

High Accuracy Fingerprint Recognition on FPGA Using a Dual Path Lightweight CNN

Abdolah Amirany

University of Florida

Belal JAHANNIA

University of Florida

HAMED Dalir

`hamed.dalir@ufl.edu`

University of Florida

Research Article

Keywords: Fingerprint Recognition, Convolutional Neural Network, FPGA Implementation, Biometric Identification, Hardware Acceleration

Posted Date: October 23rd, 2026

DOI: <https://doi.org/10.21203/rs.3.rs-11086184/v1>

License:   This work is licensed under a Creative Commons Attribution 4.0 International License.

[Read Full License](#)

Additional Declarations: No competing interests reported.

High Accuracy Fingerprint Recognition on FPGA Using a Dual Path Lightweight CNN

Abdollah Amirany, Belal Jahannia, Hamed Dalir*

Department of Electrical and Computer Engineering, University of Florida
Gainesville, FL 32611, USA

Abstract

This paper addresses the increasing demand for reliable identity verification in various applications and highlights the importance of biometric features such as fingerprints due to the limitations and potential distrust of identification cards. Given the strong performance of neural network-based methods, this work focuses on convolutional neural networks (CNNs) for fingerprint image classification and identification, with an emphasis on efficient hardware implementation on field-programmable gate arrays (FPGAs). A lightweight CNN architecture is proposed that achieves higher accuracy compared to prior approaches while operating directly on raw images without pre-processing. The proposed model is evaluated under two data augmentation scenarios, achieving accuracies of 70.3% and 63.1%, respectively, outperforming previous studies. The network response time per sample is 77.9 ms on GPU and requires approximately 483 KB of memory for parameter storage. The architecture is implemented on a Xilinx Virtex-7 FPGA using VHDL, utilizing 70% of available resources. The FPGA implementation consumes 28.72 W of power and operates at a maximum frequency of 108.6 MHz.

Keywords: Fingerprint Recognition, Convolutional Neural Network, FPGA Implementation, Biometric Identification, Hardware Acceleration

1 Introduction

Today, many activities and accesses require the identification of individuals. Relying solely on passwords and identification cards is insufficient to ascertain someone's identity; thus, leveraging biometric characteristics such as fingerprints, facial features, iris patterns, and voice can provide accurate authentication [1,2]. Biometric characteristics are unique to individuals and can be used as reliable means of recognition. A few important and commonly researched biometric features include fingerprint, retina, iris patterns, face, facial thermography, hand geometry, and hand veins [3,4].

Fingerprints are widely used for their ease of collection, uniqueness, and stability over time. Figure 1 demonstrates a fingerprint that has been sampled over several years. As this figure shows, the fingerprint's overall shape has remained the same and it has maintained its stability over time [3]. Fingerprint recognition systems have become ubiquitous. They can be seen in various fields such as forensic medicine [5,6], IoT device authentication and expansion [7,8], mobile phone security and payments, border crossing and national ID [3].

Nowadays, two main methods, machine learning and image processing, are used to identify and classify images [8,9]. In recent works, sometimes these two methods are used in combination [10]. Image processing algorithms are used less often than learning-based algorithms such as

*Corresponding author: Hamed Dalir, Email: hamed.dalir@ufl.edu

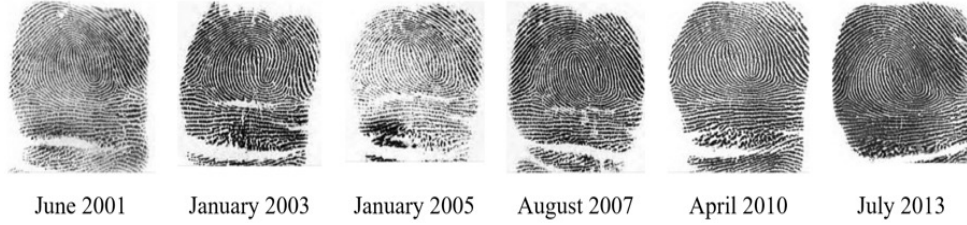


Figure 1: An example of a fingerprint over the years [3]

neural networks. This is due to the complexity and different mathematical calculations involved in image processing algorithms along with their lower flexibility and generalization. The extensive mathematical calculations in image processing algorithms also increase the processing costs [11]. Consequently, more robust hardware is required for image processing, limiting its effectiveness in large databases and reducing its flexibility compared to machine learning algorithms.

Convolutional neural networks (CNNs) are commonly used for classification and identity recognition tasks [10]. However, they pose challenges due to their significant memory requirements for storing learning weights and parameters during training. This challenge has become one of the most critical issues in neural network development [9], as it necessitates powerful hardware to handle the memory demands. Consequently, this leads to increased financial costs and higher power consumption of the hardware.

To address the memory challenge in CNNs, researchers have designed and tested lighter models and architectures that offer improved performance while mitigating the memory problem. Despite advancements, CNNs remain computationally intensive, executing billions of operations during inference [12]. Typically, CPUs handle network processing, but due to these challenges, a combination of GPUs, ASICs, and FPGAs is used to accelerate CNN inference [13, 14].

GPUs are highly effective in performing floating-point matrix operations with high throughput and possess a significant memory bandwidth. Therefore, GPUs are widely used in the training and implementation of CNNs. However, their considerable power consumption makes them unsuitable for power-restricted systems. Conversely, ASIC and FPGA hardware accelerators, although having limitations in memory [15], bandwidth, and computing resources compared to GPU-based accelerators, offer low power consumption and efficient processing speed. ASIC-based accelerators are made for specific applications and have lower flexibility compared to FPGAs. They are also costly and time-consuming to develop. Therefore, FPGA-based accelerators are more commonly used due to their good performance, high energy efficiency compared to GPUs, and superior flexibility compared to ASICs [15, 16].

In this paper, a new light and accurate model for fingerprint matching based on deep CNN is proposed and implemented on FPGA. Contributions of this paper are as follows:

- The method does not use pre-processing on the images, simplifying the approach.
- The number of network parameters is reduced, addressing the memory challenge.
- Actual dimensions of images are used, leading to higher accuracy.
- Data augmentation is employed to increase data and mitigate overfitting.
- Performance comparison of different models is performed using cross-validation for fair assessment.

- Architecture-level design and optimization of each network block are carried out for FPGA implementation.
- The entire model is implemented on FPGA.

The rest of the paper is organized as follows: Section 2 reviews the background of the study. In Section 3, previous works in the field of fingerprint matching using CNNs and FPGA implementation of these networks are investigated. The proposed model and its hardware implementation are presented in Section 4. Section 5 presents the results of the model verification and the performance of the hardware implementation. Finally, Section 6 concludes the paper.

2 Prerequisites

2.1 Convolutional Neural Network Structures

CNNs are one of the most important and widely used algorithms in deep learning. They are extensively used in various computer vision applications such as image processing, speech processing, and text processing [17]. These networks are composed of several different layers, including the convolutional layer, pooling layer, and fully connected layer [18].

The convolutional layers act as filters, performing convolution operations on input images. The number and dimensions of filters, zero-padding for the input image, and the strides of these layers are known as hyperparameters. Mathematically, the extracted feature $S(i, j)$ is defined by Eq. 1 [19].

$$S(i, j) = (H * V)(i, j) = \sum_a \sum_b V(i + a, j + b) H(a, b) \quad (1)$$

In Eq. 1, the $*$ operator represents the two-dimensional convolution operation and $V(a, b)$ and $H(a, b)$ are input image and convolution filter, respectively. After the convolution operation, the final value is added with bias [19].

