JTE HCM-UTE
ISSN: 2615-9740

JOURNAL OF TECHNICAL EDUCATION SCIENCE
Ho Chi Minh City University of Technology and Engineering
Website: https://jte.edu.vn
Email: jte@hcmute.edu.vn

Optimizing Binary Neural Network for Resource-Constrained Edge Devices in IoT Applications

Van Minh Nguyen[1], Tien Tu Ngo[2], Tien Dung Tran[3], Minh Tam Nguyen[1], Minh Huan Vo[1]*

[1]*Ho Chi Minh City University of Technology and Engineering, Vietnam*
[2]*Kookmin University, Korea*
[3]*Le Hong Phong High School for The Gifted, Vietnam*

*Corresponding author. *Email: huanvm@hcmute.edu.vn*

ARTICLE INFO

Received: 03/01/2025
Revised: 27/02/2025
Accepted: 30/06/2025
Published: 28/02/2026

KEYWORDS

Binary Neural Network;
XNOR-popcout;
Edge device;
IoT;
Resource-constrained hardware.

ABSTRACT

The implementation of artificial intelligence models on edge devices is increasingly popular, bringing many values in reducing latency, effectively utilizing bandwidth, improving data security, enhancing privacy and reducing costs for users. However, this work poses many challenges in terms of accuracy, processing speed, hardware resources and model size for devices constrained by limited hardware. Binary Neural Network (BNN) is proposed as a potential solution to reduce resource requirements by using only 1 bit for quantizing. In this study, BNN network is optimized by binary quantizing both weights and activation functions with XNOR-popcout multiplication to optimize BNN network. The results show that BNN network model is lighter in memory footprint when deployed on hardware with limited computational resources, less computational time than conventional BNN network which helps the model execute faster as the network architecture becomes less complex, with acceptable accuracy on two datasets MNIST and Fashion MNIST. The proposed BNN model resul can be deployed on edge devices for IoT applications.

Doi: https://doi.org/10.54644/jte.2025.1756

1. Introduction

Artificial Intelligence (AI) is a field within computer science. AI systems are based on the idea of human intelligence and are programmed to mimic human learning and problem-solving abilities. Deep Neural Networks (DNNs) are increasingly achieving significant breakthroughs in various areas of life. However, most deep neural networks today use 32 bits to encode each weight and activation, which makes the size of the network model larger, slower, and more power-consuming [1]. This is because neural network models are becoming increasingly complex and require increasingly large amounts of computational resources in the era of rapid technological development [1], [2]. This poses a great challenge for computer systems, especially in applications that require fast and efficient computation. Current deep learning neural networks often use 32 bits, 16 bits or 8 bits to encode each weight and activation function [2], [3], making the network model larger, slower and more energy consuming. This makes developing applications for resource-limited hardware more complicated, requiring more optimization techniques [4], [5].

One promising network model that is of interest is the Ternary Neural Network (TNN). However, TNNs face several problems during training and deployment. First, training TNNs can be difficult because optimization in the discrete weight space can be more complex than in real-time networks. Second, using only three states (-1, 0, 1) to represent weights and activations can lead to information loss, reducing the accuracy of the network [6]. Finally, although TNNs are more memory-efficient than real-time networks, they still require a moderate number of computational resources, especially when deployed on devices with limited computational resources such as embedded devices.

Binary Neural Networks (BNNs) have been proposed as a potential solution to reduce resource requirements, by quantizing weights to binary values and replacing complex real-world operations with

simple binary operations [7], [8]. BNNs encode weights and activations with just 1 bit, which significantly reduces the required memory space and improves processing speed [9], [10]. This is especially useful in applications that require high performance but are limited by hardware resources, such as mobile devices, embedded devices, and IoT systems.

However, optimizing and applying BNNs effectively is still a problem that needs further research and development. The tools supporting BNN development are still in the process of being perfected, including specialized optimizers, training metrics, and performance analysis tools. This development not only opens up new opportunities for researchers in the field of efficient deep learning, but also enables experts to exploit the potential of BNNs in practical applications.

To address these limitations, the research proposes a solution to build and train neural networks with quantized weights and activation functions to 1 bit. This is not only a solution for a lighter neural network when deployed on hardware with limited computational resources but also helps the model execute faster as the network architecture becomes less complex. The implementation of BNN models not only reduces memory footprint but also improves processing speed and saves energy, opening up new opportunities for applications in resource-constrained environments. With this potential, BNNs promise to contribute to the development of artificial intelligence in diverse and rich fields.

