Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

Accelerating Neural Networks on FPGA via Tan-Sigmoid Approximation

Manal T. Ali[1*], Bassam H. Abd [2]

[1]University of Diyala, Department of Electronic Engineering, Diyala, Iraq.E-mail: manal_t@uodiyala.edu.iq
[2] University of Technology, College of Electrical Engineering, Baghdad, Iraq.E-mail: 30022@uotechnology.edu.iq
[*] Corresponding author: manal_t@uodiyala.edu.iq

https://doi.org/10.46649/fjiece.v4.2.5a.21.9.2025

Abstract. *This research examines the execution of feed-forward artificial neural networks on field-programmable gate arrays (FPGAs), addressing the challenge of limited hardware resources that restrict network size and hinder the direct execution of nonlinear activation functions. The proposed approach reduces resource usage through data normalization and restructuring, while approximating the tan-sigmoid function via a logarithmic method. Implemented on the Xilinx Spartan-3A (3sd3400afg676-4) platform, the design achieves a 3% optimization in slices and a 2% optimization in lookup tables (LUTs), demonstrating improved efficiency, enabling larger and more cost-effective neural networks on a single FPGA chip.*

Keywords: *Neural networks (NNs), field-programmable gate arrays (FPGAs), data normalization, tan-sigmoid implementation, and restructuring approach.*

1. INTRODUCTION

A major emphasis has been placed on biological principles such as neural networks, evolution, and learning as a result of the increased interest in automated design and adaptive, fault-tolerant systems that are sophisticated and intelligent. These biological notions have been investigated by engineers and computer scientists in attempt to reproduce the properties that are wanted in computing systems[1].

Numerous kinds of classification, perception, association, and control usages are being used in artificial neural networks [2].

Advancements in high-speed computing have facilitated the mapping, modeling, and classification of nonlinear systems through Artificial Neural Network (ANN) simulations, contributing significantly to the development of intelligent systems. Real-time neural computation requires efficient, fast, and cost-effective implementations. To meet these requirements, ANNs have been deployed in practical applications, highlighting their potential for real-world intelligent system design[3], [4].

Focusing on the application rather than the internal mechanics of neural networks allows developers to leverage their capabilities more effectively. In addition, hardware-based neural networks provide enhanced performance in real-time scenarios compared to software implementations, making them particularly suitable for time-critical tasks[5],[6]. In recent years, academics have been attempting to develop a highly efficient neural network using FPGA. Mellit et al. [7] describe a FPGA-based implementation of an intelligent predictor for worldwide solar irradiation. The scientists created a multilayer perceptron (MLP) neural network trained with the backpropagation method to reliably estimate solar radiation based on inputs, including ambient temperature, relative humidity, and sunshine duration. The network was implemented and simulated on a Xilinx Virtex-II FPGA using VHDL, enabling a detailed assessment of timing performance and hardware resource utilization. The model was trained on 1,460 patterns and tested on 365 patterns. The results demonstrate high prediction accuracy and highlights the potential of FPGA-based neural networks for real-time applications in renewable energy systems,

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

while efficiently utilizing hardware resources. Wess et al[8]introduces an FPGA-based implementation of a neural network for ECG anomaly detection. The authors used Principal Component Analysis (PCA) for featuring reduction in conjunction with a multilayer perceptron (MLP) that had six neurons in the hidden layer and twelve inputs. To optimize hardware performance, fixed-point arithmetic and piecewise linear approximations of activation functions were utilized, which resulted in efficient use of FPGA resources, including logic elements, LUTs, and slices. The implementation demonstrated high accuracy, achieving an average of 99.82% on the MIT-BIH database, while maintaining low hardware utilization suitable for real-time applications in medical diagnostics. Wu et al[9] propose a compute-efficient FPGA-based neural network accelerator targeting GoogLeNet v1 inference. Implemented on a Xilinx VU9P (VCU1525) board, the design achieves 3,046 images/sec with 3.3 ms latency per image. It uses 56% of DSP48 tiles, 95% DSP utilization, and on-chip UltraRAM/BRAM for tensor storage, demonstrating high throughput while minimizing resource usage for real-time applications. In [10], researchers provide a hardware implementation of radial basis function (RBF) neural networks with Gaussian activation functions on FPGA. The work employs a piecewise linear approximation of the Gaussian function to reduce hardware resource utilization while maintaining acceptable accuracy. The design, which was implemented on a Xilinx Spartan-3 (xc3s5000-4fg1156) with 16-bit fixed-point arithmetic, achieved an approximation error of ±0.005 and a latency of 29.3 ns for the Gaussian unit. A complete 4-1 RBF network consumed 1193 LUTs, 14 multipliers (MULT18x18), and one BRAM block, with a total delay of about 101.6 ns. The study highlights the effectiveness of mathematical approximation techniques for efficient FPGA-based neural network accelerators targeting real-time applications with strict computational constraints.

This study aims to optimize feed-forward neural networks on FPGA by standardizing data, employing simplified approximation gates, and implementing an efficient tan-sigmoid function to achieve high performance with minimal hardware resources. The following is the structure of the paper: The theoretical underpinnings of artificial neural network covered in Sections 2 and 3, Section 4 introduce the suggested approach, Section 5 analyzes the findings and contrasts them with earlier research, and the work is concluded in the last section.

2. ARTIFICIAL NEURAL NETWORK

2.1. Model of the Neuron

The basic component of neural networks, artificial neurons, replicates the four crucial functions of genuine neurons: dendrites, soma, axons, and synapses[11]. A neuron is a fundamental unit responsible for processing information and plays a crucial role in the functioning of neural networks. Figure1 defines three essential components of the neuron model.

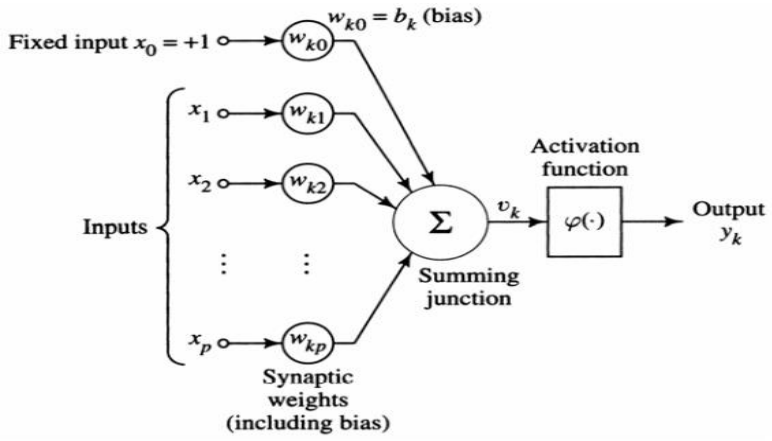

Fig. 1. A neuron's nonlinear model [12].

