Lab 2 - Filters and histogram equalization

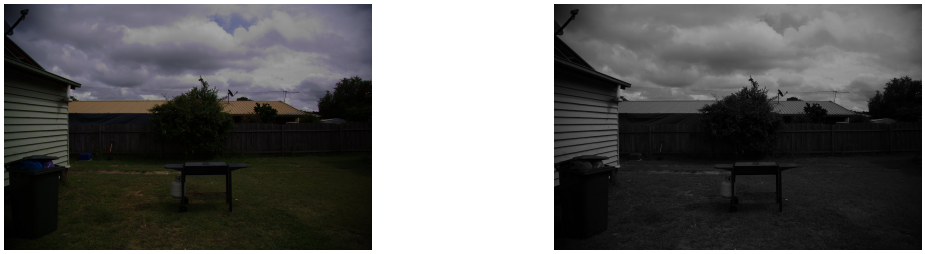
Computer Vision

Alberto Pasqualetto, 2121718

24 March 2024

Task 1

The first task is pretty straightforward, it asks to convert a provided image into grayscale and then save it. An example at figure 1.



(a) Example of original image

(b) Grayscale image

Figure 1: Example of conversion to grayscale

Task 2

Task 2 requires to implement a max and a min filter as functions. They are implemented taking an image as `cv::Mat` and a provided kernel size as input. In order to apply the kernel on every single pixel, a new image with a padding of `kernelSize/2` is used to compute the result. The padding is set to the maximum value of the image (255 assuming an `uchar` type) for the min filter and to the minimum value (0) for the max filter, so that the filter will not be affected by the border.

In "Lena_corrupted" image, min and max filters behave very badly, as they are not able to recover the original image. If both filters are applied one after the other the results are still bad, but they give a better visualization of the corrupted image. The second filter can also have a bigger kernel. Order of application only changes the final brightness. The results are shown in figure 2.

(a) Corrupted image

(b) Min filter (kernel 3)

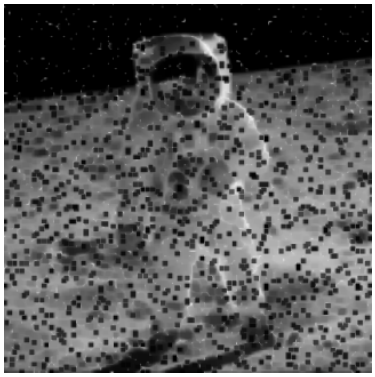
(c) Min (kernel 3) then max (kernel 3) filter

(d) Max (kernel 3) then min (kernel 5) filter

Figure 2: Min and max filter on corrupted Lena

In "Astronaut_salt_pepper" image, with salt and pepper noise, applying a single filter, makes the image even worse (kernels greater than 3 are even worse). Applying both filters one after the other the salt (min, then max) or the pepper (max, then min) is almost removed and the other is not affected. If using larger kernels in the second pass (5 is the sweet spot), the image content is more clear since it has less white/black peaks, but it is more blurry and many small squares can be seen. The results are shown in figure 3.

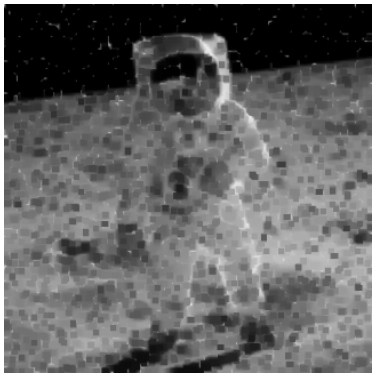
(a) Corrupted image



(b) Min filter (kernel 3)

(c) Max filter (kernel 3)

(d) Min (kernel 3) then max (kernel 3) filter



(e) Max (kernel 3) then min (kernel 5) filter

Figure 3: Min and max filter on corrupted Astronaut

In "Garden" image the aim is removing the electric cables. Applying a max filter with kernel 3 removes all but one cables without any artifact, with kernel 5 it removes all cables introducing some artifacts especially on the tree; if a min filter with kernel 3 is applied after the max with kernel 5, the results are even better, reducing artifacts on the tree. If min filter is applied first, electric cables are not removed because cables are black. therefore they are the minimum value when kernel is applied; increasing kernel size introduces too much artifacts. The results are shown in figure 4.