Pooling layers play a crucial role in CNNs by enhancing the training process. They reduce the number of features and parameters extracted in the convolutional layer, thereby decreasing computational costs and addressing issues like invariance to details and distortions in the input [9, 20]. The most common types of pooling layers are maximum pooling and average pooling [20].

The fully connected layer is usually positioned towards the end of the CNN architecture. It is characterized by a high number of parameters [13]. The fully connected function is described by Eq. 2, where x represents the input of the layer, w represents the weight matrix, b is the bias, and y is the output [21].

$$y_{i'} = \sum_i w_{ii'} x_i + b_{i'} \quad (2)$$

2.2 Hardware Implementations of CNN

GPUs have attracted attention in the hardware implementation of CNNs due to their faster processing speeds compared to CPUs. They are now preferred for neural network computations due to their speed advantages. However, the high-power consumption of GPUs and other processors limits their optimal daily use. Moreover, as neural networks become more integrated into IoT applications and smart devices, the physical size of the hardware, particularly GPUs with cooling systems, becomes a significant factor [16].

The potential of FPGAs and ASICs in neural network implementation has been explored due to their high-speed processing capabilities. While ASICs are efficient, they are limited by inflexibility and high production costs. In contrast, FPGAs provide more flexibility without such cost issues, making them a focus for neural network implementation.

A crucial aspect in implementing neural networks on FPGAs is the choice of data formats for inputs, weights, and activation functions, which significantly impacts network accuracy. While neural networks typically use real numbers, employing floating-point formats on FPGAs ensures high precision but requires more resources. Alternatively, fixed-point arithmetic uses a constant number of bits for number representation but may sacrifice computational precision. This trade-off between accuracy and resource efficiency is a crucial consideration in neural network research and development [22].

3 Previous Works

3.1 Fingerprint Recognition Using Neural Networks

Recent research in fingerprint recognition has focused on three primary tasks:

1. Classification of Dominant Patterns in Fingerprints [9, 23, 24]: This task involves categorizing fingerprints into classes, such as central loop, right loop, left loop, and arch, using neural networks for classification.
2. Extraction of Additional Features from Fingerprints [25]: This task involves identifying features like gender (male or female), hand type (right or left), and finger number from fingerprint images.
3. Identity Recognition Using Individual Fingerprints [7, 26–28]: This task concentrates on recognizing and verifying identities based on individual fingerprints.

Each of these approaches has its own advantages and disadvantages. The first approach is limited to identity recognition as it focuses on separating fingerprint classes. The second approach offers more specific identification but lacks comprehensiveness. The third approach provides accurate and practical results for individual identification but may require more detailed feature extraction. For instance, authors in [23] used CNN for classifying fingerprints into four distinct classes (central loop, left loop, right loop, and whorl) on the IAFIS database and achieved high accuracy compared to benchmark fingerprint recognition models. The proposed method in [23] included reducing image dimensions and employing data augmentation techniques, which expedited training time without compromising accuracy. Similarly, [23] defined five classes and utilized CNN-based model for their proposed fingerprint recognition model. They reported high accuracy and reduced training time through image dimension alteration.

However, these methods may show varied effectiveness across different databases and configurations. For example, in [24] examination of three pre-trained CNNs (AlexNet, GoogleNet, and ResNet) across different class configurations and databases showed varying performance, suggesting that these methods might not be universally applicable [24]. In [7] a new model proposed that known as Finger ConvNet, this model is based on the VGG-16 architecture. In [27] K-nearest neighbors (KNN) algorithm used for fingerprint identification, the proposed method in [27] achieved high accuracies but required extensive preprocessing and data augmentation to mitigate overfitting. In [28] authors focused on improving performance by addressing image quality through preprocessing and address the issues of overfitting beyond certain epochs. In [25] with innovative integration

of Winner filters, multi-level histogram techniques, and CNNs a high accuracy is achieved but required a complex architecture and extensive data division. All the previous works in field of fingerprint recognition using NN highlight the necessities of trade-off between complexity, data processing, performance, and accuracy.

3.2 Implementation of Neural Network on FPGA

The advancements in implementing neural networks on FPGA demonstrate significant advantages along with some limitations in different studies. Kalms and colleagues focused on implementing the MobileNet network on FPGA using the High-Level Synthesis tool and the HiFlipVX library [13]. This implementation showed that weight and bias values could be efficiently managed within the FPGA, reducing delays in intermediate layers and achieving higher speed. However, they faced constraints like synthesizable structures being limited to those with more than one element, but they effectively utilized Xilinx libraries for efficient resource execution. This approach streamlined operations and optimized performance but also revealed the complexities and limitations inherent in using specific tools and libraries in FPGA environments [13].

Nguyen and colleagues' implementation of the YOLO on FPGA for real-time object detection highlighted both high performance and energy efficiency [29]. By quantizing the network and optimizing the data path, they reduced power consumption significantly by eliminating off-chip memory access. This implementation was fully parameterized, facilitating network expansion. However, while achieving impressive operational power and energy efficiency on the VC707 FPGA series, there was a slight reduction in accuracy compared to other research, highlighting the trade-off between energy efficiency and accuracy in FPGA implementations. Westby and colleagues' work on implementing a perceptron network for recognizing handwritten English digits on an Ultrascale FPGA showcased the advantages of FPGA in terms of resource usage, power consumption, and network latency [30]. By employing an eight-stage pipeline method and using Verilog, they managed to create a compact and efficient network with 93.2% accuracy. Their approach significantly reduced resource usage and power consumption, highlighting the FPGA's ability to host complex neural networks efficiently. However, this study also reflects the intricate nature of hardware-specific implementations and the necessity of careful design choices to balance efficiency, accuracy, and resource constraints [30].

4 Proposed Model and Its Implementation

4.1 Proposed Model Architecture

As mentioned in [24], the AlexNet network has shown a better capability in fingerprint recognition compared to other networks. Furthermore, since the ultimate goal is to implement the network on an FPGA, the intended network should have a low number of parameters and layers to facilitate easier hardware implementation. Among the networks mentioned in the literature, GoogleNet and ResNet have complex structures and a large number of layers. Therefore, considering the relatively better accuracy of AlexNet and its simpler structure, this network is chosen as the base of the network structure. The AlexNet network has 96, 256, and 384 filters (in three layers) in the convolutional layers, as well as 4096 neurons in the fully connected layers. This structure results in a very large number of parameters.

To refine and improve the model, efforts were made to reduce the number of convolutional filters in the layers. Additionally, the stride of the convolutional steps was adjusted, and a pooling layer was added after each convolution operation. This approach aimed to have convolutions traverse

the input image with smaller dimensions and extract more features. The inclusion of pooling layers helped reduce the image dimensions at the next stage, thus decreasing computations and parameters.

For the fully connected layer, the number of neurons was reduced, and an additional fully connected layer was added to compensate for the network’s accuracy and performance. To address overfitting during training, two dropout layers were used after each maximum pooling layer. Therefore, by implementing these changes, the AlexNet model has been improved.

As mentioned in [7], a network named Finger ConvNet was introduced, which was designed and implemented by modeling the VGG-16 network. This model has suitable accuracy and performance, but it has a large number of parameters. To optimize it for hardware implementation, changes in its structure are needed. Similar changes to those made to improve the AlexNet model were applied to reduce parameters and make the network lightweight. At this stage, two improved structures have been obtained: one based on AlexNet and the other derived from Finger ConvNet.

