
Hood College

CS 428: Artificial Intelligence

# Image Processing Filters Explained

(Extended)

*Instructor:*
Prof. Qian

*Date:*
November 6, 2024

# Contents

# Chapter 1

# Introduction

In this document, we explore various image processing techniques using OpenCV. Each technique is explained in detail, including how to adjust parameters to achieve the desired effect. We'll use the variable `roi` as our source image throughout the examples.

# Chapter 2

# Filtering Techniques

## 2.1 Averaging Filter

```
blur = cv2.blur(roi, ksize=(5, 5))
```

### Purpose

The averaging filter smooths an image by reducing noise and details. It computes the average of all the pixels under the kernel area and replaces the central pixel with this average value.

### How It Works

- **Kernel Size (`ksize`)**: Defines the size of the neighborhood over which the average is calculated. In this case, a $5 \times 5$ kernel is used.

### Adjusting Parameters

1. **Kernel Size (`ksize=(5, 5)`)**
   - **Effect**:
     - *Smaller Kernel*: Less blurring; more detail preserved.
     - *Larger Kernel*: More blurring; more noise reduction but can blur important details.
   - **Adjustment**: Change `ksize` to `(3, 3)`, `(7, 7)`, etc., depending on the desired level of smoothing.

### Practical Tips

- **Odd Kernel Sizes**: Kernel dimensions should typically be odd numbers to have a central pixel.
- **Uniform Weighting**: All pixels in the kernel contribute equally to the average.

## 2.2 Gaussian Filter

```
gaussian = cv2.GaussianBlur(roi, ksize=(5, 5), sigmaX=0)
```

#### Purpose

Applies a Gaussian filter to smooth the image and reduce noise while preserving edges better than the averaging filter.

#### How It Works

- **Gaussian Kernel**: Uses a kernel where the weights decrease with distance from the center pixel, following a Gaussian distribution.

#### Adjusting Parameters

1. **Kernel Size (`ksize=(5, 5)`)**
   - **Effect**:
     - *Smaller Kernel*: Less blurring.
     - *Larger Kernel*: More blurring.
   - **Adjustment**: Use odd integers like `(3, 3)`, `(7, 7)`.
2. **SigmaX (`sigmaX=0`)**
   - **Effect**:
     - *`sigmaX=0`*: Automatically calculated based on `ksize`.
     - *Larger `sigmaX`*: More blurring; controls the spread of the Gaussian curve.
   - **Adjustment**: Specify `sigmaX` to control the amount of blurring independently of `ksize`.

#### Practical Tips

- **Edge Preservation**: Gaussian blur is better at preserving edges compared to the averaging filter.
- **Anisotropic Blurring**: You can specify `sigmaY` for different blurring in the Y direction.

## 2.3 Median Filter

```
median = cv2.medianBlur(roi, ksize=5)
```

#### Purpose

Reduces salt-and-pepper noise by replacing each pixel's value with the median of its neighboring pixels.

#### How It Works

- **Median Calculation**: Sorts the pixel values under the kernel and replaces the central pixel with the median value.

### Adjusting Parameters

1. **Kernel Size (`ksize=5`)**
    - **Effect**:
        - *Smaller Kernel*: Less noise reduction; more detail preserved.
        - *Larger Kernel*: More noise reduction; can blur edges.
    - **Adjustment**: Use odd integers greater than 1, like `ksize=3`, `ksize=7`.

### Practical Tips

- **Effective Against Impulse Noise**: Particularly good at removing isolated noisy pixels without blurring edges.
- **Kernel Size Limit**: For performance, avoid excessively large kernel sizes.

## 2.4 Non-Local Means Denoising

### For Color Images

```
dst_color = cv2.fastNlMeansDenoisingColored(roi, h=10, hColor=10)
```

### For Grayscale Images

First, ensure the image is grayscale:

```
gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
dst_gray = cv2.fastNlMeansDenoising(gray_roi, h=10)
```

### Purpose

Removes noise while preserving significant details and textures.

### How It Works

- **Non-Local Means**: Considers the similarity of pixels across the entire image, not just immediate neighbors.