Here is a method for extracting the output of a neuron:

1. One of the differentiating characteristics would be the weight or strength of a large number of synapses. There is a factor known as the synoptic weight wkj that is responsible for scaling the input signal Xj at the

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

synapse j that is coupled to neuron k. It is essential to keep in mind the manner in which the subscriptions representing the wkj synaptic weight are written. Specifically, the neuron is the subject of the first subscript. The second subscript denotes the synapse's input end, which is attached to the object's weight. The weight wkj is positive if the synapse that connects to the brain networks is activating. The weight is negative, however, if the synapse is inhibitory.

2. A summation mechanism that combines the input signals, taking into account the synaptic weights of each individual neuron.

3. The amount of output that a neuron may produce is limited by activation functions. Because it limits the output signal's amplitude spectrum to a particular finite value, the triggering function is occasionally referred to as a squash function in the literature. The closed unit interval [0, 1] or [-1, 1] [12] is commonly used to describe the normalized amplitude range of a neuron's output.

The weighted adder, the main component of a neuron, uses equation 1 to calculate the neuron's net input:

$$a = \sum_{i=1}^{N} w_i x_i \qquad (1)$$

The weight and input are multiplied by one block [13].

The net inputs are the most essential component of the design of neural networks. They are in the process of generating an output known as the "unit's activation." This is made possible by the use of a scalar-to-scalar function, also known as the activation function, threshold function, or transfer function. [5][6]. Activation functions are very useful in artificial neural networks for converting input signals into output signals that are then delivered to the network's next layer. This process is repeated until the desired destination is reached. The output of a given layer in an artificial neural network can be obtained by first determining the input quantity of a product and the weights that correspond to it then employing an activation function to obtain the output of that particular layer. In the subsequent layer, this result is then used as the input for the layer that comes after it. Both the number of layers that are utilized and, most crucially, the particular activation function that is utilized are factors that determine the accuracy of the predictions that are generated by a neural network[14]. The actualization of artificial neural networks presents a number of obstacles, one of which is the utilization of non-linear functions in the neural network models [15]. Implementing neural networks in hardware is a difficult endeavor that involves more than just carrying out sigmoid functions without any additional steps. Because of the endless exponential series and division operations that are needed, the implementation of non-linear Sigmoid functions by direct methods is prohibitively expensive[16],[17]. Effective implementation of the sigmoid feature in the FPGA is a difficult task that designers are faced with. Several ideas were presented as potential answers to these problems in order to make implementation easier[18][19]. This includes the approximation log sigmoid approach, which is one of them.

2.2. The Approximation Log Sigmoid Method

Due to the fact that it is so effective at carrying out its duty, the sigmoid activation function is extremely popular. This activation function is used for making decisions or predicting results because it is basically a probabilistic approach to decision-making and reaches between 0 and 1. Thus, the prediction would be more accurate because the range is the smallest. Another feature of the sigmoid function is that it is not symmetric to zero, meaning that all neuron output values have the same signs. It is possible to make this issue better by scaling the sigmoid function[14]. An infinite sequence of exponentials is contained within the log-sigmoid function, which means that it cannot be applied directly to hardware. Piece-wise linear approximation, also known as PWLA, is a practical approach that may be employed with simple FPGA designs. In most cases, computationally simplified sigmoid functional alternatives and approximate sigmoid functions are used. This defines a set of shifts and operations that can be used to perform sigmoid functions. This defines the combination of Y=ax+b lines used to approximate the log

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

sigmoid function [18]. By dividing the sigmoid function into five linear components that are referred to as segments, the PWL technique provides an approximation of the function. Table1 is a representation of the segments of the sigmoid function. By increasing the number of segments, it is possible to obtain the approximation precision in an economical manner; however, the implementation of the hardware will become slightly more involved [20]. A comparison of the performance of the approximation function with that of the Log-Sigmoid function is presented in Figure 2.

Table 1. Approximation concerning log-sigmoid

Condition	*Operation*
$x \leq -8$	$f(x) = 0$
$-8 < x \leq -1.6$	$f(x) = (\frac{8-\|X\|}{64}) - 1$
$\|x\| < 1.6$	$f(x) = \left(\frac{X}{4} + 0.5\right)$
$1.6 \leq x < 8$	$f(x)$=1-$(\frac{8-\|X\|}{64})$
$x > 8$	$f(x) = 1$

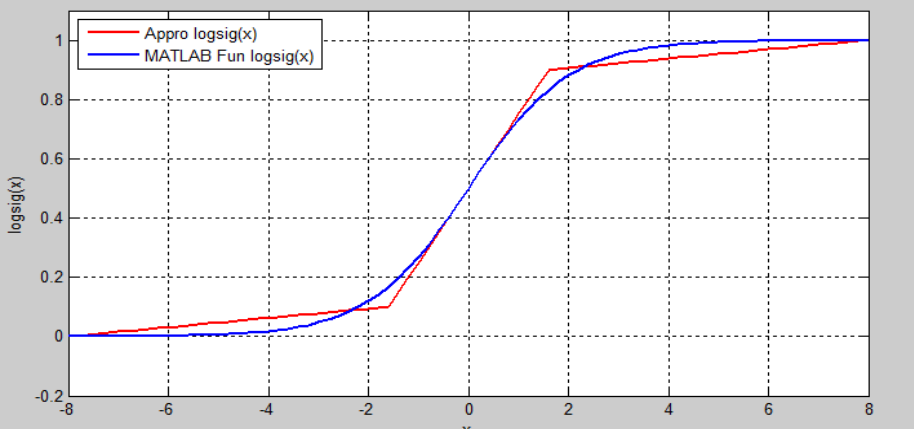

Fig 2: Comparison of output between the approximation and the log sigmoid function

3. THE IDEA FOR THE NEURAL NETWORK DESIGN THAT IS SUGGESTED

The proposed architecture is constructed using the Xilinx System Generator (XSG), which is a combination of MATLAB Simulink and ISE 14.7 programmer. This study demonstrates the practical application of a very efficient tan-sigmoid function, which is based on the approximation log approach. Following that, a neural network is constructed using the proposed tan-sigmoid function. The

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

aforementioned neural network will then be formed using data standardization and restructuring techniques.

