Assignment #2

This assignment is due on Monday, October 23rd 23:59:59 class via email to christian.wallraven+AMF2023@gmail.com.

Important: You need to name your file properly. If you do not adhere to this naming convention, I may not be able to properly grade you!!!

If you are done with the assignment, make one zip-file of the assignment1 directory and call this zip-file `STUDENTID_A2.zip` (e.g.: `2016010001_A2.zip`). The correctness of the IDs matters! **Please double-check that the name of the file is correct!!**

Also: Please make sure to comment all code, so that I can understand what it does. Uncommented code will reduce your points!

Finally: please read the assignment text carefully and make sure to implement EVERYTHING that is written here – if you forget to address something I wrote, this will also reduce your points! Precision is key ☺!

Part1 Taking derivatives of images (60 points):

Make an ipython notebook `image_derivative.ipynb`.

In class, we talked about derivatives of functions. Now, an image is nothing more than a function of two input variables I(x,y) that maps image coordinates (x and y) to intensity values. Note that we are going to restrict ourselves here to greyscale images, not color images!

Since this is a function, we can try to determine its derivative. And since there are two input variables, we are also going to need to determine two (partial) derivatives:

$$\frac{\delta I}{\delta x}, \frac{\delta I}{\delta y}$$

We also know that the definition of the derivative exists in limit form:

$$f'(x) := \lim_{h \to 0} \frac{f(x+h)-f(x)}{h}$$

The problem is that in an image on the computer you cannot take this derivative "properly", as you cannot make pixel differences go to zero. The best we can do in images with pixels is to take a one-pixel difference and approximate like so:

$$I'(x) \sim \frac{I(x+1)-f(x)}{1}$$

(and the same for the y-direction).

Create an image yourself as a numpy array (of floats) that has 301x301 pixels of value 255 (so, white), and then black out a square of 50x50 black pixels in the middle.

Next, create a function `image_grad(im)` that takes as input a numpy array and returns the **two** derivative images (one in x-direction, and one in y-direction) as

numpy arrays using the approximation from above. Be careful - the resulting images do NOT have the same dimension as the original image!!

Show the two derivatives together with the original image in one set of subplots using matplotlib on screen (using color bars!)

Carefully explain the pattern that you can see. Pay close attention to positive and negative values and discuss why you get the result that you get.

Next, take a greyscale(!!!!) image of your choice, insert the code that loads it (for this it is better not to use colab, because of the permission trouble you need to go through) from disk (you can use matplotlib for this!!!), and display the image and its two derivatives as subplots. Include the image in your zip-file.

Interpret the results similarly to before.

Next, you should realize that the two derivatives that you have just calculated (the image gradient) actually are a vector with two components (one in x-, the other in the y-direction). This means, we can visualize the gradient in a different way! Let's say we create our gradient vector like so:

$$\overline{g} = (\frac{\delta I}{\delta x}, \frac{\delta I}{\delta y})$$

Then this vector has a length:

$||\overline{g}||^2$ from its components using the Euclidean L2 definition

and an angle (counted from the x-axis in the image):

$$< \overline{g} = atan\frac{\frac{\delta I}{\delta y}}{\frac{\delta I}{\delta x}}$$

Note that in order to get degrees back in numpy, you will need to do:

```
np.rad2deg(np.atan2(...))
```

We can therefore visualize these two quantities. For this, create a new function called `display_image_grad(imx,imy)` and implement the visualization using matplotlib given the two gradient images. Now, we are NOT going to create two images, but will just use ONE image for visualizing these two quantities!!

In order to do this, we are going to create our visualization in HSV space, where "H" can loop through colors from red to blue (given by the angle in our case), "S" is the color's saturation (always =1), and "V" is the color's brightness or value (given by the gradient magnitude in our case).

For this, you will create a three-dimensional numpy-array `grad_hsv` with appropriate dimensions, and assign the first dimension to angle, the second dimension to "1", and the third dimension to its length at that position.

Next, normalize all values in `grad_hsv`, so that each dimension is between 0 and 1.

Once you have this numpy array, you can pass it to

`skimage.color.hsv2rgb`

which returns the rgb-image that you can then show with matplotlib.

Insert all necessary code, and show the resulting visualizations for the square and your own image.

Hint, the hsv-visualization should look similar in style to this: