

DEVELOPMENT OF MOBILENET V1 USING FPGA

Dr.Jenitha A

Department of Electronics & Communication Engineering

Dr. T Thimmaiah Institute of Technology KGF ,India

Jenitha@drttit.edu.in

Prof.TamilVani R

Department of Electronics & Communication Engineering

Dr. T Thimmaiah Institute of Technology KGF ,India

tamilvani@drttit.edu.in

Prof. Mohana C

Department of Electronics & Communication Engineering

Dr. T Thimmaiah Institute of Technology KGF ,India

mohana@drttit.edu.in

Prof.Jeevika Mary

Department of Electronics & Communication Engineering

Dr. T Thimmaiah Institute of Technology KGF ,India

jeevika@drttit.edu.in

Abstract: Mobile Net V1 is a compact and efficient convolutional neural network (CNN) architecture designed for mobile and embedded systems, where power and processing capabilities are often limited. To address the challenges posed by standard convolutions which are computationally expensive. Mobile net V1 adopts depth wise separable convolutions, a technique that significantly reduces both the model size and Computational load. This method breaks the convolution process into two distinct steps, a depth wise convolution, which applies one filter per input channel and a point wise convolution that uses 1x1 Kernels to merge the depth wise outputs across channels. This separation minimizes computation while preserving accuracy, making it suitable for real time tasks such as object detection and image classification on edge devices. In this project, we present a hardware implementation of the key building blocks of Mobile Net V1 using Verilog HDL. Particularly the design was placed on the optimization of the depth wise and point wise convolution modules at the hardware level. We used the Vivado 2018.2 design suite for simulation and functional verification. The results confirm that our design is capable of performing neural network inference directly on FPGAs, providing a power efficient and practical solution for at the edge. This work underscores the value of hardware acceleration in enabling deep learning on low-resource platform and performed synthesis, implementation further the code was dumped in the hardware, the target device was KC705 Kintex family.

Keywords: MobileNetV1, Verilog HDL, Hardware Acceleration, Vivado Simulation, Convolutional Neural Network (CNN), FPGA.

I. INTRODUCTION

All Mobile Net V1 is a CNN (Convolutional Neural Network)[1] model introduced by Google facilitate efficient deep learning on devices with limited hardware capabilities such as smart phones and embedded Systems. Conventional CNNs, although highly effective, demand substantial computational power and memory, which makes them less suitable for deployment on low- power platforms.

Mobile Net V1 addresses these constraints by utilizing depth wise separable convolutions, a technique that decomposes standard convolutions into smaller, more manageable operations. This significantly reduces both the number of parameter and the overall computational load. At the heart of Mobilenet V1's efficiency is the separation of a full convolution in to two steps: A depth wise convolution that filters each input channel individually and a point wise convolution (1x1 convolution) that integrates the output across channels[9]. This approach reduces complexity by nearly an order of magnitude, while maintaining a level of accuracy suitable for many real world applications such as image classification, face detection and object recognition because of this balance between efficiency and performance. Mobile Net V1 is particularly well-suited for hardware acceleration FPGA's and other low power platforms where real time inference and energy conservation key priorities.

II. METHODOLOGY

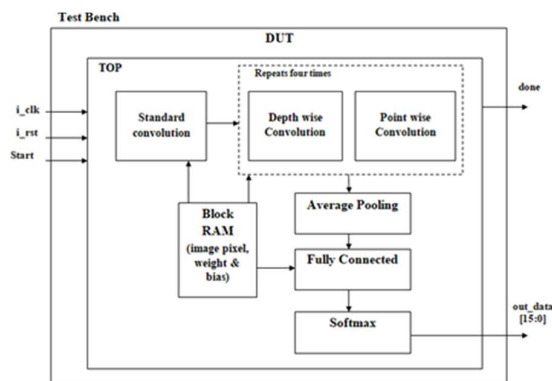


Fig1: General Block diagram of Mobile net V1

The given Fig 1 showcases a hardware-based implementation of a CNN (Convolutional Neural Network)[6] accelerator, most likely deployed on an FPGA. It outlines the core functional blocks necessary for running inference on a CNN model. The modular and layered structure suggests the hardware is tailored for a lightweight architecture, such as Mobile Net, which is known for its efficiency due to the use of standard, depth wise and point wise convolutions. Surrounding the entire setup is a Test Bench, which is used for simulation and verification purposes.

The Test Bench acts as a controlled environment to validate the behavior of the core design, referred to here as the Design Under Test (DUT). It provides key control signals like *i_clk*(clock), *i_rst*(reset), and Start, which are typical in

synchronous systems. It observes outputs like `done` (which flags the end of processing) and `out_data[15:0]` (which holds the classification result). The test bench itself doesn't perform computations but ensures the DUT functions correctly under various test conditions.

At the heart of the DUT is a top-level module labeled TOP, which orchestrates the data flow and manages interactions between processing blocks and memory units. Inputs such as image data, weights, and biases are stored in Block RAM (BRAM), which provides high-speed, concurrent access to multiple data streams essential in CNN workloads where large amounts of data must be accessed rapidly and often in parallel.

The processing pipeline begins with a Standard Convolution block. This component performs typical 2D convolution using 3×3 filters over the input feature maps. It slides kernels across the image, applying multiply-accumulate operations, then adds biases and applies activation functions like ReLU. In hardware, this step is often optimized for pipelining and reuse to maximize throughput.

After the initial convolution, the data flows into a repeated sequence of Depthwise Convolution[3] followed by Pointwise Convolution, repeated four times. These two stages together form the basis of depthwise separable convolutions, a major innovation in efficient CNNs like MobileNet. The Depthwise Convolution applies a filter to each channel individually, which significantly cuts down the number of operations. The Pointwise Convolution[4] then performs 1×1 convolutions across all channels, mixing the per-channel results into new feature representations. Repeating this block multiple times allows the system to learn more abstract features while keeping computational cost low.

Following these convolutional layers, the architecture includes an Average Pooling module. This unit reduces the spatial size of the feature maps by averaging values in small local regions. Pooling helps in reducing the amount of data to process in later layers and adds some robustness to feature location shifts. In hardware, average pooling is efficient as it involves simple arithmetic operations that can be easily pipelined.

The next stage is a Fully Connected (FC) layer. It takes the pooled features, flattens them, and applies a dense layer using learned weights and biases to produce a fixed-size output vector. This layer acts as a classifier, translating extracted features into class scores.

Lastly, a Softmax unit transforms the raw output values (logits) from the FC layer into normalized probability values. Each value in the result represents the likelihood of the input image belonging to a particular class. In fixed-point hardware implementations, softmax can be challenging due to its reliance on exponential functions and division, but it's typically approximated using lookup tables or streamlined arithmetic.

Once the softmax completes, the classification result is presented at the `out_data[15:0]` port, and the `done` signal is asserted to indicate the end of inference. The output might represent the predicted class index or a probability score, depending on how the result is encoded.

Table1: Layers details of Mobile net V1

Layer No.	Layer Type	Input Size	Kernel Size	Strid e	Paddi ng	Outp ut Size
1	Standard Convolution	32x32 x3	3x3x3x 32	2	1	16x16 x32
2	Depthwise Conv_1	16x16 x32	3x3x32	1	1	16x16 x32
3	Pointwise Conv_1	16x16 x32	1x1x32 x64	1	-	16x16 x64
4	Depthwise Conv_2	16x16 x64	3x3x64	2	1	8x8x6 4
5	Pointwise Conv_2	8x8x6 4	1x1x64 x128	1	-	8x8x1 28
6	Depthwise Conv_3	8x8x1 28	3x3x12 8	1	-	8x8x1 28
7	Pointwise Conv_3	8x8x1 28	1x1x12 8x128	1	-	8x8x1 28
8	Depthwise Conv_4	8x8x1 28	3x3x12 8	2	-	4x4x1 28
9	Pointwise Conv_4	4x4x1 28	1x1x12 8x256	1	-	4x4x2 56
14	Global Avg Pooling	4x4x2 56	4x4	-	-	1x1x2 56
15	Fully Connected	1x1x2 56	1x1x25 6	-	-	1x1x1 0
16	Softmax	1x1x1 0	-	-	-	1x1x1 0

Overall, this architecture is highly modular, making each processing block reusable and easy to debug or extend. The design's emphasis on repeated depth wise and point wise

convolution modules underlines its alignment with efficient models like Mobile NetV1. With all data housed in BRAM, the need for external memory is reduced, which conserves power and boosts speed—ideal for embedded and low-power AI applications.

The Mobile Net V1 accelerator begins with a



Fig2: Python Predicted output

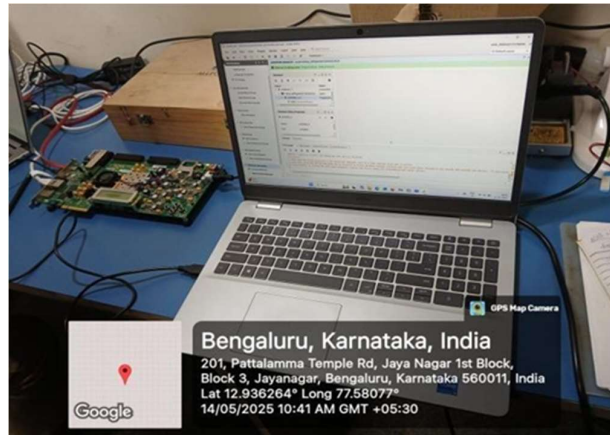


Fig4: Mobile net V1 implemented using KC705 Evaluation board

Standard Convolution layer that processes a $32 \times 32 \times 3$ input using 32 filters of size $3 \times 3 \times 3$ with stride 2 and padding 1, producing a $16 \times 16 \times 32$ output. This is followed by a Depth wise Convolution that operates on the $16 \times 16 \times 32$ input with 32 individual 3×3 filters (one per channel), maintaining the $16 \times 16 \times 32$ output shape. A Point wise Convolution then applies $1 \times 1 \times 32$ filters to the same

