Enhancer for Scanned Images

Algorithm Engineering 2023 Project Paper

Sheela Orgler
Friedrich Schiller University Jena
Germany
sheela.orgler@uni-jena.de

ABSTRACT

The five-finger pattern [1]:

(1) **Topic and background:** What topic does the paper deal with? What is the point of departure for your research? Why are you studying this now?
(2) **Focus:** What is your research question? What are you studying precisely?
(3) **Method:** What did you do?
(4) **Key findings:** What did you discover?
(5) **Conclusions or implications:** What do these findings mean? What broader issues do they speak to?

KEYWORDS

enhancer, scanned images, image processing, median filter, adaptive thresholding, C++, CMake

1 INTRODUCTION

This paper introduces an enhancer for scanned images. The enhancer is part of a university project. It is written in C++ and uses several techniques to improve the quality of the images. Furthermore, methods are used to improve the perfomance of the program like vectorization, parallelization and cache-friendly code.

1.1 Background

The improvement of scanned images revolves around the topic of image processing, which involves enhancing the quality of digital images obtained from scanned documents or photographs. The goal is to improve the quality of the image and make it easier to read. One of the main challenges when enhancing scanned images is the presence of various disruptive elements, such as noise, blur, distortion, and color cast. Several techniques have been developed, to address these issues. Two of these are noise reduction and adaptive thresholding.

1.2 Related Work

Scanned documents are an important source of information for many applications, ranging from historical archives to legal and financial documents. However, the quality of scanned documents is often compromised due to various factors such as poor scanning conditions, aging of the original documents, and degradation during storage. To address these issues, several image processing techniques have been proposed for enhancing the quality of scanned document images.

One common problem in scanned document images is the presence of noise and artifacts, which can reduce the readability of the text and images. To mitigate this issue, several noise reduction and image restoration techniques have been proposed, including median filtering, wavelet denoising, and deblurring.

For this project several existing projects and papers have been evaluated. Specifically, regarding adaptive thresholding there are several methods that can be used. The paper Adaptive Thresholding: A comparative study from the International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT) has evaluated four different methods and compared them to each others. With adaptive thresholding a threshold is computed, that changes dynamically over the image. The paper discusses the differences between these methods pointing out there advantages as well as their disadvantages.

1.3 Our Contributions

This project uses two methods combined to improve the quality of scanned documents. First, a median filter is used for noise reduction. Then, an adaptive thresholding method is applied to dynamically improve the brightness of the image.

1.4 Outline

2 THE ALGORITHM

2.1 Input and Output

The algorithm uses ppm images as input and output. PPM (Portable Pixel Map) is a file format used to store digital images. It is a simple and flexible format that is commonly used in computer graphics and image processing. PPM images are uncompressed and consist of a plain text header followed by the binary pixel data. The header contains metadata about the image, such as the image width, height, and maximum pixel value. There are two variants of PPM: PPM P3 and PPM P6. PPM P3 is the ASCII version of the format, where the pixel values are represented as ASCII characters. PPM P6 is the binary version of the format, where the pixel values are represented as binary data. To interact with the program a terminal application is provided. After executing the program one can input the desired path of the image that should be processed. The program accepts images in the ppm format described previously.

2.2 Median filter

A median filter is a method used for image processing to remove noise from an image. The filter works by sliding a window over the image and replacing the center pixel of the window with the median value of the neighboring pixels. Unlike linear filters that take the weighted average of the pixels, the median filter simply selects the middle value of the pixel values within the window. The median filter is effective in removing impulse noise, which is random noise that manifests as bright or dark pixels that are not present in the original image. It is particularly useful in preserving

sharp edges and fine details in an image, as it does not blur the edges like linear filters can.

