İTÜ

25.10.2023

Istanbul Technical University
Computer Engineering Department
BLG 453E - Computer Vision - Fall 23/24
Assignment II
Due: 31.10.2023, 23.59

Introduction & Note

In this assignment, you will be coding filtering, upsampling, downsampling, rotation, and interpolation operations from scratch.

- You should do your own work!
- You **can use** the NumPy library **only** to create arrays, change the data type, round the floating values in your array, use the trigonometric functions (sin() and cos(), etc). The other functions you are required to code should be written from scratch.
- You can work as a group of **up to 2 students** for the assignments. Make sure to include your name, surname, and student IDs in your report.

Q1 - Image Filtering

In Question 1, you will be filtering RGB and Grayscale images with box filters in the shape of 3x3 and 5x5. The box filters take the mean around a pixel with a given shape of region. A sample figure with input and output images is given in Figure 1.

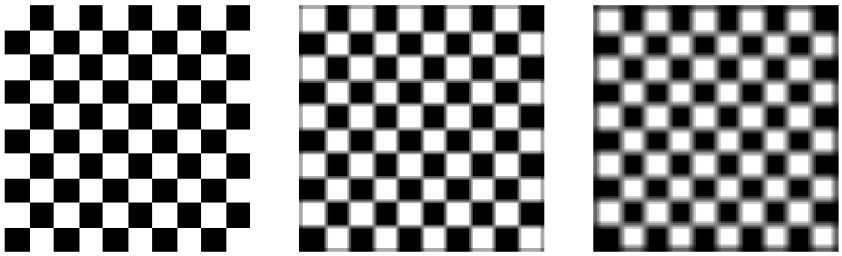

Figure 1: The sample Grayscale image, 3x3 and 5x5 box filtered outputs, respectively.

For this task,

- Choose an RGB image (try to choose a low-resolution image to see the blurry effects better) to process, and read it. (Use the *OpenCV* library to **read** the image. **Make sure to convert into RGB first.**)
- Convert the RGB image into grayscale with the NTSC formula as in Homework 1 and convert the resulting image into the **unsigned integer 8-bit** data format.
- Filter the RGB and grayscale images using 3x3 and 5x5 box filters. For RGB images, filter each channel separately and merge afterward. Comment on the filtered RGB and grayscale images in your report. Zero-pad your images so that the final shape does not change and set stride as 1. (What looks different in the filtered RGB images than in the input? Why do you think that happened? Comment on it.)
- Save the original RGB, grayscale images and the filtered outputs into separate NumPy arrays (*.npy*).

- Add the input and output images to your report. Explain the steps and the impact of the filters, the change in the filtered RGB and grayscale images, etc.
- Upload the zipped folder that has the input image, output images, the NumPy arrays of the input and output images, and the code file (*.py* or *.ipynb*).

Q2 - Image Upsampling

In Question 2, you will be implementing an upsampling algorithm that takes a grayscale image, upsamples it by two times its original size (2x), and linearly interpolates in a 3x3 region, as shown in Eq. 1. A sample figure with input and output images is given in Figure 2. Please, see the dimensions of the input, upsampled, and interpolated images in the figure.

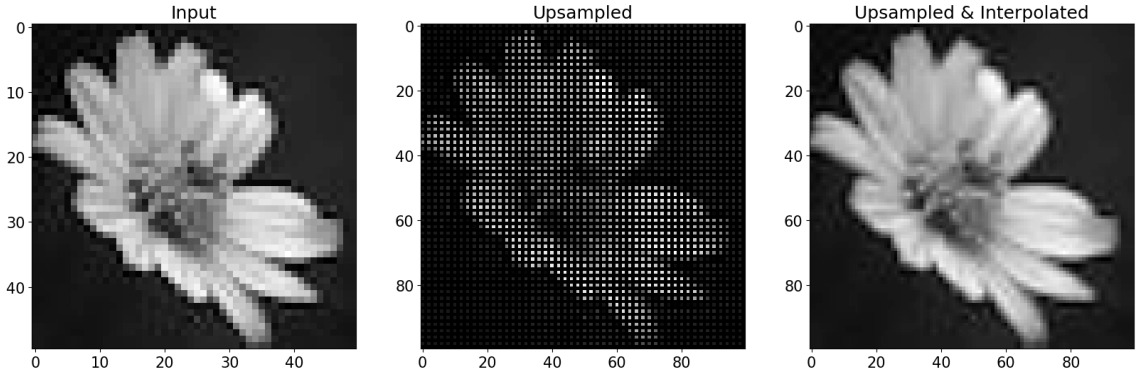

Figure 2: A sample upsampling process. (First) A sample input image in grayscale format, (Second) the upsampled (2x) grayscale image, and (Third) linearly interpolated output.

$$\text{Mask} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \frac{1}{9} \tag{1}$$

For this task,

- Choose an RGB image (try to choose a low-resolution image, or reduce the resolution of the image before processing) to process and read it. (Use the *OpenCV* library to **read** the image.)
- Convert the RGB image into grayscale with the NTSC formula as in Homework 1 and convert the resulting image into the **unsigned integer 8-bit** data format.
- Upsample the image by a factor of 2 by adding rows and columns with the value of 0. (Please, see the second image in the figure).
- Perform linear interpolation in a region of 3x3 mask (Equation 1). Similar to the convolution with a box filter, you should modify only the unknown pixels (with the value of 0), and get the mean of the known pixels only for each stride. Use zero-padding and stride as 1. The pixel values from the original image **should be kept the same**. Do not forget to convert the resulting image into the **unsigned integer 8-bit** data format.
- Save the original RGB image, the upsampled output, and the interpolated outputs as separate NumPy arrays (*.npy*).
- Add the input and output images to your report. Explain the steps, comment on the linear interpolation, etc.
- Upload the zipped folder that has the input image, output image, the NumPy arrays of the input and output images, and the code file (*.py* or *.ipynb*).