Batch normalization layers can also be utilized. This layer is used in modern CNNs and has a regularizing effect, improving convergence speed and facilitating better training in deep networks. Features are standardized by estimating the mean and variance across channels, and then transformed using a function with learnable parameters γ and β [31, 32]. Equation 3 describes this operation.

$$\text{BN}(x) = \gamma \frac{x - E(x)}{\sqrt{\text{Var}(x) + \epsilon}} + \beta \quad (3)$$

In Eq. 3, $E(x)$ and $\text{Var}(x)$ are the estimated mean and variance respectively, both estimated mean and variance have computed from the current batch during training. During inference, the averages of the values $E(x)$ and $\text{Var}(x)$ during training are used [31, 32].

According to experimental results, placing batch normalization layer before the activation function led to better network performance. Performing this normalization before applying nonlinear activation functions yields better results because, after nonlinear activations, the mean, variance, and other statistical parameters no longer change linearly. In this model, the Relu activation function was used due to its higher speed and easier implementation. Consequently, by placing the batch normalization layer before the activation function, we will achieve better network performance.

Considering these optimizations, the proposed model in this paper is shown in Figure 2. This architecture is a combination of two separate paths. The Conv layers in Figure 2 consist of a convolutional layer, a batch normalization layer, and a ReLU activation function.

The dimensions of the convolution filters used should be proportional to the dimensions of the images. Due to the large dimensions of the input images, convolution with a 5×5 filter is used. The dimensions of the filters are proportional to the number of parameters. Due to the high importance of convolution layers for feature extraction, an attempt has been made to select suitable and optimized dimensions. In Figure 2, the dimensions of the processed images at different layers are shown for a 300×300 input. The various layers sequentially reduce the dimensions of the input image. As observed, the smallest image is at the final convolutional layer. The smallest image for the final convolutional layer has dimensions of 11×11 . Despite the filter size being 5×5 , it still provides full coverage for 11×11 images and is suitable.

In the left path, after each convolutional layer, there is a max-pooling layer. Additionally, the number of layers is fewer compared to the right path. However, in the other path, two convolutional layers are stacked, followed by a max-pooling layer. Due to the fewer filters and layers in the left path, it extracts fewer features and captures fewer details. Conversely, the right path has more filters and parameters, allowing it to cover more details. The right path would focus more on

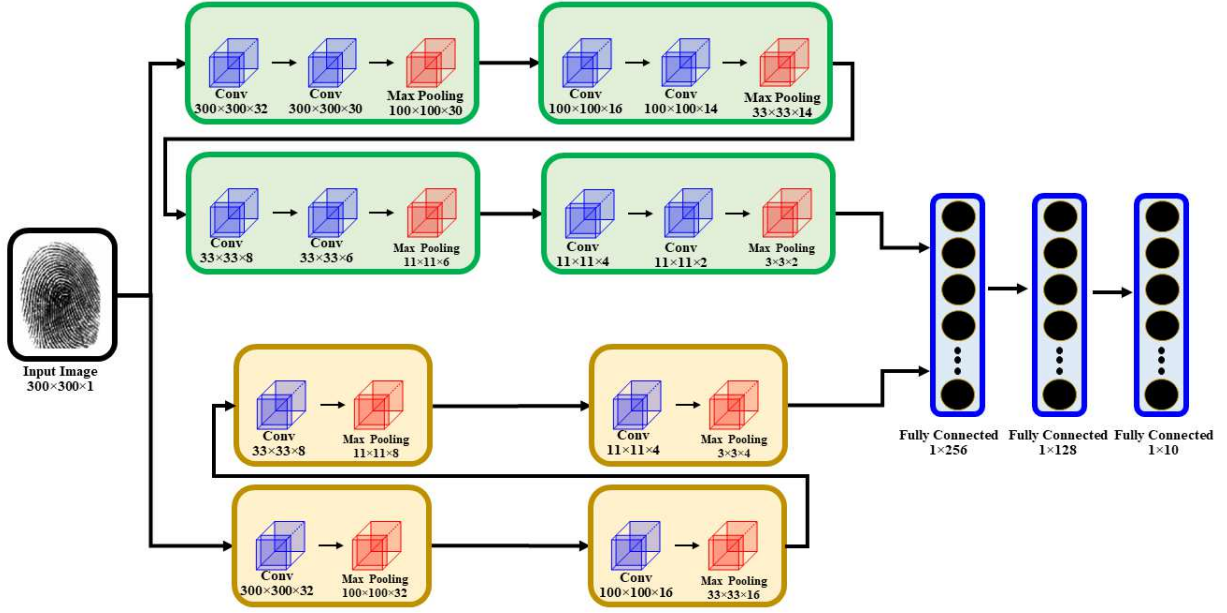


Figure 2: Structure and architecture of the final model

smaller detailed fingerprint features, while the left path would extract larger-scale features from each sample. Thus, at the end of each path, both detailed features and overall fingerprint structures are obtained. At the end, the outputs from both paths are concatenated and fed into three fully connected layers. The initial two layers have 256 and 128 neurons respectively, and the final layer's neuron count depends on the number of classification classes. To prevent overfitting due to the network's large structure, dropout layers are utilized. Each dropout layer is positioned at the end of the paths before the fully connected layer.

One of the crucial reasons for employing two convolutional structures in parallel is the simultaneous utilization of two networks. When using two networks with separate architectures, each of them extracts a unique and specific set of features. Consequently, the network resulting from the combination of these two structures will benefit from high quality features.

4.2 FPGA Implementation of the Proposed Model

A register-level design using the VHDL hardware description language was utilized for the FPGA implementation of the proposed model. This method offers advantages over methods like high-level synthesis (HLS) through detailed computation design and parallelization techniques, which result in reduced power consumption and improved resource efficiency [14].

The structure of hardware implementation closely resembles those designed in software using the Python language. The most crucial aspect of hardware implementation for a neural network is the accurate and detailed understanding of computational operations. In neural networks, due to the presence of numerous computational layers and training parameters, a significant number of memory registers must be used to store the results from various layers. Additionally, all training parameters, input images, and other relevant data must be stored in memory.

To accommodate the storage of the aforementioned items, the dimensions of input images have been reduced. Moreover, the existing block RAMs within the FPGA have been utilized due to their

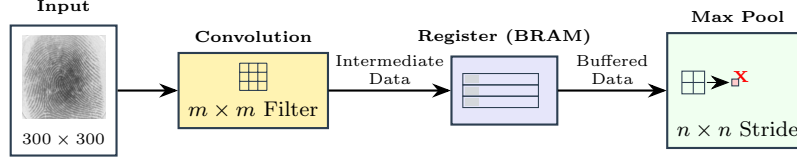


Figure 3: An example of how memory is placed between layers

high speed and easier access compared to external DRAM.

Neural network layers are sequential in structure, yet the operations within these layers can be implemented and executed in parallel. This inherent parallelism significantly increases computational speed, demonstrating the advantages of executing neural network layers on FPGA. To further expedite computations, a set of parallel operations has been employed. For instance, when data needs to be processed by a convolutional layer with 32 filters, a module is created to read the image and distribute it to 32 calculation modules simultaneously. Initially, the data is read and dispatched to these modules. Then, the 32 modules perform the convolution operations in parallel, applying the predetermined weights to the input image. This parallel processing approach has been similarly adopted in other layers.

Figure 3 illustrates how memory is positioned between layers in the network, crucial for all layer types to ensure the presence of memory between different network layers for storing each layer's output. After the completion of a layer's output, it is stored in memory before forwarding it to the subsequent layer. Due to the substantial memory requirements in the hardware section, the dimensions of the input images have been reduced.

