Int J Comput Vis (2015) 113:54–66
DOI 10.1007/s11263-014-0788-3

Spiking Deep Convolutional Neural Networks for Energy-Efficient Object Recognition

Yongqiang Cao · Yang Chen · Deepak Khosla

Received: 9 February 2014 / Accepted: 10 November 2014 / Published online: 23 November 2014

Abstract Deep-learning neural networks such as convolutional neural network (CNN) have shown great potential as a solution for difficult vision problems, such as object recognition. Spiking neural networks (SNN)-based architectures have shown great potential as a solution for realizing ultra-low power consumption using spike-based neuromorphic hardware. This work describes a novel approach for converting a deep CNN into a SNN that enables mapping CNN to spike-based hardware architectures. Our approach first tailors the CNN architecture to fit the requirements of SNN, then trains the tailored CNN in the same way as one would with CNN, and finally applies the learned network weights to an SNN architecture derived from the tailored CNN. We evaluate the resulting SNN on publicly available Defense Advanced Research Projects Agency (DARPA) Neovision2 Tower and CIFAR-10 datasets and show similar object recognition accuracy as the original CNN. Our SNN implementation is amenable to direct mapping to spike-based neuromorphic hardware, such as the ones being developed under the DARPA SyNAPSE program. Our hardware mapping analysis suggests that SNN implementation on such spike-based hardware is two orders of magnitude more energy-efficient than the original CNN implementation on off-the-shelf FPGA-based hardware.

Keywords Deep learning · Machine learning · Convolutional neural networks · Spiking neural networks · Neuromorphic circuits · Object recognition

Communicated by Marc'Aurelio Ranzato, Geoffrey E. Hinton, and Yann LeCun.

Y. Cao · Y. Chen (✉) · D. Khosla
HRL Laboratories, LLC, 3011 Malibu Canyon Road,
Malibu, CA 90265-4797, USA
e-mail: ychen@hrl.com

1 Introduction

Convolutional neural network (CNN) is a supervised deep-learning neural network with multiple layers of similarly structured convolutional feature extraction operations followed by a linear neural network (NN) classifier. Each of the convolution layers is composed of linear filtering (convolution), non-linearity and feature pooling stages (LeCun et al. 1998). The linear NN classifier is a fully connected NN, and can have one or more hidden layers itself.

Lately, deep-learning neural networks, especially CNN, have shown great performance advantages over other machine learning approaches in vision tasks, winning several high-profile machine learning competitions. For example, a team in University of Toronto Deep Learning Group won the ILSVRC-2012 competition in the ImageNet Large Scale Visual Recognition Challenge (Krizhevsky et al. 2012); another team in the same group won the Merck Drug Discovery Competition; Ciresan et al. (2012) won the final phase of the German traffic sign recognition benchmark and achieved a better-than-human recognition rate of 99.46 %. More recently, with the public release of OverFeat (Sermanet et al. 2014), a deep and large CNN trained using 15 million ImageNet images, researchers have suggested the end of "man-made" features (such as SIFT or HOG) and instead to use off-the-shelf OverFeat features for all image recongnition and scene understanding tasks (Razavian et al. 2014). A CNN implementation based on work by Farabet et al. (2010) was also used by the HRL Laboratories, LLC team (Khosla et al. 2013) in the Neovision2 program with superior object recognition performance over other neuromorphic and computer vision approaches (DARPA 2011).

Deep-learning neural networks have their roots in the famous discovery of Hubel and Wiesel (1959, 1962) that concerns information processing by simple and complex

cells in the primary visual cortex. Inspired by the Hubel and Wiesel model, Fukushima developed Neocognitron in 1980 (Fukushima 1980, 1988). Neocognitron is one of the first networks consisting of many layers of cells that build up an increasingly complex and invariant object representation in hierarchical stages from lower levels. LeCun et al. (1989) introduced CNN by first applying supervised backpropagation (Rumelhart et al. 1986) to such multi-layer hierarchical architectures. Riesenhuber and Poggio (1999) developed HMAX, an unsupervised feedforward hierarchical architecture based also on the Hubel and Wiesel model (see also Serre et al. 2007). Then ARTSCAN, ARTSCENE and pARTSCAN models (Fazl et al. 2009; Grossberg and Huang 2009; Cao et al. 2011; Grossberg et al. 2011) were developed by Grossberg and his colleagues to propose how attention can enable autonomous object learning and invariant representation on visual cortex in a multilayer hierarchical neural network with feedback. Hinton and his colleagues made a breakthrough in 2006 (Hinton and Salakhutdinov 2006; Hinton et al. 2006a, b) by showing how a multilayer neural network could be effectively pre-trained as an unsupervised restricted Boltzmann machine (RBM) one layer at a time; then using supervised backpropagation to fine-tune the weights. Hinton's work was quickly followed and extended by others (e.g., Bengio 2009; Ranzato et al. 2007 and Jarrett et al. 2009).

CNN is a *rate-based* neural network, meaning that it is suitable for implementation on conventional CPUs with significant numerical processing capabilities. While accurate and suitable for difficult vision problems, conventional CNN algorithms are becoming increasingly complex and often require more powerful computing platforms (e.g., GPUs and FPGAs) to implement, thus limiting their practical applications. CNN-like neuromorphic architectures that use *spike-based* computation and communication (e.g., spiking neurons and synapses) are increasingly desirable to take advantage of the emerging class of ultra-low power spike-based hardware architectures (Cruz-Albrecht et al. 2012; Merolla et al. 2011). We hereafter refer to spiking CNNs as SNNs.

There is no prior art per se for converting CNNs to SNNs. Cao and Grossberg (2012) showed how a rate-based neural network model for 3D boundary and surface perception can be converted to a spike-based neural network model, without loss of performance. However, their approach does not teach us how to convert a rate-based supervised NN into spike-based implementation. The closest prior work are either spike-based HMAX-like methods (Folowosele et al. 2011; Masquelier and Thorpe 2007), or the frameless spike-based convolution network by Perez-Carrasco et al. (2010). The spike-based HMAX-like method of Folowosele et al. (2011) is a rudimentary implementation of the original HMAX algorithm (Riesenhuber and Poggio 1999; Serre et al. 2007), and does not teach us a principled approach for conversion of general HMAX architecture to spike implementation, much less CNN architectures. Masquelier and Thorpe (2007) use STDP-like unsupervised methods to learn pre-classification features, then use a non-spiking classifier on these features. Not only were their classifiers non-spiking, but also they could not reach the recognition accuracy similar to those achieved by CNNs. The biggest disadvantage, though, is the lack of methodology to carry out supervised learning using STDP-like mechanism. The frameless spike-based convolution network of Perez-Carrasco et al. (2010) exploits address event representations (AER) for spike communication, and employs kernel projection to implement convolution computation. Although this approach can achieve fast input-to-output response time, it requires massive amount of digital hardware resources, such as memory and accumulator for map integration, which has not yet shown any power saving benefits. There is also no known good solution to implementing the sigmoid function (Perez-Carrasco et al. 2010).

This work describes a method for converting CNN architecture into a SNN architecture that can be directly mapped to certain types of spike-based neuromorphic hardware with little performance loss. The performance is assessed by evaluating the SNN (in software simulation) on publicly available Neovision2 Tower video dataset (Itti 2013) and comparing it to object recognition performance of the original CNN (Khosla et al. 2013). We also compared the performance of a CNN trained on the CIFAR-10 (Krizhevsky 2009) image data set and the SNN software simulation of the same CNN. Our initial analysis of mapping such an SNN implementation to the emerging class of spike-based neuromorphic hardware, such as those being developed under the DARPA SyNAPSE (Systems of Neuromorphic Adaptive Plastic Scalable Electronics) program (e.g., Cruz-Albrecht et al. 2012; Merolla et al. 2011; Arthur et al. 2012; Cassidy et al. 2013) suggest two orders of magnitude lower power consumption than its rate-based counterpart CNN running on conventional off-the-shelf FPGA-based hardware.

