Image Analysis - TP2 - Local Filtering and Histograms

Jander Nascimento, Raquel Oliveira

February 26, 2011

1 Filtering

1.1 Binomial

Binomial filter uses Pascal's triangle to create a filter. This filter has the blurry effect with some differences. One example of a 3x3 Binomial filter can be seen in the Figure 1.

1	2	1
2	4	2
1	2	1

Figure 1: 3 x 3 Binomial filter

To apply this kernel to the image, it is necessary to perform the convolution in the original image with this kernel.

This kernel is used to remove the noise in image, but it do not preserves the edges of the image.

1.2 Median

With a purpose of noise reduction, in certain situations it can preserve the edges while reduce the noise of the image.

For this reason this kind of filter is regularly used as a pre-treatment for edge detection.

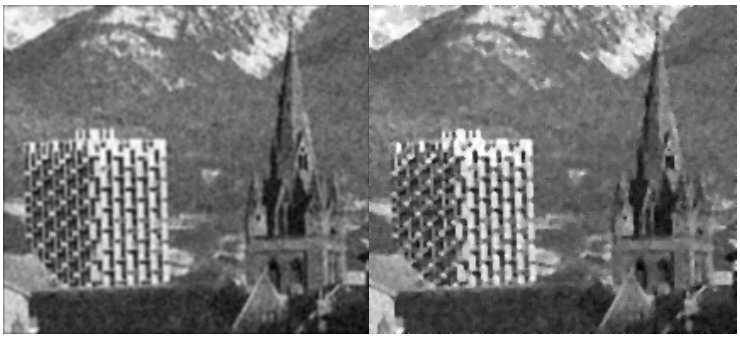
The Median filtering differs from others filters in the way it is applied to the image. While other filters are defined by kernels, the Median filter depends on each sector of the image analyzed to generate a new kernel.

Median kernel copies a certain sector of the image analyzed and change the value of the central pixel in the original image.

The central pixel in the original image receives the median value of the kernel. It is possible to do it by simply sorting the content of the kernel and

assign the value in the middle (central position of the kernel) to the original image.

1.3 Binomial *versus* Median



(a) Binomial (b) Median

Figure 2: Removing noise

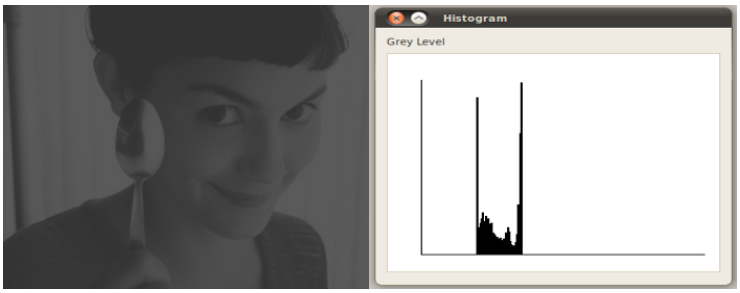
In the Figure 2 we applied Binomial and Median filter separately. Median preserves the edges better than the Binomial, although the Binomial preserves some central details in the image. This behavior can be seen in the window at the chapel, in the Binomial is possible to see the borders of the line, while in the Median this detail is not displayed.

2 Histogram

The histogram is an important tool when dealing with image analysis. It shows the color distribution of a certain image regarding to its color intensity. The histogram is represented in a Cartesian, where x axis is the color intensity and y represents the number of pixels that have such intensity.

In this practical work, we wrote a function in c that computes the intensity histogram. The function has the following steps:

- take from the main argument the name of the file that will be analyzed.
- create a matrix (with the dimensions of the image) to store the pixels' intensity of the image
- go through such matrix and count the number of pixels in each intensity at the intensity of the color (from 0 to 255)
- store that in an array of 256 positions.
- display such array on the standard output device (screen)



(a) Image (b) Histogram

Figure 3: Histrogram of an image

2.1 Stretching

The histogram stretching is a technique by which the color histogram is used to evaluate and possibly change the color intensity range. This enhances the detail level of some images that might appear too dark or too bright. This is done by spreading the pixels to use the entire color intensity.

In this practical work, we wrote a function in c that transforms an image with the histogram stretching. The function has the following steps:

- take from the main argument the name of the file that will be analyzed.
- create a matrix (with the dimensions of the image) to store the pixels' intensity of the image
- in the matrix, find the current range minimum of the image($current_{min}$), which is the minimum intensity of the intensity of the color that is used in the image
- find also the current range maximum of the image($current_{max}$), which is the maximum intensity of the intensity of the color that is used in the image
- for each pixel of the image, calculate its new intensity based on the stretching formula:

$$y[n] = new_{min} + \frac{new_{max} - new_{min}}{current_{max} - current_{min}} * (x[n] - current_{min}) \quad (1)$$

  where:

  - n is the pixel
  - new_{min} is 0;
  - new_{max} is 255;
  - $x[n]$ is the current intensity of the pixel.

(a) Original Image (b) Stretched Image (c) Stretched Histogram

Figure 4: Stretching Transformation

2.2 Equalization

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. This method usually increases the global contrast of many images, especially when the usable data of the image is represented by close contrast values. Through this adjustment, the intensities can be better distributed on the histogram. This allows for areas of lower local contrast to gain a higher contrast. Histogram equalization accomplishes this by effectively spreading out the most frequent intensity values.

In this practical work, we wrote a function in c that transforms an image with the histogram equalization. The function has the following steps:

- compute the histogram of the image (following the steps of the question before)
- create an array to store the cumulative distribution function of the histogram, which is the values of the histogram in a cumulative way.
- in such array, find the minimum value at all the intensity of the color (cdf_{min})
- for each pixel of the image, calculate its new intensity based on the equalization formula:

$$h[v] = round\left(\frac{cdf(v) - cdf_{min}}{(M * N) - cdf_{min}} * (L - 1)\right) \tag{2}$$

  where:

  - v is the current intensity of the pixel
  - *cdf(v)* is the value of such intensity in the array that stores the accumulated values of the histogram
  - M is one height of the image

– N is the width of the image

– L is the number of color levels used (in most cases, 256)

(a) Original Image (b) Equalized Image (c) Equalized Histogram

Figure 5: Equalization Transformation

2.3 Where to find the source code?

The code is hosted at `https://jfimageanalysis.googlecode.com/svn/trunk/TP2/`. To download the code execute this command:

svn checkout https://jfimageanalysis.googlecode.com/svn/trunk/TP2/

Remark: you **must** have installed subversion.

2.4 How to execute?

First compile the source (using *make*). The application name is *imagetransform*

Here are some examples:

Apply binomial filter to an image:

```
imagetransform -f binomial -i image_in.pgm > image_out.pgm
```

You still can combine this command to specify the size of the kernel (-s) or the number of times that the filter should be applied (-n)

Apply median filter to an image:

```
imagetransform -f median -i image_in.pgm > image_out.pgm
```

And of course you can always type *imagetransform –help* to check for more options.