# TOT-Net: An Endeavor Toward Optimizing Ternary Neural Networks

Najmeh Nazari*, Mohammad Loni+, Mostafa E. Salehi*, Masoud Daneshtalab+, Mikael Sjodin+
*School of Electrical and Computer Engineering, University of Tehran, Tehran , Iran.
+School of Innovation, Design and Engineering, Malardalen University Vasteras, Sweden
{najme.nazari, mersali}@ut.ac.ir,{mohammad.loni, masoud.daneshtalab, mikael.sjodin}@mdh.se

***Abstract*—High computation demands and big memory resources are the major implementation challenges of Convolutional Neural Networks (CNNs) especially for low-power and resource-limited embedded devices. Many binarized neural networks are recently proposed to address these issues. Although they have significantly decreased computation and memory footprint, they have suffered from accuracy loss especially for large datasets. In this paper, we propose TOT-Net, a ternarized neural network with [-1, 0, 1] values for both weights and activation functions that has simultaneously achieved a higher level of accuracy and less computational load. In fact, first, TOT-Net introduces a simple bitwise logic for convolution computations to reduce the cost of multiply operations. To improve the accuracy, selecting proper activation function and learning rate are influential, but also difficult. As the second contribution, we propose a novel piece-wise activation function, and optimized learning rate for different datasets. Our findings first reveal that 0.01 is a preferable learning rate for the studied datasets. Third, by using an evolutionary optimization approach, we found novel piece-wise activation functions customized for TOT-Net. According to the experimental results, TOT-Net achieves 2.15%, 8.77%, and 5.7/5.52% better accuracy compared to XNOR-Net on CIFAR-10, CIFAR-100, and ImageNet top-5/top-1 datasets, respectively.**

**Keywords—convolutional neural networks, ternary neural network, activation function, optimization.**

## I. INTRODUCTION

Convolutional Neural Networks (CNNs) have attained the state of the art results in many application domains, especially in computer vision tasks such as image and video classification [1] [2], object recognition [3], and image segmentation [4]. For providing more accurate results, CNNs are becoming more sophisticated containing hundred deep layers and millions floating-point operations. Therefore, CNNs suffer from huge amount of memory accesses and require significant processing capacity.

CNNs have been used in a wide spread spectrum of applications, such as deployment of CNN on mobile, wearable devices, and Internet of Things (IoT) platforms. On the other hand, embedded devices have strict constraints on the computation resource and power consumption that are not compatible with the requirements of CNNs. Plus, embedded applications usually have real-time constraints which necessitates compact neural networks to meet resource, power, and response-time constraints. MobileNets [5] and SqueezeNet [6] are popular compact networks which are introduced for smartphones and embedded vision devices. In addition, many prior works attempted to reduce the computational cost and memory footprint of CNNs by decreasing accuracy. Generally, the efficiency of the CNN implementation can be enhanced via the following techniques:

- CNN hardware accelerators try to overcome these challenges by parallel computing, and efficient data reuse [7], [8], [9].
- Pruning techniques such as [10], [11], [12] have eliminated redundant and ineffective weights to reduce the amount of computation, and afterward, reclaim the accuracy through fine-tuning.
- Customized CNN architectures are designed for resource budget limitations [13], [14].
- CNN parameters quantization is a popular approach to diminish the amount of computation, data storage and transfer time [15], [16], [17].

In this paper, we focus on the quantizing techniques, since quantized architectures are extremely suitable for embedded devices such as smartphones, without considerable accuracy loss. Table I compares different outstanding methods including the full-precision network AlexNet [1], BNN [18], XNOR-Net [19], and our proposed network named TOT-Net.

TABLE I. COMPARISON OF DIFFERENT QUANTIZATION METHODS ON IMAGENET DATASET.

| Methods | Operations Used in CNN | Memory Saving (Inference) |
|---|---|---|
| AlexNet [1] | +, -, × | 1x |
| BNN [20] | XNOR, bitcount | ~32x |
| XNOR-Net [23] | XNOR, bitcount | ~32x |
| **TOT-Net** | XOR, AND, bitcount | ~16x |

The quantization has demonstrated to be quite effective due to:

- Compressing the network theoretically up to 32×, when compared with full precision floating point networks.
- Reducing the computation complexity and accelerating the inference time by replacing the 32-bit floating point multiply-accumulations with bitwise operations.

Although the binary quantization methods provide considerable efficiency, they suffer from accuracy loss, especially in large datasets [20]. To tackle this challenge, we propose TOT-Net, a ternarized neural network with ternary weights and activations. TOT-Net benefits from sparsity by adding zero states to both weights and activations. Exploiting the zero states, TOT-Net, disregard the useless computations by an enable signal, hence energy consumption is decreased

more than BNNs and would make it more suitable for embedded systems. Moreover, we exploit a simple logic instead of multiplication operations which matches with the standard sign and magnitude representation. In addition, to obtain higher accuracy level, we conduct a two-level optimization strategy for finding the most proper network activation functions and learning rates for TOT-Net. In nutshell our main contributions in this paper are three-fold:

- We introduce TOT-Net, a novel ternarized neural network architecture that provides higher level of accuracy, less computational load, and simpler computation units.
- We present an automatic method to find the most proper ternary activation functions. To do so, we utilize an evolutionary optimization approach to efficiently explore the design space. In addition, we have analyzed the impact of learning rate on the accuracy of Ternary Neural Networks (TNNs).
- We evaluate the impact of TOT-Net on three popular classification datasets including ImageNet, CIFAR-100, and CIFAR-10. The evaluation results demonstrate huge improvement over typical used baselines.

The rest of this paper is organized as follow. In Section II, we overview a background on CNN, ternary weight networks, and evolutionary optimizations for deep neural networks. Section III reviews related works in this scope. Our proposed ternary neural network, its architecture, and its optimizations are presented in Section IV. Section V presents and discusses the experimental results. Finally, we conclude this essay in Section VI.

