COMP425 Assignment - 1

Lucía Cordero Sánchez
l_corde@live.concordia.ca
40291279

February 18, 2024

1 Summary

This assignment was divided in four parts. 1) Implementation of 2d filtering and partial derivatives methods , 2) Implementation of the edge detector 3) Implementation to create Harry's corner detector and 4) Implementation of downsampling function.

2 Part 1

2.1 Implement the missing code in the function "filter2d" in "utils.py"

The line **out[m, n, i] = np.sum(image[m:m + Hk, n:n + Wk, i] * filter** is the core of the *filter2d* function. This line calculates the filtered value at position (*m, n*) in the output image for channel i. Here's a breakdown of what happens:

- **image[m:m + Hk, n:n + Wk, i]** selects a region from the input image image centered at (m, n) with height *Hk* and width *Wk* for channel *i*.
- **filter** is the filter kernel, which is applied to the selected region of the image.
- **image[m:m + Hk, n:n + Wk, i] * filter** performs element-wise multiplication between the selected image region and the filter kernel.
- **np.sum(...)** calculates the sum of all the multiplied values, which represents the result of multiplying the filter with the selected image region.
- Finally, the result is assigned to *out[m, n, i]*, which is the output image at position *(m, n)* for channel *i*.

This process is repeated for every pixel (m, n) in the output image and for every channel i, effectively applying the 2D image filtering operation to the entire image.

2.2 Implement the functions "partial_x" and "partial_y" in "utils.py". In your report, write down the filter you used in each of these two functions

In the partial_x and partial_y function, the Sobel filter for the x-direction is applied, which is [1, 0, -1; 2, 0, -2; 1, 0, -1]. This filter emphasizes vertical and horizontal edges in the image.
I chose the Sobel filters over other filters like Prewitt for various reasons:

- Sobel filters are more sensitive to edges due to their larger central coefficients, which can lead to better edge detection performance in certain scenarios.
- In Sobel operator, the coefficients of masks are adjustable according to our requirement provided they follow all properties of derivative masks.
- Its noise-suppression characteristics are slightly better fue to the properties of the derivative masks that it has.

3 Part 2

3.1 Implement part of an edge detector. Apply your code on the image "iguana.png" and visualize these 3 gradient images (gradient image on x direction, gradient image on y direction, gradient magnitude). Put these 3 figures in your report.

I have followed these steps to implement my edge detector:

1. Loaded an image of an iguana in grayscale.
2. Applied a Gaussian smoothing filter to the image with a kernel size of 5x5 and a standard deviation of 1.0. This step helps reduce noise in the image and provides a smoother gradient for edge detection.
3. Computed the partial derivatives of the smoothed image in the x and y directions using the Sobel filters. These partial derivatives represent the rate of change of pixel intensity in the respective directions.
4. Gradient magnitude is computed using the formula $\sqrt{\text{img\_dx}^2 + \text{img\_dy}^2}$, where *img_dx* and *img_dy* are the partial derivatives in the x and y directions, respectively. This magnitude represents the strength of the gradient at each pixel and is used to detect edges.
5. The detector visualizes the results by displaying the x gradient, y gradient, and gradient magnitude as three separate images using *matplotlib* library (see Figure 3).

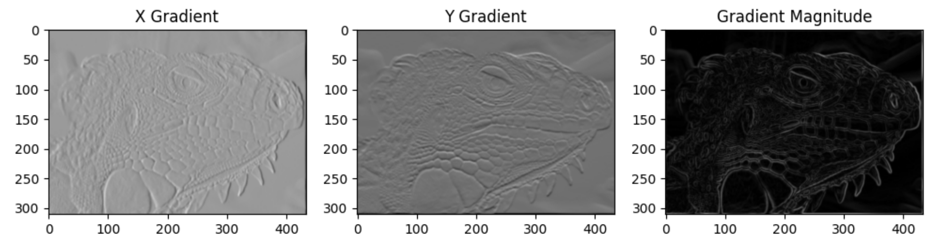

Figure 1: X-direction gradient, Y-direction gradient and Gradient Magnitude (left to right)

4 Part 3

4.1 Fill in the missing code in "corner.py" to implement the Harris corner detector and apply it on the image "building.jpg". Generate 3 figures.

The Harris corner detector involves three major steps: computing the corner response map on each pixel,thresholding on the response and Non-maximum suppression (NMS).

The implemented code follows these steps. First, it smooths the input image with a Gaussian kernel before computing the gradients. This is done to reduce noise and improve the stability of the corner detection. The gradients of the smoothed image along the x and y directions are then computed.

Next, the elements of the structure tensor M are computed using the gradients: I_x^2, I_y^2, and $I_x I_y$. A square window of size *window_size* is used to sum up the elements of M in the neighborhood of each pixel.

