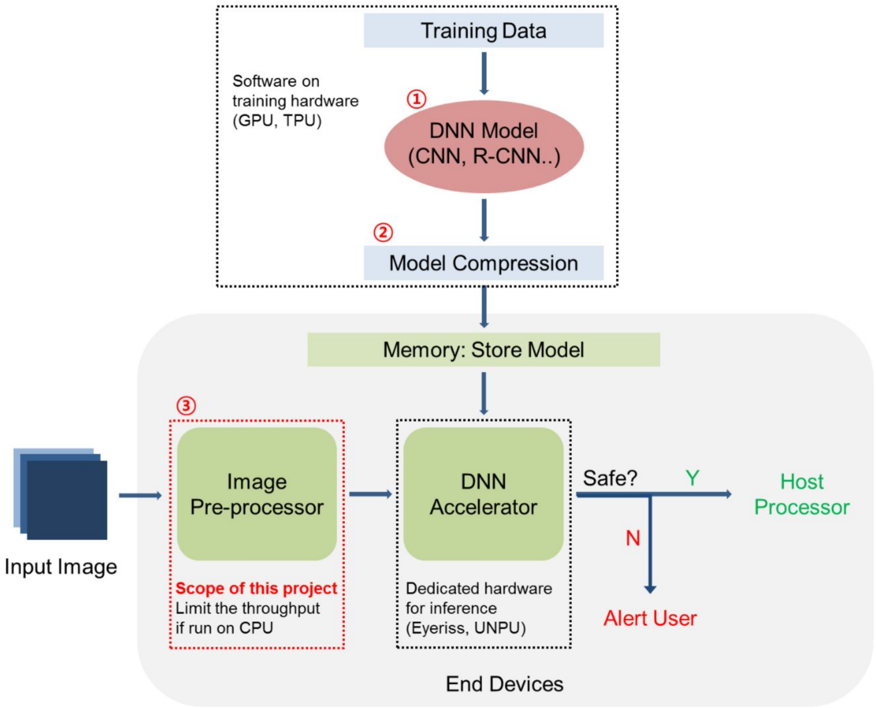
Massachusetts Institute of Technology
6.374 Final Project

Image Pre-processor for Robust Deep Neural Network Inference Hardware

Kyungmi Lee & Zishen Wan

Motivation and Background



Three possible ways to achieve robust DNN:

① Train DNN model itself

② Model compression

③ Image preprocessor
(Scope of this project)

Model Overview

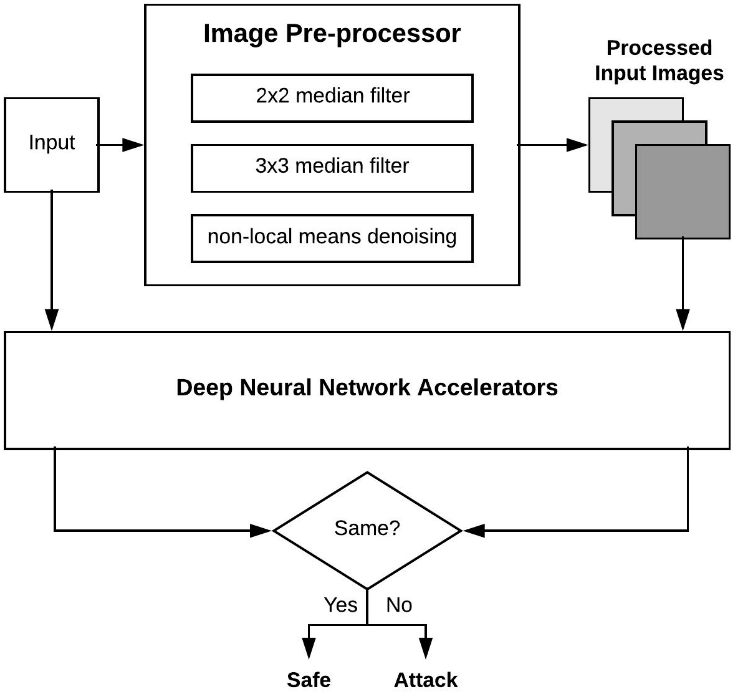
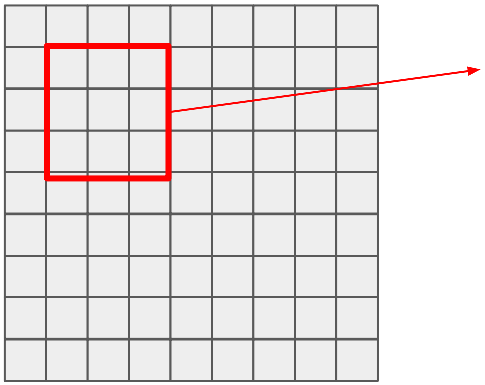


