Exercise 2: Convolution & Edge Detection

Due date: 28.4.2022

The purpose of this exercise is to help you understand the concept of the convolution and edge detection by performing simple manipulations on images.

This exercise covers:

- Implementing convolution on 1D and 2D arrays
- Performing image derivative and image blurring
- Edge detection
- Hough Circles
- Bilateral filter

1 Convolution 10 pt

Write two functions that implement convolution of 1 1D discrete signal and 2D discrete signal. The two functions should have the following interfaces:

```
def conv1D(in_signal: np.ndarray, k_size: np.ndarray) -> np.ndarray:
    """
    Convolve a 1-D array with a given kernel
    :param in_signal: 1-D array
    :param k_size: 1-D array as a kernel
    :return: The convoluted array
    """

def conv2D(in_image: np.ndarray, kernel: np.ndarray) -> np.ndarray:
```

```
    """
    Convolve a 2-D array with a given kernel
    :param in_image: 2D image
    :param kernel: A kernel
    :return: The convoluted image
    """
```

The result of conv1D should match *np.convolve(signal, kernel, 'full')* (link) and conv2D should match *cv.filter2D* (link) with option 'border Type'=cv.BORDER_REPLICATE.

2 Image derivatives & blurring 20 pt

2.1 Derivatives 10 pt

Write a function that computes the magnitude and the direction of an image gradient. You should derive the image in each direction separately (rows and column) using simple convolution with $[1, 0, -1]^T$ and $[1, 0, -1]$ to get the two image derivatives. Next, use these derivative images to compute the magnitude and direction matrix and also the x and y derivatives.

```
def convDerivative(in_image: np.ndarray) -> (np.ndarray, np.ndarray):
    """
    Calculate gradient of an image
    :param in_image: Gray scale image
    :return: (directions, magnitude)
    """
```

Reminder:

$$Mag_G = ||G|| = \sqrt{I_x^2 + I_y^2} \tag{1}$$

$$Direction_G = \tan^{-1}\left(\frac{I_y}{I_x}\right) \tag{2}$$

2.2 Blurring: 10 pt

You should write two functions that performs image blurring using convolution between the image f and a Gaussian kernel g. The functions should have the following interface:

```
    def blurImage1(in_image: np.ndarray, k_size: int) -> np.ndarray:
```

```
    """
    Blur an image using a Gaussian kernel
    :param in_image: Input image
    :param k_size: Kernel size
    :return: The Blurred image
    """
def blurImage2(in_image: np.ndarray, k_size: int) -> np.ndarray:
    """
    Blur an image using a Gaussian kernel using Open CV built-in functions
    :param in_image: Input image
    :param k_size: Kernel size
    :return: The Blurred image
    """
```

blurImage1 should be fully implemented by you, using your own implementation of convolution (conv2D) and Gaussian kernel. blurImage2 should be implemented by using pythons internal functions: filter2D and getGaussianKernel.

Comments:

- In your implementation, the Gaussian kernel g should contain approximation of the Gaussian distribution using the binomial coefficients. A consequent 1D convolutions of [1 1] with itself is an elegant way for obtaining a row of the binomial coefficients. Explore how you can get a 2D Gaussian approximation using the 1D binomial coefficients. **Remember:**

$$\sum_{i,j} kernel_{i,j} = 1$$

.

- The border of the images should be padded same as in the 'Convolution' section (cv.BORDERREPLICATE).
- The size of the Gaussian' $kernelSize$, should always be an odd number.

3 Edge detection 10 pt

You should implement $edgeDetectionZeroCrossingLOG$ OR $edgeDetectionZeroCrossingSimple$

- ```
  def edgeDetectionZeroCrossingSimple(img: np.ndarray) -> np.ndarray:
      """
      Detecting edges using "ZeroCrossing" method
      :param img: Input image
      :return: Edge matrix
      """
  ```
- ```
  def edgeDetectionZeroCrossingLOG(img: np.ndarray) -> np.ndarray:
      """
      Detecting edges using "ZeroCrossingLOG" method
      :param img: Input image
      :return: Edge matrix
      """
  ```

For simple zero-crossing use a simple image like the 'codeMonkey', and for LoG zero-crossing try something more challenging like 'boxMan', adjust the image/Gaussian kernel size to get good results. You can find the description of each of the methods at `https://docs.opencv.org/4.0.0/`.

4 Hough Circles 30 pt

You should implement the Hough circles transform.

```
def houghCircle(img: np.ndarray, min_radius: int, max_radius: int) -> list:
    """
    Find Circles in an image using a Hough Transform algorithm extension
    To find Edges you can Use Open CV function: cv.Canny
    :param img: Input image
    :param min_radius: Minimum circle radius
    :param max_radius: Maximum circle radius
    :return: A list containing the detected circles,
                [(x,y,radius),(x,y,radius),...]
    """
```

I is the intensity image, min_radius, max_radius should positive numbers and $min_radius < max_radius$. Use the Canny Edge detector as the edge detector, you can use the function: *cv.Canny()* (link). The functions should return a list of all the circles found, each circle will be represented by:(x,y,radius). Circle center x, Circle center y, Circle radius.

* This function is costly in run time, be sure to keep the min/max Radius values close and the images small.

5 Bilateral filter 30 pt

You should implement the Bilateral filter, compare your implementation with Open CV implementation *cv.bilateralFilter()* (link).

```
    def bilateral_filter_implement(in_image: np.ndarray, k_size: int, sigma_color: float, sigma_sp
> (np.ndarray, np.ndarray):
        """
        :param in_image: input image
        :param k_size: Kernel size
        :param sigma_color: represents the filter sigma in the color space.
        :param sigma_space: represents the filter sigma in the coordinate.
        :return: Open CV implementation, my implementation
        """
```

6 Important Comments

- Self-submission
- The input of all the above functions will be gray-scale images.
- Your edges should be reasonable, but don't worry if they are not as good as OpenCV's.
- Don't wast your time on input validation.
- Do not have any plots in ex2_utils.py!

7 Submission

You should submit a zip file ,under the name "Ex2", containing:

- ex2_utils.py - This file will have all the functions above.

- ex2_main.py - This file will be the main file that executes all the functions, including your thresholds which gave you the best results. The program should print your ID at the beginning.
- All the images you used in ex2_main.py

8 Good Luck