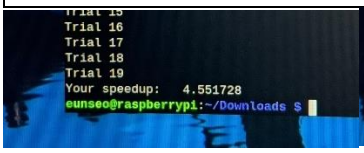
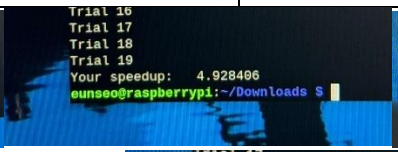
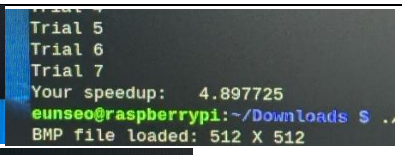
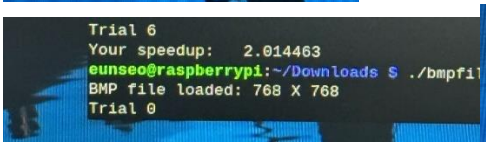
- **Result**

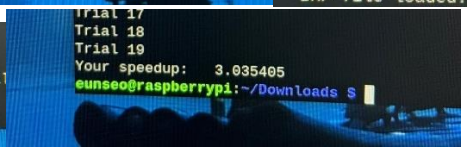
Image size	The speedup of my best
img_128	4.551728
img_256	4.928406
img_512	4.897725
img_768	2.014463
img_1024	3.035405











Every time I run the program, the speedup value comes out slightly different, and the largest value among the values that were returned several times was put in the report.

- **Optimization approach**

The first is memory allocation and release removal. In filter_baseline, memory is allocated for each pixel using malloc and released using free. This memory allocation and release process is a very expensive operation, and instead, necessary variables are assigned to the stack and automatically released when the function is terminated, improving the speed.

The second is the removal of function calls. In filter_baseline, a convolution function is called for each pixel. The function call itself also has overhead, and especially if the number of pixels is large, it can have a big impact on performance. The contents of the convolution function were directly integrated into the loop to remove the overhead of function calls, thereby improving the speed.

The third is the optimization of the memory access pattern. In filter_baseline, since pixels are processed in units of columns, memory access may not be continuous. In language C, since the arrays store row elements in consecutive memory positions, the memory access pattern may be optimized by processing pixels in units of rows. Continuous memory access increases cache hit rate, thereby improving speed.

The fourth is clamping optimization. In filter_baseline, clamping is processed inside the convolution function. In the optimized code, if the value is less than 0 for each channel, it is set to 0, and if it is greater than 255, it is set to 255, which simplifies the conditional statement by using overlapping three-term operators.

Finally, the calculation was simplified and the repetition statement was optimized. In filter_baseline, complexity increases due to unnecessary memory allocation and function calls. Therefore, by performing the calculation directly, unnecessary repetition statements and function

calls were reduced and optimized to perform only the necessary calculations inside the repetition statement.

Using the above strategy, the filter_optimized function significantly improved overall performance by removing memory allocation and release, reducing the overhead of function calls, and optimizing memory access patterns.