Summary of Research Paper

Underwater Image Enhancement Using FPGA-Based Gaussian Filters with Approximation Techniques
Summary Date: August 1, 2025

1. **Introduction / Abstract**
   The paper addresses the problem of poor visual quality in underwater images caused by natural effects like light absorption and scattering, which create haziness and noise. To improve these images, the authors propose an efficient method using **Gaussian filters** (a mathematical tool for smoothing images and reducing noise) implemented on **FPGA** (Field-Programmable Gate Array, a type of reconfigurable hardware that allows fast and parallel processing). The key innovation is a pipeline architecture combined with approximate adders (simplified arithmetic units that trade some accuracy for better speed and lower power use). The main findings show over 150% speed improvement and more than 34% power reduction, with some increase in hardware area. This trade-off suits error-tolerant applications like image and video processing.

2. **Methodology**
   The researchers designed a pipeline Gaussian filter architecture on FPGA, which processes image pixels in stages to increase throughput. They used line buffers to store image rows and window buffers to hold local pixel areas for filtering, minimizing memory access. The Gaussian filter applies a 3×3 kernel (a small matrix) to smooth images by weighted averaging of pixels. To reduce hardware complexity and power, they incorporated various approximate full adders that simplify addition operations by allowing controlled errors in less significant bits. They tested ten different approximate adder designs, evaluating their impact on image quality and hardware performance. Simulations were conducted in MATLAB with real underwater images corrupted by Gaussian noise, and hardware synthesis was performed on an Intel MAX10 FPGA.

3. **Theory / Mathematics**
   The Gaussian filter is based on the two-dimensional Gaussian function:

$$f(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

where (f(x,y)) is the filter value at coordinates ((x,y)), and (\sigma) controls the spread of the smoothing effect. This function weights pixels near the center more heavily, effectively blurring noise while preserving image structure.

The convolution operation for filtering is expressed as:

$$h[i,j] = A P_1 + B P_2 + C P_3 + D P_4 + E P_5 + F P_6 + G P_7 + H P_8 + I P_9$$

where (P_1) to (P_9) are pixel values in the 3×3 window, and (A) to (I) are corresponding weights from the Gaussian kernel.

Image quality metrics used include:
- **Peak Signal-to-Noise Ratio (PSNR):**

$$PSNR(f,g) = 20 \log_{10} \left(\frac{2^B - 1}{\sqrt{MSE(f,g)}}\right)$$

where (B) is bit depth, and (MSE) is mean squared error between original (f) and filtered (g) images.

- **Mean Squared Error (MSE):**

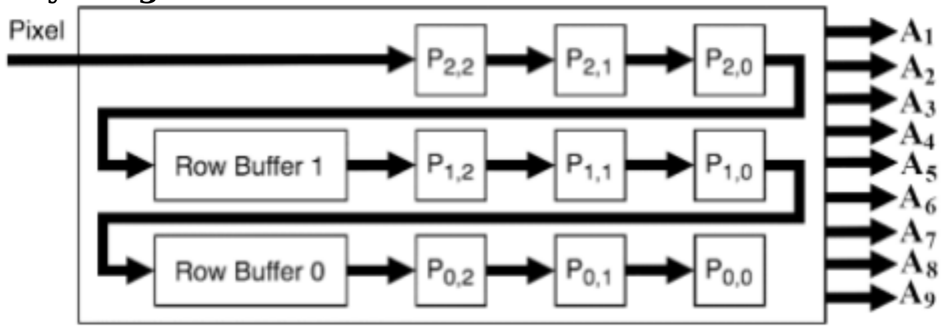
$$MSE(f,g) = \frac{1}{MN} \sum_{i=1}^M \sum_{j=1}^N (f_{ij} - g_{ij})^2$$

- **Structural Similarity Index (SSIM):** Measures perceptual similarity considering luminance, contrast, and structure.

- **Error Distance (ED):** Euclidean distance between corresponding pixels in original and processed images, quantifying spatial accuracy.