The rest of the paper is organized as follows. Sect. 2 describes our approach by first outlining the CNN architecture (2.1), then discussing the issues and challenges in converting rate-based CNN to spike-based implementation (2.2), followed by the our proposed solutions to overcoming these challenges (2.3). Sect. 3 describes our experimental results of applying SNN to Neovision2 and CIFAR-10 datasets. Sect. 4 describes our energy efficiency analysis of mapping SNN to spike-based hardware. Sects. 5 and 6 discusse the findings and conclusions.

2 Method

2.1 Traditional CNN Architecture

Figure 1 shows a typical CNN architecture, which was also used in HRL's Neovision2 approach (Khosla et al. 2013). It

has three convolution levels, with three layers for each level. The first layer is a spatial convolution layer consisting of a bank of filters. The second layer is a point-wise nonlinear sigmoid function typically implemented by the function tanh(). Spatial convolution is a two-dimensional linear filtering with a set of filtering kernels for all neurons in the preceding level, producing a set of feature maps. Here, "tanh()" is the hyperbolic tangent function, which is a standard way to model neuron output activation in CNN. The third and last layer is spatial max-pooling, which takes the maximum output values over a small image neighborhood in the feature maps produced by the non-linearity layers. In the example shown in Figure 1, the last convolution level does not have a max-pooling layer; a coincidence due to the choice of architecture parameters. The last level of the CNN is a fully connected one layer linear classifier. The entire network can be trained using a standard error backpropagation algorithm with stochastic gradient descent. The goal of training the CNN is to adjust the system parameters (i.e., the convolution kernel weights and biases) so that the output of the CNN predicts the class (i.e., label) when an unknown input image is presented.

2.2 Challenges in Implementing a Spiking CNN

As outlined in Sect. 1, it is desirable to convert an existing design of CNN into a SNN with the end goal of mapping to low power neuromorphic hardware. There are two potential approaches for converting a CNN into an SNN. The first approach is to directly train a spiking network with CNN-like architecture. Although there are some STDP-like spiking learning rules to train a SNN in a self-organized, unsupervised way, research in this area is in its infancy, and it is still unclear how to train a deep SNN effectively to achieve a higher level function, for example, classification with supervised learning. The second approach is to train the original CNN and then apply the learned weights to a SNN with a similar architecture as the trained CNN. The main challenge in this approach is the potential for unacceptable accuracy loss in classification when doing the conversion of network from CNN to SNN. In fact, we observed this loss in accuracy when we converted CNN architecture of Fig. 1 directly into a spike-based architecture. There are several reasons for this loss in accuracy:

1. Negative output values in CNN layers are more difficult to be represented accurately in SNN. The negative values come from the following CNN computations:
   a. The sigmoid function tanh() has output values between -1.0 and 1.0;
   b. In each convolution layer, the values of each output feature map are weighted sums of inputs plus biases. Weights and biases can both be negative causing the output value to be negative;
   c. Output values from preprocessing (e.g., color transform and spatial normalization) can produce negative values.

Though it is possible to represent negative values in a spiking network with inhibitory neurons, it will require doubling the number of neurons and make it a larger and more complicated network with more interconnections. Increasing the neuron count translates to more hardware resources and higher power consumption, both of which are undesirable.

2. Unlike CNN, there is no good way to represent biases in spiking networks. The biases in each convolution layer can be positive or negative, which cannot be represented easily in SNN.
3. Max-pooling requires two layers of spiking networks. In CNN, spatial max-pooling is implemented as taking the maximum output values over a small image neighborhood in the input. In SNN, we need two-layer neural networks to

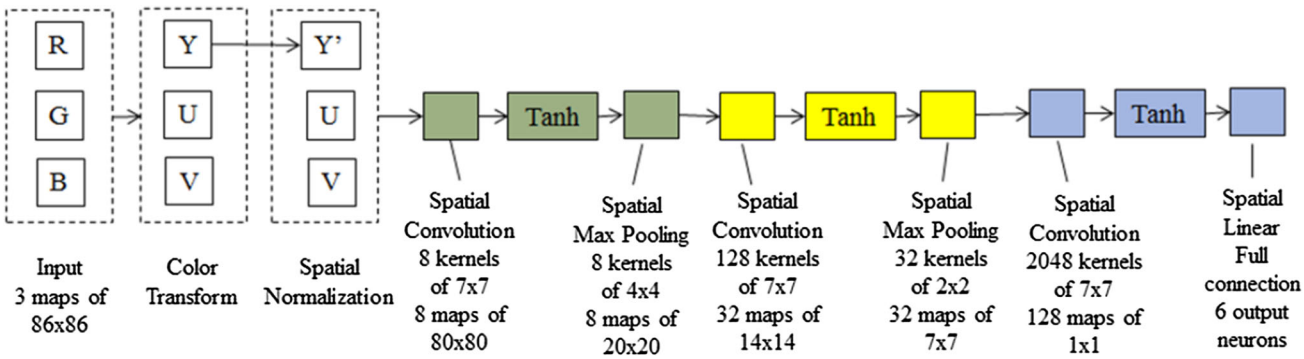

Fig. 1 A typical CNN architecture with 3 convolution layers and a fully connected classification layer. An additional "softmax" output layer is used during training but is omitted here and in the rest of the paper for brevity

Fig. 2 Flow-diagram of converting a CNN into SNN architecture

accomplish this, with lateral inhibition followed by pooling over these small image regions. This approach requires more neurons and can cause accuracy loss due to the added complexity.

2.3 Proposed Spiking CNN Architecture

Our solution to overcoming the challenges described in Sect. 2.2 is to tailor the CNN architecture to fit the requirements of the SNN. We then train the tailored CNN and apply the learned network weights to a spiking network with architecture similar to that of a tailored CNN. These steps are illustrated in Fig. 2.

More specifically, we make the following changes to the CNN architecture to fit the requirements of SNN:

1. Make output values in all layers positive:
   a. Add an abs() function (absolution value) layer after preprocessing phase that includes color transformation and spatial normalization. The abs() function ensures that the input values to the first convolution layer are all non-negative. This step can be omitted if the preprocessing method itself gives positive output (e.g., the original RGB values are used);
   b. Change the sigmoid activation function (after convolution) from tanh() to HalfRect(). HalfRect(x), also referred to as rectified linear units (ReLU), is defined as:

$$\text{HalfRect}(x) = \max(x, 0)$$

For a justification for using $\max(x, 0)$ instead of a sigmoid function, see Krizhevsky et al. (2012), where they showed that the non-saturating non-linearity, such as $\max(x, 0)$, converges much faster in CNN training than the saturating non-linearity function, such as tanh(). Besides, there is another advantage to use HalfRect() instead of tanh(); HalfRect(x) is linear when x is positive, which minimizes the accuracy loss when converting a CNN into SNN architecture with integrate-and-fire spiking neurons.

2. Remove biases from all convolution and the fully connected layers. A simple implementation without changing existing code is to reset all bias values to zero after each training iteration.
3. Use spatial linear subsampling instead of spatial max-pooling. The spatial linear subsampling adds all pixels over a small image neighborhood using a kernel of uniform weights which sum to 1.0. A spatial linear subsampling function can be easily converted to spike domain.

Following the above steps we can convert the CNN in Fig. 1 into a tailored CNN as shown in Fig. 3.

