*Cioloca Mihaela, al. Int. International Journal of Engineering Research and Applications*
*www.ijera.com*
*ISSN: 2248-9622, Vol. 15, Issue 8, August 2025, pp 26-32*

RESEARCH ARTICLE OPEN ACCESS

# Design & Synthesis of an Efficient Fully Connected NN Model in DNN

P. Dileep Kumar Reddy*, Dr. Kota Venkata Ramanaiah**
*Department of Electronics and Communication Engineering, Y.S.R Engineering College of Yogi vemana University, proddatur-516360*
*** Department of Electronics and Communication Engineering, Y.S.R Engineering College of Yogi vemana University, proddatur-516360*

**ABSTRACT**
The deep neural network consists of several data processing subsystems, one of the key components being the Fully Connected Network (FCNN) model. This FCNN model has several layers of neurons, which are implemented using arithmetic units with appropriate numerical representation to optimize for area, output, and speed. This study examined the network parameters and removed any redundant weights. The architecture of the FCNN was designed to be both piped and parallelised in order to improve the processing of network information. The proposed FCNN consists of 16 input layers, 3 hidden layers, and one output layer, each containing 4 neurons. This design details the connections between inputs and neurons in hidden layers to process raw data. A hardware description language (HDL) model has been developed for this architecture, a refined architecture consisting of registers, de-multiplexers, weight registers, multipliers, and adders, and read-only memory.

***Keywords –*** CNN, Neuron, Mux, ALU, Pipelined

---

Date of Submission: 01-08-2025 Date of acceptance: 11-08-2025

---

## I. INTRODUCTION

Deep learning systems have become ubiquitous across various sectors, powering applications like image recognition, speech interfaces, and language translation. To support these computationally intensive tasks—especially in real-time scenarios—DNNs (Deep Neural Networks) are now deployed across a range of hardware platforms, including CPUs, GPUs, and increasingly, FPGAs. The rise of edge computing has further emphasized the necessity for fast, efficient on-device processing, as relying solely on cloud resources often fails to meet stringent latency requirements.

Yet, implementing DNNs on hardware platforms is fraught with challenges. The convolutional layer in CNNs [1] (Convolutional Neural Networks) stands out as particularly demanding, requiring significant computational resources to maintain accuracy and performance. Achieving an optimal balance between computational complexity, accuracy, and resource consumption remains a persistent concern. These challenges are compounded in fully connected layers, which require large-scale matrix multiplications and substantial parameter storage—both of which are especially problematic in edge environments with limited memory and power budgets.

To address these obstacles, several optimization strategies are employed. Techniques such as parallelization, batch processing, and partitioning are standard approaches to improve throughput. More advanced strategies, including pruning, quantization, and decomposition, have proven effective at reducing computational load without sacrificing significant accuracy. Methods leveraging the Fast Fourier Transform (FFT) [2] enable certain operations to be performed more efficiently in the frequency domain, particularly beneficial for large-kernel convolutions.

Furthermore, hardware-specific innovations have emerged. For example, resource multiplexing algorithms help optimize arithmetic complexity and utilization of FPGA lookup tables. Multiplier-less operations, such as using XOR-based computations, can further reduce hardware demands. Identifying and eliminating redundant parameters also decreases unnecessary computation, streamlining the network.

Implementations on FPGAs illustrate the practical impact of these methods. Xiaokang's work[3], for instance, integrates XOR operations, pipelined structures, and intermediate storage to achieve notable parallelism and minimize data access latency, demonstrating successful deployment

on an Artix-7 FPGA at 150 MHz. Similarly, Binfeng's[4] CNN accelerator leverages both hardware (Zynq FPGA) and software (ARM Cortex-A9) co-design, operating at 100 MHz with power consumption maintained below 1.6 W.

The presented work specifically focuses on Designing a two-layer fully connected neural network architecture. Optimizing arithmetic operations within this architecture. Carefully considering trade-offs related to delay. Applying parallelism and intermediate storage logic to enhance latency and throughput. Notably, a CNN model in this context integrates the Advanced Encryption Standard (AES) algorithm for data encryption, with the CNN generating the encryption key. The complexity of this CNN model has been optimized through appropriate training methods. The increasing complexity of the fully connected layer "as the number of hidden layers increases" also underscores the continuous challenge of scaling these designs[5].

