Homework 3

ECE 172A
Introduction to Intelligent Systems

Winter 2022

Make sure you follow these instructions carefully during submission. Not doing so may result in a significant penalty.

- All problems are to be solved using Python unless mentioned otherwise.
- Code templates and other files are available on GitHub.
- Submit your homework electronically by following the steps listed below:
    1. Upload a pdf file with your write-up on Gradescope. This should include your answers to each question and relevant code snippets. Problems with unmarked pages may receive a 0. Make sure the report mentions your full name and PID. Finally, carefully read and include the following sentences at the top of your report:
    Academic Integrity Policy: Integrity of scholarship is essential for an academic community. The University expects that both faculty and students will honor this principle and in so doing protect the validity of University intellectual work. For students, this means that all academic work will be done by the individual to whom it is assigned, without unauthorized aid of any kind. By including this in my report, I agree to abide by the Academic Integrity Policy.
    2. Upload a zip file with all your scripts and files on Gradescope. Name this file: ECE_172A_hw3_lastname_studentid.zip. This should include all files necessary to run your code out of the box.

1 Histograms & Adaptive Histogram Equalization (25 points)

Histograms are a great statistical tool to analyze the distribution of intensity values in an image. You may use loops if necessary while solving the following problems.

1. Write a function named computeNormGrayHistogram with the following specifications:
   - One input (RGB image), one output (32-element vector)
   - Convert the image to grayscale
   - Use 32 bins to compute the histogram for the gray image. Intervals should be spaced equally between 0 and 255, resulting in a 32-length vector.
   - Normalize the histogram such that the sum of all bin values is 1.
2. Write a function named computeNormRGBHistogram with the following specifications:
   - One input (RGB image), one output (96-element vector)
   - Compute the histogram for each of the three channels (Red, Green, Blue)
   - Use 32 bins to for each color channel. Intervals should be spaced equally between 0 and 255, resulting in a 32-length vector for each channel.
   - Normalize each histogram such that the sum of all bin values is 1, and concatenate the three histograms together in the order R, G, B to make a combined 96-element histogram. Use color in your visualization to make the histogram quickly interpretable.
3. Call your functions to plot the histograms for the image forest.jpg.
4. Flip the image horizontally and plot the histograms of the new image. What are the differences?
5. Double the values of the R channel in forest.jpg and display the image and histograms of the new image.

It is often found in image processing and related fields that real world data is unsuitable for direct use. This warrants the inclusion of pre-processing steps before any other operations are performed. An example of this is histogram equalization (HE) and its extension adaptive histogram equalization (AHE). Your next task is to implement a function for AHE as described in Chapter 1 of Adaptive Histogram Equalization - A Parallel Implementation (reading shared in Canvas). The function has the following specifications:

- Two inputs: the image (im) and the contextual region size (winSize).
- Using the pseudocode in Algorithm 1 as a reference, compute the enhanced image after AHE.
- One output: the enhanced image after AHE.

Evaluate your function on the image beach.png for winSize = 33, 65, and 129. In your report, include the original image, the 3 images after AHE, and the image after simple HE (you may use Python inbuilt functions for this). Additionally, include your answers to the following questions:

- How does the original image qualitatively compare to the images after AHE and HE respectively?
- Which strategy (AHE or HE) works best for beach.png and why? Is this true for any image in general?

Algorithm 1 Adaptive Histogram Equalization

```
1: function AHE(im, winSize)
2:     Pad the image im on all 4 sides by mirroring intensity values so that the contextual
   region for edge pixels remains valid.
3:     for each pixel (x, y) in im,  do
4:         rank = 0
5:         contextual region = winSize x winSize window centered around (x, y)
6:         for (i, j) in contextual region do
7:             if im(x, y) > im(i, j) then
8:                 rank = rank + 1
9:         output(x, y) = rank x 255/(winSize x winSize)
       return output
```

2 Filtering (25 points)

Sometimes our image capture process introduces noise to the data (pixels whose values are influenced by random processes, hardware problems, or other artifacts). One way to reduce noise in an image is to apply a filter which takes the average of the neighborhood around (and including) a pixel, and use this value in place of the original value of the pixel central to the neighborhood. There are two main considerations to make in such filtering: first, how large of a neighborhood should be considered, and second, what method should we use for averaging?