The computational blocks designed for the FPGA encompass operations related to convolution, image addition, the fully connected layer, max pooling, and the combiner. Additionally, the data transmission modules include a sender to the convolution layer, a sender to the max pooling layer, and a sender to the fully connected layer. These modules are designed and tested individually and must be assembled for the system to function in its final form. It should be noted that the two layers of batch normalization and dropout are not included in the hardware implementation section because they are only used in the training.

1. **Convolution Sender Module:** In the proposed architecture, all filters in the convolution layers have dimensions of 5×5 , with movement steps of 1×1 and zero-padding applied. This setup requires applying a 5×5 filter over a set of pixels at each step, which is read from RAM, and then transmitted in a streaming fashion to the subsequent convolution module. The purpose of this module, as its name suggests, is to send data to the convolution layer. Convolution layers are configured in parallel, each with its own specific parameters. The data stream output is simultaneously dispatched to all these layers, and they begin processing the data concurrently at a specific time. The output from each convolution module is then stored in its dedicated memory area. Figure 4 demonstrates how four convolution modules are positioned in parallel following the sender module. Considering the 5×5 dimensions of the convolution filters and the application of zero-padding, two rows of zeros should be added around both the rows and columns of the images.
2. **Convolution Computation Module:** Due to the extensive number of multiplications and additions required in convolution operations, these tasks are executed in parallel on an FPGA. Typically, modules called multiply and accumulation (MAC) are employed for this process. In a MAC module, all inputs for a layer, along with their corresponding weights, must be

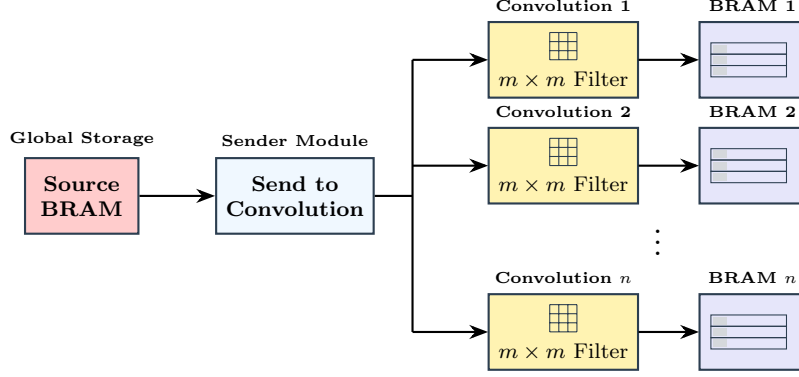


Figure 4: Architecture of convolution computation module

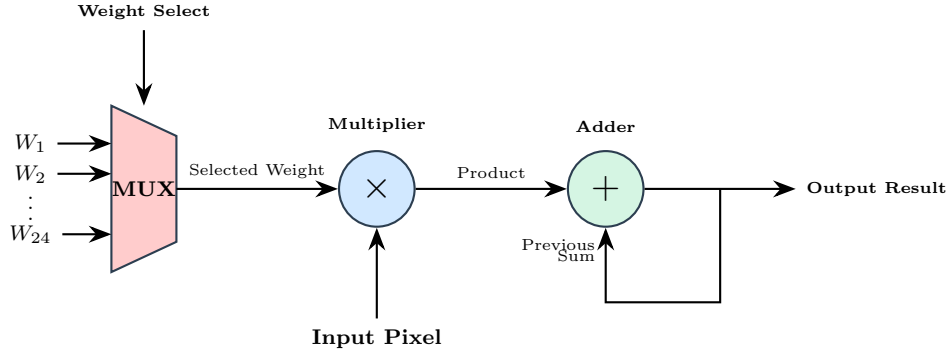


Figure 5: Serial convolution calculation method

loaded simultaneously, and the multiplication operations are executed within one clock cycle. In the following clock cycles, results are accumulated in pairs, and finally, the bias is added to produce the output [33]. While this parallel approach offers low latency, it requires a significant number of resources, particularly multipliers, which may not be resource-efficient for architectures with numerous convolution layers. Hence, our design opts for serial and sequential calculations for convolution operations to conserve resources. Figure 5 depicts the serial convolution computation method used in our design. In this method, input pixels are transmitted serially to the convolution module, with weights preloaded. Each input pixel is multiplied by its corresponding weight, and the product is added to the accumulating value in each clock cycle. This process continues until all data is processed. Eventually, the accumulated result is added to the bias, and the final outcome, along with the pixel address, is stored in the Block RAM memory and moved to the next register for subsequent calculations. This method minimizes resource usage, requiring only one adder and one multiplier. The operation progresses pixel by pixel, following the one-pixel movement steps, and repeats for each new pixel position.

3. **Max-Pooling Layer:** This layer consists of two components: data transmission and maximum pooling computation. The data transmission component reads data from the Block RAM and forwards it to the pooling calculation module. According to the proposed architecture, all pooling layers operate using the maximum pooling. In the maximum pooling layer, the concept of layering is implicitly applied. A window of the specified dimensions is placed

over the input pixels, streaming them as data. Then, this window progresses according to the defined stride, reading and resending values, thereby covering the entire image. This module is designed generically and functions correctly for images of any dimension.

4. **Activation Functions:** In the proposed architecture discussed in this paper, two activation functions, ReLU and SoftMax, are employed. The computation process for the ReLU activation function follows a simple rule: if the input value is greater than zero, the output remains the same as the input; otherwise, the output becomes zero. As mentioned, this function is not implemented as a separate module, but it is used as part of both the convolutional and fully connected layers. For SoftMax implementation, the initial step involves comparing two inputs. Following this, two multiplexers are utilized to monitor the input number along with its respective value. Once the maximum value is identified, its index is transmitted as the final output. The crucial aspect of SoftMax lies in determining the number associated with the maximum value. This involves comparing different outputs to ascertain the maximum value, and eventually, the index corresponding to that value is selected as the final output.
5. **Image Aggregator:** In convolutional layers, new outputs are generated by applying convolution filters to images, with each output resulting from a convolution filter applied to the input image. In subsequent convolutional layers, multiple input images result in a greater number of output images due to the application of multiple filters. These output images should be aggregated according to the number of filters in each layer. For instance, consider a two-layer convolutional network with three filters in the first layer and two filters in the second layer. After applying these filters, the first layer produces three output images. These three images then enter the second convolutional layer in parallel. Then, the two filters of the second layer are applied separately to each of the images. Consequently, the output of the second layer consists of six images. It is essential to aggregate these images to ensure the accuracy of the layer's output. For this purpose, an image aggregator module has been implemented, consisting of a sending section and a calculation section. The sending section reads and transmits the images once, while the calculation section sums the corresponding elements in parallel and forwards them to the next aggregator. The output from these aggregator modules is then passed to the subsequent module, and upon completion of all summations, the final results are stored in the Block RAM.
6. **Image Concatenator:** At the end of the two distinct convolutional pathways in the proposed architecture, the outputs are combined into sequential strings of values and then fed into the first fully connected layer. The primary function of this module is to retrieve data from the Block RAMs and allocate it to the input neurons of the fully connected layer. In this process, the data leaves the memory and is sequentially transferred to the next phase. One of the most crucial aspects in designing this module is the preservation of data order. Designing this module involves critical decisions regarding the arrangement and sequencing of data on the input neurons, including determining which pathway to begin reading data and in what sequence to arrange it on the input neurons.
7. **Fully Connected Layer:** The fully connected layer is composed of three principal components: the data transmission module, neurons, and the module receiving inputs from these neurons. As previously stated, the fully connected layer functions similarly to a multi-layer perceptron network, featuring neurons across input, hidden, and output layers. In our proposed architecture, there are three fully connected layers towards the end, with the initial layer containing 256 neurons, followed by a second layer comprising 128 neurons. The final

