4

Closing the Gap Between AI Models and Silicon: Application Deployment for Next-Generation Edge Accelerators

Michal Szczepanski, Benoit Tain, Raphael Millet, Axel Farrugia, Cyril Moineau, and Sebastien Thuries

Université Paris-Saclay, CEA-List, France

Abstract

The deployment of deep neural networks on edge devices requires a carefully constructed pipeline flow from algorithm design to hardware execution. In this work, we present a complete application development pipeline targeting J3DAI, a low-power DNN accelerator integrated in a 3D-stacked CMOS image sensor. Starting from model training in PyTorch, we use the Aidge framework for post-training quantization and hardware-specific transformations, enabling the direct generation of binary files optimized for J3DAI execution. Our evaluation is performed using silicon-accurate simulation and gate-level netlists, providing hardware-realistic performance insights without relying on fabricated silicon. We highlight practical considerations in network design, including quantization trade-offs, tiling strategies, and the mapping of common operators to accelerator primitives. Beyond the current deployment flow, we discuss new perspectives for extending accelerator support, such as advanced operator sets, adaptive memory partitioning, and tighter hardware-software co-design. Together, these contributions demonstrate a robust pipeline that bridges machine-learning development with hardware accurate execution, offering a practical route toward real-time vision tasks directly on sensor in future silicon implementation.

Keywords: edge AI, low power, DNN accelerator, 3D-stacked CMOS, quantization.

4.1 Introduction and Background

The rise of edge computing has sharply increased the need to run deep neural networks (DNNs) directly on small, resource-limited devices. Many modern applications, from real-time computer vision to autonomous systems and privacy-focused analytics depend on fast, low-latency inference while operating within strict power limits. However, bringing DNN models from development frameworks to actual silicon hardware at the far edge remains a complex and ongoing challenge.

Edge devices often operate under tight constraints: limited on-chip memory, low energy availability, and customized accelerator architectures that may only handle a fraction of common neural network operations. These accelerators tend to be optimized for specific tasks (for instance, chains of convolution layers) while processing others less efficiently (like residual connections). This imbalance can cause slowdowns due to data movement or fallback of unsupported operations. In addition, popular machine learning frameworks such as PyTorch and TensorFlow were not originally designed with hardware-aware deployment as a core priority, making the gap between model design and deployment even wider. Their abstractions often hide the underlying memory and computational costs, and do not allow the magnitude (float, int,) of intermediate values in different operators to be determined, causing a mismatch between software models and accelerator capabilities.

In this work, we present a deployment pipeline built around J3DAI, a tiny deep neural network accelerator integrated into a 3-layer 3D-stacked CMOS image sensor. The system features a dedicated AI chip in the bottom die, designed to execute quantized models under strict power and memory constraints. Our pipeline connects high-level model development to silicon execution: models trained in PyTorch are quantized and adapted using the Aidge framework, then exported into format optimized for hardware execution on J3DAI.

The main contributions of this work are as follows:

- We describe an end-to-end workflow for deploying DNNs on constrained accelerators, from PyTorch training through Aidge quantization to J3DAI execution.
- We introduce J3DAI as an ultra-compact edge AI solution, tightly integrated into a 3D-stacked image sensor for near-sensor processing.
- We provide experimental validation on both classification and segmentation tasks, demonstrating that the pipeline achieves efficient inference without significant loss of accuracy.

- We open a discussion on future perspectives, highlighting opportunities to expand operator support, improve memory utilization, and advance hardware–software co-design.

4.2 Related Work

4.2.1 Edge AI Accelerators and In-Sensor Processing

The increasing demand for low-latency vision has driven research into accelerators designed for edge inference. Conventional approaches include NPUs in mobile SoCs and external devices such as Google EdgeTPU and Intel Movidius, which provide efficient CNN acceleration but remain separated from the sensor. More recent work explores processing-in-sensor and 3D stacking techniques to bring computation closer to the pixel array. Examples include NeuroSensor [1] and 3D-stacked designs embedding convolutional logic image sensors [2–5]. These works demonstrate substantial reductions in data movement and energy consumption, though they typically support limited operator sets or shallow networks. The J3DAI architecture extends this trend by providing a programmable, quantization-aware accelerator tightly integrated into a 3-die CMOS stack [6].