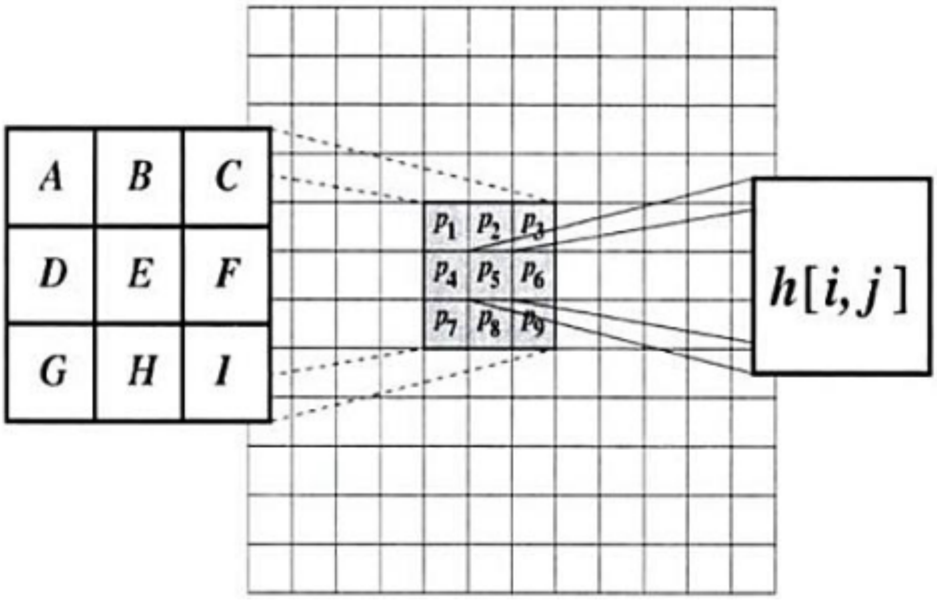
These formulas quantify how well the filter restores image quality while balancing computational efficiency.

1. **Key Diagrams or Visual Elements**

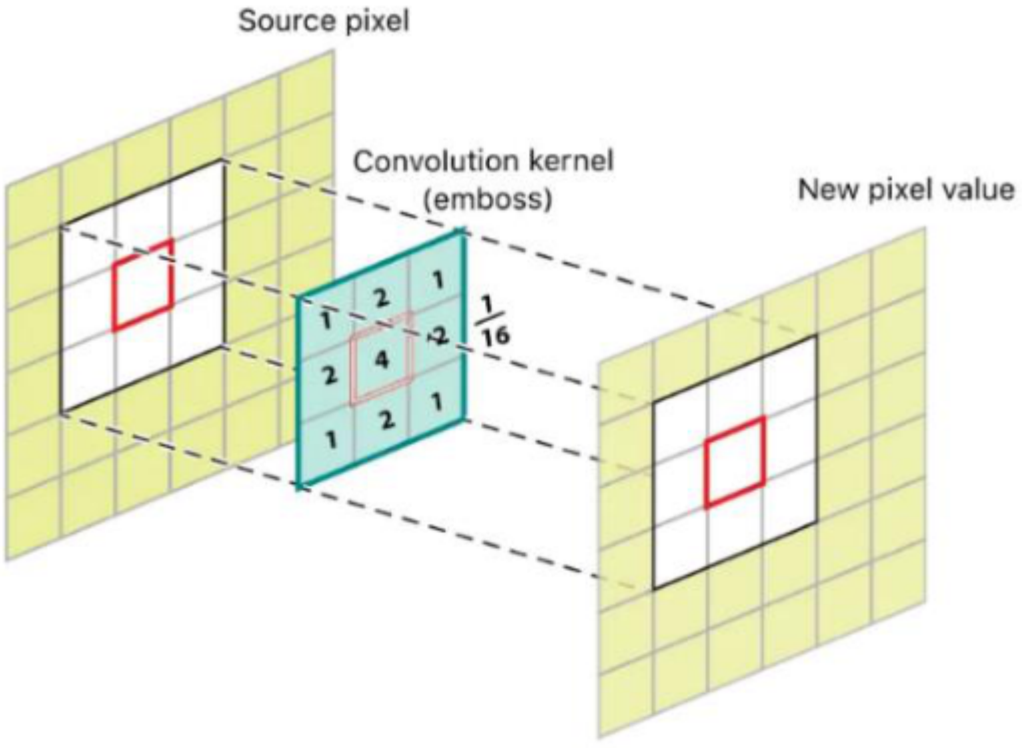


2. **Figure 1:** Shows the delay line buffer structure, illustrating how pixel rows are stored and accessed efficiently during filtering to reduce