Table 1: Specifications of FVC2000 [34]

Dataset	Sensor Type	Image Size	Resolution
DB1	Low-cost Optical Sensor	300×300	500 dpi
DB2	Low-cost Capacitive Sensor	256×364	500 dpi
DB3	Optical Sensor	448×478	500 dpi
DB4	Synthetic Generator	240×320	~ 500 dpi

layer contains the same number of neurons as categories in the dataset. As previously mentioned, each neuron holds parameters refined during the training phase. Within every neuron, a unique weight corresponds to each input; this weight, once multiplied by its respective input, is added to an accumulated sum, and finally, this sum is adjusted by the bias to yield the neuron’s output. These computed inputs activate each layer, and depending on the designated output function, they undergo a subsequent transformation. In this implementation, weights and biases are stored within memory, and input data is dispatched serially to the neurons. With each passing clock cycle, the specified weight is multiplied by its corresponding input, and this result is accumulated with the sum from the previous cycle; then, the bias is added to arrive at the final output value.

Notably, in our proposed hardware implementation, several strategies have been employed to minimize power consumption. The most significant strategies include: employing Clock Gating for inputs, utilizing a state machine to lower power usage, implementing parallel processing, maintaining layers in an inactive state until data is received, minimizing the utilized bit size, and using intellectual property (IP) core in low power mode.

5 Results and Discussions

To evaluate the proposed model, Python programming language and the KERAS platform with the TensorFlow tool are employed. The implementations were conducted on the Google Colab platform, and the data and images were stored in Google Drive. Google Colab provides 12.7GB of RAM, and three types of processors: GPU, TPU, and CPU. In all Evaluations, GPU was utilized due to its higher speed. The employed GPU was the NVIDIA Tesla K80 with a 12GB memory.

5.1 Dataset

Three databases of FVC2000 [34], FVC2002 [35], and FVC2004 (version B) [36] used for training and evaluation of the proposed model. These databases belong to the University of Bologna, Italy, and have been widely used in various research studies. Tables 1 to 3 show the characteristics of these datasets. Given the characteristics outlined in Tables 1 to 3 each of these datasets contains images with various dimensions and non-uniform sample distributions across different sections. Therefore, it is important not to combine the data or to use them with a unified perspective.

A notable aspect of this study is the preservation of original image dimensions during data input to the neural network. Contrary to many studies where researchers resize images to reduce the number of training parameters or to expedite the testing process, our approach prioritizes maintaining image details to prevent potential accuracy loss.

Table 2: Specification of FVC2002 [35]

Dataset	Sensor Type	Image Size	Resolution
DB1	Optical	388×374	500 dpi
DB2	Optical	296×560	569 dpi
DB3	Capacitive	300×300	500 dpi
DB4	Synthetic	288×384	500 dpi

Table 3: Specification of FVC2004 [36]

Dataset	Technology	Image size	Resolution
DB1	Optical Sensor	640×480	500 dpi
DB2	Optical Sensor	328×364	500 dpi
DB3	Thermal Sweeping Sensor	300×480	512 dpi
DB4	Synthetic Generator	288×384	~ 500 dpi

5.2 Proposed Data Augmentation Method

Proper data augmentation plays a critical role in addressing the limitations of datasets with a restricted number of images. The integrity of raw data is fundamental, and any insufficiency in this data can negatively affect the accuracy of the proposed model [37]. Data augmentation enhances both the size and quality of datasets, thereby improving the performance of a deep learning model. Augmentation techniques introduce variations in several aspects such as rotation, cropping, color space adjustments, noise injection, and more, with the primary goal of addressing data shortages and mitigating overfitting issues [38]. In this paper, data augmentation was employed with specific modifications including:

1. Rotation of images by 15 degrees in both clockwise and counterclockwise directions.
2. Shifting of images by 50 pixels.
3. Adjustment of image brightness levels.

Due to the importance of the initial alignment of the fingerprint images, a rotation of 15 degrees has been applied. If the rotation exceeds this, the images would not be correctly aligned, leading to the creation of incorrect data. Similarly, the movement of the image is also crucial. If the displacement in terms of pixels is too large, some parts of the images and fingerprint lines may be lost. For this reason, only a 50-pixel displacement has been applied.

Initially, each label in the datasets comprised only 8 images. With the application of the aforementioned data augmentation method, the number of images for each label was increased to 80. Figure 6 presents an example of these augmented images. The original image is highlighted in red, while the subsequent images are generated by applying various modifications to the original image. Additionally, two scenarios were considered for data augmentation in this paper:

1. Data Splitting and Augmentation for Each Class Separately [39]: in this scenario, data are split into two groups: training and testing. The data augmentation technique is then applied separately to each class within these groups.
2. Data Splitting and Augmentation Only for Training Data [38]: Like the first scenario, the data are divided into training and testing groups. However, in this case, the data augmentation technique is exclusively applied to the training data and testing data remains unaltered.

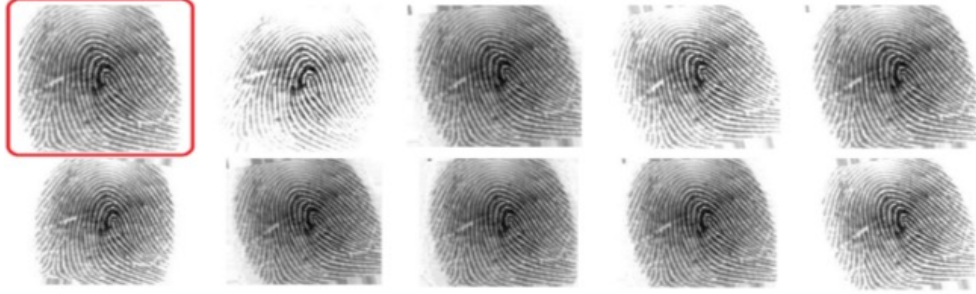


Figure 6: New images generated using data augmentation

5.3 Proposed Model Evaluation

In this section, we explore accuracy of the proposed model in fingerprint matching. The implementation utilized the Adam optimizer, with a batch size of 32, coupled with a callback function. This function’s role was selecting the optimal weights from the last 30 iterations. If no improvement was observed over these iterations, the training process was terminated, ensuring that the most effective weights were obtained as the final outcome.

The proposed model underwent a comparative analysis with several preceding studies, including AlexNet [40], Finger ConvNet [7], proposed model in [40], and proposed methods in [28]. It is important to note that efforts were made to accurately replicate all aspects and methodologies from these studies. Furthermore, to ensure a standardized evaluation, the same two data partitioning scenarios, employing cross-validation techniques, were applied across all evaluated models.

The evaluations encompass various evaluative criteria, including accuracy, hardware-oriented metrics, and the degree of optimization for practical deployment.

Table 4 presents the accuracy results for the first and second data augmentation scenarios, respectively. An analysis of these tables reveals that the accuracies reported in Table 4 (second scenario) are generally lower than those reported in Table 4 (first scenario). This discrepancy can be attributed to the smaller number of test samples in the second scenario. In the first scenario, the data augmentation method is applied to both the training and test data, thereby providing a broader range of samples for testing. Conversely, in the second scenario, the test data does not undergo augmentation, leading to fewer testing opportunities for the neural network. As a result, the first scenario typically shows higher accuracy due to the increased diversity and volume of test samples.