Image pre-processor includes:

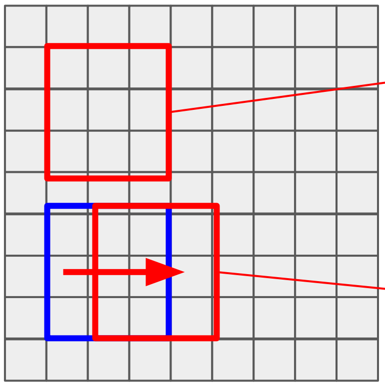
- 2x2 median filter
- 3x3 median filter
- Non-Local means denoising
  - conventional NL-means denoising
  - optimized NL-means denoising

Median Filters



Sort elements in the window,
Find the median value

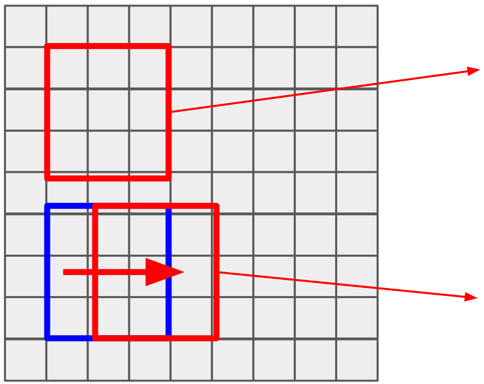
Median Filters



Sort elements in the window,
Find the median value

Only one column changes at each step
→ **Data reuse** allows elements to be **‘partially sorted’**

Median Filters



Sort elements in the window,
Find the median value

Only one column changes at each step
→ **Data reuse** allows elements to be **‘partially sorted’**

1. Sort the **new** column (i.e. 3 pixels)
2. **Find median** value with 6 pixels (sorted from the previous window) and the sorted 3 new pixels
3. Prepare sorted 6 pixels for the **next** cycle

Median Filters

Reduce Computation

- ***Data reuse***
  partially sorted array can be easily sorted with selection sort
- ***Remove unnecessary operations***
  finding median and preparing partially sorted array for the next cycle do not require sorting of all elements in the window

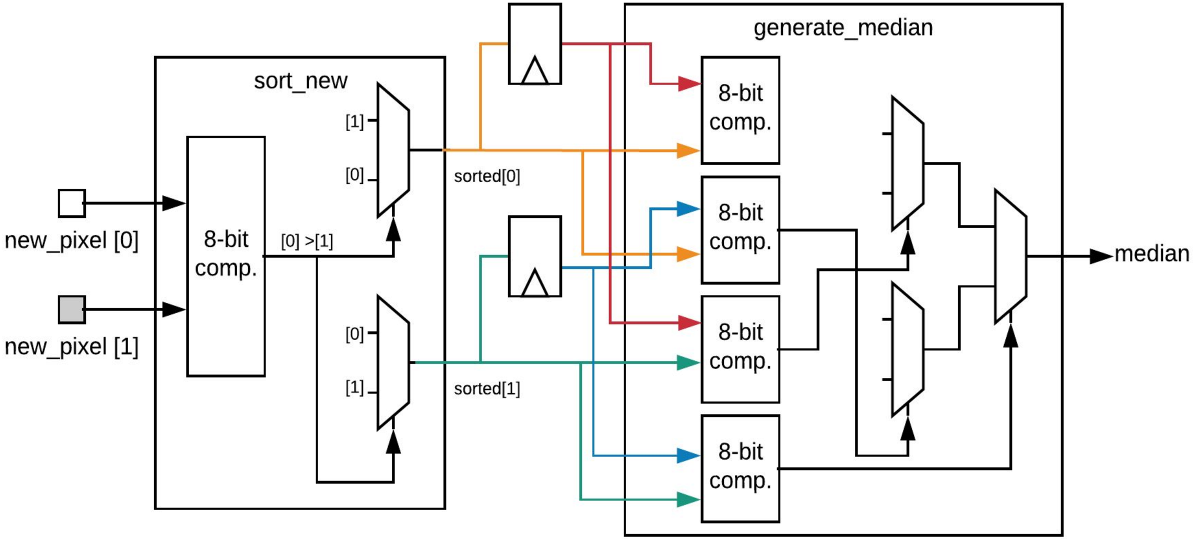
Reduced reconfigurability

Increase Throughput

- ***Unroll operations in sorting***
  sequential sorting operations to unrolled comparisons and mux-trees
- ***Pipelining***
  prepare data for the next cycle while finding the median for the current window