## II. BACKGROUND

This Section briefly outlines CNN, piece-wise activation functions and ternary weight networks.

### *1) Convolutional neural networks:*

Convolutional neural networks are typically comprised of a combination of three main layers that are called convolutional layers, pooling layers, and fully connected layers. A significant amount of computations, over 90%, are performed in the convolutional layers whereas fully connected layers are mainly memory bounded [21]. It should be mentioned that some new proposed CNN architectures such as ResNet [22] and NIN [23] have removed fully-connected layers because of remarkable energy consumption of memory accesses in this layer. Quantized networks would also reduce memory footprint and hence improve the energy efficiency [15][24].

The convolutional layer is the principal layer of CNNs which extracts high-level abstraction of its inputs called feature map by using various filters. Equation (1) demonstrates the operation of a 3D convolutional layer that convolves the inputs via a filer $W \in R^{C\times X\times Y}$ for each feature map where C, X and Y are the number of input channels and spatial dimensions of the filter, respectively. It is obvious that a lot of multiply and accumulate (MAC) operations are required to just obtain one point of the output feature map.

$$conv3D = f_{act}\Big(\sum_{k=0}^{C-1}\sum_{i=0}^{X-1}\sum_{j=0}^{Y-1} I[k][X-i][Y-j] \times W[k][i][j]\Big) \qquad (1)$$

Where *conv3D*, *I* and *W* are the output feature maps, input feature maps, and $k\times k$ weight filters, respectively. *U* determines the stride size that is the step for moving filters over the input feature maps. Pooling layers perform down-sampling on data to decrease the amount of computation. Commonly, in CNNs, pooling layers such as max pooling and mean pooling are used after some convolutional layers. As demonstrated by their names, max pooling selects the maximum feature map and mean pooling computes the average of feature maps in the pooling window. Mostly, after distinguishing high-level abstraction features, fully connected layers are applied in the CNN to classify images.

### *2) Piece-wise activation functions:*

In general, each neuron computes the summation of its weighted inputs and then performs a function called activation function. In general, nonlinear activation functions are better suited for classification tasks [13].

TABLE II. CANDIDATE PIECE-WISE ACTIVATION FUNCTIONS. BY Y(X < 0) AND Y(X ⩾ 0) WE INDICATE THE LEFT- AND RIGHT-PIECE, RESPECTIVELY.

| Activation Function | Expression |
|---|---|
| HardTanh | $y(x) = \max(-1, x)$ |
| ELiSH | $y(x<0) = \frac{e^x - 1}{1 + e^{-x}}$ *and* $y(x \geq 0) = \frac{x}{1, e^{-x}}$ |
| Swish | $y(x) = \frac{x}{1 + e^{-x}}$ |
| ReLU | $y(x) = \max(x, 0)$ |
| ELU | $y(x<0) = e^x - 1$ *and* $y(x \geq 0) = x$ |
| SeLU | $y(x<0) = \lambda\alpha(e^x - 1)$ *and* $y(x \geq 0) = \lambda x$ |
| Tanh | $y(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$ |
| PRelu | $y(x) = \begin{cases} ax & for\ x < 0 \\ x & for\ x \geq 0 \end{cases}$ |
| Leaky_Relu | $y(x) = \begin{cases} 0.01x & for\ x < 0 \\ x & for\ x \geq 0 \end{cases}$ |
| Sigmoid | $y(x) = \frac{1}{1 + e^{-x}}$ |
| Linear | $y(x) = x$ |

The design of these activation functions was inspired by the recently proposed Swish activation function [25]. These functions are unbounded, smooth, and non-monotonic [25]. Furthermore, good data flow through CNN is provided by mentioned functions [26]. A family of activation functions such as *Swish(x)* has been recognized by [25] and [26], which betters the information propagation and solves the vanishing gradient issue. Inspired from Swish, negative and positive inputs have a different influence on learning accuracy. Equation 2 has described the piece-wise activation function, named F(x), consisting two separate activation functions. Table II presents commonly used activation functions which

can be used as the left and right parts of the piece-wise activation function.

$$F(x) = (\mathrm{F1} \quad if\ x \geq 0,\ \ F2 \quad if\ x < 0) \tag{2}$$

The optimization problem will be highlighted when the training dataset is giant, and the optimization process needs prior knowledge. Section IV.B represents an efficient metaheuristic optimization approach that automatically tunes networks activation functions. To the best of our knowledge, this is the first try to optimize the piece-wise activation function for ternary neural networks.

*3) Ternary weight network:*

BinaryConnect lets weights to be {-1, 1}, hence, the whole learned weights must be -1 alternatively 1. However, during the training phase, many learned weights are zero or near zero. It arises from the stochastic sampling process which permits that the average value of sampled weight would be zero. On the other hand, the network becomes sparse if weight values are zero. This mean, we can disregard operations related to zero terms. In these circumstances, the gating technique can be used to avoid unnecessary energy consumption. Therefore, Ternary Weight Network (TWN) [27] in comparison with BinaryConnect, profits from the mentioned advantage by adding an extra zero state.