2. Methodology

2.1. Artificial Neural network (ANN)

ANNs are composed of basic computational units, called neurons, arranged and interconnected in a multilayer network structure. Each neuron in the network receives input signals from other neurons and then performs processing through a characteristic activation function. The results of this process are then passed on to neurons in the next layers of the network. This network structure allows ANNs to learn and adapt from data by adjusting the connection weights between neurons [11], [12]. Neurons receive inputs, x_i, from other neurons or from input data, each input is multiplied by a corresponding weight. These weights, w_i are parameters learned during the training of the network, reflecting the degree of connection between neurons. In addition, each neuron has a bias value, which is a fixed parameter. Mathematically, the output of the above neuron is represented as follows:

$$o = f(\sum_{i=1}^{n} w_i x_i + \text{bias}) \quad (1)$$

Training a neural network is a complex and important process, in which the weights w_i and bias are adjusted during the learning process. The most common method for adjusting these parameters is the backpropagation algorithm. This algorithm works by calculating the gradient of the loss function with respect to the weights w_i and bias, and then adjusting them in a way that reduces the error between the predicted value and the actual value. Thanks to its ability to learn and adapt from data, ANNs have become a powerful tool and are widely used in many different fields. In image processing, ANNs are used to detect and classify objects, improve image quality, and perform tasks such as face recognition. In speech recognition, ANNs help convert audio to text, distinguish between different voices, and even perform automatic translation. In data prediction, ANNs are applied to predict market trends, diagnose diseases, and forecast weather [13], [14].

2.2. Quantizing weights and activation functions

Quantizing weights and activation functions is the process of reducing the amount of memory stored for each weight (float32) to smaller data types such as 16-bit, 8-bit, 4-bit, 2-bit and 1-bit, in order to reduce the memory footprint and computational complexity of the neural network. In a neural network, weights and activations are the main parameters stored in memory. Typically, these values are represented as 32-bit floating-point numbers (float32) to ensure high precision, but this results in a large memory footprint. For example, the ResNet-50 architecture with approximately 26 million weights would require 168 MB of memory if using 32-bit values [15].

Quantization reduces the precision of these parameters by converting from 32-bit values to lower-precision values, such as 8-bit. This process not only reduces the model size but also improves network

latency and energy consumption. Integer operations require less computational resources than floating-point operations, resulting in improved processing speed and energy savings.

There are two main quantization methods: post-training quantization and training-while-training quantization. Post-training quantization is a technique in which the neural network is trained entirely with real number computations and then quantized. Meanwhile, training-while-training quantization performs quantization during the training process, which helps to reduce quantization errors and maintain higher accuracy [7], [8].

2.3. Sign function

Basically, to implement binary connections in artificial neural networks, while training the neural network, the weights and/or activation functions are quantized to 1-bit states (0 and 1 or -1 and 1). In most cases, they are usually quantized to -1 and 1. The sign function commonly used for quantization is as follows:

$$x^b = sign(x) = \begin{cases} +1, & x \geq 0 \\ -1, & Others \end{cases} \quad (2)$$

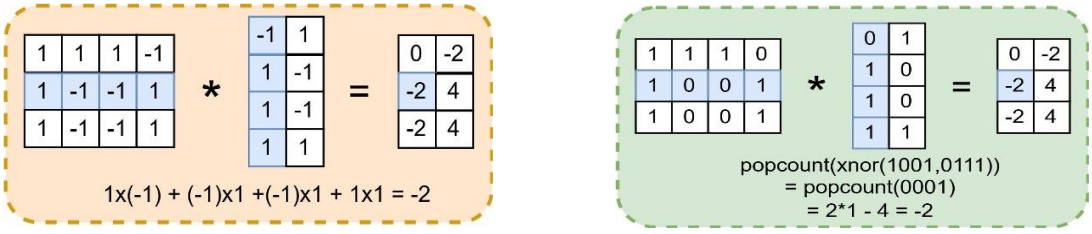
This approach is to quantize the weights and activation functions to a predetermined threshold (usually 0). Because the backpropagation process through the sign function requires calculating the gradient for the sign function. However, most of the derivatives of the sign function are usually zero or undefined, so it is impossible to backpropagate through the derivative of the sign because the values of the weights and biases of the model will not be updated, leading to the model not being learned. To solve that problem, the method used is STE (Straight-Through Estimator) [16]. This method is used to approximate the derivative of a function. It can be seen that when applying the STE method, the derivative of the sign has a value of 1 in the range from -1 to 1. Then, when updating the weights, the gradient of the sign will update the weights and biases in the range from -1 to 1 [16].

