Image processing Task No. 1

Task variant N2

Tuesday 15:15-16:45

Winter semester 2020/2021

Maria Golińska

Wojciech Krasnowski

Technical description of the application

Description

Image modification program. It's run via command line. You should enter the path to the image and the action to be performed. The program will return the modified photo. To import and export the image, CImg library was used. We load the pixel value distinguishing it into R, G and B channels in this way: image(x, y, 0) for Red, image(x, y, 1) for Green and image(x, y, 2) for Blue.

Description of implementation of basic image operations

- Brightness (B1)

  Via command line program imports the image as well as the parameter. Pixel values for each channel R, G and B are read to variables valR, valG and valB. Next, temporary variables newR, newG and newB are created, and brightness function values are thrown into them. In function, "if" condition is used to brighten or darken each pixel depending on the given parameter.

- Contrast (B2)

  It obtains parameter from command line (input and output path as well) and then creates variable of type float. It calculates contrast correction factor and then performs actual contrast adjustment given by following formula: R '=F(R-128)+128 The value of each pixel component is then saved into separate variables, its value is changed with the use of formula above and finally put through check() function. It checks whether any pixel value is in the suitable range (0-255). Changed value is put in correct place back to the initial image. All of it happens inside two for loops, which guarantee that we go through whole image. Finally, saved to output file.

- Negative (B3)

  Via command line program imports the image. Pixel values for each channel R, G and B are read to variables valR, valG and valB. Next, temporary variables newR, newG and newB are created and are assigned the value "255 – val". It reproduces the bright portions of image as dark and the dark parts as light areas.

- Horizontal flip (G1)

  Via command line program imports the image. Firstly, a blanc image fliph is created of the same size as imported image. Pixel values for each channel R, G and B are read to variables valR, valG and valB. Next we assign each pixel of image to fliph with slight change; we enter the values of x in reverse order leaving y in the same place (fliph((image.width()-1) -x, y, 0)).

- Vertical flip (G2)

  Via command line program imports the image. Firstly, a blanc image fliph is created of the same size as imported image. Pixel values for each channel R, G and B are read to variables valR, valG and valB. Next we assign each pixel of image to fliph with slight change; we enter the values of y in reverse order leaving x in the same place (flipv(x, (image.height()-1) - y,0)).

- Diagonal flip (G3)

  Via command line program imports the image. Firstly, a blanc image fliph is created of the same size as imported image. Pixel values for each channel R, G and B are read to variables valR, valG and valB. Next we assign each pixel of image to fliph with slight change; we swap x with y (flipd1(y, x, 0)). Another diagonal flip can be represented as combination of previous flips: horizontal and vertical (flipd2((image.height()-1) – y, (image.width()-1) -x, 0)).

- Shrinking (G4)

  New image is created with smaller height and width, then four arrays are created for each colour component. Inside two for loops, the values of 4 pixels in the neighbourhood are read into the suitable arrays, then the average values are calculated. Now calculated average values are put into created image with smaller height and width. If statements ensure, that right values go into corresponding place. Finally, the image is saved.

- Enlargement (G5)

  New image is created with greater height and width. Inside two for loops, the values of each pixel components are read into variable and put into right place in created image. If statements assure us that the value of initial pixel is put into 4 pixels in new image. It is then saved.

Description of implementation of noise reduction methods

- **MAXIMUM FILTER:**

New image is created as well as four arrays for each pixel components. Inside two for loops, the program reads information about neighbour pixels (3x3) and puts it into corresponding places in arrays. Next, insertion sort is used in order to sort arrays. Lastly, the maximum value of each component of the pixel is put into new image. Then it's saved.

Image processing Task No. 1

- **MINIMAL FILTER:**

New image and four arrays for each pixel components are created. Inside two for loops, the program reads information about neighbourhood pixels (3x3) and puts it into corresponding places in arrays. Next, insertion sort is used in order to sort arrays. Lastly, the minimum value of each component of the pixel is put into new image. Then it’s saved.

- **MEDIAN FILTER:**

New image is created as well as four arrays for each pixel components. Inside two for loops, the program reads information about neighbour pixels (3x3) and puts it into corresponding places in arrays. Next, insertion sort is used in order to sort arrays. Lastly, the median (as long as we have 3x3 area, we have 9 pixels to analyse, median of odd amount of values would be just the middle one) of each component of the pixel is put into new image. Then it’s saved.

Analysis of parameters of the noise reduction methods

Image quality after median filter has clearly improved. Thanks to this method of noise removal the image smoothed out. The median used for a given pixel group has compensated for any imperfections in the designated area. The only parameter we can pass here, is actually the amount of pixels we take median from. In our case (3x3 matrix) the output image was rather sharp, and we get rid of noise pixels. Using bigger matrix can cause greater deformation of the output image simply because into new image we load less diversity of pixel values. Unfortunately, minimal and maximum filters were completely ineffective. Instead of removing damaged pixels, they made them even more visible. Same as above, if we consider bigger matrix there would be less diversity in pixel values and the output image would be even worse.

Analysis of the noise reduction methods w. r. t. the possible applications for various types of noise

Next we wanted to analyze the noise reduction filters using various methods such as Mean square error (MSE), Peak mean square error (PMSE), Signal to noise ratio (SNR), Peak signal to noise ratio (PSNR) and Maximum difference (MD). For the purposes of the report, we present the results of given filters obtained using the MSE method. We used 3 varieties of noisy photos: Impulse2, Unifrom2 and Normal2. It’s easy to notice that the Median filter worked very well. The difference in pixel values between original and modified photo has decreased dramatically. Max Filter and Min Filter completely didn’t cope with the task. The difference in pixel values has clearly increased.

Noise type	**MEDIAN FILTER(MSE)**		**MAX FILTER(MSE)**		**MIN FILTER(MSE)**	
	before	**after**	**before**	**after**	**before**	**after**
Impulse2	**783.115**	**75.4151**	**783.115**	**3341.26**	**783.115**	**3646.03**
Uniform2	**613.503**	**84.2733**	**613.503**	**2521.63**	**613.503**	**2527.49**
Normal2	**657.241**	**128.824**	**657.241**	**1972.95**	**657.241**	**2026.56**

In conclusion, given analysis methods of noise reduction filters come in handy. In this way, it’s easy to see if a given filter worked correctly and improved the quality of the image.

Teacher's remarks