International Journal of Smart Electrical Engineering, Vol.10, No.3, Summer 2021 ISSN: 2251-9246
EISSN: 2345-6221

pp. 135:140

Detect and Implement Facial Modes in Image on FPGA using Multilayer Neural Network

Ali Abdolazimi[1], Amir Sabbagh Mollahoseini*[2],Farshid Keynia[3]

[1,2] Department of Computer Engineering, Kerman Branch, Islamic Azad University, Kerman, Iran
[3]Department of Energy Management and Optimization, Institute of Science and High Technology and Environmental Sciences, Graduate University of Advanced Technology, Kerman, Iran

Abstract

Face is a unique characteristic of the human. Detecting the state of the human face, due to its difficulty on the one hand and its many useful features on the other hand, is one of the most important issues in the image processing. In this paper, a five-layer perceptron artificial neural network (MLP) with a supervisor as a complete connection has been used to separate the different facial modes. Learning in the MLP network is done deeply with a high number of layers. The network has 4 class: anger, fear, happiness and surprise. First, the main points and areas of the face that are effective in detecting the state of the face are extracted by edge finding, and then, using the matching of the Fourier series diagram on the operational points of the face, the diagram of those points is obtained. From this diagram, a number of features in the form of three coefficients and an angular velocity are used for network training. Face database images with fixed backgrounds are used for network training. This network is first implemented with Matlab and then MLP layer multiplex is used to implement on FPGA. The results show that the proposed method can be implemented on FPGA platforms with low cost and limited resources, with appropriate output accuracy. In this paper, in addition to speed, accuracy has been tried to create an application system for communication between humans and computers.

Keywords: Detecting facial modes, MLP, implementation FPGA, Neural Network.
Article history: Received 05-July-2021; Revised 12-July-2021; Accepted 18-Aug-2021. Article Type: Research Paper

http://dx.doi.org/10.30495/ijsee.2021.684015

1. Introduction

Detection of the face and its states, due to its sensitive and diverse features, has been one of the most important research topics in the last two decades. Detected human behavior or state are used for stronger and more effective interaction between human and computer. To achieve natural and comprehensive human-computer interaction, human facial modes can be used as an interface. Face detection uses in the field of video surveillance, control of users' access to system resources and image retrieval from large image databases. In designing animations, designing different characters in games and also in computer graphics, face detection can be used properly [1]. On the other hand, among the configurable hardware, it has high speed, multiple resources and prompt processing. There are several hardware platforms for hardware implementation. Each of these platforms provides a kind of balance between efficiency and flexibility and planning. Performance here refers to computational efficiency, and the usual measure for measuring.

It is the number of instructions per second. With comparing hardware platforms such as FPGA, ZISC and DSP [2] it can be seen that FPGA is reprogrammable and can provide a lot of flexibility for the designer. Although FPGA-based customizable computational architectures are suitable for hardware implementation of neural networks, implementation networks with a large number of neurons and high, computational volume on FPGA artificial neural ANN is still a challenging task. It is performed on the features of the facial organs, the most important of which are the eyes, mouth and eyebrows. Facial modes are created by changes in different points of the face. Each of these parts is called the active unit (AU). Using the rules, the movements of these limbs are determined by different AUs. He described each specific state of the face with a number of AUs, but the way changes in AUs varies from person to person, even to show a particular state. It then groups the features based on their geometric relationship, and in addition, it can be extended to different views of the face [5]. Input data should be independent of rotation, distance, size and intensity of ambient light as well as facial skin color.

With the prepared matrix in hand, facial modes can be detected in various ways such as statistical methods, phase, support vector machine (SVM), Markov hidden model, decision tree (DT) and neural network. The method used in this paper is based on AU. At first, a 5-layer neural network is used to classify facial modes in Matlab. Then a 5-layer network on the FPGA Spartan3 Xillinx using optimizations performed and Layer multiplexing is performed in the neural network by determining the geometric distances and angles of the eyebrows, lips and eyelids, the input matrices of the diagnosis. The direct implementation of the neural network is too time consuming and requires complex calculations [6], [7] and consumes a lot of resources. This mathematical model takes the digital neural network to the gate level and implements the gates directly, but in the parallel architecture presented in this paper, all layers are parallel. They are able to work with a multiplexer in the control layer, and for this reason, the performance speed is very high. The sigmoid function has been implemented in hardware, and a new method has been used to store weights in computational units that can simplify the implementation of neural networks on hardware. Using the addition and subtraction operations instead of multiplying [8] and doubling the value of the activation function (sigmoid function) is another trick used to save resources and reduce computational volume.

2. Literature Review

Please use automatic hyphenation and check your spelling. Additionally, be sure your sentences are complete and that there is continuity within your paragraphs. Check the numbering of your graphics and make sure that all appropriate references are included.

In order to compare the paper with the existing methods, the related works are divided into three categories.

- Diagnosis of facial modes using neural network without hardware implementation.
- Methods that have recently implemented neural networks with different optimizations for specific applications on FPGA, in the second part is only the implementation of neural networks and methods of optimizing and reducing available resources.
- Implementation a neural network to detect face and state on FPGA.

Peng et al. [9] introduced a technique called LCCR to increase discrimination in representative images. The LCCR applies to five different databases with five remote measurements. In the case of minor faces, they use three facial features, namely the right eye, nose and mouth with chin, by covering the main images. The results show that the right eye, mouth and chin have a high detection rate [10]. Murphy et al. [11] show the mechanism of human facial perception based on facial stimuli. Their work shows that it is difficult for a human being to perceive a face when he has turned back. In addition, in their experiments, they tried to measure the ability of a divider to classify the presented faces as a whole and region by region using a dynamic diaphragm that gradually moved from the face image.

The main idea in this work is to recognize the limitations of human ability to understand and recognize. In their work, they tested the idea in four modes: identity, gender, age, and emotion in four conditions: full face right and left, full face rotated, straight diaphragm, and inverted diaphragm. The results show that the detrimental effects of the inverted face on showing a partial face to the participants are not less than the absence of diaphragm. Andre and Nomena [12] have studied face detection on partial faces due to the presence of emotions on the face. Minor faces mean incomplete faces that only some AUs can detect. In one of their experiments, they tested facial detection for six common emotions: happiness, anger, sadness, disgust, normaly, and fear. In the case of a partial face, the face was divided into two parts, one containing the eyes and the other containing the mouth. The remarkable result of their work is that humans have a poor detection rate when they only reach the condition of the eyes and mouth. On the other hand, they noted that the feeling of a smile produces a relatively better detection rate. However, when dealing with the acute closure of a face, the performance of current methods is significantly reduced. Many previous studies point out that when it comes to detecting the human face, familiarity seems to be a key cognitive factor. Of course, the effect of familiarity changes when the image of the target face is partial, closed, with emotions, or his age has changed. Lahazan et al. [13] have proposed a framework called OSPE for face detection in a variety of situations. For example, closed faces, facial modes, and changing brightness are some of the tips used in testing them. Again, their experimental results show that improvements in detection rates are achieved by partial facial data. In addition, Dona et al. [14] have developed a technique called TPGM to improve the cognitive process of using partial faces. In the next section, we take a brief look at parallel and optimal implementations that reduce resource utilization. In a solution [15] for the implementation of high-volume networks with a high number of layers is considered. In this study, a multiplier implements each neuron, and nonlinear operations, such as non-linear network activation functions, are converted into linear blocks, which are then implemented. In this implementation, for example, if an FPGA has 3600 multipliers, a 40 × 40 neuron can be implemented with it. Internal memory is used to store the results of each layer along with the input and weights, and for that, Some FPGAs with limited memory use external memory.

