Intro to Image Understanding (CSC420)

Available: September 30^{th}, 2020
Due Date: October 13^{th}, 10:59 pm, 2020
Total: 150 marks

Notes: You are encouraged to become familiar with the **Python** environments to use it for the experimental parts of the assignment. We EXPECT EVERYONE TO SUBMIT **ORIGINAL WORK** FOR ASSIGNMENTS. This means that if you have consulted anyone or any sources (including source code), you must disclose this explicitly. Anything you submit must reflect your work.

Submission Instructions:

- Your submission should be in the form of an electronic report (PDF), with the answers to the specific questions (each question separately), and a presentation and discussion of your results. For this, please submit a file called "**report.pdf**" to MarkUs directly.
- Submit documented codes that you have written to generate your results separately. Please store all of those files in a folder called Assignment1, zip the folder and then submit the file Assignment1.zip to MarkUs. You should include a **"README.txt"** file (inside the Assignment 1 folder) which details how to run the submitted codes.
- Don't worry if you realize you made a mistake after submitting your zip file: you can submit multiple times on MarkUs.

Part I: Theory (60 marks)

Q1) LTI Systems and Convolution (20 marks)

Linear time-invariant systems (LTI systems) are a class of systems used in signals and systems that are both linear and time-invariant. Linear systems are systems whose outputs for a linear combination of inputs are the same as a linear combination of individual responses to those inputs. In other words, for a system T, signals $x_1(n)$ and $x_2(n)$, and scalars a_1 and a_2, system T is linear iff:

$$T[a_1x_1(n) + a_2x_2(n)] = a_1T[x_1(n)] + a_2T[x_2(n)]$$

If $T[x(t)] = y(t)$, system T is time-invariant if a shift in its input merely shifts the output; i.e.:

$$T[x(n - n_0)] = y(n - n_0)$$

Now, consider a linear time invariant system T with input $x(n)$ and impulse response $h(n)$. Recall that the impulse response is defined as the output of the system when the input is an impulse $\delta(n)$, ı.e. $T[\delta(n)] = h(n)$. Here

$$\delta(n) = \begin{cases} 1, & \text{if } n = 0, \\ 0, & \text{else.} \end{cases}$$

Prove that $T[x(n)] = h(n) * x(n)$.
Hint: Consider that $x(n)$ can be represented via an expression that involves the function $\delta(n)$.

Q2) Polynomial Multiplication and Convolution (20 marks)

Vectors can be used to represent polynomials. For example, 3^{rd} degree polynomial $(a_3x^3 + a_2x^2 + a_1x + a_0)$ can by represented by vector $[a_3, a_2, a_1, a_0]$.
If $\mathbf{u}$ and $\mathbf{v}$ are vectors of polynomial coefficients, prove that convolving them is equivalent to multiplying the two polynomials they each represent.

Q3) Laplacian Operator (20 marks)

The Laplace operator is a second-order differential operator in the "n"-dimensional Euclidean space, defined as the divergence $(\nabla.)$ of the gradient (∇f). Thus if f is a twice-differentiable real-valued function, then the Laplacian of f is defined by:

$$\Delta f = \nabla^2 f = \nabla \cdot \nabla f$$

where the latter notations derive from formally writing:

$$\nabla = \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right).$$

Now, consider a 2D image $I(x, y)$ and its Laplacian, given by $\Delta I = I_{xx} + I_{yy}$. Here the second partial derivatives are taken with respect to the directions of the variables x, y associated with the image grid for convenience. Show that the Laplacian is in fact rotation invariant. In other words, show that $\Delta I = I_{rr} + I_{r'r'}$, where r and r' are any two orthogonal directions.
Hint: Start by using polar coordinates to describe a chosen location (x, y). Then use the chain rule.

Part II: Application (90 marks)

Q4) Edge Detection (45 marks)

You have become familiar with edge detection in Lecture 3. Edge detection algorithms try to find boundaries and outlines of objects or structures in images. In this question, you are required to implement an improved edge detection algorithm with the following steps:

Step I - Gaussian Blurring (5 marks): Implement a function that returns a 2D Gaussian matrix which can be used for filtering an image. Attempting this question will help you understand what a 2D Gaussian function is. Please note that you should NOT use **any** of the existing libraries to create the filter, e.g. fspecial() or cv2.getGaussianKernel(). Moreover, visualize this 2D Gaussian matrix for two choices of sigma and choose the size of the filter appropriately for each case.

Step II - Gradient Magnitude (10 marks): In the lectures, we discussed how partial derivatives of an image is computed. We know that the edges in an image are from the sudden changes of intensity and one way to capture that sudden change is to calculate the gradient magnitude at each pixel. The edge strength or gradient magnitude is defined as:

$$g(x, y) = |\nabla f(x, y)| = \sqrt{g_x^2 + g_y^2}$$

where g_x and g_y are the gradients of image $f(x, y)$ along x and y-axis direction respectively. Using the Sobel operator, g_x and g_y can be computed as:

$$g_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * f(x, y) \text{ and } g_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * f(x, y)$$

Implement a function that receives an image $f(x, y)$ as input and returns its gradient $g(x, y)$ magnitude as output using the Sobel operator. You SHOULD implement convolution within the body of this algorithm and NOT use existing libraries.