2.4. Optimizing Operations with XNOR-Popcount

Table 1. *XNOR-Popcount operation*

Inputs	Weights	Outputs
0 (-1)	0 (-1)	1 (+1)
0 (-1)	1 (+1)	0 (-1)
1 (+1)	0 (-1)	0 (-1)
1 (+1)	1 (+1)	1 (+1)

In order to replace and optimize matrix operations such as multiplication and addition in traditional 32-bit neural networks, the proposed XNOR-popcount will optimize the computation time by up to 58 times [17]. The operation of XNOR is shown in Table 1. XNOR-Popcount has great potential to replace traditional matrix multiplication, saving memory and reducing execution time. Figure 1 presents a matrix multiplication replaced by XNOR-Popcount. As seen in the Figure 1, XNOR-Popcount is implemented through an XNOR operation for 4 bits, then the result is calculated using the Popcount operation.



(a) (b)

Figure 1. *A matrix multiplication replaced by XNOR-Popcount.*
(a) normal multiplication (b) XNOR popcount multiplication.

3. Binary neural network (BNN)

Binary neural networks are a special form of neural networks that use binary values (-1 and 1) for both weights and/or activations. This significantly reduces computation and memory requirements compared to networks that use floating-point values, making BNNs suitable for resource-constrained devices and applications and real-time applications [18], [19].

The architecture of BNNs is similar to that of traditional neural networks, with the main difference being in the representation of weights and activations. Weights in BNNs are binarized, which reduces storage requirements and increases the efficiency of matrix multiplications. For example, a floating-point weight matrix might be represented as {-0.5, 0.8, -1.2}, while a binary weight matrix would be {-1, 1, -1}. The activations in BNNs are also binarized, which simplifies the calculations in the network. Instead of floating point values, binary activations will only have the value 1. Binarization typically uses sign functions for the weights and for the activation functions. In backpropagation, the derivatives are calculated relative to the actual values of the weights, while in forward propagation binary values are used.

To train binary neural networks, special techniques such as STE and Batch Normalization have been developed to address the challenges of binary quantization [16]. STE approximates the derivative during backpropagation, allowing binary weights to be trained by bypassing the problem of the unknown derivative of the sign function. Batch normalization improves the stability and performance of BNNs by normalizing the input to each layer, minimizing the impact of quantization, and accelerating convergence.

To train a BNN model, we cannot use conventional methods because the model loss will be very high, leading to difficulty in convergence and greatly reducing the accuracy. The method commonly used to train BNN is backpropagation through STE. This method is presented through the Algorithm 1 described below:

Algorithm 1: Updating weights for Binary Neural Network

1: **Input:** Training dataset **D(x, y)**, Learning rate η, Number of iteration *epochs*, Number of layer ***L*,** Number of sample ***S***

2: Initiating weights W randomly

3: **for** i=1 to *epochs* **do:**

4: **for** (x, y) ∈ **D do:**

5: **for** i=1 to ***L*** **do:**

6: $x_{b_i} = sign\,(x_i),\ W_{b_i} = sign\,(W_i)$

7: $z_i = x_{b_i} \cdot W_{b_i}$

8: $a_i = f\,(z_i)$

9: **end for**

10: *#Update weights*

11: **for** j = ***L*** to 1 **do**:

12: **if** j = ***L*** **do**:

13: loss = $\frac{1}{N}\sum_{k=1}^{N}\sum_{q=1}^{S}(a_{j_{kq}} - y_{j_{kq}})^2$

14: $\frac{\partial loss}{\partial W_j} = \frac{\partial \text{loss}}{\partial a_j}\frac{\partial a_j}{\partial z_j}\frac{\partial z_j}{\partial W_j} = \frac{2}{N}\sum_{k=1}^{N}\left(a_{j_k} - y_{j_k}\right) * a_{j-1}$

15: $Let\ \delta_j = \frac{\partial \text{loss}}{\partial z_j}$

16: $\Delta W_{b_j} = \eta\frac{\partial \text{loss}}{\partial W_j}$

16: $W_{b_j} = sign\,(W_{b_j} - \Delta W_{b_j})$

17: **else do**:

18: $\delta_j = \sum_{K=1}^{K} \delta_{j+1_K} W_{K,j}$
19: $\frac{\partial loss}{\partial W_j} = \delta_j * a_{j-1}$
20: $\Delta W_{b_j} = \eta \frac{\partial \text{loss}}{\partial W_j}$
21: $W_{b_j} = sign\,(W_{b_j} - \Delta W_{b_j})$
22: **end if**
23: **end for**
24: **end for**
25: **end for**

The Algorithm 1 shows that quantizes all weights and inputs during training. The output is also quantized to 2 values +1 and -1. The algorithm 1 is designed to train a BNN using a STE for handling non-differentiable operations, specifically the sign function. Initially, the weights of the network are randomly initialized. The training proceeds over multiple epochs, with each epoch iterating over the dataset. In each iteration, both the inputs and the weights are binarized using the sign function, converting continuous values into binary ±1 values. A forward pass is then performed to compute the network's output. The mean squared error (MSE**)** loss is calculated by comparing the predicted output with the true labels. During the backward pass, the gradient of the loss with respect to the output is computed, and backpropagation is applied to update the weights. However, since the sign function is non-differentiable, the STE approximates the gradient by treating the sign function as an identity function for the purpose of backpropagation, allowing the gradients to propagate through the binary weights. Finally, the updated weights are re-binarized using the sign function to maintain the binary constraint, and the process continues until the network is trained over the specified number of epochs. This approach enables the efficient training of binary neural networks, suitable for resource-constrained environments.