Commonly, there exist two approaches for ternarizing weights that are explained as follows. It should be mentioned that full precision weights must be clipped into the [-1, 1] interval. Equation (3) shows the first method in which $r$ specifies the level of sparsity. This approach is similar to the deterministic approach for binarizing values and can be implemented by a simple comparator.

$$Wt = \begin{cases} +1 & if\ W \geq \mathrm{r}, \\ 0 & if\ |W| \leq r, \\ -1 & if\ W \leq -\mathrm{r}. \end{cases} \tag{3}$$

The second approach which is roughly analogous to the stochastic approach of binarizing values, is demonstrated in Equation (4). Basically, real value weights stand into two sub-interval [-1, 0] and (0, 1] that their ternary values are determined according to their distance from -1 and 1. Where $Wt$ is a ternarized weight, $W$ is the full precision weight and $P$ is the probability function.

$$Wt = \begin{cases} P(Wt = 1) = W;\ P(Wt = 0) = 1 - W & if\ W \geq 0 \\ P(Wt = -1) = -W;\ P(Wt = 0) = 1 + W & if\ W < 0 \end{cases} \tag{4}$$

Hardware implementation of the former is simpler than the latter. However, the stochastic approach is generally more accurate than the deterministic one. Thus, depending on the application, we can select the ternarization method. It is worth noting that the scaling factor of the ternarized weights plays a crucial role to compensate the accuracy loss. By substituting full precision values with ternary values {-1, 0, 1}, all multiplications in the inference phase are removed and convolutions are just calculated by simple accumulations and multiplexers instead of MAC operations like BinaryConnect. Furthermore, TWN is more energy-efficient due to its sparsity that arises from disregarding operations related to zero value weights. Also, TWN saves 16× more memory storage and access for weight parameters compared to the full precision one. To put it simply, just 2 bits are required for weights instead of 32 bits.

## III. Related Work

### A. Network Quantization

Generally, binarized neural networks (BNNs) suffer from the accuracy loss, especially in large datasets. [20] tried to address this issue for BNNs by proposing an efficient training strategy. Some works such as [28] and [29] applied reduced precisions to reduce memory storage and computation. However, they still require computation-intensive MAC operations to perform the convolutional operations, hence, suffer from heavy operations. Bit Fusion [15] demonstrated that 8 bits and less than it is enough for weights and activations in a wide range of CNNs, especially for small datasets. [16] uses bit widths less than 6 bits for quantization and achieves good accuracy compared to the full precision CNNs. LQ-Nets [17] applies proper quantization bits by a learnable quantizer and ReLeQ [30] automates DNN quantization based on a Reinforcement Learning (RL) algorithm, respectively. Ristretto [31] as a CNN approximation framework allows experimental exploration to trade between the classification accuracy and the bit-width of weights and activations. Ristretto also concludes that 8-bit dynamic fixed-point operations are appropriate for large-scale image classifications such as ImageNet.

Some recent researches [18][19][32] surpass quantized neural networks and have aggressively reduced precision even till 1 bit. BinaryConnect [32] eliminates multiplication in the forward pass by substituting full precision weights with -1 and 1 values. Ternary weight network [27] achieves more accuracy by applying ternary weights {-1, 0, 1}. Also, it adds sparsity to network, hence, it is more energy-efficient than binary connect. [33] presents Sparse Ternary Connect (STC) which reduces computation complexity by raising sparsity, and just leads less than 0.5% accuracy loss. Binarized neural network [18] binarizes both activations and weights and substitutes computation-intensive MAC operations with XNOR-bitcount operations. Therefore, the computations are drastically reduced, and also memory footprint has been intensely decreased. XNOR-Net [19] achieves more accuracy compared to BNN by using scaling factors for both activations and weights. TBN [34] proposes using ternary activations and binary weights, and hence, attains more accuracy for the ImageNet dataset compared to XNOR-Net. Since binarized neural networks can meet the embedded device constraints, hardware implementation of BNN has recently gotten more attention. XNOR Neural Engine [24] and BRein [35] are two samples in this area.

### B. Neural Network Optimization

To enable more accurate learning results, selecting the architectural parameters of CNNs are crucial since the network architecture strongly affects the inference time, memory

footprint, the accuracy level, and the network generalization proficiency. However, the hand-crafted designing of CNN parameters is overwhelming due to requiring a lot of trial-and-error and deep expertise since the design space is huge. Therefore, an automatic method for designing CNN architectures has emerged as a significant alternative for decreasing efficiency risk and design cost.

There are different automated neural optimization approaches, including random search, Bayesian optimization, RL, neuro-evolutionary methods. Using random search is challenging due to extremely random sampling in the search space [26], while Bayesian-based methods suffer from immense computational cost, suitable only for searching architectures with a fixed-length space and focuses on low-dimensional continuous problems [36]. RL-based methods are mainly slow and require considerable computational resources in both exploration and training steps [36] [26]. Evolutionary algorithms are feasible solutions for optimizing the neural architecture due to exploring improved design space without any prior assumptions [26]. [36] proposed a multi-objective evolutionary solution to reduce the complexity of CNNs, while tries to increase the accuracy. In addition, [13] proposed two new activation functions, ELiSH and HardELiSH, for tiny-ImageNet by leveraging an evolutionary method. In this paper, we use a similar strategy to optimize the ternary activation functions.

## IV. ARCHITECTURE

As mentioned in the prior section, although, TWN doubles memory storage for weight parameters compared to BinaryConnect, it surpasses BinaryConnect in terms of accuracy by adding a further zero state to {-1, 1}. Moreover, it is more energy-efficient due to eliminating computations related to zero parameters. In this section, we comprehensively explain the proposed ternary neural network architecture in part A. Then, in part B, we describe the optimization algorithm for finding the best piece-wise activation function.

### *A. Ternary Neural Networks*

As mentioned in Section I, binarized neural networks (BNNs) substitute MAC operations with just bitwise operations (XNOR-bitcount) by applying {-1, 1} for both weights and activations. In this work, we propose a ternarized neural network (TNN) that ternarizes both weights and activations. GXNOR-Net [37] also uses the ternary values for weights and activations and introduces a method that is similar to XNOR-Net approach. This means that they use XNOR-bitcount operations instead of MAC operations. Thus, they must apply a coding which is compatible with these operations. As shown in Table III, XNOR-Net encodes sign values -1 and 1 by 0 and 1, respectively. GXNOR-Net encodes -1 and 1 by 00 and 11, respectively and 0 can be encoded either by 01 or 10. Furthermore, [38] proposes ternary neural network which applies the teacher-student approach. They use ternary activations and full precision weights in the teacher network and use ternary weights and activations in the student network. However, our proposed method uses ternary weights and activations during both the training and inference phases. We use the standard sign and magnitude coding to encode ternary value {-1, 0, 1} according to Table III. This means that we encode -1 and 1 by 11 and 01, respectively. Also, either 00 or 10 are used for encoding 0.

