STB600 Lab 2: Intensity transformation and spatial filtering

Atieh Sahraeidolatkhaneh: atieh.sahraeidolatkhaneh@hv.se

Yongcui Mi: yongcui.mi@hv.se

1. Purpose of the Lab

This lab provides practice with key image processing operations: intensity transformations and spatial filtering. You will implement, analyze, and compare different methods, using real images in Python and OpenCV.

2. Learning Outcomes

After completing this lab, you will be able to:

- Apply common intensity transformations (negative, log, gamma).
- Implement and compare spatial filters (averaging, Gaussian, median).
- Explain the visual effect and practical use of each method.
- Interpret results and reflect on advantages and limitations.

3. Environment Setup

Before starting the lab, make sure you activate the Python environment created in Lab 1:

1. Open the **Anaconda PowerShell Prompt**.
2. Navigate to your lab folder (you can copy the path from File Explorer and use `cd`).
3. Activate the course environment:

```
conda activate stb600
```

If the command `conda activate stb600` does not work, you are probably not using the Anaconda PowerShell Prompt.

4. Instructions

Work through each task below. During the oral demonstration with the lab assistant, you should be able to:

- Show the original and processed images on your screen.
- Explain the effect of each method in 2–3 sentences.
- Discuss when the method is useful and its limitations.
- Compare the filtering methods based on your observations.
- Be prepared for follow-up questions to confirm your understanding of the methods.

5. Intensity Transformations

In this section, use `img.tif`. Before starting, load the image:

```
import cv2
import numpy as np

img = cv2.imread('img.tif', cv2.IMREAD_GRAYSCALE)
cv2.imshow('Original', img)
cv2.waitKey(0)
```

5.1 Image Negative

Concept. Image negatives invert brightness: bright areas become dark and vice versa. Useful for analyzing biomedical images, X-rays, and low-intensity details.

You may find the following operations useful.

- `cv2.bitwise_not()`
- or simply computing `255 - img`

```
## Your code
```

Reflection. When would a negative image make details easier to see?

5.2 Log Transformation

Concept. The log transform enhances dark regions by compressing high-intensity values.

$$s = c \cdot \log(1 + r)$$

You may find the following operations useful.

- `np.log()` or `np.log1p()`
- `cv2.normalize()` to scale the result back to the range 0–255

```
img_float = img.astype(np.float32)

c = 255 / np.log(1 + np.max(img_float))
## Your code
```

5.3 Gamma (Power-Law) Transformation

Concept. Gamma modifies contrast according to:

$$s = cr^{\gamma}$$

- $\gamma < 1$: enhances dark regions.
- $\gamma > 1$: enhances bright regions.

You may find the following operations useful.

- `np.power()` or `(img / 255.0) ** gamma`
- `cv2.normalize()` if you need to rescale to 0–255

```
gamma = 0.5   # try values < 1 and > 1
## Your code
```

6. Spatial Filtering

Use `gaussianNoiseImg.tif` and `peppersaltImg.tif`.

6.1 Averaging Filter

Concept. Reduces noise by averaging neighbouring pixels; however, it blurs edges strongly.

You may find the following functions useful.

- `cv2.blur()`
- or `cv2.filter2D()` with a uniform kernel

```
img = cv2.imread('gaussianNoiseImg.tif', 0)
## Your code
```

6.2 Gaussian Filter

Concept. Uses a weighted kernel; smoother and preserves edges better than simple averaging.

You may find the following function useful.

- `cv2.GaussianBlur()`

```
## Your code
```

6.3 Median Filter

Concept. Very effective for salt-and-pepper noise because it replaces each pixel with the median of its neighbourhood.

You may find the following function useful.

- `cv2.medianBlur()`

```
## Your code
```

Reflection Questions

Possible follow-up questions include, but are not limited to:

- Which filter performs best on salt-and-pepper noise? Why?
- How does increasing kernel size affect smoothing and edge detail?
- Compare the visual differences between averaging and Gaussian filtering.

7. Troubleshooting Tips

- If you get dtype errors, convert using `img.astype(np.float32)`.
- Gaussian kernel sizes must be odd (3, 5, 7, ...).
- If windows don't display, ensure `cv2.waitKey(0)` is included.
- If images appear too dark/bright, scale them to `uint8`.

8. Additional Resources

References