CS 6643 - Computer Vision, Fall 2024 **Guido Gerig**

Project 2

Out: Mon Feb.. 26, 2024
Due: Fri March 15, 2024 (deadline: 11:59PM)

Late submissions: Late submissions result in 10% deduction for each day. The assignment will no longer be accepted 3 days after the deadline.
Grading: 25% for implementations and 75% for experiments and written report including graphs, results, and critical discussion.

Office hours:

		Mon	Tue	Wed	Thur	Fri
Guido Gerig	gerig@nyu.edu				14.00-15.00 ZOOM	
Pragnavi Ravuluri Sai	pr2370@nyu.edu					8.00-10.00
Sai Rajeev Koppuravuri	rk4305@nyu.edu			12.00-14.00 ZOOM		

Please remember that we also use campuswire for communication on homeworks.

Please read the instructions carefully. Note: we will not be running code. Rather, we will check your code to make sure your implementation is your own, and it matches your results. Your grade is primarily based on your written report. This means going beyond just showing results, but also describing them. You should produce a standalone lab report, describing results in enough detail for someone else (outside of class) to follow. Please read every page in this assignment. **Please submit a single PDF/HTML with all code included as an appendix.**

1) Convolution and Derivative Filters

Correlation/Convolution: Implement 2D convolution (denoted *) **without the use of built in functions**. Your function will take as input two 2D arrays, a filter *f* and an image *I,* and return $f*I$. You can handle the boundary of *I* in any reasonable way of your choice, such as padding with zeros based on the size of the filter *f*. You are free to make use of any part of your previous indexing implementation.

Project 2

Make the convolution module flexible to receive 1D line filters (used for separable filtering) and also masks representing image templates (used in the second question below).
Please note that all processing is **done in floating point** and not on byte images. As a first step, convert input images into float arrays, and also use float arrays for all the processing steps. Would you not have a display of floats, you can at the very end convert the results back to byte or integers.

```
def convolution(f, I):
      # Handle boundary of I, e.g. pad I according to size of f

      # Compute im_conv = f*I
      return im_conv
```

Edge Detection Filtering: Using the example image 'cameraman.png or zebra.png.

- All operations have to be performed with floating point images and operation, only for output and report you would convert back to byte [0..255] images.
- Denoise the image with a Gaussian filter. Please remember the in class demo and pdf description (uploaded to Brightspace) to create a 1D Gaussian filter, but you must use your own implementation of convolution. Use separability to consecutively filter first with a horizontal 1D Gaussian Gx and second a

vertical 1D Gaussian Gy. Reasonable choices are Gaussian width of sigma 2.0 or 3.0, with filter sizes of 3*sigma to the left and the right, which results in filter widths of 6*sigma+1.

