Visual Recognition February 4, 2024

Visual Recognition Assignment 1

Instructor: Prof. Dinesh Babu J Goutham U R (IMT2021045)

1 Introduction

1.1 Project Objectives

This report presents the outcomes of a virtual reality (VR) project aimed at tackling two fundamental tasks:

- To build a model capable of counting the number of books in an image.
- To remove shadow regions from an image and inpaint them.

2 Task 1: Counting Books in an Image

2.1 Method 1

2.1.1 Hough Transform

The Hough transform is a technique used in image processing to detect simple geometric shapes, such as lines, circles, and ellipses, within an image. It is particularly useful for tasks like object detection and feature extraction. In the context of this project, I implemented the Hough Circle Transform to detect circular shapes representing books in the given image.

2.1.2 Implementation

The provided Python code utilizes the OpenCV library for image processing tasks. The `count_books_with_parameters` function takes an image path and several parameters for the Canny edge detection and Hough Circle Transform. It returns the count of books detected in the image along with the parameters used for the detection.

2.1.3 Libraries Used

The implementation of the book counting task relies on the following libraries:

- **NumPy**: Used for numerical operations and array manipulations.
- **OpenCV (cv2)**: Utilized for image processing tasks, including edge detection and Hough Circle Transform.

2.1.4 Inbuilt functions used

For the book counting task, the major functions utilized from the OpenCV library include:

- `cv2.imread`: Used to read the input image.
- `cv2.cvtColor`: Utilized for converting the image to grayscale.
- `cv2.Canny`: Applied for edge detection using the Canny algorithm.
- `cv2.HoughCircles`: Employed to perform the Hough Circle Transform for circle detection.

2.1.5 Hyperparameter Tuning

To achieve accurate book counting, a wide range of hyperparameters were chosen for the Canny edge detection and Hough Circle Transform. The combinations that yielded the best results with the training image were selected for testing. Hyperparameters such as `canny_minVal`, `canny_maxVal`, `hough_dp`, `hough_minDist`, `hough_param1`, `hough_param2`, `hough_minRadius`, and `hough_maxRadius` were varied to optimize the detection performance.

2.1.6 Training

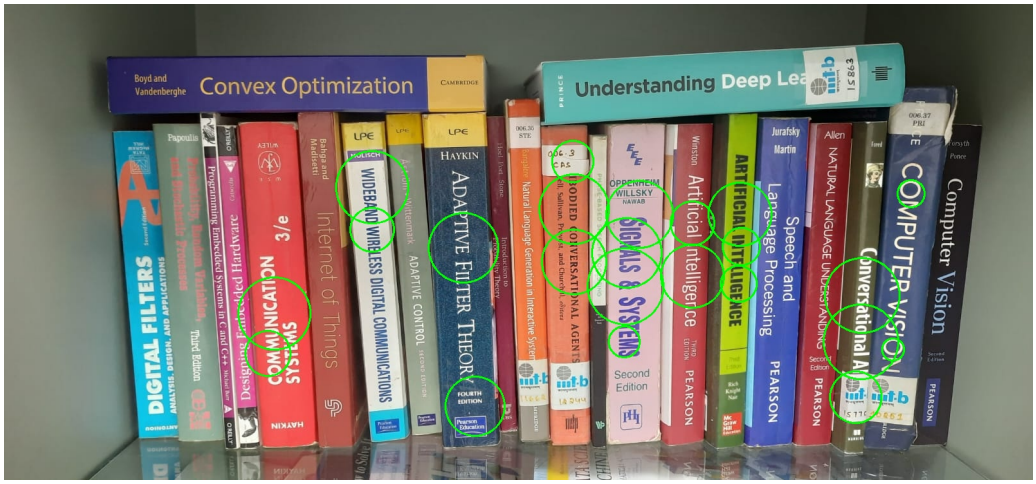
Hyperparameters that resulted in a book count within the range of 20 to 26 were stored in a list and were used for training the model. The original count of the number of books in the training image is 23. Instead of choosing the exact count, a small range around 23 was selected to avoid overfitting and ensure the model's generalization ability.