Increased combinational logics,
Careful timing control @ edges

2x2 Median Filter



Two submodules:

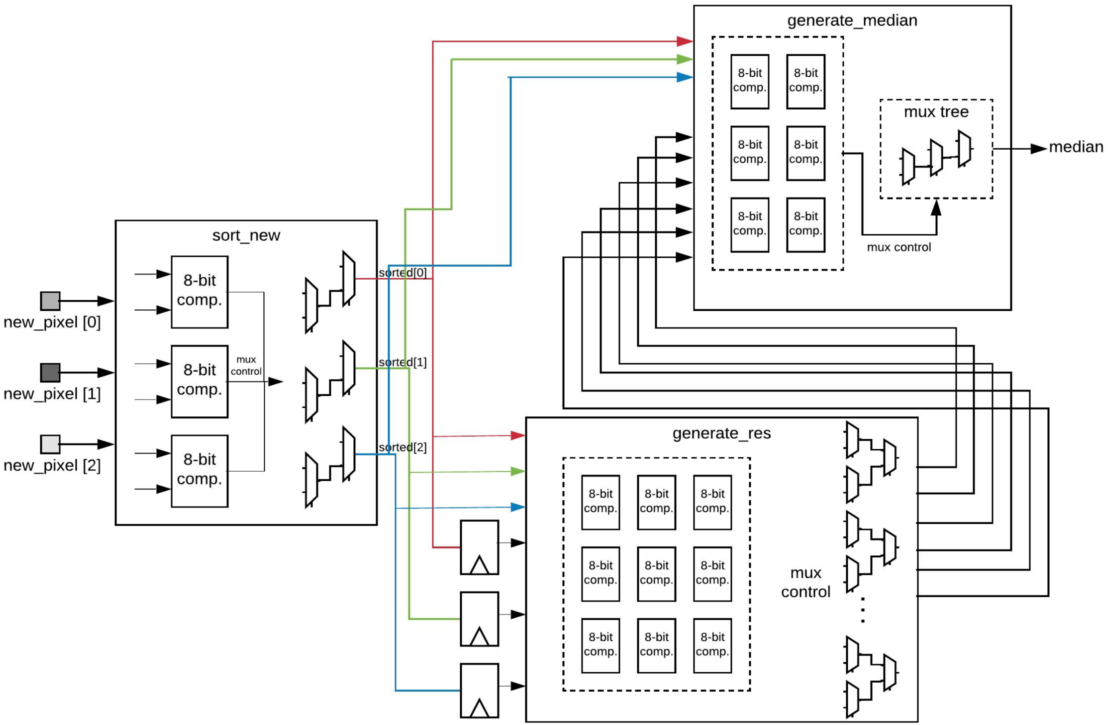
- sort_new:

  1 comparison, 2 mux

- generate_median:

  4 comparison, 3 mux

3x3 Median Filter



Three submodules:

- sort_new

  3 comparison, 6 mux

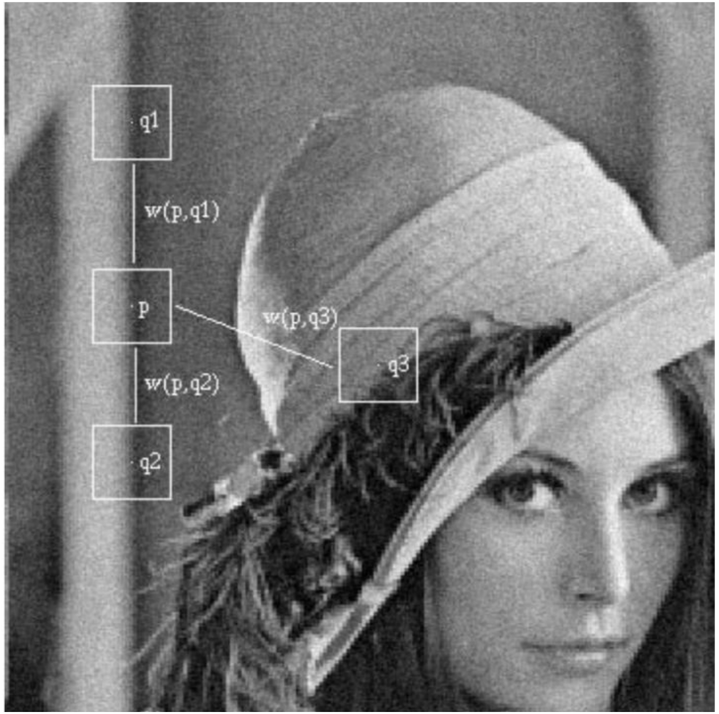
- generate_res

  9 comparison, 18 mux

- generate_median

  6 comparison, 6 mux

Non-local Means Denoising



- Pixel denoising: weighting average of all pixels in the image

$$NL[v](i) = \sum_{j \in I} w(i,j) v(j)$$

- Weight: depend on the similarity between pixels i and j

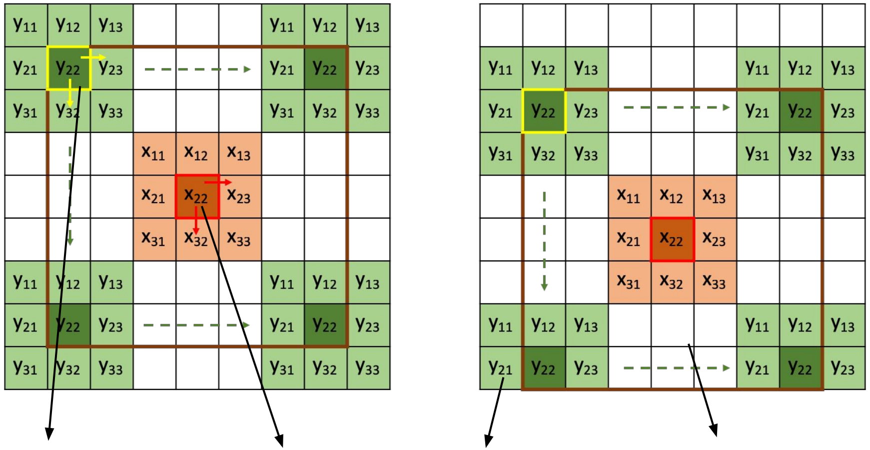
$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{\|v(\mathcal{N}_i) - v(\mathcal{N}_j)\|^2_{2,a}}{h^2}}$$

- Normalizing Constant

$$Z(i) = \sum_j e^{-\frac{\|v(\mathcal{N}_i) - v(\mathcal{N}_j)\|^2_{2,a}}{h^2}}$$

Non-local Means Denoising

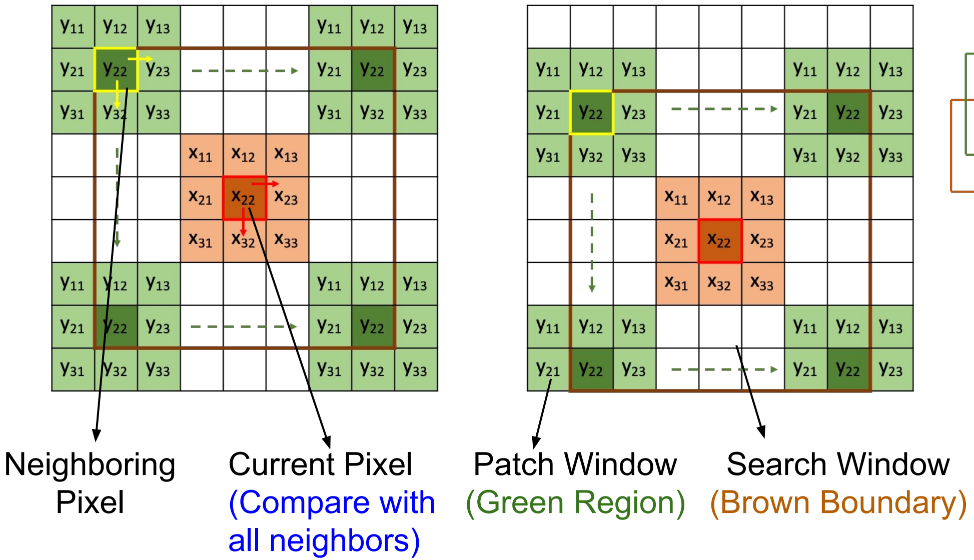
- Conventional Method:



For each comparable patch window: 9-pixel pairs’ distances
Denoising one pixel: calculate 3x3x7x7 distances

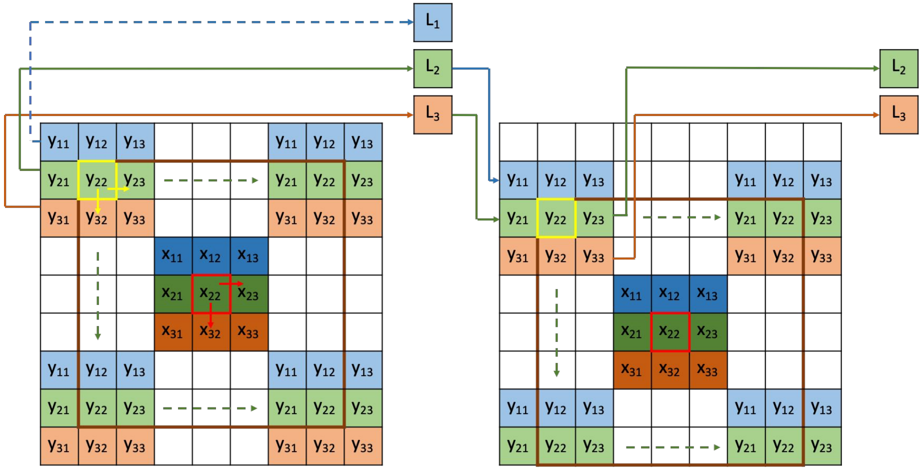
Non-local Means Denoising

- Conventional Method:



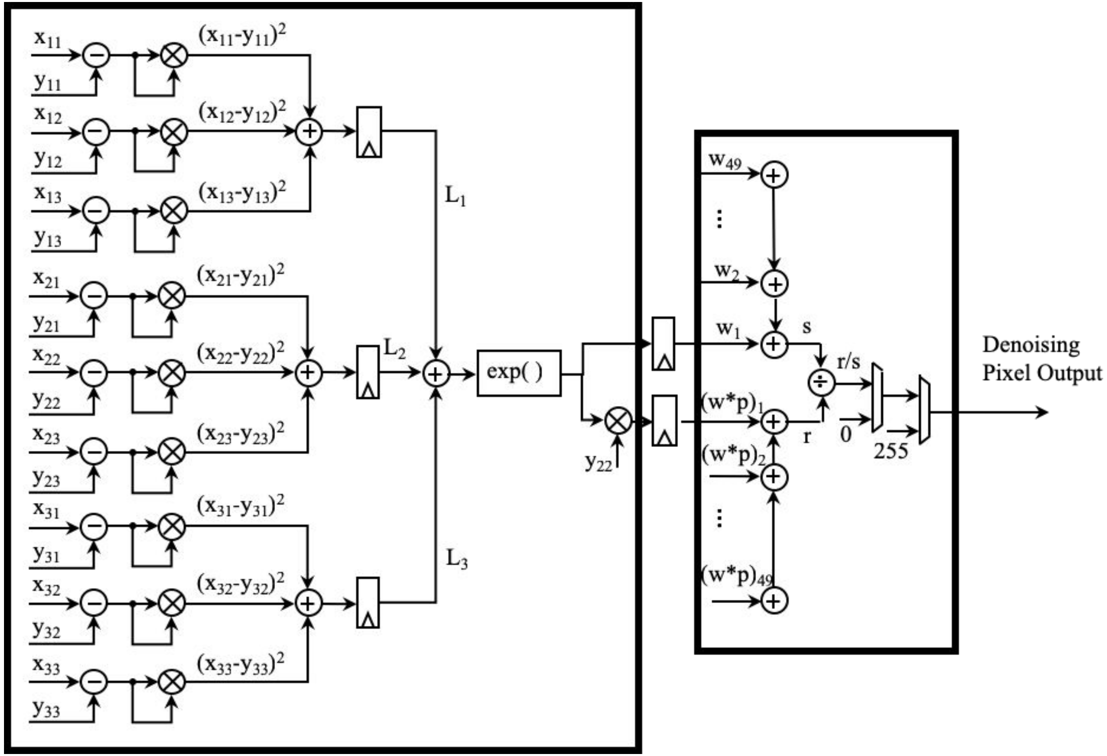
For each comparable patch window: 9-pixel pairs' distances
Denoising one pixel: calculate 3x3x7x7 distances

