Image Pre-processor for Robust Deep Neural Network Inference Hardware

Zishen Wan, Kyungmi Lee

We propose the image pre-processor for robust deep neural network inference hardware resistant to adversarial attacks. We design and simulate three image filtering algorithms—2x2 median, 3x3 median, and non-local means denoising. To reduce the computation and increase the throughput of our image pre-processor, we adopt different optimization methods, such as data reuse, parallelism, and approximations. We further examine trade-offs in designing the image pre-processor with voltage scaling analysis and comparing different architectures.

Ⅰ. Robust Deep Neural Network Inference

Deep neural networks (DNN) are shown to be vulnerable to adversarial attacks [1], in which imperceptible manipulation of inputs results in wrong behaviors, such as wrong classification. Algorithm solutions to protect DNNs from adversarial attacks often require large model sizes [2] that are unsuitable for resource-constrained end devices. Alternatively, Feature Squeezing [3] proposed to utilize image pre-processing methods to detect adversarial attacks without changing the DNN itself. These methods include 2x2 median filtering, 3x3 median filtering, and non-local means denoising (NL-means denoising). Image pre-processing stage can limit the throughput of DNN inference hardware, due to large number of computations and communication overhead between different chips, motivating the low-overhead hardware implementation of the image pre-processor that can be later integrated with the DNN accelerator. Overall system design is shown in Figure 1.

Ⅱ. Median Filtering

Median filtering is a non-linear algorithm that involves sorting for the exact computation. Naive implementation of median filtering has high computational complexity with the increasing window size mainly due to sorting. Many studies aimed to reduce the computational complexity of median filtering at the algorithm level [4] [5] by leveraging different data structures to store and process pixel values, such as histograms or trees. Alternatively, faster sorting algorithms, such as Quick sort or Radix sort, can be utilized to accelerate median filtering.

However, these algorithm-based methods can increase memory requirements in hardware. For instance, histogram-based median filtering requires a 256-bin histogram to be stored and updated for each operation to produce a single pixel value. In hardware, the histogram can be implemented as registers or a SRAM, and frequent read and write access for these memory elements will increase the power consumption. Furthermore, faster sorting algorithms often show irregular computational pattern due to the splitting of the array, which makes them unsuitable for parallelization in hardware.

We aim to reduce the computational complexity of median filtering with 1) data reuse from previous window, and 2) removing computations that are unnecessary to produce a median value. As a window is sliding across the image, only one column changes for each computational step. Therefore, pixels in a window are partially-sorted with data reuse, and computation complexity of partially sorted arrays can be small even with basic sorting algorithms, such as selection sort. Also, only few comparisons on pixels can determine the median value for partially-sorted pixels, and we can eliminate unnecessary computations. These computational optimizations can reduce the area and the power consumption of the hardware, but hardwire the characteristics of median filtering with fixed window size, reducing the reconfigurability.

Furthermore, we aim to increase the throughput with 1) unrolling the sequential sorting operation to parallel comparison and mux-trees, and 2) pipelining. Comparison of different elements can be executed in parallel, and the comparison results can be used as control signals for mux-trees to generate a sorted array and a median value. This increases the combinational logic, but reduces the cycles for one sorting operation, and removes the storage of intermediate values during the sorting. Pipelining can be achieved for the modules that have sequential dependency. For example, preparing the next new column can be done while the other module is calculating the median value for the current window. Pipelining can increase the throughput, but the timing has to be carefully managed especially at the edge of images.

2x2 median filter (Figure 2) takes two pixels from the new column as inputs, and produces the median value of the 2x2 window as the output. It is implemented with two submodules, which sorts the new pixels (sort_new) and generates the median value (generate_median), respectively. Sorting the new pixels requires only a single comparison and two 2-1 multiplexers. Generating median uses the current and the previous cycle's sorted new pixels, and four comparisons and three 2-1 multiplexer operations to get the output. These two modules are implemented with two-stage pipeline.

3x3 median filter (Figure 3) has three submodules, which sorts the three pixels from the new column (sort_new), generates the median value of the current window (generate_median), and sorts two rightmost columns in the current window for the next cycle operation (generate_res). Submodule sort_new and submodule generate_res prepare the pixels to be partially-sorted, so that computations for generating the median value can be simplified. Submodule sort_new and submodule generate_median are similar to those in 2x2 median filter, except that the number of comparators and the multiplexers has to be adjusted to the new window size. Submodule generate_res takes the current and the previous cycle's new three pixels, and sorts them to produce a 6-element sorted array, which then is used for submodule generate_median. These three modules are implemented as three-stage pipeline.