An implementation for the Xor issue was performed on the Virtex chip [16]. According to the paper, this implementation is based on a multilayer perceptron network and the use of a post-diffusion algorithm, and is used in real-time fields such as pattern detection, image processing, and audio processing, and so on. It consists of three main control units: the post-emission unit, the forward unit, and the control unit, which controls both of the previous units. The implemented network has three layers, the first layer, like the Xor gate, has two inputs, the hidden layer consists of two neurons, and the output layer has one neuron. In a leading network [17] using two activation functions on the hardware platform 5 series Virtex, FPGA is implemented. The innovation of this paper is in implementation activation functions that use digital computer algorithm with coordinate rotation. This algorithm converges to the answer using iteration. Two types of activation functions are implemented using this algorithm: sigmoid function and hyperbolic tangent function.

Neurons and connections between layers are also implemented directly using a multiplier. VHDL hardware language and ISE implementation environment is Xilinx and ISim emulator. According to the paper, with this method, the accuracy of the output results is very high compared to other implementations of activation functions, and even the speed has been increased. The next section examines the implementation of neural networks on the FPGA for face detection and its state.

McCreedy et al. [18] designed and implemented by 2-Transmogrifier configured software. This implementation uses 9 FPGA boards. Cedri and colleagues [19] implemented a neural network based on face detection on the FPGA Model II-Vertex pro. Skin color filtering and edge detection are used to reduce processing time. However, some operations on the PowerPC processor are implemented with the software installed. Cho et al. [20] have proposed a method for using FPGAs to accelerate face detection based on the Haar feature classifier. They retrained the Haar attribute with 16 categories in each step. Although only classifiers are implemented in FPGA. A host microprocessor creates images integral and detects face. However, the most powerful Model 5-Vertex FPGA is used for implementation because the designed size is too large. Hiromoto and his colleagues [21] have implemented a real-time object discovery based on the AdaBoost algorithm. They provide a hybrid architecture of parallel processing modules for step formatting as well as a sequential module for sequential steps in the cascade design. Because the parallel processing module and the sequential processing module are split after processing time evaluation, they must be redesigned and implemented to provide data with new features. However, the experimental results and the analysis of the implemented system are not discussed.

3. Implement different face modes in Matlab

In this paper, the 5-layer perceptron network (MLP) with the most widely used algorithm used in MLP networks, namely the post-diffusion algorithm is used for separation. In the post-diffusion algorithm, there are two computational paths: feed path and return path. Route path the network parameters do not change during the computation and the stimulus functions act on each neuron. In the return path, the work starts from the last layer, the output layer, where the error vector is available. Then the error vector from the side Right to left is distributed from the last layer to the first layer, and the local gradient, neuron to neuron, is computed by the recursive algorithm [22]. For the input pattern pm, the square of the output error for all cells of the network output layer becomes as equation 1 [23]:

$$E_p = \frac{1}{2}(d^p - y^p)^2 = \frac{1}{2}\sum_{j=1}^{I}(d_j^p - y_j^p)^2 \quad (1)$$

Where djp is the desired output for the j cell in the output layer and output is the actual d for the j cell in the output layer, s are the dimensions of the output vector, yp is the actual output vector, and dp is the desired output vector. The total error reference E for the pattern P is equation (2):

$$E = \sum_{P=1}^{P} E_P = \frac{1}{2}\sum_{P=1}^{P}\sum_{j=1}^{I}(d_j^p - y_j^p)^2 \quad (2)$$

The weights are adjusted in order to reduce the cost function E to a minimum by descending gradient method. The equation for updating weights as equation (3) is:

$$w_{ij}(t+1) = w_{ij}(t) + \eta\Delta w_{ij}(t) + \alpha\Delta w_{ij}(t-1) \qquad (3)$$

Where $\Delta w_{ij}(t) = -(\frac{\partial E_P}{\partial w_{ij}(t)})$, learning rate η , temporal coefficient α of new weight $w_{ij}(t+1)$ and old weight $w_{ij}(t)$. Also in this method, weights are frequently updated for all learning patterns. The learning process stops when the total error, E, for the pattern p falls below the set threshold value or the total number of training periods ends. The error propagation training method reduces the probability of convergence in local minima [23]. During the learning process, the network learning rate is regularly measured by the target functions, and finally the networks with the lowest error rate and the highest accuracy and sensitivity are accepted. Some objective functions are: root mean square error (RMSE), mean absolute error (MAE), sum of squares error (SSE) and good fit coefficient (R.) In this network, the neurons of each layer are completely connected to the neurons of the previous layer. After affecting the actuator function, each layer becomes the input of the next layer, and this process continues until the network output is obtained.

Activation function in this paper is the logistic unipolar sigmoid function. The unipolar sigmoid function, whose diagram is S-shaped, is the most common form of activation function in artificial neural networks. This function, also known as the incremental uniform function, shows a good balance between linear and nonlinear behavior. An example of a sigmoid function is a logistic function in which g is the slope coefficient of the sigmoid function. By changing the parameter g in (4), the sigmoid function with different slopes is obtained [24].

$$F(net) = {}^{1}\!/_{(1+e^{-g.net})} \qquad (4)$$

As g tends to infinity, the sigmoid function becomes the threshold function. The important thing about this function is that it is a derivative function and this is a very important issue in neural networks. Common modes of emotion detection are six classes: anger, fear, happiness, normally, sad, and surprise. For optimal implementation on FPGA, two common modes have been removed so the network has four classes: Anger, Fear, Happiness, and Surprise. First, the main points and areas of the face that are effective in detecting facial modes are extracted by edge finding. Fig. 1 shows a happy one-face image of the Kanade-Cohn dataset with three edge-finding methods applied to it [25]. Here, the Bobby Sobel edge method is used. After the edge-finding step, it is time to remove the extra lines and join the desired lines. For this purpose, morphological filters have been used. The first operator used is the bridge operator, which is used to bridge between discrete pixels and is as follows:

$$\begin{array}{c}001\\101\\100\end{array} \Rightarrow \begin{array}{c}011\\111\\110\end{array}$$

Pixels with a value of zero become one in a binary image. The next operator is the diagonal bridge that converts the zeros in the original diameter to one, as follows:

$$\begin{array}{c}010\\101\\101\end{array} \Rightarrow \begin{array}{c}110\\111\\101\end{array}$$

The result of applying these two operators is filling the lines of the mouth and eyes. After applying the morphological agents, it is time to connect the eyes, mouth and eyebrows to get their place. With the help of imfill and imclose morphological operators in the form of 4-point, 6-point and 8-point dots for filling and points 0 and 1 of the array for closing, the lines of eyes, mouth and eyebrows can be turned into full objects [26]. Fig. 2a is the result of applying morphological filters to the original image. The next filter is for deleting continuous pixels that are smaller than a certain value. This filter is done by the “bwareaopen” command and has 4, 6 and 8 pixel connection points. Finally, in Fig. 2c, the desired areas appear on the main image, which includes the mouth and eyes. Using Fourier series and w which is shown in the equation 5, 20 face features such as mouth , eye, and eyebrow can be obtained. That is, in fact, the inputs of the neural network are 20. Fourier series coefficients describe the state of the face and are taught to the network as a feature.

$$f(x) = a_0 + a_1\cos(x\omega) + b_1\sin(x\omega) \qquad (5)$$

Where x and $f(x)$ are obtained as coordinates of points and coefficients and they are multiple by a_0 , a_1, b_1 and ω is angular velocity [27].