This observation underscores the significant impact of data augmentation on the training and testing. The application of data augmentation not only enhances the training process by introducing more varied data but also expands the testing scope, offering a more comprehensive evaluation of the model’s performance. Consequently, some accuracies reported in Table 4 for second scenario are lower, reflecting the limitations in testing scope when data augmentation is not applied to test data.

Figure 7 also illustrates the average accuracy achieved by each method across all datasets. As this figure shows, the proposed model offers better accuracy than all previous models over both data augmentation scenarios.

Table 5 provides the number of parameters for the network when using input images with original dimensions. Notably, previous works have typically featured a high number of parameters, whereas one of the primary objectives of this paper is parameter reduction. To contextualize the memory requirements, Table 5 displays the memory space needed for each model, considering that

Table 4: Accuracy of Different Models

Model	AlexNet [40]		Proposed in [26]		Finger ConvNet [7]		LOG [28]		Prewitt [28]		Proposed Model	
	First	Second	First	Second	First	Second	First	Second	First	Second	First	Second
FVC2000												
DB1	48.2	15.7	77.1	50.1	86.0	59.2	73.2	52.3	78.8	62.5	81.8	19.8
DB2	58.4	46.4	78.4	74.4	69.1	61.6	68.1	56.2	69.0	66.0	85.1	71.2
DB3	68.7	66.6	78.1	79.4	77.1	76.6	78.4	75.5	78.3	78.5	90.5	90.8
DB4	37.2	37.5	53.7	55.3	78.2	76.2	57.4	56.9	55.4	60.8	65.6	54.2
FVC2002												
DB1	27.7	29.1	38.4	36.7	53.8	55.8	35.6	34.8	37.2	38.7	57.5	59.6
DB2	35.1	31.4	42.8	42.3	50.3	46.4	39.8	39.8	40.7	39.6	60.3	57.8
DB3	53.3	32.6	62.5	58.9	74.9	60.5	56.5	32.1	56.3	39.2	76.2	67.3
DB4	34.5	33.0	41.5	43.3	57.8	55.7	42.9	36.7	39.1	38.3	74.5	78.9
FVC2004												
DB1	26.2	28.3	30.0	32.6	41.1	43.7	26.2	29.8	27.0	33.0	47.1	50.8
DB2	23.5	23.3	27.7	28.0	42.2	40.8	24.2	25.5	22.7	24.4	47.4	45.5
DB3	75.3	74.8	73.3	73.0	86.1	86.0	73.1	76.9	79.3	83.7	94.1	94.4
DB4	32.1	30.3	37.2	39.8	52.3	53.9	40.1	41.2	31.5	31.9	64.3	67.8
Average	43.3	37.4	53.3	51.1	64.0	59.7	51.2	46.4	51.2	49.7	70.3	63.1

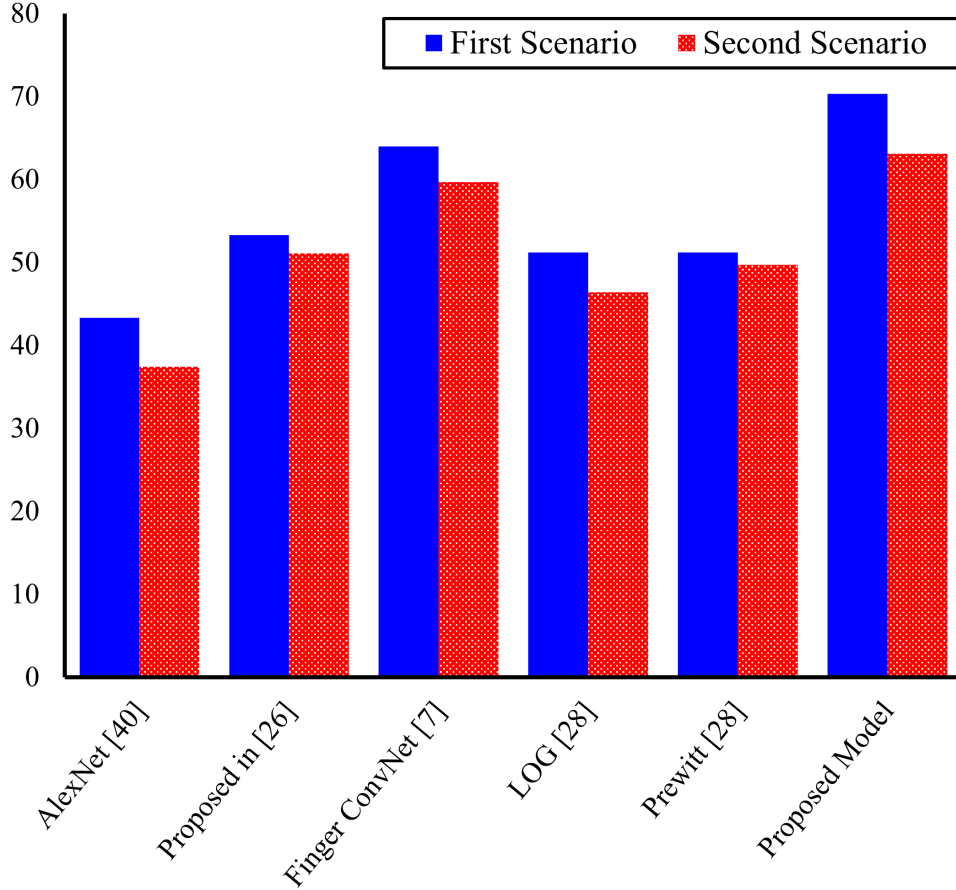


Figure 7: Average accuracies of the different methods

each parameter consumes 32 bits (equivalent to 4 bytes) of space. The proposed model in this paper significantly reduces the amount of memory used compared to other models, as evidenced

Table 5: The Number of Parameters and Required Memory of Each Model

	AlexNet [40]		Proposed in [26]		Finger ConvNet [7]		LOG & Prewitt [28]		Proposed Model	
	Memory (MB)	Parameters ($\times 10^3$)	Memory (MB)	Parameters ($\times 10^3$)	Memory (MB)	Parameters ($\times 10^3$)	Memory (MB)	Parameters ($\times 10^3$)	Memory (MB)	Parameters ($\times 10^3$)
FVC2000										
DB1	87664.7	334.41	972.1	3.70	5584.3	21.30	2983.1	11.37	113.9	0.43
DB2	83470.4	318.41	999.1	3.81	5379.5	20.52	3040.4	11.59	118.5	0.45
DB3	184133.7	702.41	2459.1	9.38	11216.3	42.78	6907.0	26.34	138.5	0.52
DB4	70877.4	270.41	803.1	3.06	4560.3	17.39	2483.4	9.47	109.3	0.41
FVC2002										
DB1	135899.2	518.41	1610.1	6.14	7837.1	29.89	4941.0	18.84	124.6	0.47
DB2	154773.5	590.41	1871.1	7.13	9680.3	36.92	5473.4	20.87	127.7	0.48
DB3	87664.7	334.41	972.1	3.70	5584.3	21.30	2983.1	11.37	113.9	0.43
DB4	93956.1	358.41	1211.1	4.61	5993.9	22.86	3564.7	13.59	118.5	0.45
FVC2004										
DB1	265922.6	1014.41	3530.1	13.46	15824.3	60.36	9856.2	37.59	153.8	0.58
DB2	114927.6	438.41	1303.1	4.97	7222.7	27.55	3982.5	15.19	124.6	0.47
DB3	129607.7	494.41	1592.1	6.07	8144.3	31.06	4695.2	17.91	123.1	0.46
DB4	93956.1	358.41	1211.1	4.61	5993.9	22.86	3564.7	13.59	118.5	0.45
Average	116655.3	445.00	1544.5	5.89	8059.0	30.74	4539.5	17.31	123.7	0.47

by Table 5.

