information

MDPI

*Article*

# CNN Architecture Optimization for Multiplierless Inference

**Ivan Al Khayat, John Reuben * and Dietmar Fey**

Chair of Computer Architecture, Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), 91054 Erlangen, Germany; ivan.al@fau.de (I.A.K.)
* Correspondence: johnreuben.prabahar@fau.de

**Abstract**

Many CNN architectures are primarily optimized for classification accuracy and software-level metrics, but they do not necessarily account for resource-constrained integer inference. This work addresses this gap by optimizing CNN architectures with subsequent In-Memory Computing (IMC) implementation in mind, using an analytical resource–cost model for multiplierless integer inference. Motivated by the need to reduce data movement and avoid explicit multipliers, the proposed methodology constrains candidate CNN architectures toward integer computations that can be expressed through memory look-ups, additions, and shifts, rather than conventional multiply–accumulate operations. Distributed Arithmetic provides the computational basis for this multiplierless formulation, while the proposed HWCost model is used together with validation performance and validation efficiency to rank and select candidate architectures. Candidate architectures are generated by combining six redesign techniques: Global Average Pooling (GAP) to reduce classifier complexity, Quantization-Aware Training (QAT) to optimize weight precision, reduction in convolution filter size, selection of the downsampling strategy, power-of-two constraints on the feature map size before GAP, and training oriented toward integer-only inference. We evaluate the combined effect of these techniques on MNIST, FashionMNIST, CIFAR-10, and PneumoniaMNIST. The results show that, across MNIST and FashionMNIST, the selected compact models reduce the estimated HWCost by up to 2.52× with approximately one percentage point or less degradation in test accuracy. On CIFAR-10, the lowest-cost candidate reduces the estimated HWCost by 3.94×, relative to the highest-accuracy candidate, at the cost of a larger accuracy reduction. On PneumoniaMNIST, the efficiency-oriented candidate achieves comparable independent-test-balanced accuracy to the highest-validation-balanced accuracy candidate while requiring substantially lower estimated HWCost.

**Keywords:** CNN; neural network; multiplierless; shift–add; convolution; Distributed Arithmetic; optimization

Check for updates

Academic Editors: Nikolaos Mitianoudis and Ilias Theodorakopoulos

Received: 30 July 2026
Revised: 8 September 2026
Accepted: 12 September 2026
Published: 17 September 2026

**Copyright:** © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

## 1. Introduction

Artificial Intelligence and the rise of Deep Neural Networks (DNNs) have changed the way we live. Convolutional Neural Networks are a class of DNN whose application is mainly for image classification, object detection and the like. The field of DNN design is fast evolving with many architectural refinements and innovation for increasing the accuracy of NN. However, there is a gap in the research between "high-accuracy" networks and their hardware implementation. Many high-accuracy neural networks are computationally elegant on software but inefficient, memory-heavy, or power-hungry in real hardware. When

a CNN architecture is developed, neural network architects focus mostly on accuracy and FLOPS (Floating Point Operations Per Second). When CNNs are considered for resource-constrained inference, aspects beyond classification accuracy, such as energy consumption, memory access frequency, arithmetic precision, and parameter storage, become important architectural considerations. Therefore, a CNN optimized mainly for accuracy by increasing the depth, width, or number of trainable parameters may still have high computational and memory-related complexity. This motivates CNN architecture optimization strategies that explicitly consider both classification performance and estimated resource costs.

To set the research context concretely, we first present some energy costs and elaborate on why redesign is necessary. The main components of classical CNNs are orderly sequences of three operations to achieve the desired accuracy in the recognition of an image: convolution, max-pooling and a fully connected layer (usually functions as classifier). The convolution operation is basically multiplication and accumulation, also called the inner product. The computational complexity of CNN inference is mostly dominated by multiply–accumulate (MAC) operations in convolutional and fully connected layers (with max-pooling requiring comparators forming a small part of the CNN computations during inference). Multipliers consume more energy than adders, e.g., the energy required to multiply two 8-bit integers is 6.7× the energy required to add them, and the energy required to multiply two 32-bit integers is 31× the energy required to add them [1]. Hence, redesigning CNNs to reduce multiplication operations (partially or completely if possible) is necessary.

Second, floating-point arithmetic consumes substantially more energy than integer arithmetic because floating-point addition requires exponent alignment, normalization, rounding, and exception handling in addition to mantissa addition. A 32-bit floating-point addition consumes 9× more energy than 32-bit integer addition in 45 nm CMOS technology [1]. Hence, redesigning the CNN to replace FP weights (original output from CNN training) with integer weights is necessary and beneficial for hardware. Quantization-Aware Training (QAT [2]) and Post-Training Quantization (PTQ [3]) are important prior works in this direction. These works enable the entire inference to be performed only using the integers in the hardware. Finally, as depicted in Figure 1, the energy required to transfer 32-bit data from DRAM is 6400× the energy required to add two 32-bit integers, proving that 'data movement energy dominates computation energy'. Since CNNs weights have to be stored in a memory and moved to an accelerator/processor during inference, such data movement energy use can be avoided if inference can be performed in memory. In-Memory Computing (IMC) is a recent computing paradigm where memory and compute units are co-located—the weights are stored in memory and the computations required during inference are performed either in the memory array or peripheral circuits placed very close to the memory array. IMC overcomes the DRAM-access energy problem and is being increasingly seen as a viable alternative for the inference of neural networks [4–7]. However, the implementation of present CNNs will be a natural avenue for future work based on the obtained architectural results. In summary, a redesign is needed to make the CNN not only hardware-friendly but also memory-friendly, i.e., suitable for in-memory implementation.

Building on these observations, we explore CNN optimization strategies which will enable efficient inference when the CNN is implemented in hardware. Our goal in this research is to answer the following question: "How can existing CNNs be optimized for efficient in-memory inference without introducing a drastic loss in classification accuracy?" To this end, we take some established CNNs and optimize them to be hardware-friendly and memory-friendly. We considered image-classification datasets such as MNIST, fashionMNIST, CIFAR-10, and PneumoniaMNIST because they can be correctly classified by

compact and lightweight CNN architectures usually deployed at the edge. We "optimize" them by some techniques (called redesign techniques, see Section 2.1) which will be beneficial from a hardware and in-memory implementation point of view. To clarify, our goal in this work is not CNN implementation on a specific hardware like ASIC/FPGA. Instead, our goal is to optimize the CNN to the maximum extent possible before it is implemented in hardware. Such optimization is not possible without inputs from hardware engineers and we use HWcost to essentially capture the cost of the hardware required. However, we do not perform the end hardware implementation of the optimized CNN in this work.

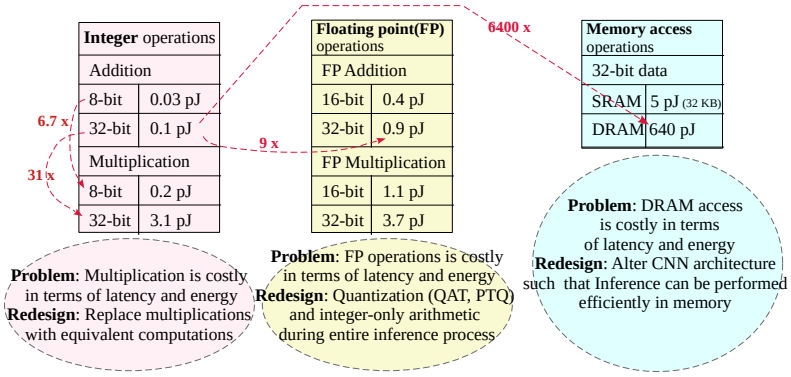



**Figure 1.** Energy problems associated with CNN inference (energy numbers from 45 nm process, taken from [1,8]) and the need for redesign.

The rest of the paper is organized as follows. Section 2.1 presents the six redesign techniques we employed in this work. Section 2.2 briefly presents the method to perform convolutions in memory without multipliers and formulates a methodology to estimate hardware cost. We use this estimation methodology to ascertain the hardware cost of each of the candidate CNN architectures and rank them based on this cost and classification performance. Section 3 elaborates our training and inference methodology for the CNNs considered in this work and the software used. Sections 4–6 presents the CNN optimization methodology and illustrates how we arrived at the efficient candidate for LeNet5, VGGlite and PneumoniaMNIST, respectively. Section 7 summarizes our results and presents the outlook for future work.

## 2. Redesign Techniques and Efficiency Metrics

### *2.1. Redesign Techniques*

Although numerous hardware-oriented CNN optimization techniques have been proposed in the past, they are often evaluated under different assumptions and target hardware models. This work does not introduce the following techniques as independent algorithmic contributions. Instead, it uses them as hardware-motivated design constraints within a Distributed-Arithmetic-oriented in-memory inference framework. The goal is to evaluate the aggregate effects of all six techniques to find an accuracy–HWCost trade-off for the resulting CNN candidates, rather than to identify the effect of each technique individually. The six redesign techniques investigated are as follows:

1. **Reduction in classifier complexity through Global Average Pooling:** When the baseline architecture contains large fully connected classification layers, these layers can be replaced by Global Average Pooling (GAP) followed by a compact linear classifier. This transformation greatly reduces the number of weights in the CNN and consequentially its computational complexity.
2. **Integer-oriented quantization and power-of-two requantization:** Weights and activations are quantized to low-bit integer (6 and 8 bits for weights, 8 bits for activa-

tions) representations in order to reduce the memory storage requirement and enable multiplierless inference (multiplierless inference is achieved by a technique called Distributed Arithmetic, which is elaborated upon in Section 2.2). Quantization-Aware Training incorporates the effects of integer arithmetic and power-of-two activation requantization during training, enabling efficient integer deployment while preserving classification performance.