The Algorithm 2 outlines a process for efficiently implementing BNN using compressed binary representations for both input data and model weights. First, the test dataset, with inputs and weights quantized to ±1, is compressed by packing these binary values into 32-bit unsigned integer vectors, using the sign function and bit-shifting to map each input and weight to a distinct bit position. The binary weight vectors are then similarly compressed for efficient storage and computation. During inference, the model performs a bitwise XNOR operation between the compressed input and weight vectors to compute their similarity, followed by a popcount operation to count the number of matching bits. The final output is derived from the popcount, scaled and adjusted to produce the model's prediction. This approach significantly reduces memory usage and computation time, making it ideal for resource-constrained environments, particularly in hardware implementations of binary neural networks.

When executing the model using XNOR-Popcount presented in Algorithm 2 with all binary values of 0 and 1, the model executes quickly, but in return the accuracy is not as high as desired. When executing the network, the bit compression technique will be applied as shown in the Algorithm 2:

Algorithm 2: Running BNN based on XNOR Popcount

1:*Require***:** unsigned int 32 vectors **V**, test dataset **D(x, y)** has been quantized to +1 or -1, trained weights $\mathbf{W_b}$, number of layers **L**.
2: *Data compression***:**
3: **for** i=1 to *len(x)/32* **do:**
4: **for** j=1 to 32 **do:**
5: $V_{i_x} = V_{i_x} \mid sgn(x_j) \ll j$
6: **end for**
7: **end for**

8: Weight compression:
9: **for** i=1 to *len(**W**$_b$)/32* **do:**
10: **for** j=1 to 32 **do:**
11: $V_{i_W} = V_{i_W} \ |sgn(W_{b_j}) \ll j$
12: **end for**
13: **end for**
14: Implementing model:
15: **for** i=1 to *len(y)* **do:**
16: **for** j=1 to **L do:**
16: $z_j = XNOR(V_x, V_w)$
17: $o_j = 2 * \text{Popcount}(z_j) - len(z_j)$
18: **end for**
19: **end for**

Where the sgn (x) function is defined by:

$$sgn(x) = f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & Others \end{cases} \quad (3)$$

Algorithm 2 above defines a simple bit compression function, in which the binary values +1 and -1 are converted to two bits 0 and 1 and compressed into unit32 vectors. Then, they are calculated and executed via the XNOR-Popcount operation. This not only reduces the network size by approximately 32 times, but also makes the network execution faster by replacing multiplication and addition with XNOR-Popcount.

4. Experimental results

The network model is trained with the following hardware: CPU: Intel® Core™ i5 1035-G1CPU @ 1.00GHz. Memory: 12GB of RAM. GPU: Intel® UHD Graphics. Back to the network model, it is trained with 100 epochs, learning rate is 0.1 and batchsize is 200. The network model uses 1 hidden layer with 512 neurons to train and evaluate the feed-forward binary network, 1 input layer with 784 neurons, 1 output layer with 10 neurons.