TABLE III. XNOR AND GXNOR CODING VERSUS SIGN AND MAGNITUDE CODING.

| | **XNOR-Net** | **GXNOR-Net** | **Sign and Magnitude** |
|---|---|---|---|
| -1 | 0 | 00 | 11 |
| 0 | - | 01 or 10 | 00 or 10 |
| 1 | 1 | 11 | 01 |

By using sign and magnitude coding, we present a new simple logic instead of multiplication. Actually, by applying 00 or 01 for 0 values, multiplications with at least one zero-operand would be simply disregarded. As shown in Figure 1, we substitute multiplication with the XOR and AND gates to calculate most and least significant bits, respectively. MSB and LSB stand for most significant and least significant, respectively.

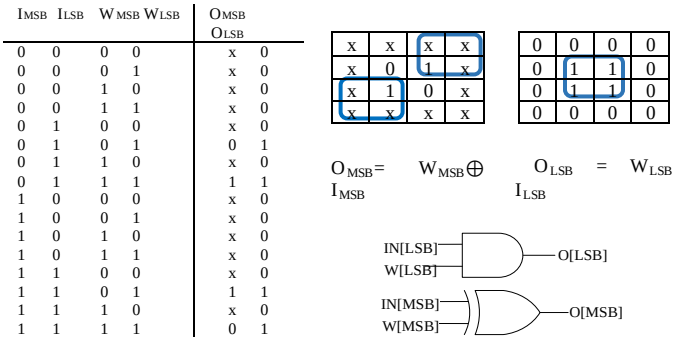



Fig. 1. The proposed multiplier unit.

Compared to TWN, TNNs increase sparsity due to adding zero activation, as well. Therefore, there is no need to calculate operations related to either zero weights or zero activations. To put it simply, if both activation and weight are non-zero, the operation is performed. The sparsity of TNNs is illustrated in Figure 2 with an example. As Shown in Figure2, only computation of continuous lines must be done, and others are unnecessary. In this Figure, X and Y are inputs and outputs, respectively. Besides, neuron and weight are demonstrated by circle and line, respectively.

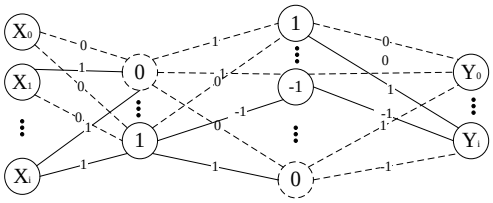
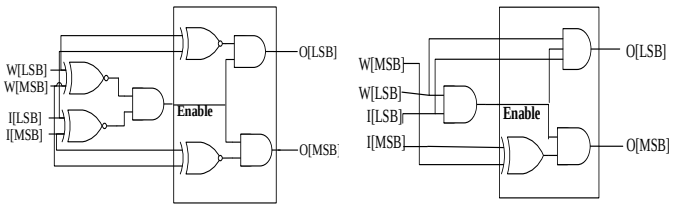



Fig. 2. Neurons and their synaptic weights in ternary neural networks, dot lines illustrate unnecessary computations.

We used the enable signal as a controller to avoid inessential computations and hence, reduce energy consumption. The enable signal is used for both activations and weights. In other words, the enable signal is active if both weight and activation are non-zero. By using the GXNOR-Net coding, the control gates which specify enable signals are more complicated compared to our proposed approach. Figure 3 (a) and (b) illustrate multiplication operations with the enable

signal which avoids inessential computations in GXNOR-Net and our proposed method. We can implement one multiplication and control gate with four logic gates. By this technique we roughly halve the number of gates compared to GXNOR-Net by using sign and magnitude coding as shown in Figure 3. Moreover, weights are constant during inference phase since they are obtained during the training phase. This means that we can prune all zero weight operations and for the remaining weights, just the LSB of the activation specifies the enable signal and there is no need to use extra control gates for the enable signal in this case. Since the inference phase is performed in embedded devices (mentioned in Section I), our proposed method is appropriate for them and multiplication is just implemented by three logic gates. To put it simply, as shown in Figure 3, by removing zero values from weights, the LSB of inputs are used as an enable signal, hence, one AND gate is removed and three logic gates are adequate. Furthermore, the memory storage is lowered in the inference phase by omitting the zero weights.

Since weights and activations are represented by two bits, GXNOR-Net uses twice XNOR gates for multiplication operations compared to XNOR-Net and uses three gates for determining the enable signal. We have exploited two gates (AND and XOR) instead of two XNOR gates, one AND gate as a control gate just during the training phase and another AND gate to apply the enable signal to outputs.




(a) (b)

Fig. 3. Multiplication operation in (a) GXNOR-Net. (b) our proposed logic in TOT-Net.

Since the enable signal avoids the computation of zero operands, the multiplication results can just be 01 (1) and 11 (-1). Therefore, instead of accumulation, we would only count the inverse of the MSB bits to achieve the results.

As discussed in Section II, there are two approaches (stochastic and deterministic methods) for ternarizing values. For ternarizing activations, GXNOR-Net applies deterministic approach according to Equation (2) and uses constant *r* depending on the sparsity level. For instance, by exploration on different *r* values for activations on the MNIST dataset, they obtain 0.7 as an appropriate value for *r*. For each dataset, we must explore various *r* values to find a proper value for it. Therefore, this approach is not much effective. Results of GXNOR-Net show that the accuracy loss grows by escalating the sparsity. Also, it uses complicated equations for ternarizing weights. [38] uses the stochastic and deterministic methods for ternarizing activation in teacher and student network, respectively. [33] selects 1/3 as a constant value for *r* during the inference phase to simply implement the deterministic ternarization. It should be noted that [33] just ternarizes weights. Since the fixed *r* is used for all datasets and architectures, it slightly losses the accuracy which has been tolerated.