2.3 Adaptive thresholding

Furthermore, to improve the lighting conditions of the image adaptive thresholding is used. Adaptive thresholding is a technique used in image processing to separate the foreground from the background of an image. It is commonly used when the lighting conditions or contrast within an image vary across different regions. The algorithm works by dividing the image into small regions and computing a threshold value for each region based on its local statistics, such as the mean or median pixel value. Pixels within a region that have a value above the threshold are considered part of the foreground, while pixels with a value below the threshold are considered part of the background.

2.4 Performance improvements

Several techniques are used to improve the performance of the algorithm.

OpenMP for parallelization. For parallelization the directives and functions defined by OpenMP (Open Multi-Processing) are used. In OpenMP, parallel regions can be defined for concurrent execution of specific code sections. Specifically when using for-loops, the directive #pragma omp parallel for can be used to parallelize a loop.

OpenMP for performance measurements. To analyze the performance of the written code, the function omp_get_wtime() is used. When used at two different points, the difference can be computed to get the execution time. This was used in the project, to identify bottlenecks and the effects of specific perfmance imrpovement methods.

2.5 Internal Representation of Mock Labels

In Figure 1 we convert the mock labels to sorted integer sets.

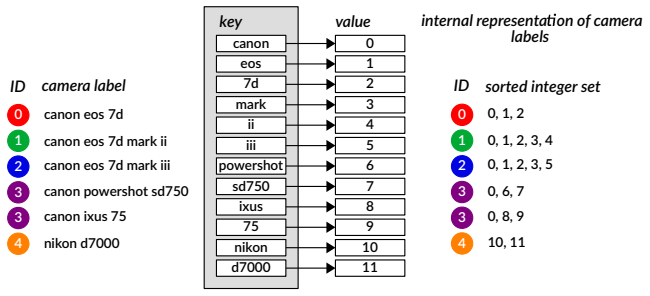

Figure 1: Conversion of mock camera labels to sorted integer sets. We map each unique token (key) in camera labels to a unique value. Based on these key-value-mappings, we convert camera labels to sorted integer sets. A camera can have different names in different countries. Therefore, repeating IDs reference the same cameras (see, for example, ID=3).

2.6 Efficient Preprocessing of Input Data

The following findings are important to speed up preprocessing of the input data:

- Reading many small files concurrently, with multiple threads (compared to a single thread), takes advantage of the internal parallelism of SSDs and thus leads to higher throughput [2].
- C-string manipulation functions are often significantly faster than their C++ pendants. For example, locating substrings with `strstr` is around five times faster than using the C++ `std::string` function `find`.
- Hardcoding regular expressions with *while, for, switch* or *if-else* statements results in faster execution times than using standard RegEx libraries, where regular expressions are compiled at runtime into state machines.
- Changing strings in place, instead of treating them as immutable objects, eliminates allocation and copying overhead.

3 EXPERIMENTS

Table 1 shows the running times of the resolution step of the five best placed teams.

Table 1: Comparison of the F-measure and the running times of the resolution step of the five best placed teams. The input data for the resolution step consisted of 29,787 in JSON formatted e-commerce websites. Measurements were taken on a laptop running Ubuntu 19.04 with 16 GB of RAM and two Intel Core i5-4310U CPUs. The underlying SSD was a 500 GB 860 EVO mSATA. We cleared the page cache, dentries, and inodes before each run to avoid reading the input data from RAM instead of the SSD.

Team	Language	F-measure	Running time (s)
PictureMe (**this paper**)	C++	0.99	**0.61**
DBGroup@UniMoRe	Python	0.99	10.65
DBGroup@SUSTech	C++	0.99	22.13
eats_shoots_and_leaves	Python	0.99	28.66
DBTHU	Python	0.99	63.21

4 CONCLUSIONS

REFERENCES

[1] Felicitas Macgilchrist. 2014. *Academic writing.* UTB.
[2] Zhenyun Zhuang, Sergiy Zhuk, Haricharan Ramachandra, and Badri Sridharan. 2016. Designing SSD-Friendly Applications for Better Application Performance and Higher IO Efficiency. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC).* IEEE. https://doi.org/10.1109/compsac.2016.94