3. **Reduction in the computational cost of the convolution layer:** The computational cost of convolutional layers is reduced by exploring hardware-relevant modifications appropriate to the considered CNN family. For LeNet-like networks, this includes filter size reduction (e.g., from $5 \times 5$ to $3 \times 3$) and sparse inter-layer connectivity. For VGG-like networks which already employ $3 \times 3$ kernels, this includes reducing the number of feature maps and the number of convolutional layers in deeper stages.
4. **Hardware-aware optimization of down-sampling operations.** Max-pooling is the down-sampling operation which CNNs use to reduce the size of the feature maps. As the CNN proceeds through the stages of inference, the size of the feature maps is progressively reduced to minimize computation. Since this max-pooling is implemented using comparator circuits outside the memory array, it is preferable to minimize such operations. However, removing max-pooling by equivalent operations (strided convolutions) is not always beneficial and can affect classification accuracy. Therefore, pooling alternatives are explored according to their overall impact on the hardware cost and accuracy, rather than minimizing the number of pooling operations unconditionally.
5. **Power-of-two-compatible (PoT-compatible) Global Average Pooling:** The spatial size of the feature maps entering GAP is constrained so that the number of averaged elements is a power of two. Consequently, the 'averaging' operation required during GAP can be implemented using integer accumulation followed by a truncating right shift only, avoiding division operations which are difficult to implement in memory.
6. **Deployment-aligned training and integer export:** When required by the baseline architecture, Batch Normalization is used during floating-point pretraining and subsequently folded into the convolutional parameters before quantization-aware fine-tuning. Input normalization is similarly folded into the first convolutional layer. The resulting selected networks are exported as integer representations and validated through pure integer inference using the same shift-based arithmetic assumed by the target in-memory hardware.

### *2.2. Efficiency Metric*

For efficient inference, a CNN must be implemented completely using integers and must avoid multiplications and divisions and minimize memory access (each access to the weights stored in memory requires energy, and the energy available depends on the memory technology, e.g., the energy consumed reading a 32-bit weight from a DRAM is $128\times$ the energy needed to read the same weight from a SRAM; Figure 1). To satisfy these requirements, we pioneered a method which exploits 'Distributed Arithmetic' (DA) to perform convolution without hardwired multipliers (presented in earlier works [7,9]). In addition to enabling "multiplierless" convolution, DA can save 'memory access' energy because DA uses adders/shifters located very close to memory. We briefly explain how 'Distributed Arithmetic' is used to achieve multiplierless convolution and the reader is referred to [7,9] for a detailed explanation.

Distributed Arithmetic (DA) is an efficient procedure for computing inner product between a fixed vector and a variable vector [10]. The inner product of two vectors is the sum of the products of the corresponding components of these vectors.

$$Y = X^T W = \sum_{i=1}^{N} X_i W_i \tag{1}$$

where $X^T = [X_1, X_2, \ldots, X_N]$ is the variable vector and $W^T = [W_1, W_2, \ldots, W_N]$ is a fixed vector. $[W_1, W_2, \ldots, W_N]$ are the weights of the CNN and they remain constant throughout the 'inference'. (Once the CNN is trained, the weights do not change and they can be considered as constants.) Traditionally, this inner product is calculated using a multiply–accumulate (MAC) unit that multiplies the weight, $W$, with the input vector, $X$, and adds them to the partial product to obtain the final sum [11]. DA is a different method to calculate the same inner product, $Y = \sum X^T \cdot W$, without employing multipliers. As illustrated in Figure 2b, the sum of the weights is calculated in advance and kept in the memory as a reference table. The input vector, $X$, is shown as bits and, as illustrated in Figure 2a, one bit from $X_1, X_2, X_3$ is supplied to the memory array in a bit-serial fashion. This forms the address and the data in the memory is retrieved in stages 1–3. At every step, the read out MR is shifted to the left by 1 bit (to accomplish $\times$ 2) and added to cumulatively calculate $Y$. In this way, DA reaches an architecture without multipliers when one of the vectors is a constant.

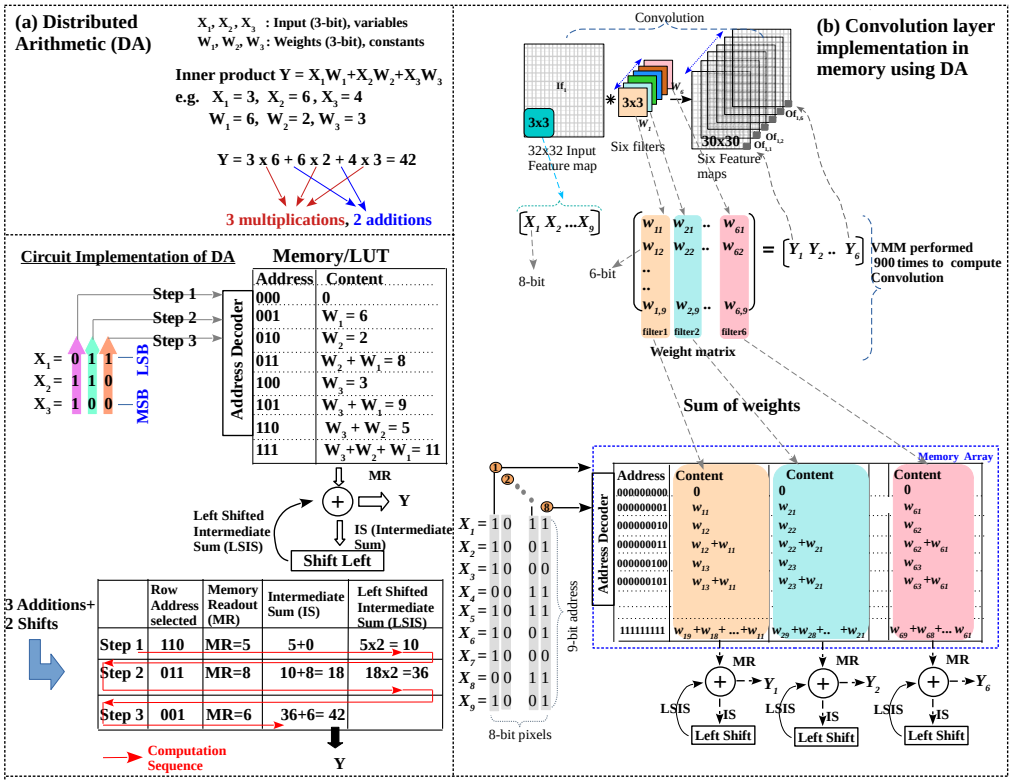



**Figure 2.** (**a**) Concept of Distributed Arithmetic (DA): When $W$ is constant, $Y = \sum X_i W_i$ can be computed with addition and shifts instead of addition and multiplications. (**b**) A convolution layer's computation formulated as a vector–matrix multiplication, which is in turn computed using adders, shifters and a memory array ('$*$' denotes convolution).

Figure 2b illustrates how one convolution layer of a CNN can be implemented using DA. Let us assume a convolution layer involves convolution of an image with six 3 $\times$ 3 filters. A 3 $\times$ 3 portion of the image is convolved with six filters simultaneously. This 3 $\times$ 3 portion of the input image becomes a 9-element vector, $X$. The six 3 $\times$ 3 filters

form the $9 \times 6$ weight matrix. A single stride of convolution can be formulated as this vector–matrix multiplication (VMM). For the convolution of this $32 \times 32$ image, this VMM is performed 900 times. This VMM can be implemented by storing the sum-of-weights in the columns of the memory, as shown in Figure 2b. Since $X$ is 8-bit, a single VMM requires 8 cycles. Since convolution is implemented using adders and shifters, the hardware cost is reduced (no multipliers) and the energy for convolution is the sum of adder energy, shifter energy and memory read-out energy. In this manner, the convolution layer of a CNN is implemented in an energy-efficient manner in the memory without requiring a separate processor. In the rest of the paper, we optimize CNNs with this target architecture in mind, i.e., we redesign CNNs such that they can be implemented using Distributed Arithmetic (multiplications replaced by a memory read, followed by an add and shift) in the target hardware.

Since the considered CNN families are composed of convolutional layers, max-pooling and fully connected layers (classifiers), we compute the hardware cost of each layer as follows. First, by hardware cost, we refer to the cost of the electronic circuitry used to perform these operations, and our estimate is for DA-inspired implementation. For CNN optimization at the neural network level, abstracting the hardware cost in terms of number of transistors enables one to obtain a rough estimate of the electronic circuitry required for inference. In what follows, we elaborate upon how we calculated the hardware cost for each layer of a CNN.