### Adjusting Parameters

1. **Filter Strength**
    - `h=10` for grayscale, `h=10`, `hColor=10` for color.
    - **Effect**:
        - *Lower* `h`: Less denoising; preserves more details.
        - *Higher* `h`: More denoising; may lose fine details.
    - **Adjustment**: Increase `h` for images with high noise levels.
2. **Template Window Size (`templateWindowSize`)**

- **Default**: 7
- **Effect**: Size of the patch used to compute weights.
- **Adjustment**: Larger sizes may improve denoising but increase computational time.

3. **Search Window Size (`searchWindowSize`)**
    - **Default**: 21
    - **Effect**: Size of the window used to search for similar patches.
    - **Adjustment**: Larger sizes increase computation time and may improve denoising.

**Practical Tips**

- **Color vs. Grayscale**: Use `fastNlMeansDenoisingColored` for color images to preserve color fidelity.
- **Computationally Intensive**: May be slower than other filters; use when preserving textures and details is important.

# Chapter 3

# Edge Detection Techniques

## 3.1 Sobel Operator

```
sobelx = cv2.Sobel(roi, cv2.CV_64F, 1, 0, ksize=5)
sobely = cv2.Sobel(roi, cv2.CV_64F, 0, 1, ksize=5)
```

### Purpose

Detects edges by calculating the gradient of the image intensity in the x and y directions.

### How It Works

- **Gradient Calculation**: Computes the first-order derivative, highlighting regions of high spatial frequency that correspond to edges.

### Adjusting Parameters

1. **Data Depth (`cv2.CV_64F`)**
   - **Purpose**: Allows for negative gradient values and prevents data overflow.
   - **Adjustment**: Use higher precision to capture fine gradients.
2. **Derivative Order (`dx`, `dy`)**
   - `dx=1`, `dy=0`: Gradient in x-direction.
   - `dx=0`, `dy=1`: Gradient in y-direction.
3. **Kernel Size (`ksize=5`)**
   - **Effect**:
     - *Smaller Kernel*: Detects finer edges but more sensitive to noise.
     - *Larger Kernel*: Smoother gradients; less sensitive to noise.
   - **Adjustment**: Use odd numbers from 1 to 31.

### Practical Tips

- **Edge Magnitude and Direction**: Combine `sobelx` and `sobely` to compute the gradient magnitude and direction.
- **Visualization**: Convert the output to an 8-bit image using `cv2.convertScaleAbs()` before displaying.

## 3.2 Laplacian Operator

```
laplacian = cv2.Laplacian(roi, cv2.CV_64F)
```

**Purpose**

Detects edges by calculating the second-order derivative of the image intensity.

**How It Works**

- **Second Derivative**: Highlights regions where the intensity changes rapidly, which often correspond to edges.

**Adjusting Parameters**

1. **Data Depth (`cv2.CV_64F`)**
   - **Purpose**: Captures negative and positive values resulting from second derivatives.
   - **Adjustment**: Use higher precision data types to prevent overflow.
2. **Kernel Size (`ksize`)**
   - **Default**: 1
   - **Effect**: Larger kernels can smooth the image more but may miss finer details.
   - **Adjustment**: Use odd numbers, typically 1, 3, or 5.

**Practical Tips**

- **Noise Sensitivity**: The Laplacian operator is sensitive to noise; consider smoothing the image first.
- **Zero-Crossings**: Useful for edge detection algorithms that utilize zero-crossings.

# Chapter 4

# Morphological Operations

## 4.1 Erosion and Dilation

**Define the Kernel**

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5, 5))
```

**Erosion**

```
erosion = cv2.erode(roi, kernel, iterations=1)
```

**Dilation**

```
dilation = cv2.dilate(roi, kernel, iterations=1)
```

**Purpose**

- **Erosion**: Removes pixels at the boundaries of objects; shrinks the foreground.
- **Dilation**: Adds pixels to the boundaries of objects; expands the foreground.

**How They Work**

- **Structuring Element (`kernel`)**: Determines the neighborhood over which the operation is performed.

**Adjusting Parameters**

1. **Kernel Shape and Size**
   - **Shapes**: Rectangle (`cv2.MORPH_RECT`), Ellipse (`cv2.MORPH_ELLIPSE`), Cross (`cv2.MORPH_CROSS`).
   - **Size**: `(5, 5)` determines the extent of the operation; larger kernels have a more significant effect.
2. **Iterations**
   - **Effect**:
     - *More Iterations*: Increases the amount of erosion or dilation.
   - **Adjustment**: Modify `iterations` to control the extent of the operation.

**Practical Tips**

- **Noise Removal**: Erosion can eliminate small white noises.
- **Gap Bridging**: Dilation can fill in small holes or gaps in objects.

## 4.2 Opening and Closing

