AIDA 2154: Computer Vision
Assignment 01

Fall 2025, RDP

Problem 1 [10 Points]: Convolution by Hand (no calculators)

You are given a 4×4 grayscale image I and a 3×3 kernel K:

$$I = \begin{bmatrix} 0 & 3 & 0 & 4 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 3 & 1 & 1 & 0 \end{bmatrix} \qquad K = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Convolution settings:

- Operation: **2-D convolution**.
- Stride: **1**
- Padding: **zero padding of 1 pixel** on all sides ("same" size).

Tasks

1. Draw (or clearly describe) the zero-padded version of I.
2. Compute the convolution result **only** at the following locations. For each, show: the 3×3 patch used, element-wise products with the (flipped) kernel, and the final sum.
   (a) Pixel $(2, 2)$ (assume indexes start at 1)
   (b) Pixel $(3, 3)$
   (c) (Optional practice) A border pixel of your choice, e.g., $(1, 2)$ or $(4, 3)$.

Problem 2 [10 Points]: Understanding Noise

Define noise in images. What are the possible sources of noise?

Problem 3 [15 Points]: Implementing Gamma Correction

Background: Gamma correction is a nonlinear operation used in image processing to adjust the luminance of an image. It applies a power-law transformation to pixel values. For an 8-bit grayscale image, the transformation is defined as:

$$LUT(i) = \left(\frac{i}{255}\right)^{\gamma} \cdot 255, \quad \gamma > 0, \quad i \in [0, 255]$$

If $\gamma < 1$, the image becomes brighter. If $\gamma > 1$, the image becomes darker.
In practice, this transformation is implemented efficiently using a lookup table (LUT).

Task: Write a Python program that:

1. Reads an image in grayscale or color using OpenCV.
2. Constructs a gamma correction lookup table (LUT) for a user-specified γ value.
3. Applies the LUT to the input image using `cv2.LUT()` to produce the gamma-corrected output.
4. Saves the corrected image to disk.

Example: - Input image: `gamma_2.jpg` - Gamma value: $\gamma = 0.5$ - Output image: `output_gamma_0_5.jpg`

Deliverables:

- Your Python code implementing gamma correction (you can use Assignment 1 code from our course's GitHub repository as starting point).
- At least two output images with different γ values to demonstrate the effect. Comment on which value of γ gives best result.

Problem 4 [15 Points]: Edge Detection

In Week 3 (Day 2), we introduced the fundamentals of **edge detection** and explored how Sobel kernels (S_x, S_y) can be used to approximate image derivatives. We then implemented the function `edge_detection_chart()` (see the code here), which computes the gradient of an input image using horizontal and vertical kernels (`kernel_h` and `kernel_v`) and produces a binary edge map by applying a threshold.

In this Exercise, you will **experiment with and tune the parameters of the Sobel operator** to analyze how they affect the quality of edge detection. Using the input image `medical_3.jpg`, apply edge detection and display the resulting edge map.

Specifically, investigate the following parameters:

1. **Normalization factor**
   Does multiplying the kernels by $\frac{1}{4}$ (or another factor) influence the clarity or intensity of the detected edges?

2. **Smoothing strength (kernel weights)**
   The coefficient 2 in the Sobel kernels controls smoothing in the orthogonal direction. What changes do you observe if you replace 2 with 1 or with 3?

3. **Kernel size**
   Extend the Sobel operator from the standard 3×3 form to larger sizes such as 5×5 or 7×7. Compare the effect of larger kernels on noise reduction and edge sharpness.

Your goal is to **evaluate how each modification affects the final edge image** and determine which settings yield the most accurate edge detection for the given medical image.

How to Submit?

Prepare your solutions in a Jupyter Notebook, push the completed notebook (showing code and cells' output) to your GitHub repository, and submit the repository link on Blackboard under Assignments.