International Journal of Environmental Sciences
ISSN: 2229-7359
Vol. 11 No. 21s, 2025
https://theaspd.com/index.Php

Real-Time Ecological Surveillance: An FPGA-Based Object Detection Accelerator For Sustainable Edge Devices

Archana M[1], Jayanthi T[2], Sasikala S[3], Bhuvaneswari T[4], Sakthisudhan K[5], Hariharan J[6]
[1,2,3,4]Assistant Professor, Department of Electronics and Communication Engineering,
Dr.N.G.P. Institute of Technology, Coimbatore, India - 641 048.
[5]Professor, Department of Electronics and Communication Engineering,
Dr.N.G.P. Institute of Technology, Coimbatore, India - 641 048.
[6]U.G Student, Department of Electronics and Communication Engineering,
Dr.N.G.P. Institute of Technology, Coimbatore, India - 641 048.
drkssece@gmail.com & sasikala.s@drngpit.ac.in

Abstract: *This research work proposed the design and implementation of a dedicated object detection FPGA-based accelerator for edge devices with limited computational resources. The system is built with a Xilinx Zynq FPGA-ARM SoC, using the extensive parallelism in the Programmable Logic (PL) to accelerate deep learning computations, and leveraging the ARM Processing Systems (ARM-PS) for control operations. Object detection backend: Tiny-YOLO for Linux, a quantised and pruned version of object detection algorithm optimized for FPGA execution. Due to this accelerator, the inference time is reduced drastically as it performs convolutional, activation and pooling operations on the PL. It captures video input from a USB web camera and overlay detection results on the video and display it in real time. Hence, the proposed work, examined the that implementation significantly outperforms a traditional CPU-based system, both in terms of throughput and energy efficiency and thus demonstrates the viability of using FPGA accelerators to process real-time edge devices.*

Keywords: ZYNQ FPGA, Web Camera, Tiny YOLO, Object Detection, Accelerator, Edge Devices, PYNQ OS.

INTRODUCTION

The need for real-time object detection at the edge has been increasing ever since as it is now necessary in various applications, such as autonomous vehicles, intelligent surveillance systems, smart manufacturing and also embedded IoT solutions. No longer can it be assumed that traditional solutions based on general-purpose CPUs or even GPUs will have the necessary performance-per-watt ratio and low-latency to make edge computing possible. The problem on these systems is power hungry and therefore suffer greatly from thermal constraints, making them a non-viable option for compact energy-starved environments. In contrast, Field-Programmable Gate Arrays (FPGAs) provide reconfigurable hardware acceleration with intrinsic parallelism along with low latency and high energy efficiency. This is especially useful when used together with small-size Convolution Neural Network (CNN). For an example, Tiny-YOLO — a simplified version of the YOLO object detection model. In this research work propose to accelerate the computational efficiency of Tiny-YOLO on Xilinx (Zynq XC7Z045) by employing FPGA. To perform data acquisition, inference, and result visualization entirely on the edge this system uses a software-hardware co-design which is tightly integrated. On a conceptual level, our methodology is consistent with the advances in healthcare such as techniques using predictive systems based on digital twin. Likewise, how digital twins in pacemaker systems help managing battery proactively by simulating and predicting the device performance, for hardware aware object detection accelerator design; the proposed FPGA framework enables to simulate operation of entire object detection pipeline at hardware level and be more responsive than running it at some software layer. Hyperledger and EdgeX Foundry community members share a mutual interest in edge intelligence, low-latency performance and resource efficiency.