```
opening = cv2.morphologyEx(roi, cv2.MORPH_OPEN, kernel)
closing = cv2.morphologyEx(roi, cv2.MORPH_CLOSE, kernel)
```

**Purpose**

- **Opening**: Performs erosion followed by dilation; removes small objects or noise.
- **Closing**: Performs dilation followed by erosion; fills small holes and gaps.

**How They Work**

- **Opening**:
  - **Erosion**: Removes small foreground objects.
  - **Dilation**: Restores the size of larger objects.
- **Closing**:
  - **Dilation**: Fills small holes in the foreground.
  - **Erosion**: Restores the original size.

**Adjusting Parameters**

- **Kernel**: Same as in erosion and dilation; adjusting the kernel changes the effect.
- **Iterations**: Can be increased for a more pronounced effect.

**Practical Tips**

- **Noise Reduction**: Opening is effective for removing small, isolated noises.
- **Object Enhancement**: Closing can help in connecting disjointed parts of an object.

# Chapter 5

# Custom Kernel Filtering

```
kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])
filtered = cv2.filter2D(roi, -1, kernel)
```

### Purpose

Applies a custom convolution kernel to the image, allowing for specialized filtering like sharpening.

### How It Works

- **Convolution**: The kernel is slid over the image, and for each position, the sum of the element-wise multiplication of the kernel and the overlapping image region is computed.

### Adjusting Parameters

1. **Kernel Definition**
   - **Sharpening Kernel**: The given kernel enhances edges and details.
   - **Adjustment**: Modify the kernel matrix to achieve different effects:
     - *Edge Detection*: Use kernels like `[[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]]`.
     - *Blur*: Use a kernel where all values are equal and sum to 1.
2. **Data Depth (`ddepth=-1`)**
   - `-1`: Output image has the same depth as the source image.
   - **Adjustment**: Use specific depths to prevent data overflow or underflow.

### Practical Tips

- **Normalization**: Ensure the sum of kernel elements is 1 for averaging effects.
- **Edge Artifacts**: Be cautious of edge effects; padding methods can mitigate artifacts at image borders.

# Chapter 6

# General Tips for Parameter Adjustment

1. **Understand Your Image**
    - Analyze the image to determine the type of noise or features you want to enhance or suppress.
    - Choose filters and parameters accordingly.
2. **Experiment Iteratively**
    - Start with default parameters.
    - Adjust one parameter at a time to observe its effect.
3. **Visual Inspection**
    - Display the processed image after each operation.
    - Compare with the original to assess improvements.
4. **Combine Techniques**
    - Use multiple filters in sequence to achieve complex effects.
    - Example: Denoise with a median filter before edge detection.
5. **Performance Considerations**
    - Larger kernels and higher iterations increase computational load.
    - Optimize parameters for a balance between quality and performance.

# Chapter 7

# Additional Notes

- **Data Types**: Be mindful of the data types when performing operations. Some functions may require conversion between data types to prevent overflow or underflow.
- **Color Spaces**: Some operations are more effective in different color spaces. Consider converting images to grayscale or other color spaces like HSV if appropriate.
- **Edge Handling**: Filters can behave differently at the edges of images. Padding strategies can influence the results.
- **OpenCV Documentation**: Refer to the OpenCV documentation for detailed information on functions and additional parameters.

# Chapter 8

# Conclusion

By understanding each of these techniques and their parameters, you can effectively process images to enhance features, reduce noise, and prepare images for further analysis. Experimentation and practice will help you choose the right combination of filters for your specific application.