3.1 The proposed tan-sigmoid architecture utilizes an approximation method based on the log-sigmoid function.

According to the theory, this strategy was successful in solving the problem of the non-linear activation function. The FPGA design technique has taken into consideration the practical tan-sigmoid function in order to facilitate the implementation of the artificial neural network (ANN) model within the system. Due to the non-linear behavior that is brought about by the exponential component, the direct implementation of this function is not only challenging but also costly. Furthermore, it is difficult to synthesize FPGA library blocks that are suitable for every type of parallel neural network, and these blocks do not include exponential functions or other functions. Additionally, this activation function requires a division operation, which VHDL can accomplish, but it is neither fastness nor space efficient enough to be appropriately included into this design.

The Tan-Sigmoid and Log sigmoid activation functions are widely used in the field of artificial neural networks. Equations (2) and (3) can express the following: [18], [21], and [22].

$$\log - \text{sigmoid} = \frac{1}{1+e^{-x}} \quad (2)$$

$$\tan - \text{sigmoid} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (3)$$

The variety of values for the log-Sigmoid is distinct from that of the tan-sigmoid, which is an S-shaped curve that can take values between -1 and 1. equation (4), which is the tan-sigmoid possible alternative formulation that finds a link between log sigmoid and tan-sigmoid while simultaneously simplifying the problem and reducing hardware cost is as follows:

$$\tan - \text{sigomid}\,(x) = \frac{\text{Sinh}(x)}{\cosh\,(x)} = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \quad (4)$$

$$= \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} * \frac{e^{-x}}{e^{-x}} \quad (5)$$

$$= \frac{1-e^{-2x}}{1+e^{-2x}} \quad (6)$$

By separating the numerator and denominator, Equation (6) becomes:

$$= \frac{1}{1+e^{-2x}} - \frac{e^{-2x}}{1+e^{-2x}} \quad (7)$$

$$= \text{logsigmoid}\,(2x) - \left[\frac{1+e^{-2x}-1}{1+e^{-2x}}\right] \quad (8)$$

$$= \log \text{sigmoid}(2x) - \,[1 - \text{logsigmoid}(2x) \quad (9)$$

$$\text{tansigmoid} \;= 2\text{logsigmoid}(2x) - 1 \quad (10)$$

There is a direct correspondence between tan-sigmoid and log sigmoid. Table 2 shows the values in the segmentation regions for the suggested design, and Figure 3 shows that the tan curve was separated into five pieces. An approximation log sigmoid strategy was utilized in this work.

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

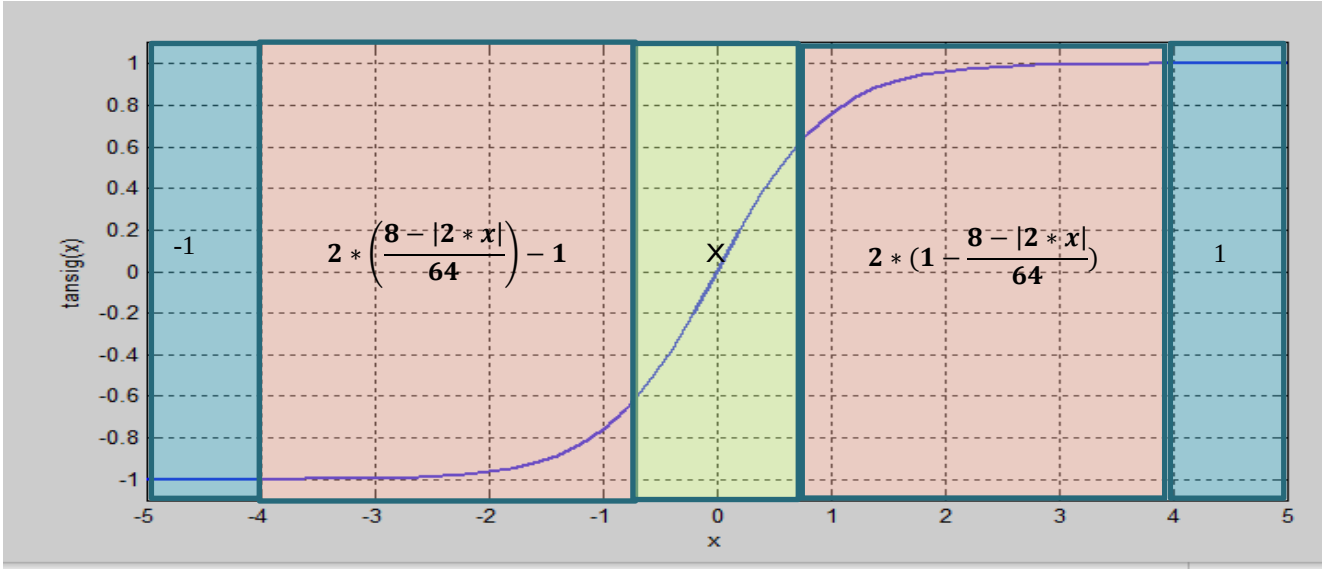

Fig.3 A method of approximation for the division of the tan-sigmoid curve

Table 2. Approximation of the tan- sigmoid

Condition	***Operation***
$x \leq -4$	$f(x) = -1$
$-4 < x \leq -0.8$	$f(x) = 2 * (\frac{8 - \|2 * x\|}{64} - 1$
$\|x\| < 0.8$	$f(x) = 2 * \left(\frac{2 * x}{4} + 0.5\right)$
$0.8 \leq x < 4$	$f(x) = 2 * \left(1 - \frac{8 - \|2 * x\|}{64}\right)$
$x > 4$	$f(x) = 1$

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

Both high precision and efficient hardware implementation are concurrently provided by the solution that has currently been offered. Figure 4 illustrates the proposed hardware implementation of the FPGA hardware by making use of an approximation.

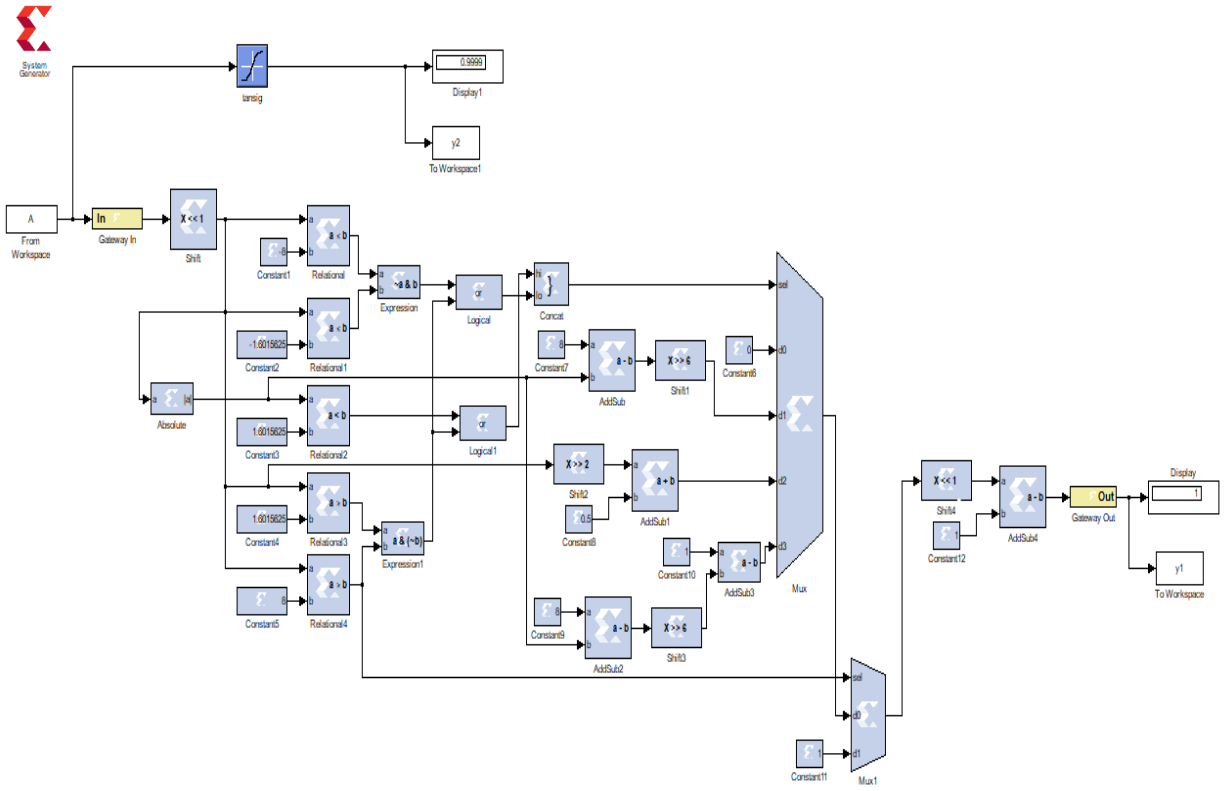

Fig 4: Proposed implementation of the tan-sigmoid function on FPGA hardware through an approximation method.

3.2 Implementation of the tan-sigmoid function on FPGA hardware

This proposal aims to substitute simpler shift operations for multiplication. For instance, when X is the input, it is shifted to the left by one place, producing (2X). The conditions are then added based on the standards listed in Table 3.1. Based on the given criteria, the output was determined using two multiplexers. When the input is less than or equal to -4, the first multiplexer (Mux) is set to zero. If the input is between -4 and -0.8, including both endpoints, select option (01). If the absolute value of the input is less than 0.8, select the value (10). Choose option 11 if the input falls between 0.8 and 4, excluding both ends. When the input reaches 4, the second multiplexer selects 1 as the value. The output is then shifted to the left, resulting in multiplication by a factor of two and subtraction by one. By directly mapping the input to a sigmoidal output, it is possible to eliminate the need for shifts and add operations and replace them with a simple logic design. This method created a highly compact and quick digital approximation of a tan-sigmoid function by using a streamlined gate design instead of the traditional multiplication and addition processes. Proposed architecture for constructing a neural network

The MLP network's remarkable performance and ease of usage makes it popular across a wide range of networks and designs. The following parameters were present in the initial Neural Network experiment: Three layers make up the feed-forward network design that is being used: the input layer, the hidden layer,

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

and the output layer. The output layer has two neurons, while the hidden layer in Figure 5 contains three neurons. The tan-sigmoid function is the transfer function for the hidden layer.

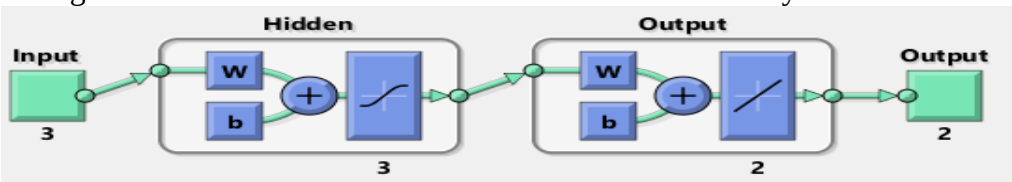

Fig 5: Artificial Neural Network

A neuron is the primary component of an artificial neural network. The neuron performs three steps: multiplying the input by the weights, adding the biases, and filtering the result using the activation function (tan-sigmoid). The neural network was trained using an algorithm that created skin and non-skin samples by picking B, G, and R values at random from face images of people of various ages, races, and genders. We obtained these photos from the FERET and PAL databases. We have got a whopping 245,057 learning samples in this dataset. Out of those, 50,859 are classified as skin samples and a whopping 194,198 are classified as non-skin samples. We have quite a selection here! One variable is the focus of the set characteristics, the attribute characteristics are genuine, and the related tasks entail classification. The PAL face database and the color FERET Image database are obtained from the Active Ageing Laboratory at The University of Texas at Dallas [23]. The data matrix is 4*245,057 in size. In the first three columns, variables X1, X2, and X3 are denoted by the labels B, G, and R, respectively. The data point is labeled as either skin (designated as 1) or non-skin (designated as 2) in the fourth column. The matrix has 245,057 rows, as Figure 6 illustrates.

