Ternary Neural Networks for Resource-Efficient AI Applications

Hande Alemdar[1], Nicholas Caldwell[1], Vincent Leroy[1],
Adrien Prost-Boucle[2], and Frédéric Pétrot[2]
[1] Université Grenoble Alpes (UGA), LIG, CNRS
[2]TIMA Laboratory - CNRS/Grenoble-INP/UGA
`{name.surname}@imag.fr`
Grenoble, France

Abstract

The computation and storage requirements for Deep Neural Networks (DNNs) are usually high. This issue limit their deployability on ubiquitous computing devices such as smart phones or wearables. In this paper, we propose ternary neural networks (TNNs) in order to make deep learning more resource-efficient. We train these TNNs using a teacher-student approach. Using only ternary weights and ternary neurons, with a step activation function of two-thresholds, the student ternary network learns to mimic the behaviour of its teacher network. We propose a novel, layer-wise greedy methodology for training TNNs. During training, a ternary neural network inherently prunes the smaller weights by setting them to zero. This makes them even more compact thus more resource-friendly. We devise a purpose-built hardware design for TNNs and implement it on FPGA. The benchmark results with our purpose-built hardware running TNNs reveal that, with only 1.24 µJ per image, we can achieve 97.76% accuracy with 5.37 µs latency and with a rate of 255 K images per second on MNIST.

1 Introduction

Deep neural networks (DNNs) have achieved state-of-the-art results on a wide range of AI tasks including computer vision [1], speech recognition [2] and natural language processing [3]. As DNNs become more complex, their number of layers, number of weights, and computational cost increase. While DNNs are generally trained on powerful servers with the support of GPUs, they can be used for classification tasks on a variety of hardware. Our goal in this paper is to train DNNs that are able to classify at a high throughput on low-power devices.

In recent years, two main directions of research have been explored to reduce the cost of DNNs classifications. The first one preserves the floating point precision of DNNs, but drastically increases sparsity and weights sharing for compression [4, 5]. This has the advantage of preserving compatibility with most standard training algorithms, while significantly diminishing memory and power consumption. However, the power savings are limited by the need for floating-point operation. The second direction completely eliminates the need for floating-point operations using weights discretization [6, 7, 8], with extreme cases such as binary neural networks eliminating the need for multiplications. The main drawbacks of these approaches are, a significant degradation in the classification accuracy, and the need for custom training methods that deviate from well-known back-propagation.

This paper addresses these issues and makes the following contributions:

- We propose a teacher-student approach for training Ternary NNs with weights constrained to $\{-1, 0, 1\}$. The teacher network is trained with stochastic firing using back-propagation, and can benefit from all techniques that exist in the literature such as dropout [9], batch normalization [10], and convolutions. The student network has the same architecture and, for each neuron, mimics the behavior of the equivalent neuron in the teacher network.

- We present a FPGA-based architecture that is able to process ternary NNs at 255 K images per second with an energy cost of 1.24 µJ.

2 Training Ternary Neural Networks

We use a teacher-student approach for training TNNs. First, we train the real-valued teacher network with stochastically firing ternary neurons. Then, we let the student network learn how to imitate the teacher's behavior using a layer-wise greedy algorithm. Both the teacher and the student network have the same architecture. The student network's weights are the ternarized version of the teacher network's weights. The student network uses a step function with two thresholds as the activation function. In Table 1, we provide our notation and descriptions. In general, we denote the discrete values with a bold font. Real-valued parameters are denoted by normal font. We use [.] to denote a matrix or a vector. We describe the details of the two stages in the following subsections.

Table 1: Ternary Neural Network Definitions for a Single Neuron i

	Teacher network	**Student Network**
Weights	$W_i = [w_j], w_j \in \mathbb{R}$	$\mathbf{W_i} = [\mathbf{w_j}], \mathbf{w_j} \in \{-1, 0, 1\}$
Bias	$b_i \in \mathbb{R}$	$\mathbf{b_i}^{lo} \in \mathbb{Z}$ $\mathbf{b_i}^{hi} \in \mathbb{Z}$
Transfer Function	$y_i = W_i^{\intercal}\mathbf{x} + b_i$	$\mathbf{y_i} = \mathbf{W_i}^{\intercal}\mathbf{x}$
Activation Function	$\mathbf{n_i^t} = \begin{cases} -1 & \text{with prob. } -\rho \text{ if } \rho < 0 \\ 1 & \text{with prob. } \rho \text{ if } \rho > 0 \\ 0 & \text{otherwise} \end{cases}$ where $\rho = tanh(y_i), \rho \in (-1, 1)$	$\mathbf{n_i^s} = \begin{cases} -1 & \text{if } \mathbf{y_i} < \mathbf{b_i}^{lo} \\ 1 & \text{if } \mathbf{y_i} > \mathbf{b_i}^{hi} \\ 0 & \text{otherwise} \end{cases}$