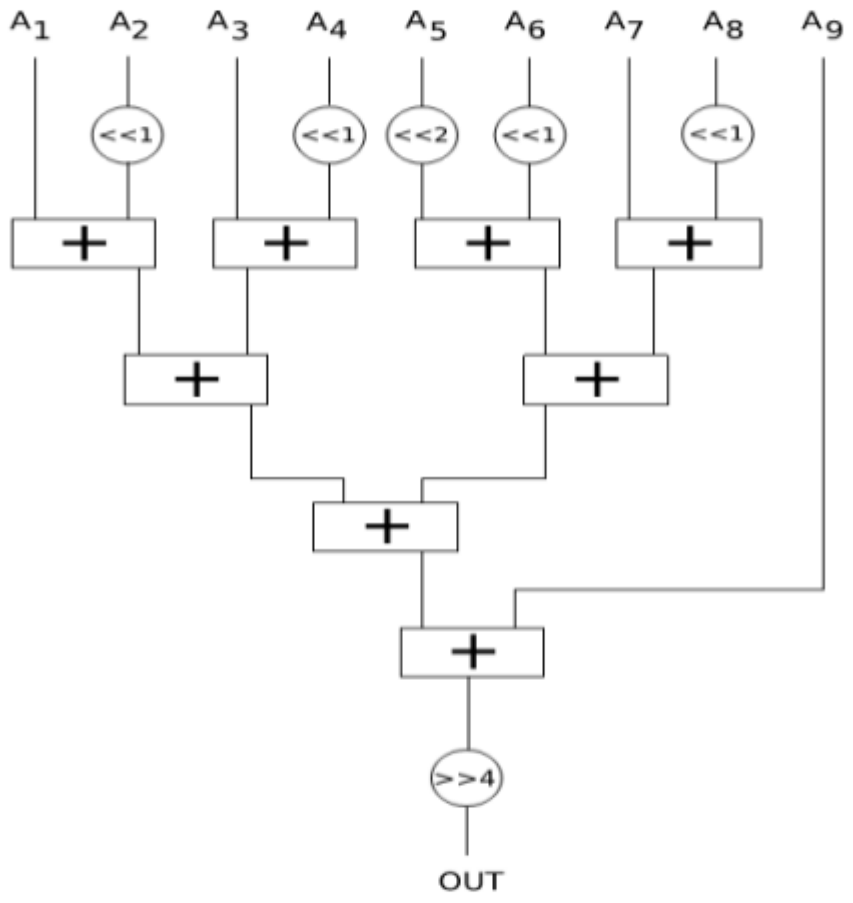
memory bottlenecks.



3. **Figure 2:** Depicts the convolution operation with a 3×3 kernel sliding over the image, explaining how each output pixel is computed from weighted sums of neighboring pixels.