$16 \times 16 \times 32$ input, expanding the output to $16 \times 16 \times 64$. These depthwise and pointwise layers are repeated with appropriate input/output adjustments. After four such blocks, the final feature map is reduced via Global Average Pooling from $4 \times 4 \times 256$ to a compact $1 \times 1 \times 256$ vector. This vector is fed into a Fully Connected layer with a 10×256 weight matrix, resulting in a 10-element output vector. Finally, a Softmax layer converts this vector into class probabilities for classification.

III IMPLEMENTATION

Mobile Net V1[2][5] was coded using Verilog language, a separate modules were created including a top module. The weights, pixel values and bias values

was extracted using python pretrained model and the values was saved in BRAM. In the design 3 BRAM were used for read and write configuration and a wrapper was created for FPGA Dumping. Once the simulation is performed and verified its functionality, synthesis and implementation was completed using a constraint file. The clock frequency used was 200Mhz and since the FPGA used 100Mhz, the clock frequency was scaled down to 100Mhz using a clocking wizard. The target device was KC705, the code was dumped into FPGA and verified the output using GPIO LED.

V CONCLUSION

In this paper, we presented the hardware-level design and simulation of the Mobile Net V1 architecture using Verilog HDL, focusing specifically on the implementation of depth wise and point wise convolution operations. MobileNetV1 is well-regarded for its efficient structure, which replaces standard convolutions with depth wise separable convolutions, significantly reducing the number of parameters and computational load. This design choice makes it highly suitable for low-power and real-time applications on mobile and embedded platforms. Our implementation aimed to validate the feasibility of deploying such efficient neural network models on digital hardware. Using Vivado 2018.2, we simulated and functionally verified the Verilog modules corresponding to the core operations of

MobileNetV1. The simulation results confirmed the correctness of the convolution operations, including bias addition and activation functions, using fixed-point arithmetic to reflect hardware constraints. Although the design was not deployed on a physical FPGA, the successful simulation demonstrates that the architecture can be effectively mapped to hardware, serving as a foundational step toward actual deployment. This work highlights the potential of hardware-friendly neural network architectures for embedded AI applications. By implementing and verifying the Mobile Net V1 model at the RTL level, we gained insights into the challenges and considerations of hardware-based deep learning. Future work could involve synthesizing the design for FPGA, analyzing resource utilization, and testing performance under real-time constraints. Additional enhancements such as pipelining, quantization, and memory optimization would further improve the design's suitability for practical low-power AI systems.

VI RESULTS

Mobile net V1 was implemented using Verilog language and executed using FPGA–KC705 Evaluation board[7]. In the Fig 2 represents the output extracted from python which is used as input sand Fig 3 depicts the simulated waveform which matches the python output and Verilog simulated output and in Fig 4 FPGA implemented picture is depicted, which consists of PC, JTAG cable that connects FPGA kit and P Canda KC705 Evaluation board and output is verified.

REFERENCES

- [1]. Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications” 17th April 2017.

- [2]. J. Liao et al “Design of accelerator for MobileNet convolutional neural network based on FPGA” Electronics(Basel) 2021.
- [3]. BaiL, ZhaoY, Huang X “ACNN Accelerator on FPGA Using Depth wise Separable Convolution [J]” IEEE Transactions on Circuits & Systems II Express Briefs, 2018.
- [4]. Binh-SonHua, Minh-Khoi Tran, Sai-Kit Yeung “Pointwise Convolutional Neural Networks” Singapore University of Technology and Design in 2017.
- [5]. Safa Bouguezzi et al., “An Efficient FPGA-Based Convolutional Neural Network for Classification: Ad-MobileNet” Electronics 2021.
- [6]. T. Yuan, W. Liu, J. Han and F. Lombardi, "High Performance CNN Accelerators Based on Hardware and Algorithm Co- Optimization,"in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 1, pp. 250-263, Jan. 2021.
- [7]. L. Bai, Y. Zhao and X. Huang, "A CNN Accelerator on FPGA Using Depth wise Separable Convolution, "in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 65,no.10,pp.1415- 1419,Oct.2018.
- [8]. W. You and C. Wu, "RSNN: A Software/Hardware Co- Optimized Framework for Sparse convolutional Neural Networks on FPGAs,"in IEEE Access, vol.9, pp.949-960, 2021.
- [9]. [9G. Dinelli, G. Meoni, E. Rapuano and L. Fanucci, "Advantages and Limitations of Fully on Chip CNN FPGA- Based Hardware Accelerator,"2020 IEEE International Symposium on Circuits and Systems (ISCAS), 2020.