The conversion of the tailored CNN to SNN architecture is straightforward. Figure 4 illustrates the converted SNN architecture in block diagram, while Fig. 5 illustrates details of the same SNN. The SNN architecture, as shown in Fig. 4, consists of Preprocessing, Spike Generation, first Spatial Convolution and Linear Subsampling layers, second Convolution and Linear Subsampling layers, third Convolution and Linear Classification layers, and Spike Counter. Compared to Fig. 3, the new additions to the architecture are Spike Generation and Spike Counter, with the rest of the processing blocks carried over from the tailored CNN, except for HalfRect. The HalfRect function layers are implied properties of our spiking neurons, therefore are omitted in the SNN architecture. The connections in the three levels of convolution and between the levels are similar to those in the tailored CNN architecture. Therefore, the connections that previously existed in the tailored CNN between two layers of neurons still exist in the SNN architecture, except that the connections in SNN represent axons and carries spikes from a previous layer to the next (see inset Fig. 5). The kernel weights for the connections in the tailored CNN become the synaptic strength (i.e., weights) in the SNN architecture. As discussed earlier in this section (see Step 2 above), all the biases in the tailored CNN have been reduced to 0; therefore they are eliminated in the SNN architecture.

The membrane potential $V(t)$ of a spiking neuron in the SNN architecture is updated at each time step by the following equation according to the integrate-and-fire neuron model:

$$V(t) = V(t-1) + L + X(t), \tag{1a}$$

$$\text{If } V(t) \geq \theta, \text{ spike and reset } V(t) = 0, \tag{1b}$$

$$\text{If } V(t) < Vmin, \text{ reset } V(t) = Vmin. \tag{1c}$$

where L is the (constant) leakage parameter, $X(t)$ is the summed input at time t from all synapses connected into the neuron. Whenever $V(t)$ exceeds its threshold θ, the neuron fires and produces a spike, and its membrane potential $V(t)$ is reset to zero. The membrane potential V is not allowed to go below its resting state $Vmin$ which is usually set to 0, but can be changed to allow V to go negative. The parameters used (L and θ) in our simulation can be found in Table 1 in Sect. 3.

For example, for a neuron (i, j) in a convolution map in the first convolution layer in Fig. 5, $X(t)$ can be defined as

$$X_{ij}(t) = \sum_{p,q=-3}^{3} A_{p+i,q+j}(t) K_{pq}, \tag{2}$$

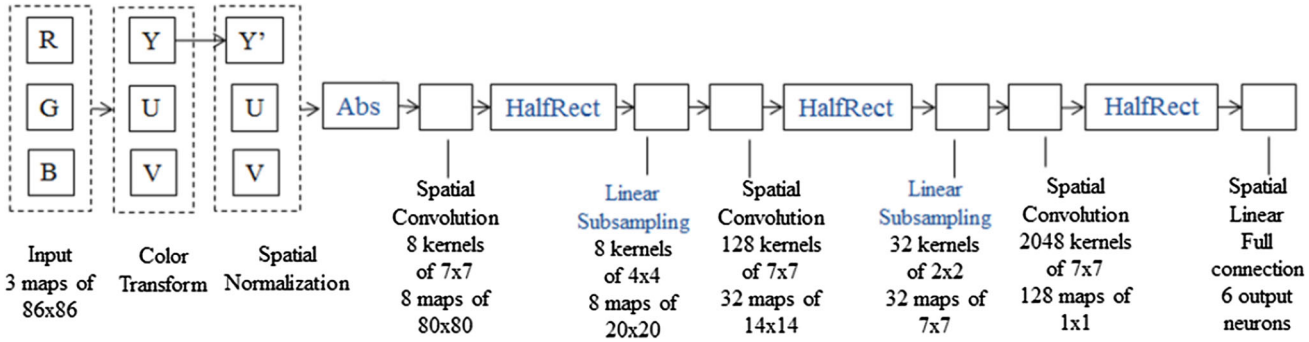

Fig. 3 Tailored CNN architecture

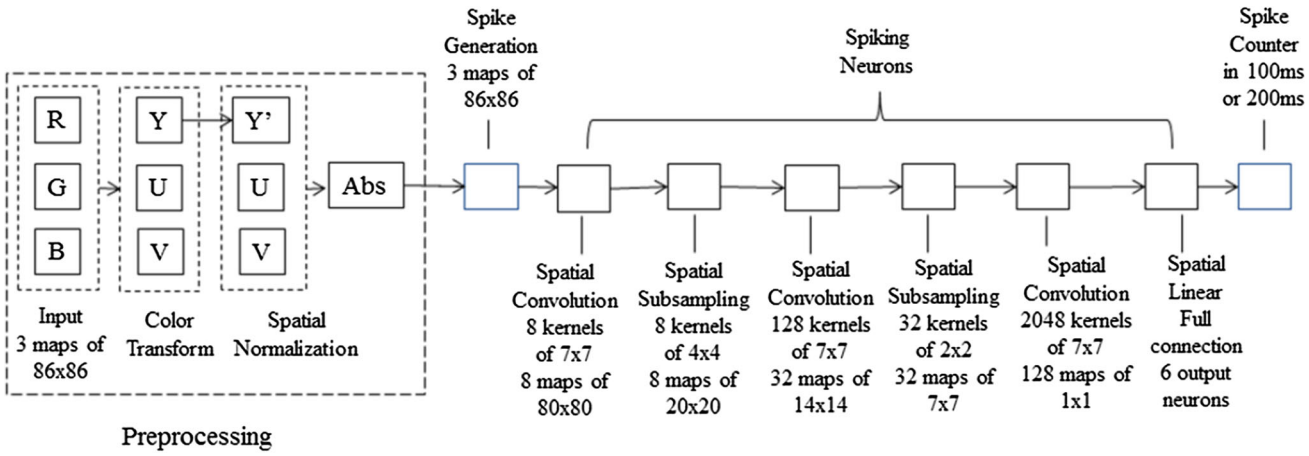

Fig. 4 Block Diagram of the Spiking CNN (SNN) architecture

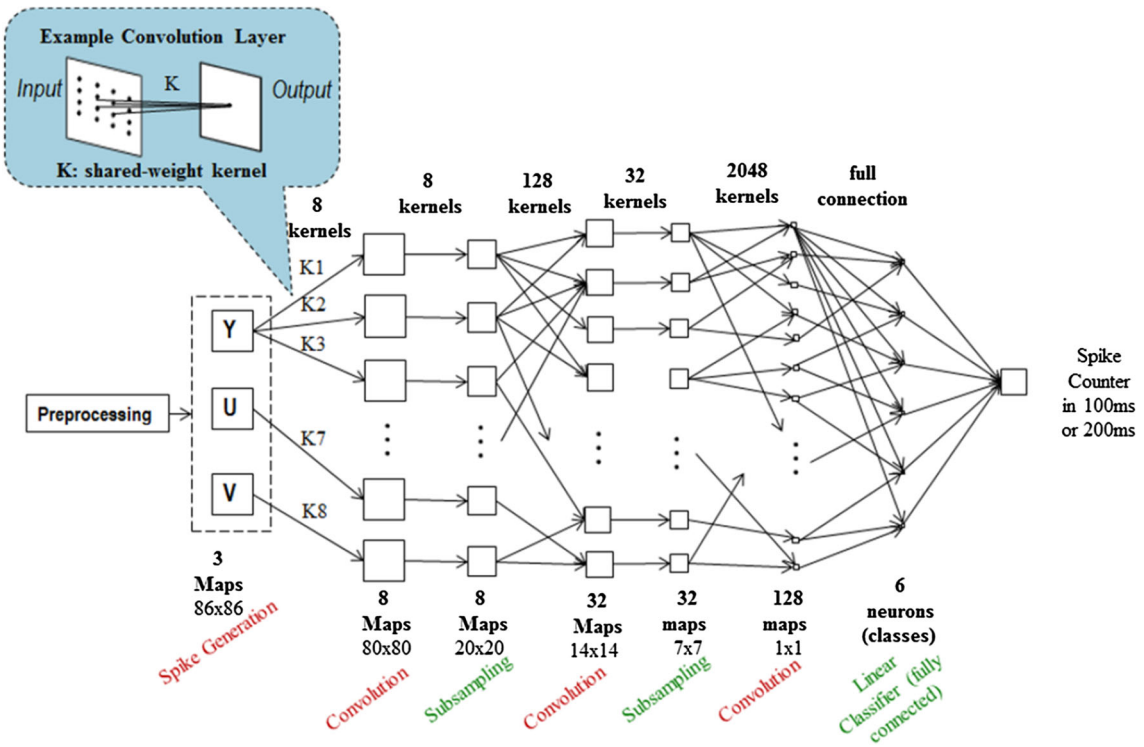

Fig. 5 Detailed Spiking CNN (SNN) architecture

Table 1 Parameters of spiking neurons in SNN simulator (refer to Eq. 1) for the experiments discussed in this paper unless stated otherwise

