electronics

MDPI

Article

Real-Time Super-Resolution for Drone Imagery: A Low-Power, Low-Precision Approach with Hardware Acceleration

Güner Tatar [1,*] and Mahmud Esad Arar [2]

[1] Electrical Electronics Engineering, Istanbul Medeniyet University, Istanbul 34700, Türkiye
[2] Electrical Electronics Engineering, Fatih Sultan Mehmet Vakıf University, Istanbul 34445, Türkiye; mearar@fsm.edu.tr
* Correspondence: guner.tatar@medeniyet.edu.tr

Abstract

This paper presents a hardware–software co-design framework for real-time super-resolution (SR) of low-quality video on resource-constrained edge platforms. At its core is a compact residual network obtained by once-for-all (OFA) neural architecture search over the Residual Channel Attention Network (RCAN) design space, trained conventionally and then optimized with quantization-aware training (QAT) for deployment on an integer-only deep-learning processing unit (DPU). Loop tiling and data-flow scheduling are applied within a custom high-level synthesis (HLS) pre-processing pipeline that feeds the DPU, and a per-directive ablation isolates the contribution of each optimization to post-route resource usage and timing. Deployed on a Kria KV260 board with a 128 × 128 network input, the INT8 network sustains 96.37 FPS at the ×2 scale at a measured board power of 5.38 W, corresponding to 6.32 Mpixel/s of reconstructed output at 1.17 Mpixel/J, within 63.2% of the device LUT budget and with timing closed at 275 MHz. Relative to the FP32 model, INT8 quantization costs 0.274 dB of peak signal-to-noise ratio (PSNR) on Set5, 0.172 dB on Set14, 0.116 dB on B100, and 0.146 dB on Urban100, a loss dominated (81–90%) by activation rather than weight quantization. On a held-out UAV subset drawn from VisDrone2019, which is the operating domain the system targets, the network reconstructs at 25.94 dB and 0.748 SSIM. These results show that a twenty-three-layer residual SR network can be deployed within a 5.38 W envelope on a low-cost integer-only edge FPGA, making the approach suitable for autonomous systems, robotics, and airborne surveillance.

Keywords: edge computing; FPGA acceleration; image super-resolution; low-precision neural networks; real-time processing; remote sensing; UAV imagery

Check for updates

Academic Editor: Vijay Kakani

Received: 17 July 2026
Revised: 2 August 2026
Accepted: 6 August 2026
Published: 8 August 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

Uncrewed aerial vehicles (UAVs) have become routine instruments for agriculture, infrastructure inspection, environmental monitoring, and security, owing to their mobility and their ability to collect imagery in dynamic, otherwise inaccessible environments. That same mobility, however, imposes tight limits on onboard compute, communication bandwidth, and sensor quality, so that the video a UAV can capture and transmit in real time is frequently degraded relative to what its mission requires [1,2]. This limitation propagates directly into downstream tasks such as object detection and scene understanding.

Single-image SR offers a natural restoration; given a low-resolution (LR) frame, SR reconstructs a high-resolution (HR) estimate by recovering high-frequency detail lost during acquisition [3–5]. Deploying SR onboard a UAV, however, is not simply a matter of porting

an existing network. The strongest SR architectures, including RCAN [3], were designed for offline GPU-class inference and are too computationally expensive for a power- and weight-constrained airframe; even a modest network must contend with motion blur, changing illumination, and a hard real-time deadline once airborne. Meeting these constraints calls for Hw-Sw co-design rather than model design in isolation [6–10]. Among the available accelerator classes, GPUs deliver high throughput at a power cost that few UAV platforms can afford, and ASICs offer efficiency without the flexibility an evolving deep-learning (DL) pipeline requires. FPGAs occupy the middle ground, combining reconfigurability with power efficiency, and are the platform adopted in this work [11].

Among FPGA-based accelerators, DPUs [12] are a natural fit, pairing the fabric's reconfigurability with a fixed, efficient instruction set for convolutional inference. Applying a DPU to UAV-based SR nonetheless requires targeted co-design: a network compact enough for integer-only execution, together with a pre- and post-processing datapath engineered to the same throughput budget as the network itself. This is the gap the present work addresses.

The main contributions of this work are as follows:

1. Custom DPU Deployment: A custom DPU-Target Reference Design (TRD) optimized for real-time SR tasks is designed and implemented on the Kria KV260 board, enabling efficient parallel processing of SR operations.
2. QAT for INT8 Deployment: QAT reduces the network to INT8 precision so that it can execute on the integer-only DPU. The fidelity cost is quantified per dataset and per scale, and decomposed into the parts attributable to weight and to activation quantization.
3. Hardware–Software Co-Design Optimizations: Pipeline scheduling, loop tiling, dataflow execution, and LUTRAM-bound array partitioning are integrated in the HLS pre-processing IP, the last of these keeping the tile buffers clear of the block RAM required by the DPU core. The contribution of each directive is measured against a matched control, namely the same function on the same device without pragmas, and reported as post-route resource and timing figures.
4. Drone-Specific SR Evaluation: The system is validated on a real-time drone video stream at $\times 2$, $\times 3$, and $\times 4$ scaling factors using the standard benchmarks (Set5, Set14, B100, Urban100) together with a UAV-representative subset of the public VisDrone2019 collection.
5. MPSoC-FPGA Optimization: SR algorithms were optimized for MPSoC-FPGA platforms, providing a balance of flexibility and performance for real-time deployment. HLS tools were employed to customize the hardware for specific SR operations.

To support reproducibility, the source code, the deployed model checkpoints, the HLS project, and the evaluation scripts will be made publicly available on GitHub upon acceptance.

The remainder of this paper is organized as follows. Section 1 provides context, motivation, and background. Section 2 details the hardware platform, optimization strategies, datasets, training procedures, and implementation. Section 3 reports experimental results on power consumption, inference efficiency, and FPGA resource utilization. Section 4 presents the conclusion and future work.

1.1. Motivation and Background

This work is motivated by the gap between state-of-the-art SR techniques and their practical deployment on drones. Real-time video enhancement matters most precisely where it is hardest to provide: in dynamic scenes such as urban corridors or agricultural fields, where high-resolution imagery is a prerequisite for reliable object detection and decision-making, and where the platform delivering it has the least power and compute to spare. We address this by treating precision, network capacity, and hardware scheduling as a single design space rather than three separate problems, combining QAT with FPGA-level

architectural refinement (pipeline scheduling, loop tiling, and data-flow execution) to raise throughput while reducing both latency and resource footprint.

Building on established SR architectures [3,13–15], the optimizations proposed here are shaped by the operating constraints of drone-based imagery, specifically: power, memory, and timing budgets an order of magnitude tighter than those assumed by GPU-oriented SR research. The system pursues this balance through reduced arithmetic precision, a compact network, and fine-grained control over the scheduling of the pre-processing datapath.

1.2. PSNR and SSIM Metrics

PSNR and SSIM are the two metrics used throughout this paper to evaluate reconstruction quality. PSNR, defined in (1), compares the maximum pixel value R to the mean squared error (MSE) between the reconstructed and reference HR images; higher values indicate better pixel-level fidelity, though PSNR does not always track perceived visual quality, since it weighs pixel differences rather than structural content.

$$\mathrm{PSNR} = 10 \cdot \log_{10}\left(\frac{R^2}{\mathrm{MSE}}\right) \tag{1}$$

where R is the maximum pixel value (255 for an 8-bit image). SSIM, defined in (2), instead compares brightness, contrast, and structural pattern between the two images and returns a value in $[0, 1]$, with higher values indicating closer alignment with human visual perception.

$$\mathrm{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2}$$

Here x and y are the two images being compared; $\mu_x, \mu_y, \sigma_x, \sigma_y, \sigma_{xy}$ are their means, variances, and covariance; and $C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$ are stabilizing constants, with L the pixel-value range and $K_1 = 0.01$, $K_2 = 0.03$ by convention.

1.3. Research Gap and Findings

Prior work on SR under environmental and hardware constraints has generally optimized one axis at the expense of another. Vedavyas et al. [16] propose an FPGA-based real-time enhancement system for drone imagery, reporting MSE, RMSE, and PSNR and demonstrating CLAHE for foggy conditions and ESRGAN for fine object enhancement; their evaluation, however, centers on frame rate and does not report power consumption or resource utilization, which this work treats as first-class metrics alongside fidelity.

Cheng et al. [17] use recursive squeeze-and-excitation blocks to improve feature efficiency in software, reaching faster inference with fewer parameters, but do not address the hardware side of deployment; here we target the hardware directly, reducing resource usage and power draw on an MPSoC-FPGA. Zhao et al. [18] report a lightweight network and accelerator achieving 37.44 dB PSNR, but on a high-end development board that limits accessibility; the present system reaches comparable or slightly higher PSNR on a substantially cheaper board, and its 8-bit quantization offers a better efficiency–accuracy trade-off than the 16-bit quantization used in that work.

Zhang et al. [3] introduced RCAN for single-image super-resolution, leveraging a residual-in-residual structure and channel attention mechanisms. Despite its effectiveness, its reliance on an energy-intensive Nvidia Titan Xp limits its practicality for power-constrained drone applications. This work deploys a compact network derived from the RCAN design space on the Kria KV260 development board. In FP32, the deployed model trails the reference RCAN by 0.6–2.4 dB across the four standard benchmarks (37.66 dB against 38.27 dB on Set5), which is the price of an 86.6 K parameter network operating with 180× fewer parameters than the original 15.6 M parameter design (Section 3.3). Under

INT8 quantization, the ×2 model, the only scale evaluated against a properly QAT-trained checkpoint (the one published with the model), loses at most 0.27 dB (Set5) and 0.14 dB (Urban100), well within the range normally expected from INT8 QAT. A concise summary of state-of-the-art studies is provided in Table 1.

Table 1. Performance comparison of state-of-the-art studies for image SR applications: an analysis based on PSNR and SSIM metrics.