4.2.2 Quantization and Deployment Frameworks

On the software side, multiple frameworks address model adaptation for constrained hardware. TensorFlow Lite [7], TensorRT [8] offer quantization and operator fusion for NPUs or CPUs, but they depend on backend-specific operator libraries and offer limited visibility hardware constraints. Recent surveys [9–10] highlight different strategies, advantages and the importance of post-training quantization (PTQ) for reducing the footprint of DNNs while maintaining accuracy highlighting its effectiveness as a practical deployment strategy.

Bridging ML models with accelerator execution requires careful hardware–software co-design. Compiler-based solution such as TVM [11] and Vitis AI [12] translate model to heterogeneous hardware through optimization passes for scheduling and memory management. Other efforts, such as PISA [13], integrate binarized or quantized compute into the sensor itself. Nevertheless, these approaches either target general-purpose accelerators or focus on limited model families limiting their adaptability.

The deployment tools are rarely designed for sensor-integrated accelerators. The Aidge framework fills this gap by enabling quantization and export

into a hardware-ready format, ensuring that models trained in PyTorch can be directly executed directly on J3DAI. By contrast, the proposed pipeline explicitly considers operator compatibility and SRAM footprint analysis, ensuring that the deployment flow is both hardware-aware and adaptable.

4.3 System Overview

The experimental study was conducted on the system referenced in [6]. Below is a brief description of its key components and configuration to provide context for the experimental results.

The system is a 3D-stacked CMOS image sensor, designed for low-power on-sensor deep learning inference. The stack is composed of three dies, each implementing a distinct function:

- **Top die:** a 12-megapixel RGB pixel matrix (4096×3072), responsible for image capture.
- **Middle die:** analog readout and image signal processing (ISP), a lightweight RISC-V host processor, and a portion of the shared L2 memory. This die also manages communication between the sensor and external host interfaces
- **Bottom die:** the dedicated J3DAI DNN accelerator, based on PNeuro [14] which integrates Neural Compute Blocks (NCBs), a multi-bank on-chip memory hierarchy, and data-movement units such as the Direct Memory Parallel Access (DMPA) engine. The accelerator is optimized for quantized neural networks and provides high throughput under strict area and power constraints.

During deployment, the RISC-V host orchestrates execution, loading PNeuro instructions into the instruction memory and managing scheduling across compute clusters. The NCBs execute supported layers in parallel, while the DMPA unit enables efficient transfer of weights and activations, mitigating the cost of data movement—a critical bottleneck in memory-limited accelerators.

The J3DAI accelerator was implemented as the bottom die of a 3D-stacked CMOS image sensor, enabling direct execution of quantized neural networks at the sensor level. The hardware integrates multiple Neural Compute Blocks (NCBs) connected through a high-bandwidth on-chip interconnect. Each NCB supports parallel multiply–accumulate (MAC) operations, activation functions, and pooling, while memory hierarchies (L1 cluster SRAM, shared L2 SRAM, and instruction/data SRAM) ensure efficient

tiling of large models. A key architectural feature is the Direct Memory Parallel Access (DMPA) engine, capable of transferring 1024 bits per cycle, which dramatically reduces the overhead of weight and activation movement compared to conventional DMA schemes.

This integration provides a streamlined path from high-level model definition to real-time hardware execution. Crucially, the workflow also includes compatibility and memory footprint checks, ensuring that the chosen model fits within the available instruction and data SRAM and does not rely on unsupported operators. As such, the system bridges the gap between software frameworks and silicon constraints, enabling practical deployment of edge vision applications directly within the sensor stack.

4.4 Deployment Pipeline

Deploying a deep neural network to the J3DAI accelerator requires an end-to-end flow that connects high-level model training with hardware-level constraints. The deployment pipeline is composed of four stages: model training, quantization, compatibility/memory analysis, and hardware execution.

Model Definition and Training (PyTorch).

Development begins in PyTorch, where models are defined, trained, and validated using conventional floating-point arithmetic (FP32). This stage benefits from the ecosystem of state-of-the-art vision architectures, datasets, and training tools. However, model design is not entirely hardware-agnostic: candidate architectures must be evaluated against the accelerator's supported operator set and memory hierarchy. In particular, operators must be structurally compatible with J3DAI execution, and activation buffers must fit within the available SRAM capacity. This consideration is especially critical for complex applications such as semantic segmentation or object detection, where diverse operators and large intermediate feature maps—driven by high input resolutions—can easily exceed hardware limits. By integrating these constraints during training, rather than postponing them to later stages as in many existing deployment flows, the pipeline establishes a hardware-aware feedback loop that reduces the risk of incompatibility and ensures that even complex models remain deployable on constrained accelerators.