For a more detailed and precise examination of the proposed architectural pathways, tests were conducted separately to evaluate each main pathway of the model. Specifically, in the first test, only the right pathway was considered; in the second test, only the left pathway was considered; and in the third test, the final architecture, which combines both pathways, was examined. In all tests, the final section of the model, which includes the fully connected layers, remained constant. After conducting several tests on one of the databases used (FVC2000.DB1), the results were as follows: The average accuracy for the right pathway was 79.2%, for the left pathway was 85.1%, and for both pathways combined (the final model) was 87.7%. These results indicate that each pathway and different sections of the model extract specific features, leading to varying accuracies. However, combining the two pathways improved the model’s performance. Additionally, the right pathway, which extracts smaller features, achieved lower accuracy compared to the left pathway, suggesting that larger features are more important for fingerprint recognition.

5.4 Hardware Implementation Results

After training and evaluating the proposed model, it is time for hardware implementation of the proposed model, specifically on an FPGA platform. As mentioned earlier, the network training phase is performed on the GPU platform, and the training parameters can be extracted and transferred to the FPGA for implementation. In this phase, the implementation of the network on the FPGA platform is carried out layer by layer. This process is accomplished using the VHDL language on the Vivado 2020.2. The modules were implemented separately, and after calculating the consumed resources and maximum operating frequency of the system, Xilinx power estimator was used to more accurately determine the power.

The selected hardware is xc7vx1140tflg1930-2L, which belongs to the Virtex-7 series of Xilinx FPGAs. This hardware is a pure FPGA and does not include a CPU. The Virtex-7 series from Xilinx utilizes 28-nanometer fabrication Technology and is considered a cost-effective series [41]. This FPGA contains 1930 input/output pins, 712,000 look-up tables (LUTs), 1,424,000 Flip-Flops, 1880 blocks of 36-kilobit RAM, and 3360 digital signal processing (DSP) units.

Table 6 shows the FPGA resource utilization after the implementation of the proposed model. The model’s computational complexity and memory requirements have led to significant utilization

Table 6: FPGA Resource Utilization				
Resources	LUT	Flip Flop	DSP	BRAM
Utilized	537319	415204	2985	1880
Available	712000	1424000	3360	1880
Utilization (%)	75.4	29.1	88.8	100

of FPGA resources, with over 70% of the total resources being consumed. Specifically, a substantial portion of DSP units, LUTs, flip-flops, and memory blocks have been utilized. The highest usage of DSP resources is observed for multiplication operations, which are commonly used in convolutional and fully connected layers. LUTs are used in various sections, such as addition in different layers, value comparison in pooling layers, and implementing activation functions. Flip-flops and memory blocks are also employed for storing values, signals, weights, and outputs of different layers. The implemented model can operate with frequency of 108.6 MHz while consuming 27.54 watts of dynamic power and 1.18 watts of static power. Dynamic power in FPGAs is derived from various components. Table 7 provides different components of dynamic power consumption.

Table 7: Different Portion of Dynamic Power Consumption

Components	Power consumption (W)
Clocking	2.22
Logic	16.13
I/O	1.11
BRAM	2.30
DSP	5.78
Total	27.54

6 Conclusion

This paper proposes a lightweight yet high accuracy CNN model for fingerprint-based individual identification. The proposed model consists of nine layers and achieves significant accuracy improvements with a 95% reduction in parameters compared to previous fingerprint-based individual identification methods. This reduction in parameters directly addresses hardware implementation challenges, particularly memory constraints, making the model exceptionally memory-efficient. Employing the cross-validation method as the evaluation mechanism, the model demonstrates an accuracy of 70.3% with data augmentation and 63.1% without, also outperforming the previous fingerprint-based individual identification methods in terms of accuracy. Furthermore, the FPGA implementation highlights the model’s operational efficiency, achieving a maximum operating frequency of 108.6 MHz and demonstrating the model’s real-time processing capabilities. The total power consumption of 28.72 watts also shows an optimized balance between performance, accuracy, and energy efficiency. This result makes the proposed model well-suited for FPGA implementations.

Declarations

Ethics Approval: Not applicable. This study used existing fingerprint datasets, namely FVC2000 [34], FVC2002 [35], and FVC2004 [36]. The authors did not recruit human participants or collect new human-subject data for this study.

Consent to Participate: Not applicable. No human participants were directly recruited or enrolled by the authors in this study.

Consent for Publication: Not applicable.

Human Ethics and Consent to Participate Declarations: Not applicable.

Funding: The authors received no specific funding for this work.

Data Availability: The datasets analysed during the current study are publicly available from the Biometric System Laboratory at the University of Bologna. The datasets used in this study are listed below:

- FVC2000 [34]: <https://biolab.csr.unibo.it/DatabaseSoftware.asp?organize=Databases&select=1>
- FVC2002 [35]: <https://biolab.csr.unibo.it/DatabaseSoftware.asp?organize=Databases&select=11>
- FVC2004 [36]: <https://biolab.csr.unibo.it/DatabaseSoftware.asp?organize=Databases&select=12>

Clinical trial number: Not applicable.

References

- [1] S. Minaee, A. Abdolrashidi, H. Su, M. Bennamoun, and D. Zhang, “Biometrics recognition using deep learning: A survey,” *Artificial Intelligence Review*, vol. 56, no. 8, pp. 8647–8695, 2023.
- [2] Y. Feng and A. Kumar, “Detecting locally, patching globally: An end-to-end framework for high speed and accurate detection of fingerprint minutiae,” *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 1720–1733, 2023.
- [3] J. J. Engelsma, K. Cao, and A. K. Jain, “Learning a fixed-length fingerprint representation,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 6, pp. 1981–1997, 2019.
- [4] S. A. Grosz and A. K. Jain, “Afr-net: Attention-driven fingerprint recognition network,” *IEEE Transactions on biometrics, behavior, and identity science*, vol. 6, no. 1, pp. 30–42, 2023.
- [5] K. Cao and A. K. Jain, “Automated latent fingerprint recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 41, no. 4, pp. 788–800, 2018.
- [6] R. Cappelli, M. Ferrara, and D. Maltoni, “Minutia cylinder-code: A new representation and matching technique for fingerprint recognition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 12, pp. 2128–2141, 2010.
- [7] Y. Liu, B. Zhou, C. Han, T. Guo, and J. Qin, “A novel method based on deep learning for aligned fingerprints matching,” *Applied Intelligence*, vol. 50, no. 2, pp. 397–416, 2020.
- [8] J. Song, C. Cho, and Y. Won, “Analysis of operating system identification via fingerprinting and machine learning,” *Computers & Electrical Engineering*, vol. 78, pp. 1–10, 2019.