Ref.	Precision Weights-Act.	Scale	Set5		Set14		B100		Urban100		VisDrone21	
			PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM	PSNR	SSIM
[3]	N	×2	**38.27**	**0.9614**	**34.12**	**0.9216**	**32.41**	**0.9027**	**33.34**	**0.9384**	N	N
[6]	10–14		37.53	0.9587	33.03	0.9124	31.90	0.8960	30.76	0.9140	N	N
[18]	16–16		37.44	0.9592	32.74	0.9120	N	N	30.29	0.9088	N	N
[19]	13–13		37.05	0.9560	32.66	0.9090	31.53	0.8920	29.88	0.9020	N	N
[20]	N		36.66	0.9542	32.45	0.9067	31.36	0.8879	29.50	0.8946	N	N
[21]	32–32		37.52	0.9591	33.08	0.9130	31.08	0.8950	30.41	0.9101	N	N
[22]	N		38.24	0.9614	34.01	0.9212	32.34	0.9017	32.89	0.9353	N	N
This Study	FP32		37.6584	0.9592	33.1837	0.9140	31.9012	0.8966	30.9856	0.9166	25.9374	0.7481
	INT8		37.3844	0.9560	33.0118	0.9105	31.7852	0.8935	30.8393	0.9126	24.9959	0.7205
[3]	N	×3	**34.74**	**0.9299**	**30.65**	**0.8482**	**29.32**	**0.8111**	**29.09**	**0.8702**	N	N
[6]	10–14		33.66	0.9213	29.77	0.8314	28.83	0.7990	27.14	0.8290	N	N
[18]	16–16		33.43	0.9210	29.39	0.8307	N	N	27.78	0.8435	N	N
[19]	13–13		33.18	0.9140	29.37	0.8240	28.53	0.7910	26.43	0.8080	N	N
[20]	N		32.75	0.9090	29.30	0.8215	28.41	0.7863	26.24	0.7989	N	N
[21]	32–32		33.82	0.9227	29.87	0.8320	28.82	0.7980	27.07	0.8280	N	N
[22]	N		34.71	0.9296	30.57	0.8465	29.26	0.8093	28.80	0.8653	N	N
This Study	FP32		33.8660	0.9220	29.9320	0.8342	28.8256	0.7991	27.2114	0.8313	26.5106	0.7591
	INT8		33.7461	0.9205	29.8755	0.8329	28.7790	0.7979	27.0787	0.8273	26.5312	0.7599
[3]	N	×4	**32.63**	**0.9002**	**28.87**	**0.7889**	27.77	0.7436	**26.82**	**0.8087**	N	N
[6]	10–14		31.35	0.8838	28.01	0.7674	**31.90**	**0.8960**	25.18	0.7524	N	N
[18]	16–16		31.09	0.8798	27.58	0.7658	N	N	24.89	0.7443	N	N
[19]	13–13		30.72	0.8660	27.61	0.7550	26.98	0.7150	24.62	0.7280	N	N
[20]	N		30.48	0.8628	27.50	0.7513	26.90	0.7101	24.52	0.7221	N	N
[21]	32–32		31.54	0.8850	28.19	0.7720	27.32	0.7270	25.21	0.7560	N	N
[22]	N		32.47	0.8990	28.81	0.7871	27.72	0.7419	26.61	0.8028	N	N
This Study	FP32		31.5659	0.8841	28.1684	0.7708	27.2791	0.7277	25.2133	0.7568	26.6607	0.7433
	INT8		31.5393	0.8836	28.1513	0.7705	27.2688	0.7275	25.2041	0.7565	26.6567	0.7430

Bold denotes the most favorable value in each column; 'N' = not reported. Comparisons across rows involve different hardware platforms and precision formats and should be read as indicative.

2. Hardware Platform Setup

The target platform is a custom FPGA-based system built on the Kria KV260 Vision Board (Figure 1) and designed to accelerate vision-AI inference through joint hardware and software customization. The development environment was Vitis 2022.1 and Vivado 2022.1.

The Vitis platform creation workflow [23] was followed to build a custom Board Support Package (BSP) tailored to the project's requirements: the hardware description was generated in Vivado, and the component kernels were integrated into the overlay. Pipeline, data-flow, and loop-tiling directives were applied within the HLS pre-processing IP, whose post-route cost is quantified per directive in Section 3.1; the network itself executes on the DPUCZDX8G core, whose schedule is fixed by the Vitis AI compiler. PetaLinux was used to build a customized embedded Linux environment for system integration.

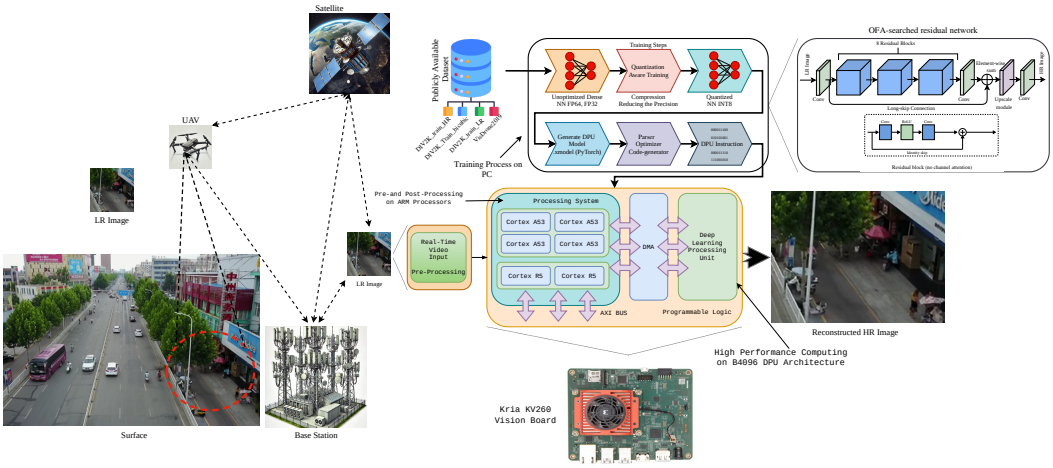


Figure 1. Detailed top-level overview of the proposed study.

2.1. Hardware Design

The hardware design centers on the Zynq UltraScale+ MPSoC processing system (PS), a quad-core ARM Cortex-A53 paired with an ARM Cortex-R5F real-time unit, which configures the programmable-logic (PL) side of the architecture and manages data flow between the two. The PL carries the DPUCZDX8G B4096 core and the HLS pre-processing IP, both synthesized using the programmable logic resources of the device, including LUTs, flip-flops, DSP slices, block RAM, and UltraRAM. An AXI joins the two blocks, with the pre-processing IP exposing AXI-Stream ports for pixel traffic and an AXI-Lite port for control. Digital hardware development used Vivado 2022.1, with HLS applied to the PL-targeted components. Frame decoding, buffer management, and post-processing run on the ARM cores in C++ with OpenCV, while pixel normalization is performed in the PL by the HLS IP of Algorithm 1; the Vitis AI runtime couples the ARM cores to the DPU for network execution. The resulting overlay is shown in Figure 2.

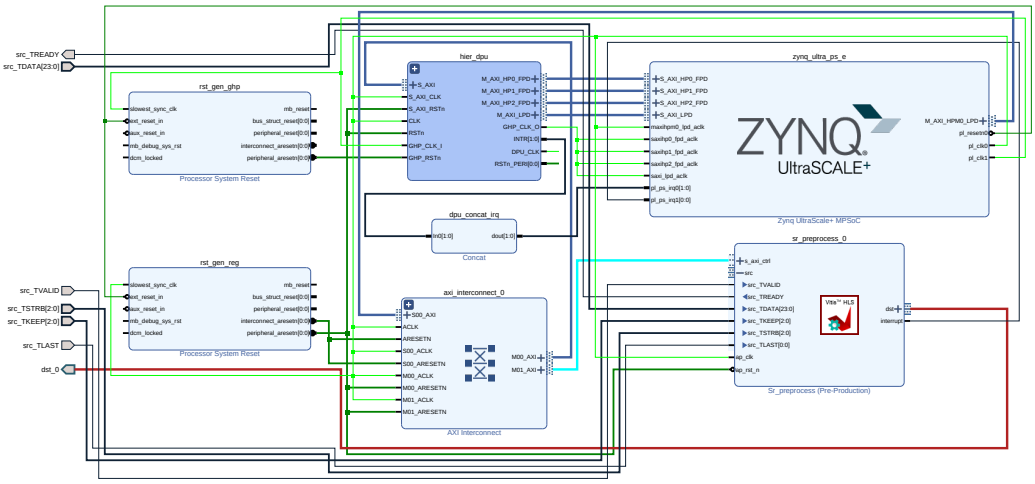


Figure 2. Custom overlay design including the DPUCZDX8G B4096 core and HLS SR pre-processing IP.

A single fixed accelerator configuration is rarely efficient across diverse DL models, so the DPU was configured for this workload rather than left at its default settings. The B4096 architecture was selected as the largest configuration that leaves sufficient fabric for the pre-processing IP, and the tile buffers of that IP were bound to distributed LUTRAM so as not to compete for the block RAM the DPU requires; this division of on-chip memory is what allows the two blocks to coexist within the device budget (Section 3.1). DPUs are deep-

pipelined architectures composed of programmable building blocks, enabling configuration for specific applications. A custom SD card image was built to provide flexibility in modifying hardware and software components. The final platform configuration and the DPU architecture criteria are summarized in Table 2.

Algorithm 1 HLS SR pre-processing IP: loop tiling, pipelining, and data-flow. The tiling and `DATAFLOW` directives govern on-chip buffering of the pixel-wise normalization stage only; they do not partition the network inference, which the DPU executes full-frame.