The hardware cost for convolution layer predominantly depends on the number of filters and the number input feature maps. Consider the example in Figure 2b: a $32 \times 32$ image is convolved with six $3 \times 3$ filters. This requires a memory array with six adders and six shifters. Hence,

$$HWCost_{Convolution/Inputchannel} = 1 \cdot T_{PMA} + n \cdot T_{adder} + n \cdot T_{shifter} \quad (2)$$

where $n$ is the number of filters (number of columns of the matrix) an input feature map is simultaneously convolved with and $PMA$ is the Processing Memory Array. (PMAs are memory arrays which store the sum-of-the-weights, as illustrated in Figure 2. We call them 'Processing' Memory Arrays because they form a significant part of the convolution computation.) If we have $m$ such input feature maps (e.g., later convolution layers), the hardware cost is:

$$HWCost_{Convolution} = m \times (1 \cdot T_{PMA} + n \cdot T_{adder} + n \cdot T_{shifter}) \quad (3)$$

where $m$ is the number of input channels. Note that we do not precisely model the size of the adder/shifter in Equation (3). For 8-bit weights (precision of $W$) and 8-bit inputs (precision of $X$), we need a 21-bit adder and 21-bit shifter per column of the matrix [7]. If we have 6-bit weights, the required adder bit width will be less, but we do not model this since we are interested in the number of adders/shifters required and not in their exact bit width at this level of abstraction. A 21-bit adder will need 1372 transistors ($T_{adder}$); a 21-bit shifter will require 21 D-flip flops= $21 \times 24$ transistors = 504 transistors ($T_{shifter}$) [9]. For the memory array, we assume a $512 \times 512$ SRAM and if another memory technology is intended, the transistor count will vary accordingly. Assuming a standard 6-transistor SRAM cell, the estimate for the memory array will be $T_{PMA}$= $1.6 \times 10^6$ transistors ($512 \times 512 \times 6$ = 1,572,864 transistors + peripheral circuitry requiring $\approx$ 29,000 transistors (row decoder, sense amplifiers, precharge, write drivers, control logic), estimated based on [12]).

The hardware cost of ReLU+PoT RQ is one comparator (for ReLU operation) and an adder+shifter+comparator combination for requantization. (Since we do QAT, we add $2^{r-1}$ to the pixel value and then right shift it by $r$ to do the division where $r$ is learned

during training. The resulting value is clamped to be within 255 which requires another comparator.) Hence, we need (adder+shifter+2.comparator) per input channel: Hence, we need an adder, a shifter and two comparators per input channel:

$$HWCost_{ReLU+POTRQ} = T_{adder} + T_{shifter} + 2 \cdot T_{comparator} \quad (4)$$

The hardware cost of max-pooling operation can be estimated as the number of comparators needed for it. This depends heavily on the end hardware implementation. Since our CNN redesign strategy is agnostic to a specific hardware, we approximate the cost assuming a streaming data flow model which is generally used for CNNs in both FPGA and ASIC implementations. In streaming model, one does not wait for the entire layer to be computed to start computing the next layer. Instead, as soon as a portion of a layer of the CNN is computed, it is sent to the next layer to minimize latency [13]. In addition to reducing latency, this minimizes the number of registers/memory to hold the intermediate pixel values. If we assume such a streaming data flow model, then max-pooling is typically computed using one comparator per channel. Since max-pooling is performed after ReLU+PoT RQ, the input to max-pooling is unsigned 8-bit integers. Essentially, we need a comparator to compare two positive integers. We estimate such a comparator to require 142 transistors ($T_{comparator}$) [14].

The hardware cost for $HWCost_{POT-GAP}$ is simply the cost of averaging the pixels values of the input feature map, i.e., addition followed by division. Note that we have designed our candidate CNNs such that the size of input feature maps is a power of two ($8 \times 8$, $16 \times 16$, etc.). Hence, division can be implemented as a simple right-shift operation. Assuming addition is performed sequentially using a single adder per input channel:

$$HWCost_{POT-GAP} = T_{adder} + T_{shifter} \quad (5)$$

The fully connected classifier is modeled as a DA-based vector–matrix multiplication using the same nine-input PMA organization adopted for the convolutional layers. Since the 512 addressable rows of the memory array correspond to $2^9$ possible input-bit combinations, each PMA processes groups containing at most nine input activations. Therefore, for a fully connected layer with $N_{\text{in}}$ input features and $N_{\text{out}}$ output neurons, the number of input groups is

$$G_{\text{FC}} = \left\lceil \frac{N_{\text{in}}}{9} \right\rceil. \quad (6)$$

If $N_{\text{in}}$ is not an integer multiple of nine, the unused positions in the final group are set to zero. Under the adopted resource model, each input group requires one PMA and one adder–shifter datapath per output neuron. Consequently, the hardware resource counts are

$$N_{\text{PMA,FC}} = G_{\text{FC}}, \quad (7)$$

$$N_{\text{adder,FC}} = N_{\text{shifter,FC}} = G_{\text{FC}} N_{\text{out}}. \quad (8)$$

The corresponding transistor-level hardware cost is

$$\text{HWCost}_{\text{FC}} = G_{\text{FC}} T_{\text{PMA}} + G_{\text{FC}} N_{\text{out}} (T_{\text{adder}} + T_{\text{shifter}}). \quad (9)$$

Bias terms are handled within the existing integer accumulation datapath and therefore do not introduce an additional hardware resource term in the adopted model. The final class decision is implemented using one comparator reused sequentially across the output logits. Hence,

$$N_{\text{comparator,Argmax}} = 1, \quad (10)$$

and

$$\mathrm{HWCost}_{\mathrm{Argmax}} = T_{\mathrm{comparator}}. \tag{11}$$

In this manner, the hardware cost of all layers of the CNN is estimated in terms of the adders, shifters, comparators and memory arrays needed for each of these layers. These components are in turn expressed in terms of number of transistors, enabling us to obtain the total hardware cost in terms of transistors. Therefore,

$$\begin{aligned}\mathrm{HWCost}_{\mathrm{CNN}} = &\sum_{\ell} \mathrm{HWCost}_{\mathrm{Convolution},\ell} + \sum_{\ell} \mathrm{HWCost}_{\mathrm{ReLU+PoTRQ},\ell} \\ &+ \sum_{\ell} \mathrm{HWCost}_{\mathrm{MaxPool},\ell} + \mathrm{HWCost}_{\mathrm{PoT\text{-}GAP}} \\ &+ \mathrm{HWCost}_{\mathrm{FC}} + \mathrm{HWCost}_{\mathrm{Argmax}}.\end{aligned} \tag{12}$$

where $\ell$ denotes the layer index. The summations are evaluated over all convolutional, ReLU–PoT requantization, and max-pooling layers present in the considered CNN.

## 3. Experiments and Discussions

### *3.1. Training/Inference Setup and Reproducibility*

All experiments were trained on an NVIDIA GeForce RTX 3070 GPU. The software environment was managed using Conda (version 26.3.2), and the complete software stack used for model training, evaluation, integer inference, and architecture diagram generation is summarized in Table 1. The preprocessing pipeline was kept fixed across training, validation, testing, and integer-deployment evaluation.

**Table 1.** Software libraries used for training, evaluation, integer inference, and architecture diagram export.

| Library/Tool | Version | Role |
|---|---|---|
| Python | 3.11.9 | Runtime environment |
| PyTorch | 2.3.1 | Model definition and QAT training |
| Torchvision | 0.18.1 | datasets and transforms |
| PyTorch-CUDA | 12.1 | GPU-accelerated training backend |
| NumPy | 1.26.4 | Pure integer CPU inference and NPZ handling |
| Scikit-learn | 1.4.2 | Evaluation utilities/analysis support |
| Torchview | 0.2.7 | PyTorch blueprint graph visualization |
| python-graphviz | 0.21 | Python interface for Graphviz rendering |
| Graphviz | 14.1.2 | SVG/PDF/PNG architecture diagram export |

To improve the reproducibility of the training and evaluation procedure, we adopted the following measures:

- We fixed the random seed for PyTorch, NumPy, and Python's built-in random module.
- We fixed the CUDA random seed when GPU training was available.
- We disabled cuDNN benchmarking to avoid non-deterministic algorithm selection.
- We enabled deterministic cuDNN behavior.
- We enabled deterministic PyTorch algorithms where available using `torch.use_deterministic_algorithms(True, warn_only=True)` (https://docs.pytorch.org/docs/stable/generated/torch.use_deterministic_algorithms.html (accessed on 20 July 2026)). This setting improves reproducibility on the same software and hardware stack, although PyTorch does not guarantee bit-exact reproducibility across different platforms, releases, or CPU/GPU backends (https://docs.pytorch.org/docs/stable/notes/randomness.html (accessed on 20 July 2026)).
- We fixed the random generator used for the train/validation split.

- We fixed the random generator used for shuffling the training data loader.
- We used the same train/validation split, batch size, optimizer, learning rate, and cosine annealing schedule across comparable experiments.
- We selected the best checkpoint according to validation accuracy and evaluated the final model on the independent test set.
- We exported the trained model to an integer-only NumPy inference pipeline to verify deployment consistency.

Our experiments employ four widely adopted benchmark datasets spanning diverse image domains: MNIST for handwritten digits, Fashion-MNIST for clothing articles, CIFAR-10 for real-world objects, and PneumoniaMNIST for medical chest X-ray classification. For MNIST and FashionMNIST, the original $28 \times 28$ grayscale images were zero-padded to $32 \times 32$ in order to match the LeNet-style spatial geometry adopted during the architecture search. CIFAR-10 images were used at their native $32 \times 32$ RGB resolution, whereas the PneumoniaMNIST samples were resized to $64 \times 64$ grayscale images. In all quantized experiments, input tensors were scaled to the $[0, 255]$ range to emulate UINT8 activations at the network input. The only exception was the floating-point LeNet-5 baseline, for which the conventional $[0, 1]$ input range was retained because the `tanh` activations can easily saturate when exposed to larger input magnitudes. The experiments were conducted using the official dataset partitions. MNIST and FashionMNIST contain 60,000 training images and 10,000 test images, with a validation subset extracting 10% from the training partition. CIFAR-10 contains 50,000 training images and 10,000 test images, again using a validation subset derived from the training set. PneumoniaMNIST contains 4708 training images, 524 validation images, and 624 test images. Table 2 summarizes the main characteristics of the datasets used throughout this work.

