Exercise 02 for MA-INF 2201 Computer Vision WS23/24
22.10.2023
Submission on 05.11.2023

1. **Fourier Transform** In this task, we will show that the convolution in spatial domain can be computed by multiplication in the frequency domain. Read the image `einstein.jpeg`, and create a 7×7 Gaussian kernel with `sigma=1` (you can use `cv2.getGaussianKernel`).

   - Blur the image using convolution in spatial domain (`cv2.filter2D`) with the created Gaussian kernel; then plot the image.
   - Blur the image using your own implementation of convolution in spatial domain with the created Gaussian kernel; then plot the image.
   - Blur the image using Fourier Transform and multiplication; then plot the image. You can use `numpy.fft`.
     There is something strange with the result. Fix it and give your explanation of why the result was not as expected.
   - Print the mean absolute difference of the two blurred images.

   (2 Points)

2. **Template Matching** In this task, we will implement template matching using two different similarity measures: Sum Square Difference (SSD) and Normalized Cross-Correlation (NCC). Read the image `lena.png` and the template `eye.png` and convert them to `float` in the range $[0, 1]$.

   - Implement template matching using your implementation of SSD and NCC.
   - In the image, draw the rectangles around the pixels where *similarity* ≤ 0.1 for SSD and where *similarity* ≥ 0.7 for NCC. You can use `np.where`.

   Now, try to subtract 0.5 to the image (make sure that the values do not become negative) and repeat the template matching. Are there any differences between using SSD and NCC? If so, why in your opinion?

   (3 Points)

3. **Gaussian Pyramid** The professor, during the lecture, told you that using the Gaussian pyramid for template matching will make it faster. Let's see if this is true. Read image `traffic.jpg` and the template `traffic-template.png`

   - Build a 4 level Gaussian pyramid.
   - Build a 4 level Gaussian pyramid using `cv2.pyrDown`. Compare it with your implementation by printing the mean absolute difference at each level.
   - Do the template matching by using your implementation of normalized cross-correlation, print the time taken by this routine.
   - Use the pyramid technique to make template matching faster. Follow the procedure described in the lecture slides. Print the time taken by this routine.
   - Show the template matching results using the pyramid technique.

   (8 Points)

4. **Edges** In this task, we will detect edges in the image using derivative of a $2D$ Gaussian kernel. Read the image `einstein.jpeg`.

    - Compute the weights of the derivative (in x) of a 7x7 Gaussian kernel with `sigma=2`.
    - Compute the weights of the derivative (in y) of a 7x7 Gaussian kernel with `sigma=2`.
    - To get the edges, convolve the image with the kernels computed in previous steps. You can use `cv2.filter2D`.
    - Compute the edge magnitude and direction (you can use `numpy.arctan2`). Display the magnitude and direction.
    - Use your implementation of Non-Maximum Suppression (NMS) to make the edges thinner. Then apply a threshold of your choice.
    - Compute again the edges using a $3x3$ Sobel kernel. Then compute the mean absolute error with respect to the previous result.

    (5 Points)

5. **Distance Transform** In this task, we will compute the precise Euclidean distance transform.

    - Read the image `traffic.jpg`, convert it to grayscale, extract the edges using `cv2.Canny`. Display the result.
    - Compute the precise Euclidean distance transform of the image by using `cv2.distanceTransform`
    - Repeat the entire process, but this time filtering more the high-frequency edges.

    (2 Points)