**HCMUS - VNUHCM / FIT / Computer Vision & Cognitive Cybernetics Department**

**Digital Image and Video Processing Application**

**Student ID:** 21127690

**Student name:** Ngo Nguyen Thanh Thanh

**Report: Morphological Operations** (operators up to week of 14 Feb 2025)

**I. Evaluation summary:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **No** | **Task** | **Implementation (Without OpenCV)** | **Time Complexity (Without OpenCV)** | **Implementation (With OpenCV)** | **Time Complexity (With OpenCV)** | **Completion (%)** |
| **1** | **Binary Dilation** | Implemented using manual convolution and max filter. | O(n2⋅k2) (for kernel size 𝑘) | Uses cv2.dilate() with a structuring element. | O(n2) (optimized) | 100% |
| **2** | **Binary Erosion** | Implemented using manual convolution and min filter. | O(n2⋅k2) (for kernel size 𝑘) | Uses cv2.morphologyEx(..., cv2.MORPH\_OPEN). | O(n2) | 100% |
| **3** | **Binary Opening** | Combines manual erosion and dilation sequentially | O(n2⋅k2) (for kernel size 𝑘) | Uses cv2.morphologyEx(..., cv2.MORPH\_OPEN). | O(n2) (optimized) | 100% |
| **4** | **Binary Closing** | Combines manual dilation and erosion sequentially. | O(n2⋅k2) (for kernel size 𝑘) | Uses cv2.morphologyEx(..., cv2.MORPH\_CLOSE). | O(n2) | 100% |
| **5** | **Hit-or-Miss** | Implemented using two structuring elements and logical operations. | O(n2⋅k2) (for kernel size 𝑘) | Uses cv2.morphologyEx(..., cv2.MORPH\_HITMISS). | O(n2) | 100% |
| **6** | **Boundary Extraction** | Subtracts eroded image from the original manually. | O(n2⋅k2) (for kernel size 𝑘) | Uses cv2.subtract(image, cv2.erode(image, kernel)). | O(n2) | 100% |
| **7** | **Region Filling** | Uses iterative processing with a seed-based algorithm. | O(n2) to (n3) (depending on number of iterations) | Uses OpenCV flood-fill (cv2.floodFill()). | O(n2) (optimized) | 100% |

1. **List of features and file structure:**

**File structure:**

**A screenshot of a computer

Description automatically generated**

* **morphological\_operations/**: A package containing functions for morphological operations.
* **\_\_init\_\_.py**: Defines morphological\_operations as a package, allowing imports.
* **binary.py**: Implements image processing functions.
* **main.py** (outside the package): The main script that likely imports binary.py and runs the program.
* **Image files (.png)**: Used as input or for testing.

This structure keeps functions modular (binary.py) and execution separate (main.py).

**Functions and methods used:**

**1. binary.py (Custom Binary Image Processing Library)**

Main functions:

* **pad\_image()**: Adds padding to the image to avoid errors during processing.
* **erode()**: Performs the erosion operation to shrink the bright areas.
* **dilate()**: Performs the dilation operation to expand the bright areas.
* **opening()**: Performs the opening operation (Erosion → Dilation) to remove small noise.
* **closing()**: Performs the closing operation (Dilation → Erosion) to fill small gaps.
* **hit\_or\_miss()**: Applies the Hit-or-Miss operation to detect specific shapes or patterns.
* **boundary\_extraction()**: Extracts the boundary of the bright regions.
* **region\_filling()**: Fills the bright regions based on dilation.

**2. main.py (Main Program)**

**Main functions:**

* Read the input image and convert it to a binary image.
* Provide two processing modes:
  + **Manual**: Use custom algorithms from **binary.py**.
  + **OpenCV**: Use the OpenCV library for processing.
* Support operations: Dilate, Erode, Open, Close, HitMiss, Boundary, Fill.
* Display and save the processed image.
* Measure and display the execution time of each method.

**Details:**

* **apply\_manual(img, kernel):** Applies custom morphological operations.
* **apply\_opencv(img, kernel):** Applies morphological operations using OpenCV.
* **operator(in\_file, out\_file, mor\_op, mode, wait\_key\_time=0):** Processes image with specified operation and mode, displays and saves results, measures execution time.
* **main(argv):** Parses command-line arguments and calls operator() to process the image.

**How to run code:**

**1. Install Required Libraries**

First, make sure you have the necessary libraries installed, such as opencv, numpy, and morphological\_operator. You can install them using pip:

**pip install opencv-python numpy**

**Note**: The morphological\_operator library appears to be a custom-written library, so make sure it exists in the same directory or has been correctly installed.