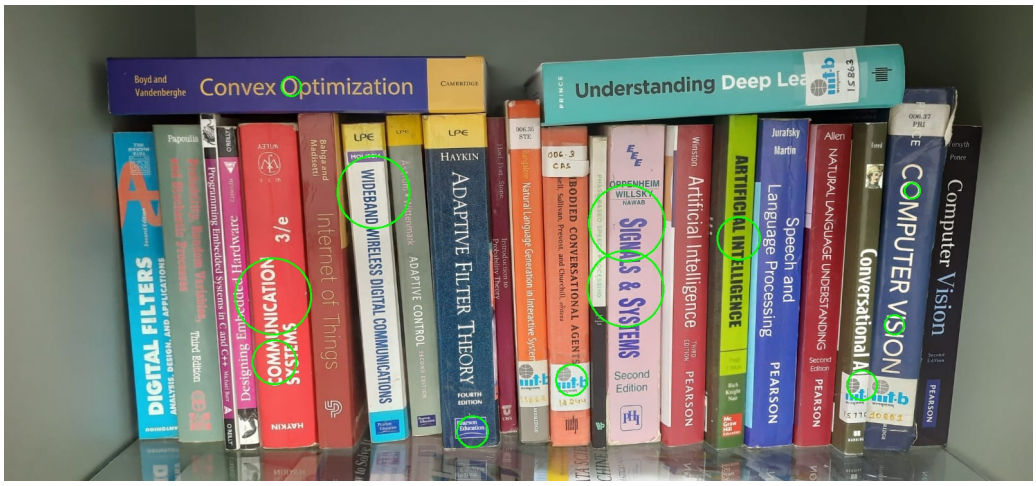
2.1.7 Testing and Results

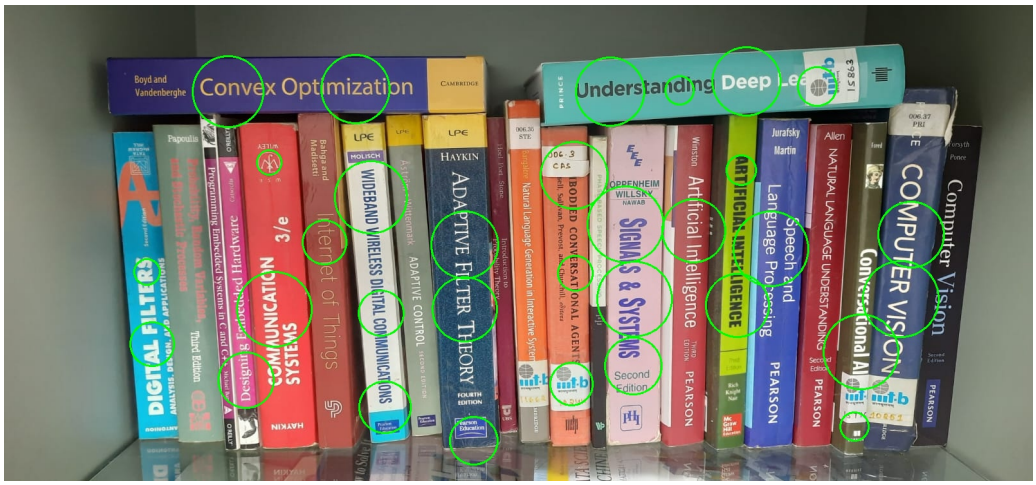
After training the model with the selected hyperparameters, testing was conducted to evaluate its performance on new images. The closest result obtained during testing was a book count of 15, which was achieved by multiple sets of hyperparameters. The sets are listed below:

- [80, 150, 1, 100, 100, 30, 15, 50]
- [80, 150, 1, 100, 150, 30, 15, 50]
- [80, 150, 1, 100, 200, 30, 15, 50]
- [100, 150, 1, 50, 100, 40, 10, 60]
- [100, 150, 1, 50, 150, 40, 10, 60]
- [100, 150, 1, 50, 200, 40, 10, 60]
- [100, 150, 1, 100, 100, 30, 15, 50]
- [100, 150, 1, 100, 150, 30, 15, 50]
- [100, 150, 1, 100, 200, 30, 15, 50]

2.2 Images with Hough Circles




Boyd and Vandenberghe
Convex Optimization
CAMBRIDGE
Understanding Deep Lea
DIGITAL FILTERS
ANALYSIS, DESIGN, AND APPLICATIONS
Papoulis
Third Edition
Designing Embedded Hardware
3/e
COMMUNICATION SYSTEMS
HAYKIN
Internet of Things
WIDEBAND WIRELESS DIGITAL COMMUNICATIONS
ADAPTIVE CONTROL
HAYKIN
ADAPTIVE FILTER THEORY
FOURTH EDITION
Natural Language Generation in Interactive Systems
BODIED CONVERSATIONAL AGENTS
OPPENHEIM
WILLSKY
NAWAB
SIGNALS & SYSTEMS
Second Edition
Artificial Intelligence
PEARSON
ARTIFICIAL INTELLIGENCE
Jurafsky
Martin
Speech and Language Processing
PEARSON
Allen
NATURAL LANGUAGE UNDERSTANDING
Second Edition
PEARSON
Conversational A
COMPUTER VISION
Computer Vision
PEARSON


Boyd and Vandenberghe
Convex Optimization
CAMBRIDGE
Understanding Deep Lea
DIGITAL FILTERS
ANALYSIS, DESIGN, AND APPLICATIONS
Papoulis
Third Edition
Designing Embedded Hardware
3/e
COMMUNICATION SYSTEMS
HAYKIN
Internet of Things
WIDEBAND WIRELESS DIGITAL COMMUNICATIONS
ADAPTIVE CONTROL
HAYKIN
ADAPTIVE FILTER THEORY
FOURTH EDITION
Natural Language Generation in Interactive Systems
BODIED CONVERSATIONAL AGENTS
OPPENHEIM
WILLSKY
NAWAB
SIGNALS & SYSTEMS
Second Edition
Artificial Intelligence
PEARSON
ARTIFICIAL INTELLIGENCE
Jurafsky
Martin
Speech and Language Processing
PEARSON
Allen
NATURAL LANGUAGE UNDERSTANDING
Second Edition
PEARSON
Conversational A
COMPUTER VISION
Computer Vision
PEARSON

2.3 Method 2

2.3.1 Overview

In Method 2, I utilized the combination of Canny edge detection and contour detection techniques to count the number of books in an image. Canny edge detection helps in identifying the edges of objects within the image, while contour detection outlines the boundaries of these objects. By detecting contours corresponding to the shapes of books, I can determine their count.

2.3.2 Libraries Used

The implementation of Method 2 relies on the following libraries:

- **OpenCV (cv2)**: Utilized for image processing tasks, including edge detection and contour detection.
- **os**: Used for operating system related tasks, such as file path manipulation.

2.3.3 Inbuilt Functions Used

For the contour detection task, the major inbuilt functions utilized from the OpenCV library include:

- `cv2.imread`: Used to read the input image.
- `cv2.cvtColor`: Utilized for converting the image to grayscale.
- `cv2.Canny`: Applied for edge detection using the Canny algorithm.
- `cv2.findContours`: Employed to find contours in the edge-detected image.
- `cv2.drawContours`: Utilized to draw the detected contours on the original image.
- `cv2.imwrite`: Used to save the image with contours drawn to the output directory.

2.3.4 Training

In the training phase of Method 2, I set a minimum perimeter threshold, and only contours with perimeters exceeding this threshold were considered for counting. By setting a minimum perimeter, I aimed to filter out smaller contours that may not represent complete books, thus improving the accuracy of the book counting algorithm.

