Image Processing Pipeline using OpenCV

February 23, 2025

1 Introduction

Image processing plays a crucial role in computer vision tasks, where preprocessing is essential for noise reduction and feature enhancement. This report details an OpenCV-based pipeline implementing grayscale conversion, Gaussian smoothing, morphological transformations, and edge detection, followed by image segmentation and coin counting.

2 Methodology

2.1 Loading and Converting Image to Grayscale

The first step in the pipeline involves reading an image and converting it to grayscale. This is achieved using OpenCV's `cv2.imread()` and `cv2.cvtColor()` functions:

```
image = cv2.imread('images/one.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Significance: Grayscale conversion simplifies computation by reducing three color channels (RGB) into a single intensity channel while retaining essential information.

Figure 1: Your caption here

2.2 Gaussian Smoothing for Noise Reduction

To reduce noise, we apply Gaussian blur using the `cv2.GaussianBlur()` function:

```
blur = cv2.GaussianBlur(gray, (11,11), 0)
```

Hyperparameter Tuning:

- Kernel size: Experimented with (5,5), (9,9), and (11,11). Larger kernel sizes increased smoothing at the cost of feature blurring. The optimal choice was (11,11).
- Standard deviation: A value of 0 lets OpenCV determine it automatically based on kernel size.

Alternative Approach: Median filtering (`cv2.medianBlur()`) could be used for preserving edges while reducing noise.

Figure 2: Smoothed Grayscaled image

2.3 Morphological Opening for Noise Removal

Morphological transformations, specifically opening (erosion followed by dilation), help remove small noise. This is implemented using `cv2.morphologyEx()`:

```
kernel = np.ones((15,15), np.uint8)
opened = cv2.morphologyEx(blur, cv2.MORPH_OPEN, kernel, iterations=2)
```

Hyperparameter Tuning:

- Kernel size: A (15,15) kernel provided sufficient noise removal without overly distorting structures.
- Iterations: Tested with 1, 2, and 3 iterations; 2 iterations were optimal for eliminating noise without excessive erosion.

if we don't use Morphological Opening for Noise Removal we will get such image, I removed text edges because we don't need text for counting coin.

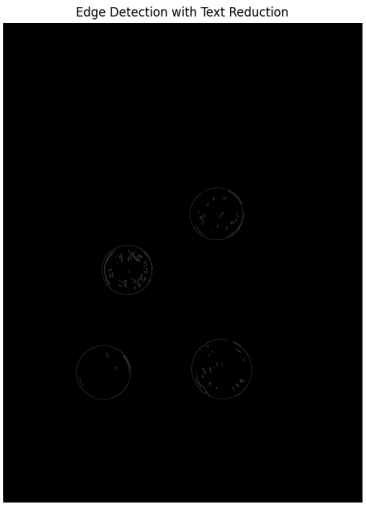

Figure 3: Your caption here

2.4 Edge Detection Using Canny

Edge detection highlights object boundaries using the `cv2.Canny()` function:

```
canny = cv2.Canny(opened, 50, 90)
```

Hyperparameter Tuning:

- Lower threshold: Experimented with 30, 50, and 70. A value of 50 provided better noise suppression.
- Upper threshold: Tested 80, 90, and 100. A value of 90 retained sufficient edges without excessive noise.

Alternative Approach: Sobel or Laplacian operators were also used for edge detection, but Canny provided better control over noise.

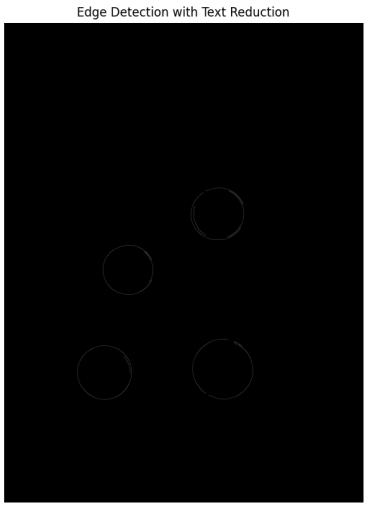

Figure 4: Your caption here

2.5 Segmentation Using Thresholding

Segmentation is performed using thresholding to separate foreground from background:

```
ret, thresh = cv2.threshold(opened, 170, 255, cv2.THRESH_BINARY)
```

Hyperparameter Tuning:

- Threshold value: Tested values 150, 170, and 190. A value of 170 provided the best separation without losing smaller details.

Alternative Approach: Adaptive thresholding (`cv2.adaptiveThreshold()`) could be used for non-uniform lighting conditions.

2.6 Coin Counting Using Contours

Contours are detected and filtered based on area to count the number of coins:

```
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
area = {}
for i in range(len(contours)):
```