**Table 2.** Datasets used for architecture search and evaluation.

| Dataset | Classes | Input Size | Train | Validation | Test |
|---|---|---|---|---|---|
| MNIST | 10 | $28 \times 28$ (padded to $32 \times 32$) | 54,000 | 6000 | 10,000 |
| FashionMNIST | 10 | $28 \times 28$ (padded to $32 \times 32$) | 54,000 | 6000 | 10,000 |
| CIFAR-10 | 10 | $32 \times 32 \times 3$ | 45,000 | 5000 | 10,000 |
| PneumoniaMNIST | 2 | $64 \times 64$ | 4708 | 524 | 624 |

### *3.2. Training and Integer Inference Methodology*

The proposed methodology was evaluated on MNIST, FashionMNIST, CIFAR-10, and PneumoniaMNIST. MNIST was used as a baseline dataset to validate the Quantization-Aware Training (QAT) and integer inference pipeline, FashionMNIST provided a more challenging grayscale classification task, CIFAR-10 was adopted to assess the scalability of the proposed hardware-aware design methodology on a more complex natural image dataset, while PneumoniaMNIST served as a representative medical imaging case study. LeNet-like architectures were explored on MNIST and FashionMNIST through an automated architecture search, whereas compact VGG-inspired CNNs were investigated on PneumoniaMNIST and CIFAR-10. Candidate models were ranked according to classification performance and hardware cost estimate assuming a DA-oriented in-memory implementation. Classification accuracy was used for MNIST, FashionMNIST, and CIFAR-10, while balanced accuracy was adopted for PneumoniaMNIST due to class imbalance.

Candidate selection was performed exclusively on the validation set. For MNIST, FashionMNIST, and CIFAR-10, a fixed-seed architecture search was used to identify the representative accuracy, efficiency, and HWCost-oriented candidates, which were then retrained over five independent seeds for final reporting. For PneumoniaMNIST, due to its

smaller validation+test partitions and stronger validation+test sensitivity, all 64 candidates were ranked over five independent seeds before selecting the final representatives.

The selected architectures were exported as integer `NPZ` representations containing quantized weights, biases, and requantization parameters for subsequent hardware-oriented deployment studies.

## 4. LeNet-5-Inspired Network on MINST and FashionMNIST

We first implemented a modernized LeNet-5-style baseline (Figure 3) preserving the original sparse C3 connectivity pattern, while using standard PyTorch average pooling and a conventional linear classification head trained with cross-entropy loss. The inputs pixels were normalized in the range 0–1 (conventional and necessary step due to the presence of tanh activations, easily saturating). This model was evaluated in floating point and was used as the initial accuracy reference before introducing the hardware/in-memory oriented architectural simplifications and integer-inference constraints discussed in the following sections.

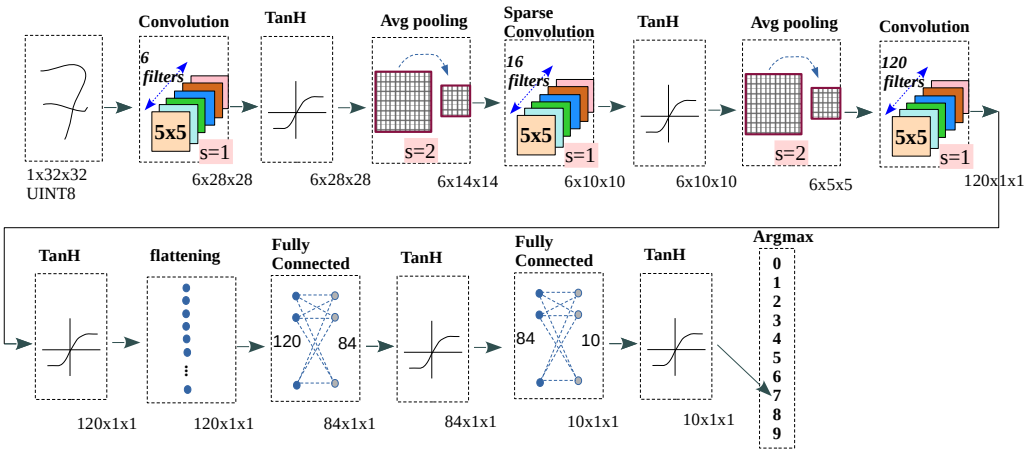



**Figure 3.** Original LeNet-5-style floating-point baseline. The network follows the classical path, using tanh activations, average-pooling layers, sparse connectivity, and a dense final classifier.

The corresponding parameter breakdown is reported in Table 3. The sparse C3 layer contains 60 active input–output feature map connections. Since each connection uses a $5 \times 5$ kernel, this corresponds to $60 \times 25 = 1500$ weights, plus 16 bias terms, for a total of 1516 parameters. The model was trained for 200 epochs on both MNIST and FashionMNIST using the train/validation/test split strategy. The best checkpoint was selected according to validation accuracy and then evaluated on the independent test set. The results are summarized in Table 4.

**Table 3.** Parameter breakdown of the LeNet-5-style baseline.

| Layer | Parameters |
|---|---|
| Convolution 1: $1 \rightarrow 6$, $5 \times 5$ convolution | 156 |
| Convolution 2 (sparse): $6 \rightarrow 16$, sparse $5 \times 5$ convolution | 1516 |
| Convolution 3: $16 \rightarrow 120$, $5 \times 5$ convolution | 48,120 |
| Fully connected: $120 \rightarrow 84$ fully connected | 10,164 |
| Output: $84 \rightarrow 10$ fully connected | 850 |
| **Total** | **60,806** |

**Table 4.** Floating-point PyTorch performance of the LeNet-5-style baseline.

| Dataset | Best Epoch | Best Val. Acc. | Test Loss | Test Acc. |
|---|---|---|---|---|
| MNIST | 160 | 98.88% | 0.0777 | 99.01% |
| FashionMNIST | 63 | 90.00% | 0.4838 | 89.78% |

The LeNet-5-style baseline achieves strong floating-point classification performance, reaching a final test accuracy of 99.01% on MNIST and 89.78% on FashionMNIST. These results provide an upper reference point for the subsequent hardware-oriented models, which are intentionally constrained by low-bit quantization, integer arithmetic, reduced parameter count, and simplified datapaths.

However, this accuracy is obtained with a relatively large parameter budget of 60,806 trainable parameters. As shown in Table 3, most of the parameters are concentrated in the final dense stages, especially the C5 convolution and the fully connected classifier. From an in-memory oriented perspective, this parameter distribution is not ideal: large dense stages increase coefficient storage and enlarge the vector–matrix multiplication structures required during inference. Moreover, the use of tanh activations and average-pooling operations is less aligned with a compact low-bit integer datapath, being replaced by ReLU. Therefore, this baseline is used primarily as a floating-point accuracy reference, while the following sections investigate how much of this performance can be retained after replacing parameter-heavy and hardware-unfriendly components with in-memory oriented alternatives.

### *4.1. Model Selection Strategy*

In Table 5 there is the search space explored to make our LeNet-5-inspired CNN efficient for in-memory implementation. The search space was further constrained by hardware-oriented geometric rules: pooling layers were accepted only when their windows exactly tiled the feature map, and only power-of-two feature map areas were allowed before the PoT-GAP stage, for a total of 10 combinations.

**Table 5.** Architectural design parameters explored during the LeNet-inspired model selection process.

| Parameter | Explored Values |
|---|---|
| Conv1 kernel size | $\{3 \times 3,\ 5 \times 5\}$ |
| MaxPool1 window | $\{0,\ 2 \times 2,\ 3 \times 3,\ 5 \times 5\}$ |
| Conv2 kernel size | $\{3 \times 3,\ 5 \times 5\}$ |
| MaxPool2 window | $\{0,\ 2 \times 2,\ 3 \times 3,\ 5 \times 5\}$ |
| Weight precision | {INT6, INT8} |
| Activation precision | UINT8 (fixed) |
| Conv2 connectivity | Sparse LeNet C3 (fixed) |
| Activation function | ReLU + PoT requantization (fixed) |
| Convolution stride | 1 (fixed) |
| Pre-GAP feature map size | $\{4 \times 4,\ 8 \times 8,\ 16 \times 16\}$ |

### *4.2. Selected Model for MNIST Dataset*

For MNIST, the selected representative candidates were retrained and evaluated over five independent QAT runs (Table 6). The highest-validation-accuracy model, `cand_007`, achieved a validation accuracy of $96.72 \pm 0.27\%$ and a test accuracy of $96.74 \pm 0.19\%$, whereas the validation-efficiency-oriented model, `cand_001`, in Figure 4, achieved $95.32 \pm 0.39\%$ and $95.51 \pm 0.33\%$, respectively. Thus, the validation and test accuracy gains are approximately 1.40 and 1.23 percentage points, respectively, while the estimated transistor-level HWCost increases from 14,707,402 to 37,108,822, corresponding

to approximately 2.52× the hardware cost. The same candidate, `cand_001`, is also selected as the minimum-HWCost architecture.

The main architectural difference lies in the convolutional kernel sizes. Candidate `cand_001` employs $3 \times 3$ kernels in both convolutional layers, whereas `cand_007` uses $5 \times 5$ kernels. Under the adopted multiplierless resource model, each $3 \times 3$ convolution requires one PMA per input feature map, while a $5 \times 5$ convolution requires three PMAs per input feature map because the 25-input dot product must be partitioned across three PMAs. Consequently, the number of PMAs increases from 9 to 23, which dominates the increase in the estimated transistor-level HWCost. The numbers of adders and shifters remain unchanged, while the comparator count increases only moderately from 51 to 61.