Quantization and Export (Aidge).

The trained model is processed through the Aidge framework, which applies post-training quantization (PTQ) to reduce weights and activations to low-precision (e.g., int8). This process involves calibrating the model using a

representative dataset to determine optimal scaling factors for weights and activations, ensuring minimal loss of precision during quantization. Consequently, it significantly decreases the model's memory footprint, while preserving acceptable accuracy. Model quantization requires a careful examination of both linear and non-linear operators to ensure compatibility with the accelerator's supported execution set. For instance, in dense applications such as semantic segmentation, operators like bilinear interpolation—commonly used to restore input resolution—must to be replaced or restructured to meet hardware constraints. In such cases, this stage may trigger a feedback loop to the training process so that the architecture remains both accurate and deployable. Once validated, the quantized model is exported into the 8-bit graph format, a lightweight hardware-aware representation that encodes network topology in Aidge framework.

The export will retrieve the graph in 8-bit format along with the hardware configuration of the DNN accelerator. The hardware configuration mainly includes the number of available NCBs and the size of the associated L1 and L2 memories. This configuration and the graph are essential for placing the feature maps and network parameters in the accelerator's memories. A dedicated solver explores multiple mapping solutions to find the optimal placement. Based on this mapping, the export configures each advanced function specific to PNeuro, such as automatic indexes and multidimensional address generators.

Then, the export automatically generates the assembly code and the host code. The assembly code includes the PNeuro instructions. These instructions are compiled to obtain a binary code that can be loaded into the instruction memories. The host code, written in C, consists of the main source code that calls the functions of each layer of the network with an execution routine. Each layer routine begins by loading the program into the instruction memory, then loads the kernel parameters and the input data (if first layer or use of tiling). Subsequently, the execution of the accelerator is started and synchronized via a control register. In case of tiling, the multi-pass execution continues, saving the intermediate results in the global memory at every step.

All these steps have enabled the generation of the binary that can be loaded to execute the network on the target hardware.

Execution on J3DAI.

As physical hardware was not available during the development phase, hardware execution of the neural network was implemented using both a high-level simulator and a post-routed netlist. The high-level simulator, based

on an Instruction Set Simulator (ISS), enabled rapid and effective assessment of the model's functional performance at an early design stage. For strict cycle-accurate temporal analysis, a post-implementation netlist generated after place-and-route was employed, allowing detailed investigation of real-time circuit behavior and precise reproduction of hardware-level interactions. The combined use of these two complementary simulation methods provided robust and comprehensive validation of hardware execution despite the immediate unavailability of the physical device.

Overall, this pipeline enables a smooth transition from PyTorch training to silicon execution, while actively addressing the challenges of limited memory and selective operator support. By incorporating compatibility and memory checks into the deployment workflow, the system ensures robust and efficient operation for real-world edge applications.

4.5 Experimental results

To validate the proposed deployment pipeline, we conducted experiments on the J3DAI accelerator integrated within a 3D-stacked CMOS image sensor prototype. Evaluation focused on classification and segmentation tasks, emphasizing accuracy retention, inference latency, and energy efficiency under realistic operational conditions. All tests were performed with the J3DAI AI core operating at 200 MHz, processing input directly from the sensor interface to avoid off-chip data transfers.

4.5.1 Model Presentation

Three representative networks were selected to characterize the proposed deployment pipeline across both classification and dense-prediction tasks. The classification models: MobileNetV1 [15] and MobileNetV2 [16] were chosen for their compact structure and well-established efficiency in mobile-scale inference, while the third model implements a lightweight encoder–decoder segmentation network representative of real-time pixel-wise applications. For classification, pretrained weights provided by the torchvision library were used as the initial backbone. For segmentation, the architecture uses encoder (MobileNetV1 $\alpha = 0.5$), couple with a feature pyramid style decoder implemented as a linear combination of convolution block, trained from a scratch [17]. This configuration achieves a total integer model size below 4 MB, allowing full on-chip storage of weights and activations during inference. To ensure full compatibility with the J3DAI instruction

Table 4.1 Key performance metric of selected models

Model	**MobileNetV1**	**MobileNetV2**	**Segmentation**
MMACs	557	289	877
Image Input	256x192	256x192	512x384
Accuracy	62% Top-1	66% Top-1	36.4 mIoU
Latency @200MHZ	4.96 ms	4.04 ms	7.43 ms
Power @30FPS	47.6 mW	30.5 mW	63.8mW
Power efficiency	0.77 TOPS/W	0.62 TOPS/W	0.82 TOPS/W
MAC/Cycle efficiency	76.8%	46.6%	76.5%

set, for segmentation, bilinear interpolation layers—unsupported by the accelerator—were replaced with nearest-neighbour upsampling

Input dimensions were adjusted to fit the native 4:3 aspect ratio of the image sensor: 256 $\times$ 192 pixels for classification and 512 $\times$ 384 pixels for segmentation. This scaling preserves the natural field of view of the sensor while balancing resolution and computational load.