As an example for the image shown in Fig. 1, the above values for the lower lip are as $a_0 = 176 \cdot 4$ $a_1 = 70 \cdot 82$ $b_1 = 6.66$ $\omega = 0 \cdot 071$

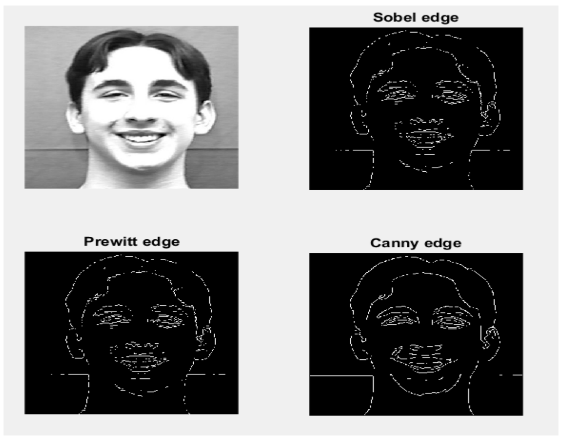


Fig. 1. Drawing a happy face from the Kanade-Cohn Dataset

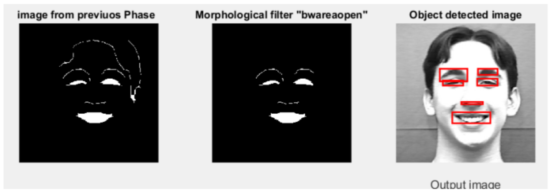


Fig. 2. Apply morphological filters on the main image, remove excess lines by this filter and detect the main areas of the face on the main image

A) Database

In the dataset matrix, we have 21 columns, 20 of which are related to the Fourier series coefficients, ie our inputs, and the last column is the class number (1 to 4 for 4 classes). At this stage, the last column of each class is removed and replaced by four columns The following 4 classes are added to the mark:

- 0001 means class one of four classes
- 0010 means class two of four classes
- 0100 means class three of four classes
- 1000 means class four of four classes

Then we produce training and test matrices from these 4 classes in the form of 60% training and 40% testing. Having a diverse and powerful database on the one hand helps in accurate detection and on the other hand in processing speed 215 images used to be created from the Kanade-Cohn database. Fixed face images with a specific background are used to teach the network. The sample number of each class is as follows:

- Class1: 57
- Class2: 57
- Class3: 44
- Class4: 58

Rows of test and training matrices are also moved so that examples from all classes are evenly distributed. The training matrix has 147 examples and the test matrix has 68 examples, which adds up to 215, which is the sum of the total samples. After making the matrix, it is time to normalize the matrix elements, which is done for ease of calculation so that all the elements are placed between zero and one. Then four classes are separated from each other and the rows of each class are moved randomly.

B) Network input matrix

The network input matrix has 20 properties and 4 classes and has 24 columns. From each selected member (upper lip, lower lip, and right eyelid, left and right eyebrow) four Fourier series feature coefficients are extracted so 20 features are extracted. Table (1) shows the number of features for each example.

Table.1.
Number of features to generate neural network input matrix

	Upper lip, Lower lip, Right eyelid, Left eyebrow, Right eyebrow			
EX.1	ω	b_1	a_1	a_0
EX.2	ω	b_1	a_1	a_0
EX.3	ω	b_1	a_1	a_0
EX.4	ω	b_1	a_1	a_0

C) MLP network structure

The network selected in this part of the MLP network is supervised learning by five layers that have three hidden layers, in the form of 20-8-8-8-4 input layer of 20 neurons, output layer of 4 neurons and hidden layers each have 8 neurons. First, the database matrix is called and the output and properties are separated from each other. Then, three matrices are randomly assigned as weights for the links between the input and hidden layers first, the first and second hidden layers, and the second hidden layer. Output layers are generated.

The network starts for a certain number of iterations. Examples Line by line pass through the activation function, which is the sigmoid function, in each layer, multiplied by the values of the weights, and move on to the next layer. Weights are updated after the repetitions are completed. Therefore, the method used in the network is Batch learning. Weight update values are obtained from equation (3). Then the error signal and the mean of the total error are obtained for all examples of relations (1) and (2). An error or repetition condition can be set for the end of the network. Fig. 3 shows the structure of a five-layer perceptron network, including layer one, layer two, layer three, and the output and number of per layer neurons.

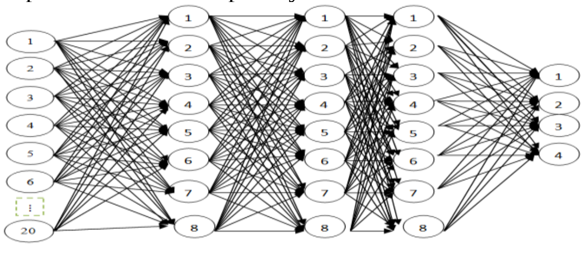


Fig. 3. Five-layer perceptron network including hidden layer one, hidden layer two, hidden layer three and output.

D) Results of 5-layer perceptron network in Matlab

The three parameters Specificity, Sensitivity and Accuracy are calculated for each class. which is obtained from relations 6 to 8 .

$$sensitivity = \frac{TP}{TP + FN} \tag{6}$$

$$specificity = \frac{TN}{TN + FP} \quad (7)$$

$$accuracy = \frac{TP + TN}{TP + tn + fp + FN} \quad (8)$$

The learning rate of the network is 0.001, which is more suitable for coming out of the local minimum, but the processing speed is slower and in fact the forward steps of weight correction are much smaller. In fact, the 5-layer network in normal mode and with three hidden layers in the form of 8-8-8 has 96.73% of the total accuracy and the total error of training is 0.42 but the main reason for this increase is actually the training and test and normalization matrices that have been done in these matrices. In fact, when the data is between zero and one, the network has less variance and scatter, and is trained much more accurately, and the results are better. The following are the results and tables related to the optimized network.

The average accuracy of all four classes in 24000 repetitions is 96.73%, which has a desirable value. Fig. 4 shows the error diagram for the four classes. As can be seen, the network has an average error (Eav) of 0.037 in 3000 repetitions, which is the same as the mean error in relation (2). Table 2 shows the specifications of the 5-layer MLP network used in terms of structure and parameters used. In this table, the training type is sequential Learning, which means that unlike Learning Batch, weights are updated in each repetition.

The training error of each class along with the network simulation results and the parameters of accuracy, sensitivity and specificity in each class are shown in Table 3. Fig. 5 also shows the bar chart of network parameters. Table 3 shows the sensitivity, specificity and accuracy of each class according to the relations 6, 7 and 8. In the last row of the table, the data used in each class is shown and it can be seen that the examples in all four classes are evenly distributed and total number is 215.

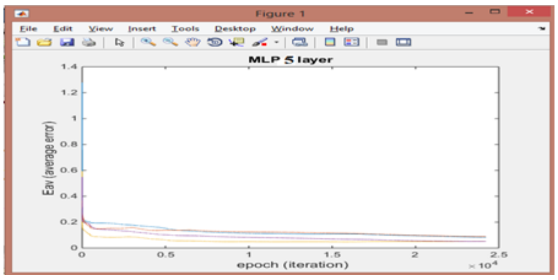


Fig. 4. Training error reduction chart for all 4 classes in 24000 repetitions.

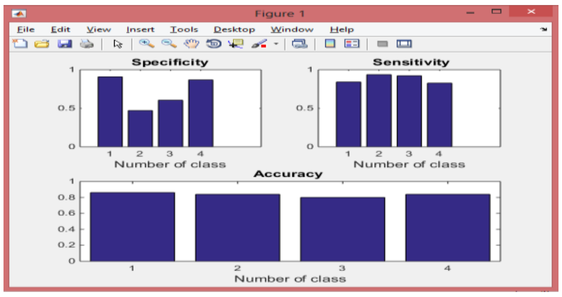


Fig. 5. Bar chart of three parameters of accuracy, sensitivity and specificity for 5-layer network.