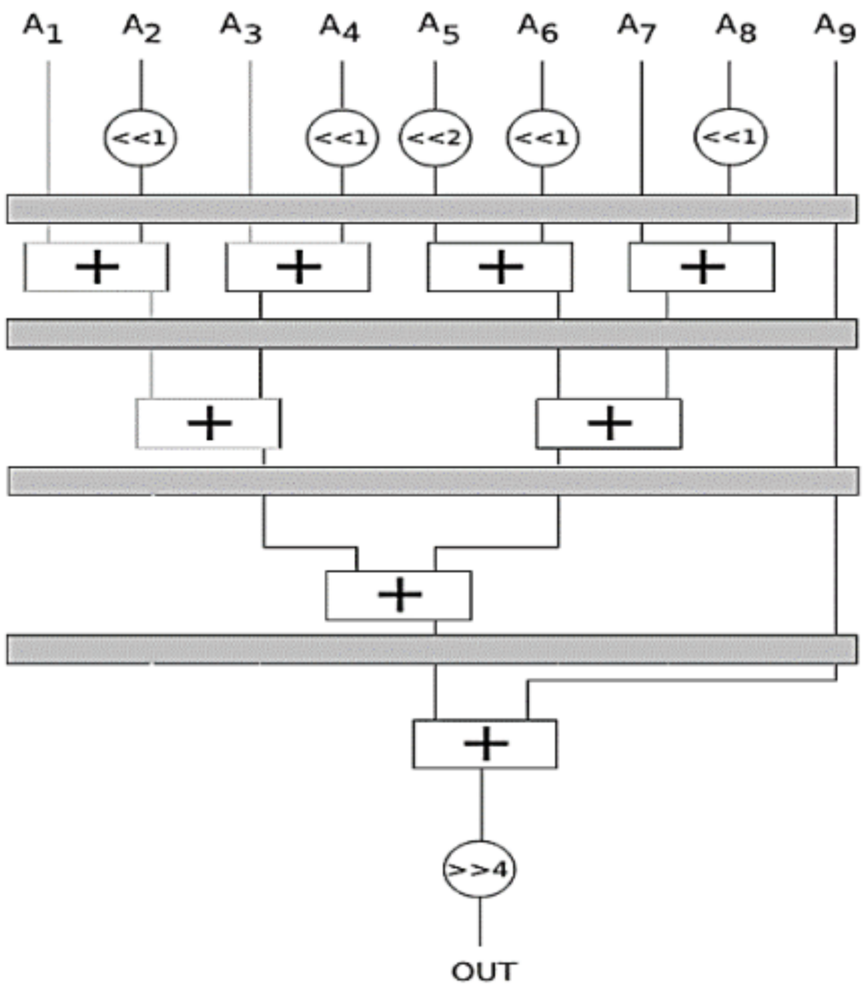


4. **Figure 3:** Displays the Gaussian filter kernel weights used in the 3×3 window, highlighting the importance of central pixels.

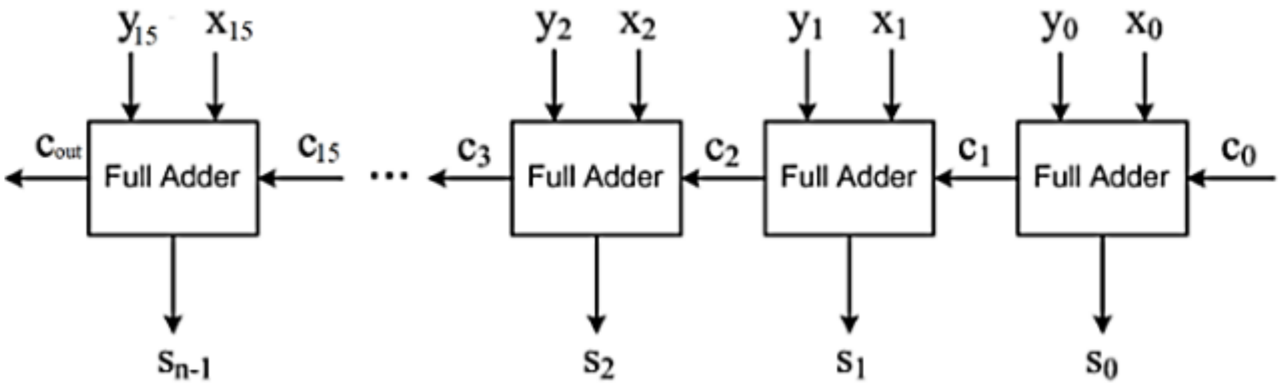


5. **Figure 4:** Block diagram of the Gaussian filter implementation, showing adders and shifters used for multiplication and division by

powers of two (implemented as bit shifts for hardware efficiency).

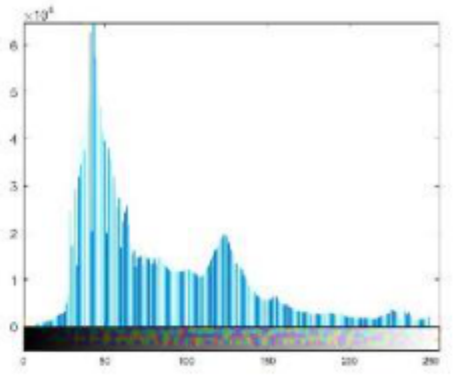


6. **Figure 5:** Pipeline Gaussian filter architecture with four stages, enabling simultaneous processing steps to increase speed.

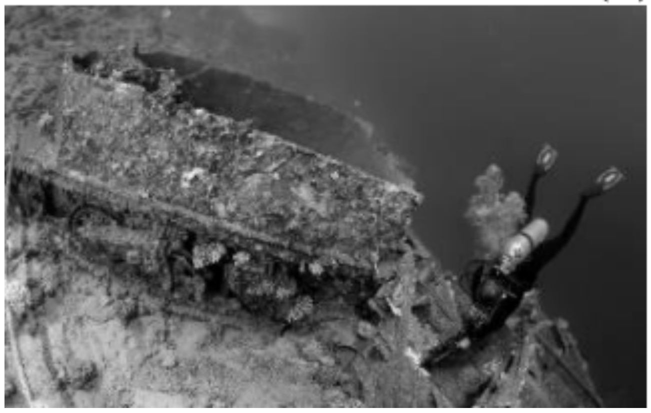
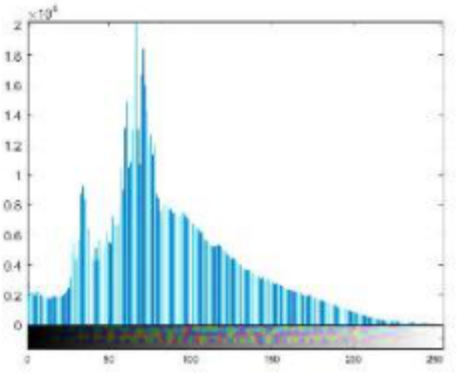