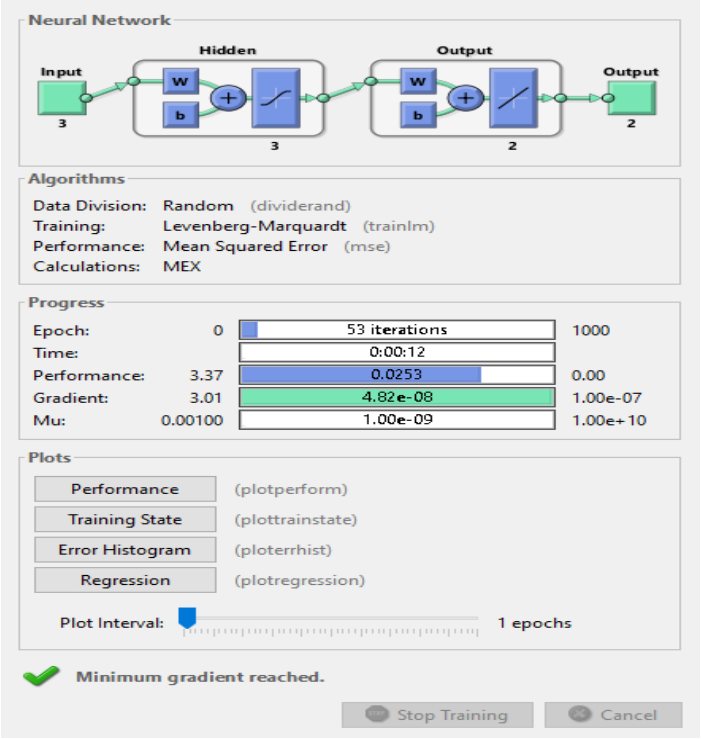

Fig 6: Training a neural network

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

3.3 Applying the approximate log-sigmoid approach to FPGA in order to implement a neural network with the suggested tan-sigmoid design.

It is pretty nifty how you can use FPGA to implement ANN's in hardware, as demonstrated in Figure 7

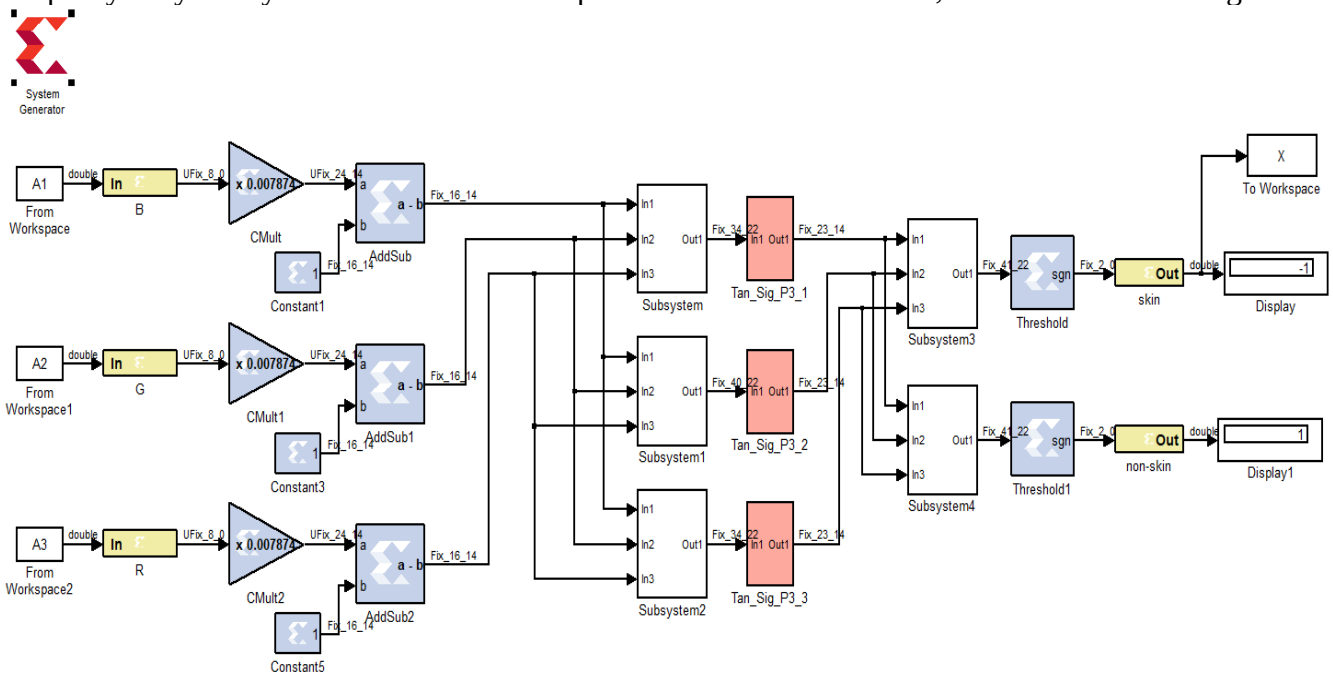

Fig7: Constructing a Neural Network with the suggested tan-sigmoid structure, utilizing an approximation log-sigmoid technique on FPGA

A1, A2, and A3 are the three elements that make up the component of the training set. The color feature indicated by the letter A1 is blue, the color feature indicated by the letter A2 is green, and the color feature indicated by the letter A3 is red. This information is acquired from the m-file in MATLAB, and it is used to determine the classification type. The classification type is established based on whether the skin is classed as 1 or no skin at all. The gain block, also known as CMult, will perform a multiplication operation on the input features. After that, a constant value of one will be removed from the output that was produced by the gain block for the outcome. The first layer, which consists of the subsystems subsystem1 and subsystem2, will be the output that is produced as a result of the add/sub operation. Figure 8 illustrates the weights that are contained within the hidden layer, which are represented by these subsystems.

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

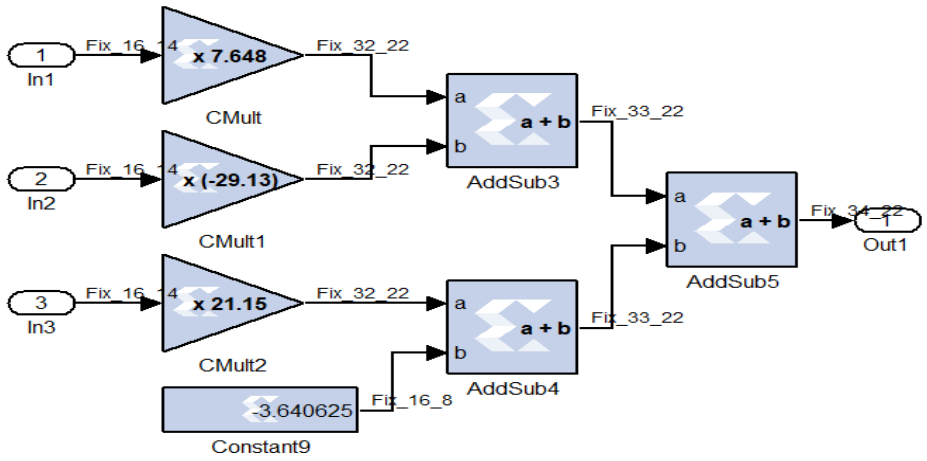

Fig 8. Block diagram of the neural network subsystem on FPGA.