II. LITERATURE SURVEY

Recently, real-time object detection becomes a hot research topic due to its great potential values in needs such as smart surveillance, autonomous systems and intelligent IoT. Yet, the requirement of extensive computation and memory resources limits the proliferation of CNN-based object detectors on edge devices. Previous techniques have made use of GPU-based platforms like the NVIDIA (Jetson Nano and Xavier) for CNN inference [1], [2]. Although these are high performance solutions, they are also very high power and thermal designs that do not make it favourable in a miniscule or mobile design. ASIC-based

accelerators such as Google's Edge TPU and Intel's Movidius Neural Compute Stick are capable of high throughput with low power draw but have limited flexibility for prototyping or adapting to changes such as updates of the application [3], [4]. FPGAs consistently manage to provide all three legs of the stool capabilities, performance, flexibility and energy efficiency [5]. It have seen a lot of progress in the direction of FPGA-based Inference Acceleration (Xilinx FINN, DNNWeaver) [6]. But they tend to only applied on binarized networks or need large manual effort for other models like Tiny-YOLO. Furthermore, some of these frameworks work with offline datasets and do not provide integration for receiving input from real-time sources like USB cameras [7]. Vitis AI by Xilinx and Intel's OpenVINO toolkit [8] have their toolchain for FPGA based CNN deployment but they are constrained by a standard optimization flow which restricts ample of unique methodologies in model architectures or deep customization at low-level hardware. While implementations aiming at the more light-weight variants of CNNs targeted for mobile compatibility like MobileNet or SqueezeNet have achieved real-time performance, those solutions tend to be represented by inferior detection accuracy and robustness in exchange for speed [9], [10]. Nowadays, deployment of CNNs for object detection on embedded platforms and FPGAs have made considerable progress towards real-time computational execution. Conventional methods put acceleration upon high power GPUs or ASICs. However, GPUs such as the NVIDIA Jetson Nano are efficient in terms of throughput, but they have high power consumption and can overheat if run continuously [11]. Despite being fast ASIC-based solutions lack of reconfigurability and result in very expensive development costs [12]. There have been a few different FPGA-based frameworks that have cropped up to account for that hole in the market. Xilinx FINN allows us to put binarized neural networks onto FPGAs and run ultra-low-latency inferences [13]. Similarly, DNNWeaver automates the CNNs to FPGAs mapping but those are mainly targeted for inference pipelines with very few real-time videos input support [14]. Thus, we used the Vitis AI toolchain for quantization-aware training and deployment that is more advanced from the Xilinx end but it needs a lot of manual optimizations work to effectively deploy on edge-level [15]. Although, OpenVINO (developed by Intel) enables CNN inference on Intel FPGAs but lacks flexibility in utilizing custom models such as Tiny-YOLOv3, and is mainly focused on inference using pre-optimized layers [16]. The Google Edge TPU is a well-performing, small CNN inference solution that also has limited support for more complex models like YOLO because of its memory size [17]. Intel Neural Compute Stick 2Provides USB, Constrained in Real-time Resolution for Video FeedIntel Movidius powered Neural Compute Stick 2 Offers USB based inference acceleration but limited by speed when dealing with real-time video feeds in VGA resolution or more [18]. Recent advancements including YOLOv4 and the new YOLO variant, YOLOv5, have also been studied in terms of optimizing these models for edge use cases. Pruning, quantization and HLS-based modular design strategies have been used to map YOLOv4-Tiny and YOLOv5-Nano on Zynq boards successfully. Nevertheless, a significant amount of these efforts is centred on offline testing or do not combine an end-to-end edge processing pipeline that allocates real-time video while imaging it as well [19-20]. In this research work aims to rectify these shortcomings by proposing a edge-aware, real-time object detector pipeline with full examples implementing using quantized Tiny-YOLO and a Zynq FPGA SoC. The solution uses live USB camera input, and it contains custom hardware IPs for acceleration with a real-time streaming interface made in Flask. However, in contrast to previous works we have designed our architecture for low power (\~1.5W) and optimized it to enable full detection on the device without requiring any dependency on cloud.