Table.2.
5-layer MLP network specifications used

Item	***Value***
Number of Features	20
Number of Epoch	24000
E_{av}	0.037
ACC$_{av}$	96.73%
Connection	Full connection
Number of layers	5
Learning rate	0.001
Train example	147
Test example	68
Learning way	Sequential learning
Number of neurons	20-8-8-8-4

Table.3.
5-layer MLP network results in Matlab.

	Anger	***Surprise***	***Fear***	***Happiness***	***Total***
Accuracy	98	97	90	98	96.73
Sensitivity	95	88	92	100	93.75
Specificity	100	98	93	98	97.25
FN	1	1	5	0	-
FP	0	1	4	1	-
TN	48	60	48	48	-
TP	20	8	20	20	-
E_{av}	0.083	0.089	0.0521	0.051	0.068
Number of instances	57	57	44	57	215

Table.4.
Compare the results of 5-layer MLP network with other

	MLP 5-Layer	***[28]***	***[29]***	***[30]***	***[31]***	***[32]***
Average Accuracy	96.73	99	96.76	98	91.5	89.85
Average Sensitivity	93.75	-	-	-	-	-
Average Specificity	97.25	-	-	-	-	-
E_{av}train (MSE)	0.027	-	-	0.004	-	-
E_{av} test (MSE)	0.11×10^{-4}	0.07	0.024	0.002	-	-

E) Compare paper results with other sources

In this section, the results of the 5-layer MLP network in Table 4 are compared with other sources. It should be noted that according to Tables 3 and 4, the accuracy of this network is repeated for 6000 and later repetitions, which has a value of 96.73% , along with sensitivity and specificity in this repetition. In reference [28], the phase separator method is used in such a way that the colour image of the face is immediately divided into two areas of the eyes and mouth, and in fact, only these two parts of the face are used. From 54 Gabor functions and filters are used to detect states, then the database matrix is given to the PCA algorithm to optimize the properties and then to a phase separator. 97% accuracy is for one area and 99% for two areas. In reference [28], the phase separator method is used in such a way that the colour image of the face is immediately divided into two areas of the eyes and mouth, and in fact, only these two parts of the face are used. From 54 Gabor functions and filters are used to detect states, then the database matrix is given to the PCA algorithm to optimize the properties and then to a phase separator. 97% accuracy is for one area only and 99% for two areas.

[29] uses convolutional neural networks in which facial images can be fed directly to the network for analysis. In this research, three databases -BU, JAFFE +, CK 3GGG have been used and according to the author's claim, the accuracy has reached 96.76%. In reference [30], Gabor filters have been used for the feature. Then the two algorithms PCA and LDA are used sequentially for database and extraction of the best features and this feature matrix is given to a neural network for segmentation.

According to the author, for 199 faces, they have obtained a value of 98%. Reference [31] has selected one of the support vector machine separator network in addition to rabid features to diagnose facial modes in 3 tones. In this research, three states of surprise, sadness and happiness have been used, and the state of surprise has the highest accuracy and value of 97%. Sad and happy states have values of 84.5% and 93%, respectively. The overall accuracy of the grid is 91.5. Reference [32] uses an RBF radial approximation grid with Gabor 2D filters for 6 face modes. In this research, two databases, JAFFE +, CK are used, the accuracy of which is 88% for JAFFE database and 91.7% for CK + database. If we want to calculate the average values of the two bases for the total accuracy of the network, the accuracy of the whole network is 89.85%.

4. Performed implementation on FPGA without layer multiplexing

The implementation is done in two ways: normal implementation and implementation using multiplexing layers. The following modules and the Verilog language implement the network implemented in Matlab. The neural network implementation modules 20-8-8-8-4 without multiplex and with multiplex layers are as follows:

– Middle layer neurons
– Output layer neurons
– Multiply Booth
– Sigmoid stimulation function
– Control unit for neurons and layers
– RAM corresponding to the weights corresponding to the links between the layers

When using layer layers, some of the modules introduced undergo changes, including the neuron and layer control module and the middle layer module. The number of intermediate modules in the layer multiplex is reduced from three to one. The reason for this is to reduce the middle or hidden layers from three layers to one layer. Each neuron in each layer is assigned a module. For example, hidden layer and output neurons have two separate modules. For multiplication, shift and addition operations are used instead of multiplication. This algorithm is performed in Booth multiplication and will be explained below.

To maintain the values related to the weights and bias of the neurons, six RAMs have been used. If we show the number of neurons in the middle layer with N, the RAM of the computational model of the link is placed so that in the zero address of this RAM the values of the weights. The links between the inputs and the number one neuron in the middle layer are placed and the other weights are placed in the same way and a total of N addresses are required.

After the weight of the links and the bias values in the learning phase are calculated by Matlab software, Data are included in the FPGA by the above rules. All neural network links are accessible by sequential increment of the address. This is shown in Fig. 6. When the address is zero, the operation shown in Fig. 6 is performed. By increasing the address to 1, more links are covered. These links are shown in Fig. 7. Therefore, all network links are available by increasing the address from zero to N (N number of middle layer neurons).

In Fig. 6, only the number one neuron of the first middle layer is complete, the neurons one to eight have two other middle layers, and the neurons one to four of the output layer are incomplete.

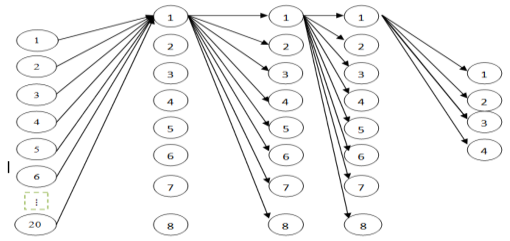


Fig. 6. Sequential increase of address and how to calculate links and preparation of the amount of neurons in the first middle layer

Since in hardware implementation all the code is executed in the software together, so we need to address and move forward systematically because it is not possible to get the output by multiplying all the neurons in the weights at once. Here the values of the neurons The output cells are not complete but multiplied by their own weight, ie 8, 7, 6, 5, 4, 3, 2 neurons from the second and third middle layers are still remain, so the output is complete when the amount of neurons in 8 middle layers in the second and Third be prepared. In this clock, we used below RAMs to multiply the values.

- Related to the first input and middle layer (named Mid_weight_Mem) in the implementation at the address A0.
- For the first and middle layers (called Mid2_weight_Mem) in implementation at address A0. The second middle layer and the third middle layer) called A0
- Address in pavement (Mem_weight_Mid3 refers to the bias of all intermediate layers, which is a value of RAM that is connected to all three layers because bias is an arbitrary fixed number) usually one (in the address A0.
- Related to interlayer values the third intermediate and the output layer at address A0.
- Corresponds to the bias of the output layer.

In Fig. 7 at address A1, as can be seen, the neurons No. 2 of the first middle layer are completed and the other three layers are one-step closer to completion. If careful, a new value is added to the second and third middle layer neurons number one, which is the value of the second neuron of the first middle layer.

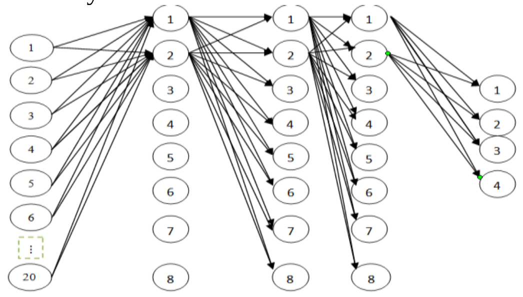


Fig. 7. Increase the address to one and how to calculate the second neuron of the first middle layer.

As mentioned earlier, the values of these neurons are incomplete until all the neurons in the first middle layer are complete and the output of one-step is closer to its final value. Therefore, with eight clock (the number of neurons in the middle layer), the amount of all neurons is complete and the output is complete.