Constants: TILE $= 16$, $N_{\text{tiles}} = \lceil H/\text{TILE} \rceil \times \lceil W/\text{TILE} \rceil$ // $8 \times 8 = 64$ *at the deployed* 128×128 *geometry*

Function `sr_preprocess(src, dst, W, H)`
  `#pragma HLS INTERFACE axis port=src` // *`ap_axiu`*$\langle 24 \rangle$*: packed 8-bit R|G|B*
  `#pragma HLS INTERFACE axis port=dst` // *`ap_axiu`*$\langle 32 \rangle$*: one float per channel*
  `#pragma HLS INTERFACE s_axilite port=width,height,return bundle=ctrl`

  declare tile buffers $R, G, B \in$ `float`$^{\text{TILE}\times\text{TILE}}$
  `#pragma HLS ARRAY_PARTITION variable=R cyclic factor=4 dim=2`
  `#pragma HLS BIND_STORAGE variable=R type=RAM_2P impl=LUTRAM`

for $t \leftarrow 0$ **to** $N_{\text{tiles}} - 1$ **do**
    `#pragma HLS DATAFLOW`
    $r_0 \leftarrow (t \div 120) \cdot \text{TILE}, \quad c_0 \leftarrow (t \bmod 120) \cdot \text{TILE}$

    // Stage 1: *`stage_read`**, consume tile and normalize*
  for $r \leftarrow 0$ **to** $\text{TILE} - 1$ **do**
      `#pragma HLS PIPELINE II=1` *// on the row loop*
    for $c \leftarrow 0$ **to** $\text{TILE} - 1$ **do**
        `#pragma HLS UNROLL factor=4`
        // superseded: the PIPELINE above forces a complete
        // unroll of factor 16 (HLS 214-291/214-186)
      if $r_0 + r < H$ **and** $c_0 + c < W$ **then**
          px $\leftarrow$ `src.read()`
          $R[r][c] \leftarrow$ `(float)` px[23:16] / 255.0f
          $G[r][c] \leftarrow$ `(float)` px[15:8] / 255.0f
          $B[r][c] \leftarrow$ `(float)` px[7:0] / 255.0f
      else
          $R[r][c], G[r][c], B[r][c] \leftarrow 0$ *// boundary pad*
      end if
    end for
  end for

    // Stage 2: *`stage_write`**, emit 3 float words per pixel*
  for $r \leftarrow 0$ **to** $\text{TILE} - 1$ **do**
      `#pragma HLS PIPELINE II=1`
    for $c \leftarrow 0$ **to** $\text{TILE} - 1$ **do**
        `#pragma HLS UNROLL factor=4` *// likewise superseded*
      if $r_0 + r < H$ **and** $c_0 + c < W$ **then**
          `dst.write(`$R[r][c]$`);` `dst.write(`$G[r][c]$`);`
          `dst.write(`$B[r][c]$`)` *// TLAST on final pixel*
      end if
    end for
  end for
end for
End Function

Table 2. DPU (DPUCZDX8G) configuration for TRD.

DPU Configuration	TRD Kria KV260	Model Recompile Required
DPU_Clk_MHz	275	No
Architecture	B4096	Yes
DPU Number	1	No
Channel Augmentation	Enabled	Yes
DSP48 Max Cascade	4	No
Depthwise Conv.	Enabled	Yes
RAM & DSP48 Usage	High/High	Yes
URAM	48	Yes
Average Pooling	Enabled	Yes
Conv. ReLU Type	ReLU + ReLU6	Yes

2.2. Optimization Strategies

2.2.1. Pipeline Architecture Optimization

The processing pipeline is organized into five stages within a PetaLinux environment:

1. Data Pre-processing: Decoded frames are cropped to the network input geometry on the ARM cores and normalized to the $[0, 1]$ range by the HLS IP in the PL.
2. Model Execution: The INT8 network is executed on the DPU to reconstruct the high-resolution estimate.
3. Post-processing: Reconstructed outputs are clipped to the valid pixel range and converted back to the display colour space on the ARM cores.
4. Final Output: Resulting HR frames are composited to the display and made available to downstream tasks such as object detection and scene analysis.

Within the pre-processing IP, each stage (pixel fetching, normalization, and streaming to the DPU) is scheduled with PIPELINE directives to maximize instruction-level parallelism, so that successive stages overlap rather than execute in strict sequence. This overlap removes inter-stage idle time within the IP and keeps its latency well below that of the inference stage, so that the pre-processing datapath does not constrain the sustained frame rate (Section 3).

2.2.2. Loop Tiling Optimization

Loop tiling bounds the on-chip buffer size and enables overlapped (dataflow) execution across tiles during SR pre-processing by subdividing each input frame into smaller, independently processed tiles that stream through the pipeline with minimal idle time between them. A tile size of 16×16 pixels was chosen to fit within the KV260's on-chip BRAM/LUTRAM budget. It keeps each tile's buffers small enough to map efficiently onto distributed LUTRAM and lets DATAFLOW overlap the read and write stages of consecutive tiles, keeping the pre-processing latency well below that of the inference stage.

2.2.3. Data-Flow Architecture Optimization

A dataflow architecture maps each functional stage of the SR pipeline to an independent processing element connected by continuous AXI-Stream transfers, removing the synchronization barriers and memory-stall cycles of sequential execution and making fuller use of the FPGA's available parallelism. The contribution of the HLS optimization directives is evaluated independently from the effects of INT8 quantization through a

stepwise ablation study. Their impact on the HLS pre-processing IP is further analyzed using post-implementation resource and timing results. The pseudocode below reflects the actual HLS implementation, with explicit pragmas for pipeline scheduling, loop tiling, and dataflow execution. It specifies the pre-processing IP alone; the super-resolution network itself executes full-frame on the DPU, as detailed after the listing.

Algorithm 1 reproduces the HLS SR pre-processing IP implemented in Vitis HLS 2022.1, including the arithmetic precision of the tile buffers, the placement of the PIPELINE directive, and the output stream format. The IP performs only pixel-level pre-processing by reading raw pixels from the input AXI-Stream, normalizing them from the $[0, 255]$ integer range to the $[0, 1]$ range, and writing the normalized pixels to the output AXI-Stream.

Four implementation details govern the hardware implementation. First, the tile buffers store single-precision `float` values, and normalization is implemented as a floating-point division by 255. The synthesis reports indicate three `fdiv` operators mapped to programmable logic, each with a latency of 29 clock cycles. Second, `#pragma HLS PIPELINE II = 1` is applied to the row loop rather than the innermost pixel loop, causing Vitis HLS to fully unroll the pixel loop instead of applying the requested factor of four specified by the accompanying UNROLL directive. Third, the input stream transfers one packed 24-bit RGB word per transaction, whereas the output stream produces three 32-bit floating-point values per pixel. This mismatch, rather than the unroll factor, determines the achieved interval of 790 cycles for each 16×16 tile. Finally, `#pragma HLS DATAFLOW` is applied at the tile level, enabling overlap between the read stage of one tile and the write stage of the previous tile, while the partitioned tile buffers are mapped to distributed LUTRAM to avoid contention with the block RAM resources required by the DPU. The system comprises two compute engines. The first is the HLS pre-processing IP described in Algorithm 1, which resides in the programmable logic, processes the input frame as a stream of independent pixels, and performs a single floating-point normalization per pixel. Because this operation has no spatial dependency, the 16×16 tile functions only as a buffering unit, defining both the granularity of the LUTRAM buffers and the level at which DATAFLOW overlaps the read stage of one tile with the write stage of the previous one. It is not a receptive-field tile; therefore, no boundary handling, halo exchange, or overlap-and-add operation is required.

The second compute engine is the DPUCZDX8G B4096 core, which receives the entire normalized frame and executes the compiled `xmodel` through the Vitis AI runtime. Consequently, the tiling and `DATAFLOW` directives influence only the latency and resource utilization of the HLS pre-processing stage and do not affect the execution schedule of DPU inference.

2.3. Datasets

This study uses the DIV2K [24], VisDrone [2], and standard super-resolution benchmark datasets for training and testing. Table 3 summarizes these datasets and their roles in training and evaluation. The DIV2K dataset is widely used for super-resolution training and contains 900 HR images with resolutions up to 2040×1356 pixels, together with their bicubic-downsampled LR counterparts at $\times 2$, $\times 3$, and $\times 4$. These 900 images comprise the 800-image training split and the 100-image validation split defined by the NTIRE 2017 challenge. The two splits serve different purposes and are never mixed: the training split is used for network optimization, whereas the validation split is reserved for model selection and quantization calibration, as described in Section 2.4. For evaluation, we additionally use a set of standard super-resolution benchmarks –Set5,Set14,B100, andUrban100–together with a UAV-captured subset from VisDrone, each provided with both HR and LR images at $\times 2$, $\times 3$, and $\times 4$ scales. These datasets evaluate the model's performance on natural and urban imagery at multiple resolutions, as well as on genuinely UAV-captured imagery.

Table 3. Use of datasets.

Datasets	DIV2K							VisDrone							Benchmark															
	DIV2K (HR)	DIV2K (LR) Bicubic			DIV2K (LR)			VisDrone (HR)	VisDrone (LR) Bicubic			VisDrone (LR)			B100				Urban100				VisDrone21	Set14				Set5			
															LR Bic.			HR	LR Bic.			HR	HR	LR Bic.			HR	LR Bic.			HR
		×2	×3	×4	×2	×3	×4		×2	×3	×4	×2	×3	×4	×2	×3	×4		×2	×3	×4			×2	×3	×4		×2	×3	×4	
Train	800	800	800	800	800	800	800	N															N							
Test	N														100	100	100	100	100	100	100	100	21	14	14	14	14	5	5	5	5

The UAV-representative data used here is drawn from the publicly available VisDrone2019 dataset [2], which provides drone-captured imagery distinct in content from the four standard SR benchmarks while remaining reproducible and citable.

We draw a subset of 121 images from VisDrone2019-DET, one per source sequence, so that no two images share a flight or video clip, spanning a native resolution range of 960×540 to 1920×1080: 21 held-out images from disjoint sequences form the evaluation set (VisDrone21), and the remaining 100 (VisDrone) are reserved for future domain adaptation.

The acquisition conditions of the UAV dataset are summarized in Table 4. LR counterparts were generated using bicubic downsampling with scaling factors of $\times 2$, $\times 3$, and $\times 4$ following the same modcrop-then-decimate protocol adopted by DIV2K and the standard benchmark datasets. Consequently, the reported PSNR and SSIM values remain directly comparable with those presented in Table 1. The LR images of both VisDrone subsets were generated solely via bicubic decimation, ensuring consistency with the degradation model used in DIV2K and the benchmark datasets.

Table 4. Acquisition conditions of the VisDrone and VisDrone21 subsets, as documented for the source dataset [2]. These properties describe VisDrone2019 as a whole; the 121-image subset used here carries no per-image altitude, time-of-day, or weather labels.

Property	Value
Source dataset	VisDrone2019-DET [2], CC BY-NC-SA 3.0
Platform/sensor	DJI Mavic and Phantom series UAVs (multiple sensors)
Native capture	JPEG, subset resolution 960×540 to 1920×1080
Images (train/test)	100 (VisDrone)/21 (VisDrone21), disjoint source sequences
Geographic coverage	14 cities
Altitude/viewpoint	Varying altitudes and viewpoints (urban and suburban); not individually labelled per image
Scene categories	Urban and suburban scenes
Time of day/weather	Cloudy, sunny and night conditions represented in the source dataset
LR generation	Bicubic decimation at $\times 2$, $\times 3$, $\times 4$ (DIV2K protocol)

2.4. Training and Testing Procedure

The training phase uses images from the DIV2K datasets (HR, LR bicubic, and LR subsets); fine-tuning the $\times 3$ and $\times 4$ tails takes approximately 5.5 h on the development workstation, and evaluation is conducted on the benchmark datasets described above. The deployed model is a compact residual network obtained via once-for-all (OFA) neural architecture search under a latency constraint, rather than the original, full-sized RCAN configuration. The searched network consists of a single residual group containing 8 residual blocks with a base feature width of 16 channels; each block's internal expansion width is individually determined by the OFA search (16, 32, 32, 48, 16, 16, 64, and 24 channels for blocks 1–8, respectively), followed by a sub-pixel (pixel-shuffle) upsampling head. The searched configuration does not retain the explicit channel-attention (CALayer) branch of the original RCAB design, and since channel attention is what the name RCAN denotes, the deployed model is described here as an OFA-searched residual network derived from the RCAN design space: the residual-in-residual topology, long skip connection, and pixel-shuffle tail are preserved, while the squeeze-and-excitation branch is not. Its removal is architectural. The CALayer applies global average pooling, two 1×1 convolutions, and a sigmoid, then rescales the feature map channel-wise; on the B4096, the pooling and channel-wise multiply are not fused with the surrounding convolutions, so each branch forces a DDR round trip and a synchronization point in an otherwise streaming schedule.

In a network whose widest block is 64 channels, that latency penalty outweighs the accuracy benefit under the search's latency budget. Channel-wise capacity is instead allocated statically through the per-block expansion widths. The network totals 86,627 trainable parameters, verified by direct inspection of the exported checkpoint and several orders of magnitude below the ∼16 M-parameter configuration of Zhang et al. [3]. ReLU activations are used throughout.

The provenance of this architecture is as follows. The backbone is the OFA-RCAN model distributed with the Vitis AI Model Zoo [15], used as released and pretrained; the once-for-all search that produced it was carried out by the authors of that release for the $\times 2$ scaling factor only. Parsing the released model confirms 86,627 parameters and 1.75×10^5 multiply–accumulate operations per input pixel, matching the published model card. For deployment on the KV260 the network was recompiled for a 128×128 input tile, which is the geometry at which all runtime measurements in Section 3 are reported; the reconstruction tails for $\times 3$ and $\times 4$ were instantiated and fine-tuned as described below.

The contribution of this work with respect to the network is therefore one of deployment and characterization rather than of architecture search: the integration of this model with a custom DPU configuration and a purpose-built pre-processing accelerator, recompiled for the region-of-interest geometry the application requires, together with the resource, timing, and power characterization of that integration reported in Sections 3.1 and 3.

The released model is specific to $\times 2$: its reconstruction tail is a single 3×3 convolution producing $16 \cdot 2^2 = 64$ channels followed by a pixel shuffle of factor two, and no $\times 3$ or $\times 4$ variant is published. The backbone, by contrast, is scale-independent, since the eight residual blocks operate entirely in the low-resolution feature space. Models for the remaining two scaling factors were therefore obtained by retaining the searched backbone, replacing the tail with a convolution producing $16r^2$ channels followed by a pixel shuffle of factor r, initializing the backbone from the published $\times 2$ weights, and fine-tuning on DIV2K under the degradation model used for evaluation. Table 1 distinguishes these scales accordingly: the $\times 2$ row is the released model evaluated as distributed, while the $\times 3$ and $\times 4$ rows are the fine-tuned derivatives.

The deployed backbone was searched from the RCAN design space [3] under a latency constraint, as detailed in Section 2.4; fine-tuning the $\times 3$ and $\times 4$ reconstruction tails on this backbone follows the schedule below. The tails were fine-tuned for 300 epochs using the Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1 \times 10^{-8}$, and no weight decay or gradient clipping. The learning rate followed a cosine annealing schedule from $\eta_0 = 2 \times 10^{-4}$ to $\eta_{\min} = 1 \times 10^{-6}$:

$$\eta_t = \eta_{\min} + (\eta_0 - \eta_{\min}) \frac{1 + \cos\left(\frac{t}{300}\pi\right)}{2} \tag{3}$$

The batch size was 16, with 96×96 HR training patches (LR patches 48×48, 32×32, and 24×24 for $\times 2$, $\times 3$, and $\times 4$ respectively), drawn from a pre-extracted patch bank to keep the data path off the critical loop. Data augmentation comprised random horizontal flipping and 90° rotation, and the L_1 (MAE) pixel loss was used as the training objective. The $\times 2$ configuration is the published, pre-trained model, used as distributed; the $\times 3$ and $\times 4$ tails were fine-tuned with the backbone initialized from these published $\times 2$ weights and only the reconstruction tail freshly instantiated, as described above.

Model selection is carried out on a held-out validation split rather than on the evaluation benchmarks, keeping three roles of the data strictly disjointed. Training uses the 800-image DIV2K training split together with the 100-image VisDrone training subset. Model selection, comprising the choice of OFA sub-network, the stopping epoch of the FP32 schedule, and the stopping epoch of QAT fine-tuning, uses the 100-image DIV2K

validation split, disjoint from both the training split and every benchmark. Final evaluation uses Set5, Set14, B100, Urban100, and VisDrone21, each read exactly once after all architectural and training decisions have been frozen, so the results in Table 1 are free of test-set leakage. The same 100 DIV2K validation images are also used for post-training quantization calibration, described next; since calibration only estimates activation ranges and never selects between candidate models, this does not couple selection to evaluation.

QAT is a widely used method for optimizing DL models for deployment on resource-constrained devices [25,26]. Unlike post-training quantization (PTQ), QAT simulates quantization effects during training, letting the model adapt to the precision loss introduced by INT8 representations [27]. Training with low-precision weights and activations was shown to be feasible without prohibitive accuracy loss by Hubara et al. [28], and the design choices that govern the resulting trade-off, in particular per-channel versus per-tensor scaling and the placement of quantization nodes, are surveyed by Krishnamoorthi [29]; both inform the configuration adopted here. The deployed $\times 2$ network is distributed with QAT already applied, and the quantized checkpoint is used as released. The procedure behind it, reproducible from the scripts shipped with the model, applies symmetric uniform INT8 quantization to weights and activations in all convolutional layers (per-channel for weights, per-tensor for activations) through the Vitis AI quantizer (`vai_q_pytorch`, release 3.0), with fake-quantization nodes inserted after each convolutional layer and activation using the straight-through estimator (STE) [30] and learned quantizer thresholds [31]. Fine-tuning runs for 250 epochs at an initial learning rate of 5×10^{-3} with a batch size of 16. Bilinear upsampling and pixel-shuffle layers run at full precision on the ARM PS because the B4096 DPU does not support them natively. The $\times 3$ and $\times 4$ tails introduced in this work have not been through QAT; their INT8 figures in Table 1 are weight-only quantization, and are labeled as such. The quantization function is defined as:

$$q_x = \text{clip}\left(\text{round}\left(\frac{x}{s_x}\right), \alpha_q, \beta_q\right) \tag{4}$$

where x is the floating-point value, s_x is the scale factor, and α_q, β_q are the clipping bounds (in the quantized/integer domain). De-quantization reconstructs the approximation as $\hat{x} = s_x \times q_x$. Since rounding is non-differentiable, the STE [30] approximates the gradient of the de-quantized value with respect to the original floating-point input as

$$\frac{\partial \hat{x}}{\partial x} \approx \begin{cases} 1, & \text{if } s_x\,\alpha_q \le x \le s_x\,\beta_q \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

i.e., the gradient passes through unchanged inside the representable range of the quantizer (mapped back to the original scale) and is clipped to zero outside it. Symmetric uniform quantization is employed with $\alpha_q = -2^{b-1}$ and $\beta_q = 2^{b-1} - 1$, well-suited for integer arithmetic on FPGA-based accelerators.

3. Experimental Results

The experimental setup comprises a Kria KV260 board, a host PC, and a monitor (Figure 3). A pre-recorded drone video stream is read from an SD card, and the processing pipeline is partitioned between the PS and PL sections as described in Section 2.1. Source footage was captured at 1920×1080 and 30 FPS with a DJI Phantom 4 Pro V2.0 camera and decoded in software on the ARM cores.

The accelerator is compiled for a 128×128 RGB input and produces a $128r \times 128r$ output. The demonstration follows the same degradation protocol used for the benchmark evaluation: each source frame is bicubically downscaled to 128×128 to form the low-

resolution input, which the network then reconstructs. All figures in Table 5 refer to this geometry rather than to the capture resolution of the source footage.