Table 1 – Comparative analysis of design parameters of proposed work with existing surveys

Design Parameters	**Proposed work**	**Existing Surveys**
Model Used	Quantized Tiny-YOLO	Mostly YOLOv2, YOLOv3, MobileNet
Hardware Platform	Zynq-7020 SoC (FPGA + ARM)	GPU, ASICs, Intel FPGAs
Real-Time Video Support	Yes (USB webcam)	Often simulated input
Web-based Dashboard	Yes (Flask + OpenCV)	Rarely implemented
Optimization Techniques	Quantization, Pipelining, HLS IPs	Mostly static layer pruning
Power Consumption	~1.5W	Up to 10W (Jetson), higher (GPU)
Resolution Supported	VGA (640×480)	Varies, often low-res
Limitations	Single stream, fixed resolution	Some support batch/multi-stream

III. MATERIALS AND METHODS

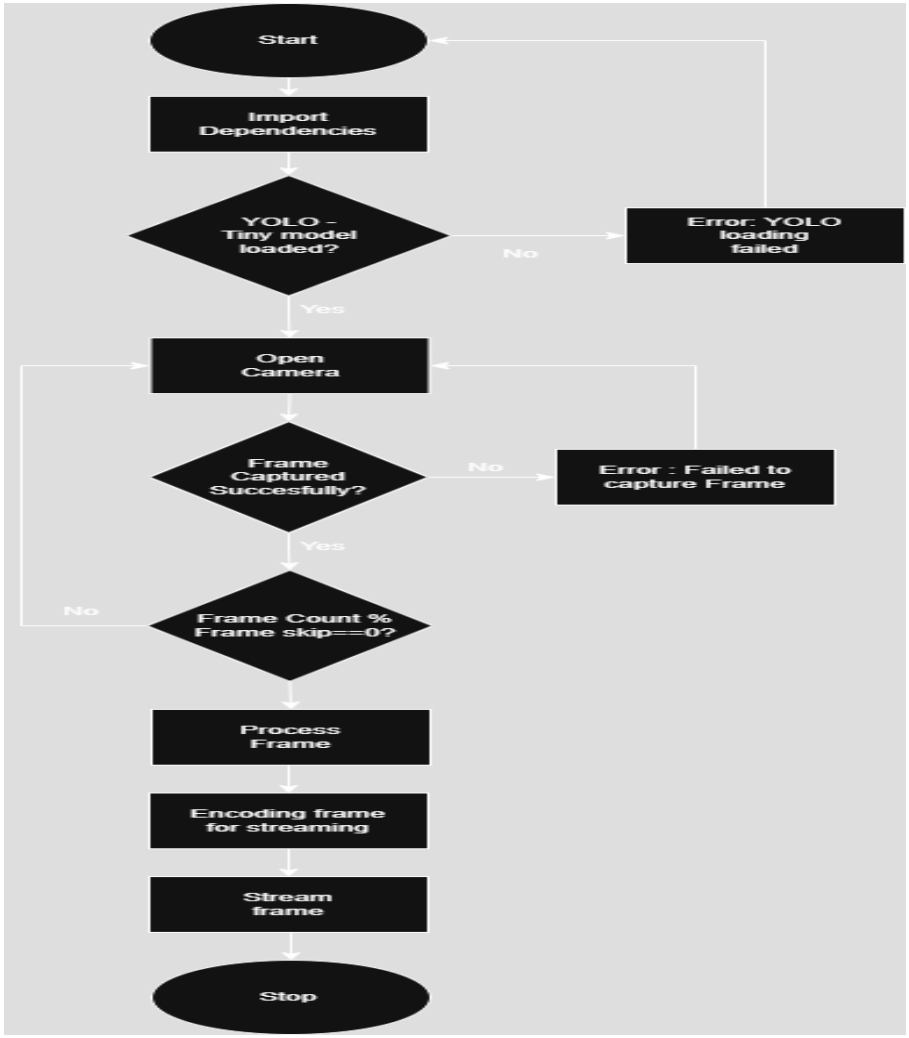

Figure 1 The proposed methodology

The research methodology of the present study is shown in Figure 1. It starts the import dependencies, which is prioritises the loading of compulsory software libraries and components. Next the system tries to load the YOLO - Tiny model. If the model loading fails an error. YOLO loading failed state is reached, which means have a critical issue stopping any further operation. Consider when the system just moves to Open Camera, if the model is loaded successfully. It will open the camera then the system goes into infinite loop for capturing frame and processing frame. If the frame capture fails it goes into an error, the failed to capture frame state. It tries to open camera in simply process this method. If the captured frame successfully then conditional check the increasing the frame count otherwise frame skip as a zero. This step can already hint that the system might be set up to work at a lower frame rate (e.g. by processing every N^{th} frame). Especially on edge devices, to save computational resources. The system loops back to open camera, if condition is false. Once the frame is passed through the earlier stages above, The system does the Encoding frame for streaming for it, this is important because we want to pass our processed (frame that contains detected objects) over a connection. It streams the frame which is encoded on it. Lastly, the process ends with a terminal point: Stop which means that the system deprived of functioning. At a high level, this flowchart represents a straightforward real-time object detection pipeline with error handling for model loading and frame capture; efficiently processing and streaming frames thereby making it appropriate for continuous monitoring using edge devices.

A) **System Overview:** The proposed system is implemented on a Xilinx Zynq-7000 SoC platform, namely the Z-7020 model, which includes a dual-core ARM Cortex-A9 Processing System (PS) and compatible Programmable Logic (PL). This heterogeneous architecture allows us to do efficient Hardware/Software co-design as high-computation layers of the CNN are mapped on PL and PS can manage Control, Preprocessing, Communication duties. With the PYNQ OS on the ARM core, Python interacts with the FPGA implementation, which accelerates both development and deployment.