2.1 The Teacher Network

The teacher network is trained as a real-valued neural network, it has stochastically firing ternary neurons with output values of -1, 0, or 1. In order to achieve a ternary output for teacher neuron $\mathbf{n_i^t}$, we add a stochastic firing step after the hyperbolic tangent, *tanh*, activation function, as described in Table 1. Although we use *tanh* for obtaining the range $(-1, 1)$ before ternarization, any nonlinear function such as *hard tanh* or *soft-sign* that has the same range is applicable. We do not impose any restrictions to the weights of the teacher network, nor do we ternarize them at this stage. The benefit of this approach is that we can use any technique that already exists for efficient NN training, such as batch normalization [10], dropout [9], etc. The teacher network can have any architecture with any number of neurons, and can be trained using any of the standard training algorithms.

2.2 The Student Network

After the teacher network is trained, we begin the training of the student network. The goal of the student network is to predict the output of the teacher real-valued network. Since we use the same architecture for both networks, there is a one-to-one correspondence between the neurons of both. Each student neuron $\mathbf{n_i^s}$ learns to mimic the behavior of the corresponding teacher neuron $\mathbf{n_i^t}$ individually and independently from the other neurons. In order to achieve this, a student neuron uses the corresponding teacher neuron's weights as a guide to determine its own ternary weights using two thresholds t_i^{lo} and t_i^{hi} on the teacher neuron's weights. This step is called the *weight ternarization.* In order to have a ternary neuron output, we have a step activation function of two thresholds $\mathbf{b_i}^{lo}$ and $\mathbf{b_i}^{hi}$. The *output ternarization* step determines these.

Figure 1 depicts the ternarization procedure for a sample neuron. In the top row, we plot the distributions of the weights, activations and ternary output of a sample neuron in the teacher network respectively. The student neuron's weight distribution that is determined by t_i^{lo} and t_i^{hi} is plotted below the teacher's weight distribution. We use the transfer function output of the student neuron, grouped according to the teacher neuron's output on the same input, to determine the thresholds for the step activation function. In this way, the resulting output distribution for both the teacher and the student neurons are similar. In the following sub sections we provide the details of each step.

2.2.1 Output Ternarization

The student network uses a two-thresholded step activation function to have ternary output as described in Table 1. Output ternarization finds the step activation function's thresholds $\mathbf{b_i}^{lo}$ and $\mathbf{b_i}^{hi}$, for a ternary neuron i, for a given set of ternary weights $\mathbf{W}$. In order to achieve this, we compute three different transfer function output distributions for the student neuron, using the teacher neuron's ternary output value on the same input. We use $\mathbf{y}^-$ to denote the set

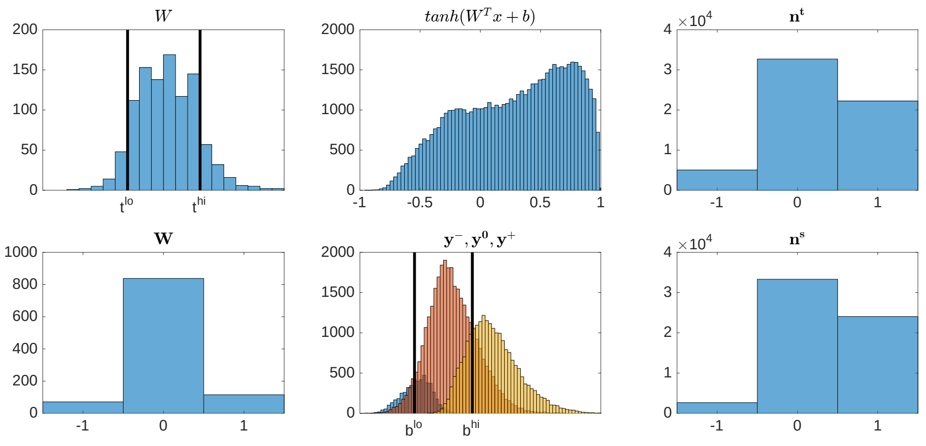

Figure 1: Example ternarization for a single neuron

of transfer function outputs of the student neuron for which the teacher neuron's output value is -1. $\mathbf{y^0}$ and $\mathbf{y^+}$ are defined in the same way for teacher neuron output values 0 and 1, respectively.