The introduction of the paper should explain the nature of the problem, previous work, purpose, and the contribution of the paper. The contents of each section may be provided to understand easily about the paper.

## II. LITERATURE SURVEY

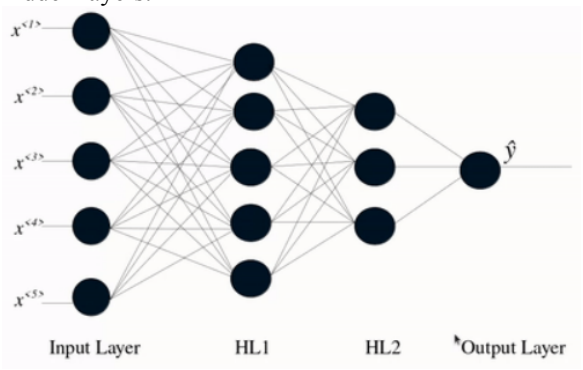
Neural network optimization is a critical area of research, particularly focused on balancing key performance metrics such as latency, area utilization, and computational efficiency. This field encompasses various techniques aimed at enhancing the overall performance of neural networks. The literature highlights three primary methods that have garnered significant attention: parallel processing, redundancy reduction, and pipelining. Parallel processing is widely recognized for its substantial ability to enhance computational speed and reduce latency in neural networks. Research, such as that by Zhang et al., [6], has demonstrated that employing parallel architectures can significantly improve the processing time of deep learning models, especially in applications that demand real-time data analysis. While parallel processing effectively reduces latency, it often introduces a trade-off, leading to increased area requirements. This necessitates a careful design balance between speed and hardware resource consumption. Redundancy reduction addresses the critical issue of redundant weight vectors within neural network computations. Studies by Chen et al[7]. have delved into methods for estimating these redundancies in weight matrices. They proposed algorithms specifically designed to minimize the number of multiplication operations required during the inference phase of neural network execution. This approach is highly beneficial as it not only reduces computational overhead but also improves the overall efficiency of the neural network, making it particularly suitable for deployment in resource-constrained environments.

Pipelining, a well-established technique in digital circuit design, has been effectively adapted for neural network architectures to boost performance. Kumar and Singh [8] highlighted the benefits of pipelining in increasing the processing speed and throughput of neural networks. However, they also emphasized that while pipelining significantly improves throughput, it comes with a trade-off in terms of latency. Therefore, careful design of pipelined architectures is crucial to efficiently handle data flow while maintaining acceptable latency levels for real-time applications. Fully Connected Neural Networks (FCNNs), which remain a foundational architecture in deep learning, have been a subject of significant optimization efforts. Recent advancements have focused on optimizing FCNNs for specific applications. For instance, Lee et al.[9] proposed a two-layer FCNN architecture that ingeniously incorporates both redundancy reduction and pipelining techniques. Their work showcased significant improvements in arithmetic operation efficiency, demonstrating the considerable potential for optimizing FCNNs to meet stringent delay requirements. The broader research, as indicated by Patel et al. [10], underscores the necessity of a holistic approach to neural network optimization. This framework emphasizes that architectural decisions must be informed by the specific requirements of the application domain. This integrated strategy, leveraging parallel processing, redundancy reduction, and pipelining, aims to achieve an optimal balance between arithmetic operation efficiency and delay requirements in neural

## III. FCNN

Referencing Figure 1, it depicts the architecture of a standard fully connected neural network, including an input layer, one or more hidden layers, and an output layer. The dimensions of the input and output layers are determined by the features present in the dataset and the specific nature of the desired output. Selecting the number of hidden layers, however, presents a nuanced challenge. There is no universally optimal choice; too few hidden layers may prevent the network from capturing underlying patterns, while too many can lead to overfitting and unnecessary complexity. Striking the right balance between model capacity and generalization is essential. For guidance, a straightforward mathematical formula is sometimes

employed to estimate the appropriate number of hidden layers:




n=(m+k)+ln = \sqrt{(m + k)} + ln=(m+k)+l

where mmm is the number of input nodes, k is the number of output nodes, and l is a constant ranging from 1 to 10. In a fully connected network, N is the total number of neurons:

N=m+n+kN = m + n + kN=m+n+k

The total number of weights D required for the fully connected neural network with NN nodes is given by:

D=mn+nk+n+kD = mn + nk + n + kD=mn+nk+n+k

A fully connected (FC) layer in a neural network architecture consists of an input layer, one or more hidden layers, and an output layer. The primary function of FC layers is to extract meaningful features from the output of preceding convolutional layers and map these features into a recognizable data space for classification or regression tasks.