**Table 6.** Selected deployable LeNet-inspired candidates on MNIST, including architecture-search knob values, FP32/PTQ baselines, QAT performance, and estimated transistor-level hardware resources. FP32, PTQ, and QAT results are reported over five independent seeds.

| Property | Highest Validation Efficiency | Highest Validation Accuracy | Lowest HWCost |
|---|---|---|---|
| **Classification performance** | | | |
| Candidate ID | `cand_001` | `cand_007` | `cand_001` |
| QAT validation accuracy [%] | $95.32 \pm 0.39$ | $96.72 \pm 0.27$ | $95.32 \pm 0.39$ |
| QAT validation accuracy 95% CI [%] | [94.84, 95.80] | [96.39, 97.05] | [94.84, 95.80] |
| QAT test accuracy [%] | $95.51 \pm 0.33$ | $96.74 \pm 0.19$ | $95.51 \pm 0.33$ |
| QAT test accuracy 95% CI [%] | [95.10, 95.92] | [96.50, 96.98] | [95.10, 95.92] |
| FP32 test accuracy [%] | $96.00 \pm 0.20$ | $96.99 \pm 0.15$ | $96.00 \pm 0.20$ |
| FP32 test accuracy 95% CI [%] | [95.75, 96.26] | [96.80, 97.18] | [95.75, 96.26] |
| PTQ test accuracy [%] | $92.23 \pm 1.95$ | $93.94 \pm 1.69$ | $92.23 \pm 1.95$ |
| PTQ test accuracy 95% CI [%] | [89.81, 94.65] | [91.84, 96.03] | [89.81, 94.65] |
| Parameters | 786 | 1842 | 786 |
| **Architecture-search knob values** | | | |
| Conv1 kernel size | $3 \times 3$ | $5 \times 5$ | $3 \times 3$ |
| MaxPool1 window | $3 \times 3$ | 0 (none) | $3 \times 3$ |
| Conv2 kernel size | $3 \times 3$ | $5 \times 5$ | $3 \times 3$ |
| MaxPool2 window | 0 (none) | $3 \times 3$ | 0 (none) |
| Weight precision | `INT6` | `INT6` | `INT6` |
| Pre-GAP feature map size | $8 \times 8$ | $8 \times 8$ | $8 \times 8$ |
| **Fixed deployment setting** | | | |
| Activation precision | `UINT8` | `UINT8` | `UINT8` |
| **Estimated hardware resources** | | | |
| $N_{\text{PMA}}$ | 9 | 23 | 9 |
| $N_{\text{adder}}$ | 160 | 160 | 160 |
| $N_{\text{shifter}}$ | 160 | 160 | 160 |
| $N_{\text{comparator}}$ | 51 | 61 | 51 |
| Estimated HWCost [transistors] | 14,707,402 | 37,108,822 | 14,707,402 |
| Relative HWCost | 1.00× | 2.52× | 1.00× |
| Mean validation efficiency | $6.481 \times 10^{-6}$ | $2.606 \times 10^{-6}$ | $6.481 \times 10^{-6}$ |

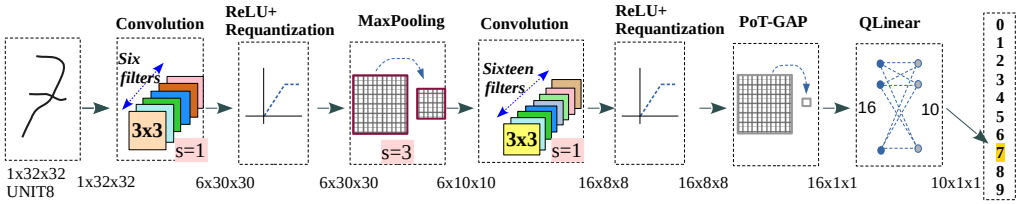



**Figure 4.** Architecture of the highest-validation-efficiency LeNet-inspired model selected for MNIST. 's' means stride. The network corresponds to candidate `cand_001` and employs two $3 \times 3$ convolutional layers with `INT6` weights, a $3 \times 3$ max-pooling stage after the first convolutional layer, sparse LeNet C3 connectivity in the second convolutional layer, ReLU activation followed by power-of-two requantization, an $8 \times 8$ PoT-GAP layer, and a final integer-only fully connected classifier. This candidate is also the minimum-HWCost architecture among the explored MNIST architectures.

### *4.3. Selected Model for FashionMNIST Dataset*

For FashionMNIST, the selected representative candidates were retrained and evaluated over five independent QAT runs. The validation-efficiency-oriented candidate is `cand_006` (Figure 5), which is also selected as the minimum-HWCost architecture. The architecture employs $3 \times 3$ kernels in both convolutional layers, a $5 \times 5$ max-pooling operation after the first convolution, no second pooling operation, and `INT8` weights. It achieves a validation accuracy of $85.93 \pm 0.34\%$ and a test accuracy of $85.01 \pm 0.43\%$, with an estimated transistor-level HWCost of 14,707,402.

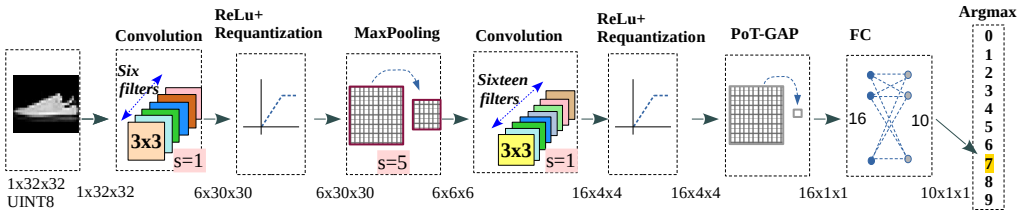



**Figure 5.** Architecture of the highest-validation-efficiency LeNet-inspired model selected for FashionMNIST. 's' means stride. The network corresponds to candidate `cand_006` and employs two $3 \times 3$ convolutional layers with `INT8` weights, a $5 \times 5$ max-pooling stage after the first convolutional layer, sparse LeNet C3 connectivity in the second convolutional layer, ReLU activation followed by power-of-two requantization, a $4 \times 4$ PoT-GAP layer, and a final integer fully connected classifier. This candidate is also the minimum-HWCost architecture among the explored FashionMNIST architectures.

In Table 7 the highest-validation-accuracy candidate is `cand_008`, which achieves a validation accuracy of $86.70 \pm 0.27\%$ and a test accuracy of $85.92 \pm 0.28\%$. Compared with `cand_006`, this corresponds to improvements of approximately 0.77 and 0.91 percentage points on validation and test accuracy, respectively. However, the estimated HWCost increases to 37,108,822 transistors, corresponding to approximately $2.52\times$ the minimum estimated hardware cost. The hardware cost increase is primarily associated with the use of $5 \times 5$ kernels in both convolutional layers, which require the corresponding dot products to be partitioned across more 9-input PMAs, increasing the total number of PMAs from 9 to 23. In contrast, the numbers of adders and shifters remain unchanged, while the comparator count increases only moderately from 51 to 61. Consequently, `cand_006` provides the most favorable validation-accuracy-to-resource trade-off for FashionMNIST.

**Table 7.** Selected deployable LeNet-inspired candidates on FashionMNIST, including architecture-search knob values, FP32/PTQ baselines, QAT performance, and estimated transistor-level hardware resources. FP32, PTQ, and QAT results are reported over five independent seeds.

| Property | Highest Validation Efficiency | Highest Validation Accuracy | Lowest HWCost |
|---|---|---|---|
| **Classification performance** | | | |
| Candidate ID | `cand_006` | `cand_008` | `cand_006` |
| QAT validation accuracy [%] | $85.93 \pm 0.34$ | $86.70 \pm 0.27$ | $85.93 \pm 0.34$ |
| QAT test accuracy [%] | $85.01 \pm 0.43$ | $85.92 \pm 0.28$ | $85.01 \pm 0.43$ |
| FP32 test accuracy [%] | $85.07 \pm 0.23$ | $86.24 \pm 0.30$ | $85.07 \pm 0.23$ |
| FP32 test accuracy 95% CI [%] | [84.79, 85.36] | [85.86, 86.61] | [84.79, 85.36] |
| PTQ test accuracy [%] | $83.04 \pm 1.28$ | $83.87 \pm 0.77$ | $83.04 \pm 1.28$ |
| PTQ test accuracy 95% CI [%] | [81.46, 84.63] | [82.91, 84.83] | [81.46, 84.63] |
| Parameters | 786 | 1842 | 786 |
| **Architecture-search knob values** | | | |
| Conv1 kernel size | $3 \times 3$ | $5 \times 5$ | $3 \times 3$ |
| MaxPool1 window | $5 \times 5$ | 0 (none) | $5 \times 5$ |
| Conv2 kernel size | $3 \times 3$ | $5 \times 5$ | $3 \times 3$ |
| MaxPool2 window | 0 (none) | $3 \times 3$ | 0 (none) |
| Weight precision | `INT8` | `INT8` | `INT8` |
| Pre-GAP feature map size | $4 \times 4$ | $8 \times 8$ | $4 \times 4$ |
| **Fixed deployment settings** | | | |
| Feature map profile | 6/16 | 6/16 | 6/16 |
| Activation precision | `UINT8` | `UINT8` | `UINT8` |
| **Estimated hardware resources** | | | |
| $N_{\text{PMA}}$ | 9 | 23 | 9 |
| $N_{\text{adder}}$ | 160 | 160 | 160 |
| $N_{\text{shifter}}$ | 160 | 160 | 160 |
| $N_{\text{comparator}}$ | 51 | 61 | 51 |
| Estimated HWCost [transistors] | 14,707,402 | 37,108,822 | 14,707,402 |
| Relative HWCost | 1.00× | 2.52× | 1.00× |
| Mean validation efficiency | $5.843 \times 10^{-6}$ | $2.336 \times 10^{-6}$ | $5.843 \times 10^{-6}$ |

## 5. CIFAR-10 Evaluation with a Hardware-Optimized VGG-Lite Network

For CIFAR-10, the procedure explores a compact VGG-inspired design space adapted to $32 \times 32$ RGB inputs. All candidate networks are composed of four convolutional stages with $3 \times 3$ kernels, ReLU activation followed by power-of-two requantization, a power-of-two-compatible Global Average Pooling (PoT-GAP) layer, and an integer classifier. Rather than modifying a single baseline architecture, the exploration jointly varies the stage channel profile, the number of convolutions per stage, the weight precision, and the downsampling mechanism.