Begin by looking at *mural.jpg*, *mural_noise1.jpg*, and *mural_noise2.jpg*, and answer the following questions:

1. (Extra Credit, 1 point) Where could you go to find this mural?
2. Generate a histogram and describe the issue with *mural_noise1.jpg*. Is there a name for this?
3. Generate a histogram and describe the issue with *mural_noise2.jpg*. Is there a name for this?

Now, we will try to solve the issues with four different methods.

4. Implement and apply the following four filters each to *mural_noise1* and *mural_noise2*, and display the resulting images and histograms (8 total). The output image should be the same size as the original image. You may use library functions to pad your input image, but you may not use a library function for the filtering step.
    - Mean filter, 5x5 neighborhood
    - Median filter, 5x5 neighborhood
    - Mean filter, 81x81 neighborhood
    - Median filter, 81x81 neighborhood
5. Rank the four filters by their computation time, from smallest to largest.
6. Which filter would you choose to restore *mural_noise1*?
7. Which filter would you choose to restore *mural_noise2*?

Please show your code, all images, and all sets of histograms in your report.

Can filters do more than cleaning noise from an image? Consider the template image (*template.jpg*). We would like to detect the presence of this template and similar templates in our image. We could scan through all rows looking for an exact match, but this would only work in cases when there is no noise between the template and the intended matches - nearly impossible in real-world image processing!

But, as an estimation technique, we can write this as a minimization problem:

8. Using $f(i, j)$ to represent the value of original image f at row i and column j, and $t(i, j)$ to represent the value of the template image t at row i and column j, write an expression $E(i, j)$ which takes on minimal values when the template is found at location (i, j) in the original image. *Hints: Think about a way you can bound the expression such that 0 is the minimal possible value. Your expression should involve one or more summations.*
9. Expand your expression (this is another hint for part 8), and note that the t^2 term is constant, and the f^2 term is assumed to be near constant for a large enough template size and ideal image conditions. So, in order to minimize the expression in part 8, we actually seek to maximize what expression from the expansion?
10. We use the name cross-correlation to describe the expression in part 9. Display the result of cross-correlation of the image with the template using the OpenCV library function matchTemplate(img, template, 'cv2.TM_CCORR'). Include a rectangle of size (100, 100) centered on the detected location in the original image. What do you notice? Why is this happening?

11. One way to mitigate the problem you identified in part 3 is using normalized cross-correlation, where we divide each correlation value by the magnitude of the template and the overlapping image region. Display the result of cross-correlation of the image with the template using the OpenCV library function matchTemplate(img, template, 'cv2.TM_CCORR_NORMED'). Include a rectangle of size (100, 100) centered on the detected location in the original image. Is this an improvement over part 10? Explain how this method improves (or fails to improve) from cross-correlation.

3 Canny Edge Detector (30 points)

3.1 Canny Edge Detector

In this problem, you are going to implement *Canny Edge Detection* on a grayscale image. You are allowed the use loops. A brief description of the algorithm is given below. Make sure your code reproduces the each step as given.

1. **Smoothing**: It is inevitable that all images taken from a camera will contain some amount of noise. To prevent noise from being mistaken for edges, noise must be reduced. Therefore the image is first smoothed by applying a Gaussian filter. A Gaussian kernel with standard deviation $\sigma = 1.4$ (shown below) is to be used.

$$k = \frac{1}{159} \cdot \begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

2. **Finding Gradients** The next step is to find the horizontal and vertical gradients of the smoothed image using the *Sobel* operators. The gradient images in the x and y-direction, G_x and G_y are found by applying the kernels k_x and k_y given below:

$$k_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, k_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

   The corresponding gradient magnitude image is computed using:

$$|G| = \sqrt{G_x^2 + G_y^2},$$

   and the edge direction image is calculated as follows:

$$G_\theta = arctan(\frac{G_y}{G_x}).$$

3. **Non-maximum Suppression (NMS)**: The purpose of this step is to convert the thick edges in the gradient magnitude image to "sharp" edges. This is done by preserving all local maxima in the gradient image, and deleting everything else. This is carried out by recursively performing the following steps for each pixel in the gradient image:

   - Round the gradient direction θ to nearest $45°$, corresponding to the use of an 8-connected neighbourhood.
   - Compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient direction i.e. if the gradient direction is north ($\theta = 90°$), then compare with the pixels to the north and south.
   - If the edge strength of the current pixel is largest; preserve the value of the edge strength. If not, suppress (remove) the value.