Q3 - Image Downsampling

In Question 3, you will be implementing a downsampling algorithm that takes a grayscale image, filters it by using the 3x3 box filter, and then downsamples it by two times its original size (2x). A sample figure with input and output images is given in Figure 3. Please, see the dimensions of the input, and the downsampled images in the figure.

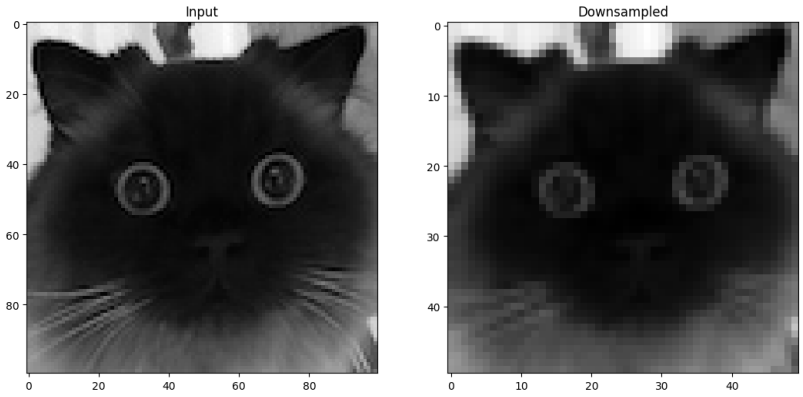

Figure 3: A sample downsampling process. (First) A sample grayscale image, (Second) the box-filtered and downsampled (2x) output.

For this task,

- Choose an RGB image (try to choose a low-resolution image, or reduce the resolution of the image before processing) to process and read it. (Use the *OpenCV* library to **read** the image.)
- Convert the RGB image into grayscale with the NTSC formula as in Homework 1 and convert the resulting image into the **unsigned integer 8-bit** data format.
- Filter the image using a 3x3 box filter (use zero-padding and stride as 1).
- Downsample the image by a factor of 2 by removing rows and columns. (Please, see the second image in the figure).
- Save the original RGB image and the downsampled output as separate NumPy arrays (*.npy*).
- Add the input and output images to your report. Explain the steps, comment on the reason behind filtering the image before performing downsampling, etc.
- Upload the zipped folder that has the input image, output image, the NumPy arrays of the input and output images, and the code file (*.py* or *.ipynb*).

Q4 - Image Rotation

In Question 4, you will be implementing a rotation algorithm that changes the orientation of the image counter-clockwise in 45, 90, 135, and 180 degrees. In addition to the rotations, you will need to perform linear interpolation (as in Q2) to 45 and 135-degree rotated images. A sample figure with input and output images is given in Figure 4.

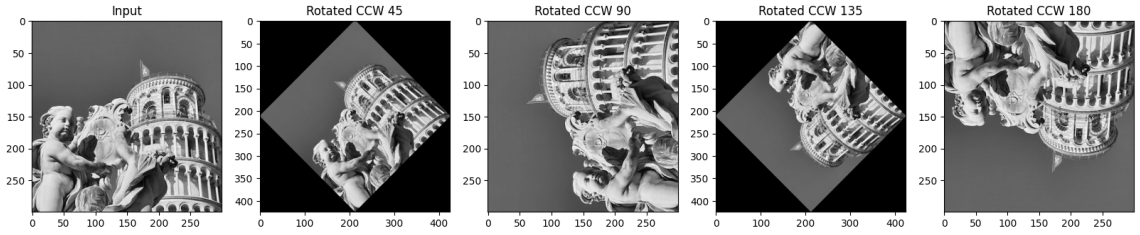

Figure 4: A sample rotation process. (First) A sample grayscale image and (Second to Fifth) counter-clockwise rotated images in 45, 90, 135, and 180 degrees, respectively.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} \tag{2}$$

For this task,

- Choose an RGB image and read it. (Use the *OpenCV* library to **read** the image.)
- Convert the RGB image into grayscale with the NTSC formula as in Homework 1 and convert the resulting image into the **unsigned integer 8-bit** data format.
- Rotate the images counter-clockwise with 45, 90, 135 and 180 degrees. You should code the transformation operation from scratch. First, determine the width and height of the resulting image and create an empty array to fill. Then, while iterating over the empty array, find the pixel locations in the original image that belong to each empty pixel in the new image by using the transformation matrix as shown in Eq. 2. In the equation, θ is the degree of rotation, x and y are the pixel coordinates of the image that is being iterated over, and x' and y' are the pixels belonging to the original image. You will need to check if the x' and y' are valid coordinates and have a pixel intensity value in the original image. Otherwise, assign 0 in the new array.
- Perform linear interpolation with the mask in Eq. 1 to the images rotated with 45 and 135 degrees counter-clockwise to eliminate the discontinuity in the images.
- Save the original RGB image, rotated outputs, and interpolated outputs as separate NumPy arrays (*.npy*).
- Add the input and output images to your report. Explain the steps, comment on the transformation, the difference when you apply linear interpolation, etc.
- Upload the zipped folder that has the input image, output image, the NumPy arrays of the input and output images, and the code file (*.py* or *.ipynb*).