Table 5. Runtime performance across all scaling factors, measured on a 128 × 128 network input. The two configurations run on different platforms, since the integer-only B4096 cannot execute the FP32 model; the table is therefore a platform comparison rather than a precision ablation. Output geometry is $128r \times 128r$ and throughput follows Equation (6).

Scale	Configuration/Platform	FPS (Mean ± Std)	Power (W)	T_{in} (Mpixel/s)	T_{out} (Mpixel/s)	Energy Eff. (Mpixel/J)
×2	FP32, RTX 4080 Laptop GPU	1367.6 ± 14.2	56.2 ± 2.1	22.41	89.63	1.60
×2	INT8, KV260 B4096	96.37	5.38 ± 0.2	1.58	6.32	1.17
×3	FP32, RTX 4080 Laptop GPU	1195.5 ± 11.8	66.9 ± 1.9	19.59	176.29	2.64
×3	INT8, KV260 B4096	80.20	5.11 ± 0.2	1.31	11.83	2.31
×4	FP32, RTX 4080 Laptop GPU	949.0 ± 9.6	69.8 ± 1.7	15.55	248.77	3.56
×4	INT8, KV260 B4096	66.10	4.93 ± 0.2	1.08	17.33	3.52

The reported frame rate is the end-to-end rate of the deployed demonstrator, read from the on-screen counter of the live display. It spans the complete pipeline: SD-card read, software H.264 decoding on the ARM cores, pre-processing in the HLS IP, DPU inference, post-processing, and on-screen composition. Four processing threads allow decoding and I/O to overlap with inference, so that the accelerator determines the sustained rate. Host-side measurements were repeated 5 times and are reported as mean ± standard deviation; the on-board rates are steady-state readings. Board-level power was measured with `xpower` at a 500 ms sampling interval across all active rails, and energy efficiency is expressed as reconstructed output per joule, T_{out}/P.

The B4096 core executes integer arithmetic only, so the two precision configurations run on different hardware: the FP32 baseline on an NVIDIA RTX 4080 Laptop GPU (175 W TDP) under PyTorch 2.13.0, with board power sampled by `nvidia-smi`, and the INT8 model on the DPUCZDX8G B4096 of the Kria KV260 through the Vitis AI runtime. Table 5, therefore, compares two deployment platforms rather than two numerical formats: arithmetic format, memory hierarchy, clock frequency, and compiler toolchain differ simultaneously, and the frame-rate and power differences cannot be attributed to quantization. The comparison shows the trade-off characteristics of the two device classes. The discrete GPU is roughly fourteen times faster in absolute frame rate, consistent with its 175 W envelope and floating-point datapath. At the same time, the KV260 sustains real-time operation within 5.38 W, an order of magnitude below the GPU's measured board power. The effect of quantization, which is isolable since both models are evaluated on identical images under identical metrics, is on reconstruction fidelity and is reported per dataset and per scale in Section 3.3.

The surrounding data path is matched across the two configurations. Both consume the same decoded frames at the same 128 × 128 input geometry, apply the same normalization to the $[0, 1]$ range before inference, and apply the same clipping and color-space conversion afterward. The pixel-shuffle operation of the reconstruction tail executes at full precision in both cases, on the ARM PS for the KV260 configuration, since the B4096 does not support it natively, and on the host for the GPU configuration.

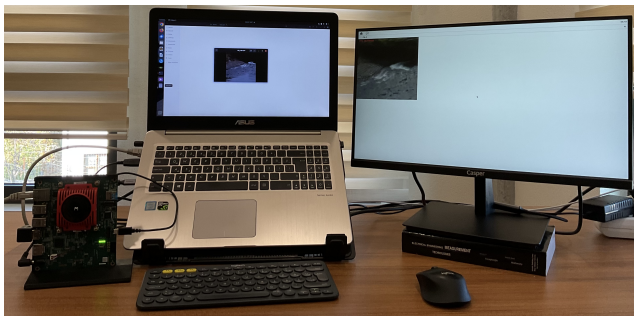


Figure 3. Experimental setup for real-time drone video enhancement at ×2 scale.

Although this study examines three distinct scaling factors, the results presented focus on the ×2 scale, where the highest performance was achieved. Runtime performance at ×3 and ×4 scales is additionally reported in Table 5. The experimental results, illustrated in Figure 4, showcase three formats of the ×2 output: the original image, the LR image, and the enhanced SR image.



(a) Pavement texture (b) Building facade (c) Facade, INT8 output

Figure 4. Real-time SR output at the ×2 scale on three test scenes. In each panel, the red box on the source frame marks the ROI enlarged alongside its ground-truth, LR, and reconstructed counterparts; panels (**a**) and (**b**) show the FP32 reconstruction, while panel (**c**) for comparison.

Throughput is defined explicitly below and computed from measured quantities, so that every entry of Table 5 can be verified directly. The definition matters because capture resolution and network input are distinct: the source video is 1920 × 1080, whereas the deployed accelerator operates on a 128 × 128 input tile and produces a $128r \times 128r$ output. All rates in this paper are reported against the network input.

$$T_{\text{in}} = \text{FPS} \times W_{\text{in}} H_{\text{in}}, \qquad T_{\text{out}} = \text{FPS} \times r^2 W_{\text{in}} H_{\text{in}}, \tag{6}$$

where $W_{\text{in}} \times H_{\text{in}} = 128 \times 128 = 0.0164$ Mpixel and $r \in \{2, 3, 4\}$ is the scaling factor. The measured rates are consistent with the computed budget of the network. Parsing the compiled `xmodel` confirms 86,627 parameters and 1.75×10^5 operations per input pixel, which at 128 × 128 amounts to 2.865, 3.313, and 3.940 GFLOP per frame at ×2, ×3, and ×4 respectively. Against the DPUCZDX8G B4096 peak of 1.126 TOPS at 275 MHz, the measured frame rates correspond to sustained DPU utilizations of 24.5%, 23.6%, and 23.1%. The consistency of these three figures, obtained from independent measurements at three scaling factors, indicates that the accelerator operates in a regime where per-layer scheduling overhead, rather than raw arithmetic throughput, is the limiting factor, as expected for a network of this depth on a small input tile.

The B4096 core executes integer arithmetic only and cannot run the FP32 model, so the FP32 baseline was obtained on a discrete GPU (NVIDIA RTX 4080 Laptop GPU, 175 W TDP) while the INT8 configuration ran on the KV260. Table 5 is therefore a comparison between two deployment platforms rather than an ablation of numerical precision: arithmetic format and hardware change together, so no quantity in the table isolates the effect of

quantization, and none is attributed to it. Read as a platform comparison, the table shows the expected trade-off. The discrete GPU is an order of magnitude faster in absolute frame rate, thanks to its 175 W power envelope and full floating-point datapath. In contrast, the KV260 delivers real-time rates of 5.38 W, an order of magnitude below the GPU's measured board power, while maintaining comparable energy efficiency per reconstructed pixel (1.17 against 1.60 Mpixel/J at $\times 2$). For completeness, the same FP32 model was also measured on the workstation CPU (Intel i9-14900HX, all cores): 80.1, 76.0, and 55.7 FPS at the three scales at approximately 65 W by RAPL, which places a general-purpose CPU below the embedded accelerator on both rate and power.

Energy efficiency of the deployed system, T_{out} divided by measured board power, is 1.17, 2.31, and 3.52 Mpixel/J at $\times 2$, $\times 3$, and $\times 4$. It rises with r because all convolutions are performed in the low-resolution feature space. At the same time, the reconstructed output area grows quadratically, so the cost per output pixel falls as the scaling factor increases.

3.1. Per-Directive Ablation of the HLS Optimization Strategies

Because pipelining, loop tiling, and data-flow execution are architecturally interdependent, their individual contributions cannot be measured by removing any one directive from the fully optimized design in isolation. A stepwise ablation is nonetheless possible by building an incremental ladder, reported in Table 6. Each configuration adds exactly one directive class to the previous one; the variants were produced from the deployed source by pragma removal only, so the C++ they synthesize is otherwise byte-identical, and all five were synthesized and implemented in the same tool version and reported after `route_design`, keeping the comparison internally controlled.

Table 6. Per-directive ablation of the HLS SR pre-processing IP. Post-route values on xck26-sfvc784-2LV-c at a 275 MHz target (3.636 ns), Vitis HLS/Vivado 2022.2; each row adds one directive class to the row above. These runs use a later tool version than the post-route resource results reported later in this section, so absolute values differ slightly, but the ladder is internally consistent. The IP performs a scale-independent pixel-wise normalization, so one ablation covers all three scaling factors.

	Directives	LUT	LUTRAM	FF	DSP	BRAM	URAM	Post-Route	Timing
V0	Base (no pragmas)	1688	36	2226	4	0	0	3.058 ns	Met
V1	+ `PIPELINE II=1`	6583	825	8762	0	3	0	3.463 ns	Met
V2	+ `DATAFLOW`	7166	1005	9443	2	6	0	3.235 ns	Met
V3	+ `ARRAY_PARTITION` + LUTRAM	8448	1641	10,351	2	0	0	3.561 ns	Met
V4	+ `UNROLL factor=4`	8448	1641	10,351	2	0	0	3.561 ns	Met

Four observations follow from this ablation. First, pipelining dominates the cost. Adding `PIPELINE II=1` to the row loops raises LUT usage from 1688 to 6583, an increase of 290%, and flip-flops from 2226 to 8762. This is the price of the complete unroll of the inner pixel loop implied by the pipeline directive, which replicates the unpacking and normalization datapath sixteen times. It also eliminates the four DSPs of the base design, since the replicated multipliers are mapped to logic rather than to DSP slices.

Second, `DATAFLOW` is comparatively cheap: 583 additional LUTs, an increase of 8.9%, for the channel FIFOs and handshaking that let the read stage of one tile overlap the write stage of the previous one. This directive delivers the task-level overlap on which the achieved interval depends, and it is the least expensive of the three.