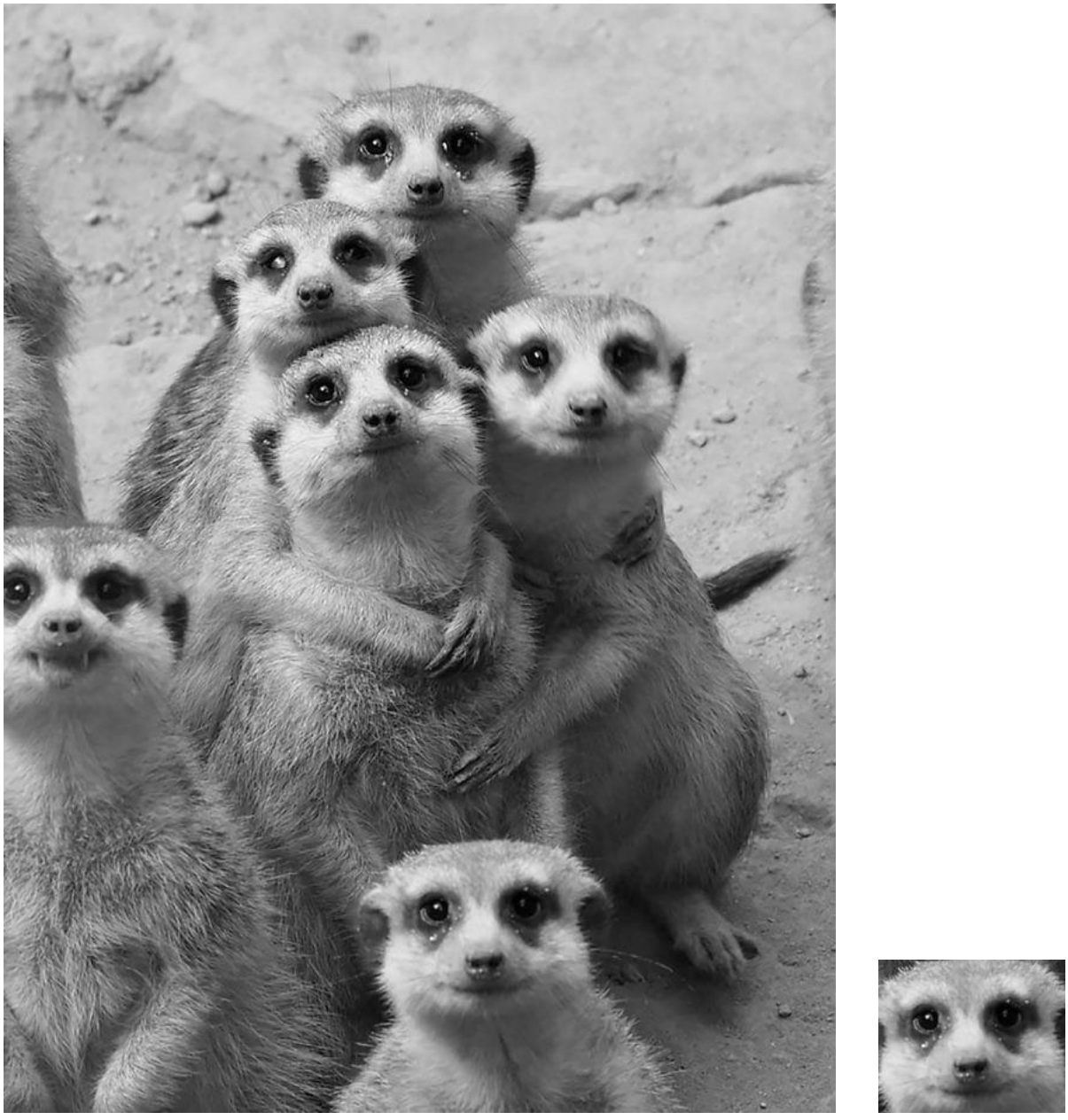
- **1) Version via first derivatives:**
  - Compute derivative images (with respect to x and with respect to y) using the separable derivative filter of your choice, e.g. [-1 0 1] and $[-1\ 0\ 1]^T$. You can hardcode the derivative filter, but use your implementation of convolution.
  - Compute the gradient magnitude image that combines x- and y-derivatives.
  - Create binary edge images from the gradient magnitude image using a threshold of your choice (trial and error till you like the result).
- **2) Version via zero-crossings of second derivative**
  - Based on the Gaussian-smoothed image, apply a 3x3 Laplacian filter to mimick the 2nd derivative in 2D: [[0,-1,0],[-1,4,-1],[0,-1,0]].
  - Write a 3x3 procedure for detection of zero-crossings following the rule "to find pixels with positive values that have at least one negative neighbor":
    - Scan the Laplacian filtered Gaussian-smoothed image, for each pixel which is positive, check all 8 neighbors for negative pixels.
    - An edge is found and marked in the output image if there is at least one negative pixel, resulting in an image where all edges are found, even in the noise.
    - **Better**: You can additionally filter for strong edges (strong slopes) if you only select negative pixels which show at least a difference value ***delta*** **or more** to your positive center pixel. The value ***delta*** is a heuristic choice which you select by trial and error.
  - The output is an edge image with thin lines of mostly one-pixel width.

Show all image results, also from intermediate steps, and discuss the outcome together with your choice of thresholds: Original image, x- and y-derivatives, magnitude image, thresholded magnitude image, Laplacian-filtered image, edge image.

Note that the x- and y-derivative, magnitude and Lapacian images may be either signed values with positive/negative values, or very dark when displayed. Here, it is best to stretch their values from [Min-Max] to [0..255] just for display.

Best is to write a small **subroutine "*ImageAdjust*"** to a) calculate Min/Max of our image array and b) adjust these values from [0..255] for output to a byte image.

2) Cross-correlation and Template Matching



Original image and single object template

The uploaded image '***animal-family-25.jpg'*** is a single image containing multiple instances of different objects.

To detect and count objects, perform the following steps:

- Use the template image "***animal-family-25-template.jpg***" as your filter-mask.
- Convert the original byte image and template to float when reading them into your program.
- Prepare the template image as follows:
  - Calculate the min, max and mean value of the template image.
  - Normalize the template image by subtracting the mean value from all pixel values, so that the mean of the resulting template becomes 0.0. This will be your *zero-mean template*.
- Implement cross-correlation with the original image and the zero-mean template image as mask. Please note that this mask is not separable.
- Please recall that this filtering results in peak values (maxima) if the template matches the specific image region which was selected for extracting your template.
- Apply the procedure "***ImageAdjust***" from task 1 to the correlation image, so that maxima show values closer to 255.
- Own experiments show that thresholding of the adjusted correlation image at 60% of 255 (150) generates peaks for the whole family, and results in a byte image of the resulting peaks.
- Overlay the peaks onto the original image for checking (TAs will provide a hint on an existing script).
- In your report, show the original image, peak image (output of cross-correlation), binarized peak image and overlay. Experiment with higher thresholds and report what you see, eventually not detecting all animals?
- Include all results and discussion in your report. Remember your report needs to be a standalone document that someone outside of class can follow.

3) Creative Part (Bonus)

You may now use your filtering modules to solve problems of your choice, below just a few ideas you may want to pursue:

1. **Counting cars in a car manufacturer's lot (e.**g. link or link2). A manufacturer may want to know how many cars of a specific type are parked on its parking lot. Procedure same as .in 2).
2. **Sorting by similarity**: Peak images show different strengths of objects that deviate from the template itself (best is to check this with the animal picture). Using connected component labeling on thresholded peak images as in 1), you could detect the maximum peak value within each connected component and sort them by size. More animal family pictures are found at https://www.boredpanda.com/animal-family-portraits/ , you may experiment with others.
3. **Etc**.

(*) connected component analysis tool provided by the TA's