In this work, we apply the deterministic method to both activation and weight ternarizations. There is a fundamental difference between our deterministic methods and the aforementioned methods. In fact, we use Δ instead of constant r in Equation (3) that can be varied depending on weight and activation values. Therefore, our proposed network becomes more accurate. Based on [27] which demonstrates roughly proper Δ values for weights, we use Equation (5) to obtain the suitable value of both activations and weights depending on their values.

$$\Delta = \frac{0.7}{n} \sum_{i=1}^{n} |X_i| \qquad (5)$$

Where Δ is a threshold parameter, n is the total number of activations or weights in a filter, and X denotes either activations or weights. Note that, Δ changes continuously for each filter and the activation window during the training and test phases. As mentioned before, the scaling factor plays a crucial role in reducing the accuracy loss which is imposed by constraining weights and activations just to {-1, 0, 1}. Thus, we eliminate $|X| < \Delta$ values which are ineffective in the computation. Hence, the suitable scaling factor is obtained by calculating the average of absolute $|X| > \Delta$ values. Therefore, the scaling factor is computed by Equation (6).

$$\beta = \frac{1}{|n_\Delta|} \sum_{i \,\epsilon\, n_\Delta} |X_i| \qquad (6)$$

Where β is a scaling factor and $|n_\Delta|$ demonstrates the total number of $|X| > \Delta$.

CNNs' block commonly contains convolution layer, batch normalization, activation, and pooling, respectively. Based on XNOR-Net, for reducing the dynamic range of the activations and decreasing the information loss due to ternarization, we have normalized the activation before the ternary convolution.

### *B. Ternary Neural Networks Optimization*

Existing approaches to design activation functions lack theoretical foundation which is hard to understand in the context of practical applications. Or they are based on inefficient search schemes, since there is no guarantee that NP-hard complex problems such as Neural Architectural Search (NAS) problem can be solved in a satisfactory manner in a limited time. The neuro-evolutionary methods are iterative population-based exploration solutions mimicking the process of natural selection and evolution where the characteristics of the process can be utilized in solving optimization problems. In this paper, we leverage a neuro-evolutionary solution to automatically explore the design space of the network activation function.

All evolutionary methods have an initial population where selection, crossover, mutation operators are applied to initial population for producing improved population. The operations will be repeated until satisfying the user criteria (reaching

suitable results) or stopping after a predefined number of iterations. Genetic Algorithms (GA) is a popular evolutionary method which can locate a near-optimal solution. Algorithm 1 represents pseudo-code of the GA for optimizing activation functions explained as following steps: **Step 1. Generating initial population:** Creating an initial population $U_0$ with the size N. To generate the initial random population, the network hyperparameters are represented as a string of genomes using a direct encoding shown as the genome type in Figure 4. **Step 2. Fitness evaluation:** Calculating the accuracy (fitness values) of the individuals in the $U_0$. **Step 3. Crossover:** GA randomly selects two genomes from the population set based on a certain crossover rate. Then two genome strings exchange parts of their corresponding chromosomes to create two new genomes. Step **4. Mutation:** The main goal of the mutation operator is to increase genetic diversity and forces GA to get rid of local optima. For doing the mutation, we need to randomly select one gene in the chromosome and modify its assigned value to a new valid number. **Step 5. Selection:** Obviously the individuals with better fitness values are selected as the next generation and the others will be removed from the population set.

**Algorithm : Pseudo Code of GA**

**Input:** • Piece-wise activation function candidates
• $N$: Population Size
• $T$: Maximum Number of Iterations
**Output:** The most proper activation function
**Function** GA ($N$, $T$) :
  (***Step 1***): $U_0$= **Random_Population** (N); //Creating initial random population.
  (***Step 2***): **Fitness_Function** ($U_0$); //Evaluating the fitness value of each individual in $U_0$
  $t = 1$;
  **while** $True$ **do**
    (***Step 3***): $U'_t$ = **Crossover** ($U_t$)
    (***Step 4***): $U''_t$ = **Mutation** ($U'_t$)
    (***Step 5***): $U_{t+1}$ = **Select** ( $U''_t$); //Select the best chromosomes from the $U''_t$.
    (***Step 6***): **if** $t > T$ **then**
      $Break$;
    $t = t + 1$;
    (***Step 2***): **Fitness_Function** ($U_{t+1}$);
    **return** The most accurate individual in $U_T$

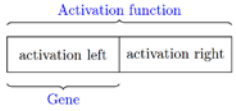



Fig. 4. Representing the piece-wise activation function in a genome type [35].

The specification of GA parameters is shown in Table IV. The design space roughly consists of 100 different design points. The average training time is one hour for each design point by employing an NVIDIA GTX 1080ti. Thus, leveraging exhaustive search is not reasonable since exploration takes 100 GPU-hours, while our search scheme requires 40 GPU-hours demonstrating 2.5x reduction in the exploration time.

