscience, vol. 9, 2015.
11. 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, pp. 3105–3113.
12. Z. Wu, D. Lin, and X. Tang, "Adjustable bounded rectifiers: Towards deep binary representations," arXiv preprint arXiv:1511.06201, 2015.
13. P. U. Diehl, B. U. Pedroni, A. Cassidy, P. Merolla, E. Neftci, and G. Zarrella, "Truehappiness: Neuromorphic emotion recognition on truenorth," arXiv preprint arXiv:1601.04183, 2016.
14. S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.
15. S. K. Esser, R. Appuswamy, P. Merolla, J. V. Arthur, and D. S. Modha, "Backpropagation for energy-efficient neuromorphic computing," in Advances in Neural Information Processing Systems, 2015, pp. 1117–1125.
16. M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1," arXiv preprint arXiv:1602.02830, 2016.
17. A. J. Bell and T. J. Sejnowski, "The independent components of natural scenes are edge filters," Vision research, vol. 37, no. 23, pp. 3327–3338, 1997.
18. A. Vedaldi and K. Lenc, "MatConvNet – convolutional neural networks for MATLAB," 2015.
19. A. Amir, P. Datta, W. P. Risk, A. S. Cassidy, J. A. Kusnitz, S. K. Esser, A. Andreopoulos, T. M. Wong, M. Flickner, R. Alvarez-Icaza et al., "Cognitive computing programming paradigm: a corelet language for composing networks of neurosynaptic cores," in Neural Networks (IJCNN), The 2013 International Joint Conference on. IEEE, 2013, pp. 1–10.
20. E. Painkras, L. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, S. B. Furber et al., "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," Solid-State Circuits, IEEE Journal of, vol. 48, no. 8, pp. 1943–1953, 2013.
21. T. Pfeil, A. Grübl, S. Jeltsch, E. Müller, P. Müller, M. A. Petrovici, M. Schmuker, D. Brüderle, J. Schemmel, and K. Meier, "Six networks on a universal neuromorphic computing substrate," Frontiers in Neuroscience, vol. 7, 2013.
22. S. Moradi and G. Indiveri, "An event-based neural network architecture with an asynchronous programmable synaptic memory," Biomedical Circuits and Systems, IEEE Transactions on, vol. 8, no. 1, pp. 98–107, 2014.
23. J. Park, S. Ha, T. Yu, E. Neftci, and G. Cauwenberghs, "A 65k-neuron 73-mevents/s 22-pj/event asynchronous micro-pipelined integrate-and-fire array transceiver," in Biomedical Circuits and Systems Conference (BioCAS), 2014 IEEE. IEEE, 2014, pp. 675–678.
24. S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," arXiv preprint arXiv:1502.03167, 2015.
25. A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.
26. A. S. Cassidy, P. Merolla, J. V. Arthur, S. K. Esser, B. Jackson, R. Alvarez-Icaza, P. Datta, J. Sawada, T. M. Wong, V. Feldman et al., "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores," in Neural Networks (IJCNN), The 2013 International Joint Conference on. IEEE, 2013, pp. 1–10.
27. G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," arXiv preprint arXiv:1207.0580, 2012.
28. J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," arXiv preprint arXiv:1412.6806, 2014.
29. R. Preissl, T. M. Wong, P. Datta, M. Flickner, R. Singh, S. K. Esser, W. P. Risk, H. D. Simon, and D. S. Modha, "Compass: A scalable simulator for an architecture for cognitive computing," in Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE Computer Society Press, 2012, p. 54.
30. A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep., 2009.
31. Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011. [Online]. Available: http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
32. J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel, "Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition," Neural networks, vol. 32, pp. 323–332, 2012.
33. S. Romberg, L. G. Pueyo, R. Lienhart, and R. V. Zwol, "Scalable logo recognition in real-world images," in Proceedings of the 1st ACM International Conference on Multimedia Retrieval. ACM, 2011, p. 25.
34. J. Garofolo, L. Lamel, W. Fisher, J. Fiscus, D. Pallett, N. Dahlgren, and V. Zue, "TIMIT Acoustic-Phonetic Continuous Speech Corpus LDC93S1," Philadelphia: Linguistic Data Consortium, 1993.
35. A. Varga and H. J. M. Steeneken, "Assessment for Automatic Speech Recognition II: NOISEX-92: A Database and an Experiment to Study the Effect of Additive Noise on Speech Recognition Systems," Speech Communications, vol. 12, no. 3, pp. 247–251, Jul. 1993.
36. C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," arXiv preprint arXiv:1409.5185, 2014.
37. D. Cireşan, U. Meier, J. Masci, and J. Schmidhuber, "Multi-column deep neural network for traffic sign classification." Neural networks, vol. 32, pp. 333–338, Aug. 2012.
38. T. V. Pham, C. T. Tang, and M. Stadtschnitzer, "Using artificial neural network for robust voice activity detection under adverse conditions," in International Conference on Computing and Communication Technologies. IEEE, 2009, pp. 1–8.

39. H.-A. Chang and J. R. Glass, "Hierarchical large-margin gaussian mixture models for phonetic classification," in IEEE Workshop on Automatic Speech Recognition and Understanding, 2007.
40. A. Graves, N. Jaitly, and A. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in IEEE Workshop on Automatic Speech Recognition and Understanding, 2013, pp. 273–278.
41. P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm," in IEEE Custom Integrated Circuits Conference (CICC), Sept. 2011, pp. 1–4.
42. S. K. Esser, A. Andreopoulos, R. Appuswamy, P. Datta, D. Barch, A. Amir, J. Arthur, A. Cassidy, M. Flickner, P. Merolla et al., "Cognitive computing systems: Algorithms and applications for networks of neurosynaptic cores," in Neural Networks (IJCNN), The 2013 International Joint Conference on. IEEE, 2013, pp. 1–10.
43. P. U. Diehl, G. Zarrella, A. Cassidy, B. U. Pedroni, and E. Neftci, "Conversion of artificial recurrent neural networks to spiking neural networks for low-power neuromorphic hardware," arXiv preprint arXiv:1601.04187, 2016.
44. S. Das, B. U. Pedroni, P. Merolla, J. Arthur, A. S. Cassidy, B. L. Jackson, D. Modha, G. Cauwenberghs, and K. Kreutz-Delgado, "Gibbs sampling with low-power spiking digital neurons," IEEE International Symposium on Circuits and Systems, 2015.