4.5.2 Experimental Validation

Under this configuration, the number of MAC operations for MobileNetV1 is 557 M, comparable to the classic 224 $\times$ 224 input format (569 M MACs). Similarly, MobileNetV2 requires 289 M MACs, close to the 300 M MACs used in the standard configuration. The MobileNetV2 architecture, however, introduces branching structures and depthwise–pointwise sequences, which increase data movement and reduce MAC-per-cycle efficiency, resulting in higher latency relative to its nominal compute cost.

Across classification tasks, quantization causes minimal degradation compared to FP32 baselines. MobileNetV1 achieved 62 % Top-1 accuracy, and MobileNetV2 reached 66 % Top-1, both within 1–2 % of their floating-point counterparts validated on ImageNet dataset. Inference latency ranged from 4.0 ms to 5.0 ms, corresponding to real-time throughput above 200 FPS, while maintaining $<$ 50 mW power consumption at 30 FPS. The resulting power efficiency to 0.77 TOPS/W demonstrates the suitability of J3DAI for always-on edge vision applications.

For the segmentation task, evaluation was performed on the Cityscapes dataset. The model archives 36.4 mIoU, below large state of art architectures, with only 877 MMACs, which other models work in orders of magnitude more parameters and are not deployable in such constrained environments. The reported performance therefore reflects an optimal balance between accuracy, memory footprint, and on-chip efficiency.

Because the input resolution of 512 × 384 differs from the Cityscapes size (1024 × 2048), we present two evaluation results in each Figure 4.1 and 4.2 below. Above is a cropped-input evaluation, in which smaller section of 512 × 384 pixels is extracted from validation images. This configuration preserved fine detail and allowed reliable detection of small elements such as traffic lights and pedestrians. Below we present full-frame resizing, where

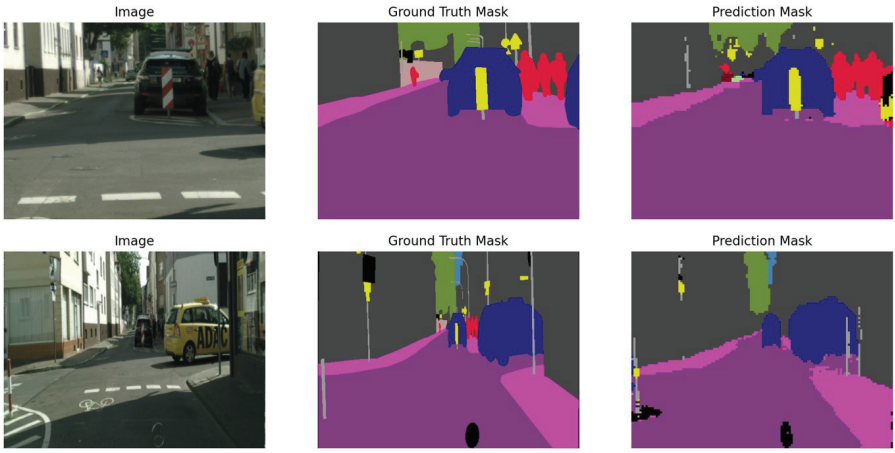

Figure 4.1 Input Image, Ground Truth Mask and Prediction for Segmentation Network in float32.