Figure 8 illustrates the weight blocks CMult, CMult1, and CMult2 (the increase block) combined with constant 9. The output of these subsystems linked to tan-sigmoid blocks is based on an approximation method, as described in section 4.1. The tan-sigmoid outputs will then be routed to the second layer's subsystems 3, and 4. The output layer will represent these subsystems. After being driven to the threshold blocks, these subsystems are routed to the display blocks in the following step.

4. PROPOSED NEURAL NETWORK NORMALIZATION

The skin and non-skin datasets have been standardized so that the weight values were to be lower and easier to transform into simpler circuits. The skin dataset contains samples from a BGR color space, with three input columns designated B, G, and R, while the non-skin dataset has two columns labeled "2" and one labeled "1." To normalize each column of the dataset in accordance with equation 11, the following law was applied:

$$\text{Data Normalization} = \frac{\text{x}-\text{mean}}{\text{standrd}} \qquad (11)$$

Table 3 shows the neural network's weight and bias following data normalization.

Table3. A data normalization neural networks weight and bias

Neuron	W1	W2	W3	b
first neuron in the hidden layer	7.6465	-29.1235	21.1466	-3.6388
second neuron in the hidden layer	1.8501*10^3	0.2875*10^3	-2.2931*10^3	729.7445
third neuron in the hidden layer	-16.6496	43.4573	-23.7389	11.4078
first neuron in the output layer	0.8697	-0.1255	0.9686	-0.9777
second neuron in the output layer	-0.8697	0.1255	-0.9686	0.9777

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

4.1Utilizing FPGA to implement neural network normalization with a tan-sigmoid architecture via an approximation log sigmoid technique.

It is possible to implement the hardware using FPGA, as demonstrated in Figure 9.

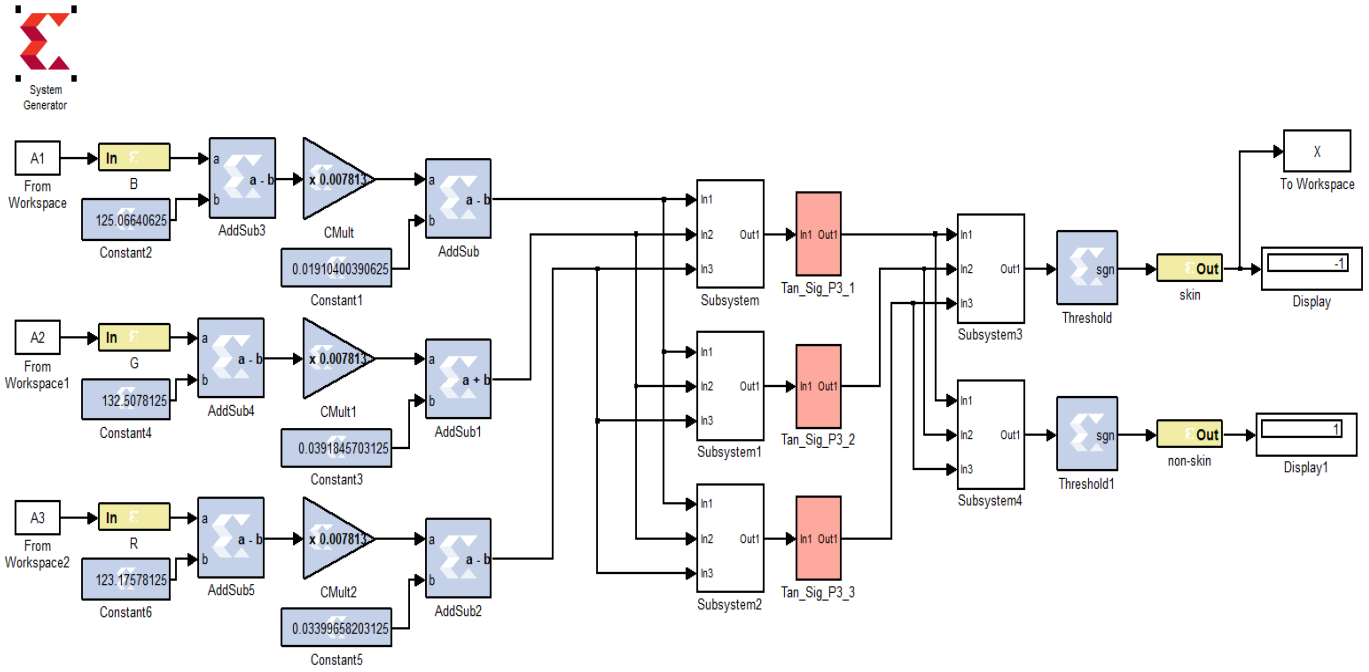

Fig 9: A neural network utilizing a tan-sigmoid for normalization design is proposed, employing an approximation of the log sigmoid function on an FPGA.

There is a classification in the dataset called A1, which corresponds to the average value of the first column. This functionality will be linked to the add/sub3 block, which merges the input (A1) with the constant 2 block. The dataset includes a feature that is not connected to the skin. The subsystem block will then get its output from the add/sub blocks, which combine the constant1 blocks with the output created by the CMult block. Next, the output of the CMult block will be connected to this block. The add/sub4 block is linked to the input (A2) and the constant 4 block, which reflects the mean of the second column in the dataset (A2). A2 is a prevalent characteristic present in both the skin and non-skin datasets. Subsequently, the gain block (CMult1) obtains its input from the add/sub4 block, while the add/sub1 block combines the signals from the constant3 blocks with the signal from CMult1. Subsystem1 obtained the output from the add/sub4 blocks. The constant6 block represents the third column (A3) of the dataset and functions as both its input and output. It is connected to the add/sub5 block. Differentiating from skin and non-skin is expressed by the dataset's A3 attribute. The output of this block was subsequently fed into the add/sub2 block, which fused the outputs of the CMult and constant5 blocks, after which it was connected to the gain block (CMult2). Finally, the results of the add/sub2 blocks were connected to the subsystem 2 blocks. The blocks comprising the initial (hidden) layer of the network are the Subsystem, subsystem1, and subsystem2. Figure 10 depicts the composition of these bricks, consisting of bias and weight blocks.