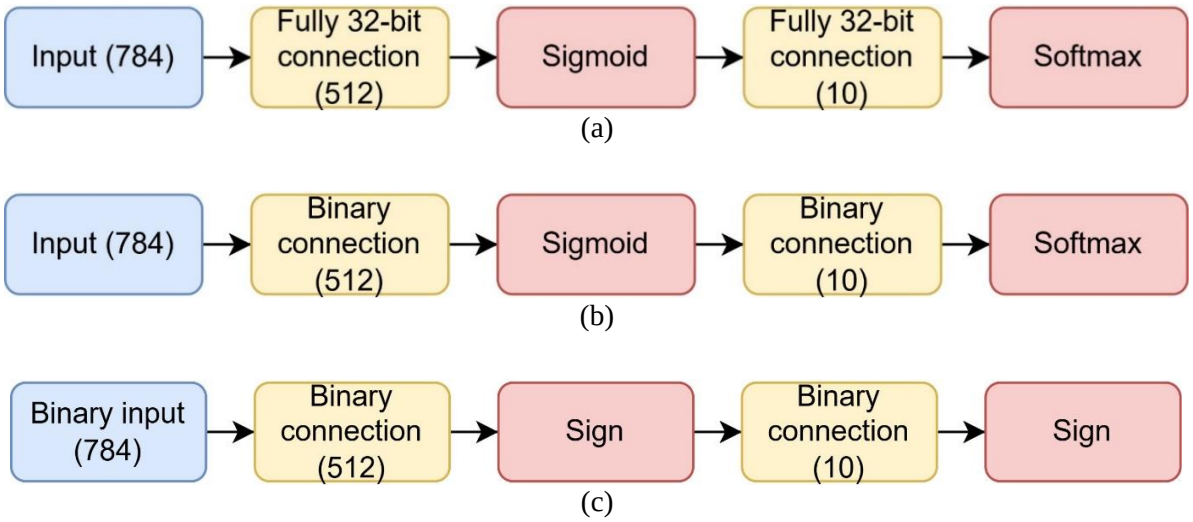


Figure 2. *(a) 32-bit model Architecture of a 1-hidden-layer network with 512 neurons. (b) BNN based on Binary Connect which only weights are quantized to 1 bit. (c)BNN based on XNOR Popout which both weights and activations are quantized to 1 bit.*

Figure 2(a) shows the 32-bit mode architecture using fully connected layer, sigmoid and softmax activation which the values are presented in 32 bit floating point. The sigmoid and softmax activation are mentioned in above section 2.1 and 2.2. Figure 2(b) shows a binary neural network that only has weights quantized to two values +1 and -1. Thus, the network size is reduced by 32 times, but the execution time is only reduced by a fraction or remains unchanged because it still uses addition and multiplication operations like a traditional neural network.

Case study 1: Applying BNN to MNIST dataset

The MNIST dataset is one of the most popular datasets in the field of machine learning and computer vision. It consists of 60,000 labeled training images of size 28x28, containing handwritten characters from 0 to 9. The test data consists of 10,000 labeled images of size 28x28. To process the input data and train this dataset, the study needs to reshape the image to size 60,000x784, with 784 being the result of multiplying 28x28. Therefore, the number of inputs of this dataset will be 784 corresponding to 784 pixels. The output consists of 10 layers from 0 to 9. Therefore, the number of neurons in the output layer is also 10. Here, the threshold for quantizing the pixels is chosen to be 127.5 because the pixels have values from 0-255.

Looking at the Figure 3, the model accuracy of the BNN based on XNOR-popcount is no longer as smooth as that with 32-bit weights and 32-bit activation function. This happens because the weights are immediately quantized after being updated, causing the loss to increase while the accuracy decreases. The network model uses those quantized values to continue training. Therefore, it is difficult for the network model with binary weights to train smoothly like the 32-bit model did.

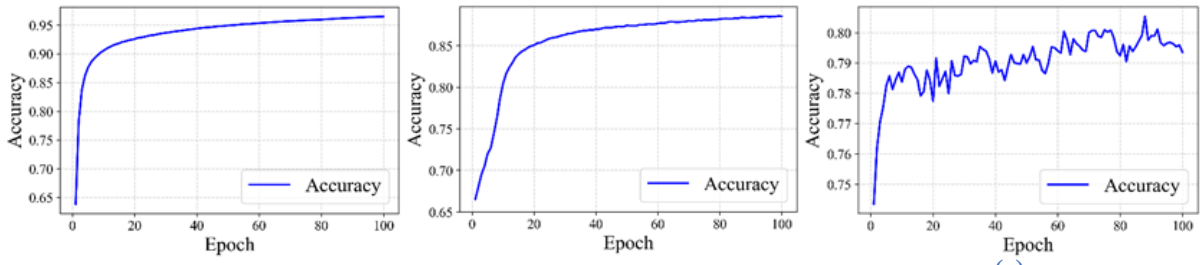


Figure 3. *Training accuracy in MNIST dataset of (a) fully 32-bit connection (b) BNN based on binary connection (c) BNN based on XNOR-popcout with threshold=127.5*

Table 2. *The accuracy and execution time of the neural models.*

Network model	# bits of weight	#bits of activation function	Accuracy (%)	Memory footprint (Kb)	Execution time (ms/image)
BNN based on XNOR-Popcount (Threshold 127.5)	1	1	80.82	51	0.0196
BNN based on Binary connection	1	32	88.13	51	0.786
Fully 32-bit connections	32	32	96.55	1587	0.784

Table 2 compares the performance of three different network models, focusing on their weight and activation bit sizes, accuracy, memory footprint, and execution time per image. The BNN based on XNOR-Popcount (Threshold 127.5) model uses both 1-bit weights and 1-bit activations, achieving an accuracy of 80.82% with a memory footprint of 51 KB and a fast execution time of 0.0196 ms per image. This model is highly efficient in terms of memory and speed, but the accuracy is lower compared to the other models. The BNN based on Binary connection model, which uses 1-bit weights and 32-bit activations, has a higher accuracy of 88.13%, while maintaining a memory footprint of 51 KB. However, its execution time per image is significantly higher, at 0.786 ms, due to the larger bit-width for

activations. The Fully 32-bit connections model, with both 32-bit weights and activations, achieves the highest accuracy of 96.55%, but it requires a much larger memory footprint of 1587 KB and a longer execution time of 0.784 ms per image. This demonstrates a trade-off between accuracy, memory usage, and execution speed, with the XNOR-Popcount model being the most efficient but less accurate, and the fully 32-bit model offering the best accuracy at the cost of higher memory usage and slower execution.

