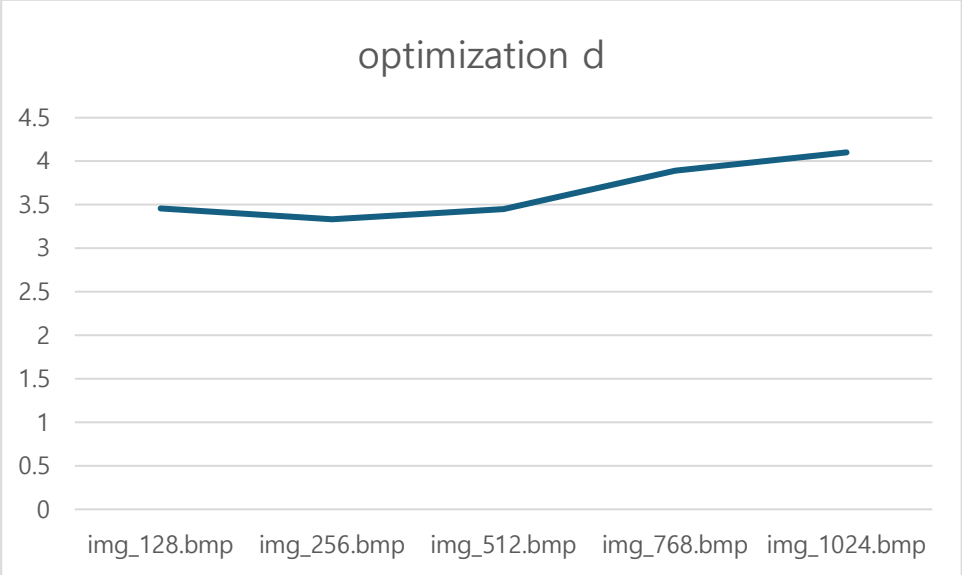
[시스템 프로그래밍 Project 2]

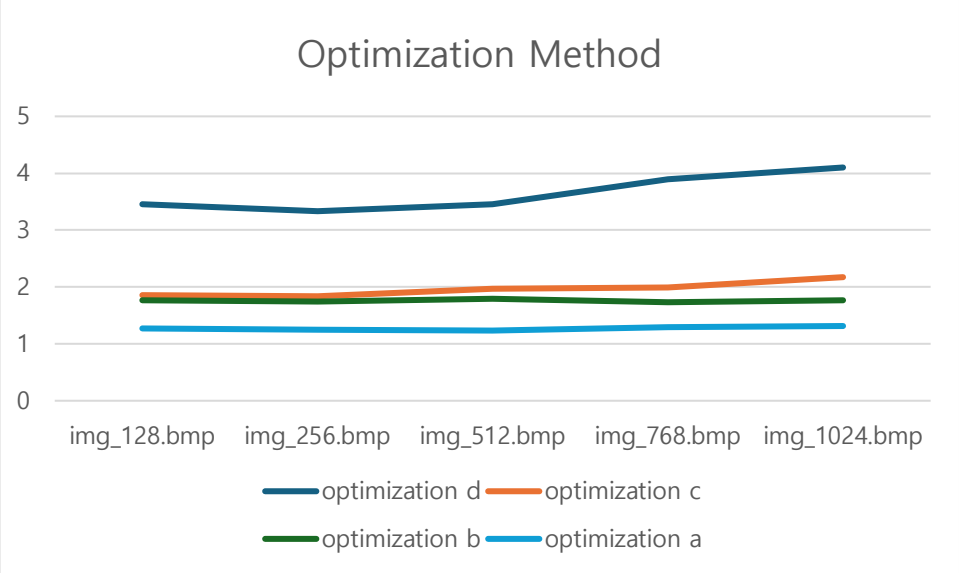
202011192 장현영

1. Implementation Result



I used the optimized version to check the speed-up for each image file in a Raspberry Pi environment. Due to variability, each file was executed three times, and the average value was calculated. The result showed that img_1024.bmp had the highest speed_up of 4.1013.

2. Optimization Approaches



a. filter_optimized() with code motion

In the loop, many multiplications were repeatedly calculated in baseline code. By dividing the calculation into multiple units and processing the operations that can be handled outside the loop as much as possible, I applied code motion and confirmed the

first performance improvement technique. And there was a performance improvement of speed_up = 1.314.

b. filter_optimized() with loop unrolling and code motion

The 'for' loop inherently show overhead due to its instruction level structure. Therefore, excluding the dynamic 'width' and 'height' of the input image, loop unrolling was performed in the direct convolution operation based on the size of various filter(gaussian, edge), utilizing the characteristic that the filter is fixed at a 3*3 size. And there was a performance improvement of speed_up = 1.764.

c. filter_optimized() with pre-processing(zero-padding), loop unrolling and code motion

After completing step b, there was no significant performance improvement. The main bottleneck was the repetitive checks for whether the convolution operation went out of image bounds and real-time zero-padding. This was handled by repetitive 'if' code-level instruction. Therefore, I performed pre-processing in advance, allowing the operation to be executed within the image size without 'if' statements. And there was a performance improvement of speed_up = 2.173.

d. filter_optimized() with integer arithmetic, pre-processing(zero-padding,), loop unrolling and code motion

After completing all methods up to step **c**, various optimization techniques were applied but did not yield significant results. During the process of changing to appropriate data types, I found that the data type of the filter, which plays the most crucial role in the convolution operation, was `float`. Since integer operations are the most efficient for optimization, I processed the data type of the filter appropriately and converted it to an integer type without affecting the computation results. After applying the processed filter to the convolution operation, a significant speed-up was achieved. And there was a performance improvement of speed_up = 4.1013.