Third, the array partitioning and LUTRAM binding cost 1282 LUTs, an increase of 17.9%, and they do what they were introduced to do. Without them, in V1 and V2, the tile buffers are inferred as block RAM, and the design consumes 3 and 6 BRAM tiles,

respectively. With them, in V3, BRAM usage returns to zero, and LUTRAM rises to 1641. Since the DPUCZDX8G core requires 79 of the 144 BRAM tiles available on the KV260, this migration is not a micro-optimization but a condition of the two blocks coexisting, and the ablation shows it directly rather than by assertion.

Fourth, UNROLL factor = 4 has no measurable effect. V3 and V4 differ only by that directive, and their post-route results are identical in every column: 8448 LUTs, 1641 LUTRAM, 10,351 flip-flops, 2 DSPs, zero BRAM, zero URAM, and the same 3.561 ns post-route period. This confirms what the Vitis HLS log itself reports: the enclosing PIPELINE II = 1 forces a complete unroll of factor 16 and supersedes the requested factor of 4. The directive is therefore redundant rather than unsynthesizable, which is consistent with the synthesized implementation, which completes successfully and meets timing.

Power was also estimated for each configuration. Vivado's vectorless estimator was run on the routed checkpoint for every variant under identical switching assumptions (a default toggle rate of 12.5% and a static probability of 0.5), so that variants can be compared even though the absolute values are estimates rather than measurements. The base IP draws an estimated 36 mW of dynamic power; the pipelined configuration draws 179 mW, the addition of `DATAFLOW` brings this to 171 mW, and the partitioned LUTRAM configuration to 175 mW. Device static power is 288–289 mW throughout and is a property of the part rather than of the design. Two observations follow. The dynamic power of the pre-processing accelerator, at roughly 175 mW, accounts for approximately 3% of the board-level figure reported in Section 3, so the accelerator is not a significant contributor to system power, and the design freedom it offers is better spent on latency than on energy. V3 and V4 again return identical estimates in milliwatts, consistent with their identical placed-and-routed netlists and confirming once more that the `UNROLL` directive has no effect.

These are estimates derived from switching assumptions rather than measurements from an instrumented board, and are reported as such. They are included because they are reproducible from the supplied checkpoints and because the comparison between variants, which is what the ablation requires, is unaffected by the estimator's absolute calibration.

One limitation of this ablation should be stated. It isolates the contributions of the HLS directives to the pre-processing IP, which is the part of the system these directives govern. It does not isolate the contribution of INT8 quantization to frame rate or power, because the DPUCZDX8G B4096 has no floating-point datapath and cannot execute an FP32 model at all; a same-device precision ablation is not constructible on this platform, as discussed in Section 3. The effect of quantization that is separable, namely its effect on reconstruction fidelity, is reported in Section 3.3. Table 7 lists the logic capacity and approximate cost of the devices used in the comparison of Section 3, for reference.

Table 7. Board resources and estimated prices.

Board (Xilinx)	Series	LUTs	FFs	DSP	Est. Price (USD)
ZCU111	Zynq UltraScale+	930,240	1,860,480	4272	1500–2000
XCKU040	Kintex UltraScale	1,176,000	2,352,000	1920	2000–3000
XC7K410T	Kintex-7	508,400	1,016,800	1540	1000–1500
XC7VX485T	Virtex-7	303,600	607,200	2800	2500–3500
XCKU15P	Kintex UltraScale+	663,360	1,326,720	3432	2500–3500
Kria KV260	Zynq UltraScale+	117,120	234,240	1248	200–300

3.2. Performance Comparison and Inference Efficiency

Table 8 reports post-route resource usage and timing for the HLS SR pre-processing IP, obtained with the `#pragma HLS UNROLL factor = 4` directive applied against a single-

word-per-cycle AXI-Stream interface. The directive is present on the innermost pixel loop of both the read and the write stage, and `#pragma HLS PIPELINE II = 1` is applied to the enclosing row loop. The design synthesizes, places, and routes without error: zero unrouted nets, worst hold slack $+0.055$ ns, and worst setup slack $+0.225$ ns against the 3.636 ns target. The configuration is realizable, and the resource and timing figures in Table 8 are genuine post-route values from that implementation, with the LUTRAM column completed from the same reports (36 for the base IP, 1641 for the optimized IP, consistent with the `BIND_STORAGE/impl=LUTRAM` directive on the tile buffers).

The measured schedule confirms this directly: the dataflow region completes one 16×16 tile every 790 cycles. A tile contains 256 pixels, and because the output stream carries one 32FP word per color channel, writing a tile requires $256 \times 3 = 768$ transfers. The interval is thus set almost exactly by the output transfer count. The optimization that the directive set delivers is task-level overlap between the read and write stages via `DATAFLOW`, together with the LUTRAM-resident partitioned tile buffers that supply the concurrent ports those stages need; the $5.1\times$ LUT and $4.4\times$ FF increase over the base IP is the cost of that overlap and of the fully unrolled inner loop, not of any additional stream bandwidth. DSP usage falls from 4 to 2 because the dataflow decomposition allows the floating-point divider to be shared differently between the stages.

At the 275 MHz implementation clock, 790 cycles per tile places the pre-processing stage well clear of the system bottleneck at the deployed geometry: a 128×128 input comprises 64 tiles, or 50,560 cycles, corresponding to 0.18 ms per frame and a pre-processing rate above 5000 FPS in isolation. The DPU inference stage, which sustains 96.37 FPS at $\times 2$, is therefore the limiting element of the pipeline by a wide margin, and the pre-processing IP is sized so as not to constrain it. At Full-HD geometry, the same IP would require 23.44 ms per frame, or 42.7 FPS, which is why region-of-interest rather than full-frame operation is the intended deployment mode.

The lower half of the table reports the full overlay. Adding the HLS IP raises LUT usage from 62,145 to 74,034 and FF usage from 111,951 to 121,493. At the same time, DSP rises by exactly the 2 consumed by the IP and BRAM, and URAM remains unchanged at 79 and 48, confirming that the tile buffers are indeed mapped to distributed LUTRAM and do not compete for the block-RAM resources of the DPUCZDX8G core. The complete design occupies 63.2% of LUTs, 51.9% of flip-flops, and 54.9% of block RAM on the KV260, and meets timing closure at the 275 MHz target, confirming that the optimization strategies are realizable within the platform resource budget.

Table 9 compares the proposed system against state-of-the-art FPGA-based SR implementations. On throughput, defined as in Equation (6) and computed against the 128×128 network input, the proposed system delivers 1.58 Mpixel/s of input and 6.32 Mpixel/s of reconstructed output at the $\times 2$ scale, against 149–753 Mpixel/s for the compared accelerators. The difference is one of frame geometry rather than of speed. Dividing each reported throughput by its reported frame rate recovers the area processed per frame: 2.1–2.5 Mpixel for the designs of [7,18] and 8.3–8.4 Mpixel for those of [8–10], corresponding to full-resolution 1920×1080 and 3840×2160 frames respectively, against 0.016 Mpixel for the 128×128 input used here. Since throughput is the product of frame rate and frame area, an input two to three orders of magnitude smaller yields a proportionally smaller figure even at a higher frame rate. The compared works implement small, hardwired networks (FSRCNN at roughly 12 K parameters, VDSR at 665 K) as fixed-function datapaths that stream a full frame at one or more pixels per clock; the present work executes an 86.6 K parameter, 23-layer residual network on a compiler-scheduled DPU at a smaller input geometry. Megapixels per second, therefore, measures the property on which the fixed-function designs are strongest, and the two design points are not interchangeable. On capacity, two rows give parameter count and operation

count. At 86,627 parameters, the deployed network is roughly 180× smaller than the RCAN of [3], whose PSNR it is compared against, and the reader should attribute the frame rate primarily to that reduction rather than to the quantization and HLS strategies alone.

Table 8. Post-route resource usage and timing for the HLS SR pre-processing IP. Top: standalone IP (base vs. optimized); bottom: full overlay (DPUCZDX8G B4096 + HLS IP + AXI interconnect). All values are read from the Vivado implementation utilization report on the Kria KV260 (xck26-sfvc784-2LV-c) at a 275 MHz target, Vitis HLS/Vivado 2022.1.

Configuration	LUT	LUTRAM	FF	DSP	BRAM	URAM	Timing
Standalone HLS SR pre-processing IP							
HLS IP, base (no pragmas)	1663	36	2358	4	0	0	3.104 ns Met
HLS IP, optimized	8496	1641	10,365	2	0	0	3.411 ns Met (WNS +0.225 ns)
Full overlay (post-implementation)							
Full overlay, DPU only	62,145	8275	111,951	718	79	48	3.104 ns Met
Full overlay, DPU + HLS IP	74,034	9964	121,493	720	79	48	3.411 ns Met
Available (KV260)	117,120	57,600	234,240	1248	144	64	—
Utilization % (DPU + HLS IP)	63.2	17.3	51.9	57.7	54.9	75.0	—

The comparison this table does support is narrower and platform-specific. Among the compared systems, the proposed one attains the highest frame rate (96.37 FPS) at the highest clock frequency (275 MHz), on much the cheapest platform ($200–300 against $1500 and above for most of the others), at 5.38 W of measured board power, and with the smallest reported memory footprint after the fixed-function designs. It does so for a network two orders of magnitude more complex than those comparators, at a correspondingly smaller frame geometry. Which trade-off is preferable depends on the application: for a fixed region of interest under a strict power budget, which is the UAV scenario targeted here, the present design is appropriate; for full-resolution upscaling of a Full HD stream, the fixed-function accelerators remain the better choice.

In terms of raw parameter storage, INT8 quantization delivers the expected benefit: the deployed 86,627-parameter network requires 86.6 KB at INT8 precision versus 346.5 KB at FP32, a 75% reduction. This saving is not visible, however, at the level of the on-disk deployment artifacts reported in Table 9 (FP32 checkpoint: 363.9 KB; compiled INT8 `.xmodel`: 474.5 KB), because the compiled DPU graph additionally embeds the instruction schedule and per-tensor quantization metadata that a bare FP32 checkpoint does not carry. The two figures therefore reflect different artifact types rather than a genuine memory regression from quantization; the underlying 4× weight compression is real and is what allows the DPU to execute INT8 arithmetic more cheaply, which is what drives the throughput and power gains reported above.

3.3. Fidelity Cost of Quantization

Reconstruction accuracy was measured from first principles; both the resulting values and the procedure used to obtain them are set out below so that the figures can be independently reproduced. The evaluation follows the protocol distributed with the reference implementation of the deployed model. Differences between the reconstruction and the ground truth are projected onto the luminance channel using the coefficients

[65.738, 129.057, 25.064]/256, a border of r pixels is discarded, and the peak signal-to-noise ratio is computed as $-10\log_{10}(\mathrm{MSE})$; the structural similarity index follows the implementation used in the EDSR benchmark suite. Low-resolution inputs were regenerated rather than taken as distributed, because the publicly available low-resolution sets are misaligned with their high-resolution counterparts whenever the latter have dimensions that are not divisible by the scaling factor, a condition that holds for all one hundred images of B100. Each high-resolution image was therefore cropped to a multiple of r and decimated bicubically. That this reproduces the standard protocol may be verified from the bicubic upper bound, which our pipeline places at 33.67, 30.32, 29.56 and 26.88 dB on Set5, Set14, B100 and Urban100 respectively, against the values of approximately 33.7, 30.2, 29.6 and 26.9 dB reported in the literature.

Table 9. Performance comparison of SR methods across Xilinx hardware platforms.

Specifications		**[7]**	**[6]**	**[8]**	**[9]**	**[18]**	**[10]**	**This Study**
Method		USR-s	CNN-Based	FSRCNN	Fovea Selective	LSR	RNN-based	RCAN-SR
Scale Ratio		×2,3,4	×2	×2,3,4	×2	×2,3,4	×2	×2,3,4
Platform		Xilinx ZCU111	Xilinx XCKU040 Kintex UltraScale	Xilinx XC7K410T Kintex	Xilinx XC7K410T Kintex-7	Xilinx XC7VX485T Virtex-7	Xilinx XCKU15P Kintex UltraScale+	XCK26 UltraScale+
Frequency (MHz)		188	150	**130**	222	200	160	275
Precision Weights—Act.		12–12	10–14	13–13	16–16	16–16	16–12	32–32 **8–8**
Power (W)		N	5.69	5.40	3.7	2.08	5.47	5.38
Resource Consumption		**4.30**% LUT **2.69**% FF 24.08% DSP	12.84% LUT 5.14% FF **10**% DSP	32.84% LUT 15.54% FF 98.18% DSP	5.51% LUT 8.06% FF 100% DSP	24.04% LUT 8.73% FF 25.14% DSP	24.04% LUT 8.73% FF 25.14% DSP	63.2% LUT 51.9% FF 57.7% DSP
Model size (params)		N	665 K (VDSR)	12 K (FSRCNN)	N	N	N	**86.6 K**
Model size (FLOPs/frame)		N	N	N	N	N	N	**2.87 G**
Memory Footprint (KB)		1089	**194**	945	542	496	4842	FP32: **363.9** INT8: 474.5
Input geometry		N	N	N	N	N	N	128 × 128
Throughput (Mpixel/s, input)		149	600	520	753	168	638	1.58
Throughput (Mpixel/s, output)		N	N	N	N	N	N	6.32
FPS		60	60	62.7	90.8	81	76	**96.37**
PSNR (×2)	Set5	N	37.53	37.05	33.23	37.44	36.51	**FP32: 37.6584** INT8: 37.3844
	Set14	N	33.03	32.66	29.73	32.74	32.46	**FP32: 33.1837** INT8: 33.0118
	B100	N	31.90	31.53	29.32	N	31.32	**FP32: 31.9012** INT8: 31.7852
	Urban100	N	30.76	29.88	30.06	N	29.32	**FP32: 30.9856** INT8: 30.8393
	VisDrone21	N	N	N	N	N	N	**FP32: 25.9374** INT8: 24.9959
SSIM (×2)	Set5	N	0.9487	0.9560	0.9446	N	0.9556	**FP32: 0.9592** INT8: 0.9560
	Set14	N	0.9124	0.9090	0.9043	N	0.9079	**FP32: 0.9140** INT8: 0.9105
	B100	N	0.8960	0.8920	0.8563	N	0.8891	**FP32: 0.8966** INT8: 0.8935
	Urban100	N	0.9140	0.9020	0.8628	N	0.8957	**FP32: 0.9166** INT8: 0.9126
	VisDrone21	N	N	N	N	N	N	**FP32: 0.7481** INT8: 0.7205

Bold denotes best-in-class; N = not reported. FPS, power, and throughput values for this study represent the mean of five independent runs; standard deviations are reported in Table 5.

Table 1 reports these measurements. In floating point the network attains 37.6584, 33.1837, 31.9012, and 30.9856 dB on the four benchmarks, agreeing to within 0.015 dB with the figures published for this model, which confirms that the evaluation pipeline is

faithful. The resulting degradation attributable to quantization is small and unremarkable: 0.274 dB on Set5, 0.172 dB on Set14, 0.116 dB on B100, and 0.146 dB on Urban100. This is well within the range typically reported for INT8 QAT and consistent with the fact that the quantized model deploys successfully in practice. A sensitivity analysis provides a partial decomposition of the observed quantization loss. Quantizing the weights alone, using per-channel symmetric 8-bit scales, incurs a PSNR reduction of 0.051 dB on Set5 and less than 0.021 dB on the remaining three benchmarks. By contrast, the full quantization-aware training (QAT) procedure, which additionally quantizes activations on a per-tensor basis, results in a 0.274 dB reduction on Set5. Approximately four-fifths of the total degradation is therefore attributable to activation quantization rather than weight quantization. This behavior is consistent with the structure of the deployed network: the widest intermediate tensor contains 64 channels whereas the narrowest contains only 16, requiring a single scale factor per activation tensor to accommodate a substantially wider dynamic range than that faced by the per-channel weight quantizer. These findings suggest that any further reduction in quantization error is more likely to come from per-channel or mixed-precision treatment of activations than from finer weight quantization.

Two points follow from these measurements. First, the deployed network sits between 0.6 and 2.4 dB below the reference RCAN across the four benchmarks, which is the cost of a 180-fold reduction in parameter count. Second, VisDrone21 was measured under the same protocol: 25.9374 dB/0.7481 SSIM in FP32 and 24.9959 dB/0.7205 SSIM for the deployed INT8 model on the 21 held-out images. The lower absolute PSNR relative to the classical benchmarks is expected, since VisDrone frames carry fine urban structure and JPEG compression at native resolutions the network was not tuned for. The INT8 figure was obtained by loading the released quantized checkpoint into the Vitis AI QatProcessor and converting it to its deployable form.

3.4. Limitations

The proposed system is optimized for the Kria KV260 platform, and its portability to other FPGA devices or embedded architectures has not been evaluated. Although the implementation achieves real-time performance (Table 5), execution speed may vary with input resolution, scene complexity, and the memory bandwidth characteristics of the target platform.

At the $\times 2$ scale, INT8 quantization introduces only a limited reconstruction loss (Table 10), with most of the degradation originating from activation quantization rather than weight quantization (Section 3.3). The $\times 3$ and $\times 4$ models reported in Table 1 are FP32 fine-tuned variants of the released $\times 2$ backbone and were not optimized through QAT. Extending QAT to these scaling factors remains future work.

Table 10. Reconstruction accuracy at the $\times 2$ scale on the four standard benchmarks, under the protocol described in the text.

Configuration	Set5	Set14	B100	Urban100
Bicubic (reference bound)	33.67	30.32	29.56	26.88
FP32 (measured)	37.6584	33.1837	31.9012	30.9856
INT8, weights only (measured)	37.6073	33.1657	31.8886	30.9657
INT8-QAT, deployed	37.3844	33.0118	31.7852	30.8393
Δ from weight quantization	0.051	0.018	0.013	0.020
Δ from full quantization	0.274	0.172	0.116	0.146
Share attributable to activations	81%	90%	89%	86%

The UAV evaluation is based on bicubically generated LR–HR image pairs from the VisDrone2019 dataset (Section 2.3). While this protocol enables reproducible comparisons with standard SR benchmarks (Table 1), it does not fully represent the optical blur, sensor noise, motion artifacts, and video compression encountered in real UAV imaging systems. A paired UAV dataset containing real optical degradations would enable a more comprehensive evaluation.

Finally, the cross-platform comparison presented in Table 9 should be interpreted with care, since the reported systems differ in FPGA platform, numerical precision, model capacity, evaluation datasets, throughput definitions, and measurement methodology. The isolated contribution of the proposed hardware optimizations is therefore better reflected by the ablation study (Table 6) and the post-route resource analysis (Table 8), whereas Table 9 provides an indicative comparison across heterogeneous FPGA-based implementations.

4. Conclusions

This study presents a Hw-Sw co-design framework for real-time SR of low-resolution drone video streams on the Xilinx Kria KV260 platform. The proposed system balances reconstruction quality against hardware cost for power-constrained embedded deployment by integrating an OFA-searched compact residual network, QAT, a customized DPU configuration, and HLS-based pipeline, loop-tiling, and data-flow optimizations.

With a 128×128 network input, the deployed INT8 model sustains 96.37 FPS at the $\times 2$ scale while consuming 5.38 W of measured board power, corresponding to a reconstructed-output throughput of 6.32 Mpixel/s and an energy efficiency of 1.17 Mpixel/J. The complete design occupies 63.2% of the available LUT resources on the KV260 and satisfies timing closure at 275 MHz. Measured DPU utilizations of 24.5%, 23.6%, and 23.1% at the $\times 2$, $\times 3$, and $\times 4$ scales indicate that the deployed system is primarily limited by scheduling overhead rather than computational throughput.