Neurons in CNN layers	Leakage (L)	Firing threshold (θ)
1st Convolution layer	0	10
1st Subsampling layer	0	0.99
2nd Convolution layer	0	1
2nd Subsampling layer	0	0.99
3rd Convolution layer	0	1
Linear classification layer	0	1

where $A_{p+i,q+j}\,(t)$ are the input spikes (0 or 1) that come from the previous layer, and K_{pq} are weights (or coefficients) of convolution kernel of size 7×7 that is shared by all neurons in the same map.

The integrate-and-fire neuron model in Eq. 1 is a simple model and has been implemented in many neuromorphic circuit designs (Cruz-Albrecht et al. 2012; Merolla et al. 2011; Cassidy et al. 2013). Therefore the SNN architecture shown in Figs. 4 and 5 can in principle be implemented with these types of neuromorphic hardware. Our approach is not limited to the integrate-and-fire spiking neuron model defined in Eq. 1; we can just as easily use other models for spiking neurons in our SNN architecture. However, the activation function HalfRect() in the tailored CNN may need to be adapted for special properties of other spiking neuron models.

The Spike Generation layer is usually defined as follows: Let I_{ijk} $(k = 1, 2, 3)$ be the image maps that are input to the Spike Generation layer. At time t, neuron $(i,\ j)$ in the kth image map produces a spike if

$$\text{rand}(\,) < c\, I_{ijk} \tag{3}$$

where *rand*() is a random number generator with uniform distribution on (0.0, 1.0), and c is a constant to scale the frequency of spikes generated. For example, we used $c = 1/3$ in the simulations on Neovision2 Tower dataset described in the next section, where the initial input RGB images (see Fig. 4) have been normalized to the range [0.0, 1.0]. The Spike Counter counts spikes of all category neurons (e.g., 6 in the example shown in Fig. 4) in the Linear Classification layer in certain time period (e.g., 100 or 200 time steps) from the start of presentation of the input image; then produce category output (classification result) corresponding to the neuron which has the maximum number of spikes.

3 Experiments

3.1 Neovision2 Tower Dataset

In this section, we describe our experiments and results. We trained the tailored CNN described in Fig. 3 using error backpropagation with stochastic gradient descent algorithm implemented in Torch 7 (Collobert et al. 2011). We then applied the learned weights of the tailored CNN to the SNN architecture described in Figs. 4 and 5. We test the SNN architecture by comparing its performance against the CNN (Fig. 1) as well as the tailored CNN (Fig. 3) trained using the same training data.

The data we use in this experiment is the Tower dataset, one of three aerial platform datasets from the DARPA Neovision2 program [publicly available, see (Itti 2013)], in which HRL team demonstrated the best object recognition performance with the highest recognition accuracy and four orders of magnitude lower power than two independent state-of-the-art computer vision systems (DARPA 2011). For the Tower dataset, there are 6 classes: *Car, Cyclist, Bus, Person, Truck, and Nontarget* (or background). A sample image with ground truth boxes are shown in Fig. 6. We extract image chips based on ground truth information from the training dataset for training, and test on different image chips similarly extracted from the test dataset. The *Nontarget* chips are extracted from the training images by sliding a fixed sized window across the images and extracting those that do not overlap with any ground truth objects. The number of training chips we used ranges from 100 (e.g. for *Bus* and *Truck*) to 30,000 (e.g. for *Cyclist* and *Person*). A regular CNN and a tailored CNN are trained using the exact same training data.

We evaluated the performance of the CNN, tailored CNN and the SNN architecture converted from the tailored CNN on the test dataset images. For the SNN architecture, before we present each test image to the SNN, we reset all the neurons so their membrane voltage V is zero, and we clear the counters in Spike Counter. The SNN simulation is written in MATLAB. It runs at 1 millisecond simulation time step for 100 steps. We count the spikes at the Spike Counter and declare the class of the input image as one of the 6 predefined classes. The neurons are then reset and counters cleared before we process the next image; this process is repeated until all test dataset images are processed. The same experiments were carried out using the regular CNN and the tailored CNN. After all test images are processed, a confusion matrix is constructed (Tables 2, 3, 4, 5, 6 and 7) for each of the test cases. Table 1 shows the leakage and firing threshold parameters of integrate-and-fire neurons (see Eq. 1) used in the spiking CNN simulation. These parameters are the same for all neurons in the same layer.

Tables 2, 3, 4, 5, 6 and 7 show the test results comparing CNN, tailored CNN and the SNN implementation of the tailored CNN. Each is a confusion matrix using one implementation on a set of images from sequence 026 or 027 of the test dataset. The labels in the first column of the tables indicate the truth categories of the images corresponding to the test samples shown in the corresponding rows, while the labels in the first row shows the categories of the images as

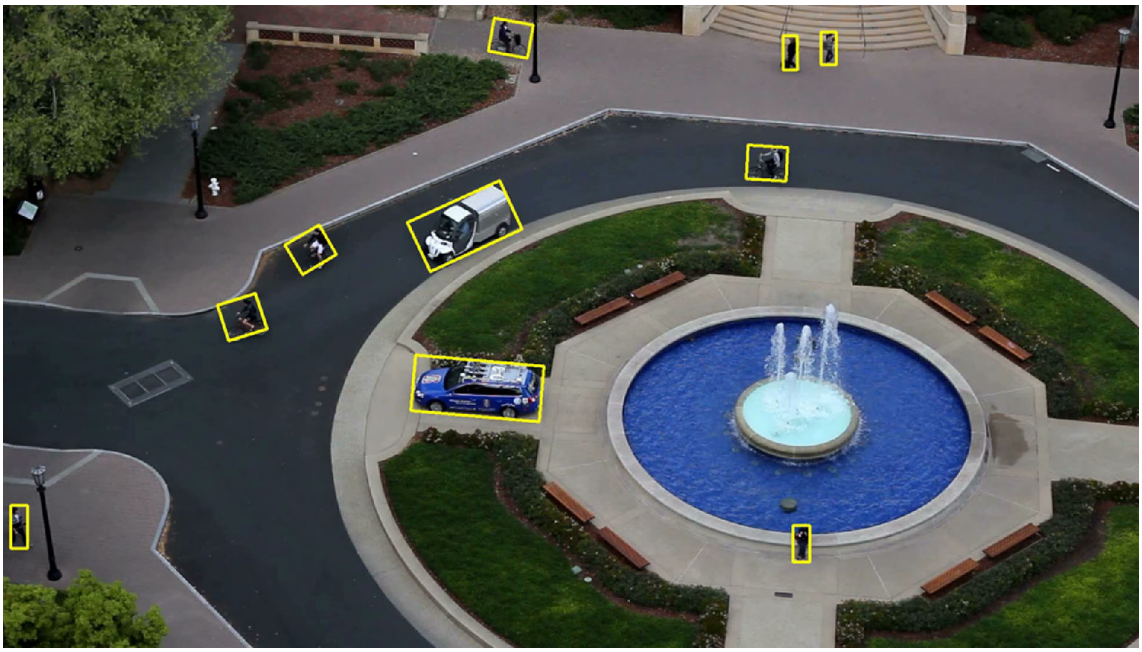

Fig. 6 Sample image frame from DARPA Neovision2 Tower data set with ground truth objects highlighted