In this work, the term *stage channel profile* denotes the number of output channels (i.e., feature maps) in the four convolutional stages. For example, the profile $[24, 48, 96, 192]$ corresponds to 24, 48, 96, and 192 output feature maps, respectively, and does not refer to

their spatial dimensions. Likewise, the stage-depth profile $[1, 1, 2, 2]$ denotes one convolution in each of the first two stages and two convolutions in each of the last two stages. The resulting combinations define candidate architectures that are trained and evaluated independently. No single candidate is treated as the starting point of the redesign process; instead, the candidates are ranked after training according to validation performance, estimated transistor-level HWCost, and validation efficiency.

The CIFAR-10 design space also explicitly compares max-pooling-based and stride-based spatial downsampling. The three possible downsampling locations are encoded using the symbols `P`, `S`, and `N`, where `P` denotes a $2 \times 2$ max-pooling operation, `S` denotes a stride-2 convolution applied to the convolutional layer that precedes the corresponding pooling operation, and `N` denotes no downsampling at that location. Therefore, `PPP` corresponds to three max-pooling downsampling stages, whereas `SSS` corresponds to three stride-2 convolutional downsampling stages without max-pooling. The hardware-relevant parameters explored during the CIFAR-10 architecture search are summarized in Table 8.

**Table 8.** Design parameters explored for the VGG-inspired CIFAR-10 architecture.

| Parameter | Explored Values |
|---|---|
| Stage channel profile | $\{[16, 32, 64, 128],\ [24, 48, 96, 192]\}$ |
| Convolutions per stage | $\{[1, 1, 1, 1],\ [1, 1, 2, 2]\}$ |
| Downsampling pattern | {`PPN`, `PPP`, `SSN`, `SSS`} |
| Weight precision | {`INT6`, `INT8`} |
| Activation precision | `UINT8` (fixed) |
| Convolution kernel size | $3 \times 3$ (fixed) |
| PoT-GAP pre-GAP size | $\{4 \times 4,\ 8 \times 8\}$ depending on the downsampling pattern |

The resulting search space contains 32 deployable candidates:

$$2 \times 2 \times 4 \times 2 = 32,$$

corresponding to two stage channel profiles, two depth profiles, four downsampling patterns, and two weight precision values. Each architecture is first pretrained in FP32 with Batch Normalization. Subsequently, Batch Normalization and input normalization are folded into the convolutional parameters, and the network is fine-tuned using hardware-aligned Quantization-Aware Training (QAT). The final deployment configuration employs `UINT8` activations and `INT6`/`INT8` weights. Convolutional requantization is implemented through arithmetic right shifts, with the associated rounding offsets absorbed into the stored `INT32` biases. PoT-GAP is realized through integer accumulation followed by a truncating right shift, while the final classifier uses integer arithmetic and an `argmax` decision stage.

*Selected Integer-Deployable Candidates*

The hardware-aware exploration identified three candidate architectures: the highest-validation-efficiency candidate, the highest-validation-accuracy candidate, and the minimum-HWCost candidate. Table 9 enable a direct comparison between classification performance and the estimated transistor-level hardware resource cost.

For the selected CIFAR-10 representatives, the final FP32, PTQ, and QAT results were recomputed over five independent training seeds. The values reported below refer to the QAT mean and sample standard deviation.

The highest-validation-accuracy solution is `cand_029`, being an architecture with a stage channel profile of $[24, 48, 96, 192]$ and a convolutional depth of $[1, 1, 2, 2]$. It adopts `SSN` downsampling and `INT6` weights. Over five QAT runs, it achieves $91.35 \pm 0.22\%$ validation accuracy and $90.30 \pm 0.07\%$ test accuracy, improving over `cand_004`, in Figure 6,

by approximately 4.78 and 4.81 percentage points, respectively. This gain comes at an estimated HWCost of 903,738,078 transistors, approximately 3.94× the minimum-HWCost architecture, mainly due to the increased numbers of PMAs, adders, and shifters.

**Table 9.** Selected deployable VGG-like candidates on CIFAR-10, including architecture-search knob values, FP32/PTQ baselines, QAT performance, and estimated transistor-level hardware resources. FP32, PTQ, and QAT results are reported over five independent seeds.

| **Property** | **Highest Validation Efficiency** | **Highest Validation Accuracy** | **Lowest HWCost** |
|---|---|---|---|
| **Classification performance** | | | |
| Candidate ID | `cand_004` | `cand_029` | `cand_006` |
| QAT validation accuracy [%] | $86.57 \pm 0.14$ | $91.35 \pm 0.22$ | $86.13 \pm 0.36$ |
| QAT test accuracy [%] | $85.49 \pm 0.16$ | $90.30 \pm 0.07$ | $85.24 \pm 0.26$ |
| FP32 test accuracy [%] | $85.19 \pm 0.15$ | $90.38 \pm 0.23$ | $84.45 \pm 0.19$ |
| FP32 test accuracy 95% CI [%] | [85.01, 85.37] | [90.10, 90.66] | [84.22, 84.68] |
| PTQ test accuracy [%] | $85.01 \pm 0.33$ | $89.73 \pm 0.13$ | $84.10 \pm 0.20$ |
| PTQ test accuracy 95% CI [%] | [84.60, 85.42] | [89.57, 89.90] | [83.85, 84.35] |
| Parameters | 98,730 | 635,674 | 98,730 |
| **Architecture-search knob values** | | | |
| Stage channel profile | [16, 32, 64, 128] | [24, 48, 96, 192] | [16, 32, 64, 128] |
| Convolutions per stage | [1, 1, 1, 1] | [1, 1, 2, 2] | [1, 1, 1, 1] |
| Downsampling pattern | `PPP` | `SSN` | `SSN` |
| Weight precision | `INT8` | `INT6` | `INT8` |
| **Derived and fixed deployment settings** | | | |
| Pre-GAP feature map size | $4 \times 4$ | $8 \times 8$ | $8 \times 8$ |
| Activation precision | `UINT8` | `UINT8` | `UINT8` |
| Convolution kernel size | $3 \times 3$ | $3 \times 3$ | $3 \times 3$ |
| **Estimated hardware resources** | | | |
| $N_{\text{PMA}}$ | 130 | 481 | 130 |
| $N_{\text{adder}}$ | 11,318 | 71,404 | 11,318 |
| $N_{\text{shifter}}$ | 11,318 | 71,404 | 11,318 |
| $N_{\text{comparator}}$ | 593 | 1297 | 481 |
| Estimated HWCost [transistors] | 229,316,774 | 903,738,078 | 229,300,870 |
| Relative HWCost | 1.00007× | 3.94× | 1.00× |
| Mean validation efficiency | $3.775 \times 10^{-7}$ | $1.011 \times 10^{-7}$ | $3.756 \times 10^{-7}$ |

The minimum-HWCost solution is `cand_006`, which shares the same stage channel and convolutional-depth profiles as `cand_004` but uses `SSN` downsampling. It therefore requires the same numbers of PMAs, adders, and shifters, but fewer comparators, reducing the estimated HWCost to 229,300,870. This is only about 0.007% lower than `cand_004`, while its QAT validation and test accuracies are $86.13 \pm 0.36\%$ and $85.24 \pm 0.26\%$. Thus, the slightly higher classification performance of `cand_004` outweighs its marginal resource increase. The transistor-level metric should be interpreted as an area-oriented hardware

resource estimate rather than as a proxy for latency or energy, since intermediate feature map size and activation traffic are not directly modeled.

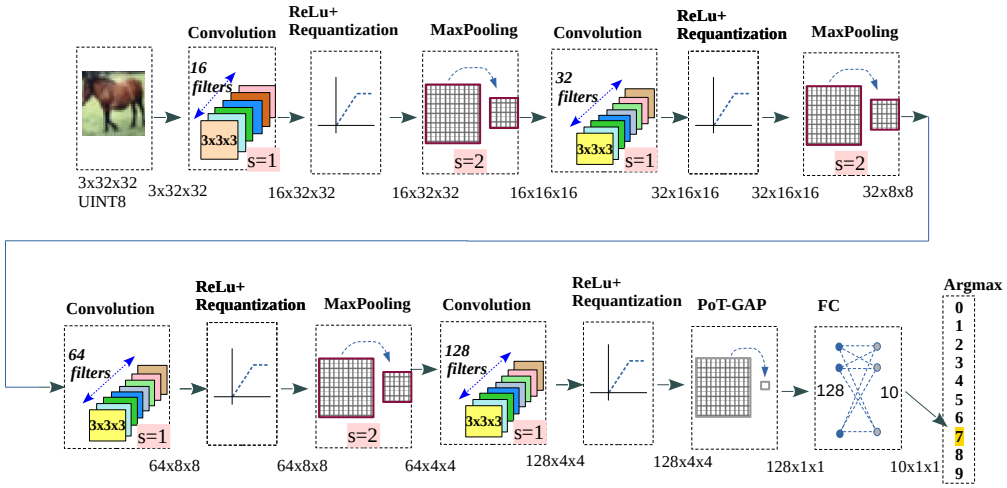



**Figure 6.** Architecture of the highest-validation-efficiency VGG-like model selected for CIFAR-10. 's' means stride. The network corresponds to candidate `cand_004` and employs the stage channel profile $[16, 32, 64, 128]$ with one $3 \times 3$ convolution per stage. Each convolution is followed by ReLU activation and power-of-two requantization to `UINT8`, while the first three stages use $2 \times 2$ max-pooling for spatial downsampling, corresponding to the `PPP` configuration. The model uses `INT8` weights, produces a $4 \times 4$ feature map before PoT-GAP, and terminates with an integer fully connected classifier.

