Brandon Mohan

CMSC 630 Project Part 1

3/04/2023

My project takes 9 inputs:

Argument 1 is used to determine what RGB value to use. It should be 'R', 'G', or 'B'.
Argument 2 is used to determine the strength of the salt and pepper noise. It can be any value above 0 and under 1,000,000.
Arguments 3 and 4 are used for the mean and sigma of the gaussian noise.
Arguments 5 and 6 are used for the mask size and pixel weight of the box filter (a linear filter).
Argument 7 is the mask for the smoothing filter. Arguments 8 and 9 are the pixelweight for the smoothing filter.

Functions:

- openImg(name) – this function will take the name/location of an image file and return a 2D numpy array of it's pixel values. I used it to simplify my main function code.
- monoColor(img, color) – this function takes a 2D numpy array a RBG to isolate. For example, if the second argument is "R", the Red values will be used to create the gray image. It determines what RGB value to isolate, then loops through the image to eliminate the other colors. Note that all functions are run using the monoColor version of the image.
- histogramCalc(imgToHist) – this function uses the sudocode found in lecture 3 slide 12/32. It creates a dictionary to store all possible 255 color/intensity values. I then loop through the image provided to put each pixel into one of the dictionary keys. Once I finish traversing, I can convert the dictionary to a numpy array. I then graph this histogram and return it to the main function.
- histoEqualization(pic, histo) – this function takes the previously calculated histogram and image to equalize the values. It does this by following the equalization formula found here: https://www.math.uci.edu/icamp/courses/math77c/demos/hist_eq.pdf
  Afterwards, I calculate a new image and return that to the user.
- saltAndPepper(image, strength) – this function will take an image and salt/pepper it depending on how much strength is provided. I do this by iterating through the image provided and randomly generating a number to compare with the strength. The probability of a pixel being ruined is strength/1,000,000. Whether a pixel becomes salt or pepper is a 50-50 chance.
- gaussianNoise(pic, mean, sigma) – this function applies a random gaussian normal distribution to each pixel of our image to apply noise. I learned how to do it from here: https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html
  I do this by created a nested loop and running a random gaussian function on every pixel. I combine the function results and the pixel value and place them into a new image.
- boxFilter(pict, maskSize, pixelWeight) – this function follows the sudocode found on lecture 6 slide 8/34. It uses 4 nested loops to move a mask around an image and change each pixel to match better with the pixels around it.

- smoothingFilter(pict, mask, multiplier=1) – this function follows the sudocode found on lecture 6 slide 10/34. It works similarly to boxFilter but uses varying weights in the mask instead of one continuous value.

Times:

Note that the times were collected with the the histogram diagram and image display commented out. This was done to ensure that only the function itself would be timed, and not the libraries used to display the output.

On average, each image takes 53 seconds to process through all the functions. Specifically:

- monoColor takes about 1.4 seconds
- calculating the histogram takes 2.1 seconds
- equalizing the histogram takes 0.7 seconds
- Adding salt and pepper noise takes 0.5 seconds
- Adding gaussian noise takes 12 seconds
- Applying a box filter takes 12.3 seconds
- Applying a smoothing filter takes 23.5 seconds

With 5 images, the average runtime for the overall program is 282 seconds.