Table 2 Confusion matrix for CNN result on data set 026

	Nontarget	Bus	Car	Cyclist	Person	Truck	% Accuracy
Nontarget	1,508	0	0	0	0	0	**100**
Car	0	0	968	22	0	0	**97.78**
Cyclist	0	0	0	486	1	0	**99.80**
Person	0	0	0	1	957	0	**99.48**
Global correct: 99.29							

Table 3 Confusion matrix for tailored CNN result on data set 026

	Nontarget	Bus	Car	Cyclist	Person	Truck	% Accuracy
Nontarget	1,492	0	7	6	3	0	**98.94**
Car	0	0	990	0	0	0	**100**
Cyclist	4	0	0	478	5	0	**98.15**
Person	4	0	1	2	955	0	**99.27**
Global correct: 99.19							

they are classified, or the result, for the corresponding column. The label "nontarget" represents a set of background image windows that do not intersect with any ground-truthed categories. The value entries in the center of the tables show the number of test images with truth categories (rows) classified as the result category (column). The right most column is the "% accuracy" which is the percentage resulting from dividing the values in diagonal of corresponding row by the total of that row. Global correct value is the weighted sum of the % accuracies. Note that since there is no presence of either *Bus* or *Truck* in 026 or 027, the corresponding rows in the confusion matrices have been omitted. The object classification performance accuracy of SNN (Tables 4 and 7) was excellent and comparable to the CNN (Tables 2 and 5) and tailored CNN (Tables 3 and 6) in absolute terms. We were able to achieve close to 99 % global testing accuracy in test data sets 026 and 027 using image chips extracted based on ground truth. This is comparable to the performance of corresponding CNN and tailored CNN, while the estimated energy consumption is far below its rate-based counterpart (see Sect. 4). Figure 7 shows a summary of the test set accuracies of the three implementations in Tables 2, 3, 4, 5, 6 and 7.

Table 4 Confusion matrix for SNN result on data set 026 ($T = 100$ ms)

	Nontarget	Bus	Car	Cyclist	Person	Truck	% Accuracy
Nontarget	1,507	0	0	0	1	0	**99.93**
Car	10	0	980	0	0	0	**98.99**
Cyclist	23	0	1	463	0	0	**95.07**
Person	14	0	5	8	935	0	**97.19**
Global correct: 98.43%							

3.2 CIFAR-10 Dataset

Since the Tower data set has not been a well-known study subject for deep-learning community, we also evaluated the approach for converting a CNN to an SNN architecture presented in this paper on the more widely used CIFAR-10 data

Table 5 Confusion matrix for CNN result on data set 027

	Nontarget	Bus	Car	Cyclist	Person	Truck	% Accuracy
Nontarget	1,345	0	0	0	0	0	**100**
Car	0	0	871	0	0	0	**100**
Cyclist	0	0	0	683	0	0	**100**
Person	0	0	0	3	1,311	0	**96.77**
Global correct: 99.93							

Table 6 Confusion matrix for tailored CNN result on data set 027

	Nontarget	Bus	Car	Cyclist	Person	Truck	% Accuracy
Nontarget	1,339	0	2	2	2	0	**99.55**
Car	0	0	871	0	0	0	**100**
Cyclist	0	0	0	683	0	0	**100**
Person	0	0	0	1	1,313	0	**99.92**
Global correct: 99.83							

Table 7 Confusion matrix for SNN result on data set 027 ($T = 100$ms)

	Nontarget	Bus	Car	Cyclist	Person	Truck	% Accuracy
Nontarget	1,345	0	0	0	0	0	**100**
Car	0	0	871	0	0	0	**100**
Cyclist	0	0	0	663	0	0	**100**
Person	0	0	0	30	1,284	0	**97.72**
Global correct: 99.29							

set (Krizhevsky 2009). This is especially needed since the performance of all three variants of CNN implementations are near the ceiling on the Tower data set (see Fig. 7) making it difficult to meaningfully compare their performance. CIFAR-10 consists of 60 thousand small color images in 10 classes. There have been several published results (see the references in Krizhevskey 2012) on CIFAR-10 using various CNN deep learning architectures. However, most of the proposed architectures in these papers contain components that are beyond the scope of this paper. For example, the response normalization proposed in (Krizhevsky et al. 2012) requires division by the sum of responses across neighboring feature maps, which is not supported by our current SNN architecture.

Since our primary goal is to compare the performance of a CNN with tailored and spiking architectures, we settled on a simple CNN architecture similar to that of Ho's (Ho 2013) as summarized in Table 8. This network consists of 3 convolution layers followed by 2 fully connected layers. The output (softmax) layer is omitted following the same convention in Fig. 1. Just as in Ho's approach (Ho 2013), we use HalfRect (ReLU) units after convolution. However, we do not use max-pooling, but rather simple linear subsampling, for our regular and tailored models. Another difference is in the input data preprocessing. We found that the preprocessing approach shown in Fig. 1 did not work well for CIFAR-10. This is likely due to the smaller size and relatively blurrier CIFAR-10 images compared with the Tower data set. We use the original RGB input, but rescale each pixel to [0.0, 1.0] by first subtracting the global minimum pixel value across R, G and B channels, and then dividing by the resulting global maximum pixel value. This approach obviates the use of abs() as in the tailored CNN in Fig. 3.

The architectural design of this model already incorporates many of the recommendations for a "tailored" CNN as outlined in Sect. 2.3. That is, we made input data all positive by normalizing and shifting to [0.0, 1.0]; used HalfRect (ReLU) activation units rather than tanh(), and linear subsampling rather than max-pooling. The only remaining step in converting this network to a "tailored" version is to remove the biases. This is dealt with during training in the same way as discussed in Sect. 2.3.

Finally, note that this model does not use overlapping in pooling, nor does it apply "drop-out" as is done in Krizhevsky

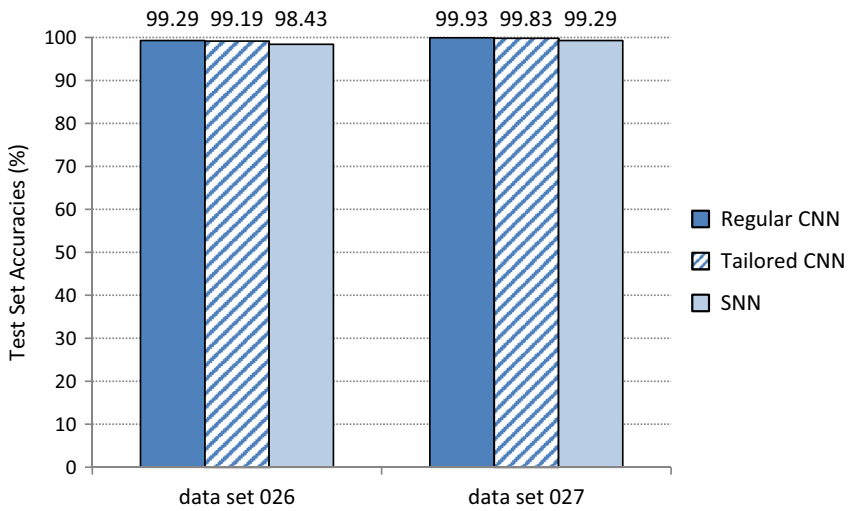

Fig. 7 Comparison summary of Tower test set accuracies (global correct %) of regular, tailored and spiking implementations of CNN. Performance details are presented in confusion matrices shown in Tables 2, 3, 4, 5, 6 and 7

Table 8 CNN baseline model for CIFAR-10 (with softmax output layer omitted)

Input layer	$3 \times 24 \times 24$ color RGB in [0.0,1.0]
Convolution 1	$3 \times 5 \times 5$ kernels,HalfRect units, 64 output maps of 20×20
Pooling 1	2×2 non-overlapping subsampling, 64 output maps of 10×10
Convolution 2	$64 \times 5 \times 5$ kernels, HalfRect units, 64 output maps of 6×6
Pooling 2	2×2 non-overlapping subsampling, 64 output maps of 3×3
Convolution 3	$64 \times 3 \times 3$ kernels, HalfRect units, 64 output maps of 1×1
Fully connected 1	Fully connected, HalfRect units, 64 output neurons
Fully connected 2	Fully connected, 10 output neurons

et al. (2012). Both of these should improve the model performance, and can be easily implemented in SNN to improve model performance. However, we did not implement these in order to compare our results with that of (Ho 2013), and we expect any performance gain by such enhancements in the models to benefit the final SNN implementation as well.

For the model in Table 8, the input image required for producing 1 by 1 feature map at the output of 3[rd] convolution layer is 24 by 24. This allows us to use data augmentation to improve training results by randomly taking 24 by 24 windows (with uniform offsets in both row and column) from the training samples during training. We also randomly flip (at 50 % chance) the samples horizontally. We did not perform other augmentation methods suggested by (Krizhevsky et al. 2012), e.g., changing image intensity. For validation and testing, we take a single centered window of 24 by 24 only and use it as input, unlike (Ho 2013) which used the average of softmax output of all 9 possible windows of 24 by 24 of a given sample.

As in (Krizhevsky et al. 2012) and (Krizhevskey 2012), we used batches 1 to 4 of CIFAR-10 as training sets, batch 5 as validation set, and the "test" batch as test set. We trained the model in Table 8 as a regular CNN and a tailored CNN (by setting the biases to 0 as discussed earlier in this section). We trained the CNNs until the validation accuracy stabilizes, and chose the best test set accuracies for the regular and tailored CNN models, which are 79.12 and 79.09 % (or error rates of 20.88 and 20.91 %) respectively. This is shown in Fig. 8. For reference, the training and validation accuracies are 86.66 and 78.76 % (or error rates of 13.34 and 21.24 %) respectively for the regular CNN, and 85.85 and 79.38 % (or error rates of 14.15 and 20.62 %) respectively for the tailored CNN. As mentioned earlier in this section, the only difference between the regular CNN and the tailored CNN for this model is that in the tailored CNN, all biases are set to 0 for all convolution, subsampling and fully connected layers. It appears that this change has no impact on the performance of the tailored CNN. The test error rates of these models are on par with that Ho (2013) achieved (20.1 %) even though Ho (2013) used the average of all 9 possible 24 by 24 windows of a test sample to score the result, whereas we only used a single center window. Although ∼20 %+ error rates is far from the state-of-the-art in CIFAR-10 results (the best is 91.2 % accuracy or 8.8 % error of Lin et al. 2014), we believe our model performed at its potential and is representative of the currently best achievable results with a CNN with no "bells and whistles" added.

The SNN implementation of the CIFAR-10 network model achieved a respectable 77.43 % accuracy (22.57 % error), also shown in Figure 8, compared to 79.09 % achieved with the tailored version of the CNN. This is in-line with our expectation and results seen on the Tower data set, given the extent of the precision loss caused by a spiking implementation.

The performance for the SNN shown in Fig. 8 corresponds to running the simulation for 400 time steps (rather than 100 for the Tower data in Tables 2, 3, 4, 5, 6 and 7). Possible reasons why the CIFAR-10 SNN model requires more time steps to achieve performance levels comparable to the tailored model is that the network is deeper and larger with more nodes than the one used for Tower data.

A few other changes related to how we achieved the performance shown in Fig. 8 for SNN model compared with the one used for Tower data are in order. First, we changed the spike generation method compared with the one outlined in Sect. 2.3 (Eq. 3). The new spike generator can be defined by an integrate-and-fire type neuron as follows:

$$V(t) = V(t-1) + I(t)$$

where $V(t)$ is the membrane potential and $I(t)$ is the analog input image pixel taking value between 0.0 and 1.0. Whenever $V(t)$ exceeds a predefined threshold θ ($\theta = 1$ is used in the simulation), a spike is generated and then the membrane potential $V(t)$ is reset to $V(t) - \theta$. Second, we changed the value of *Vmin* as described in Eq. 1c. Instead of setting it to 0, we set it to -10 θ (negative 10 times θ) to allow V to go negative, where θ is the firing threshold for the neuron (Table 1). Lastly, we changed the θ for the neurons in the first convolution layer to 5 instead of 10 as shown in Table 1 to match the spike density generated by the new spike generator. All these changes help in improving SNN performance for our CIFAR-10 model.

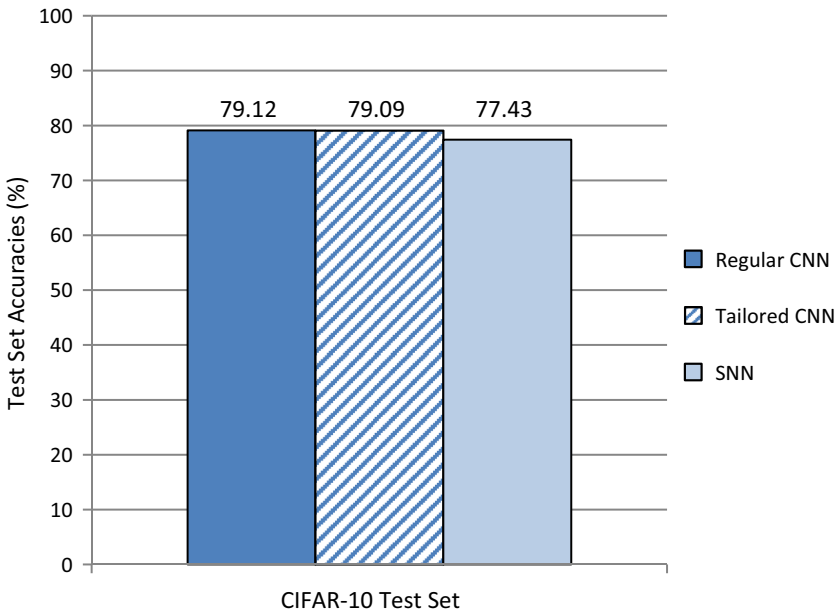

Fig. 8 CIFAR-10 test set performance of the baseline CNN model shown in Table 8 (as "Regular CNN"), the tailored version ("Tailored CNN"), and the SNN implementation of the tailored model. The baseline and tailored models achieved very good performance in-line with the type of model used (see text for discussion), and the SNN implementation's performance is very close to that of the tailored CNN

4 Energy Analysis of Mapping to Neuromorphic Hardware

One of the key benefits of the SNN architecture proposed in this work is its potential for high energy efficiency when mapped to and implemented in the emerging class of ultra-low power spike-based neuromorphic hardware, such as Cruz-Albrecht et al. (2012), Merolla et al. (2011). Based on the known capabilities and constraints of such neuromorphic hardware, we conduct a first-order analysis of power consumption of the SNN architecture mapped to these types of hardware and compare it with an implementation of CNN in conventional FPGA-based hardware (Xilinx Virtex-6 ML605) (Khosla et al. 2013b).