A wide range of values was chosen for the minimum perimeter parameter to explore various contour sizes and optimize the detection performance. Instead of choosing an exact number, a range was selected to ensure flexibility and robustness in handling different book sizes and variations in image quality. This approach helps prevent overfitting to specific contour sizes and enhances the model's generalization ability across diverse images.

The minimum perimeter thresholds obtained for the range 20 to 26 were:

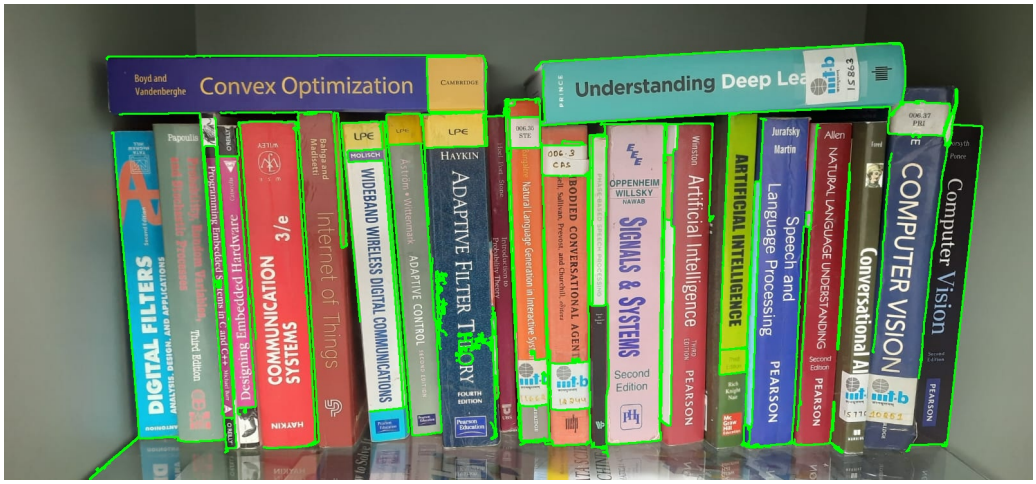
- Min Perimeter: **600**, Book Count: 26
- Min Perimeter: **650**, Book Count: 25
- Min Perimeter: **700**, Book Count: 24
- Min Perimeter: **750**, Book Count: 21
- Min Perimeter: **800**, Book Count: 20
- Min Perimeter: **850**, Book Count: 20

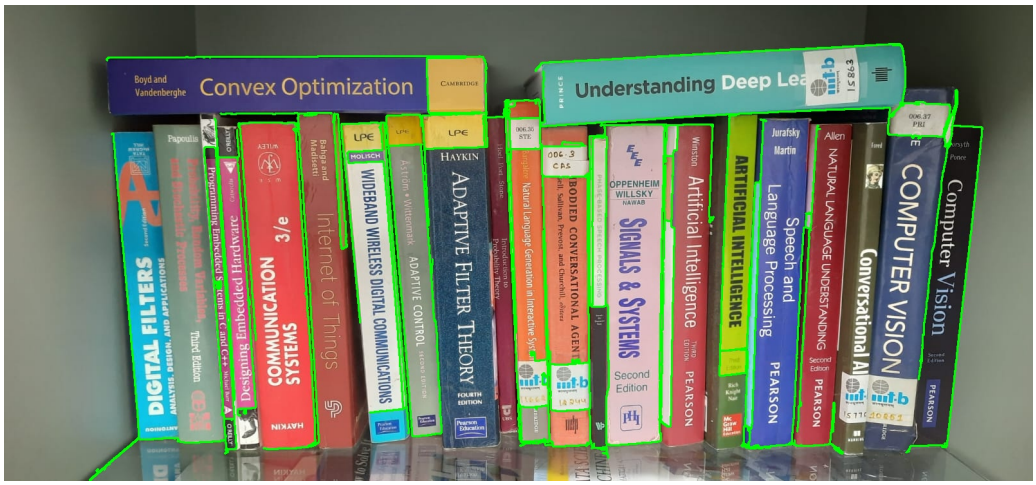
2.3.5 Testing and Results

The trained model was tested on a separate test image to evaluate its performance in real-world scenarios. The test image was processed using Method 2 with different minimum perimeter thresholds. The results obtained from testing the model on the test image are as follows:

- For min perimeter **600**, the book count is: 17
- For min perimeter **650**, the book count is: 17
- For min perimeter **700**, the book count is: 17
- For min perimeter **750**, the book count is: 16
- For min perimeter **800**, the book count is: 16
- For min perimeter **850**, the book count is: 13

2.4 Images with Contours





3 Task 2: Removing Shadows from an Image

3.0.1 Overview

In Task 2, I aimed to develop a method to automatically detect and inpaint shadows present in images. The proposed approach utilizes techniques such as Gaussian blur, thresholding, and inpainting to effectively identify

and remove shadows from images.

3.0.2 Libraries Used

The implementation of Task 2 relies on the following libraries:

- **OpenCV (cv2)**: Utilized for image processing tasks, including resizing, color conversion, Gaussian blur, thresholding, and inpainting.
- **NumPy**: Used for array manipulation and operations.
- **os**: Utilized for file and directory manipulation tasks.

3.0.3 Inbuilt Functions Used

For shadow removal, the major inbuilt functions utilized from the OpenCV library include:

- `cv2.imread`: Used to read the input image.
- `cv2.resize`: Utilized for resizing the image.
- `cv2.cvtColor`: Applied for converting the image to grayscale.
- `cv2.GaussianBlur`: Used to reduce noise and improve shadow detection.
- `cv2.threshold`: Employed for thresholding to obtain a binary mask of shadows.
- `cv2.bitwise_and`: Utilized for combining masks to preserve certain regions.
- `cv2.inpaint`: Applied for inpainting to replace shadow regions with information from surrounding areas.
- `cv2.imshow`, `cv2.waitKey`, `cv2.destroyAllWindows`: Used for displaying images and managing GUI windows.
- `cv2.imwrite`: Employed to save the inpainted images.

3.0.4 Implementation

1. `resized_image = cv2.resize(original_image, None, fx=resize_factor, fy=resize_factor)`
   Resizing the original image using OpenCV (`cv2`) with a scaling factor `resize_factor` for both width and height.
2. `gray_image = cv2.cvtColor(resized_image, cv2.COLOR_BGR2GRAY)`
   Converting the resized image to grayscale using the OpenCV function `cvtColor`.
3. `blurred_image = cv2.GaussianBlur(gray_image, (5, 5), 0)`
   Applying a Gaussian blur to the grayscale image to reduce noise.
4. `preserve_mask = np.ones_like(gray_image) * 255`
   `preserve_mask[100:300, :] = 0`
   Creating a mask (`preserve_mask`) with the same dimensions as the grayscale image, setting a specific region (e.g., trees and footpaths) to zero.
5. `final_mask = cv2.bitwise_and(blurred_image, preserve_mask)`
   Combining the blurred image and the preserve mask using bitwise AND operation.
6. `inpainted_image = cv2.inpaint(resized_image, final_mask, inpaintRadius=5, flags=cv2.INPAINT_TELEA)`
   Performing inpainting on the resized image, replacing shadow regions with information from surrounding areas using the final mask.

Figure 1: Resized image

Figure 2: Shadow mask

Figure 3: Inpainted image

Figure 4: Image 1

Figure 5: Resized image

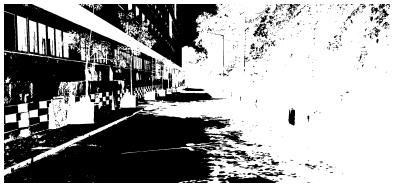


Figure 6: Shadow mask

Figure 7: Inpainted image

Figure 8: Image 2