TABLE IV. THE SPECIFICATION OF EVOLUTIONARY OPTIMIZATION STRATEGY AND NETWORK TRAINING PROCEDURE

| Evolutionary Optimization | | Network Training | |
|---|---|---|---|
| **Parameters** | **Value** | **Parameters** | **Value** |
| #iterations | 4 | Learning rate ($lr$) | {0.01, 0.001, 0.0001} |
| Population Size | 10 | Epochs | 320 |
| Mutation | One-Point | Batch Size | 256 |
| Crossover Rate | 0.2 | Optimizer | Adam |

## V. EXPERIMENTAL RESULTS

The purpose of our experiments is threefold. First, we will show the impact of our proposed ternarization on classification accuracy. Besides, we will compare the operational overhead of our proposed architecture with a different quantized convolutional neural network as a baseline used in other literatures. Second, we will present the optimized ternerized activation functions for different datasets. Moreover, the convergence of the evolutionary optimization for finding ternerized activation function will be demonstrated. Third, we explore the impact of different learning rates ($lr$) on the accuracy of TOT-Net. We run experiments on three Image classifications benchmarks consist of CIFAR-10, CIFAR-100, and ImageNet datasets that are totally different in term of the number of classes, number of samples, and complexities.

### A. The Results of Classification Accuracy

Table V and Figure 5 show the accuracy comparisons of different quantization methods on CIFAR-10, CIFAR-100 and ImageNet datasets. According to the results, TOT-Net provides 1.8%, 7.5%, and 5.7% more accuracy compared to XNOR-Net for AlexNet architecture on CIFAR-10, CIFAR-100, and ImageNet datasets, respectively. It should be mentioned that we trained both TOT-Net and XNOR-Net with 20 epochs for ImageNet on the same system to provide a fair comparison between XNOR-Net and TOT-Net. As shown in Figure 5, just the first 20 epoch of ImageNet training accuracy has been illustrated. Although TOT-Net has accuracy loss compared to full-precision networks, it provides up to 16x memory saving during the inference time. We predict TOT-Net achieves a better accuracy for the ResNet architecture on the ImageNet dataset compared to Binary Connect (BC) and Ternary Weight Network (TWN) since XNOR-Net's accuracy is higher than BC, and TWN. Figure 5 shows the variation of training accuracy over training epochs for TOT-Net and XNOR-Net. Obviously, TOT-Net overcomes XNOR-Net by providing a higher accuracy on all the epochs.

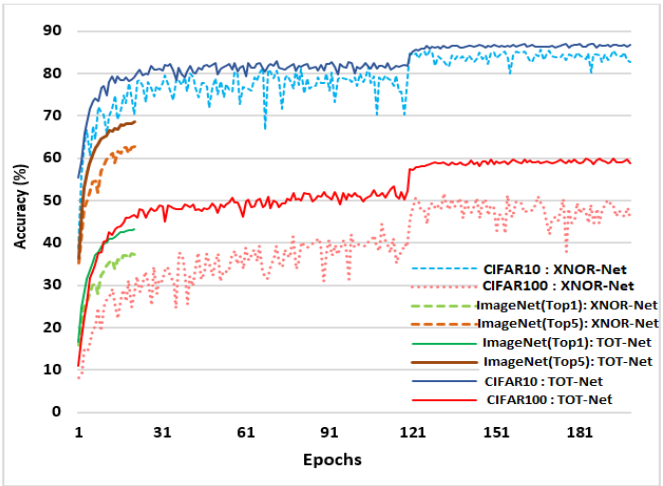



Fig. 5. Comparrison of training accuracy trends of TOT-Net and XNOR-Net on CIFAR-10, CIFAR-100, and ImageNet datasets.

We make an assumption that a neuron stands N inputs to estimate essential operations which are shown in Table VI. GXNOR-Net requires three gates to determine enable signal and perform the XNOR depending on the enable. Also, it

needs two more gates for applying the enable signal to outputs. Whereas in our proposed method just two 2-input AND gates are required to generate enable signal and apply it to outputs. It should be mentioned that in our proposed method, we just need <3N gates during the inference-test time for N as the input number of the neuron. Since we eliminate redundant weights, enable signals are just the least significant bits of the activations. Therefore, computations are reduced.

TABLE V. COMPARISON OF VALIDATION ACCURACY ON DIFFERENT NETWORKS AND DATASETS.

| Method | Accuracy %, (Neural Network Architecture) | | |
|---|---|---|---|
| | **CIFAR-10** | **CIFAR-100** | **ImageNet** § |
| Full-Precision [19] | 89.67, *(NIN)* | 64.32, *(NIN)* | 80.2/56.6, *(AlexNet)* +<br>89.2/69.3, *(ResNet-18)* + |
| BC [32] | 91.73, *(VGG-Net)* | NA | 66.3/43.1, *(ResNet-18)* +<br>61/35.4, *(AlexNet)* |
| TWN [27] | 92.56, *(VGG-Net)* | NA | 84.2/61.8, *(ResNet-18)* + |
| BNN [18] | 89.85, *(ConvNet)* | NA | 50.4/27.9, *(AlexNet)* + |
| XNOR-Net * [19] | 85.74, *(NIN)* | 54.10, *(NIN)* | 62.52/37.47, *(AlexNet)** +<br>73.2/51.2, *(ResNet-18)* |
| TNN [38] | 87.89, (VGG like) | 51.40, (VGG like) | NA |
| TOT-Net | **87.53,** *(NIN)* | **61.61,** *(NIN)* | **68.20/42.99,** *(AlexNet)** + |

* The experiments have been run by us (trained by 20 epochs).
+ Presenting both top-5 and top-1 accuracy, respectively.