7. **Figure 6:** 16-bit carry ripple adder structure used to implement approximate adders by selectively simplifying addition in least

significant bits.

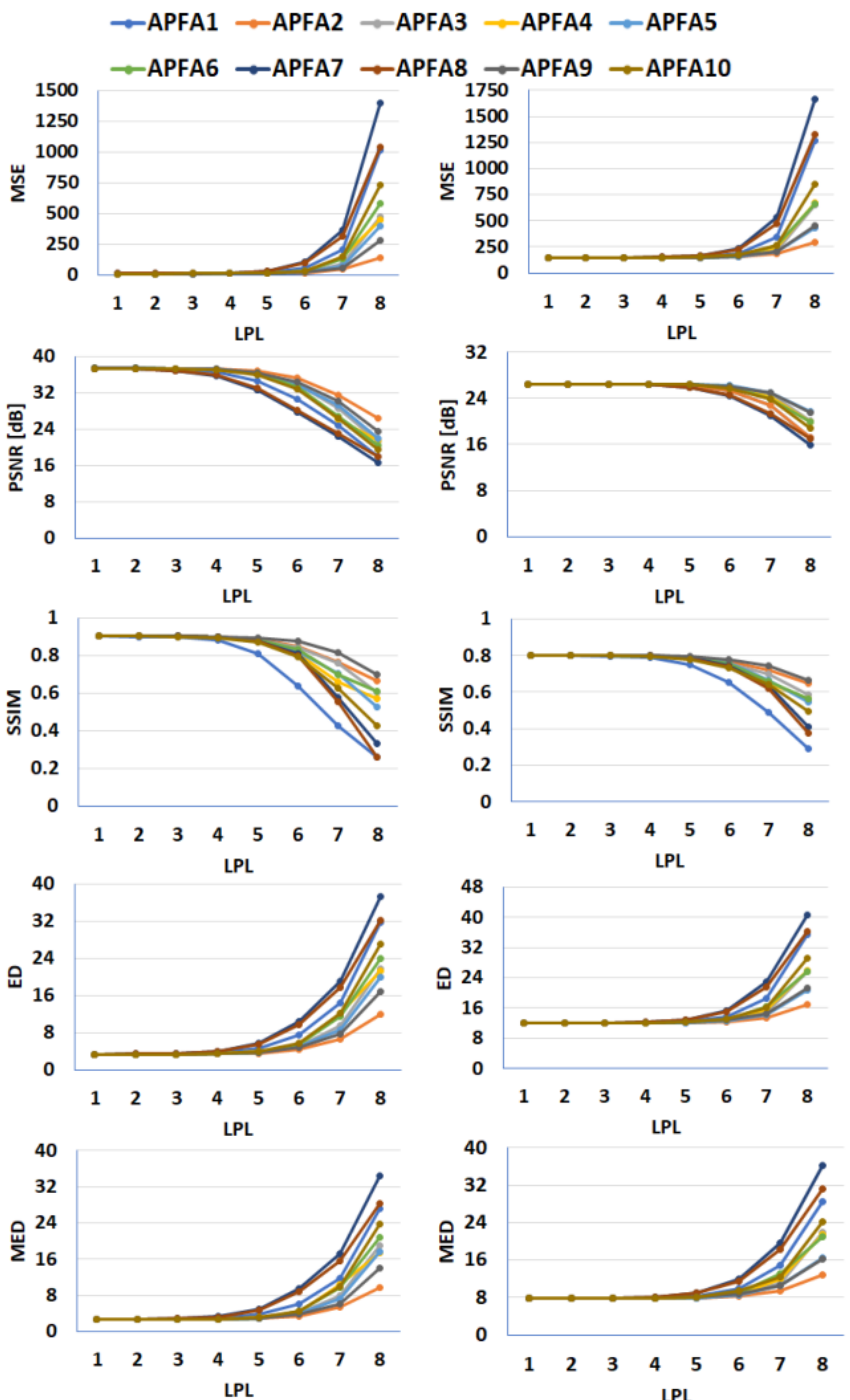


(a)

