Digital Image Processing (CSE/ECE 478)
Monsoon-2020
Assignment-2
Posted on: 09/09/2020
Due on: 07/10/2020

Introduction

1. All images in this assignment should be considered to be used as grayscale unless specified otherwise. Use the appropriate function to convert colour images to grayscale, or read the images directly as grayscale.
2. Follow the specified repository structure as in previous assignments.
3. `src` should contain the Jupyter notebooks used for the assignment.
4. `images` should contain images used for the questions.
5. Make sure you run your Jupyter notebook before committing, to save all outputs.
6. Make sure you commit and push your work regularly.

---

1. (50 points) The implementation of linear spatial filters requires moving a mask centered at each pixel of the image, computing sum of products of mask coefficients and corresponding pixels.
    1. Implement an algorithm for low-pass filtering a grayscale image by moving a $k \times k$ averaging filter of the form `ones((k,k))/(k^2)`.
    2. As the filter is moved from one spatial location to the next one, the filter window shares many common pixels in adjacent neighborhoods. Exploit this observation and implement a more efficient version of averaging filter.
    3. To appreciate the benefits of doing so, generate a plot of k vs run-time for various sized images. The plot diagram should contain a line plot for each image size you pick. Use different marker types to distinguish the default implementation and improved implementation. Just to give you a rough idea, look at `https://imgur.com/a/0HtYlTE`.
    4. Utilize the observation similar to above to implement an efficient version of a $k \times k$ median filter. Generate a plot figure similar to previous question.
    5. Using the denoising concepts discussed in the class, try to remove noise from **Noisy.jpg**

---

2. (60 points) Edge Detection

    1. Read about Canny Edge detector. Apply the Canny edge detector (use inbuilt *cv2.Canny* function for this task) to `bell.jpg`.
    2. Tweak the values of the arguments (*minVal* and *maxVal*), and report the values that give the best results for each image. (Hint: Try to detect as many edges in the bell as possible, while avoiding edges in the background.).
    3. Consider **Roberts, Prewitt, Sobel and Laplacian** filters discussed in the class. Implement and apply these filters on `kobe.png` and make observations upon comparing their outputs. Compare these with the output of Canny edge detector on the same image.

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Figure 1: Prewitt, Roberts and Sobel filters

$$\begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{-4} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \end{bmatrix} \quad \begin{bmatrix} \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{1} & \mathbf{-8} & \mathbf{1} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \end{bmatrix}$$

Figure 2: Two forms of the laplacian filter

    4. What will the 5×5 variants of **Sobel and Prewitt** filters look like? Apply these larger filters on `kobe.png` and make observations upon comparing their outputs with the corresponding smaller filters.
    5. Add noise to the input image above using Gaussian sampling. Study the effect of applying the above filters(from 2^{nd} part) on noise-affected inputs.

3. (40 points) Bilateral Filtering

    1. Implement Bilateral Filtering and apply it on **mountain.jpg**
    2. In low light imaging, images tend to be noisy and lose sharp edges. However, with flash, there is an unpleasant direct-lighting effect. Here, a **cross bilateral filter** can be used with the range and spatial filters acting on two different images.
    3. By combining a flash photograph **cake-flash.jpg** and a no-flash photograph **cake-noflash.jpg**, render a new photograph **cake-out.jpg** that has both the warm lighting of the no-flash picture and the crisp details of the flash image.

Figure 3: flash, no-flash and output images

Note: This question has to be done on RGB images. For applying filters on RGB images you can do the same filtering operation used on grayscale image for each channel (i.e., R,G,B) separately, one by one.
Reference link - `https://people.csail.mit.edu/sparis/bf_course/course_notes.pdf`

4. (30 points) In order to achieve a basic cartoon effect, all you need is essentially - a bilateral filter and some edge detection. The bilateral filter will reduce the color palette, necessary for the cartoon look, and edge detection will allow production of bold silhouettes. To get cartoonized effects following steps are needed -

    1. Apply the bilateral filtering to the color image. Keep this filtered image aside for the last step.
    2. Now, take a copy of original color image, **convert it to gray scale**.
    3. **Apply blurring** on this gray scale image to reduce noise. (Try different variants of blurring and see the performance)
    4. Now, **create an edge mask** from this blurred, gray scale image. An edge mask can be obtained using adaptive thresholding.
    5. Combine the bilateral filtered image from step 1 with edge mask(by taking bitwise AND of both at each pixel location).