We use a simple classifier to find the boundaries between these three clusters of student neuron transfer function outputs, and use the boundaries as the two thresholds $\mathbf{b_i}^{lo}$ and $\mathbf{b_i}^{hi}$ of the step activation function. The classification is done by using a linear discriminant on the kernel density estimates of the three distributions. The discriminant between $\mathbf{y^+}$ and $\mathbf{y^0}$ is selected as the $\mathbf{b_i}^{hi}$, and the discriminant between $\mathbf{y^-}$ and $\mathbf{y^0}$ gives the $\mathbf{b_i}^{lo}$.

2.2.2 Weight Ternarization

During weight ternarization, the order and the sign of the teacher network's weights are preserved. We ternarize the weights of the i^{th} neuron of the teacher network using two thresholds t_i^{lo} and t_i^{hi} such that $min(W_i) \leqslant t_i^{lo} \leqslant 0$ and $0 \leqslant t_i^{hi} \leqslant max(W_i)$. The weights for the i^{th} student neuron are obtained by weight ternarization as follows

$$ternarize(W_i | t_i^{lo}, t_i^{hi}) = \mathbf{W_i} = [\mathbf{w_j}] \tag{1}$$

where

$$\mathbf{w_j} = \begin{cases} -1 & \text{if } w_j < t_i^{lo} \\ 0 & \text{if } t_i^{lo} \geqslant w_j \geqslant t_i^{hi} \\ 1 & \text{if } w_j > t_i^{hi} \end{cases} \tag{2}$$

We find the optimal threshold values for the weights by evaluating the quality of ternarization with a score function. For a given neuron with p positive weights and n negative weights, the total number of possible ternarization schemes is np since we respect the original sign and order of weights. For a given configuration, for the positive and negative threshold values t^{hi} and t^{lo}, we calculate the following score for assessing the performance of the ternary network, mimicking the original network.

$$S_{t^{lo},t^{hi}} = \sum_{d} p(\mathbf{n^t} = \pm 1|\mathbf{x_d^t})^{\mathbb{I}(\mathbf{n^s}=\pm 1|\mathbf{x_d^s})} p(\mathbf{n^t} = 0|\mathbf{x_d^t})^{\mathbb{I}(\mathbf{n^s}=0|\mathbf{x_d^s})} \tag{3}$$

where $\mathbf{n^t}$ and $\mathbf{n^s}$ denote the output of the teacher neuron and student neuron, respectively. $\mathbf{x_d^t}$ is the input d of the layer to the teacher network, and $\mathbf{x_d^s}$ is the input to the student network. Note that $\mathbf{x_d^t} \neq \mathbf{x_d^s}$ after the of the first layer. Since we ternarize the network in a feed-forward manner, in order to prevent ternarization errors from propagating to upper layers, we always use the teacher's original input to determine its output probability distribution. $p(\mathbf{n^t}|\mathbf{x_d^t})$ is calculated using stochastic firing as described in Table 1. $p(\mathbf{n^s}|\mathbf{x_d^s})$ is calculated using the ternary weights $\mathbf{W}$ with the current configuration of t^{lo}, t^{hi}, and the step activation function thresholds. These thresholds, $\mathbf{b}^{hi}$ and $\mathbf{b}^{lo}$ are selected according to the current ternary weight configuration $\mathbf{W}$.

