E9 241 DIP - Assignment 01

September 5, 2021

Due Date: September 26, 2021 **Total Marks:** 100 + 20

Instructions:

- For all the questions, write your own functions. Use library functions for comparison only.
- Your function should take the specified parameters as inputs and output the specified results.
- Also provide the wrapper/demo code to run your functions. Your code should be self contained i.e. one should be able to run your code as-is without any modifications.
- Vectorize your code. Non-optimized code may be penalized.
- For python, if you use any libraries other than numpy, scipy, scikit-image, opencv, pillow, matplotlib, pandas and default modules, please specify the library that needs to be installed to run your code.
- Along with your code, also submit a PDF with all the results and answers to subjective questions, if any.
- Put all your files into a single zip file and submit the zip file. Name the zip file with your name.

Q1. **Contrast Enhancement:** For the images `IIScMainBuilding_LowContrast.png`, `LowLight_2.png`, `LowLight_3.png`, `Hazy.png`, `StoneFace.png`, apply the below contrast enhancement algorithms. For every image, mention which of the contrast enhancement algorithms succeeded and which of them failed. Also, explain why the algorithm succeeded or failed.

(a) Full Scale Contrast Stretching.

(b) Non-linear Contrast Stretching: Use logarithm and exponential operators (apply log or exp on each pixel intensity). Make sure the output image intensities are in the valid range.

(c) Histogram Equalization.

(d) Contrast Limited Adaptive Histogram Equalization (CLAHE): Divide the image into 8×8 non overlapping blocks (for eg, for an 80×80 image, there are 64 blocks of size 10×10. There will always be 64 blocks only). Perform Histogram equalization on each block. Also divide the image into blocks with 25% overlap. Compare both the algorithms.

For all the five algorithms, plot the image and the histogram before and after enhancement.
Function inputs: grayscale image
Function outputs: contrast enhanced images and the corresponding histograms.
. **(5+10+10+20=45M)**

Q2. **Image Upsampling:** Write a function to upsample a grayscale image by a factor of k using either nearest neighbor interpolation or bilinear interpolation. Which method gives better results? One way to quantify this is by first subsampling a given image I by a factor of k. Then upsample the subsampled image by the same factor k by using both interpolation methods. Now, compute mean squared error (MSE) between the original image and the upsampled image and compare the errors.

Perform this operation for images `StoneFace.png` and `Bee.jpg` for scaling factors $k = 2, 3$ using both interpolation methods. Plot the original and upsampled images and also provide the MSE values.
Function inputs: subsampled grayscale image, scaling factor (k) and interpolation method
Function outputs: upsampled image. **(20M)**

Q3. **Image Rotation:** Write a function to rotate the image `Bee.jpg` in the clockwise direction by 30°, 75° and in the counter-clockwise direction by 45°. Make sure the rotated image is completely visible and not cropped.
Function inputs: grayscale image, angle of rotation (in degrees, positive for clockwise and negative for counter-clockwise) and interpolation method
Function outputs: rotated image. **(20M)**

Q4. **Spatial Domain Filtering:**

(a) Mitigate the noise in the image `noisy.tif` by filtering with a square averaging mask of sizes 5, 10, 15. What do you notice with increasing mask size?

(b) Use high boost filtering to sharpen the denoised image from part (a). Choose the scaling constant for the high pass component that minimizes the mean squared error between the sharpened image and the clean image `characters.tif`.

(c) What happens if you apply high boost filtering directly on `noisy.tif` without smoothing?

Function inputs: noisy image
Function outputs: smooothed and sharpened images. **(15M)**

Q5. **Histogram Matching for Object Detection (optional bonus):** There has been a crime in a remote village and the offender has escaped the village in a car around sunset. The police did not find any clues in the village that could lead to the suspect except for a picture. While escaping, the suspect's car has been caught on a security camera installed near the road (`crime_original.png`). However, there are no other cameras for another 200 kms. The police blocked all exit routes by night, but probably the suspect has escaped by that time. Unable to solve this crime, Inspector Lestrade approached the only consulting detective available, Sherlock Holmes!

To figure out the direction in which the suspect fled, Sherlock collected images from all security cameras in the roads leading out of the village. But there are so many images and it is time consuming to check each image one by one. As you know, Sherlock is busy with other cases and nevertheless, Sherlock isn't interested in such mundane tasks. Sherlock has heard that there are some digital image processing (DIP) techniques which can identify the car automatically. Obviously, Sherlock knows nothing about image processing and has put up an advertisement for the development of an algorithm that can automatically detect the car. A DIP rookie that Lestrade knows tried to solve this using template matching (explained below). However, because the images have different lighting conditions, template matching is giving garbage results. Since you're a DIP expert, Sherlock and Lestrade are hopeful that you can come up with something better. Can you help Sherlock in this task?

Since Sherlock cannot provide all the original images, he has put up a few sample images `crime_sample1.png`, `crime_sample2.png`, `crime_sample3.png` and `crime_sample4.png`. To make job easier for you, the car image from `crime_original.png` has been extracted and provided for you (`crime_car.png`). Write a function to automatically detect the car in the sample images and put a bounding box around the car in the image (use cv2.rectangle() function). Hint: Use histogram matching before applying template matching.

Display the original image (`crime_original.png`) and the automatically detected image with a bounding box over the car for both vanilla template matching and for template matching with histogram matching. Also display the four sample images after histogram matching. Explain why template matching fails in this case and how preprocessing the image with histogram matching helps. (You need to implement both template matching and histogram matching yourself).

Function inputs: the images `crime_original.png`, `crime_car.png`, `crime_sample1.png`, `crime_sample2.png`, `crime_sample3.png` and `crime_sample4.png`
Function outputs: The images detected by vanilla template matching and histogram matching followed by template matching and the histogram matched sample images.

Template Matching: Given an image I and a pattern P, extract all possible windows W of size same as P from the image I and compute the mean squared error (MSE) between the pattern P and windows W. Whichever window W has least MSE with the pattern P is your output i.e. this window contains a pattern similar to the one in P. **(20M)**