(a) Original image (cropped)

(b) Max filter (kernel 3) (cropped)

(c) Min filter (kernel 3) (cropped)

(d) Max filter (kernel 5) (cropped)

(e) Max (kernel 5) then min (kernel 3) filter (cropped)

Figure 4: Min and max filter on Garden

Task 3

Median filter is implemented as a function that takes an image as `cv::Mat` and a kernel size as input, for each kernel to be applied, it sorts the values pushing them to a `std::vector` and takes the median one.

Median filter performs well in removing noise: in "Lena_corrupted" kernel 3 gives a better representation than min and max filters (See figure 5a) and kernel 5 trades noise with sharpness (See figure 5b); in "Astronaut_salt_pepper" kernel 3 works, but it works similar to two pass min-max filter with kernel 3 (See figure 5c).

Median filter does not help at removing electric cables in "Garden" with any kernel (See figure 5d).

(a) Median filter (kernel 3) on corrupted Lena

(b) Median filter (kernel 5) on corrupted Lena

(c) Median filter (kernel 5) on corrupted Astronaut

(d) Median filter (kernel 3) on Garden (cropped)

Task 4

Gaussian filter is applied using `cv::GaussianBlur`.

It trades noise with blur increasing kernel size (See figures 6a to 6f). It does not remove electric cables in "Garden" except for high kernels where everything is very blurred. (See figures 6g to 6i)

(a) Gaussian filter (kernel 3) on corrupted Lena

(b) Gaussian filter (kernel 21) on corrupted Lena

(c) Gaussian filter (kernel 41) on corrupted Lena

(d) Gaussian filter (kernel 3) on corrupted Astronaut

(e) Gaussian filter (kernel 21) on corrupted Astronaut

(f) Gaussian filter (kernel 41) on corrupted Astronaut

(g) Gaussian filter (kernel 3) on Garden (cropped)

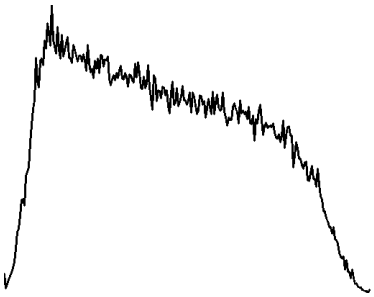
(h) Gaussian filter (kernel 21) on Garden (cropped)

(i) Gaussian filter (kernel 41) on Garden (cropped)

Figure 6: Gaussian filter

Task 5

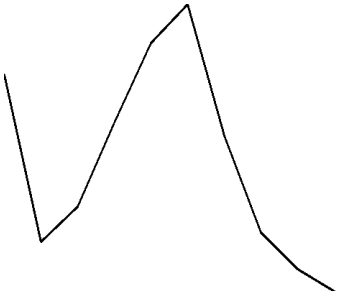
Histograms are generated from grayscale images (1 channel) using `cv::calcHist`. Histograms change based on the number of bins used ($[0, 255]$ for `uchar` images, for each channel): histograms with 255 bins represent the frequency for each single gray-level (Figures 7a, 7c and 7e); using less bins (10 in the example) levels are grouped together (in the example groups of 255/10) (Figures 7b, 7d and 7f).



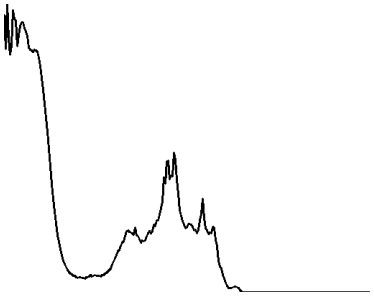
(a) Histogram with 255 bins, corrupted Lena

(b) Histogram with 10 bins, corrupted Lena

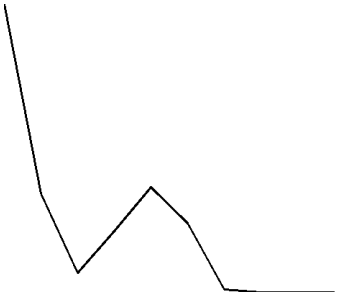
(c) Histogram with 255 bins, corrupted Astronaut



(d) Histogram with 255 bins, corrupted Astronaut



(e) Histogram with 255 bins, Garden



(f) Histogram with 255 bins, Garden

Figure 7: Histograms

Task 6

Histogram equalization is applied using `cv::equalizeHist`. Results are shown in figure 8.

(a) Histogram equalization (255 bins), corrupted Lena

(b) Histogram equalization (255 bins), corrupted Astronaut

(c) Histogram equalization (255 bins), Garden

Figure 8: Histogram equalization applied on images