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

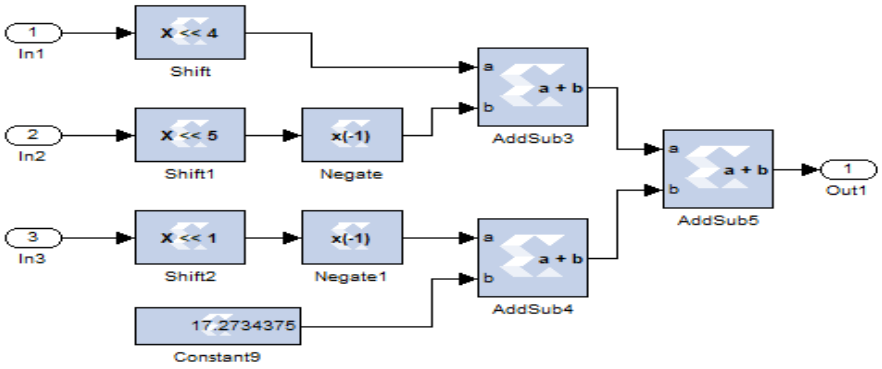

Figure 10: demonstrates the block diagram depicting the hidden layer's neural network normalization subsystem that has been built on an FPGA.

The particular weights blocks that have been swapped out for the shifting blocks on the left are shown graphically in Figure 10. The result of the shift operation (x<<4) will be passed to the add/sub3 block once it satisfies the criteria (2^16). The shift1 block has a value of 2 raised to the power of 32 when x is much less than 5. The output of the shift1 block is then linked to the Negate block's input (x (1)), and the sum of the outputs from the shift and Negate blocks is sent to the added/sub3 block. After the output of the Negative1 block has been connected to the add/sub4 block, the output of the shift2 block is delivered to Negate 1(x (-1)). Add/Sub5 will be combined with add/Sub4 to produce these results. The constant9 block, which stands for bias, will be coupled to the add/sub4 signal. The results of adds/sub4 will then be connected to the output of adds/sub5. The tan-sigmoid blocks, which were created using the approximation method described in section (4.1), will be connected to the outputs of the subsystem blocks. Ultimately, as depicted Figure 11 shows how the tan-sig blocks' output will be transferred to the output layer's subsystems 3 and 4. Subsystem 3 includes the weight block, and subsystem 4 includes the bias block.

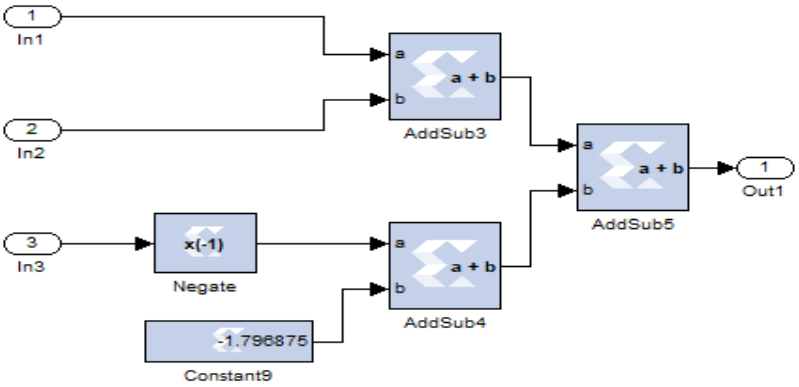

Figure 11: depicts a block diagram of the neural network normalization subsystem at the output layer, which is implemented on an FPGA.

The Addsub3, negate x (-1), and constant9 blocks exemplify the weight and bias, as seen in Figure 11. In the following steps, the subsystem's blocks send their outputs to the threshold blocks, which in turn send them to the display blocks. Finally, the blocks in the workspace stand in for the optimally created output when comparing the network output with said blocks.

Al-Furat Journal of Innovations in Electronics and Computer Engineering (FJIECE)
ISSN -2708-3985

5. RESULTS AND DISCUSSION

For the purpose of this study, two different neural network architectures are implemented by applying the suggested tan-sigmoid based on the polynomial technique. The Spartan 3A-3sd3400afg676-4, which is part of the Xilinx FPGA platform, is used to validate the recommended designs. An implementation of the suggested designs was carried out with the assistance of Xilinx System Generator (XSG) block sets. These block sets may be accessed using the ISE 14.7 Simulator and the MATLAB R2012a setup. Specifically, the number of slices, LUTs, and Flip Flop (FF) operations are compared and contrasted from the perspective of resource utilization in this study. The first one in Table 4 compares the two distinct designs for neural networks and demonstrates which one makes the least amount of use of the resources available. Subsequently, Table 4 demonstrates that, in regard to comparable activities, the proposed design has the lowest percentage of total resource use.

Table 4. Comparison of the various approaches that have been suggested

Proposed	Slice	Slices%	LUT	LUT%	FF	FF%
The suggested neural network uses a tan-sigmoid activation function based on the Approximation log method, which is implemented on an FPGA.	1,461	6	2,907	6	0	0
Suggested the implementation of a neural network for normalization.	736	3	1,414	2	0	0

The comparison of the use of resources of two neural network designs developed especially for field-programmable gate arrays (FPGAs) is provided in Table 4.. One of the designs utilizes an efficient tan-sigmoid based on approximation log approach, while the other utilizes a normalization neural network technique. The comparison demonstrates that the normalization neural network that was suggested for use on FPGA has a lower proportion of resource utilization in contrast to other neural networks, with 3% for slices, 2% for LUT, and 0% for FF.

Table 5 presents a comparison highlight that the proposed tan-sigmoid based normalization network offers a more hardware-efficient solution than the Gaussian RBF approach of Shymkovych et al[10]. While the Gaussian implementation requires additional multipliers, BRAM, and higher slice utilization to achieve nonlinear modeling, the proposed method achieves its functionality with only 736 slices (3% of the Spartan-3A device), 1,414 LUTs (2%), and no flip-flops or BRAM. This remarkably low resource utilization makes the proposed design more suitable for real-world FPGA deployments, particularly in scenarios with strict area and power constraints. Therefore, the proposed approach can be considered superior in terms of practical applicability and scalability for lightweight neural network operations. The proposed tan-sigmoid design not only minimizes FPGA resource utilization but also delivers high inference performance, achieving an accuracy of [98.8333%] with a total power consumption of [188.38mW] (static178.61mW + dynamic9.76mW). Compared to Gaussian RBF implementations, this approach offers a superior balance between efficiency, accuracy, and energy consumption, making it highly suitable for practical, real-time FPGA deployments.