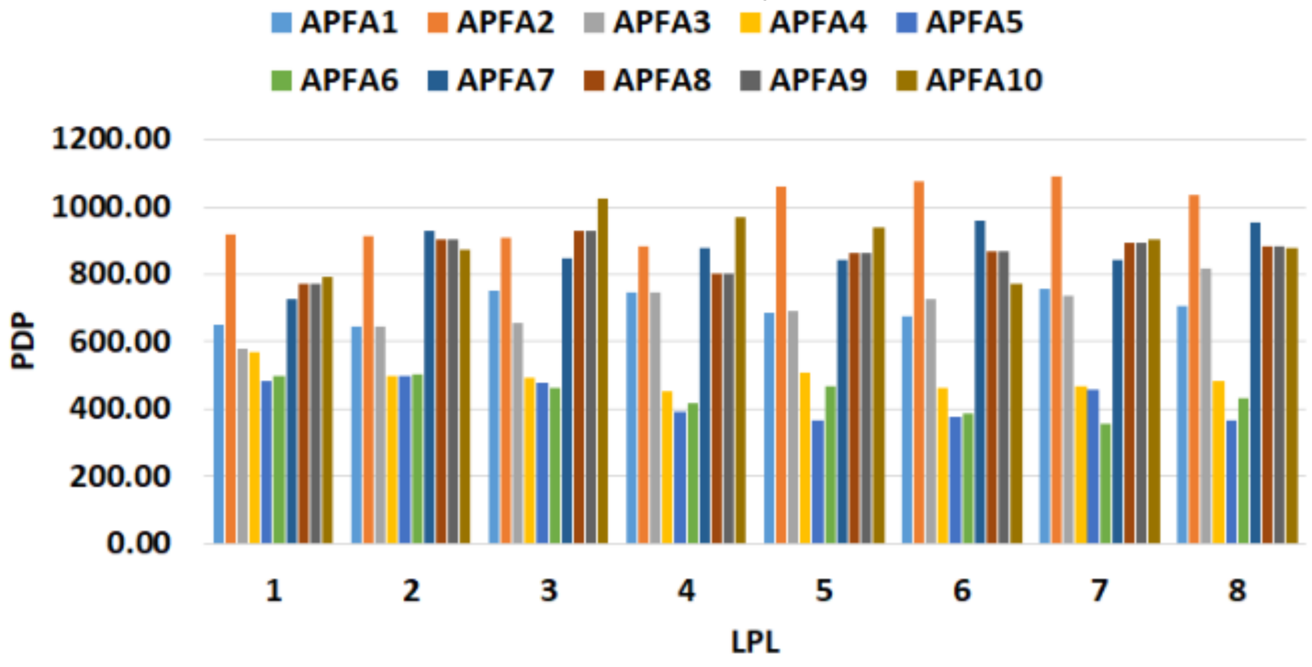



8. **Figure 7:** Sample underwater images (flatfish and diver) used for testing, along with their histograms showing pixel intensity

distributions.


APFA1
APFA2
APFA3
APFA4
APFA5
APFA6
APFA7
APFA8
APFA9
APFA10
MSE
PSNR [dB]
SSIM
ED
MED
LPL

9. **Figure 8:** Graphs of image quality metrics (PSNR, SSIM, etc.) versus the number of bits approximated in adders, demonstrating the trade-off between approximation level and output quality.



10. **Figure 9:** Power-Delay Product (PDP) comparison for different approximate adders, indicating which designs offer the best balance of speed and power efficiency.

11. **Conclusion**
    The study successfully demonstrates that implementing a pipeline Gaussian filter on FPGA with approximate adders significantly improves processing speed (over 150%) and reduces power consumption (over 34%) for underwater image enhancement. Although the hardware area increases, the trade-off is justified for applications tolerant to minor errors, such as real-time image and video processing. This approach advances underwater imaging by enabling faster, energy-efficient noise reduction, which is crucial for marine exploration, ecological monitoring, and underwater robotics where power and speed constraints are critical. The research highlights the practical value of approximate computing in balancing performance and resource use in embedded vision systems.