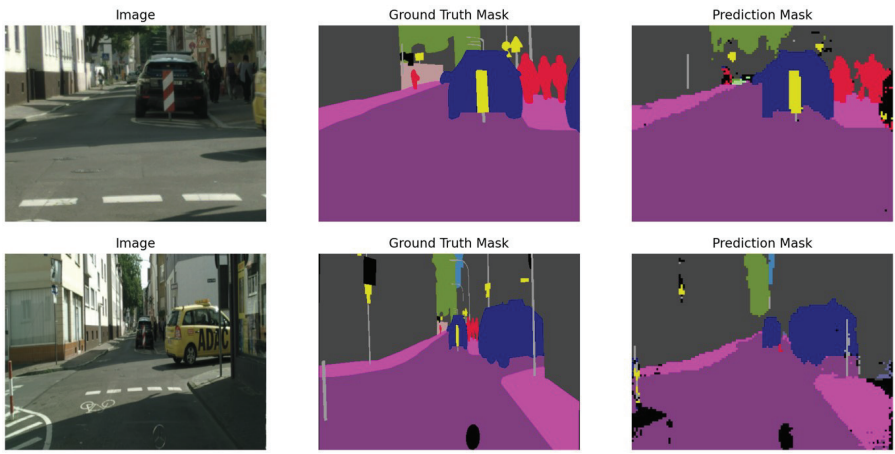

Figure 4.2 Input Image, Ground Truth Mask and Prediction for Segmentation Network in int8.

the entire image is uniformly scaled down to 512 $\times$ 384. This approach retained global context but led to a noticeable drop in small-object accuracy and an overall lower mIoU however allows to compare our method with others. Across both methods, quantization introduced less than 1.2 % relative loss in mIoU compared with the floating-point model, which is visible on prediction Mask for both Figures 4.1 and 4.2. It confirms that the Aidge PTQ process preserves segmentation quality while meeting the accelerator's resource constraints.

4.6 Perspectives & Discussion

The deployment results presented above demonstrate that the proposed PyTorch–Aidge–J3DAI pipeline provides an efficient bridge between algorithmic design and silicon execution for compact convolutional networks. However, the rapidly evolving landscape of edge AI applications and model architectures opens several avenues for future development on both software and hardware fronts.

The current J3DAI instruction set and Aidge toolchain primarily target CNNs, which remain dominant in near-sensor vision tasks. Yet, emerging architectures such as the Vision Transformer models are reshaping the state of the art in visual perception. Supporting these models on edge accelerators will require extending the operator set to include primitives such as multi-head self-attention, layer normalization, and softmax. Hardware implementation of these operators must preserve efficiency while maintaining the programmability required for future algorithmic flexibility. Integrating these capabilities would enable J3DAI to process a broader range of models, including ViT-based feature extractors and lightweight transformer decoders, thereby expanding its applicability beyond classical CNN-based pipelines.

Beyond classification and semantic segmentation, future work aims to extend support toward depth estimation, object detection, and optical flow extraction. Achieving these goals will also depend on hardware evolution. Augmented on-chip memory capacity, which would relax constraints on intermediate tensor buffering and enable bigger models to be mapped with tiling strategy.

4.7 Conclusions

In this work, we introduced an end-to-end deployment pipeline that connects deep neural network training in PyTorch with efficient execution

on the J3DAI near-sensor accelerator. The pipeline was designed to make deployment seamless, turning high-level models into hardware-ready binaries while taking into account real memory and operator constraints from the very beginning. By integrating hardware awareness directly into the training and quantization process, the approach avoids many of the late-stage compatibility issues that typically slow down deployment to edge devices.

Using the Aidge framework, models are quantized and optimized for the accelerator without sacrificing much accuracy, typically within 1–2% of the original floating-point versions. Experimental results obtained through simulation of the J3DAI hardware show that the deployed networks can achieve real-time performance with low latency and minimal power consumption, making them well-suited for embedded vision applications such as classification and segmentation directly on the sensor.

More broadly, this work demonstrates the importance of tight co-design between algorithms and hardware. Instead of treating deployment as an afterthought, the proposed flow treats it as part of the model development process, ensuring that each design decision remains compatible with hardware limits. The result is a robust and efficient system that simplifies the path from model design to silicon execution.

Looking ahead, this approach together with Aidge framework offers a strong foundation for future expansion. As AI models evolve, especially with the growing adoption of ViT architectures and more complex perception tasks, extending the J3DAI operator set and toolchain will open new opportunities for near-sensor intelligence. Overall, the proposed pipeline represents a practical and forward-looking step toward bringing high-performance, low-power AI directly to the edge.

Acknowledgements

This work was supported by the French National Research Agency (ANR) through the "Investissement d'avenir" (investments for the future) programs: ANR 10-AIRT- 005 (IRTNANO-ELEC) and by the Neurokit2E European Project (GA101112268). Furthermore, the Aidge developers are gratefully acknowledged.