Ⅲ. Non-Local Means Denoising

In NL-means denoising [6] [7], given a discrete noisy image, the estimated value for a pixel is computed as a weighted average of all the pixels in the image, where the weights depend on the similarity between two pixels. The similarity between two pixels depend on the similarity of the intensity gray level vectors of square neighborhoods of pixels. This similarity is measured as a decreasing function of the weighted Euclidean distance. Therefore, NL-means not only compares the grey level in a single point but the geometrical configuration in a whole neighborhood. This fact allows a more robust comparison than neighborhood filters.

Three are two main issues in the hardware implementation: 1) Trade of among computation complexity, power and area, 2) exponential and division implementation. For one pixel in image, we adopt the size of 7 and 3 for search window and patch window separately. Therefore, we need to calculate $3 \times 3 \times 7 \times 7$ Euclidean distances to denoise one pixel, which will cost much in hardware. To reduce computation complexity, we come up with an optimized NL-means denoising method by distance reusing. For pixels in the first row of input image, we store the distances of each row of patch window in the memory. When denoising pixels at the second row of input image, we reuse two distances stored before for the specific pixel pairs with the same relative location. So we only need to calculate distances of one row in each patch widow. Then we store the newly calculated distance and one old distance in the memory, and reuse them when we denoising pixels at the third row. By iterating this procedure, we can get the whole denoising image. In this way, we denoise each pixel in the first row with calculating 3 x 3 x 7 x 7 distances, and 3 x 7 x 7 distances for other rows, so the computational complexity decreases by 2/3. However, frequent read and write access for these distance memory elements will increase the power consumption and the memory area. A detailed comparison will be showed in section Ⅵ.

To calculate weight of each pixel in search window, we need to use exponential and division function, which are expensive in hardware implementation. Considering the accuracy of limited order Taylor expansion is not enough, we adopt look-up table method for exponential function. And we use minus and shift to achieve 22-bit division function. Design of NL-means denoising hardware is shown in Figure 4.

We design two main submodules for the conventional NL-means denoising *without* the distance data reuse: 1) weight calculation for one patch window, 2) weight calculation for one search window and denoising for one pixel. For one patch submodule, we take 18 pixels from two compare patch windows as input and calculate the distance, serving as the similarity between the centered pre-denoising pixel and one of pixels in search window. Then we output the weight of pixel in search window by exp conversion. For one search window submodule, we take 49 weights of all pixels in search window and output the estimated value of centered pixel by averaging as well as normalization, which achieve denoising for one pixel.

For the optimized NL-means denoising method *with* the distance data reuse, we design three main submodule: 1) distance calculation for one row in patch window, 2) weight calculation for one patch window, 3) weight calculation for one search window and denoising for one pixel. In one row of patch window submodule, we take 6 pixels from the same row of two comparable patch windows as input per cycle, calculate the distance and write them into memory. In one patch window submodule, we use the newly calculated distance and reuse two distances read from memory to calculate the weight of pixels in search window. The denoising module is similar with the previous method.

Ⅳ. Simulation Results

The performance of 2x2, 3x3 median filter, and non-local means denoising is summarized in Figure 5. We use Cadence 90nm technology for both synthesis and layout. For median filters, the area is measured after place and route using Cadence Encounter, and the energy is measured with Synopsys Nanosim simulation after the layout. For non-local means denoising, the area and the energy is measured after the synthesis. Note that the area and the energy is only simulated for the computation module, not including the memory for storing input image or I/O. We extrapolate the area and the power associated with the memory with the estimated read/write power of 90nm technology SRAM. We verify the correct functionality of 2x2, 3x3 median filter, and non-local means denoising by feeding the real images with varying sizes to the HDL testbench (Figure 6). Since 2x2 and 3x3 median filter exactly compute the median value, the output of the HDL testbench should match with the software-processed image. For non-local means denoising, we use approximation in the weight and division calculation, and the output of the HDL testbench might not exactly match with the output of software. Nevertheless, for the sufficient functionality, the testbench and the software outputs should be similar. We also report the speedup of our hardware implementation in comparison to the CPU runtime.

Ⅴ. Effect of Voltage Scaling on Median Filters

We examine the effect of voltage scaling on the energy and the clock frequency (Figure 7). Reducing the operation voltage at the expense of the clock frequency can be beneficial for 2x2 and 3x3 median filters when combined with the subsequent deep neural network (DNN) accelerator module, because 2x2 and 3x3 median filters have higher throughput when compared with the typical throughput of DNN accelerators [8]. Reducing the clock frequency and reducing the energy consumed by 2x2 and 3x3 median filters can reduce the energy of the overall system. We observe that our 2x2 and 3x3 median filter can operate at 0.35V without an error at 10MHz, which gives the throughput of 67 ImageNet-sized images per second.