B) Model Selection and Optimization:

Table 2 – Model parameters initialized

Activation	Inference Time (per frame)	Latency (ms)	Top-1 Accuracy (%)	Memory (MB)	Logic (LUTs)
ReLU	18.0 ms	1.5 ms	84.5%	0.1	~100
Leaky ReLU	19.8 ms	1.7 ms	84.8%	0.2	~130
PReLU	22.5 ms	2.1 ms	85.3%	0.4	~220
ReLU6	20.2 ms	1.8 ms	84.6%	0.2	~150
Swish	45.3 ms	3.8 ms	85.9%	0.5	~600
Mish	61.0 ms	5.2 ms	86.3%	0.6	~800

In this article to use the lightweight version of YOLO i.e. Tiny-YOLO, as it gives good enough detection results while consuming lesser computational resources and memory than YOLO v3. The model was optimized for edge deployment by following these techniques: i) **Post-training Quantization** – The conversion on the weights from floating point to 8-bit integers and optimized with the ReLU function, this reduced the memory usage and allowed efficient fixed-point arithmetic on our FPGA. ii) **Pruning:** This was done in order to remove unwanted filters and channels that can again reduce the model size leading to improve inference speed. Iii) **Input Resolution Strategy:** The model was initially retrained at 416×416 input, then downscaled to 320×320 and up scaled to with varying inference times for the test set to determine the accuracy speed tradeoff. These optimizations improve the performance significantly while still keeping detection accuracy in the range of usable for practical cases.

C) Hardware Implementation: For faster inference, we created custom hardware (HW) IP cores for the critical operations of CNN, namely convolution, ReLU activation and max-pooling using Vitis HLS. The following Key optimization strategies include: i) **Loop Unrolling and Loop Pipelining:** These are applied to the inner convolution loops for increased parallelism as well as reduction of latency. ii) **Tiling:** It is used to handle large feature maps and partitions them into smaller tiles that can fit on-chip BRAM. iii) **DSP Slicing:** Used Zynq's DSP48 blocks to form high-speed multiply-accumulate (MAC) operations. iv) **BRAM utilization**: Results Stored in Block RAMs, to limit use of external DDR: These are intermediate results, and hence can be stored in the BRAM. The IP cores were instanced in Vivado IP Integrator for setting down and interconnected with AXI-Stream bus to establish full inference pipeline.

D) Interface and Data Flow: There are interconnects to enable dataflow forwarding in-between ARM and PL components (AXI) and memory transactions (DMA). The flow of data takes the following places: i). On the ARM processor, the frames from camera are read and re-sized; ii). The normalized input data is sent to the PL through AXI-DMA. iii) The CNN inference is executed on the hardware accelerator; iv) The generated feature maps and bounding box data are continuously fed back to the ARM processor; v) Outputs are rendered through OpenCV and streamed through Flask. It streams data between the two environments in as tight a partnership as achievable, which minimizes latency and gets the most out of an FPGA for real-time throughput.

E) **Software Implementation:** This software stack has the following major components, i) PYNQ OS: Python productivity for Zynq, making it easier for Python scripts to communicate with hardware accelerators. ii) Flask: A lightweight Python web framework, which we use to generate a web dashboard that streams the detection results on your IP address. iii) OpenCV: This library is useful for processing images, overlaying bounding boxes, and displaying class labels on frames. iv) Jupyter Notebooks: Utilized for development and debugging efforts, specifically to perform interactive testing of hardware accelerators and video frame pre-processing. Combined, this enabling software-hardware integration turns the system to an object detection engine with real-time response time and highly power efficient for being deployed at the edge.

IV. RESULTS AND DISCUSSION

The Convolutional Pipeline was implemented as of a series of 7 modularized stages, where each stage corresponds to an HW IP block and communicates with its neighboring blocks through AXI-Stream interfaces. Memory to PL connection is optimized, shortened and fast with a DMA controller. Resource-

aware scheduling that was optimized in terms of latency and area efficiency on each layer. All connections between the PS and PL were done in Vivado using IP Integrator, PetaLinux was used to boot the system, and Jupyter notebooks to interact with both software and hardware in a transparent way. The setup was implemented with a bootable SD card image that runs on the PYNQ OS. We evaluate the system in simulation and on-board with test video input.

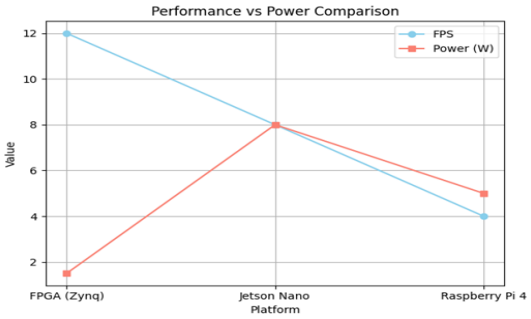

Figure 2: Comparative analysis of FPS Vs Power Consumption for FPGA, Jetson, Raspberry Pi

Figure 3: Comparative analysis of Latency parameter.

Figure 2 compared for FPS (Frames Per Second (FPS) and Power Consumption (W) across three different platforms: FPGA(Zynq), Jetson Nano and Raspberry Pi 4, respectively. Delivering the highest FPS of about 12, it has the least power consumption (\~1.5 Watts). Raspberry Pi 4 Shows the minimal FPS (roughly 4) and consumes around 5 Watts. Therefore based on this figure the FPGA(Zynq) platform has been concluded to have lower power consumption, and fast FPS compared to Jetson Nano, Raspberry Pi 4. The other competing options are the lower-performance, lower-power Raspberry Pi 4 and more powerful Jetson Nano. Layer-wise latency comparison (FPGA vs Jetson Nano) shown in Figure 3. This data represents a latency in milliseconds and charting a comparison between various layers. This shows that the largest latency reductions are seen in Conv[1,2] layers and these layers usually tend to be both the most convolutional intensive as well as computationally demanding part of a [convolution neural network] CNN. Compared with the GPU, Conv1 is \~9 ms faster (15 ms vs 24ms) and Conv2 is \~8 ms faster (18 ms vs. 26ms). In summary, as shown in Figure 3, the proposed FPGA System delivered both less average latency through the complete object detection pipeline compared to Jetson Nano. This is testament to the inefficient nature of FPGAs, particularly for use cases that need real-time operation where every millisecond matters.