## 6. PneumoniaMNIST

### *6.1. Model Selection Strategy*

Since PneumoniaMNIST exhibits a significant class imbalance, model selection was performed using balanced accuracy rather than standard classification accuracy. Balanced accuracy is defined as

$$\text{BalancedAccuracy} = \frac{1}{2}\left(\frac{TP}{TP+FN} + \frac{TN}{TN+FP}\right), \tag{13}$$

where the two terms correspond to the recall of the positive and negative classes, respectively.

A compact CNN search space was explored to identify memory-friendly architectures while limiting the implementation complexity. The explored design parameters are summarized in Table 10.

**Table 10.** Search space explored for PneumoniaMNIST architecture search.

| Parameter | Explored Values |
|---|---|
| Convolutional layers | 3, 4 |
| Channel profile | $[16, 32, 64]$, $[24, 48, 96]$ |
| Downsampling pattern | `PP`, `PS`, `SP`, `SS` |
| Weight precision | `INT6`, `INT8` |
| Activation precision | `UINT8` |
| Target sparsity | 25%, 50% |
| Batch Normalization | Disabled |
| Total candidates | 64 |

`P` denotes pooling-based downsampling, whereas `S` denotes strided-convolution downsampling. Each two-character pattern specifies the downsampling strategy used across the two spatial-resolution reduction stages.

The symbols P and S denote the downsampling operation used after the first two convolutional stages. P represents a $2 \times 2$ max-pooling layer, whereas S represents a stride-2 convolution applied at the corresponding convolutional stage. Therefore, PP corresponds to the original pooling-based downsampling structure, SS corresponds to a fully strided configuration without max-pooling layers, and PS/SP represent hybrid downsampling configurations.

The resulting search space contains 64 candidate architectures:

$$2 \times 2 \times 4 \times 2 \times 2 = 64,$$

corresponding to two depth choices, two feature map profiles, four downsampling patterns, two weight precision values, and two sparsity levels. Each model was trained using Quantization-Aware Training with UINT8 activations and deployable INT6/INT8 weights. Candidate selection was performed exclusively on the validation set, using validation-balanced accuracy and the proposed validation-efficiency metric based on the estimated HWCost under the adopted memory-centric multiplierless resource model.

### *6.2. Selected Model*

Table 11 reports the highest-validation-efficiency candidate, the highest-validation-balanced accuracy candidate, and the minimum-HWCost candidate obtained from the explored search space.

The candidate-configuration string compactly describes the architecture and the selected search-space values. For example, C3_16_32_64_DPP_W8_S25 denotes a three-convolutional-layer model with channel profile $[16, 32, 64]$, downsampling pattern PP, INT8 weights, and 25% target sparsity. The prefix D identifies the downsampling configuration, whereas the following two characters specify the operations adopted at the two spatial-resolution reduction stages. In this notation, P denotes max-pooling-based downsampling and S denotes stride-based downsampling. Therefore, DPP applies max-pooling at both reduction stages, DSS denotes a fully strided configuration, and DPS/DSP represent hybrid configurations.

The highest-mean-validation-efficiency solution is cand_003, in Figure 7, a compact three-layer architecture with channel profile $[16, 32, 64]$, PP downsampling, INT8 weights, and 25% target sparsity. The candidate achieves a mean validation-balanced accuracy of $96.83 \pm 0.45\%$ and a test-balanced accuracy of $82.94 \pm 1.42\%$ over five independent QAT runs. Its estimated transistor-level HWCost is 96,431,534, only approximately 0.007% higher than that of the minimum-HWCost representative. Despite this negligible resource increase, cand_003 improves the mean validation-balanced accuracy by approximately 0.40 percentage points relative to cand_013, making it the preferred efficiency-oriented architecture according to the revised multi-seed ranking.

The minimum-HWCost solution is cand_013, which has the same number of convolutional layers and the same channel profile as cand_003, but adopts fully strided SS downsampling and INT6 weights. Its estimated HWCost is 96,424,718 transistors. Candidate cand_013 achieves a mean validation-balanced accuracy of $96.43 \pm 0.41\%$ and a test-balanced accuracy of $83.32 \pm 1.54\%$. Thus, the lowest HWcost candidate achieves comparable test performance, but its lower mean validation balanced accuracy results in a slightly lower validation-efficiency score than cand_003. Under the adopted resource model, unstructured target sparsity does not directly reduce the instantiated hardware resources because zero-valued weights are retained and no zero-skipping or structured resource pruning is assumed.

**Table 11.** Selected deployable CNN candidates on PneumoniaMNIST after full multi-seed re-ranking, including architecture-search knob values, FP32/PTQ baselines, QAT performance, and estimated transistor-level hardware resources. QAT, FP32, and PTQ results are reported over five independent seeds.

| Property | Highest Mean Validation Efficiency | Highest Mean Validation Balanced Accuracy | Lowest HWCost |
|---|---|---|---|
| **Classification performance** | | | |
| Candidate ID | `cand_003` | `cand_048` | `cand_013` |
| Candidate configuration | `C3_16_32_64_DPP_W8_S25` | `C4_16_32_64_DSS_W8_S50` | `C3_16_32_64_DSS_W6_S25` |
| QAT validation-balanced accuracy [%] | $96.83 \pm 0.45$ | $98.03 \pm 0.48$ | $96.43 \pm 0.41$ |
| QAT test-balanced accuracy [%] | $82.94 \pm 1.42$ | $81.48 \pm 1.78$ | $83.32 \pm 1.54$ |
| FP32 test-balanced accuracy [%] | $81.73 \pm 2.29$ | $83.53 \pm 2.61$ | $81.98 \pm 1.06$ |
| FP32 test-balanced accuracy 95% CI [%] | [78.88, 84.57] | [80.29, 86.77] | [80.67, 83.30] |
| PTQ test-balanced accuracy [%] | $78.26 \pm 6.12$ | $84.07 \pm 2.12$ | $68.71 \pm 15.56$ |
| PTQ test-balanced accuracy 95% CI [%] | [70.65, 85.86] | [81.44, 86.70] | [49.39, 88.03] |
| Parameters | 23,426 | 60,354 | 23,426 |
| **Architecture-search knob values** | | | |
| Convolutional layers | 3 | 4 | 3 |
| Channel profile | [16, 32, 64] | [16, 32, 64] | [16, 32, 64] |
| Downsampling pattern | `PP` | `SS` | `SS` |
| Weight precision | `INT8` | `INT8` | `INT6` |
| Target sparsity | 25% | 50% | 25% |
| **Derived and fixed deployment settings** | | | |
| Pre-GAP feature map size | $16 \times 16$ | $16 \times 16$ | $16 \times 16$ |
| Activation precision | `UINT8` | `UINT8` | `UINT8` |
| Batch Normalization | Disabled | Disabled | Disabled |
| **Estimated hardware resources** | | | |
| $N_{\text{PMA}}$ | 57 | 121 | 57 |
| $N_{\text{adder}}$ | 2768 | 6928 | 2768 |
| $N_{\text{shifter}}$ | 2768 | 6928 | 2768 |
| $N_{\text{comparator}}$ | 273 | 353 | 225 |
| Estimated HWCost [transistors] | 96,431,534 | 206,647,054 | 96,424,718 |
| Relative HWCost | $1.00007\times$ | $2.14\times$ | $1.00\times$ |
| Mean validation efficiency | $1.004 \times 10^{-6}$ | $4.744 \times 10^{-7}$ | $1.000 \times 10^{-6}$ |

The highest-mean-validation-balanced accuracy solution is `cand_048`, a four-layer architecture with channel profile [16, 32, 64], fully strided `SS` downsampling, `INT8` weights, and 50% target sparsity. It reaches a mean validation balanced accuracy of $98.03 \pm 0.48\%$, improving over `cand_003` by approximately 1.20 percentage points on the validation set. However, its test-balanced accuracy is $81.48 \pm 1.78\%$, and its estimated HWCost increases to 206,647,054 transistors, corresponding to approximately $2.14\times$ the minimum estimated cost. Therefore, the higher validation performance does not translate into improved test set performance and is obtained at a substantially larger estimated hardware resource cost.