The computational complexity of an FC layer is heavily influenced by the size and number of weight vectors involved. When the individual weights are of small magnitude (for example, less than 0.1), the resulting multiplication operations produce relatively small values, which can simplify the computational process and potentially lead to hardware-level optimizations. By strategically analyzing the magnitude of these weights, it is possible to optimize multiplication operations and reduce overall computational load.

Fig 1. Fully Connected NN model

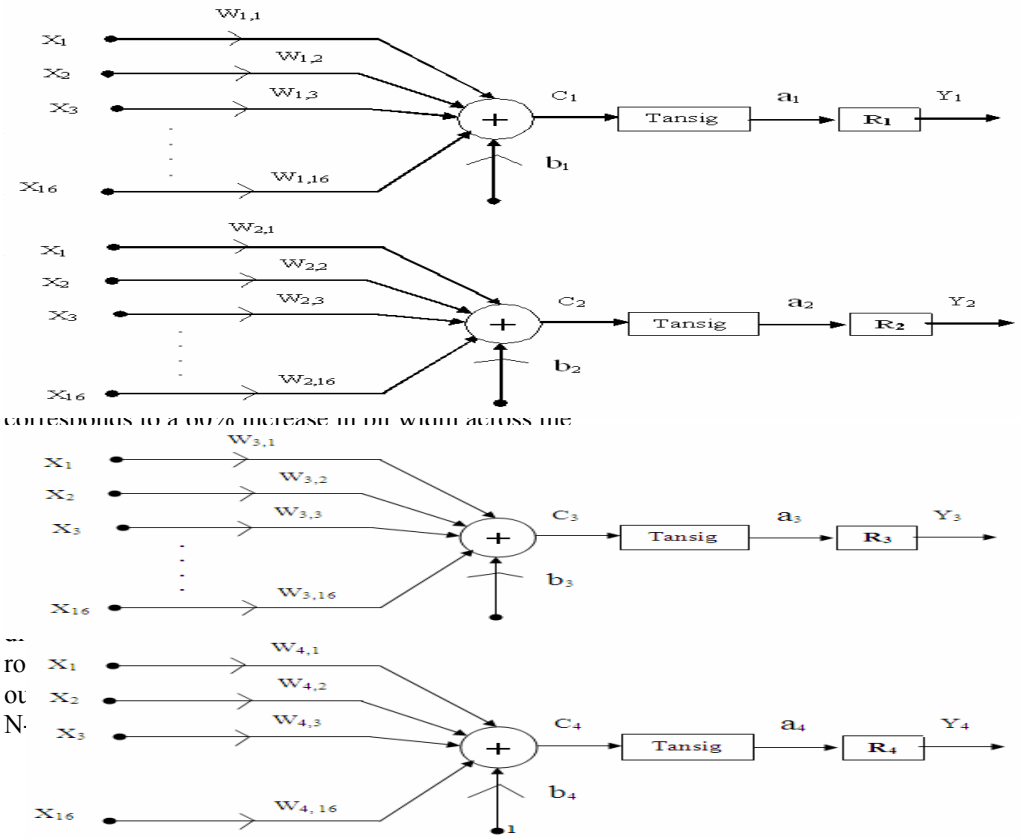
Nevertheless, as the number of weight vectors increases, the required number of arithmetic operations, memory storage elements, and overall resource utilization also rise correspondingly. Implementing a serial architecture for the FC layer can help conserve resources by allowing arithmetic units to be reused across multiple operations, albeit at the cost of increased computation time or latency. To address the trade-off between speed (delay) and hardware efficiency (area utilization), hybrid methods are often employed, aiming to strike a balance that meets both performance and resource constraints.