**2. Command Line Structure**

Here’s the command to run the program from the command line:

**python main.py -i <input\_file> -o <output\_file> [-p <morph\_operator>] -m <mode> -t <wait\_key\_time>**

**3. Explanation of Parameters:**

* -i <input\_file>: Path to the input image file.
* -o <output\_file>: Path to save the result.
* -p <morph\_operator> (Optional): The specific morphological operation you want to apply (e.g., "Dilate", "Erode", etc.).
* -m <mode>: The execution mode ("manual" for custom-written algorithms or "opencv" for OpenCV).
* -t <wait\_key\_time> (Optional): Time to wait (in milliseconds) before closing the window displaying the image.

**4. Example:**

Suppose you have an image file input.jpg and you want to apply the "Dilate" operation using the custom algorithm (manual mode) and save the result to output.jpg. You would run:

**python main.py -i input.jpg -o output.jpg -p Dilate -m manual**

**5. Options:**

* If you don't specify a morphological operation, the program will apply all available operations.
* If you don't specify a wait time, the program will not wait and will automatically close the window displaying the image.

**6. Error Information:**

If there's an error, such as a missing image file or incorrect parameter, the program will show an appropriate error message.

**Image proof**:

**Binary.py**

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Main.py**

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer code on a black background

Description automatically generated

A screen shot of a computer program

Description automatically generated

A computer screen shot of a computer

Description automatically generated

**III. Summarization of the usage**

**Detailed Usage and Algorithm Explanation:**

**1. pad\_image(img, kernel)**

* **Usage:** Adds padding to the image to prevent overflow errors when applying morphological operations. The padding size is determined by the kernel's shape.
* **Algorithm Explanation:**
  + The function calculates the padding required for height (pad\_h) and width (pad\_w) based on the kernel size.
  + It then uses np.pad() to add zero-padding to the image around the borders.
  + This ensures that the kernel fits within the image boundaries during operations.

**2. erode(img, kernel)**

**A math equations and symbols

Description automatically generated with medium confidence**

* **Usage:** Performs erosion on a binary image. Erosion shrinks bright regions by removing pixels at the borders of bright regions.
* **Algorithm Explanation:**
  + The function first pads the image to prevent boundary errors during erosion.
  + It then iterates over each pixel of the image.
  + For each pixel, it extracts a region equal to the kernel's size.
  + The region is compared with the kernel, and if it matches, the pixel is set to 1 in the result.
  + Otherwise, the pixel remains 0, thus shrinking bright areas.

**3. dilate(img, kernel)**

**A black and white math symbols

Description automatically generated with medium confidence**

* **Usage:** Performs dilation on a binary image. Dilation expands bright regions by adding pixels to the borders of bright regions.
* **Algorithm Explanation:**
  + The image is padded to handle boundary issues.
  + The function iterates through each pixel, extracting a region the size of the kernel.
  + If any pixel in the region is 1 (according to the kernel), the center pixel in the result is set to 1, thereby expanding bright regions.

**4. opening(img, kernel)**

**A group of black letters

Description automatically generated**

* **Usage:** Performs the opening operation, which is a sequence of erosion followed by dilation. It is used to remove small noise or isolate small bright regions.
* **Algorithm Explanation:**
  + First, erosion is applied to remove small bright regions.
  + Then, dilation is applied to restore the remaining regions while keeping small noise removed.
  + The result is smoother and cleaner, especially for small objects.

**5. closing(img, kernel)**

**A black text on a white background

Description automatically generated**

* **Usage:** Performs the closing operation, which is a sequence of dilation followed by erosion. It is used to close small holes or gaps in bright regions.
* **Algorithm Explanation:**
  + First, dilation is applied to expand bright regions and fill small holes.
  + Then, erosion is applied to restore the expanded regions, closing any gaps or small holes in the bright areas.

**6. hit\_or\_miss(img, kernel)**

**A math equations with black text

Description automatically generated with medium confidence**

* **Usage:** Performs the Hit-or-Miss operation to detect specific shapes or patterns in the image.
* **Algorithm Explanation:**
  + The kernel is divided into two parts: the foreground (where the kernel has 1s) and the background (where the kernel has -1s).
  + The function erodes the original image with the foreground kernel and the complement of the image with the background kernel.
  + The result is a logical AND between the two eroded images, producing regions where both conditions are satisfied (i.e., the specific shape is found).

**7. boundary\_extraction(img, kernel)**

**A black text on a white background

Description automatically generated**

* **Usage:** Extracts the boundaries of bright regions in the image.
* **Algorithm Explanation:**
  + The function performs erosion on the image and subtracts the eroded image from the original image.
  + The result is the boundary, or the outer edges, of the bright regions in the image.