The optimal ternarization of weights is determined by selecting the configuration with the maximum score.

$$\mathbf{W}^* = \underset{t^{lo},t^{hi}}{\arg\max}\, S_{t^{lo},t^{hi}} \tag{4}$$

The worst case time complexity of the algorithm is $O(\|W\|^2)$. We propose using a greedy dichotomic search strategy instead of a fully exhaustive one. We make a search grid over n candidate values for t^{lo} by p values for t^{hi}. We select two equally spaced pivot points along one of the dimensions, n or p. Using these pivot points, we calculate the maximum score along the other axis. We reduce the search space by selecting the region maximum point lies in. Since we have two points, we reduce the search space to two-thirds at each step. Then, we repeat the search procedure in the reduced search space. This faster strategy runs in $O(log^2\|W\|)$, and when there are no local maxima it is guaranteed to find the optimal solution. When there are multiple local extremum, it may get stuck. Fortunately, we can detect the possible sub-optimal solutions, using the score values we obtain for the student neuron. By using a threshold on the output score for a student neuron, we can selectively use exhaustive search on a subset of neurons. Empirically, we find these cases to be rare. We provide a detailed analysis in Section 4.1.

The ternarization of the output layer is slightly different since it is a soft-max classifier. In the ternarization process, instead of using the teacher network's output, we use the actual labels in the training set. Again, we treat neurons independently but we make several iterations over each output neuron in a round-robin fashion. After each iteration we check against convergence. In our experiments, we observed that the method converges after a few passes over all neurons.

Our layer-wise approach allows us to update the weights of the teacher network before ternarization of any layer. For this optional weight update, we use a staggered retraining approach in which only the non-ternarized layers are modified. After the teacher network's weights are updated, input to a layer for both teacher and student networks become equal, $\mathbf{x_d^t} = \mathbf{x_d^s}$. We use early stopping during this optional retraining and we find that a few dozen of iterations suffice.

3 Hardware

We devised and implemented on FPGA, a hardware architecture that exploits the fact that input values and neuron weights are restricted to ternary values, $\{-1, 0, +1\}$. These values are represented on 2 bits using usual two's complement encoding. Figure 2 illustrates the implementation of a 3-layer NN. The design is split into several blocks connected to each other. These blocks form a pipeline that correspond to the sequence of the NN processing steps. We assume that a given NN configuration is used for a large number of classification operations. Thus, for area and power efficiency reasons, we can exploit embedded memory blocks for storing at run-time, the neuron weights and output ternarization thresholds b^{lo} and b^{hi}. The compute part of each neuron is an instance of a small component containing a ternary multiplier (two logic gates) and a small accumulator (a few tens of gates). All neurons work in parallel so that one new item is processed per clock cycle. Since layers are pipelined, each of them simultaneously work on a different set of inputs, i.e. layer 2 processes image n while layer 1 processes image $n + 1$. The ternarization block processes the neuron outputs sequentially, so it simply consists of two signed comparators and a multiplexer.

When using FPGA technology, the actual board can be chosen according to the accuracy/throughput/power/price trade-off required. In this paper, we use the Sakura board [11] for experimenting. It can accommodate a 1024 neuron, 3-layer NN using 81% of the Kintex-7 160T FPGA. With a 200 MHz clock frequency and frames of size 784 (MNIST dataset), the throughput (here limited by the number of neurons) is 195 K images/s with a power consumption of 3.8 W and a classification latency of 20.5 µs.

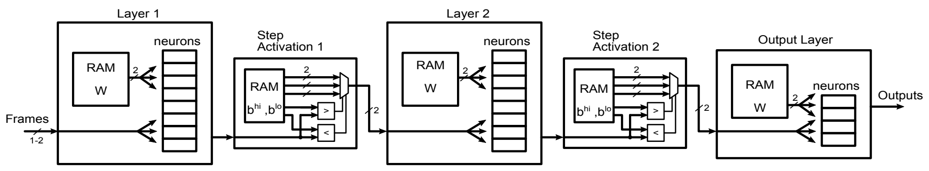

Figure 2: Hardware implementation scheme of a 2-layer neural network

4 Experiments

We perform our experiments on the MNIST database of handwritten digits [12], a well-studied database for benchmarking methods on real-world data. MNIST has a training set of 60K examples, and a test set of 10K examples of 28x28 gray-scale images. We use the last 10K samples of the training set as a validation set for early stopping and model selection.

