Exercise 2: Convolution & Edge Detection

Due date: 30/04/21

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

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(inSignal:np.ndarray,kernel1:np.ndarray)->np.ndarray:
"""
Convolve a 1-D array with a given kernel
:param inSignal: 1-D array
:param kernel1: 1-D array as a kernel
:return: The convolved array
"""
def conv2D(inImage:np.ndarray,kernel2:np.ndarray)->np.ndarray:
"""
Convolve a 2-D array with a given kernel
:param inImage: 2D image
:param kernel2: A kernel
```

```
:return: The convolved image
"""
```

The result of conv1D should match *np.convolve(signal, kernel, 'full')* (link) and conv2D should match *cv2.filter2D* (link) with option 'borderType'=cv2.BORDER_REPLICATE.

2 Image derivatives & blurring

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(inImage:np.ndarray) -> (np.ndarray,np.ndarray,np.ndarray,np.ndarray):
"""
Calculate gradient of an image
:param inImage: Grayscale iamge
:return: (directions, magnitude,x_der,y_der)
"""
```

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 Bonus

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,kernel_size:np.ndarray)->np.ndarray:
"""
Blur an image using a Gaussian kernel
:param inImage: Input image
```

```
:param kernelSize: Kernel size
:return: The Blurred image
"""
def blurImage2(in_image:np.ndarray,kernel_size:np.ndarray)->np.ndarray:
"""
Blur an image using a Gaussian kernel using OpenCV built-in functions
:param inImage: Input image
:param kernelSize: 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 (v2.BORDERREPLICATE).
- The size of the Gaussian' $kernelSize$, should always be an odd number.

3 Edge detection 50 pt

You should implement the following functions:

Implement *edgeDetectionZeroCrossingSimple* **or** *edgeDetectionZeroCrossingLOG*

Each function is 10 pt except Canny, which is 30 pt.

- `def edgeDetectionSobel(img: np.ndarray, thresh: float = 0.7)`
  `-> (np.ndarray, np.ndarray):`

```
"""
Detects edges using the Sobel method
:param img: Input image
:param thresh: The minimum threshold for the edge response
:return: opencv solution, my implementation
"""
```

- `def edgeDetectionZeroCrossingSimple(img:np.ndarray)->(np.ndarray)`

  ```
  """
  Detecting edges using the "ZeroCrossing" method
  :param img: Input image
  :return: Edge matrix
  """
  ```

- `def edgeDetectionZeroCrossingLOG(img:np.ndarray)->(np.ndarray)`

  ```
  """
  Detecting edges using the "ZeroCrossingLOG" method
  :param img: Input image
  :return: :return: Edge matrix
  """
  ```

- `def edgeDetectionCanny(img: np.ndarray, thrs_1: float, thrs_2: float)`

  ```
  -> (np.ndarray, np.ndarray):
  """
  Detecting edges usint "Canny Edge" method
  :param img: Input image
  :param thrs_1: T1
  :param thrs_2: T2
  :return: opencv solution, my implementation
  """
  ```

img is the intensity image and an *edgeImage* is binary image ($\in 0, 1$) where 1's represent an edge. Each function implements edge detections according to a different method. You can find the description of each of the methods at https://docs.opencv.org/4.0.0/.

For simple zero-crossing use a simple image like the *'codeMonkey'*, and for LoG zero-crossing try something more challenging like *'boxMan'*, adjust the Gaussin kernel size to get good results. In Canny

Edge you should use Sobel to do the smoothing and get the I_x, I_y derivatives.

4 Hough Circles 30 pt

You should implement the Hough circles transform.

```
def houghCircle(img:np.ndarray,min_radius:float,max_radius:float)->list
"""
Find Circles in an image using a Hough Transform algorithm extension
:param I: Input image
:param minRadius: Minimum circle radius
:param maxRadius: Maximum circle radius
:return: A list containing the detected circles,
[(x,y,radius),(x,y,radius),...]
"""
```

img 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. 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/maxRadius values close and the images small.

5 Important Comments

- The input of all the above functions will be grayscale 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 or imshow in ex2_utils.py!**

6 Submission

You should submit the following file:

- 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

7 Good Luck