A) Middle layer neurons

In the middle layer module, the inputs are 16 bits, and since we have 20 inputs, a total of 320 bits, which is from zero to 319, are intended for the inputs and the weights waiting for it. The first middle layer module can be considered as Fig. 8. clk, start, HZ-bus Single-bit input signals for Clock, respectively. Computational operation of impedance neurons (HZ). The output bus lines are available to this neuron. These 319 bits contain 8 weights of links between 20 inputs and middle layer neurons. Weight_Mid is the input value. Mid_Weight 319-bit input from a row of RAM addresses related to the weights of the links between the input and middle layers.

The 319-bits input contains eight weights of links between 20 inputs and the middle layer neurons. Since the weights are 16 bits like the inputs, the 319 bits of 20 contain 319 bits of input are Input_Mid. The weight_Mid is the input value. Two middle layer modules called Mid two_Neuron and Mid three_Neuron are added along with two RAM modules for the following two middle layers. As can be seen in the middle layer modules, the signals of all three modules (clk, hz-bus) are interconnected, which means that the three middle layer modules work together.

Moreover, has an 8-bit output that connects to the next middle layer. This module multiplies the weights inside the first RAM module at the inputs and passes the sigmoid activation function, which is the SIG_LUT module, and produces the result as an 8-bit number. Fig. 9 shows the second module of the middle layer.

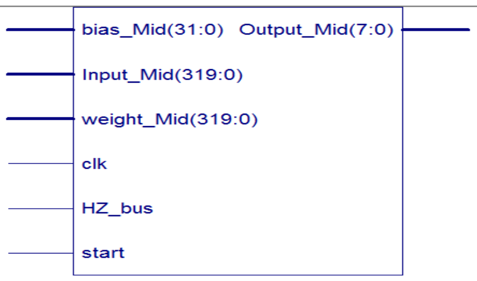


Fig. 8. The first middle layer module

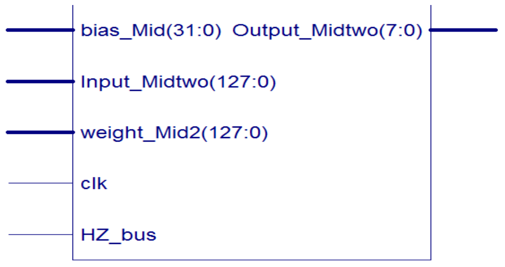


Fig. 9. Mid two Neuron The middle layer module for the second layer

This module also has a 128-bit input because each input is 16-bit and has an 8-bit output that connects to the next middle layer. Note that for the convenience of the output of the first middle layer, which was 8 bits, it becomes 16 bits. This is just to name the multiplied bits in the modules.

- Input_Mid two [15: 0], weight_Mid2 [15: 0]
- Input_Mid two [31:16], weight_Mid2 [31:16]

If we wanted to use the same 8 bits of output of the first module, it would be multiplied as follows:

- Input_Mid two [7: 0], weight_Mid2 [15: 0]
- Input_Mid two [15: 8], weight_Mid2 [31:16]

This module also multiplies the weights inside the second RAM module at its inputs, which is the output of the previous middle layer module, and passes the sigmoid activation function, which is the same as the SIG_LUT module, and produces the result as an 8-bit number. Fig. 10 shows the third module of the middle layer.

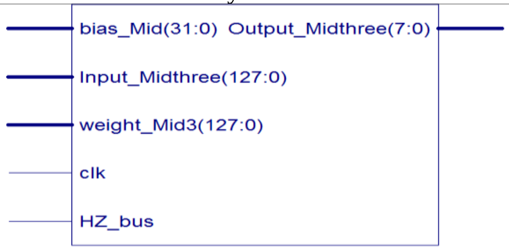


Fig. 10. Mid three_Neuron Middle layer module for the third layer.

This module also has a 128-bit input because each input is 16-bit and has an 8-bit output that connects to the next middle layer. It works just like the previous modules.

B) Output layer neurons

Fig. 11 shows the output layer neuron module. This module is similar to the middle layer module, except that here the values for the weight of the links and the bias of the neurons are obtained from the respective RAMs. Also, input-out is an 8-bit input, which is actually the same as the 8-bit output of the previous layer neurons and the third middle layer module.

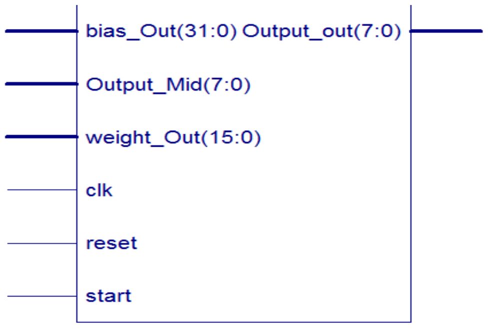


Fig. 11. Output layer neuron module

The output neuron module receives the output of the last layer, which is 8 bits, as input, and finally, by explaining the output and how it is completed, passes the sigmoid function and produces the final output. It should be noted that the meaning of bias values, which are considered as two RAMs, one for the middle layer and one for the output layer, is a series of fixed numbers that have the order of illumination for neurons, and in this project to all Layers are given the same number to save resource usage.

C) RAM related to the weights that are link between the input layer and the middle layer

In this module, the Initialization input signal is used to initialize the memory. These values are stored in memory according to the trained network in Matlab software. The 3-bit input specifies the desired address. For each value, 4 bits are assigned and we have 20 weights (number of inputs). Finally, 80-bit Data output contains weights. Subsequent RAM modules such as RAM corresponding to the weights corresponding to the links between the middle layer and the output layer. The RAM corresponding to the bias corresponding to the middle layer neurons and the RAM corresponding to the bias corresponding to the output layer neurons are similar to the above implementation, except that their output data is different based on the application of the said RAM and it is avoided. The module of this RAM has eight addresses as follows:

```
mem[0]=_80'b011000010011001000100001111110000000000010
0110010010000100011101000110001100000001;
mem[1]=80'b0101001001010010110000010001000100010101000
0101010001000100011001011100110101000;
mem[2]=80'b000110100101001001011001010101010101010100101
00011010100010011000100100001000100001;
mem[3]=80'b010110101001001000101011000100010101100100
0111010001000101010001000000111000100011;
mem[4]=80'b110100100101000001001011000110010101010100
0101100010100101100101010110010101010101;
mem[5]=_80'b001100101101001101011001000100010001001000
1110001100110010111000101001001000010010;
mem[6]=_80'b010100100101001111000000100011001000101010
0010110101010110001010110010011010110001;
mem[7]=_80'b110101010101001100111101000100010001001001
1010001010001110001010110010001001101011;
```

Each 4 bits is a weight number. Nevertheless, in the address line zero (mem (0)) the first four bits are 0110 which shows the number 5. The second four bits are 0001, which shows the number one and so on. When we check the matrix of weights in Matlab, it is observed that all values of this matrix are between zero and one, and this is because the network input vector is normalized to increase the network efficiency, each column is divided by its maximum so that the input numbers in the range are between 0 and 1. Now in the implementation to solve this problem, we multiply the weights by 10 and then put it in RAM. If we see the first 4 weights in Matlab, they are as 0/5461, 0/0901, 0/2619, 0/2162. These numbers are multiplied by 10. The first number with rendering is five, the second number is one, the third number is three, the fourth number is two. These numbers are written in RAM in the address of the first 16 bits zero as 0110 0001 0011 0010.

However, as you can see, these negative numbers RAM due to the large number of negatives. But the question that arises is that each RAM of this address line is 80 bits, so how in the first module the middle layer has a 320-bit weight_ mid input. The answer is that every 4 bits have been converted to 16 bits for ease of writing the module and multiplications, which are multiplied as follows in the first middle module:

- Input_Mid [15: 0], weight_Mid [15: 0]
- Input_Mid [31:16], weight_Mid [31:16]

Consider that the values of the numbers in the next two RAMs are because the 32-bit Mem_weight_Mid3 and Mem_weight_Mid2 have 8 values, each of which has 4 bits, and multiplied by 32 bits. Just like before, these weights are converted to 16 bits for ease of work.