- Optimized Method:



Reuse Distances
For each comparable patch window: 3-pixel pairs' distances
Denoising one pixel: calculate 3x7x7 distances

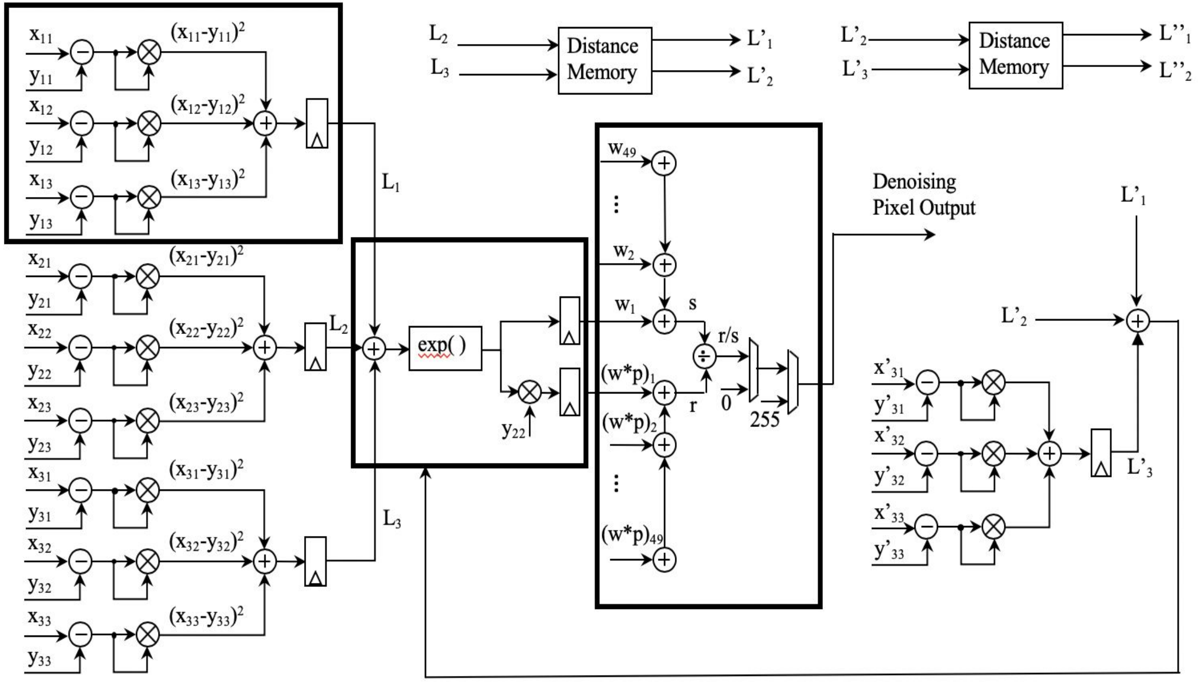
Non-local Means Denoising (Conventional)



Two submodules:

- Weight Calculation for one patch window
- Weight Calculation for one search window and denoising for one pixel

Non-local Means Denoising (Optimized)