For this analysis, we make a simplifying, but reasonable, assumption that a spike activity consumes α Joules of energy, regardless of whether the spike is pre-synaptic (i.e., delivered by synapses) or post-synaptic (i.e., generated by neurons). For the sample SNN architecture illustrated in Figs. 3 and 4, we measured on average 5×10^5 post-synaptic spikes and 2×10^7 pre-synaptic spikes for processing a single image chip during the 100 simulation steps (i.e., 100 milliseconds time) for Tower data. Therefore, the total energy consumed for processing each image chip is:

$$\begin{aligned} \text{Total energy} &= \left(5 \times 10^5 + 2 \times 10^7\right) \alpha \\ &\cong 2 \times 10^7 \alpha \text{ Joules/chip} \end{aligned}$$

To estimate the power requirements for using such hypothetical SNN "devices" to handle object classification in real-time video input, we assume a video rate of 30 frames/s and 5 image chips per image frame (e.g., 5 detected potential objects in a typical video frame of Tower dataset that require classification by SNN). This gives us the number of image chips we must process with the SNN devices in a second (i.e., $30 \times 5 = 600$). Multiplying this total by the energy consumed per chip gives us the total energy consumed in a second, which is the power requirement P:

$$\begin{aligned} P &= 2 \times 10^7 \alpha \text{ Joules/chip } \times 30 \text{ frame/s } \times 5 \text{ chips/frame} \\ &= 3 \times 10^9 \alpha \text{ Joules/s} \\ &= 3 \times 10^9 \alpha \text{ Watts} \end{aligned} \quad (4)$$

Table 9 shows the total SNN power consumption based on the power characteristics of two published spike-based neuromorphic circuits using the formula in Eq. 4. An FPGA-based implementation (Xilinx Virtex-6 ML605) of CNN using NeuFlow (Farabet et al. 2010) for the same processing as above was measured at $P_{FPGA} = 25$ Watts during a formal evaluation in the DARPA Neovision2 program (Khosla et al. 2013b). As can be seen from the last column of Table 9, the SNN architecture has more than two orders of magnitude higher energy efficiency than the FPGA-based implementation of CNN.

We should point out that in the power estimation given in Eq. 4, we assume we have multiple SNN devices at our disposal so that we can process multiple image chips simultaneously to achieve the throughput required by real-time video input, albeit with some fixed delay associated with SNN processing (i.e., 100 milliseconds).

5 Discussions

5.1 Use of abs() on Input Features

The use of abs() (see Sect. 2.3) on preprocessed input to turn the input values to all positive may have an impact on model

Table 9 Estimates of SNN power consumption mapping to spike-based hardware. The value in column 3 (P) is calculated by substituting α in Eq. 4 with the corresponding value in column 2. The value in column 4 is the ratio of P_{FPGA} to P, where $P_{FPGA} = 25$ Watts, the power consumption of FPGA-based CNN implementation

Neuromorphic circuits	Energy consumption per spike (α) *pico-Joules*	Total power consumption from Eq. 4 (P) *milli-Watts*	Ratio of power consumptions of FPGA-based to SNN spike-based CNN implementations (P_{FPGA}/P)
Cruz-Albrecht et al. (2012)	0.37	1.11	22,522
Merolla et al. (2011)	45	135	185

performance in general, but for the tailored CNN shown in Fig. 3 this does not seem to be the case as the results in Fig. 7 show performance of tailored CNN being very close to that of the regular CNN. The impact of abs() on the performance is dependent on the data and the preprocessing method used. This is confirmed by our experience with the CIFAR-10 set. In our experiments, when the RGB channels are used directly after spatial normalization (with an approach similar to the divisive contrast normalization of Jarrett et al. 2009), an abs() applied to the preprocessed features will results in a significant performance drop in a tailored CNN. Consequently, we forgo the spatial normalization step and use a simple offset and contrast stretch as outlined in Sect. 3.2.

Therefore the guideline for ensuring input features to be positive in a tailored CNN can be summarized as follows.

1. Use a preprocessing step that does not create negative outputs, as shown in Sect. 3.2 on CIFAR-10. This is the best choice;
2. If the negative outputs from preprocessing cannot be avoided, then there are two options.
   a. Use abs() if it does not impact performance, as we have done in Sects. 2 and 3.1.
   b. Otherwise, keep negative values in input features, but accommodate them in the corresponding SNN with another set of spike generators for the negative values and use inhibitory connections along with the existing excitatory connections to the first layer of the network. However, this approach effectively doubles the input connection counts and will have a negative impact on energy consumption when the network is implemented in neuromorphic circuits.

5.2 Comparison with Limited Resolution Implementation of CNN

The CNN implementations (including both regular and tailored CNN) using Torch7 does all computations in either double-precision "double" or single-precision "float", and maintains all network parameters (trained weights and biases) in the same representation. In this work, we compared the performance of this high-precision implementation with the spiking implementation that communicates with discrete spikes. It is important to point out that even though we used spikes to communicate between layers of the SNN network, the integrate-and-fire neuron computation (Eq. 1) is still carried out in "double". On the one hand, it would be more meaningful if we had an implementation of CNN in limited resolution (for example by quantizating the network weights as in NeuFlow (Farabet et al. 2010)). On the other hand, Farabet showed (Farabet 2013) that quantization of floating-point network weights to a 16-bit fixed-point representation did not affect the performance of CNN on NORB, MNIST and UMASS (face dataset). Farabet (2013) showed that the 16-bit fixed-point representation causes very minimal accuracy loss as measured by the percentage of network weights being zeroed due to the quantization. These results give us hope that the conclusions reached in this paper regarding recognition performance will stand when SNN implementation is compared with a limited precision CNN implementation.

Future work based on limited precision computation in implementing integrate-and-fire neuron (Eq. 1) and storing network weights in limited precision form, is a necessary step towards practical neuromorphic circuit implementation. Likewise, carrying out training and testing for regular and tailored CNN in limited precision could also benefit this goal.

6 Summary and Conclusions

This work presented a new approach for converting a CNN to a SNN architecture that is suitable for mapping to spike-based neuromorphic hardware. Our findings suggest that such an implementation has potential for high energy efficiency, while still maintains the high performance of state-of-the-art CNN for object recognition. This could open up the door to practical vision applications with stringent size, weight and power (SWaP) requirements.

The key findings of our study are that we can tailor a regular CNN into an architecture that is suitable for spiking architecture using the following steps: (1) eliminate the use of negative values in the input features and between layers (after activation units); (2) employ HalfRect (or ReLU) for neuron activation; and (3) remove the biases from all layers of the original CNN. Such a tailored CNN can then be trained

using conventional CNN training method. We can then easily convert the resulting tailored CNN into a similar spike-based architecture that can achieve the same level of object recognition performance as the original CNN. Our estimates suggest that the proposed SNN architecture is more than two orders of magnitude energy-efficient for real-time object recognition in spike-based neuromorphic hardware compared with a more traditional FPGA-based implementation of CNN.

Beside the advantages implied by the potential ultra-low power implementation in neuromorphic hardware, another big advantage of the proposed approach is that it scales as the original CNN. Unlike the approaches that require learning in spike domain, our approach enables mapping any large size CNN into spike-based architecture and reap the performance advantages of the original CNN.

Future directions of research include further trade analysis of spiking rate vs. performance of the spike-based architecture, since spiking rate is usually directly linked to power consumption, and the use of low-precision arithmetic operations in both learning the tailored CNN and in simulating spiking CNN to match spiking neuromorphic hardware constraints. Another near-term research direction is to map these algorithms to hardware emulators for spike-based architectures that are gradually becoming available, and conduct a more thorough and accurate power analysis. Finally, mapping to actual spike-based hardware, similar to how CNN was mapped to traditional FPGA, and demonstrating the performance and power efficiency on real-world object recognition is a longer-term, but realizable research goal.