We experiment with both multi-layer perceptrons (MLP) in a permutation-invariant manner and convolutional neural networks (CNN). For the MLPs, we experiment with different architectures in terms of depth and neuron count. We use 250, 500, 750, and 1000 neurons per layer for 2, 3, and 4 layer networks. For the CNNs, we use a LENET-like architecture with 10 and 15 filters in the first and second convolutional layers, followed by two fully connected layers with 100 and 10 neurons. Our main goals of the experiments are to demonstrate, (i) the performance of the ternarization procedure with respect to the real-valued teacher network, (ii) the classification performance of TNNs on MNIST, and (iii) the resource-efficiency, speed and throughput of TNNs deployed on our purpose-built hardware.

For that reason, we only use vanilla versions of the networks. We minimize cross entropy loss using stochastic gradient descent with a mini-batch size of 100. During training we use random rotations up to ± 10 degrees. We report the test error rate associated with the best validation error rate after 1000 epochs. We do not preform any preprocessing on the dataset, other than binarization which is a requirement of our hardware.

4.1 Ternarization Performance

The ternarization performance, is the ability of the student network to imitate the behavior of it's teacher. We measure this by using the accuracy difference between the teacher network and the student network. Table 2 shows this difference between the teacher and student networks on training and test sets for three different exhaustive search threshold values. $\varepsilon = 1$ corresponds to the fully exhaustive search case whereas $\varepsilon = 0$ represents fully dichotomic search. The results show that the ternarization performance is better for deeper networks. Since we always use the teacher network's original output as a reference, errors are not amplified in the network. On the contrary, deeper networks allow the student network to correct some of the mistakes in the upper layers, dampening the errors. Also, we perform a retraining step with early stopping before ternarizing a layer, since it slightly improves the performance. The ternarization performance generally decreases with lower ε threshold values, but the decrease is marginal. On occasion, performance has been seen to increase. These are due to teacher network's weight update, that allows the network to escape from a local minima. In order to demonstrate the effect of ε in terms of run-time and classification performance, we conduct a detailed analysis without the optional staggered retraining. Figure 3 shows the distribution of the ratio of neurons that are ternarized exhaustively with different ε, together with the performance gaps

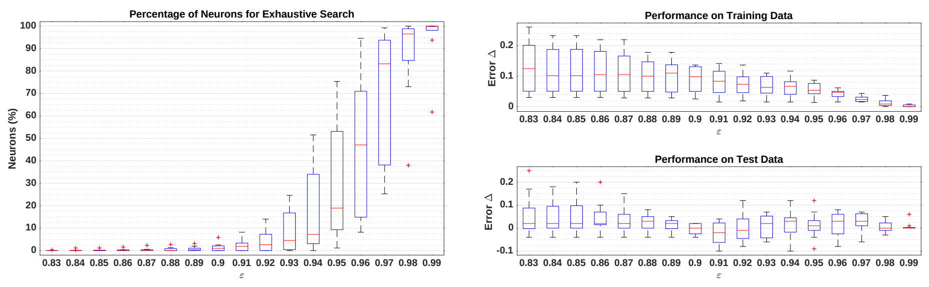

Figure 3: The effect of threshold values on run-time and classification performance

on training and test datasets. The optimal trade-off is achieved with $\varepsilon = 0.95$. Exhaustive search is used for only 20% of the neurons, and the expected value of accuracy gaps is practically 0. For the largest layer with 1000 neurons, the ternarization operations take 2 min and 63 min for dichotomic and exhaustive search, respectively, on a 40-core Intel(R) Xeon(R) CPU E5-2650 v3 @ 2.30GHz server with 128 GB RAM. For the output layer, the ternarization time is reduced to 21 min with exhaustive search.

4.2 Classification Performance

The classification performance on the MNIST dataset is provided in Table 3. We also compare ternary NNs performance to other solutions that exist in the literature that we discuss in Section 5 in more detail. We trained Binarized NNs using the code provided by the authors. In order to allow a fair comparison, we binarize the input to their algorithm as required by our purpose-built hardware. We also cite the reported performance of Bitwise NNs [6] with 1024 neurons in 3 layers. For EBP, we use the results provided in [13] and map the results to the closest architecture in our experiments. Note that the reported results are

