CS 5330: Pattern Recognition and Computer Vision

Northeastern University

# OpenCV Workshop
## Lab 6: Smoothing and Blurring

* *Contributed by Fall 2024 TAs: Byunghyun Ko, Yihan Wang and Taiwei Cui*

# Smoothing and Blurring

1. Importance of Smoothing and Blurring
2. Apply Averaging (Box Filter)
3. Apply Gaussian Blurring
4. Apply Median Blurring
5. Apply Bilateral Filtering

# Importance of Smoothing and Blurring

- Smoothing and blurring are essential image preprocessing techniques that aim to reduce noise and detail in an image.
- These methods are particularly useful in preparing images for tasks that require a focus on broader features rather than fine details.
- Applications in Computer Vision:
    - **Feature Extraction**: Simplifies images to make feature extraction more robust.
    - **Edge Detection**: Prepares the image by reducing noise, ensuring cleaner edge detection.
    - **Object Recognition**: Enhances the image quality for better object recognition accuracy.

# Apply Averaging (Box Filter)

- The simplest method for blurring an image is by averaging.
- In this technique, each pixel is replaced by the average of its neighboring pixels.

Average Filter Box Filter

https://medium.com/@shashikadilhani97/digital-image-processing-filters-832ec6d18a73

# Apply Averaging (Box Filter)

Code

```
# Apply averaging (box filter)
blurred_avg = cv2.blur(image, (5, 5))

# Display the blurred image
plt.imshow(blurred_avg)
plt.title("Averaging (Box Filter)")
plt.show()
```

**cv2.blur(src, ksize)**:

- **src**: Input image.
- **ksize**: Size of the kernel (tuple), such as (5, 5), which determines the neighborhood size for averaging.

# Apply Gaussian Blurring

- Gaussian blurring uses a Gaussian function to smooth the image. This is more effective than averaging in preserving edges while reducing noise.

In mathematics, a **Gaussian function**, often simply referred to as a **Gaussian**, is a function of the base form

$$f(x) = \exp(-x^2)$$

and with parametric extension

$$f(x) = a \exp\left(-\frac{(x-b)^2}{2c^2}\right)$$


https://en.wikipedia.org/wiki/Gaussian_function

# Apply Gaussian Blurring

Code

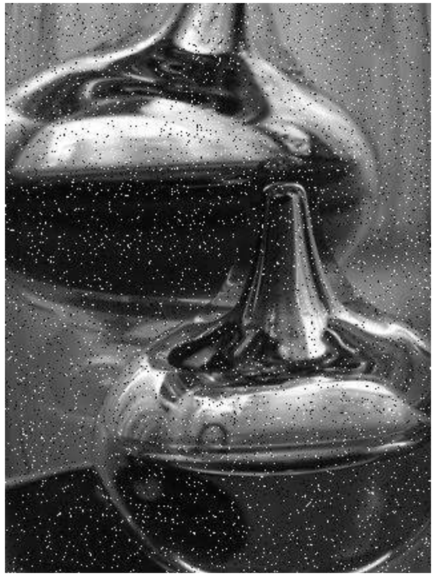
```
# Apply Gaussian blur
blurred_gauss = cv2.GaussianBlur(image, (5, 5), 0)


# Display the blurred image
plt.imshow(blurred_gauss)
plt.title("Gaussian Blurring")
plt.show()
```

**cv2.GaussianBlur(src, ksize, sigmaX)**:

- **src**: Input image.
- **ksize**: Kernel size, must be odd (e.g., (5, 5)).
- **sigmaX**: Standard deviation in the X direction. Set to 0 to let OpenCV automatically calculate based on the kernel size.

# Apply Median Blurring

- Median blurring replaces the pixel value with the median of neighboring pixel values. This method is effective at removing "salt-and-pepper" noise.
  - “Salt and pepper” noise presents itself as sparsely occurring white and black pixels.



https://en.wikipedia.org/wiki/Salt-and-pepper_noise

# Apply Median Blurring

Code

```
# Apply median blur
blurred_median = cv2.medianBlur(image, 5)


# Display the blurred image
plt.imshow(blurred_median)
plt.title("Median Blurring")
plt.show()
```

**cv2.medianBlur(src, ksize):**

- **src**: Input image.
- **ksize**: Kernel size, must be an odd number (e.g., 5).

# Apply Bilateral Filtering

- Bilateral filtering is a technique that smooths the image while preserving edges by considering both pixel intensity differences and spatial proximity.

https://en.wikipedia.org/wiki/Bilateral_filter

# Apply Bilateral Filtering

Code

```
# Apply bilateral filter
blurred_bilateral = cv2.bilateralFilter(image, 9, 75, 75)


# Display the blurred image
plt.imshow(blurred_bilateral)
plt.title("Bilateral Filtering")
plt.show()
```

**cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace)**:

- **src**: Input image.
- **d**: Diameter of the pixel neighborhood.
- **sigmaColor**: Filter sigma in the color space (larger values blur more).
- **sigmaSpace**: Filter sigma in the coordinate space (larger values mean more spatial smoothing).

# Summary

- **Averaging** is useful when you need basic noise reduction without caring much about edge preservation.
- **Gaussian Blurring** is preferred when you need smoother results with better edge preservation.
- **Median Blurring** works best for images with sharp noise, such as salt-and-pepper noise.
- **Bilateral Filtering** should be used when you need to reduce noise while preserving edges.