D) Sigmoid stimulation function

Hardware implementation of the Sigmoid Stimulus Function is a very important part of the hardware implementation of neural networks. Because this function includes nonlinear mapping and operators such as power, addition, and division, it will take up many time and hardware resources if implemented directly. Therefore, LUT has been used here to implement it. From a hardware implementation point of view, what should be considered in this method is the size of the ROM unit and the degree of complexity of the time control for LUT addressing. This implementation method proposed by Srdjan and Coric [33] is given in relation to 9 and 10:

$$a_{i(net)} = \frac{1}{1+e^{-net}} \tag{9}$$

$$a_{i(net)} = \frac{255}{1+e^{\frac{-net}{8}}} \tag{10}$$

Where net is a positive integer value. If net is a positive and negative value, the correct sigmoid value is provided by the following formula:

$$F(net) = 255 + F(net)^{*} \tag{11}$$

Where $net^{*} = 2^{N} + net$ and N is the input length .in relation 11, $F(net)^{*}$of relation 10 can be calculated .The above statements show that although this method reduces the number of ROMs involved, it does not have the necessary flexibility. Address generation scheduling becomes very complicated and the processing cycle of the computational unit becomes longer with multiple parallelization. Examining the interface (10), it can be seen that, if the input of the sigmoid function exceeds a certain value, its output should be close to 255, and on the other hand, if the input of the function is less than a certain value, the output should be close to zero. Therefore, in order to increase the efficiency in the hardware implementation of the sigmoid function in this method, the following steps are suggested:

- If the input of the function exceeds 127, its output is considered 255, and similarly, if the input of the function is less than -128, its output is taken as zero.
- If the input is between -128 and 127, the output should be calculated from the following equation:

$$a_{i(net)} = \frac{255}{1+e^{\frac{-net}{24}}} \tag{12}$$

These values are stored in LUT. When working on a neural network, the addition of 128 to the input can simply generate the desired address in the LUT. Studies show that this method is suitable for the middle and output layers. In addition, networks that use this type of implementation have similar performance to networks that calculate the sigmoid function directly with floating point data.

E) Booth Multiplication

In each neuron, we need to implement multiplication to perform computational operations. For this purpose, here is the Booth multiplication algorithm, which is used in the form of shifts and additions. The basis of the algorithm is based on the fact that the strings 0 in the multiplication factor do not need to be added, but only the movement (shift) requires .string 1 in multiplier from bit 2k to bit 2m can be considered equal to 2k+1 - 2m. Inputs and weights are both considered 4-bit and the result is 16-bit [8].

F) Neuron and layer control unit

The control unit for neurons and layers actually has two important functions.

- Controlling neurons and links and allowing modules to start and stop to reach the input to the output.
- Control layers by multiplexing layers. This unit controls the start of the computational operation of middle layer neurons by 8-output signals start _Mid1 to start_ Mid8.

The Initialization output signal activates the corresponding signal in the memory for initialization. 8-output signals HZ_ bus1 to HZ_ bus8 Manage the order in which the output bus is impeded by middle-layer neurons. The Address_ Mid and Address _ Out address lines are responsible for generating the appropriate address of the memory containing the corresponding weights of the links. The end_ work signal indicates the end of network work. However, the module of the control unit does not change and as before, it sends the start command to the middle layer modules and gives them the memory address in the appropriate clock. The boot multiplier module and the look up table are also unchanged. Finally, it should be noted that the output of this implementation might differ from the output of the content for the following reasons:

- The sigmoid function implemented by the look up table may not contain all the values or may correct the values.
- The positive and negative amount of weights cannot be applied to exactly any number.
- Weights are rendered.
- Giving all the weights to RAM is a very long job so it is placed in some lines of duplicate numbers.

In the network implemented in the content, the bias values are set by default by the content itself, which may be different from the bias values given in the implementation. Fig. 12 is a general schematic of the implementation with 20 inputs and 4 outputs.

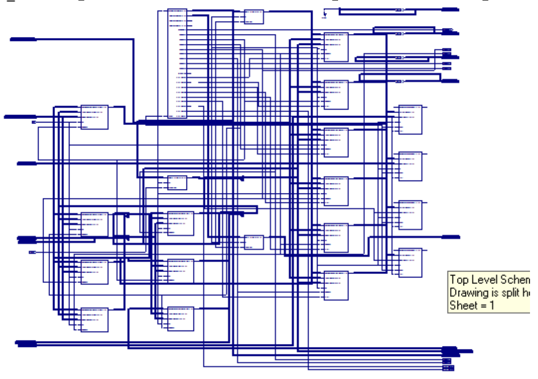


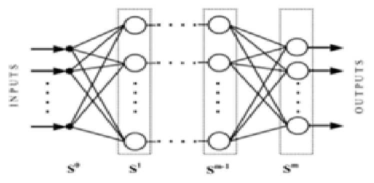
Fig. 12. General schematic of neural network implementation with 20 inputs and 4 outputs.

5. Implementation of 5-layer neural network by layers multiplex

In multi-layer implementation, the implementation modules are the same as before, except that the middle three layers become one layer. In fact, the three layers become multiplex. The multiplexer task is to select the output of the middle layer neurons in the look up table. The output of middle layer neurons is stored in a look up table .The method of multiplexing the layers is such that at each moment and in one clock, one layer is used according to the multiplex address. The results of this implementation show that the proposed method with multiplexing neural network layers reduces resource consumption by about 30 to 40%. The multiplexer selects the output of the middle layer neurons. In this implementation, we have three hidden layers, so the results of the middle layer neurons are stored three times, and the multiplexer selects the results of the third layer neurons and outputs for the neurons of the output layer. The optimizations performed for the implementation of MLP 20-8-8-8-4 include the following:

- Binary the value of the sigmoid function between zero and one in this optimization, instead of using 2 bits for the values 1-and 1 +, one bit is used for zero and one for the value of the function [34].
- Use shifts instead of direct multiplication. In this case, instead of multiplying the inputs are directly used by the weights of a collector and shifter [34].It is used which is much cheaper and requires less resources. Generates open output in one bit (zero and one) [34].
- Use a layer control to multiplex layers [35].

In this case, the number of hidden layers is simultaneously connected to a multiplexer and the selected layer is selected and used at high speed. Multiplexer implementation is done in the control unit. To control the layers, this unit receives a variable called the number of layers. In fact, this input specifies the number of hidden layers, which can be changed from one to three. Based on the number of hidden layers, the control unit operation must be repeated, if we have 3 hidden layers, to use multiplexing the layers, the control unit operation must be repeated exactly 3 times, in which case one layer instead of our 3 hardware layers. We have three schedules. In the new layer control module and neurons, the number of layers is also applied as an input to this module. Finally, a module called the task control unit is responsible for controlling all modules and multiplexing the middle layers. Fig. 13 shows the structure of a perceptron multilayer network using a multiplexer. In this method, only one layer with its number of neurons is implemented and only the values of its neurons are moved by multiplexer. For example, suppose we have only one layer with 8 neurons according to Fig. 3 of the reference [35]. First, the inputs in the weights of the first layer are calculated by the timing of the control block and stored in neurons called s0, and then by the control block and multiplex, the value of these neurons is multiplied by the corresponding weights and stored in neurons s1. Continue until all layers are calculated. When a layer is completed and its values are calculated, it is stored in a look up table connected to the Layers Control, and the next layer calculations begin. In this case, for example, instead of implementation a 20-8-8-8-4 network, a 20-8-4 network is implemented, saving overall FPGA resources [35]. Use the Look up table to implement the sigmoid function instead of using the direct implementation of these functions.