- [9] M. Ahsan, M. Based, J. Haider, M. Kowalski *et al.*, “An intelligent system for automatic fingerprint identification using feature fusion by gabor filter and deep learning,” *Computers and Electrical Engineering*, vol. 95, p. 107387, 2021.
- [10] U. Michelucci, *Advanced applied deep learning: convolutional neural networks and object detection*. Apress, 2019.
- [11] M. Mahmoodpour, A. Amirany, M. H. Moaiyeri, and K. Jafari, “A learning based contrast specific no reference image quality assessment algorithm,” in *2022 international conference on machine vision and image processing (MVIP)*. IEEE, 2022, pp. 1–4.
- [12] A. Amirany, K. Jafari, and M. H. Moaiyeri, “A task-schedulable nonvolatile spintronic field-programmable gate array,” *IEEE Magnetics Letters*, vol. 12, pp. 1–4, 2021.
- [13] L. Kalms, P. A. Rad, M. Ali, A. Iskander, and D. Goehringer, “A parametrizable high-level synthesis library for accelerating neural networks on fpgas,” *Journal of Signal Processing Systems*, vol. 93, no. 5, pp. 513–529, 2021.
- [14] A. K. Mukhopadhyay, S. Majumder, and I. Chakrabarti, “Systematic realization of a fully connected deep and convolutional neural network architecture on a field programmable gate array,” *Computers & Electrical Engineering*, vol. 97, p. 107628, 2022.
- [15] S. Messaoud, S. Bouaafia, A. Maraoui, A. C. Ammari, L. Khriji, and M. Machhout, “Deep convolutional neural networks-based hardware–software on-chip system for computer vision application,” *Computers & Electrical Engineering*, vol. 98, p. 107671, 2022.
- [16] R. O. Hassan and H. Mostafa, “Implementation of deep neural networks on fpga-cpu platform using xilinx sdsoc,” *Analog Integrated Circuits and Signal Processing*, vol. 106, no. 2, pp. 399–408, 2021.
- [17] D. Ghimire, D. Kil, and S.-h. Kim, “A survey on efficient convolutional neural networks and hardware acceleration,” *Electronics*, vol. 11, no. 6, p. 945, 2022.
- [18] V. H. Phung and E. J. Rhee, “A high-accuracy model average ensemble of convolutional neural networks for classification of cloud image patches on small datasets,” *Applied Sciences*, vol. 9, no. 21, p. 4500, 2019.
- [19] D. Sun, E. Wei, L. Yang, and S. Xu, “Improving fingerprint indoor localization using convolutional neural networks,” *IEEE Access*, vol. 8, pp. 193 396–193 411, 2020.
- [20] A. B. Mazlan, Y. H. Ng, and C. K. Tan, “A fast indoor positioning using a knowledge-distilled convolutional neural network (kd-cnn),” *IEEE access*, vol. 10, pp. 65 326–65 338, 2022.
- [21] K. Liu, G. Kang, N. Zhang, and B. Hou, “Breast cancer classification based on fully-connected layer first convolutional neural networks,” *IEEE Access*, vol. 6, pp. 23 722–23 732, 2018.
- [22] R. Solovyev, A. Kustov, D. Telpukhov, V. Rukhlov, and A. Kalinin, “Fixed-point convolutional neural network for real-time video processing in fpga,” in *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIconRus)*. IEEE, 2019, pp. 1605–1611.
- [23] J. M. Shrein, “Fingerprint classification using convolutional neural networks and ridge orientation images,” in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–8.

- [24] C. Militello, L. Rundo, S. Vitabile, and V. Conti, “Fingerprint classification based on deep learning approaches: experimental findings and comparisons,” *Symmetry*, vol. 13, no. 5, p. 750, 2021.
- [25] B. K. Oleiwi, L. H. Abood, and A. K. Farhan, “Integrated different fingerprint identification and classification systems based deep learning,” in *2022 International Conference on Computer Science and Software Engineering (CSASE)*. IEEE, 2022, pp. 188–193.
- [26] H. Shafaghi, M. Kiani, A. Amirany, K. Jafari, and M. H. Moaiyeri, “A fast and light fingerprint-matching model based on deep learning approaches,” *Journal of Signal Processing Systems*, vol. 95, no. 4, pp. 551–558, 2023.
- [27] I. G. Dakhil and A. A. Ibrahim, “Design and implementation of fingerprint identification system based on knn neural network,” *Journal of Computer and Communications*, vol. 6, no. 3, pp. 1–18, 2018.
- [28] A.-M. Dincă Lăzărescu, S. Moldovanu, and L. Moraru, “A fingerprint matching algorithm using the combination of edge features and convolution neural networks,” *Inventions*, vol. 7, no. 2, p. 39, 2022.
- [29] D. T. Nguyen, N. H. Hung, H. Kim, and H.-J. Lee, “An approximate memory architecture for energy saving in deep learning applications,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 5, pp. 1588–1601, 2020.
- [30] I. Westby, X. Yang, T. Liu, and H. Xu, “Fpga acceleration on a multi-layer perceptron neural network for digit recognition,” *The Journal of Supercomputing*, vol. 77, no. 12, pp. 14 356–14 373, 2021.
- [31] D. Rivoir, I. Funke, and S. Speidel, “On the pitfalls of batch normalization for end-to-end video learning: a study on surgical workflow analysis,” *Medical Image Analysis*, vol. 94, p. 103126, 2024.
- [32] A. Faraji, S. A. Sadrossadat, J. Jin, W. Na, F. Feng, and Q.-J. Zhang, “Hybrid batch-normalized deep feedforward neural network incorporating polynomial regression for high-dimensional microwave modeling,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 71, no. 3, pp. 1245–1258, 2024.
- [33] W. Ding, Z. Huang, Z. Huang, L. Tian, H. Wang, and S. Feng, “Designing efficient accelerator of depthwise separable convolutional neural network on fpga,” *Journal of Systems Architecture*, vol. 97, pp. 278–286, 2019.
- [34] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. K. Jain, “Fvc2000: Fingerprint verification competition,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 24, no. 3, pp. 402–412, 2002.
- [35] D. Maio, D. Maltoni, R. Cappelli, J. L. Wayman, and A. Jain, “FVC2002: Second Fingerprint Verification Competition,” in *Proceedings of the 16th International Conference on Pattern Recognition (ICPR)*, vol. 3. Quebec City, Canada: IEEE, August 2002, pp. 811–814.
- [36] D. Maio, D. Maltoni, R. Cappelli, J. Wayman, and A. Jain, “FVC2004: Third Fingerprint Verification Competition,” in *Biometric Authentication: First International Conference (ICBA 2004)*, ser. Lecture Notes in Computer Science, vol. 3072. Hong Kong, China: Springer Berlin Heidelberg, July 2004, pp. 1–7.

- [37] N. E. Khalifa, M. Loey, and S. Mirjalili, “A comprehensive survey of recent trends in deep learning for digital images augmentation,” *Artificial Intelligence Review*, vol. 55, no. 3, pp. 2351–2377, 2022.
- [38] M. M. Islam, M. B. Hossain, M. N. Akhtar, M. A. Moni, and K. F. Hasan, “Cnn based on transfer learning models using data augmentation and transformation for detection of concrete crack,” *Algorithms*, vol. 15, no. 8, p. 287, 2022.
- [39] F. Perez, C. Vasconcelos, S. Avila, and E. Valle, “Data augmentation for skin lesion analysis,” in *International Workshop on Computer-Assisted and Robotic Endoscopy*. Springer, 2018, pp. 303–311.
- [40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.
- [41] Xilinx Inc., *7 Series FPGAs Data Sheet: Overview*, Xilinx Inc., February 2020, dS180 (v2.6.1). [Online]. Available: https://www.xilinx.com/support/documentation/data.sheets/ds180_7Series-Overview.pdf