Figure 2 illustrates the configuration of 4 neurons, each handling 16 inputs and producing a single output, denoted as Y. Each neuron is equipped

with 16 multipliers, one for each input. The resulting products, along with a bias term, are processed by an adder array comprising 16 adders, which sums a total of 17 values. Each neuron is also associated with an activation function, which, in this implementation, is realized using a ROM-based structure.

For a HL containing four neurons and 16 inputs, the system requires a total of 64 multipliers, 64 adders, and four activation functions. The input data (X), as depicted in Figure 1, are represented in N-bit 2's complement format, with the most significant bit (MSB) indicating the sign and the remaining N-1 bits encoding the magnitude. The network weights, following training, also use N-bit 2's complement representation. Each multiplication operation produces a (2N-1)-bit result and requires (2N-1) clock cycles to complete. The multiplier outputs are accumulated in the adder array, where each adder processes two (2N-1)-bit inputs, yielding a 2N-bit result.

.




FFNN (Feed-Forward Neural Network) architecture is engineered with a clear focus on optimizing arithmetic resource utilization. This design is organized as a series of interconnected stages: beginning with input registers, followed by two levels of de-multiplexers, weight registers, multipliers, an adder array, a bias adder unit, a ROM/LUT stage, and culminating in the output register.

The process initiates with a bank of 16 input registers, each with a data width of N-bits.

Data is loaded sequentially, from the topmost to the bottom register. The output from each register directly interfaces with its respective de-multiplexer in the subsequent stage. A control signal, denoted 'C', determines the routing of input data—either directly to the multiplier inputs or to the next register in the sequence. During the data loading phase, 'C' is asserted ('1'), enabling all 16 input values to be written into the register array over 16 clock cycles. Once loading is complete, 'C' is de-asserted ('0'), and the register contents are made available to the multipliers for computation. Each neuron within the architecture requires 16 weights; thus, for a system comprising four neurons, a total of 64 weights are necessary. These are stored within a register array of 64-depth. The loading of weights into the weight registers is managed via the control signal 'D' for the third-stage de-multiplexer. Setting 'D' to '0' for 16 clock cycles facilitates the transfer of 16 weights into the weight registers corresponding to each neuron. Post-loading, the weights are routed to the multiplier inputs, and 'D' is set accordingly to maintain this connection. On the 17th clock cycle, the 16 multipliers are activated to perform the necessary multiplication operations.

The resulting products from the multipliers are then accumulated using an adder array. Although one might expect to require 15 adders to sum 16 products, the architectural design efficiently achieves this operation with just 7 adders, indicating a resource-conscious approach to arithmetic unit allocation