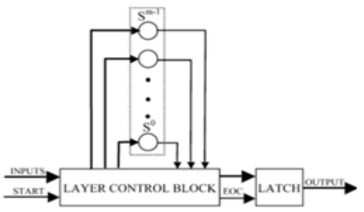


Fig. 13. Conventional MLP network structure and MLP network structure with hidden layer multiplex [35]

Fig. 14 shows the input layer with 20 neurons and the three hidden layers as a multiplex with eight neurons and the output layer with four neurons. The links between the input layer and the middle layer, along with the output neurons, also participate in this parallelization, and the results of the middle layer calculations are returned to the units by diffusion. Therefore, all computational conditions for output neurons can be examined at one time.

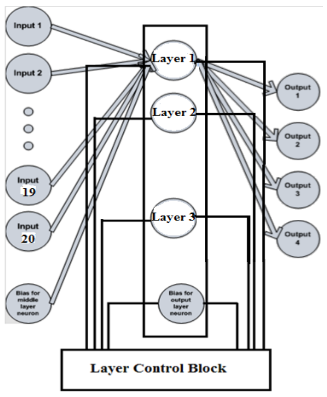


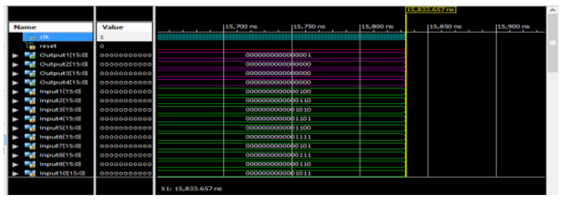
Fig. 14. Structure of 20-8-8-8-4 network implementation with parallel link and hidden layer neurons and multiplexes.

6. Synthesis and simulation

The synthesis was performed by the XST tool included in the ISE bundle, which can be used for both HDL-based or schematic design processes, using the XC3S50005fg900Spartan3 hardware. The ISE software package simulator, Isim, performs all simulations. Twenty inputs have values that are actually one of the rows of test matrix values in Matlab software. As explained earlier, the result is 4 bits, each bit of which represents a class. For example, 0010 network output means the first output is zero, the second output is one, the third output and fourth are zero. When the reset signal is zero, the network starts working. In Fig. 15 (a), the network simulation with the implementation without layer, multiplexing is examined. In this simulation, the inputs and outputs are 16 bits. It can be seen that a normal network takes 15 nanoseconds to complete the output values. Fig. 15 (b) shows the simulation using a multiplex of layers. If the network is optimized with a multiplex of layers, it takes three nanoseconds to complete the output and remain constant, and 12 nanoseconds converge faster to the answer.

7. Evaluation implementation network

The following is a summary of mapping, routing, and implementation reports on 20-8-8-8-4 networks for the Spartan 3 XC3S5000 5fg900. Table 5 compares the two types of implementations directly without layer multiplexing and with layer multiplexing.


```
Name              Value          15,833.657 ns
                                 15,700 ns  15,750 ns  15,800 ns  15,850 ns  15,900 ns
clk               1
reset             0
Output1[15:0]     00000000000    0000000000000001
Output2[15:0]     00000000000    0000000000000000
Output3[15:0]     00000000000    0000000000000000
Output4[15:0]     00000000000    0000000000000000
Input1[15:0]      00000000000    0000000000000100
Input2[15:0]      00000000000    0000000000000110
Input3[15:0]      00000000000    0000000000001010
Input4[15:0]      00000000000    0000000000001101
Input5[15:0]      00000000000    0000000000001100
Input6[15:0]      00000000000    0000000000001111
Input7[15:0]      00000000000    0000000000000101
Input8[15:0]      00000000000    0000000000000111
Input9[15:0]      00000000000    0000000000000110
Input10[15:0]     00000000000    0000000000001011
X1: 15,833.657 ns
```

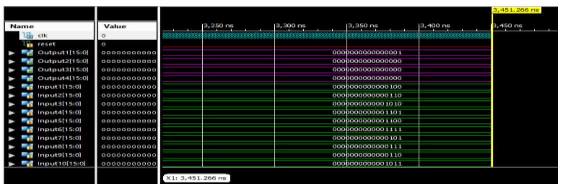

```
Name              Value          3,451.266 ns
                                 3,250 ns  3,300 ns  3,350 ns  3,400 ns  3,450 ns
clk               0
reset             0
Output1[15:0]     00000000000    0000000000000001
Output2[15:0]     00000000000    0000000000000000
Output3[15:0]     00000000000    0000000000000000
Output4[15:0]     00000000000    0000000000000000
Input1[15:0]      00000000000    0000000000000100
Input2[15:0]      00000000000    0000000000000110
Input3[15:0]      00000000000    0000000000001010
Input4[15:0]      00000000000    0000000000001101
Input5[15:0]      00000000000    0000000000001100
Input6[15:0]      00000000000    0000000000001111
Input7[15:0]      00000000000    0000000000000101
Input8[15:0]      00000000000    0000000000000111
Input9[15:0]      00000000000    0000000000000110
Input10[15:0]     00000000000    0000000000001011
X1: 3,451.266 ns
```

Fig. 15. Results of network implementation simulation 20-8-8-8-4 to detect the state of the face without multiplexing layers and results of network implementation simulation 20-8-8-8-4 to diagnose conditions Face by multiplexing layers

Table.5.
Comparison of used resources of two types of implementation performed directly without multiplex layers and by multiplex layers on Spartan 3 XC3S50005fg900

Module Name	***Optimized Neural Network20-8-8-8-4 (The proposed work)***			***Ordinary Neural Network 20-8-8-8-4(The previous work)***			***Optimized percent***
Logic Utilization	Used	Available	Utilization	Used	Available	Utilization	
slice Flip Flop	6374	6374	9%	13549	66560	20	11
4 Input LUTs	21738	21738	32%	39647	66560	59.5	27
occupied Slices	11978	11978	35%	23229	33280	70	35
4 Input LUTs	21963	21963	32%	43842	66560	66	34
Bonded IOBs	355	355	56%	360	633	57	1

As can be seen from Table 5, the use of layer multiplexing prevents a lot of resource wastage. In the last column, for the total resources of the hardware platform look-up table, it is optimized by 34%. Due to this issue, more complex functions such as hyperbolic tangent can also be used for the activation function, which results in results that are more accurate. In total, the optimizations performed in this paper uses 62408 FPGA resources, which shows 53.73%, decrease in resources consumptions compared to the direct implementation (120627 resources).

8. Conclusion

The formation of FPGA in ANN with a large number of neurons is a challenging task. Sufficient accuracy is one of the most important choices when implementation ANNs on FPGAs. Sufficient accuracy is used to measure the true capabilities of ANN against the cost of its implementation. High accuracy creates fewer error packages in the final implementation. While less accuracy leads to simpler design, more speed and reduced area required and less energy consumption. On the other hand, using LUTs reduces the need for resources and improves speed. In addition, implementation LUT does not require external RAM because internal memory is sufficient to implement induction functions. In this paper, a method for detecting facial modes on FPGA using a 5-layer neural network as a layered multiplex is presented. The results show that the choice of the number of neurons and layers is also important. If we want to implement a neural network with deep learning by a large number of layers using MLP on FPGA, we have a very difficult task ahead of us. For example, if we have a network with 100 hidden layers, it is almost impossible to implement on a low cost FPGA. However, with the method presented in this paper, it can be implemented. On the other hand, if you use FPGA with more resources, you can implement more complex functions such as hyperbolic tangent, which greatly helps the accuracy of the output.