Table 2: Accuracy gap due to ternarization for different ε

		$\varepsilon = 1$		$\varepsilon = 0.95$		$\varepsilon = 0$	
# Neurons	**# Layers**	**Train**	**Test**	**Train**	**Test**	**Train**	**Test**
250	1	0.63	0.75	0.69	0.76	0.72	0.98
	2	0.30	0.44	0.25	0.56	0.15	0.54
	3	0.08	0.47	0.10	0.62	0.03	0.52
500	1	0.50	0.94	0.49	0.83	0.48	0.70
	2	0.23	0.36	0.29	0.39	0.26	0.49
	3	0.12	0.25	0.07	0.34	0.10	0.20
750	1	0.27	0.90	0.27	0.90	0.29	1.05
	2	0.00	0.56	-0.01	0.65	0.00	0.67
	3	-0.02	0.43	-0.03	0.44	-0.03	0.26
1000	1	0.44	0.66	0.60	0.87	0.79	0.95
	2	-0.02	0.59	-0.07	0.33	-0.05	0.37
	3	-0.16	0.29	-0.14	0.34	-0.14	0.34

Table 3: Error Rates on MNIST

# Neurons	250			500			750			1000		
# Hidden Layers	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**	**1**	**2**	**3**
Ternary NN	2.24	1.87	1.86	2.25	1.86	1.71	2.27	1.90	1.67	2.37	1.91	2.11
Binarized NN [7]	6.20	4.44	3.83	3.97	2.70	2.47	3.32	2.18	1.88	2.94	1.94	1.74
Bitwise NN [6]												1.36
EBP [13]	4.63	3.97		3.45	2.89		3.10	2.68		3.08		
TrueNorth [8]		7.30										

obtained with normalized real-valued input and real-valued weights. We know that using real-valued input can be a game changer in terms of classification performance although it is not fully energy-efficient. For TrueNorth [8], we only cite the relevant accuracy results in Table 3. A more detailed comparison is provided in terms of energy efficiency in the following section.

The results show that the other methods fail to perform well with smaller networks. Ternary NNs, however, consistently outperform other techniques, with the exception of the Bitwise NN with 3 layers of 1024 neurons. The sole reason for this, is that our teacher network's performance for the largest configuration is already worse than other configurations. Since we do not impose any restrictions on the weights during initial training, bigger networks decrease the generalization performance due to over-fitting. With more appropriately sized networks, our final performance is much better. In that respect, we argue that TNNs avoid the need for over-parametrization and performs equally well even with the smaller networks, which are more suited for MNIST. For instance, the maxout networks' best accuracy on MNIST is achieved with a two layer network that has 240 neurons in each of them [14].

For the convolutional TNNs, the teacher network's performance is 97.84%, whereas the student TNN achieves 96.58%. The performance gap between the teacher and the student in CNN case is higher than MLPs. This is due to the weight sharing concept in CNNs. The number of neurons and the weights are much smaller than MLPs but they are replicated all over the input space. This makes student network's job slightly harder in mimicing the original CNNs behavior. We plot example filters for convolutional TNNs in Figure 4. Like their real-valued counterparts, convolutional TNNs also learn more generic shape descriptors in the first layer and more specialized filters in the second layer.

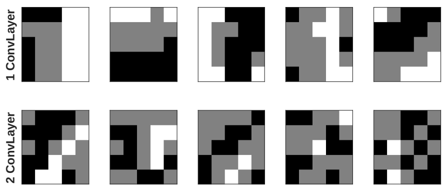

Figure 4: Sample filters in convolutional TNN

The performance of TNNs for both MLPs and CNNs are bounded by the performance of the teacher network. Since we do not impose any restrictions on the weights, state-of-the-art techniques can be used to further improve the performance. In this study, we consider only the vanilla versions of the networks and leave improving the performance of the teacher networks using data preprocessing techniques and optimized hyper-parameters as a future work.

4.3 Hardware Performance

The performance of our hardware solution in terms of latency, throughput and energy efficiency is given in Table 4. We know that TrueNorth can operate at the two extremes of power consumption and accuracy. It consumes 0.268 µJ with a network of low accuracy (92.7%), and consumes as high as 108 µJ with a committee of 64 networks that achieves 99.4%. Our hardware cannot operate at these two extremes, yet in the middle operating zone, we outperform TrueNorth both in terms of energy-efficiency - accuracy trade-off and speed. TrueNorth consumes 4 µJ with 95% accuracy with a throughput of 1000 images/s, and with 1 ms latency. Our TNN hardware, consuming 3.63 µJ achieves 98.14% accuracy at a rate of 255 102 images/s, and a latency of 8.09 µs. Moreover, if our FPGA design was built as an ASIC, it could use even less power by an order of magnitude [15]. Finally, we note that our hardware is capable of running any binarized network since TNNs are a more general form of their binary counterparts.

