Programming Assignment-I

Computer Vision

Coding Standard and General Requirements

Code for all programming assignments should be **well documented**. A working program with no comments will receive **only partial credit**. Documentation entails writing a description of each function/method, class/structure, as well as comments throughout the code to explain the program flow. Programming language for the assignment is **Python**. You can use standard python built-in IDLE, or CANOPY for the working environment. Other commonly used IDLEs are the following: PyCharm Community Edition, PyScripter, CodeSculptor, Eric Python, Eclipse plus PyDev.

Following libraries will be used extensively throughout the course:

- PIL (The Python Imaging Library), Matplotlib, NumPy, SciPy, OpenCV, python-graph.

If you are asked to implement "Gaussian Filtering", you are not allowed to use a Gaussian function from a known library, you need to implement it from scratch.

Submit by **11th of October 2021**, 11.59pm.

Question 1: Canny Edge Detection Implementation [5 pts]

In 1986, John Canny defined a set of goals for an edge detector and described an optimal method for achieving them. Canny specified three issues that an edge detector must address:

- **Error rate:** Desired edge detection filter should find all the edges, there should not be any missing edges, and it should respond only to edge regions.
- **Localization:** Distance between detected edges and actual edges should be as small as possible.
- **Response:** The edge detector should not identify multiple edge pixels where only a single edge exists.

Remember from the lecture that in Canny edge detection, we will first smooth the images, then compute gradients, magnitude, and orientation of the gradient. This procedure is followed by non-max suppression, and finally hysteresis thresholding is applied to finalize the steps. Briefly, follow the steps below for practical implementation of *Canny Edge detector*:

1. Read a gray scale image you can find from Berkeley Segmentation Dataset, Training images, store it as a matrix named I.
2. Create a one-dimensional Gaussian mask G to convolve with I. The standard deviation(s) of this Gaussian is a parameter to the edge detector (call it $\sigma > 0$).
3. Create a one-dimensional mask for the first derivative of the Gaussian in the x and y directions; call these G_x and G_y. The same $\sigma > 0$ value is used as in step 2.

4. Convolve the image I with G along the rows to give the x component image (I_x), and down the columns to give the y component image (I_y).

5. Convolve I_x with G_x to give I'_x, the x component of I convolved with the derivative of the Gaussian, and convolve I_y with G_y to give I'_y, y component of I convolved with the derivative of the Gaussian.

6. Compute the magnitude of the edge response by combining the x and y components. The magnitude of the result can be computed at each pixel (x, y) as: $M(x, y) = \sqrt{I'_x(x, y)^2 + I'_y(x, y)^2}$.

7. Implement **non-maximum suppression** algorithm that we discussed in the lecture. Pixels that are not local maxima should be removed with this method. In other words, not all the pixels indicating strong magnitude are edges in fact. We need to remove false-positive edge locations from the image.

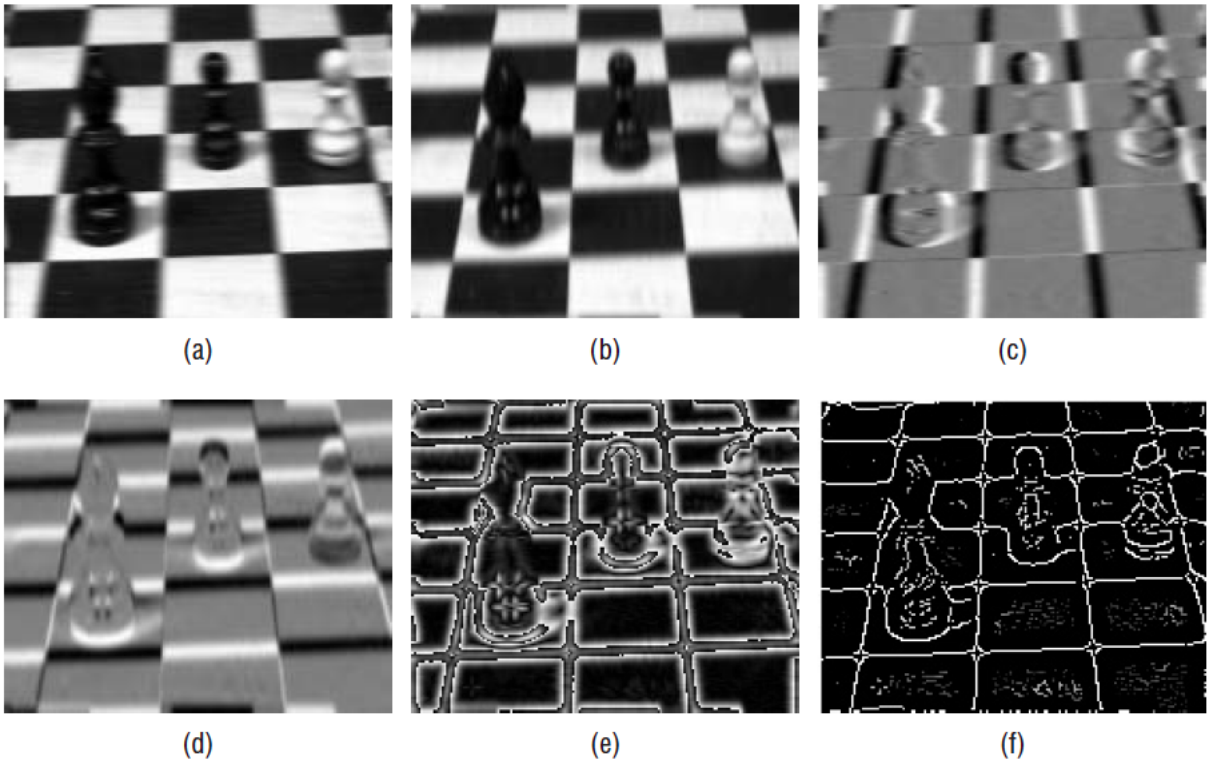
8. Apply **Hysteresis thresholding** to obtain final edge-map. You may use any existing library function to compute connected components if you want.