Case study 2: Applying BNN to MNIST Fashion dataset

Introduced in 2017, Fashion MNIST has quickly become one of the most popular and important datasets in the field of machine learning and computer vision. Replacing the classic MNIST dataset, Fashion MNIST consists of 70,000 grayscale images of 28x28 pixels, divided into 10 object classes to simulate fashion products, including jackets, shoes, bags, and uniforms. The diversity and complexity of this dataset requires advanced recognition and classification methods to achieve accurate and reliable results.

Similar to the MNIST dataset, Fashion MNIST has many similarities, so it is also possible to reuse the network models designed in Case study 1 for analysis and evaluation. The designed network model includes 1 hidden layer with 512 neurons. Accuracy in Figure 4(a) is trained with 32-bits for weights and activation functions while Figure 4(b) is trained with 32-bits for activation functions and 1 bits for weights. Figure 4(c) and (d) show training accuracy with various thresholds of BNN based on XNOR-popcount operation.

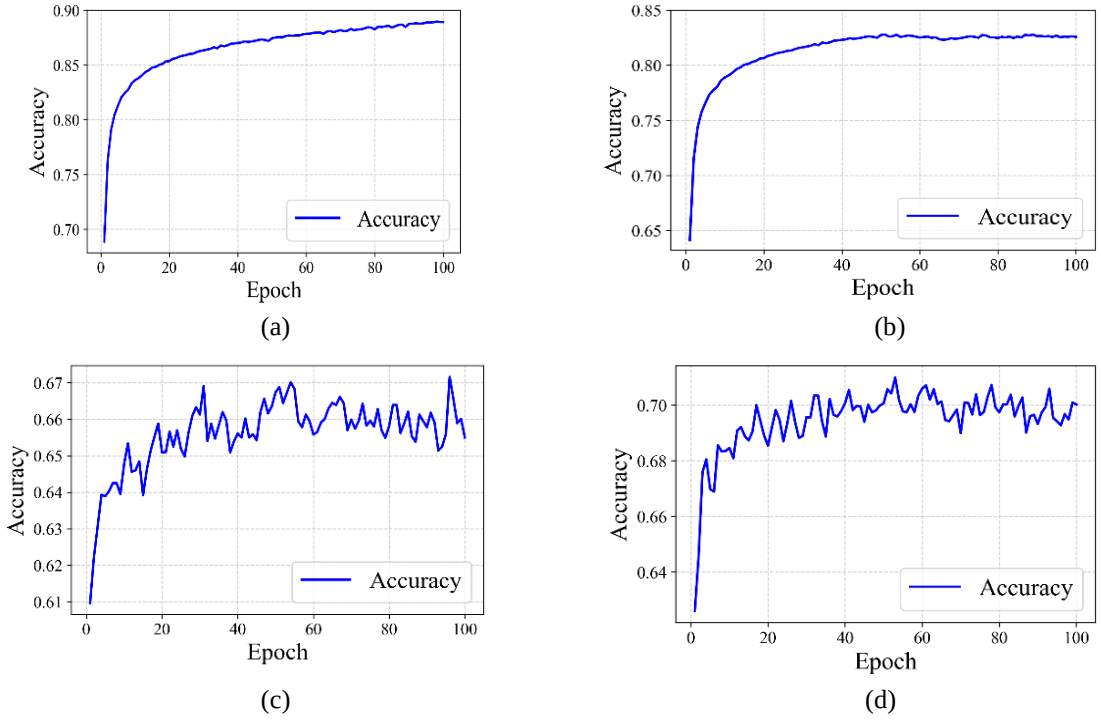


Figure 4. *Training accuracy in Fashion MNIST of (a) fully 32-bit connection (b) BNN based on binary connection (c) BNN based on XNOR-popcout with threshold=127.5 (d) BNN based on XNOR-popcout with threshold=90.*