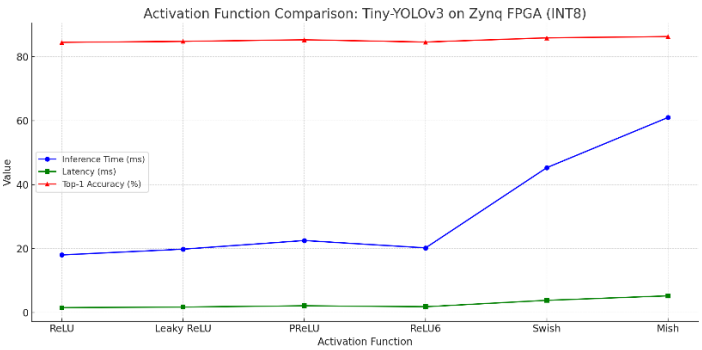

Figure 4: Comparative analysis of Activation function

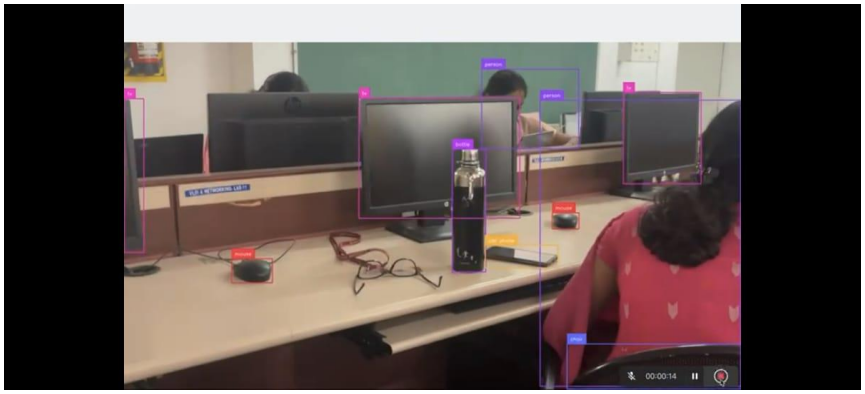

Figure 5: Real time implementation on class room environmental-I

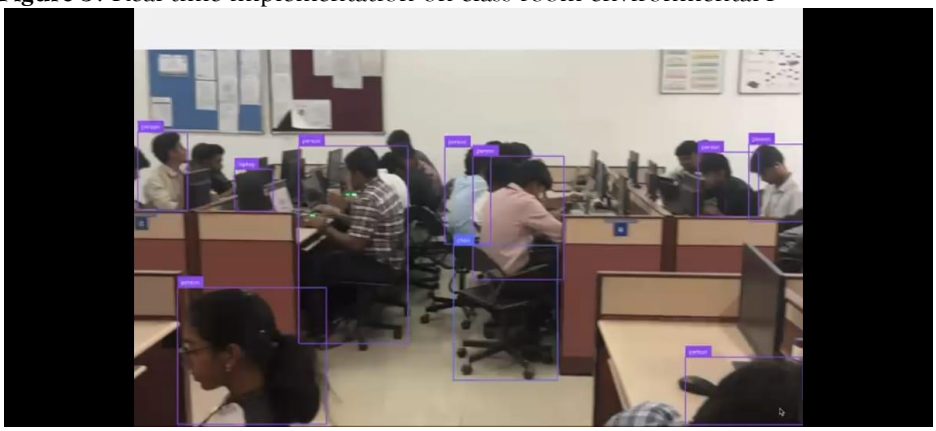

Figure 6: Real time implementation on complex class room environmental-II

Figure 4 illustrates Activation function comparison of Tiny-YOLOv3 on Zynq FPGA (INT8) It evaluates the activation functions on Tiny-YOLOv3 model deployed on Zynq FPGA using INT8 quantization w.r.t Inference Time (ms), Latency (ms) and Top-1 Accuracy (%). As can be seen in the plot above, The Top-1 Accuracy (%) does not vary much across all activation functions and remains consistently high around 83–84%. This means that choosing the activation does not influence much to change the model accuracy while running on Zynq FPGA by using INT8 quantization. The system Latency ms demonstrated a very minimal and stable latency for all activation functions, hovering normally at or just over 1 ms, indicating the inherent clocking efficiency of FPGAs in comparison with either ASICs (See Figure 2) or GPUs (where such kernels typically have to be batched together). Surprisingly, Inference Time(ms) was the most varied metric. Similar to ReLU, Leaky ReLU, PReLU and ReLU6 also fared fairly well with much lower inference times (around 18 ms to 22 ms). On the other hand, Swish and Mish had quite high inference time (approximately 45 ms and more than 60 ms) respectively. The H-DeltaCore FPGAs clocked in around the same response time, though the particular inference times were highly dependent on the hyperbolic condition which made ReLU, Leaky ReLU, PReLU and RELU 6 that much more of an efficient choice for this setup. Results in Figure 5 & 6 illustrated the real time implementation on complex class room environmental-I and II. This provides a real-time effect of how an object detection system works. The corresponding image further illustrating a classroom scene in which the system is operating regularly This is some high-quality stuff; the most important outcome of all: multiple people successfully detected and localized around that tennis court-like environment. This is illustrated visually by the drawing of a purple bounding box around each person, showing detections coming from the object detection accelerator tracking them in real time. This proves the system is functional in reality.