Ⅵ. Comparison of Different NL-means Architectures

In hardware implementation, we explore three different NL-means architectures: 1) conventional NL-means with 18-pixel input per cycle, 2) conventional NL-means with 6-pixel input per cycle, 3) optimized NL-means with 6-pixel input per cycle. We compare these three methods from 3 aspects: 1) area, 2) energy consumption per image, 3) operation cycles and speed up compared with software (Figure 8). For total area, 6-pixel input conventional NLM is smaller, because it doesn't have extra circuit for other 12 pixels operation and doesn't use distance memory to read and write. Accordingly, 18-pixel input conventional NLM is largest. For energy consumption without memory, the optimized NLM architecture is most energy saving, because it reuses distance and only compute 3 pixel pairs' distance for one pixel in search window. But with memory, the optimized NLM will read and write 74088 times from memory for one image. Considering the read/write power for the memory, the energy consumption of optimized NLM is about 1.8 times of 18-pixel input conventional NLM. For operation cycles, optimized NLM and 18-pixel input conventional NLM is about 2.8 times faster than 6-pixel input conventional NLM, and can speed up about 8 times compared with software.

Ⅶ. Future Work and Conclusion

In this work, we present the hardware implementation of 2x2 median filter, 3x3 median filter, and NL-means denoising, which are shown to be effective at detecting adversarial attacks in DNNs. Our key contributions are 1) efficient implementation of median filters with data reuse and unrolled sorting operations, 2) energy and resource aware implementation of NL-means denoising with approximation on expensive calculations such as exponential units, and 3) exploring diverse trade-offs in designing the image-preprocessor with voltage scaling analysis and architecture comparisons.

We think further work on efficient I/O design can reduce the energy consumption of median filters. SRAM read/write accounts for large proportion of the energy for median filters, and implementing I/O as streaming instead of SRAM can improve energy-efficiency. Furthermore, the effect of approximation in NL-means denoising can be studied for real adversarial attacks to ensure the robustness of our image pre-processor. Also, there can be diverse application of our image pre-processor other than robust DNN inference, such as texture synthesis where non-local algorithms are widely used.

Acknowledgements:

The authors gratefully acknowledge and Prof. Vivienne Sze and Utsav Banerjee for helpful discussion.

References:

[1] I. Goodfellow, J. Shlens and C. Szegedy, "Explaining and Harnessing Adversarial Examples,"*International Conference on Learning Representation*, 2015.

[2] A. Madry, A. Makelov, L. Schmidt, D. Tsipras and A. Vladu, "Towards Deep Learning Models Resistant to Adversarial Attacks,"*International Conference on Learning Representation*, 2018.

[3] W. Xu, D. Evans and Y. Qi, "Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks,"*Network and Distributed Systems Security Symposium*, 2018.

[4] T. Huang, G. Yang and G. Tang, “A Fast Two-Dimensional Median Filtering Algorithm,” *IEEE Transactions on Acoustics, Speech, and Signal Processing,* Vol. ASSP-27: 1, 1979.

[5] B. Weiss, “Fast Median and Bilateral Filtering,” *ACM Transactions on Graphics,* Vol. 25: 3, pp. 519-526, 2006.

[6] A. Buades, C. Bartomeu and J.-M. Morel, “A non-local algorithm for image denoising,”*Computer Vision and Pattern Recognition*, 2005.

[7] J. Froment, “A Parameter-Free Fast Pixelwise Non-Local Means Denoising,” *Image Processing On Line,* Vol. 4, pp. 300-326, 2014.

[8] Y.-H. Chen, T. Krishna, J. Emer and V. Sze, “Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks,” *Journal of Solid-State Circuits,* Vol. 52: 1, pp. 127-138, 2017.