Acknowledgments This work was partially supported by the Defense Advanced Research Projects Agency Cognitive Technology Threat Warning System (CT2WS) and SyNAPSE programs (contracts W31P4Q-08-C-0264 and HR0011-09-C-0001). The views expressed in this document are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. We would like to thank Dr. Clement Farabet of New York University for providing the initial CNN structure on which the CNN model outlined in Fig. 1 is based; and the anonymous reviewers for their invaluable comments and recommendations that led to this revised manuscript.

References

Arthur, J.V., Merolla, P.A., Akopyan, F., Alvarez, R., Cassidy, A., Chandra, S., & Modha, D.S. (2012). Building block of a programmable neuromorphic substrate: A digital neurosynaptic core. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, (pp. 1–8).

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, *2*(1), 1–127.

Cao, Y., Grossberg, S., & Markowitz, J. (2011). How does the brain rapidly learn and reorganize view-invariant and position-invariant object representations in the inferotemporal cortex? *Neural Networks*, *24*(10), 1050–1061.

Cao, Y., & Grossberg, S. (2012). Stereopsis and 3D surface perception by spiking neurons in laminar cortical circuits: A method for converting neural rate models into spiking models. *Neural Networks*, *26*, 75–98.

Cassidy, A.S., Merolla, P., Arthur, J.V., Esser, S.K., Jackson, B., Alvarez-Icaza, R., & Modha, D.S. (2013). Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores. In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*, (pp. 1–10).

Ciresan, D., Meier, U., & Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 3642–3649).

Collobert, R., Kavukcuoglu, K., & Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In: *BigLearn*, NIPS Workshop.

Cruz-Albrecht, J. M., Yung, M. W., & Srinivasa, N. (2012). Energy-efficient neuron, synapse and STDP integrated circuits. *IEEE Transactions on Biomedical Circuits and Systems*, *6*(3), 246–256.

Defense Advanced Research Projects Agency (DARPA) (2011), "Neovision2 Evaluation Results", DISTAR Case Number 18620, December 22, 2011.

Farabet, C., Martini, B., Akselrod, P., Talay, S., LeCun, Y., & Culurciello, E. (2010), "Hardware Accelerated Convolutional Neural Networks for Synthetic Vision Systems", In: *IEEE International Symposium on Circuits and Systems* (ISCAS'10), Paris, 2010.

Farabet, C. (2013), Towards real-time image understanding with convolutional neural networks, Ph.D. Thesis, Universit'e Paris-Est, Dec. 19, 2013 (http://pub.clement.farabet.net/thesis.pdf, accessed July 10, 2014).

Folowosele, F., Vogelstein, R. J., & Etienne-Cummings, R. (2011). Towards a cortical prosthesis: implementing a spike-based HMAX model of visual object recognition in silico. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *1*, 516–525.

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, *36*(4), 193–202.

Fukushima, K. (1988). Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, *1*(2), 119–130.

Fazl, A., Grossberg, S., & Mingolla, E. (2009). View-invariant object category learning, recognition, and search: How spatial and object attention are coordinated using surface-based attentional shrouds. *Cognitive Psychology*, *58*(1), 1–48.

Grossberg, S., Markowitz, J., & Cao, Y. (2011). On the road to invariant recognition: explaining tradeoff and morph properties of cells in inferotemporal cortex using multiple-scale task-sensitive attentive learning. *Neural Networks*, *24*(10), 1036–1049.

Grossberg, S., & Huang, T.-R. (2009). ARTSCENE: A neural system for natural scene classification. *Journal of Vision*, *9*(4), 6:1–19, doi:10.1167/9.4.6.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*, 504–507.

Hinton, G. E., Osindero, S., & Teh, Y. (2006a). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

Hinton, G. E., Osindero, S., Welling, M., & Teh, Y. (2006b). Unsupervised discovery of non-linear structure using contrastive backpropagation. *Cognitive Science*, *30*(4), 725–731.

Ho, N. (2013). Convolutional neural network and CIFAR-10, part-3, http://nghiaho.com/?p=1997, July 7, 2013.

Hubel, D. H., & Wiesel, T. N. (1959). Receptive fields of single neurones in the cat's striate cortex. *The Journal of Physiology*, *148*(3), 574–591.

Hubel, D. H., & Wiesel, T. N. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, *160*(1), 106–154.

Itti, L. (2013). Neovision2 annotated video datasets. http://ilab.usc.edu/neo2/dataset/, accessed July 10, 2014.

Jarrett, K., Kavukcuoglu, K., Ranzato, M. A., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? In *12th International Conference on Computer Vision (ICCV)* (pp. 2146–2153).

Khosla, D., Chen, Y., Kim, K., Cheng, S.Y., Honda, A.L., & Zhang, L. (2013a). A neuromorphic system for multi-object detection and classification. *Proc. SPIE. 8745, Signal Processing, Sensor Fusion, and Target Recognition XXII :87450X*.

Khosla, D., Chen, Y., Huber, D., Van Buer, D., & Kim, K. (2013b). Real-time, low-power neuromorphic hardware for autonomous object recognition. *Proc. SPIE 8713, Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications X: 871313*.

Krizhevsky, A. (2009). Learning Multiple Layers of Features from Tiny Images, MSc thesis, Univ. of Toronto, Dept. of Computer Science, 2009. (Also see http://www.cs.toronto.edu/~kriz/cifar.html for CIFAR-10 image data sets).

Krizhevskey, A. (2012). Cuda-ConvNet, http://code.google.com/p/cuda-convnet/, accessed July 10, 2014.

Krizhevsky, A., Sutskever, I, & Hinton, G. (2012). ImageNet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems 25* (pp. 1106–1114).

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., et al. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*(4), 541–551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Lin, M., Chen, Q., & Yan, S. (2014). Network in Network, in *International Conference on Learning Representation (ICLR2014)*, Banff, Canada, April 14–16, 2014.

Merolla, P., Arthur, J., Akopyan, F., Imam, N., Manohar, R., & Modha, D.S. (2011). A digital neurosynaptic core using embedded crossbar memory with 45pj per spike in 45nm. In: *Custom Integrated Circuits Conference (CICC), 2011 IEEE* (pp. 1–4).

Masquelier, T., & Thorpe, S. J. (2007). Unsupervised learning of visual features through spike timing dependent plasticity. *PLoS Computational Biology*, *3*, 0247–0257.

Perez-Carrasco, J.A., Serrano, C., Acha, B., Serrano-Gotarredona, T., & Linares-Barranco, B. (2010). Spike-based convolutional network for real-time processing. In: *2010 International Conference on Pattern Recognition* (pp. 3085–3088).

Ranzato, M. A., Huang, F. J., Boureau, Y. L., & LeCun, Y. (2007). Unsupervised learning of invariant feature hierarchies with applications to object recognition. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'07)* (pp. 1–8).

Razavian, A.S., Azizpour, H., Sullivan, J., & Carlsson, S. (2014). CNN Features off-the-shelf: an Astounding Baseline for Recognition, http://arxiv.org/abs/1403.6382, DeepVision CVPR 2014 Workshop, Columbus, Ohio, June 28, 2014.

Riesenhuber, M., & Poggio, T. (1999). Hierarchical models of object recognition in cortex. *Nature Neuroscience*, *2*(11), 1019–1025.

Rumelhart, D. E., Hintont, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, *323*, 533–536.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. & LeCun, Y. (2014). "OverFeat: Integrated recognition, localization and detection using convolutional networks",In *International Conference on Learning Representations (ICLR 2014)*, April 2014.

Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., & Poggio, T. (2007). Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *29*(3), 411–426.