References

[1] M. F. Amir, D. Kim, J. Kung, D. Lie, S. Yalamanchili, and S. Mukhopadhyay, 'NeuroSensor: A 3D image sensor with integrated

neural accelerator', in *2016 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, Burlingame, CA, USA: IEEE, Oct. 2016, pp. 1–2. doi:10.1109/S3S.2016.7804406.

[2] T. Haruta *et al.*, '4.6 A 1/2.3inch 20Mpixel 3-layer stacked CMOS Image Sensor with DRAM', in *2017 IEEE International Solid-State Circuits Conference (ISSCC)*, San Francisco, CA, USA: IEEE, Feb. 2017, pp. 76–77. doi:10.1109/ISSCC.2017.7870268.

[3] K. Kiyoyama, Q. Zhengy, H. Hashimoto, H. Kino, T. Fukushima, and T. Tanaka, 'Development of a CDS Circuit for 3-D Stacked Neural Network Chip using CMOS Analog Signal Processing', in *2019 International 3D Systems Integration Conference (3DIC)*, Sendai, Japan: IEEE, Oct. 2019, pp. 1–4. doi:10.1109/3DIC48104.2019.9058856.

[4] P. Vivet *et al.*, 'Advanced 3D Technologies and Architectures for 3D Smart Image Sensors', in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Florence, Italy: IEEE, Mar. 2019, pp. 674–679. doi:10.23919/DATE.2019.8714886.

[5] P. Vivet *et al.*, 'Advanced 3d Design and Technologies for 3-Layer Smart Imager', in *2022 International Symposium on VLSI Technology, Systems and Applications (VLSI-TSA)*, Hsinchu, Taiwan: IEEE, Apr. 2022, pp. 1–2. doi:10.1109/VLSI-TSA54299.2022.9771026.

[6] B. Tain *et al.*, 'J3DAI: A tiny DNN-Based Edge AI Accelerator for 3D-Stacked CMOS Image Sensor', in *2025 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Reykjavík, Iceland: IEEE, Aug. 2025, pp. 1–7. doi: 10.1109/ISLPED65674.2025.11261786

[7] R. David *et al.*, 'TensorFlow Lite Micro: Embedded Machine Learning on TinyML Systems', 2020, *arXiv*. doi:10.48550/ARXIV.2010.08678.

[8] E. Jeong, J. Kim, and S. Ha, 'TensorRT-Based Framework and Optimization Methodology for Deep Learning Inference on Jetson Boards', *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 5, pp. 1–26, Sep. 2022, doi:10.1145/3508391.

[9] Z. Yuan *et al.*, 'Benchmarking the Reliability of Post-training Quantization: a Particular Focus on Worst-case Performance', 2023, *arXiv*. doi:10.48550/ARXIV.2303.13003.

[10] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, 'A Survey of Quantization Methods for Efficient Neural Network Inference', 2021, *arXiv*. doi:10.48550/ARXIV.2103.13630.

[11] T. Chen *et al.*, 'TVM: An Automated End-to-End Optimizing Compiler for Deep Learning', 2018, *arXiv*. doi:10.48550/ARXIV.1802.04799.

[12] A. Ushiroyama, M. Watanabe, N. Watanabe, and A. Nagoya, 'Convolutional neural network implementations using Vitis AI', in *2022 IEEE 12th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA: IEEE, Jan. 2022, pp. 0365–0371. doi: 10.1109/CCWC54503.2022.9720794.

[13] S. Angizi, S. Tabrizchi, and A. Roohi, 'PISA: A Binary-Weight Processing-In-Sensor Accelerator for Edge Image Processing', 2022, *arXiv*. doi:10.48550/ARXIV.2202.09035.

[14] A. Carbon *et al.*, 'PNeuro: A scalable energy-efficient programmable hardware accelerator for neural networks', in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany: IEEE, Mar. 2018, pp. 1039–1044. doi:10.23919/DATE.2018.8342165.

[15] A. G. Howard *et al.*, 'MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications', 2017, *arXiv*. doi:10.48550/ARXIV.1704.04861.

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, 'MobileNetV2: Inverted Residuals and Linear Bottlenecks', 2018, doi: 10.48550/ARXIV.1801.04381.

[17] V. T. Quyen, J. H. Lee, and M. Y. Kim, 'Enhanced-feature pyramid network for semantic segmentation', in *2023 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, Bali, Indonesia: IEEE, Feb. 2023, pp. 782–787. doi:10.1109/ICAIIC57133.2023.10067062.