Table 4: Hardware performance on Sakura-X

# Neurons	**Throughput (images/s)**	**# Layers**	**Energy µJ (per image)**	**Latency µs (per image)**	**Accuracy (%)**
250	255102	1	1.24	5.37	97.76
		2	2.44	6.73	98.13
		3	3.63	8.09	98.14
500	255102	1	2.44	6.63	97.75
		2	4.83	9.24	98.14
		3	7.22	11.9	98.29
750	255102	1	3.63	7.88	97.73
		2	7.22	11.8	98.10
		3	10.8	15.6	98.33
1000	198019	1	6.22	10.2	97.63
		2	12.4	15.3	98.09
		3	18.5	20.5	97.89

5 Discussion and Related Work

Courbariaux et al. [16] propose the BinaryConnect (BC) method for binarizing only the weights, leaving the inputs and the activations as real-values. They use the back-propagation algorithm with an additional weight binarization step. In the forward pass, weights are binarized either deterministically using their sign, or stochastically. Stochastic binarization converts the real-valued weights to probabilities with the use of the hard-sigmoid function, then decides the final value of the weight with this. In the back-propagation phase, they use a quantization mechanism so that the multiplication operations are converted to bit-shift operations [17]. While this binarization scheme helps reducing the number of multiplications during training and testing, it is not fully hardware-friendly. More recently, the same authors extend their idea to the activations of the neurons also [7]. In Binarized NN, they use sign activation function for obtaining binary neurons.

Soudry et al. [18] propose Expectation Backpropagation (EBP), an algorithm for learning the weights of a binary network using a variational Bayes technique. The algorithm can be used to train the network such that, each weight can be restricted to be binary or ternary values. The strength of this approach is that the training algorithm does not require any tuning of hyper-parameters, such as learning rate as in the standard back-propagation algorithm. Also, the neurons in the middle layers are binary, making it hardware-friendly. However, this approach assumes the bias is real and it is not currently applicable to CNNs.

Kim and Smaragdis propose Bitwise NN [6] which is a completely binary approach, where all the inputs, weights, and the outputs are binary. They use a straightforward extension of back-propagation to learn bitwise network's weights. First, they train a real-valued network by constraining the weights of the network using *tanh*. They use a *tanh* non-linearity for the activations to constrain the neuron output to $(-1, 1)$. Then, they have a second training step for the binary network, using the real-valued network together with a global sparsity parameter. In each epoch during forward propagation, they binarize the weights and the activations of this binary network using the sign function on the original constrained real-valued parameters and activations.

Recently, IBM announced an energy efficient TrueNorth chip, designed for spiking neural network architectures [19]. Esser et al. [8] propose an algorithm for training networks that are compatible with IBM TrueNorth chip. Their algorithm is based on backpropagation with two modifications. First, they use Gaussian approximation for the summation of several Bernoulli neurons, and second, they clip the values to satisfy the boundary requirements of TrueNorth chip. They obtain ternary weights by introducing a synaptic connection parameter that determines whether a connection exits. If the connection exists, the sign of the weight is used. They use a threshold activation function to obtain binary neuron outputs.

In Table 5, we provide a comparison between the related works and our approach by summarizing the constraints put on the inputs, weights and the activations during training and testing. Unlike other studies, we ternarize the weights and neurons using a step function with two thresholds. In this way, we allow the network to prune the less important connections, and use only the most important weights' sign. For that reason, our method performs better on the smaller networks also, unlike the other methods. Since they use the sign function for binarization, small weights and larger weights have the same value in the binary network. In order to compensate the effects of this imbalance, they generally use over-parametrized networks to obtain good results. Our Ternary

NN shows that we can perform nearly as good as the real-valued valued network, even with smaller networks.