At the $\times 2$ scale, INT8 quantization incurs only a small reconstruction loss, corresponding to 0.274 dB on Set5 and 0.146 dB on Urban100, remaining within the range typically reported for INT8 QAT. The $\times 3$ and $\times 4$ models are FP32 fine-tuned variants derived from the released $\times 2$ backbone and were not optimized through QAT; consequently, no INT8-QAT fidelity results are reported for these scales. The scope and boundaries of these results are set out in Section 3.4. The FP32 baseline is included solely as a cross-platform reference rather than a same-device precision comparison, and no throughput or power improvement is attributed to quantization.

Overall, the results demonstrate that a twenty-three-layer residual SR network can be deployed in real time on a low-cost ($200–300), integer-only edge FPGA within a 5.38 W power envelope while maintaining a reconstruction loss below 0.27 dB. These findings highlight the potential of Hw-Sw co-design for enabling practical, energy-efficient SR deployment in UAVs and other resource-constrained edge vision applications.

Future Work

The most immediate priority is to QAT-train the $\times 3$ and $\times 4$ reconstruction tails and measure their deployed INT8 fidelity under the protocol used here, since only their FP32 fine-tunes have been evaluated so far. The second is a UAV dataset with genuine rather than synthetic degradation, paired by simultaneous dual-focal-length capture or by co-registered acquisition at two altitudes, which would allow robustness to real blur, noise, compression, and motion artefacts to be measured rather than inferred. Future research can also build upon this work by exploring adaptive resource management techniques that respond to varying computational demands during drone operations. This includes advanced model compression approaches such as structured sparsity, adaptive quantization, and binary

neural networks (BNNs). Another potential direction is to integrate multiple real-time tasks into a unified multi-task framework. On the hardware side, next-generation FPGA platforms and heterogeneous FPGA–GPU architectures present a promising opportunity.

Author Contributions: Conceptualization, G.T.; methodology, G.T.; software, G.T. and M.E.A.; validation, G.T.; formal analysis, G.T.; investigation, G.T.; resources, G.T.; data curation, G.T. and M.E.A.; writing original draft preparation, G.T. and M.E.A.; writing review and editing, G.T. and M.E.A.; visualization, G.T.; supervision, G.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The DIV2K dataset [24] is publicly available at https://data.vision.ee.ethz.ch/cvl/DIV2K/ (accessed on 29 July 2026). The VisDrone2019 dataset [2] is publicly available at https://github.com/VisDrone/VisDrone-Dataset (accessed on 29 July 2026) under a CC BY-NC-SA 3.0 licence; the 121-image subset used in this work, together with its train/test split and the script that regenerates the low-resolution pairs, will be released upon acceptance. The Set5, Set14, B100 and Urban100 benchmarks are the standard sets distributed with the EDSR benchmark package.

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

References

1. Mairaj, A.; Baba, A.I.; Javaid, A.Y. Application specific drone simulators: Recent advances and challenges. *Simul. Model. Pract. Theory* **2019**, *94*, 100–117. [CrossRef]
2. Zhu, P.; Wen, L.; Du, D.; Bian, X.; Fan, H.; Hu, Q.; Ling, H. Detection and tracking meet drones challenge. *IEEE Trans. Pattern Anal. Mach. Intell.* **2022**, *44*, 7380–7399. [CrossRef] [PubMed]
3. Zhang, Y.; Li, K.; Li, K.; Wang, L.; Zhong, B.; Fu, Y. Image super-resolution using very deep residual channel attention networks. In Proceedings of the European Conference on Computer Vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 286–301.
4. Li, X.; Li, Z.; Zhou, L.; Huang, Z. FOLD: Low-level image enhancement for low-light object detection based on FPGA MPSoC. *Electronics* **2024**, *13*, 230. [CrossRef]
5. Luo, W.; Hung, P.; Wang, S.; Cheung, R.C.C. Image super-resolution and FPGA hardware design. In Proceedings of the 2023 IEEE International Conference on Signal Processing, Communications and Computing (ICSPCC), Zhengzhou, China, 14–17 November 2023; pp. 1–5.
6. Kim, J.; Kwon Lee, J.; Mu Lee, K. Accurate image super-resolution using very deep convolutional networks. In Proceedings of the European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 8–16 October 2016; pp. 234–250.
7. Duan, X.; Chen, Y.; Li, M.; Rong, Y.; Xie, R.; Han, J. UArch: A super-resolution processor with heterogeneous triple-core architecture for workloads of U-Net networks. *IEEE Trans. Biomed. Circuits Syst.* **2023**, *17*, 633–647. [CrossRef] [PubMed]
8. Chang, J.-W.; Kang, K.-W.; Kang, S.-J. An energy-efficient FPGA-based deconvolutional neural networks accelerator for single image super-resolution. *IEEE Trans. Circuits Syst. Video Technol.* **2020**, *30*, 281–295. [CrossRef]
9. Spagnolo, F.; Corsonello, P.; Frustaci, F.; Perri, S. Design of a low-power super-resolution architecture for virtual reality wearable devices. *IEEE Sens. J.* **2023**, *23*, 9009–9016. [CrossRef]
10. Sun, K.; Koch, M.; Wang, Z.; Jovanovic, S.; Rabah, H.; Simon, S. An FPGA-based residual recurrent neural network for real-time video super-resolution. *IEEE Trans. Circuits Syst. Video Technol.* **2022**, *32*, 1739–1750. [CrossRef]
11. Tatar, G.; Bayar, S.; Çiçek, İ.; Niar, S. Recent advances in machine learning-based advanced driver assistance system applications. *Microprocess. Microsyst.* **2024**, *110*, 105101. [CrossRef]
12. AMD Xilinx. Vitis AI Development Flow v3.5. Available online: https://xilinx.github.io/Vitis-AI/3.0/html/docs/workflow-system-integration.html (accessed on 1 September 2024).
13. Lim, B.; Son, S.; Kim, H.; Nah, S.; Lee, K.M. Enhanced deep residual networks for single image super-resolution. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Honolulu, HI, USA, 21–26 July 2017; pp. 1132–1140.
14. Chen, Z.; Wu, Z.; Zamfir, E.; Zhang, K.; Zhang, Y.; Timofte, R.; Martinel, N. NTIRE 2024 challenge on image super-resolution (×4): Methods and results. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Seattle, WA, USA, 17–21 June 2024; pp. 6108–6132.

15. AMD Xilinx. Vitis AI Model Zoo—OFA-RCAN Model v3.5. Available online: https://github.com/Xilinx/Vitis-AI/tree/master/model_zoo/model-list/pt_OFA-rcan_3.5 (accessed on 1 September 2024).
16. Vedavyas, Y.; Vasavi, S.; Harsha, S.S.; Subhash, M.S. An FPGA-based adaptive real-time quality enhancement system for drone imagery. *SN Comput. Sci.* **2023**, *4*, 84. [CrossRef]
17. Cheng, X.; Li, X.; Tai, Y.; Yang, J. SESR: Single image super resolution with recursive squeeze and excitation networks. In Proceedings of the 24th International Conference on Pattern Recognition (ICPR), Beijing, China, 20–24 August 2018; pp. 147–152.
18. Zhao, X.; Chang, L.; Fan, D.; Hu, Z.; Yue, T.; Tu, F.; Zhou, J. HDSuper: High-quality and high computational utilization edge super-resolution accelerator with hardware-algorithm co-design techniques. *IEEE Trans. Circuits Syst. I Regul. Pap.* **2024**, *71*, 1679–1692. [CrossRef]
19. Dong, C.; Loy, C.C.; Tang, X. Accelerating the super-resolution convolutional neural network. In Proceedings of the European Conference on Computer Vision (ECCV), Amsterdam, The Netherlands, 8–16 October 2016.
20. Dong, C.; Loy, C.C.; He, K.; Tang, X. Image super-resolution using deep convolutional networks. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *38*, 295–307. [CrossRef] [PubMed]
21. Lai, W.S.; Huang, J.B.; Ahuja, N.; Yang, M.H. Deep Laplacian pyramid networks for fast and accurate super-resolution. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 21–26 July 2017.
22. Zhang, Y.; Tian, Y.; Kong, Y.; Zhong, B.; Fu, Y. Residual dense network for image super-resolution. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 1–9.
23. Xilinx. Vitis Platform Creation Tutorial. Available online: https://github.com/Xilinx/Vitis-Tutorials/blob/2024.1/Vitis_Platform_Creation/Vitis_Platform_Creation.rst (accessed on 1 September 2024).
24. Agustsson, E.; Timofte, R. NTIRE 2017 challenge on single image super-resolution: Dataset and study. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, Honolulu, HI, USA, 21–26 July 2017; pp. 1122–1131.
25. Mao, L. Quantization for Neural Networks. Available online: https://leimao.github.io/article/Neural-Networks-Quantization/ (accessed on 15 November 2024).
26. Novac, P.-E.; Boukli Hacene, G.; Pegatoquet, A.; Miramond, B.; Gripon, V. Quantization and deployment of deep neural networks on microcontrollers. *Sensors* **2021**, *21*, 2984. [CrossRef] [PubMed]
27. Jacob, B.; Kligys, S.; Chen, B.; Zhu, M.; Tang, M.; Howard, A.; Adam, H.; Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 18–22 June 2018; pp. 2704–2713.
28. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Quantized neural networks: Training neural networks with low precision weights and activations. *J. Mach. Learn. Res.* **2018**, *18*, 1–30.
29. Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv* **2018**, arXiv:1806.08342.
30. Bengio, Y.; Léonard, N.; Courville, A.C. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv* **2013**, arXiv:1308.3432.
31. Esser, S.K.; McKinstry, J.L.; Bablani, D.; Appuswamy, R.; Modha, D.S. Learned step size quantization. In Proceedings of the International Conference on Learning Representations (ICLR), Addis Ababa, Ethiopia, 26–30 April 2020.

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.