References

[1] M. Beringer, F. Spohn, A. Hildebrandt, J. Wacker, and G. Recio, “Reliability and validity of machine vision for the assessment of facial expressions,” Cogn. Syst. Res., vol. 56, pp. 119–132, 2019.

[2] F. Yang and M. Paindavoine, “Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification,” IEEE Trans. Neural Networks, vol. 14, no. 5, pp. 1162–1175, 2003.

[3] H. Tsutsui, H. Nakamura, R. Hashimoto, H. Okuhata, and T. Onoye, “An fpga implementation of real-time retinex video image enhancement,” in 2010 World Automation Congress, 2010, pp. 1–6.

[4] Y. Lee and S.-B. Ko, “FPGA implementation of a face detector using neural networks,” in 2006 Canadian Conference on Electrical and Computer Engineering, 2006, pp. 1914–1917.

[5] S. Wang, Q. Gan, and Q. Ji, “Expression-assisted facial action unit recognition under incomplete au annotation,” Pattern Recognit., vol. 61, pp. 78–91, 2017.

[6] A. Dinu, M. N. Cirstea, and S. E. Cirstea, “Direct neural-network hardware-implementation algorithm,” IEEE Trans. Ind. Electron., vol. 57, no. 5, pp. 1845–1848, 2009.

[7] A. Dinu and M. Cirstea, “A digital neural network FPGA direct hardware implementation algorithm,” in 2007 IEEE International Symposium on Industrial Electronics, 2007, pp. 2307–2312.

[8] D. Govekar and A. Amonkar, “Design and implementation of high speed modified booth multiplier using hybrid adder,” in 2017 International Conference on Computing Methodologies and Communication (ICCMC), 2017, pp. 138–143.

[9] X. Peng, L. Zhang, Z. Yi, and K. K. Tan, “Learning locality-constrained collaborative representation for robust face recognition,” Pattern Recognit., vol. 47, no. 9, pp. 2794–2806, 2014.

[10] Y. Pan, J. L. Trahan, and R. Vaidyanathan, “A scalable and efficient algorithm for computing the city block distance transform on reconfigurable meshes,” Comput. J., vol. 40, no. 7, pp. 435–440, 1997.

[11] J. Murphy and R. Cook, “Revealing the mechanisms of human face perception using dynamic apertures,” Cognition, vol. 169, pp. 25–35, 2017.

[12] M. G. Calvo, A. Fernández-Mart\'\in, and L. Nummenmaa, “Facial expression recognition in peripheral versus central vision: Role of the eyes and the mouth,” Psychol. Res., vol. 78, no. 2, pp. 180–195, 2014.

[13] B. Lahasan, S. L. Lutfi, I. Venkat, M. A. Al-Betar, and R. San-Segundo, “Optimized symmetric partial facegraphs for face recognition in adverse conditions,” Inf. Sci. (Ny)., vol. 429, pp. 194–214, 2018.

[14] Y. Duan, J. Lu, J. Feng, and J. Zhou, “Topology preserving structural matching for automatic partial face recognition,” IEEE Trans. Inf. Forensics Secur., vol. 13, no. 7, pp. 1823–1837, 2018.

[15] V. Domen and B. Simon, “Implementation of Massive Artificial Neural Networks with Field-programmable Gate Arrays,” IFAC Proc. Vol., vol. 45, no. 4, pp. 133–138, 2012.

[16] S. Murugan, K. P. Lakshmi, J. Sundar, and K. MathiVathani, “Design and Implementation of Multilayer Perceptron with On-chip Learning in Virtex-E,” AASRI Procedia, vol. 6, pp. 82–88, 2014.

[17] V. Tiwari and N. Khare, “Hardware implementation of neural network with Sigmoidal activation functions using CORDIC,” Microprocess. Microsyst., vol. 39, no. 6, pp. 373–381, 2015.

[18] R. McCready, “Real-time face detection on a configurable hardware system,” in International Workshop on Field Programmable Logic and Applications, 2000, pp. 157–162.

[19] M. S. Sadri et al., “An FPGA based fast face detector,” 2004.

[20] J. Cho, S. Mirzaei, J. Oberg, and R. Kastner, “Fpga-based face detection system using haar classifiers,” in Proceedings of

the ACM/SIGDA international symposium on Field programmable gate arrays, 2009, pp. 103–112.

[21] M. Hiromoto, H. Sugano, and R. Miyamoto, "Partially parallel architecture for adaboost-based detection with haar-like features," IEEE Trans. Circuits Syst. Video Technol., vol. 19, no. 1, pp. 41–52, 2008.

[22] P. Baldi and P. Sadowski, "Learning in the machine: Recirculation is random backpropagation," Neural Networks, vol. 108, pp. 479–494, 2018.

[23] W. Zhou and J. Jia, "A learning framework for shape retrieval based on multilayer perceptrons," Pattern Recognit. Lett., vol. 117, pp. 119–130, 2019.

[24] H. K. Ghritlahre and R. K. Prasad, "Exergetic performance prediction of solar air heater using MLP, GRNN and RBF models of artificial neural network technique," J. Environ. Manage., vol. 223, pp. 566–575, 2018.

[25] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar, and I. Matthews, "The extended cohn-kanade dataset (ck+): A complete dataset for action unit and emotion-specified expression," in 2010 ieee computer society conference on computer vision and pattern recognition-workshops, 2010, pp. 94–101.

[26] Y. Li, M. J. Zuo, Y. Chen, and K. Feng, "An enhanced morphology gradient product filter for bearing fault detection," Mech. Syst. Signal Process., vol. 109, pp. 166–184, 2018.

[27] Z. Jiang, D. Zhang, and G. Lu, "Radial artery pulse waveform analysis based on curve fitting using discrete Fourier series," Comput. Methods Programs Biomed., vol. 174, pp. 25–31, 2019.

[28] A. Hernandez-Matamoros, A. Bonarini, E. Escamilla-Hernandez, M. Nakano-Miyatake, and H. Perez-Meana, "Facial expression recognition with automatic segmentation of face regions using a fuzzy based classification approach," Knowledge-Based Syst., vol. 110, pp. 1–14, 2016.

[29] A. T. Lopes, E. de Aguiar, A. F. De Souza, and T. Oliveira-Santos, "Facial expression recognition with convolutional neural networks: coping with few data and the training sample order," Pattern Recognit., vol. 61, pp. 610–628, 2017.

[30] C. MageshKumar, R. Thiyagarajan, S. P. Natarajan, S. Arulselvi, and G. Sainarayanan, "Gabor features and LDA based face recognition with ANN classifier," in 2011 International Conference on Emerging Trends in Electrical and Computer Technology, 2011, pp. 831–836.

[31] E. M. Bouhabba, A. A. Shafie, and R. Akmeliawati, "Support vector machine for face emotion detection on real time basis," in 2011 4th International Conference on Mechatronics (ICOM), 2011, pp. 1–6.

[32] R. Saabni, "Facial expression recognition using multi Radial Bases Function Networks and 2-D Gabor filters," in 2015 Fifth International Conference on Digital Information Processing and Communications (ICDIPC), 2015, pp. 225–230.

[33] S. Coric, M. Leeser, E. Miller, and M. Trepanier, "Parallel-beam backprojection: an FPGA implementation optimized for medical imaging," in Proceedings of the 2002 ACM/SIGDA tenth international symposium on Field-programmable gate arrays, 2002, pp. 217–226.

[34] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, "FP-BNN: Binarized neural network on FPGA," Neurocomputing, vol. 275, pp. 1072–1086, 2018.

[35] Z. Lin, Y. Dong, Y. Li, and T. Watanabe, "A hybrid architecture for efficient FPGA-based implementation of multilayer neural network," in 2010 IEEE Asia pacific conference on circuits and systems, 2010, pp. 616–619.