EEE4120F Practical 1

1st Joab Kloppers
KLPJOA002

2nd Alex Hillman
HLLALE010

I. Introduction

A common method used in digital image processing is edge detection. Edge detection techniques are often used to identify the boundaries between objects seen by a camera. This is what allows image processing systems to identify objects. A common way to perform edge detection is by using 2-dimensional convolution, whereby each pixel is convolved with its neighbours to detect the the differences between pixel intensities. Specifically, the use of the Sobel edge detector, which performs a 2-dimensional spatial gradient measurement to emphasize the large pixel intensity differences.

The Sobel detection process provides a reliable benchmarking method, since the operation requires the gradient operation to be performed for every pixel in an image, hence the operation is of the order O(n^2). This allows for a vast range of testing cases since a small change in the input image scales the operational complexity dramatically.

This practical report discuses the process of benchmarking a manually implemented 2-dimensional convolution algorithm, against a pre-existing implementation. The program is written in matalab which offers a wide array of matrix operations which are useful when performing convolutions.

II. Methodology

The benchmark consisted of two separate implementations of the Sobel edge detection algorithm:

1) A manual convolution implementation.
2) An implementation using MATLAB's built-in conv2 function.

The Sobel operator uses two kernels to compute the horizontal and vertical gradients of the input image. These kernels are defined as:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\mathrm{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The manual implementation involved iterating over each pixel of the image and computing the convolution sum explicitly using nested loops. For each pixel location, the corresponding neighborhood of the image was multiplied elementwise with the Sobel kernel, and the resulting values were summed to produce the gradient response at that location.

Both the horizontal gradient G_x and vertical gradient G_y responses were computed independently. The overall gradient magnitude of the image was then calculated using:

$$G = \sqrt{G_x^2 + G_y^2}$$

This resulting gradient magnitude image represents the detected edges.

For benchmarking purposes, the same grayscale input image and Sobel kernels were used for both implementations. Execution time for each approach was measured using MATLAB's timing functions tic and toc. Multiple runs were performed to reduce variability and obtain consistent measurements. The average execution time of the manual implementation was then compared to that of the built-in conv2 implementation to determine the performance difference.

III. Results

The banchmarking operation was done using five image sizes, each benchmark was performed multiple times and an average was taken. The results from the benchmark are shown in Table I.

Image Size	Manual	Inbuilt	Speedup
128x128	0.033773	0.000289	116.8616
256x256	0.119097	0.005593	21.29394
512x512	1.887973	0.005593	337.56
1024x1024	6.262282	0.007721	811.0714
2048x2048	7.682941	0.035491	216.4758

TABLE I
Results from Benchmarking

The speedup of the inbuilt with respect to the manually implemented function is determined using the following:

$$Speedup = \frac{T_{manual}}{T_{inbuilt}}$$

The data shows that the the inbuilt function is able to able to perform on average 300 times faster than the manually implemented method.

A graph depicting the method speed up is shown in Figure 1.

IV. Discussion

The results shown in Table I clearly indicate a significant performance difference between the manual convolution implementation and MATLAB's built-in conv2 function. The manual implementation relies on nested loops to compute convolution values for each pixel individually, which introduces a large computational overhead, particularly for larger images.

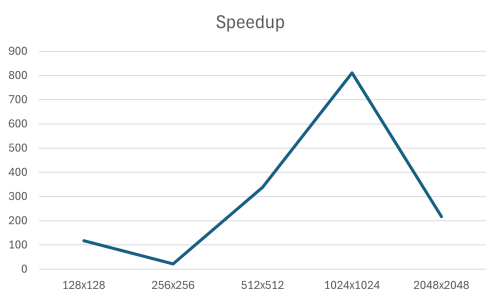

Fig. 1. Graph depicting Speed Up for Each Image Size

In contrast, the built-in conv2 function is internally optimized and implemented using low-level compiled code. This allows MATLAB to take advantage of efficient memory access patterns and hardware-level optimizations that are not accessible when using high-level interpreted loops in MATLAB scripts.

There is some discrepancy with the speed up values for the image sizes, specifically at an image size of 256x256 and 1024x1024. At an image size of 256x256, the achieved speed up from using the inbuilt function was much lower than the average over all the runs, this indicates that the inbuilt function may not be optimised for this size of image. In contrast, the inuilt function performed significantly faster than the manual implementation for the 1024x1024 image size, indicating that this size is highly correlated to the optimisation used by the inbuilt function.

V. Conclusion

In this report, a manual implementation of the Sobel edge detection algorithm was benchmarked against MATLAB's built-in conv2 convolution function. The manual approach involved explicitly computing the convolution operation using nested loops, while the built-in implementation utilized MATLAB's optimized internal routines.

The benchmarking results demonstrated that the built-in convolution function significantly outperforms the manual implementation in terms of execution time. This performance difference can be attributed to MATLAB's internal optimizations and the inefficiency of loop-based operations in interpreted environments.

Overall, while manual convolution is useful for educational purposes and understanding algorithmic behavior, built-in functions such as conv2 should be preferred in practical applications where computational performance is critical.

References