1 Implementation Result

The speedup results for various image sizes using my implementation are shown below. I take total three trial for get average result and the table contain this information. The result on "1024px" was had the outstanding average speed as shown both graph and table. As the image size increases, it requires more computational operations. I optimized the computational aspects, so the image with the largest number of operations(the 1024-pixel image) showed the greatest speedup.

Image Size(px)	128	256	512	768	1024
Try 1	1.631947	1.639187	1.593537	1.610369	1.993225
Try 2	1.624434	1.646357	1.595878	1.604302	1.91233
Try 3	1.638228	1.636347	1.60679	1.598054	1.935222
Average speedup	**1.63**	**1.64**	**1.59**	**1.60**	**1.94**

Table 1: Speedup Result for various image size

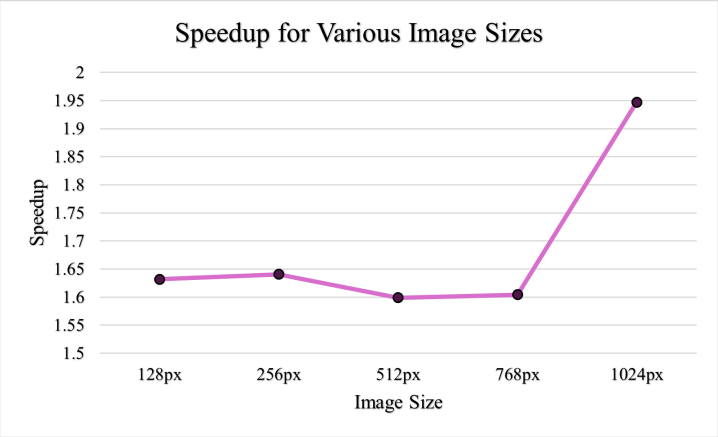

Figure 1: Speedup Graph for various image size

2 Optimization approach

2.1 Successful Strategies

I used three strategies to implement efficient optimization program. (1) loop unrolling (2) removing repeat computations (3) Batch Processing

1. Loop unrolling
   In convolution function, the code was structured with nested for loops, requiring branch instructions for execution due to heavy loads such as accessing structures within the loops. Anticipating high computational demands in this area, I resolved the issue by eliminating the nested loops and focusing solely on computations between input and filters. However, removing all loops posed challenges in handling edge cases. So I implemented a loop structure with some boundary conditions to run in all cases.

2. Removing repeat computations

   Original code has many repetitive computation. So I made variables to avoid repetitive computation which has the same number of computations like below.

```
    r += input[(x+dx)+(y+dy)*width].r * filter[(dx+1)+(dy+1)*3];
    g += input[(x+dx)+(y+dy)*width].g * filter[(dx+1)+(dy+1)*3];
    b += input[(x+dx)+(y+dy)*width].b * filter[(dx+1)+(dy+1)*3];
}
```

Figure 2: The repetitive computation

3. Block Matrix Multiplication

   I partitioned the given image into blocks of a specified size, computing smaller segments first and then aggregating them to expedite the overall calculation process. This approach reduces computational workload. Throughout these optimizations, the achieved speedup was measured with a block size set to 128.

2.2 Failed Strategies

1. inline strategy

   I used `inline` function to allow the compiler to interpret the function directly. But it was not worked. There was no change by using this function. So I copy and pasted the code in convolution to lower overhead for calling other function.