**8. region\_filling(img, kernel, seed)**

**A math equations and formulas

Description automatically generated with medium confidence**

* **Usage:** Fills regions of bright areas starting from a seed pixel. This operation is useful for filling small holes or gaps within a bright region.
* **Algorithm Explanation:**
  + The function initializes a result image with all pixels set to 0 except for the seed pixel, which is set to 1.
  + It performs dilation iteratively on the result image, expanding the region starting from the seed, but only filling into areas that are part of the background.
  + The process continues until no changes occur between iterations, indicating that the entire region has been filled.

**IV. EXPERIMENTS AND EVALUATION:**

**Test images:** The algorithm on a test set consisting of images with different sizes, brightness levels, colors, structure and complexities as below:

A black text on a white background

Description automatically generated

A black text on a white background

Description automatically generated

A black and white logo

Description automatically generated

A close up of a flower

Description automatically generated

A black and white background with white circles

Description automatically generated

A black and white image of a triangle and a triangle

Description automatically generated

A river running through a valley

Description automatically generated

***Results with opencv mode vs manual mode:***

**Text\_tnr.png**

Execution Time (Manual (Custom)): 15.685021 secondsA black and white text

Description automatically generated

Execution Time (OpenCV): 0.002056 seconds

A black and white text

Description automatically generated

**Comments:**

Similar results in morphological operations, but the OpenCV built-in method using flood filling missed some regions. OpenCV is significantly faster than the custom implementation.

**Lover.png**

Execution Time (Manual (Custom)): 6.485807 seconds

**A collage of words

Description automatically generated**

Execution Time (OpenCV): 0.000000 seconds

**A black and white image of words

Description automatically generated**

**Comments:**

Similar results in morphological operations, but the OpenCV built-in method using flood filling missed some regions. OpenCV is significantly faster than the custom implementation.

**Discover.jpg**

Execution Time (Manual (Custom)): 30.812511 secondsA screenshot of a video game

Description automatically generated

Execution Time (OpenCV): 0.004053 seconds

A screenshot of a video game

Description automatically generated

**Comments:**

Similar results in morphological operations, but the OpenCV built-in method using flood filling missed some regions. OpenCV is significantly faster than the custom implementation.

**Flower.png**

Execution Time (Manual (Custom)): 9.767103 secondsA black and white flower images

Description automatically generated

Execution Time (opencv): 0.016066 seconds

**A black and white image of flowers

Description automatically generated**

**Comments:**

Similar results in morphological operations, but the OpenCV built-in method using flood filling missed some regions. OpenCV is significantly faster than the custom implementation.   
Boundary operator (openCV) mode is more detailed than the custom version

**Circle.png**

Execution Time (manual): 7.076829 seconds

**A collage of images of circles

Description automatically generated**

Execution Time (opencv): 0.016066 seconds

**A collage of images of circles

Description automatically generated**

**Comments:**

Similar results in morphological operations, but the OpenCV built-in method using flood filling missed some regions. OpenCV is significantly faster than the custom implementation.

**Star.png**

Execution Time (manual): 299.239042 seconds

**A black and white image of circles and triangles

Description automatically generated**

Execution Time (opencv): 0.000000 seconds

**A screenshot of a computer screen

Description automatically generated**

**Comments:**

Similar results in morphological operations, but the OpenCV built-in method using flood filling missed some regions. OpenCV is significantly faster than the custom implementation.   
Boundary operator (openCV) mode is more detailed than the custom version.

Region filling (manual) is not as expected, wrong region.

**Landscape.png**

Execution Time (manual): 24.075548 seconds A collage of black and white images of trees

Description automatically generated

Execution Time (opencv): 0.005046 seconds

**A collage of different images of trees

Description automatically generated Comments:**

Similar results in morphological operations, but the OpenCV built-in method using flood filling missed some regions. OpenCV is significantly faster than the custom implementation.   
Boundary operator (openCV) mode is more detailed than the custom version

**Comparison in general**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Custom Implementation** | **OpenCV Implementation** |
| **Performance (Speed)** | Slow (nested loops, sequential processing) | Fast (optimized C++ backend, SIMD, parallel processing) |
| **Accuracy** | Can be prone to errors (padding issues, structuring element misalignment) | Highly reliable and precise |
| **Dilation & Erosion** | Works if implemented correctly but slower | Optimized and fast |
| **Opening & Closing** | Affected by dilation/erosion accuracy | Standardized and efficient |
| **Hit-or-Miss** | May introduce errors if background processing is incorrect | Handles structuring elements correctly |
| **Boundary Extraction** | Accurate if erosion is properly done | Faster with correct results |
| **Region Filling** | Uses iterative dilation, slow for complex shapes | Uses flood fill, faster and more memory-efficient |