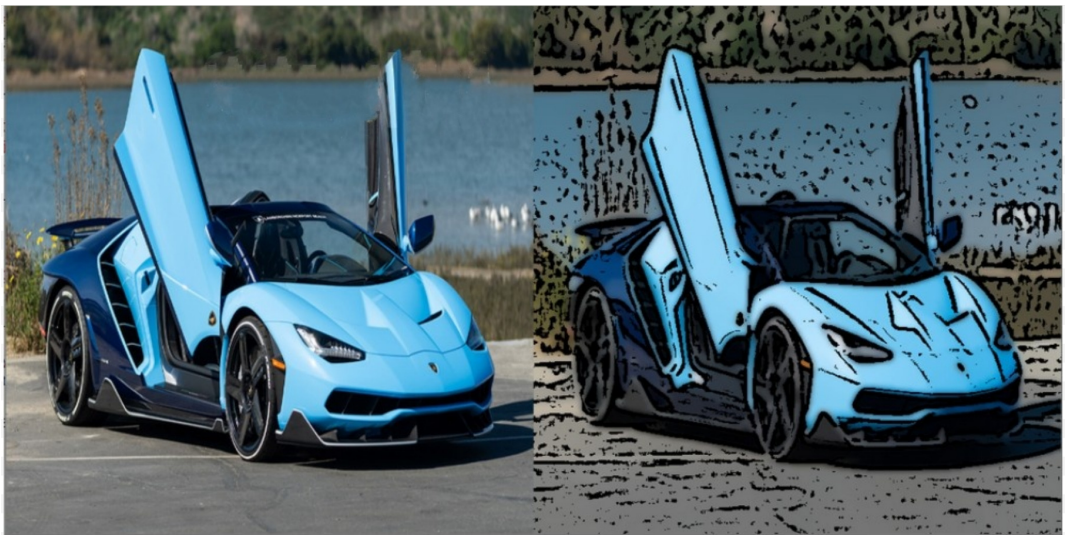

Figure 4: Sample cartoon image

Now try to do these for images of your choice. Feel free to experiment with different filters and even methods to generate even better cartoons.
Note: This question has to be done on RGB images only. But you can work on each channel separately, and then combine them for final image.

5. (40 points) Fourier Transform

    1. Implement 2D DFT.
    2. Implement 1D Fast Fourier Transform (Recursive Formulation). Use it to implement 2D FFT and display the result on suitable images of your choice. Compare the run times of your version of DFT and FFT on different sized inputs and plot them.
    3. Now, implement 2D Inverse FFT in a similar fashion.
    4. Calculate the Fourier transform of the Fourier transform of any image. You would observe that the image you get is similar to the original image. How is it different from the original image? How can you fix this in the frequency domain so that we would get the original image back instead?

6. (30 points) Pick two images f,h of same dimension (256×256). Compute their respective Fourier transforms F, H.

    1. Compute IFT(F.H) and f * h. Check whether IFT(F.H) corresponds to the center portion of f * h. Compute the average of squared difference between pixel values in IFT(F.H) and the central 256×256 portion of f * h (F.H denotes point-wise multiplication of F and H and f * h denotes convolution with padding).
    2. What changes do you observe when you zero pad the original images to dimension (511×511) and now calculate IFT(F.H) and report the new error. Choose any 64x64 image. Now, add 64 columns and rows of zeros to the right and bottom side

of the original image. Repeat this process two more times each time doubling the image size and padding the pixels on the right and bottom by zeroes. You will therefore have 4 images first one 64×64 with no zero padding and then 128×128, 256×256 and 512×512 after padding. Find the Fourier transform of all these images. Display the results and explain and justify the relationship between the four outputs you get.

7. (15 points) Denoise the given image **noisy-lena.png** and explain your process.