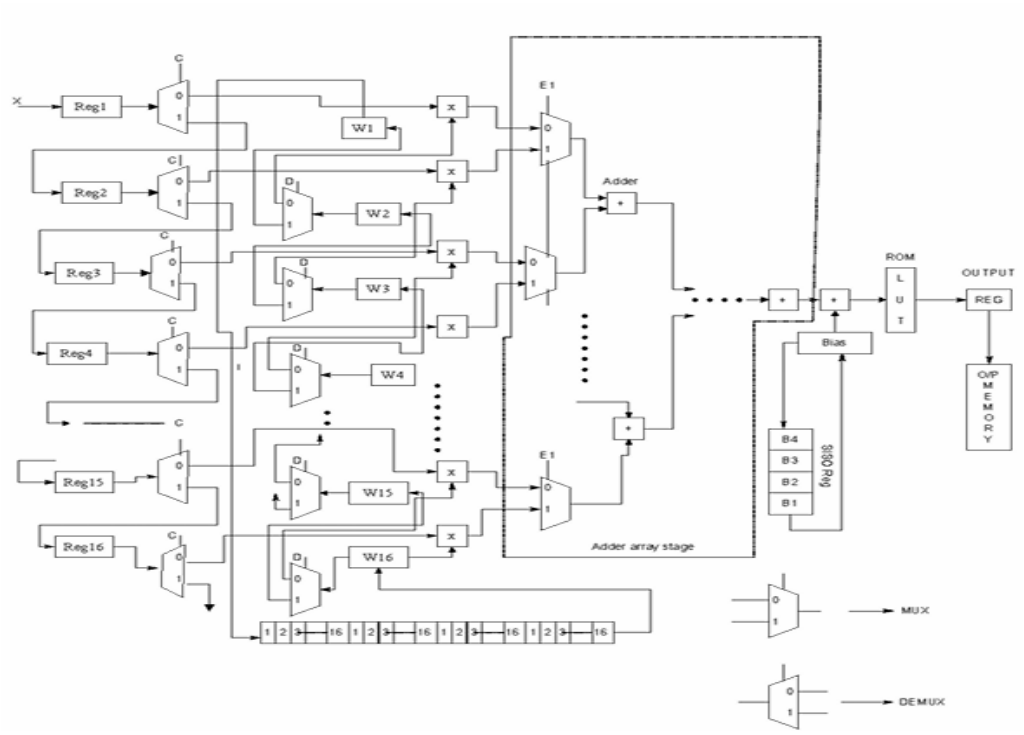


Fig 3. Pipelined architecture of single neuron of hidden unit

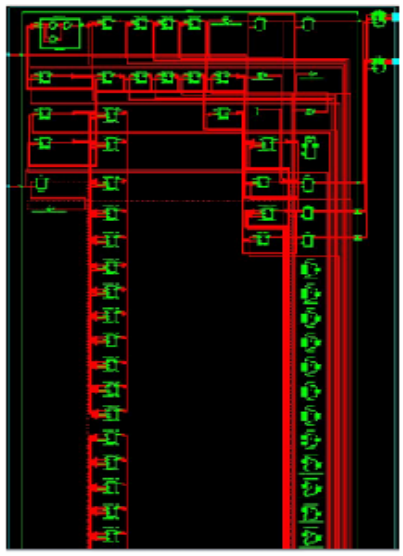
V.

The presented design offers a notably efficient FPGA-based implementation of a Feedforward Neural Network (FFNN), with substantial reductions in hardware resource consumption while preserving computational performance. Traditional approaches typically utilize 16 parallel multipliers within the hidden layer, resulting in significant area overhead. In contrast, the proposed architecture leverages a pipelined structure with only 8 multipliers, coordinated through two-stage Serial-In Serial-Out (SISO) registers. These registers are supplied weights from a 64-depth bottom SISO, facilitating alternate weight loading and minimizing the number of concurrently active multipliers. The multiplication process unfolds over two clock cycles, yielding eight partial products per cycle, which are subsequently stored in registers V1 through V16. The summation of these products is accomplished by a seven-stage pipelined adder array consisting of multiplexers, adders, demultiplexers, and registers. Bias values are managed via a dedicated FIFO register and integrated during the final computation stage. Overall, this configuration incurs a latency of 2N + 23 clock cycles, where N denotes the number of inputs. Notably, the design achieves a 75% area reduction, with only a 24% increase in computation delay relative to conventional implementations.

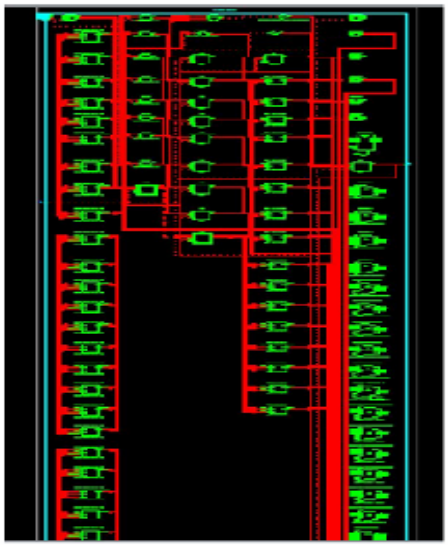
A similar optimization strategy is employed in the output layer. Each neuron utilizes a compact, pipelined configuration, comprising input and weight registers, arrays of multipliers and adders, a bias FIFO, ROM-based activation function, and output memory. To further enhance throughput, the design incorporates two parallel pipelined units, each responsible for producing eight outputs—enabling the simultaneous computation of 16 neuron outputs. This parallelization yields an 87.5% reduction in area and results in a 33% increase in computation delay compared to standard designs. The output layer's latency is measured at 2N + 8 clock cycles.

Overall, the proposed architecture effectively balances hardware efficiency and processing speed by employing pipelined computation, optimized arithmetic units, and synchronized memory management. These characteristics make it a compelling solution for real-time artificial neural network applications on FPGA platforms.