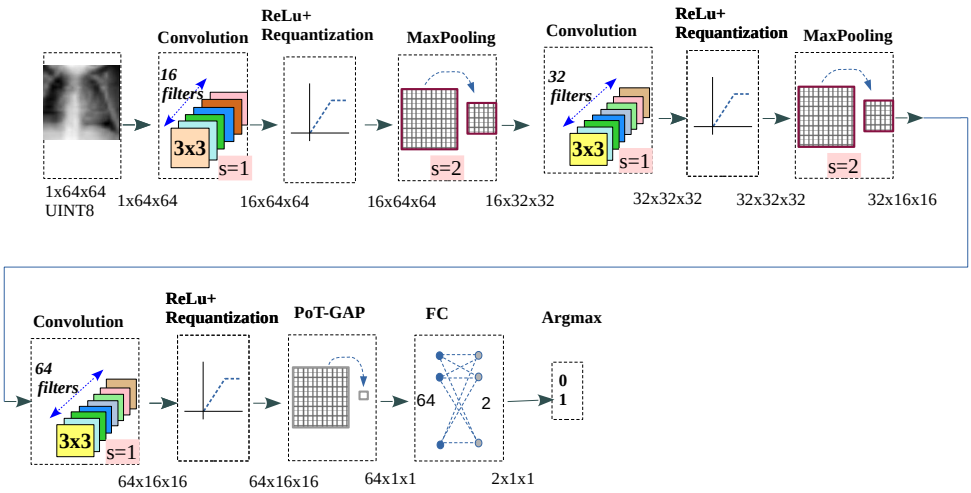



**Figure 7.** Architecture of the highest-mean-validation-efficiency CNN selected for PneumoniaMNIST after full multi-seed re-ranking. The network corresponds to candidate `cand_003` (`C3_16_32_64_DPP_W8_S25`) and employs three $3 \times 3$ convolutional stages with channel profile $[16, 32, 64]$, `INT8` weights, `UINT8` activations, and a target weight sparsity of 25%. Each convolution uses stride $s = 1$ and padding $p = 1$, and is followed by ReLU activation and power-of-two requantization. Spatial downsampling follows the `PP` configuration, with two $2 \times 2$ max-pooling layers with stride $s = 2$ applied after the first and second convolutional stages. The resulting spatial resolution is reduced from $64 \times 64$ to $32 \times 32$ and then to $16 \times 16$. The final $64 \times 16 \times 16$ feature map is processed by PoT-GAP and an integer fully connected classifier.

Overall, the revised PneumoniaMNIST results indicate that the dominant hardware cost difference among the selected representatives is associated with network depth and the resulting number of instantiated processing and arithmetic resources. In contrast, within the compact three-layer configuration, changing the downsampling pattern from `SS` to `PP` produces only a marginal HWCost variation, mainly associated with the additional comparator resources required by max-pooling. The adopted transistor-level metric should therefore be interpreted as a hardware resource estimate rather than as an accurate estimate of latency per inference or energy per inference. Among the ranked candidates, `cand_003` provides the most favorable validation-performance-to-resource trade-off, whereas `cand_013` represents the minimum-HWCost operating point and `cand_048` represents the highest-validation-balanced accuracy operating point. PneumoniaMNIST should be viewed as a cautionary case rather than as a clean accuracy-preservation result. Although all 64 candidates were ranked over five seeds, the pronounced validation–test gap suggests that validation-based selection on small, class-imbalanced splits is sensitive to seed variation and possible validation-set overfitting. Since FP32 and QAT test-balanced accuracies are comparable, this behavior appears to reflect split sensitivity more than a degradation caused by the quantized DA-oriented deployment constraints.

## 7. Conclusions and Future Work

This work presented a CNN architecture-optimization methodology intended to facilitate subsequent In-Memory Computing (IMC) implementation by identifying compact multiplierless integer-inference architectures with reduced estimated resource cost and limited accuracy degradation. Starting from compact CNN families, the proposed flow constrains the network architecture and arithmetic operations to obtain integer-deployable candidates compatible with the adopted DA-oriented inference model. The methodology combines architecture exploration, Quantization-Aware Training, integer-inference constraints, and analytical resource–cost estimation. Candidate networks are evaluated in terms of classification performance and estimated transistor-equivalent resource cost,

computed from the PMAs, adders, shifters, and comparators required by the adopted DA-oriented mapping. The resulting HWCost metric is intended as an architecture-level optimization criterion, not as a post-synthesis area, latency, or energy measurement.

The proposed approach was evaluated on four image-classification datasets spanning different image domains and difficulty levels: MNIST, FashionMNIST, CIFAR-10, and PneumoniaMNIST. Across the considered datasets, the search procedure identified compact candidates that reduce the estimated DA-oriented resource cost while preserving a large fraction of the classification performance of the highest-validation-accuracy candidates. The resulting architecture choices consistently reflected favorable performance-complexity trade-offs. Overall, the results show that substantial reductions in estimated DA-oriented resource complexity can be achieved through architecture optimization and Quantization-Aware Training, making the proposed methodology a useful front-end for subsequent implementation-level studies. In summary, the results confirm that reducing estimated DA-oriented resource cost generally involves an accuracy–resource trade-off. When the application can tolerate a small accuracy reduction, the analytical model indicates that substantially more compact CNN candidates can be selected; larger tolerated accuracy losses allow more aggressive reductions in estimated resource cost. Future work will focus on implementation-level validation of the selected candidates on FPGA and memory-centric hardware platforms, including synthesis, latency, energy, and memory traffic evaluation, as well as on extending the methodology to more advanced network topologies and larger-scale datasets.

**Author Contributions:** Conceptualization, I.A.K. and J.R.; methodology, I.A.K.; software, I.A.K.; validation, I.A.K.; formal analysis, I.A.K.; investigation, I.A.K. and J.R.; writing—original draft preparation, J.R. and I.A.K.; writing—review and editing, J.R. and I.A.K.; visualization, I.A.K. and J.R.; supervision, J.R.; project administration, J.R. and D.F.; funding acquisition, D.F. All authors have read and agreed to the published version of the manuscript.

**Funding:** This work was funded as part of the "Bavarian Chip-Design-Center" project by the Bavarian Ministry of Economic Affairs, Regional Development and Energy (funding, number:, R.M.F.-S.G.20-3410-2-17-14).

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

**Conflicts of Interest:** The authors declare no conflicts of interest.

## References

1. Horowitz, M. Computing's energy problem (and what we can do about it). In *Proceedings of the 2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC), San Francisco, CA, USA, 9–13 February 2014*; IEEE: New York, NY, USA, 2014; pp. 10–14. [CrossRef]
2. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.G.; Adam, H.; Kalenichenko, D. Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference. In *Proceedings of the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*; IEEE: New York, NY, USA, 2017; pp. 2704–2713.
3. Li, Y.; Gong, R.; Tan, X.; Yang, Y.; Hu, P.; Zhang, Q.; Yu, F.; Wang, W.; Gu, S. BRECQ: Pushing the Limit of Post-Training Quantization by Block Reconstruction. *arXiv* **2021**, arXiv:2102.05426.
4. Amirsoleimani, A.; Alibart, F.; Yon, V.; Xu, J.; Pazhouhandeh, M.R.; Ecoffey, S.; Beilliard, Y.; Genov, R.; Drouin, D. In-Memory Vector-Matrix Multiplication in Monolithic Complementary Metal–Oxide–Semiconductor-Memristor Integrated Circuits: Design Choices, Challenges, and Perspectives. *Adv. Intell. Syst.* **2020**, *2*, 2000115. [CrossRef]
5. Nägele, R.; Finkbeiner, J.; Stadtlander, V.; Grözing, M.; Berroth, M. Analog Multiply-Accumulate Cell with Multi-Bit Resolution for All-Analog AI Inference Accelerators. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2023**, *70*, 3509–3521. [CrossRef]

6. Seo, J.S.; Saikia, J.; Meng, J.; He, W.; Suh, H.S.; Anupreetham; Liao, Y.; Hasssan, A.; Yeo, I. Digital Versus Analog Artificial Intelligence Accelerators: Advances, trends, and emerging designs. *IEEE Solid-State Circuits Mag.* **2022**, *14*, 65–79. [CrossRef]
7. Zeller, F.; Reuben, J.; Fey, D. Multiplier-Free In-Memory Vector-Matrix Multiplication Using Distributed Arithmetic. In *Proceedings of the 2025 IEEE 18th International Symposium on Embedded Multicore/Many-Core Systems-on-Chip (MCSoC)*; IEEE: New York, NY, USA, 2025; pp. 718–726. [CrossRef]
8. Han, S.; Liu, X.; Mao, H.; Pu, J.; Pedram, A.; Horowitz, M.A.; Dally, W.J. EIE: Efficient inference engine on compressed deep neural network. In *Proceedings of the 43rd International Symposium on Computer Architecture*; ISCA '16; IEEE Press: New York, NY, USA, 2016; pp. 243–254. [CrossRef]
9. Reuben, J.; Zeller, F.; Seiler, B.; Fey, D. A Multiplierless Architecture for Image Convolution in Memory. *J. Low Power Electron. Appl.* **2025**, *15*, 63. [CrossRef]
10. White, S. Applications of distributed arithmetic to digital signal processing: A tutorial review. *IEEE ASSP Mag.* **1989**, *6*, 4–19. [CrossRef]
11. Mehendale, M.; Sharma, M.; Meher, P.K. DA-Based Circuits for Inner-Product Computation. In *Arithmetic Circuits for DSP Applications*; John Wiley & Sons, Ltd.: Hoboken, NJ, USA, 2017; Chapter 3, pp. 77–112 . [CrossRef]
12. Weste, N.; Harris, D. *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed.; Addison-Wesley Publishing Company: Reading, MA, USA, 2010.
13. Kang, H.J.; Yang, B.D. AoCStream: All-on-Chip CNN Accelerator with Stream-Based Line-Buffer Architecture and Accelerator-Aware Pruning. *Sensors* **2023**, *23*, 8104. [CrossRef] [PubMed]
14. Efstathiou, C.; Dimolikas, K.; Papaioannou, C.; Tsiatouhas, Y. Low Power and High Speed Static CMOS Digital Magnitude Comparators. In *Proceedings of the 2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*; IEEE: New York, NY, USA, 2018; pp. 249–252. [CrossRef]


**Disclaimer/Publisher's Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.