```
    cnt = contours[i]
    ar = cv2.contourArea(cnt)
    area[i] = ar
srt = sorted(area.items(), key=lambda x: x[1], reverse=True)
results = np.array(srt).astype("int")
num = np.argwhere(results[:, 1] > 500).shape[0]
for i in range(1, num):
    cv2.drawContours(image_copy, contours, results[i, 0], (0, 255, 0), 3)
print("Number of coins is", num - 1)
```

Significance: Sorting by area allows us to filter out small noise and count only significant objects.

Alternative Approach: Watershed segmentation could improve separation if coins are touching.

2.7 Coin Counting Results

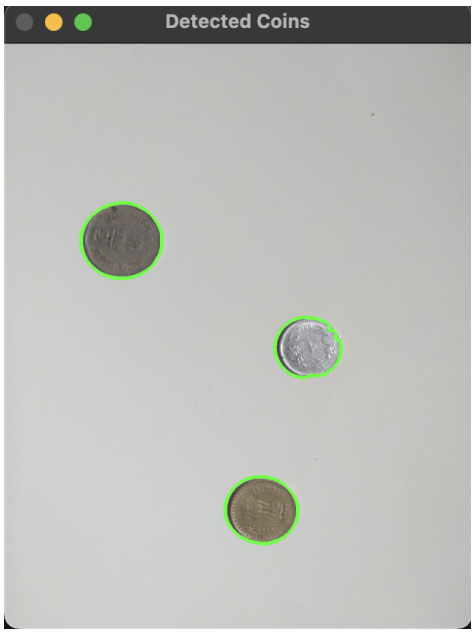

Figure 5: Result of Coin Counting - Image 1

Figure 6: Result of Coin Counting - Image 2

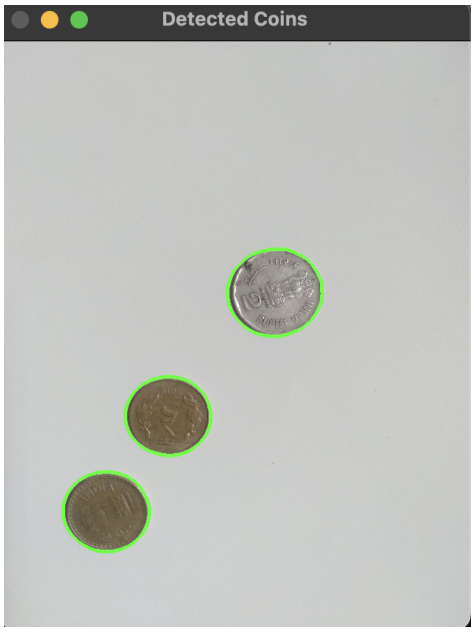

Figure 7: Result of Coin Counting - Image 3

Figure 8: Result of Coin Counting - Image 4

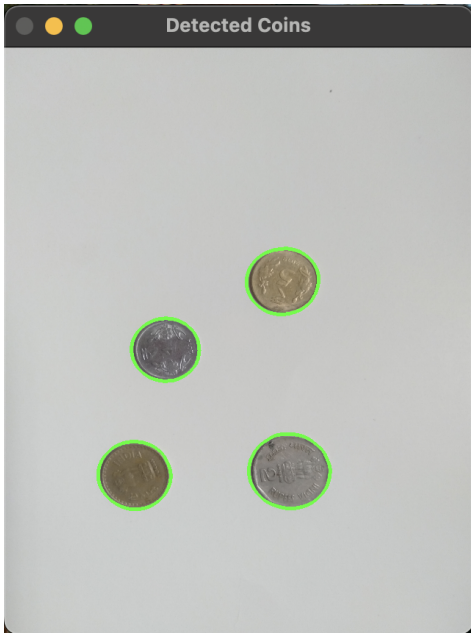

Figure 9: Result of Coin Counting - Image 5

The count of the number of images is shown on the terminal. **example :**

Figure 10: Result of Coin Counting - Image 3

3 Problem with the Current Algorithm

This method fails when coins overlap because:

- The contour detection method treats overlapping objects as a single entity.
- The morphological operations and thresholding cannot effectively separate touching coins.
- As a result, the algorithm underestimates the number of coins by grouping multiple coins into a single contour.

4 Solution: Watershed Algorithm

To accurately segment overlapping coins, we can use the Watershed algorithm, which works as follows:

1. Convert the image to grayscale and apply thresholding.
2. Perform distance transform to highlight the centers of objects.
3. Apply morphological operations to obtain sure foreground and background regions.
4. Compute markers and apply the Watershed algorithm to separate touching objects.
5. Extract individual coin contours for accurate counting.

The Watershed algorithm effectively splits overlapping objects by treating the image as a topographical map and flooding regions from markers. This ensures that touching coins are correctly segmented and counted individually.

5 Thresholding Algorithm Image

Figure 11: Example of Overlapping Coin Detection, we get wrong answer using the thresholding algorithm

6 Conclusion

This efficiently preprocesses an image by applying grayscale conversion, smoothing, morphological operations, segmentation, and contour-based coin counting. Careful hyperparameter tuning optimized noise reduction while preserving features. Alternative approaches such as adaptive thresholding and watershed segmentation could be explored based on specific requirements.