Table 5: Comparison of several approaches for resource-efficient neural networks

	Training			**Deployment**		
Method	**Inputs**	**Weights**	**Activations**	**Inputs**	**Weights**	**Activations**
BC [16]	$\mathbb{R}$	$\{-1,0,1\}$	$\mathbb{R}$	$\mathbb{R}$	$\{-1,0,1\}$	$\mathbb{R}$
Binarized NN [7]	$\mathbb{R}$	$\{-1,1\}$	$\{-1,1\}$	$\mathbb{R}$	$\{-1,1\}$	$\{-1,1\}$
EBP[18]	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}$	$\mathbb{R}$	$\{-1,0,1\}$	$\{-1,1\}$
Bitwise NN [6]	$(-1,1)$	$(-1,1)$	$(-1,1)$	$\{-1,1\}$	$\{-1,0,1\}$	$\{-1,1\}$
	$[0,1]$	$(-1,1)$	$(-1,1)$	$\{0,1\}$	$\{-1,0,1\}$	$\{-1,1\}$
TrueNorth [8]	$[0,1]$	$[-1,1]$	$[0,1]$	$\{0,1\}$	$\{-1,0,1\}$	$\{0,1\}$
Ternary NN	$\{0,1\}$	$\mathbb{R}$	$\{-1,0,1\}$	$\{0,1\}$	$\{-1,0,1\}$	$\{-1,0,1\}$

6 Conclusion

In this study, we proposed TNNs for resource-efficient applications of deep learning. Our TNNs have shown to outperform referenced resource-efficient DNNs with regards to accuracy. In tandem with our TNNs, our hardware, offers significant throughput and latency improvements too. Where both optimal classification accuracy and energy efficiency is required, we surpass previous works. We present a pipelined FPGA-based architecture that takes advantage of the assumption that the same NN configuration is reused for many operations. In doing so, embedded memory is exploited and parallel execution is performed on input streams. Resulting throughput of ternary DNNs exhibit 255 times the rate of TrueNorth. At the same time, it exhibits a lower power consumption per classification. We propose a teacher-student approach for training TNNs with weights constrained to $\{-1,0,1\}$. We allow each neuron to choose a sparsity parameter for itself, an opportunity to remove the weights that have very little contribution. In that respect, a TNN inherently prunes the unnecessary connections. This scheme helps to prevent over-parametrization observed in other variants of resource-efficient DNNs. Future research will focus on training better teacher networks using state-of-the-art techniques to further improve the accuracy of TNNs.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[2] Alex Graves, Abdel rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *ICASSP*, 2013.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[4] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems, NIPS*, pages 1135–1143, 2015.

[5] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[6] Minje Kim and Paris Smaragdis. Bitwise neural networks. In *International Conference on Machine Learning (ICML) Workshop on Resource-Efficient Machine Learning*, 2015.

[7] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[8] Steve K Esser, Rathinakumar Appuswamy, Paul Merolla, John V Arthur, and Dharmendra S Modha. Backpropagation for energy-efficient neuromorphic computing. In *Advances in Neural Information Processing Systems*, pages 1117–1125, 2015.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pages 448–456, 2015.

[11] Sakura-X Board, http://satoh.cs.uec.ac.jp/SAKURA/hardware/SAKURA-X.html. [online], *accessed* 8 May 2016.

[12] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[13] Zhiyong Cheng, Daniel Soudry, Zexi Mao, and Zhenzhong Lan. Training binary multilayer neural networks for image classification using expectation backpropagation. *arXiv preprint arXiv:1503.03562*, 2015.

[14] Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. *JMLR WCP*, 28(3):1319–1327, 2013.

[15] Ian Kuon and Jonathan Rose. Measuring the Gap Between FPGAs and ASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, Feb 2007.

[16] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3105–3113, 2015.

[17] Zhouhan Lin, Matthieu Courbariaux, Roland Memisevic, and Yoshua Bengio. Neural networks with few multiplications. In *ICLR*, 2016.

[18] Daniel Soudry, Itay Hubara, and Ron Meir. Expectation backpropagation: Parameter-free training of multilayer neural networks with continuous or discrete weights. In *Advances in Neural Information Processing Systems, NIPS*, pages 963–971, 2014.

[19] Paul A Merolla, John V Arthur, Rodrigo Alvarez-Icaza, Andrew S Cassidy, Jun Sawada, Filipp Akopyan, Bryan L Jackson, Nabil Imam, Chen Guo, Yutaka Nakamura, Bernard Brezzo, Ivan Vo, Steven K. Esser, Rathinakumar Appuswamy, Brian Taba, Arnon Amir, Myron D. Flickner, William P. Risk, Rajit Manohar, and Dharmendra S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, 2014.