Table 5. Evaluation in Relation to Past Work

Proposed	Device	Slice	Slice %	LUT	LUT %	FF	BRAM Blocks
proposed a neural network to be used for normalization.	Xilinx Spartan3 A FPGA	736	3	1414	2	0	0
[10]	Xilinx Spartan-3A FPGA	~1193	5	1193	~2	14 Multipliers (MULT18x 18) + registers	1

6. CONCLUSIONS

In this study, a polynomial-based neural network architecture is developed and implemented using hard FPGA blocks. The design incorporating weight block restructuring and data normalization achieved minimal resource usage, with LUTs at 2%, slices at 3%, and 0% flip-flop utilization through an efficient tan-sigmoid function using a logarithmic approximation. Overall, the implementation demonstrates superior efficiency compared to similar approaches. However, its applicability may be constrained when scaling to larger or more complex neural network architectures due to potential resource and performance limitations. Future work may focus on scaling the FPGA-based design to support deeper and more complex neural networks, enhancing performance for real-time applications while maintaining low resource usage. Alternative approximation methods for the tan-sigmoid function could also be explored to improve accuracy and efficiency in these extended architectures.

REFERENCES

[1] D. Mishra, A. Yadav, S. Ray, and P. K. Kalra, “Exploring biological neuron models,” Dir. Res. Mag. IIT Kanpur, vol. 7, no. 3, pp. 13–22, 2006.
[2] S. Li, W. Kang, Y. Huang, X. Zhang, Y. Zhou, and W. Zhao, “Magnetic skyrmion-based artificial neuron device,” Nanotechnology, vol. 28, no. 31, p. 31LT01, 2017, doi: 10.1088/1361-6528/aa7af5.
[3] J.-W. Lin, “Artificial Neural Network Related to Biological Neuron Network: A Review,” Adv. Stud. Med. Sci., vol. 5, no. 1, pp. 55–62, 2017.
[4] A. Zilouchian, “Fundamentals of neural networks,” Intell. Control Syst. using soft Comput. Methodol., no. 1, pp. 1–5, 2001.
[5] B. Karlik and A. V. Olgac, “Performance analysis of various activation functions in generalized MLP architectures of neural networks,” Int. J. Artif. Intell. Expert Syst., vol. 1, no. 4, pp. 111–122, 2011.
[6] L. Zhang, “Implementation of fixed-point neuron models with threshold, ramp and sigmoid activation functions,” in IOP Conference Series: Materials Science and Engineering, 2017, vol. 224, no. 1, p. 12054.
[7] A. Mellit, H. Mekki, A. Messai, and S. A. Kalogirou, “FPGA-based implementation of intelligent predictor for global solar irradiation, Part I: Theory and simulation,” Expert Syst. Appl., vol. 38, no. 3, pp. 2668–2685, 2011.
[8] M. Wess, P. D. S. Manoj, and A. Jantsch, “Neural network-based ECG anomaly detection on FPGA and trade-off analysis,” in 2017 IEEE International Symposium on Circuits and Systems (ISCAS), 2017, pp. 1–4.

[9] Wu, E., Zhang, X., Berman, D., Cho, I., & Thendean, J. (2019, February). Compute-efficient neural-network acceleration. In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (pp. 191-200).
[10] Shymkovych, V., Telenyk, S., & Kravets, P. (2021). Hardware implementation of radial-basis neural networks with Gaussian activation functions on FPGA. Neural Computing and Applications, 33(15), 9467-9479.
[11] D. Anderson and G. McNeill, "Artificial neural networks technology," Kaman Sci. Corp., vol. 258, no. 6, pp. 1–83, 1992.
[12] M. Hajek, "Neural Networks. University of Kwazala," 2005.
[13] S. Oniga et al., "FPGA implementation of feed-forward neural networks for smart devices development," 2009 Int. Symp. Signals, Circuits Syst., pp. 1–4, 2009.
[14] S. Sharma, "Activation functions in neural networks," Towar. Data Sci., vol. 6, 2017.
[15] Kumbhar, R. R., Radhika, P., & Mane, D. (2023, April). Design and optimization of an on-chip Artificial neural network on FPGA for recognizing handwritten digits. In 2023 International Conference on Recent Advances in Electrical, Electronics, Ubiquitous Communication, and Computational Intelligence (RAEEUCCI) (pp. 1-5). IEEE.
[16] Chotikunnan, P., Khotakham, W., Chotikunnan, R., Roongprasert, K., Pititheeraphab, Y., Puttasakul, T., ... & Thongpance, N. (2025). Enhanced Angle Estimation Using Optimized Artificial Neural Networks with Temporal Averaging in IMU-Based Motion Tracking. Journal of Robotics and Control (JRC), 6(2), 1069-1082.
[17] H. M. H. Al-Rikabi, M. A. M. Al-Ja'afari, A. H. Ali, and S. H. Abdulwahed, "Generic model implementation of deep neural network activation functions using GWO-optimized SCPWL model on FPGA," Microprocess. Microsyst., vol. 77, p. 103141, 2020.
[18] H. K. Ali and E. Z. Mohammed, "Design artificial neural network using FPGA," IJCSNS, vol. 10, no. 8, p. 88, 2010.
[19] Ali, M., & Abed, B. (2022). The Proposition of Three Approaching Ways to Implement Tan-sigmoid Activation Function in FPGA. (2022) Engineering and Technology Journal, 40(2), 311-321.
[20] Ali, M. T., & Abd, B. H. (2022, May). An Efficient area Neural Network Implementation using tan-sigmoid Look up Table Method Based on FPGA. In 2022 3rd International Conference for Emerging Technology (INCET) (pp. 1-7). IEEE.
[21] L. Moreira, R. Vettor, and C. Guedes Soares, "Neural Network Approach for Predicting Ship Speed and Fuel Consumption," J. Mar. Sci. Eng., vol. 9, no. 2, p. 119, 2021.
[22] Saravanan, V., Dayana, R., Rani, M. T., Sudha, M., Varun, M., & Rosi, A. (2024, August). FPGA implementation of stochastic approximate multipliers for neural networks. In 2024 First International Conference on Electronics, Communication and Signal Processing (ICECSP) (pp. 1-6). IEEE.
[23] "UCI Machine Learning Repository: Skin Segmentation Data Set." https://archive.ics.uci.edu/ml/datasets/Skin+Segmentation (accessed May 21, 2021).