Table 3 compares four different network models. The BNN based on XNOR-Popcount (Threshold 127.5) and BNN based on XNOR-Popcount (Threshold 90) models both use 1-bit weights and 1-bit activations, with the threshold value influencing their accuracy. The first model achieves an accuracy of 67.16%, while the second, with a threshold of 90, achieves a higher accuracy of 70.99%. Both models have a very low memory footprint of 51 KB and very fast execution times (around 0.0196–0.0197 ms per image). The BNN based on Binary Connection model, which uses 1-bit weights and 32-bit activations, achieves a higher accuracy of 82.74%, but the trade-off is an increased execution time of

0.784 ms per image, while the memory footprint remains the same at 51 KB. The Fully 32-bit connections model, with 32-bit weights and activations, offers the highest accuracy of 86.77% but comes at the cost of a significantly larger memory footprint (1587 KB) and a slightly slower execution time (0.785 ms per image). This comparison highlights the trade-offs between accuracy, memory usage, and execution speed, with the 32-bit model providing the best accuracy but requiring more resources, while the XNOR-Popcount models are faster and more memory-efficient but less accurate.

Table 3. *Comparing neural networks at test dataset*

Network model	# bits of weight	#bits of activation function	Accuracy (%)	Memory footprint (Kb)	Execution time (ms/image)
BNN based on XNOR-Popcount (Threshold 127.5)	1	1	67.16	51	0.0197
BNN based on XNOR-Popcount (Threshold 90)	1	1	70.99	51	0.0196
BNN based on Binary Connection	1	32	82.74	51	0.784
Fully 32-bit connections	32	32	86.77	1587	0.785

Through experiments on the Fashion MNIST dataset in both Figure 4 and Table 3, it can be seen that in addition to calculating gradients and updating weights for the network model, data preprocessing is equally important. It can directly affect the accuracy of the network model and max/2 is not always the ideal fixed threshold for quantizing the input of a binary neural network. Similar to the analysis for the MNIST dataset, it can be seen that the network model using Binary Connection has reduced the storage capacity of the network model by about 32 times, but the execution speed is not faster than the traditional network model. The BNN model using XNOR-Popcount not only reduces about 32 times when comparing the network model capacity, but also reduces about 40 times when comparing the execution time.

5. Conclusion

This research analyzes neural network models on the MNIST and Fashion MNIST datasets. This research conducted a detailed evaluation of the performance and efficiency of each network model, focusing on the binary neural network (BNN) model. The models were compared based on a series of key criteria including the number of bits for weights, the number of bits for activation, accuracy, memory size, and execution time. This process allowed me to collect detailed data and compare the models comprehensively. The analysis results showed that the BNN based on XNOR-popcount model not only achieved significant accuracy but also optimized memory size and execution time. This is especially important in the context of applications that require fast computation and resource savings. In addition, the use of low numbers of bits for weights and activation functions in BNN has proven to be highly effective in minimizing the model size without significantly reducing the accuracy. This result confirms the potential of the BNN model for application in real-world systems where efficiency and fast computation are key factors. Taken together, this study has shown that the BNN network model is a potential and effective choice for machine learning applications.

The next direction of this research includes extending and improving BNN models to apply to more complex data sets and various real-world problems. This research intends to further improve other optimization techniques to enhance the accuracy and performance of BNN. In addition, integrating BNN models into embedded devices and IoT systems is also an important direction, in order to maximize the resource-saving ability of this model. Future research will also focus on developing deeper learning methods and unsupervised learning techniques to improve the automation and efficiency of BNN.

Acknowledgments

This work belongs to the project in 2025 funded by the Ho Chi Minh City University of Technology and Education, Vietnam.

Conflict of Interest

The authors declare that there is no conflict of interest in this paper.