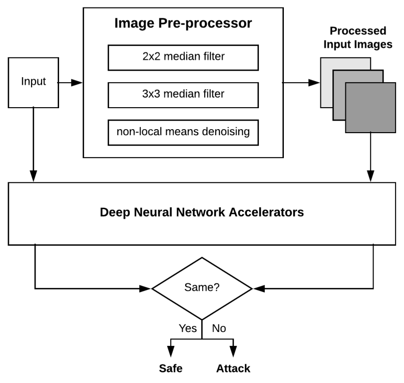

Figure 1: Overall system design of robust DNN hardware

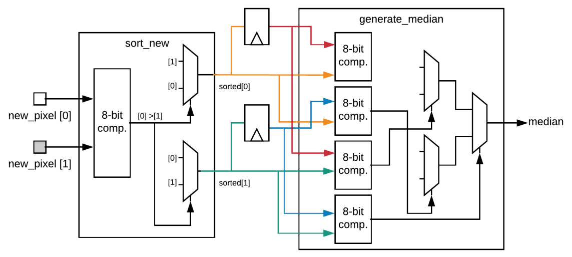

Figure 2: Design of 2x2 median filter and its submodules

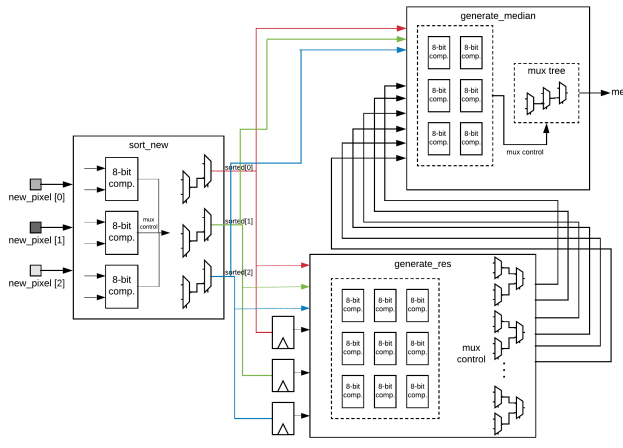

Figure 3: Design of 3x3 median filter and its submodule

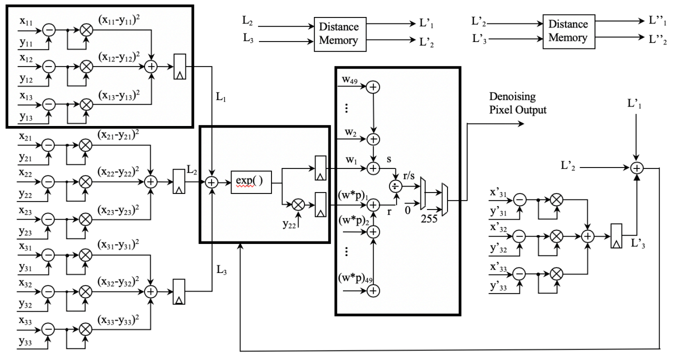

Figure 4: Design of NL-means denoising and its submodule

	2x2 median	3x3 median	NL-means	NL-means opt
Area (μm^2)	1395.7	4478.6	41052	31391
MNIST Input (28 × 28), Operation @ 50MHz, 1.2V				
Energy (nJ per image)	1.052	2.653	376.315	286.627
Energy with memory (nJ per image)	14.938	20.608	445.520	801.749
# of cycles	868	896	41552	40234
Speedup	22.47	25.28	7.06	9.26
ImageNet Size Input (3 × 220 × 220), Operation @ 50MHz, 1.2V				
Energy (nJ per image)	178.41	417.57	68615.52	54135.46
Energy with memory (nJ per image)	2879.91	4046.4	70143.2	133150.6
# of cycles	147180	147840	2510320	2506880
Throughput	339 images	338 images	13 images	13 images
Speedup	2.80	7.79	9.41	10.08

Figure 5: Performance summary table

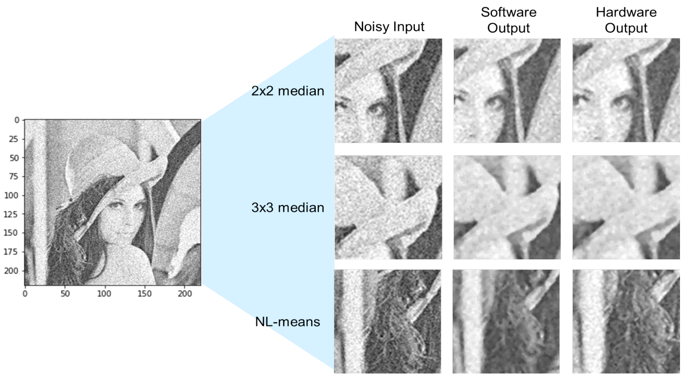

Figure 6: Functionality on noisy image

Voltage Scaling Simulation				
	Freq (MHz)	Lowest Vdd (V)	Computational Energy (nJ per image)	Throughput
2x2 median	10	0.35	10.199574	67.94401413
	50	0.38	14.9888112	339.7200707
	100	0.45	21.7038987	679.4401413
	200	0.5	24.041853	1358.880283
3x3 median	10	0.35	21.422016	67.64069264
	50	0.41	39.48422016	338.2034632
	100	0.5	53.85072	676.4069264
	200	0.57	71.73236763	1352.813853

Figure 7: Effect of voltage scaling on median filters

Comparison of three different NL-means architectures				
① 18-pixel input conventional NL-means architecture				
② 6-pixel input conventional NL-means architecture				
③ 6-pixel input optimized NL-means architecture				
Evaluation	①	②	③	Compare
Area (μm2)	41052	31391	29149	② < ③ < ①
Energy without memory (nJ)	376.315	286.627	361.212	③ < ② < ①
Energy with memory (nJ)	442.520	801.749	892.134	① < ③ < ②
# of cycles	41552	40234	121664	③ < ① < ②
speed up	7.06	9.26	2.41	③ < ① < ②
Operation with MNIST Input @ 50MHz, 1.2V				

Figure 8: Comparison of different NL-means architecture