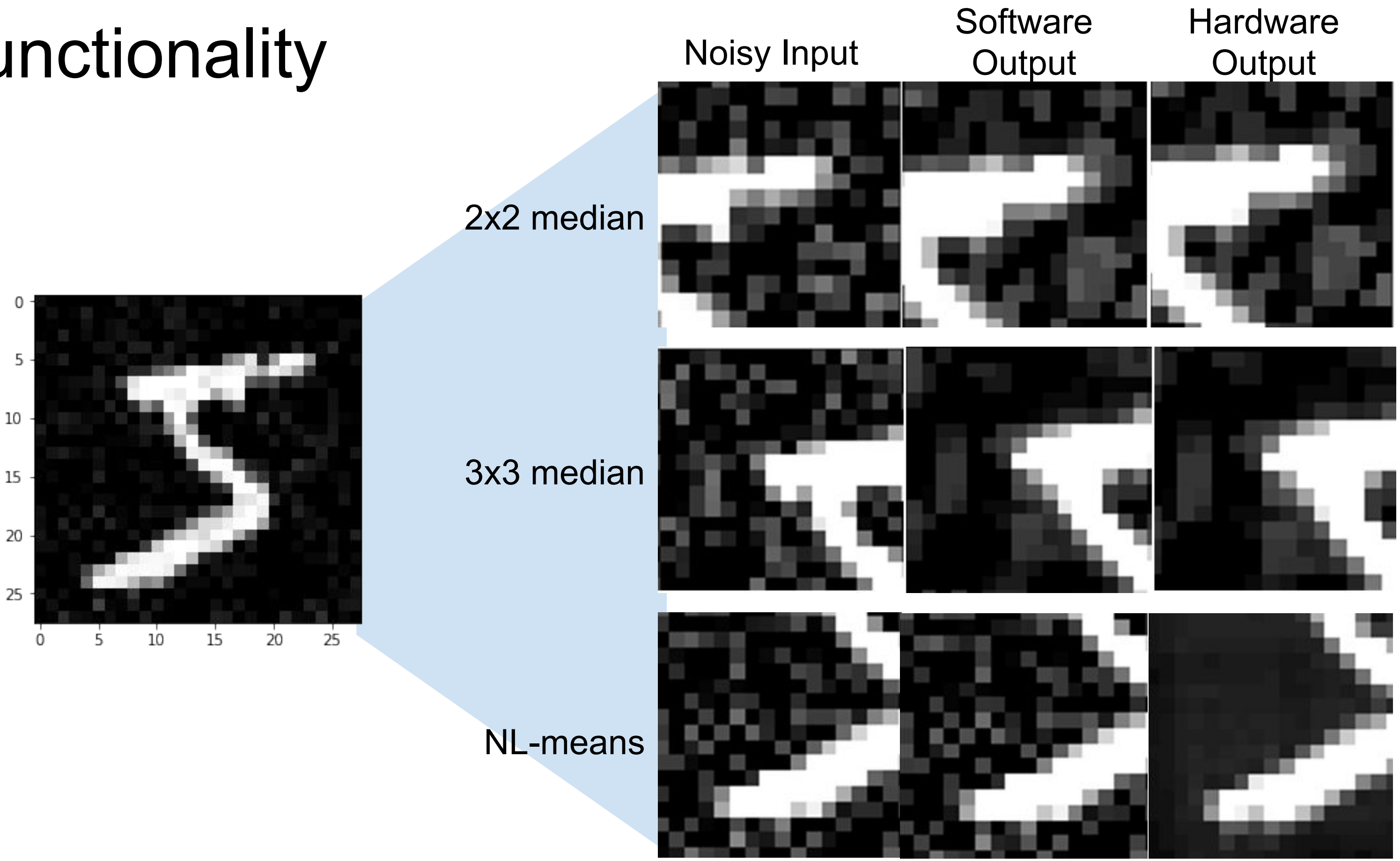
Three submodules:

- one row in patch
- one patch
- one search & one pixel denoising

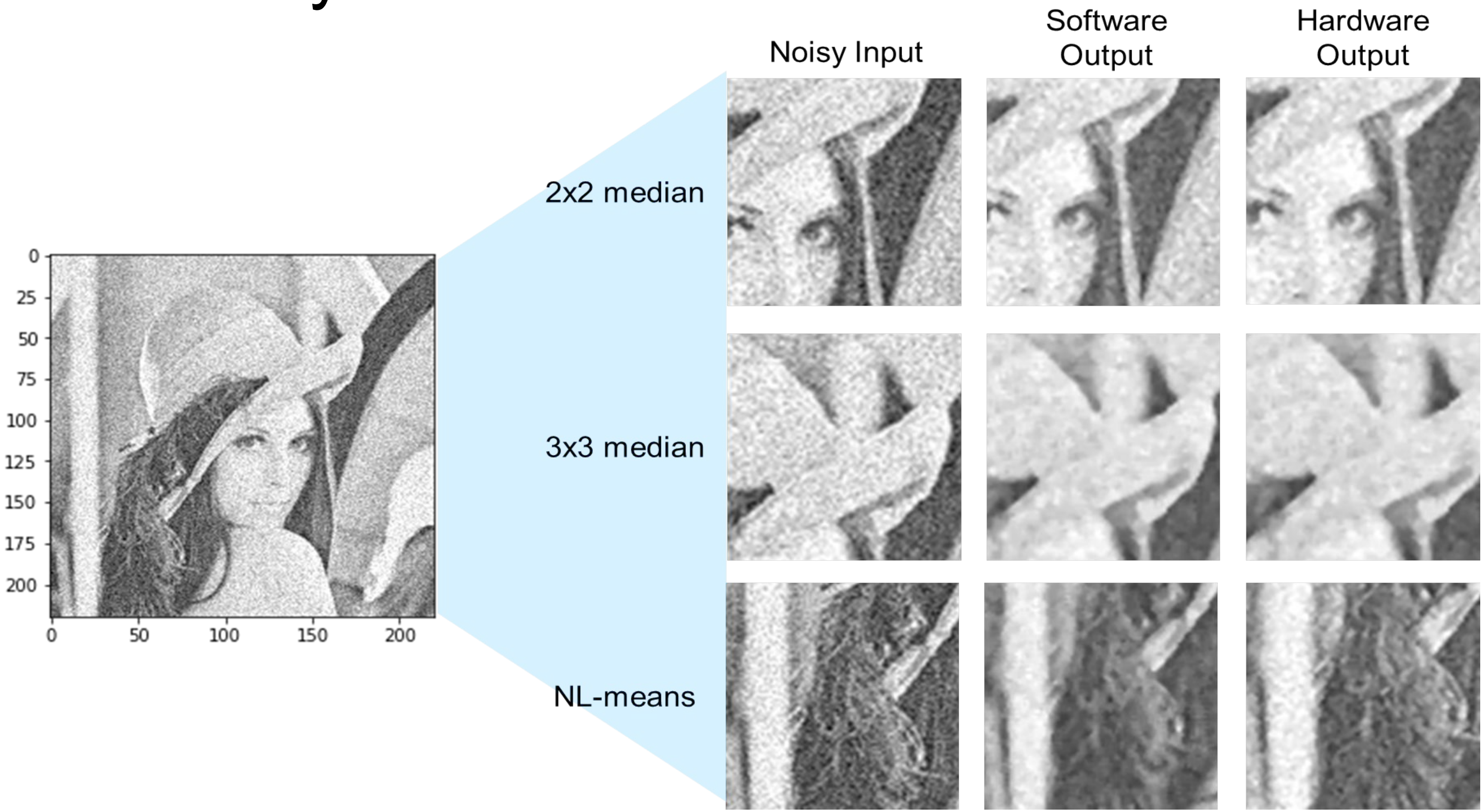
Two hardware Issues:

- Trade-offs
- exp. and div. units

Functionality



Functionality



Simulation Results

	2x2 median	3x3 median	NL-means	NL-means opt
Area (µm2)	1395.7	4478.6	41052	31391
MNIST Input (28x28), 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	445.52	801.749
# of cycles	868	896	41552	40234
Speedup	22.47	25.28	7.06	9.26
ImageNet Size Input (3x220x220), 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	2.8	7.79	9.41	10.08

Median Filter: Voltage Scaling Simulation

Voltage Scaling Simulation				
	Freq (MHz)	**Lowest Vdd** (V)	**Computational Energy** (nJ per image)	**Throughput**
2x2 Median Filter	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 Filter	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

Non-local Means: Architecture Comparison

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	29149	31391	① < ③ < ②
Energy without memory (nJ)	376.32	361.2	286.6	① < ② < ③
Energy with memory (nJ)	442.52	892.1	801.71	② < ③ < ①
# of cycles	41552	121664	40234	② < ① < ③
speed up	7.06	2.41	9.26	② < ① < ③

Conclusion and Contribution

- Conclusion
  - **Hardware implementation** of 2x2 median filter, 3x3 median filter and NL-means denoising
  - Design choices for **improved throughput** and **reduced computation**
- Contribution
  - Effective implementation of median filters with **data reuse** and **unrolled sorting operations**
  - Energy and resource aware implementation of NL-means denoising with **weight reuse**, and **approximation** on expensive calculations.
  - Explore **diverse trade-offs** in designing the image-preprocessor with voltage scaling analysis and architecture comparisons.

Future work

- Efficient I/O design:
  Streaming instead of SRAM to reduce read/write power and area overhead
- Approximation of NL-means instead of exact calculation:
  Effect on detecting adversarial attacks
- Application of the image pre-processor in other domains:
  ex) NL algorithms are widely used for texture synthesis

Thank you！

…. Questions ?