Definition: Non-maximal suppression means that the center pixel, the one under consideration, must have a larger gradient magnitude than its neighbors in the gradient direction. That is: from the center pixel, travel in **the direction of the gradient** until another pixel is encountered; this is the first neighbor. Now, again starting at the center pixel, travel in the direction opposite to that of the gradient until another pixel is encountered; this is the second neighbor. Moving from one of these to the other passes though the edge pixel in a direction that crosses the edge, so the gradient magnitude should be largest at the edge pixel.

Algorithmically, for each pixel p (at location x and y), you need to test whether a value $M(p)$ is maximal in the direction $\theta(p)$. For instance, if $\theta(p) = pi/2$, i.e., the gradient direction at $p = (x, y)$ is downward, then $M(x, y)$ is compared against $M(x, y - 1)$ and $M(x, y + 1)$, the values above and below of p. If $M(p)$ is not larger than the values at both of those adjacent pixels, then $M(p)$ becomes 0. For estimation of the gradient orientation, $\theta(p)$, you can simply use $atan2(I'_y, I'_x)$.

Hint: It is assumed that the gradient changes continuously as a function of position, and that the gradient at the pixel coordinates are simply sampled from the continuous case. If it is further assumed that the change in the gradient between any two pixels is a linear function, then the gradient at any point between the pixels can be approximated by a linear interpolation.

For a sample output, please refer to figure below: chessboard image's X component of the convolution with a Gaussian (a), Y component of the convolution with a Gaussian (b), X component of the image convolved with the derivative of a Gaussian (c), Y component of the image convolved with the derivative of a Gaussian (d), resulting magnitude image (e), and canny-edge image after non-maximum suppression (f) are shown.



Your tasks:

- Choose three example gray-scale images from Berkeley Segmentation Dataset (Training Images) CLICK HERE. When executed, your algorithm should plot intermediate and final results of Canny Edge Detection process as similar to the figure illustrated above.
- Please show the effect of σ in edge detection by choosing three different σ values when smoothing. Note that you need to indicate which σ works best as a comment in your assignment.

What to submit:

- Code
- A short write-up about your implementation with results: 1) A figure showing intermediate results as in Figure above, and 2) Similar Figures showing the effect of σ.

Question 2: Convolutional Neural Network (CNN) for Classification [5 pts]

Implement ConvNET using **PyTorch** for digit classification. Sample code files (two files) are given in the attachment. Fill the parts indicated clearly in the code. Output should be saved as **output.txt**. When you are asked to include convolutional layer, do not forget to include max pooling or average pooling layer(s) as well. If you want to use any other framework, you are free to do that. Remember, no base code will be provided for any other framework.

- STEP 1: Create a fully connected (FC) hidden layer (with 100 neurons) with sigmoid activation function. Train it with SGD with a learning rate of 0.1 (a total of 60 epoch), a mini-batch size of 10, and no regularization.
- STEP 2: Now insert two convolutional layers to the network built in STEP 1 (and put pooling layer too for each convolutional layer). Pool over 2x2 regions, 40 kernels, stride =1, with kernel size of 5x5.

- STEP 3: For the network depicted in STEP 2, replace Sigmoid with ReLU, and train the model with new learning rate (=0.03). Re-train the system with this setting.
- STEP 4: Add another fully connected (FC) layer now (with 100 neurons) to the network built in STEP 3. (remember that the first FC was put in STEP 1, here you are putting just another FC).
- STEP 5: Change the neurons numbers in FC layers into 1000. For regularization, use Dropout (with a rate of 0.5). Train the whole system using 40 epochs.

The traces from running testCNN.py <mode> for each of the 5 steps should be saved in output.txt, as indicated above. Each step is 1 point.

What to submit:

- Code
- A short write-up about your implementation with results and your observations from each training. Note that in each step you will train the corresponding architecture and report the accuracy on the test data. Also show how training/test loss and accuracy is varying with each iteration during the network training using plots.