REFERENCES

[1] W. X. Zhao *et al*., "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.
[2] X. Zhang *et al*., "Adaptive precision training: Quantify back propagation in neural networks with fixed-point numbers," *arXiv preprint arXiv:1911.00361*, 2019.
[3] S. N. Truong, "A low-cost artificial neural network model for Raspberry Pi," *Eng. Technol. Appl. Sci. Res.*, vol. 10, 2020.
[4] S. N. Truong, "A ternary neural network with compressed quantized weight matrix for low power embedded systems," *Eng. Technol. Appl. Sci. Res.*, vol. 12, pp. 8311–8315, 2022.
[5] V. Mehlin, S. Schacht, and C. Lanquillon, "Towards energy-efficient deep learning: An overview of energy-efficient approaches along the deep learning lifecycle," *arXiv preprint arXiv:2303.01980*, 2023.
[6] Y. Li, W. Ding, C. Liu, B. Zhang, and G. Guo, "TRQ: Ternary neural networks with residual quantization," *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 10, pp. 8538–8546, May 2021.
[7] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," *Adv. Neural Inf. Process. Syst.*, vol. 29, 2016.
[8] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized neural networks: Training neural networks with low precision weights and activations," *J. Mach. Learn. Res.*, vol. 18, pp. 1–30, 2018.
[9] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet classification using binary convolutional neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2016, pp. 525–542.
[10] Y. Guo, "A survey on methods and theories of quantized neural networks," *arXiv preprint arXiv:1808.04752*, 2018.
[11] X. Yao, "A review of evolutionary artificial neural networks," *Int. J. Intell. Syst.*, vol. 8, pp. 539–567, 1993.
[12] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.
[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, pp. 2278–2324, 1998.
[14] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural Comput.*, vol. 29, pp. 2352–2449, 2017.
[15] S. Agrawal, V. Rewaskar, R. Agrawal, S. Chaudhari, Y. Patil, and N. Agrawal, "Advancements in NSFW content detection: A comprehensive review of ResNet-50 based approaches," *Int. J. Intell. Syst. Appl. Eng.*, vol. 11, pp. 41–45, 2023.
[16] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or −1," *arXiv preprint arXiv:1602.02830*, 2016.
[17] T. Ma, Z. Li, Q. Li, H. Liu, Z. Zhao, and Y. Wang, "FPGA optimized architecture of XNOR-POPCOUNT," in *Proc. Int. Conf. Comput., Commun., Percept. Quantum Technol. (CCPQT)*, Xiamen, China, 2023, pp. 235–239, doi: 10.1109/CCPQT60491.2023.00046.
[18] P. V. Khoa, T. N. Quang, and N. N. Lam, "Optimizing the convolutional neural networks for resource-constraint hardwares," *J. Intell. Syst. Appl. Eng.*, vol. 5, no. 1, pp. 1332–1341, 2022.
[19] C. Yuan and S. S. Agaian, "A comprehensive review of binary neural network," *Artif. Intell. Rev.*, vol. 56, pp. 12949–13013, 2023.

Van Minh Nguyen holds a strong academic and professional background in the fields of automation and manufacturing technology. He earned his Engineer Degree in 2009, majoring in Automation Technology from Ho Chi Minh City University of Technical Education (currently Ho Chi Minh City University of Technology and Engineering). Building on this foundation, he completed his Master's Degree in Mechanical Engineering in 2011 at Southern Taiwan University, Taiwan. His research interests and expertise include Computer Numerical Control (CNC), Electrical Discharge Machining (EDM), and Computer Integrated Manufacturing (CIM), with a focus on the integration of advanced machining technologies and intelligent manufacturing systems.

Email: minhngv@hcmute.edu.vn. ORCID: https://orcid.org/0009-0009-2454-2764.

Tien Tu Ngo received a B.S. degree in Computer Engineering Technology from HCMC University of Technology and Education (currently Ho Chi Minh City University of Technology and Engineering), Vietnam, in 2024. He is pursuing a combined master's and PhD program at Kookmin University, Korea. His research interests include memristor-based circuits and architectures, neuromorphic computing systems, brain-inspired computing, and artificial intelligence. He can be contacted at:

Email: ttn.twenty.oh.two@gmail.com. ORCID: https://orcid.org/0009-0000-0554-585X.

Tien Dung Tran is graduated from Le Hong Phong High School for the Gifted and an active member of the Smart Integrated System Solutions Lab. Passionate about researching smart devices and IoT-based solutions for smart home applications.

Email: dung.s011249@ilamail.edu.vn. ORCID: https://orcid.org/0009-0003-7432-1480.

Minh Tam Nguyen received his Bachelor of Electrical Engineering and Power Supply from Ho Chi Minh City University of Technology and Education (currently Ho Chi Minh City University of Technology and Engineering) in 1995, his Master's degree in Electrical Engineering from Ho Chi Minh City University of Technology, Vietnam National University in 2003, and his PhD in Engineering from Sydney University of Technology, Australia in 2010. Dr. Nguyen Minh Tam has been teaching at the Faculty of Electrical and Electronics Engineering, Ho Chi Minh City University of Technology and Education (currently Ho Chi Minh City University of Technology and Engineering) since 1995. His main research direction is the application of soft computing techniques in modeling and control.

Email: tamnm@hcmute.edu.vn. ORCID: https://orcid.org/0009-0000-8230-1373.

Minh Huan Vo received the B.S. and M.S.E.E. degrees in Electronics and Communication Engineering from the Ho Chi Minh City University of Technology, Vietnam in 2005 and 2007. and Ph.D. degree in Electronics Engineering from Kookmin University, Seoul, Korea in 2013. He is currently working as an associate professor at the Faculty of Electrical and Electronics Engineering, Ho Chi Minh University of Technology and Education (currently Ho Chi Minh City University of Technology and Engineering), Vietnam. His current research interests include chip design, optimization algorithm, AI, and data analytics.

Email: huanvm@hcmute.edu.vn. ORCID: https://orcid.org/0000-0002-9990-9331.