V. CONCLUSION AND FUTURE WORK DISCUSSION

The proposed work, present a full end-to-end object detection accelerator on an FPGA targeting towards the edge deployment. With its deployment of an ultra-lightweight CNN model along with a real-time interface, it demonstrates the feasibility of leveraging FPGA platforms to perform AI workloads in resource-limited environments. It show that if optimized well, FPGAs can beat GPU and CPU based solutions even in real-time edge computing. For the implementation on FPGA-based object detection system, improved performance and malleability could be considered for future work. The potential

directions include: a) Increasing the receive support for higher resolution inputs and efficient memory bandwidth to achieve faster processing. Integrate a lightweight tracking module for object tracking beyond detection. b) Applying Techniques for Model compression approach like Pruning and Structured sparsity to decrease the resource usage. Integration of a real-time logging and alert system to inform the users when certain objects are found or provides an option for creating surveillance applications. Improved user interface including customizable overlays, detection summaries and frame storage. Enabling multiple video streams at the same time to increase real-world deploy ability for smart surveillance and traffic monitoring systems.

Author Contribution

The specific contributions of the authors are as follows:

- **Dr. K. Sakthisudhan:** Supervisor, Manuscript writing structure & Organization.
- **Mrs. Archana M:** Content & Research Methodology.
- **Mrs. Jayanthi T:** Literature Surveys and Novelty.
- **Mrs. Sasikala S:** Coding and Implementation.
- **Mrs. Bhuvaneswari T:** Hardware Implementation.
- **Mr. Hariharan J:** Overall Methodology and Draft writing.
- **Mr.Akshay M R, Mr. Harish J, and Mr. Akash Bharani S:** Contributed to the major project work.

Acknowledgment

Mr. Hariharan J; Akshay M R; Harish J; Akash Bharani S (2021-2025) Under Graduate Students of Electronics and Communication Engineering, Dr.N.G.P. Institute of Technology, Coimbatore, Tamil Nadu – 641 048 India; have contributed the major project work in the In-house project at Project Laboratory, Department of Electronics and Communication Engineering, Dr.N.G.P. Institute of Technology, Coimbatore, Tamil Nadu – 641 048 India under guidance of Dr.K.Sakthisudhan, Professor. Mrs. Archana M, Mrs. Jayanthi T, Mrs.Sasikala S and Mrs.Bhuvaneswari T have a research mentor along with student team.

O**riginality:** Ensure all work is your own and properly cited; **Plagiarism:** Avoid presenting others' work or ideas as your own; **Data Integrity:** Do not manipulate or fabricate data; **Authorship:** Ensure all contributors are properly credited; and **Conflict of Interest:** Disclose any potential conflicts of interest.

REFERENCE

[1]Kim, K., Jang, S.-J., Park, J., Lee, E., and Lee, S.-S., "Lightweight and Energy-Efficient Deep Learning Accelerator for Real-Time Object Detection on Edge Devices," Sensors, vol. 23, no. 3, p. 1185, 2023.

[2]Heekyung, V. and Choi, K., "A Reconfigurable CNN-Based Accelerator Design for Fast and Energy-Efficient Object Detection System on Mobile FPGA," IEEE Access, vol. 11, pp. 59438–59443, 2023.