Step III - Threshold Algorithm (20 marks): After finding the image gradient, the next step is to automatically find a threshold value so that edges can be determined. One algorithm to automatically determine image dependent threshold is as follows:

1. Let the initial threshold be τ_0 be equal to the average intensity of gradient image $g(x, y)$, as defined below:
$$\tau_0 = \frac{\sum_{j=1}^{h} \sum_{i=1}^{w} g(x, y)}{h \times w}$$
where, h and w are height and width of the image under consideration.

2. Set iteration index $i = 0$, separate $g(x, y)$ into two classes, where the lower class consists of those pixels of $g(x, y)$ which have gradient values less than τ_0, and the upper class contains rest of the pixels.

3. Compute the average gradient values m_L and m_H of lower and upper classes respectively.

4. Set iteration $i = i + 1$ and update threshold value as:
$$\tau_i = \frac{m_L + m_H}{2}$$

5. Repeat steps 2 to 4 until $|\tau_i - \tau_{i-1}| \leq \epsilon$ is satisfied, where $\epsilon \to 0$ and take τ_i as final threshold and denote it by τ.

   Once the final threshold is obtained, each pixel of gradient image $g(x, y)$ is compared with τ. The pixels with gradient higher than τ are considered as edge point and is represented as a white pixel; otherwise it is designated as black. The edge-mapped image $E(x, y)$, thus obtained is:
$$E(x, y) = \begin{cases} 255, & \text{if } g(x, y) \geq \tau \\ 0, & \text{otherwise} \end{cases}$$

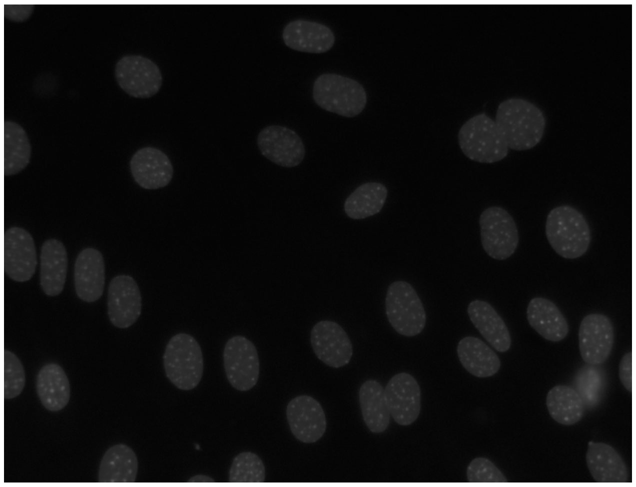
Implement the threshold algorithm. The input to this algorithm is the gradient image $g(x, y)$ and the output is a black and white edge-mapped image $E(x, y)$.

Step IV - Test (10 marks): Using the images provided along with this assignment, test all the previous steps (I to III) and visualize your results in the report. Please note that the input to each step is the output of the previous step. Besides the images provided to you, pick an image of your own (e.g. you can choose a picture of your phone camera) and feed to the algorithm implemented here (Q4). Please discuss how the algorithm works for these examples and highlight its strengths and/or its weaknesses.

Q5) Connected-component labeling (30 marks): One of the most challenging problems in computer vision is object detection. A key step in object detection is to segment a particular object from a given input image. To be able to segment an object, we can find a proposal closed contour that surrounds an object of interest. In this question, you will implement an algorithm that labels edge pixels around a region of interest by their connectivity. To learn more about connected-component labeling please check this page out: `https://en.wikipedia.org/wiki/Connected-component_labeling`. Connected-component labeling is an algorithmic application of graph theory, where subsets of connected components are uniquely labeled based on a given heuristic. Take the input image generated from Q5 (which is a binary image), with black pixels representing background and white pixels representing foreground. Here the connected components in the foreground pixels are desired and you will consider 8-connected neighbors. The algorithm steps are as follows:

1. Start from the first pixel in the image. Set current label to 1. Go to (2).
2. If this pixel is a foreground pixel and it is not already labelled, give it the current label and add it as the first element in a queue, then go to (3). If it is a background pixel or it was already labelled, then repeat (2) for the next pixel in the image.
3. Pop out an element from the queue, and look at its neighbours (the 8 neighbors around). If a neighbour is a foreground pixel and is not already labelled, give it the current label and add it to the queue. Repeat (3) until there are no more elements in the queue.
4. Go to (2) for the next pixel in the image and increment current label by 1.

Q6) Count number of cells (15 marks): Using the algorithm implemented in Question 5, try to estimate the number of cells in the image given below (the image file is available along with the assignment file). Discuss your results and explain how one can improve the



estimation.