TABLE VI. COMPARISONS OF OPERATION OVERHEAD IN VARIOUS ARCHITECTURES IN TERMS OF N (THE INPUT NUMBER OF THE NEURON).

| Methods | Operations | | | | | |
|---|---|---|---|---|---|---|
| | **MUL** | **ACC** | **Mux** | **Bitwise gate** | | **Bit count** |
| **FP** | N | N | 0 | 0 | | 0 |
| **BC** [32] | 0 | N | N | 0 | | 0 |
| **TWN** [27] | 0 | < N | < N | 0 | | 0 |
| **XNOR** [19] | 0 | 0 | 0 | N | | 1 |
| **GXNOR**[37] | 0 | 0 | 0 | 5N < Op <7N | | 1 |
| **Ours** | 0 | 0 | 0 | **train** | 3N<Op<4N | 1 |
| | | | | **test** | <3N | |

## B. Activation Function

TOT-Net is equipped with novel combined piece-wise activation functions. {Elish, Sigmoid} and {Sigmoid, Leaky_Relu} are found the superior activation functions for CIFAR-10 and CIFAR-100, respectively. The applicability of each new activation function has been empirically proved due to existing in dominant individuals for continues iterations. Figure 5 compares the training accuracy and training loss of the optimized and default activation functions, Relu, on CIFAR-10 and CIFAR-100 datasets. Optimizing activation functions reduces learning instability and provides a considerable loss function.

As shown in Figure 6, TOT-Net with optimized activation functions provides 0.4% and 0.63% higher accuracy compared to the default network with the Relu activation function. Therefore, the final achieved accuracy of TOT-Net for CIFAR-10 and CIFAR-100 datasets are 87.92% and 62.24%, respectively.

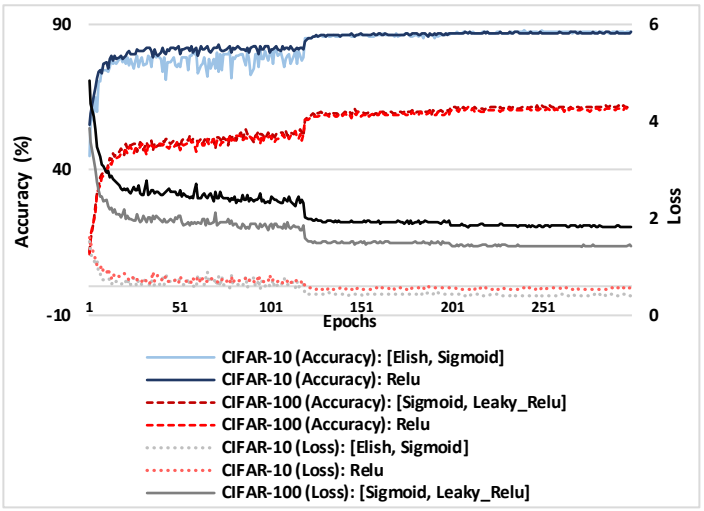



Fig. 6. Comparing the training accuracy and loss of TOT-Net with optimized activation function and TOT-Net with Relu as the default activation function.

## C. Learning Rate

Selecting proper learning rate is essential for achieving the maximum accuracy of CNN. However, manual learning rate selection could easily become exhaustive for deep neural networks. Generally, the learning rate value is considered 0.01 to train CNNs for accelerating the learning procedure [20]. Although lower learning rate (0.0001) is more preferred for training quantized BNNs [20], it is not applicable for TNNs.

Figure 7 illustrates the impact of different learning rates on the accuracy of TNNs. According to the results, ***learning rate=0.01*** provides higher accuracy level for both CIFAR-10 and CIFAR-100 datasets. To guarantee the results, we have trained the network with twice number of epochs equal to 600 for lower learning rates.

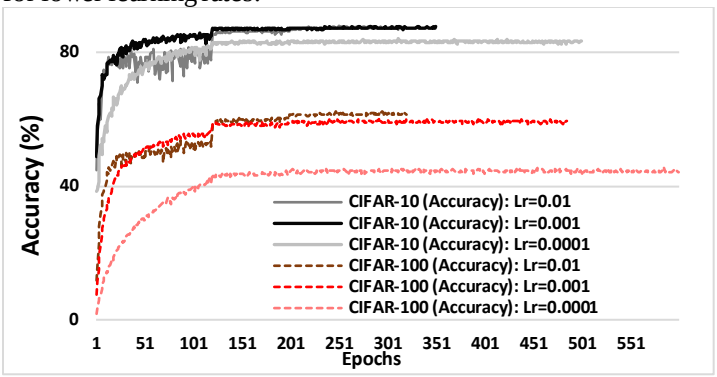



Fig. 7. The effectiveness of different learning rate on the accuracy of TNNs.

## VI. CONCLUSION AND FUTURE WORK

Ternary neural networks provide high compression rate and acceptable accuracy level compared to binarized neural networks. These features make TNNs more suitable for low-power embedded platforms. However, there is still a gap between the accuracy of state-of-the-art TNNs and the accuracy of full precision networks. To further improve the

accuracy, we first proposed ternarization on both weights and activations called TOT-Net which can significantly improve the accuracy with negligible computation overhead compared to binary neural networks. In fact, we propose a simple bitwise logic (XOR and AND gates) instead of computation-intensive traditional multiplications. Next, we optimized the activation function of TOT-Net with an evolutionary optimization strategy. Finally, we have done a practical survey on the initial learning rate for TOT-Net. The evaluation results demonstrate the impact of TOT-Net on large scale datasets.

## REFERENCES


[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *In Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R. and Fei-Fei, L., 2014. Large-scale video classification with convolutional neural networks. *In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 1725-1732.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 580-587, 2014.

[4] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation." *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431-3440. 2015.

[5] A. G. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," *ArXiv Preprint ArXiv:1704.04861*, 2017.

[6] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," *In International Conference on Learning Representations*, pp. 1-13, 2017.

[7] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li et al. "Dadiannao: A machine-learning supercomputer." *In Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 609-622. IEEE Computer Society, 2014.

[8] Y.-H. Chen, J. Emer, and V. Sze, "Eyeriss: a spatial architecture for energy-efficient dataflow for convolutional neural networks," *In ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 367–379, 2016.

[9] M. Gao, J. Pu, X. Yang, M. Horowitz, and C. Kozyrakis. "Tetris: Scalable and efficient neural network acceleration with 3d memory." *In ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 751-764. ACM, 2017.

[10] S. Han, J. Pool, J. Tran, and W. Dally. "Learning both weights and connections for efficient neural network." I*n Advances in neural information processing systems*, pp. 1135-1143. 2015.

[11] S. Han, H. Mao, and W. J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." *arXiv preprint arXiv:1510.00149 (2015)*.

[12] T. J. Yang, Y. H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," In *Proceeding of the 30th IEEE Conference Computer Vision Pattern Recognition, CVPR 2017*, pp. 6071–6079, 2017.

[13] M. Basirat, and P. M. Roth. "The Quest for the Golden Activation Function." *arXiv preprint arXiv:1808.00783* (2018).

[14] M. Loni, A. Majd, A. Loni, M. Daneshtalab, M. Sjödin, and Elena Troubitsyna. "Designing Compact Convolutional Neural Network for Embedded Stereo Vision Systems." *In 2018 IEEE 12th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pp. 244-251. IEEE, 2018.

[15] H. Sharma, J. Park, N. Suda, L. Lai, B. Chau, V. Chandra, and H. Esmaeilzadeh. "Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks." *In Proceedings of the 45th Annual International Symposium on Computer Architecture*, pp. 764-775. IEEE Press, 2018.

[16] A. Zhou, A. Yao, Y. Guo, L. Xu, and Y. Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *ArXiv Preprint ArXiv:1702.03044* , 2017.

[17] D. Zhang, J. Yang, D. Ye, and G. Hua. "Lq-nets: Learned quantization for highly accurate and compact deep neural networks." *In Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 365-382. 2018.

[18] I. Hubara, "Binarized Neural Networks," *In Advances in Neural Information Processing Systems*, pp. 4107-4115. 2016.

[19] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary," *In European Conference on Computer Vision*, pp. 525-542. Springer, Cham, 2016.

[20] Tang, Wei, Gang Hua, and Liang Wang. "How to train a compact binary neural network with high accuracy?." *In Thirty-First AAAI Conference on Artificial Intelligence*. 2017.

[21] J. Cheng, P. Wang, G. Li, Q. Hu, and H. Lu. "Recent advances in efficient computation of deep convolutional neural networks." *Frontiers of Information Technology & Electronic Engineering* 19, no. 1 (2018): 64-77.

[22] K. He , X. Zhang, Sh. Ren, and J. Sun. "Deep residual learning for image recognition." *In Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770-778. 2016.

[23] M. Lin , Q. Chen, and S. Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

[24] F. Conti, P. D. Schiavone, S. Member, and L. Benini, "XNOR Neural Engine : a Hardware Accelerator IP for 21 . 6 fJ / op Binary Neural Network Inference." *ArXiv Preprint ArXiv:1807.03010*, 2018.

[25] S. Hayou, A. Doucet, and J. Rousseau. "On the selection of initialization and activation function for deep neural networks." *arXiv preprint arXiv:1805.08266* (2018).

[26] T. Elsken, J. Hendrik Metzen, and F. Hutter. "Neural architecture search: A survey." *arXiv preprint arXiv:1808.05377* (2018).

[27] B. Liu, "Ternary Weight Networks," *ArXiv Preprint ArXiv:1605.04711*, 2016.

[28] Y. Ma, N. Suda, Y. Cao, J. Seo, and S. Vrudhula, "Scalable and Modularized RTL Compilation of Convolutional Neural Networks onto FPGA." *In Field Programmable Logic and Applications (FPL), 2016 26th International Conference on*, pp. 1-8. IEEE, 2016.

[29] P. Judd, J. Albericio, T. Hetherington, T.M. Aamodt, and A. Moshovos. "Stripes: Bit-serial deep neural network computing." *In 2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 1-12. IEEE, 2016.

[30] A. Yazdanbakhsh, A. T. Elthakeb, P. Pilligundla, F. Mireshghallah, and H. Esmaeilzadeh. "ReLeQ: An Automatic Reinforcement Learning Approach for Deep Quantization of Neural Networks." *arXiv preprint arXiv:1811.01704* (2018).

[31] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi, "Ristretto: A Framework for Empirical Study of Resource-Efficient Inference in Convolutional Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no.11, pp. 1–6, 2018.

[32] M. Courbariaux and J. David, "BinaryConnect : Training Deep Neural Networks with binary weights during propagations," *In Advances in Neural Information Processing Systems*, pp. 3123-3131. 2015.

[33] C. Jin, H. Sun, and S. Kimura, "Sparse ternary connect: Convolutional neural networks using ternarized weights with enhanced sparsity," In *Design Automation Conference (ASP-DAC), 2018 23rd Asia and South Pacific*, pp. 190-195. IEEE, 2018.

[34] D. Wan, F. Shen, L. Liu, F. Zhu, J. Qin, L. Shao, and H. T. Shen. "TBN: Convolutional Neural Network with Ternary Inputs and Binary Weights." *In Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 315-332. 2018.

[35] S. Sato, H. Nakahara, and M. Ikebe, "BRein Memory : A Single-Chip Binary / Ternary Reconfigurable in-Memory Deep Neural Network," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 4,pp. 983-994, 2018.

[36] M. Loni, M. Daneshtalab, and M. Sjödin. "ADONN: Adaptive Design of Optimized Deep Neural Networks for Embedded Systems." *In 21st Euromicro Conference on Digital System Design (DSD)*, pp. 397-404. IEEE, 2018.

[37] L. Deng, P. Jiao, J. Pei, Z. Wu, and G. Li, "GXNOR-Net : Training deep neural networks with ternary weights and activations without full-precision memory under a unified discretization framework," *Neural Networks*, vol.100,no. 4, pp.49-58, 2018.

[38] H. Alemdar, V. Leroy, A. Prost-Boucle, and F. Pétrot. "Ternary neural networks for resource-efficient AI applications." *In 2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2547-2554. IEEE, 2017.