The Harris corner response map is computed using the formula:

$$R = \det(M) - k \times \text{trace}(M)^2$$

where k is a sensitivity parameter. The response map is then thresholded to identify potential corners, with the threshold value chosen as a percentage of the maximum response value.

Finally, non-maximum suppression (NMS) is performed to find the peak local maximums in the response map. The 'peak_local_max' function from skimage is used for this purpose.

The implemented code uses the following parameters:

- **window_size**: 4, which defines the size of the window function used to compute the gradients and the structure tensor.
- **k**: 0.04, the sensitivity parameter in the Harris corner response formula.
- **threshold**: 0.005, only pixels with a corner response function value greater than 0.005 will be considered corners by the algorithm.

The code does not use Gaussian weighted derivatives explicitly. Instead, it applies a Gaussian filter to the input image before computing the gradients, which indirectly smooths the gradients and reduces the impact of noise in the corner detection process.

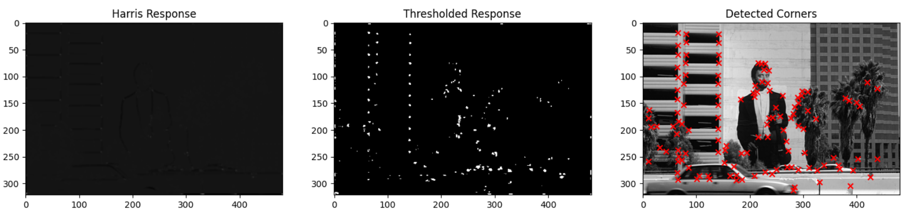

Figure 2: Harris Response map, Thresholded Response and NMS Response (left to right)

The visualizations provided by the code (see Figure 3) include the Harris response map, the thresholded response map, and the detected corners in the input image, displayed as an "X" at the corresponding coordinates.

5 Part 4

5.1 Fill in the missing code in "downsample.py". This code implements both the naïve downsampling (with aliasing) and the one without aliasing. Put the figures in the report.

The goal of the code is to downsize an input image "paint.jpg" to a factor of 2 for 5 levels. This is achieved through two methods: naive downsampling (with aliasing) and anti-aliasing downsampling.

1. **Image Loading**: The image is loaded as a floating-point array and normalized to the range [0, 1]. In contrast with the other parts, I loaded the colored image (not in black and white).

2. **Naïve Downsampling** (1st Row): The original image is copied, and for each level, the image is subsampled by a factor of 2. Each resulting image is displayed in the 1st row of the visualization.

3. **Anti-Aliasing Downsampling** (2nd Row): For each level, the image is split into its RGB channels. A Gaussian **filter is applied to each channel separately** using the filter2d function. The **filtered channels are then recombined** into a single image. This image is then subsampled by a factor of 2 and displayed in the 2nd row of the visualization.

4. **Visualization**: The visualization consists of two rows of images. The 1st row shows the results of naive downsampling, while the 2nd row shows the results of anti-aliasing downsampling.

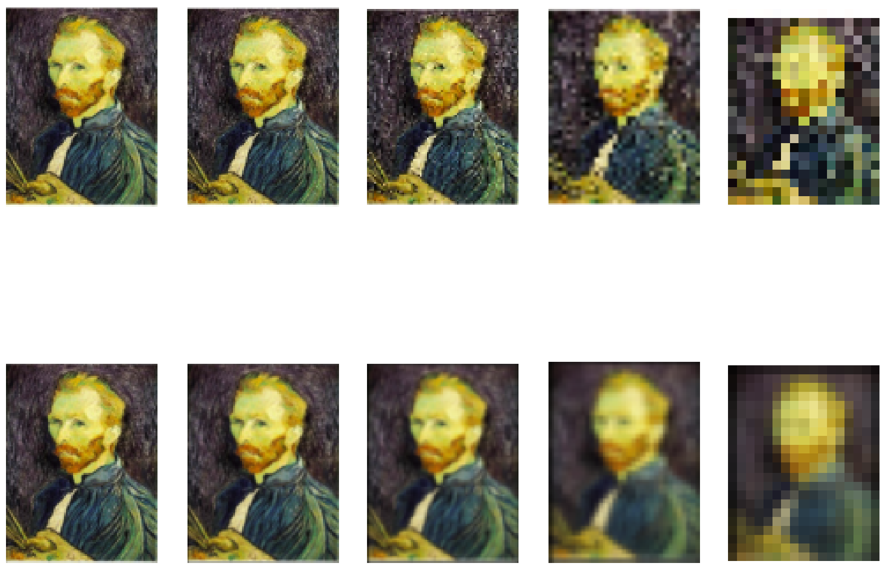

Figure 3: Naïve downsampling and Anti-aliasing downsampling (top to bottom)