## VI. RESULTS AND CONCLUSION:



(a)



(b)

Fig 4.RTL schematic of a) hidden layer and b) output layer

The RTL schematic for both the hidden and output layers primarily consists of multipliers and adders, organized to process data efficiently. In the output stage, Look-Up Tables (LUTs) are utilized, aligning with established design practices. As previously discussed, the output layer is structured around a data path unit featuring multipliers and adders, all of which are configured for 10-bit operations. The hidden layer outputs 10-bit values, and the output layer is correspondingly designed to handle this 10-bit data width. This ensures that both the multipliers

and adders within the data path unit consistently support 10-bit processing throughout the architecture.

Table 1: conventional FCN Vs modified FCN

| Parameters | Conventional FCN | | Modified FCN | |
|---|---|---|---|---|
| | Hidden unit | Output unit | Hidden unit | Output unit |
| Latency | n+22 | 2n+23 | n+8 | 2n+8 |
| Multipliers | 64 | 64 | 8 | 4 |
| Adders | 64 | 64 | 8 | 6 |
| Delay | t | t | t+24% | t+34% |
| Area | 100% | 100% | 25% | 12.5% |

## VII. CONCLUSION

This work introduces a modified Fully Connected Network (FCN) architecture, specifically tailored for improvements in area efficiency, power consumption, and processing speed. The design incorporates intermediate memory elements and implements pipelining strategies to optimize throughput and reduce latency. Integrated data flow control logic ensures the synchronization of data and weights, which contributes to further latency minimization. The parallelized structure of the architecture achieves a careful balance between resource utilization and computational speed. Overall, this approach is highly suitable for high-speed convolutional neural network (CNN) applications and can effectively function as a hardware accelerator for both deep neural network (DNN) and CNN workloads

## REFERENCES

[1]. Wu B, Wu X, Li P, Gao Y, Si J, Al-Dhahir N. Efficient FPGA Implementation of Convolutional Neural Networks and Long Short-Term Memory for Radar Emitter Signal Recognition. Sensors (Basel). 2024 Jan 30;24(3):889. doi: 10.3390/s24030889. PMID: 38339606; PMCID: PMC10857097

[2]. Hichen Wang, Hengyi Li, Xuebin Yue, Lin Meng,,"Briefly Analysis about CNN Accelerator based on FPGA",Procedia Computer Science,Volume 202,2022,Pages 277-282,ISSN 1877-0509, https://doi.org/10.1016/j.procs.2022.04.036..

[3]. Master Thesis "ZynqNet: An FPGA-Accelerated Embedded GitHub, shttps://github.com/dgschwend/zynqnet

[4]. Aabha Jain, Neha Sharma,"Accelerated AI Inference at CNN-Based Machine Vision in ASICs: A Design Approach", Published in ECS Transactions 24 April 2022 . https://www.semanticscholar.org/paper/183fc8eba11c4a30f902aa34d2d8ad108bd3bb14

[5]. Marco Rios, Flavio Ponzina, Alexandre Levisse, Giovanni Ansaloni, David Atienza,"Bit-Line Computing for CNN Accelerators Co-Design in Edge AI Inference"12th sep,2022, https://arxiv.org/pdf/2209.06108

[6]. Zhang, Y., Wang, X., & Li, J. (2020). "Parallel Processing Techniques for Deep Learning: A Survey." Journal of Parallel and Distributed Computing, 139, 1-15.

[7]. Chen, L., Zhang, H., & Liu, S. (2019). "Reducing Redundancy in Neural Networks: A Weight Vector Approach." IEEE Transactions on Neural Networks and Learning Systems, 30(5), 1450-1462.

[8]. Kumar, R., & Singh, A. (2021). "Pipelining Techniques for Accelerating Neural Network Inference." ACM Transactions on Architecture and Code Optimization, 18(3), 1-25.

[9]. Lee, J., Kim, T., & Park, H. (2022). "Optimizing Fully Connected Neural Networks with Redundancy Reduction and Pipelining." Neural Processing Letters, 54(2), 1235-1250.

[10]. Patel, V., Gupta, R., & Sharma, N. (2023). "Evaluating Trade-offs in Neural Network Architectures: A Comprehensive Framework." Journal of Machine Learning Research, 24(1), 1-30.