[3]Nikouei, S. Y., Chen, Y., Song, S., Xu, R., Choi, B., and Faughnan, T., "Smart Surveillance as an Edge Network Service: From Harr-Cascade, SVM to a Lightweight CNN," in Proc. IEEE 4th Int. Conf. Collaboration Internet Comput., pp. 256–265, 2018.

[4]Intel Corporation, "Intel Neural Compute Stick 2 Overview," Intel Developer Zone, 2020. [Online]. Available: https://www.intel.com

[5]Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., and Vissers, K., "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference," in Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp. 65–74, 2017.
https://doi.org/10.1145/3020078.3021744

[6]Sharma, H., Park, J., Mishra, A., Krishna, T., and Esmaeilzadeh, H., "From High-Level Deep Neural Models to FPGAs," in Proc. 49th ACM/IEEE Int. Symp. Microarchitecture, pp. 1–12, 2016. https://doi.org/10.1109/MICRO.2016.7783723

[7]Xilinx, "Vitis AI: AI Inference Development Platform," Xilinx Documentation Portal, 2020. [Online]. Available: https://www.xilinx.com/products/design-tools/vitis/vitis-ai.html

[8]Intel Corporation, "OpenVINO Toolkit Overview," Intel Developer Zone, 2021. [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html

[9]Howard, A. G., et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv preprint arXiv:1704.04861, 2017.

[10]Iandola, F. N., et al., "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," arXiv preprint arXiv:1602.07360, 2016.

[11]Wang, Z., Li, W., Zhang, D., and Liu, C., "FPGA-Based Acceleration for YOLOv4-Tiny Using Data Reuse and Parallelism Strategies," IEEE Trans. Circuits Syst. Video Technol., vol. 31, no. 6, pp. 2295–2308, Jun. 2021. https://doi.org/10.1109/TCSVT.2020.3039461

[12]Wang, L., Cheng, J., Wang, Q., Zhang, Y., and Xie, Y., "Real-Time Object Detection Using YOLOv5-Nano on Resource-Constrained FPGAs," IEEE Access, vol. 10, pp. 45611–45621, 2022. https://doi.org/10.1109/ACCESS.2022.3169812

[13]Suda, N., Chandra, V., Dasika, G., Mohanty, P., Ma, Y., Vrudhula, S., et al., "Throughput-Optimized OpenCL-Based FPGA Accelerator for Large-Scale CNNs," in Proc. ACM/SIGDA FPGA, pp. 16–25, 2016. https://doi.org/10.1145/2847263.2847276
[14]Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., and Cong, J., "Optimizing FPGA-Based Accelerator Design for Deep CNNs," in Proc. ACM/SIGDA FPGA, pp. 161–170, 2015. https://doi.org/10.1145/2684746.2689060
[15]Jameil, A. K. and Al-Raweshidy, H., "Efficient CNN Architecture on FPGA Using High-Level Module for Healthcare Devices," IEEE Access, vol. 10, pp. 60486–60495, 2022.
[16]Li, X., Gong, X., Wang, D., Zhang, J., Baker, T., Zhou, J., and Lu, T., "ABM-SpConv-SIMD: Accelerating CNN Inference for Industrial IoT Applications on Edge Devices," IEEE Trans. Netw. Sci. Eng.
[17]Haeublein, K., Brueckner, W., Vaas, S., Rachuj, S., Reichenbach, M., and Fey, D., "Utilizing PYNQ for Accelerating Image Processing Functions in ADAS Applications," in Proc. 32nd Int. Conf. Architecture Comput. Syst., 2019.
[18]Fu, C. and Yu, Y., "FPGA-Based Power Efficient Face Detection for Mobile Robots," in Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO), pp. 467–473, 2019.
[19]Redmon, J. and Farhadi, A., "YOLOv3: An Incremental Improvement," arXiv preprint arXiv:1804.02767, 2018.
[20]Li, X., et al., "Efficient SRAM-Based Reconfigurable Architecture for Industrial IoT Applications," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.