4. **Thresholding**: The edge-pixels remaining after the NMS step are (still) marked with their strength. Many of these will probably be true edges in the image, but some may be caused by noise or color variations. The simplest way to remove these would be to use a threshold, so that only edges stronger than a certain value would be preserved. Use the input t_e to perform thresholding on the non-maximum suppressed magnitude image.

Evaluate your Canny Edge Detection function on *lane.jpg* for a suitable threshold that retains the structural edges, and removes the noisy ones. Since the goal of our system is to detect the lanes, so you'll want to be sure your edge detector successfully retains most of the edges associated with lanes in the image.

In your report, be sure to include the original gradient magnitude image, the image after NMS, and the final edge image after thresholding (as well as your threshold value). If you decide to use a more complex thresholding technique, be sure to explain this in your report. As usual, include all relevant code.

4 Convolutional Neural Networks (25 points)

In this problem, we will train a convolutional neural network (also referred to as a ConvNet or CNN) for an image classification task and evaluate the performance of our trained model. In our case we will be determining which type of street sign is present in a given image. Please download the provided Jupyter notebook to modify and use throughout this problem.

4.1 Dataset

Download the INI German Traffic Sign Recognition Benchmark training dataset from this link and extract the folder to your working directory. This dataset contains tens of thousands of images of German street signs. How many different types of street signs are included in the dataset? Please display at least 2 different types of signs. Note: you may need to move the README.txt to a new location to avoid conflict when reading data using the code provided in the Jupyter notebook (though you are welcome to modify the code, too).

4.2 Loading Data

In this example, there are around 50,000 dataset images available. If we load them all at once with our simplified program (and at-home hardware), we may overwhelm our processors. Instead, begin by loading only 10 images per class (as opposed to the entire dataset) to list_images, a variable used to hold all of the image data.

4.3 Resizing Data

We resize all of the images as a preprocessing step to ensure the size of the image matches the size expected by the model. Explain the purpose of the interpolation argument in the resize function.

4.4 Shuffling and Splitting Data

Why do we split data into train, validation, and test sets? How many images do we include in each of the sets? Why do we shuffle list_images prior to splitting?

4.5 Model Details

There are many considerations to make when designing a neural network. In this problem, a model has been provided for you. Read the provided code, then answer the following questions. Be thorough in your answers; don't just tell us what an acronym stands for - explain the term.

1. What is a convolutional layer?
2. In the provided code, what do each of the arguments passed to Conv2D do?
3. What is maxpooling?
4. What is a dense layer? How is it different from a convolutional layer?
5. What is an epoch?
6. Which loss function do we use in this problem? Explain why this loss function is appropriate for the problem we are solving.
7. Create a diagram of the neural network model (either by hand or using software). The diagram should be detailed enough that another programmer would be able to recreate the model using a deep-learning library. Such diagrams are commonly used in ML research papers to communicate network architectures without verbose code.

4.6 Performance

Run the code and report the performance of the initial model. What are the two values provided in the final evaluation function, and what do they represent?

4.7 Improving the model

1. One way we can improve our performance is to provide the model with more **good** data to train on. Increase the number of images from each class to 20, 30, 40, 50, 60, 70, 80, 90, and 100. How does the accuracy change with data increase? You are welcome to load more images if you like, but we won't require this since not all computers can handle this load quickly. Include a graph showing accuracy vs number of images per class. For the following questions, continue using at least 100 images per class.

2. Another way to improve a model is to use dropout. What is dropout?

   To add dropout in Keras, use the line Dropout(0.2). What does the argument 0.2 represent? Add this line after each maxpooling layer, and after all but the last dense layer. Does the accuracy improve? Share and explain your results.

3. We can further improve the model is by including batch normalization. What is batch normalization?

   To add batch normalization in Keras, use the line BatchNormalization(). Include this line following every convolutional layer. Does the model accuracy improve? Share and explain your results.