TP1: Basic Image Processing

Martino Ferrari

1 Introduction

In the first exercise we experimented with some basic image commands and operations, for this exercise, and as well for the others, I used Octave instead of MatLab.

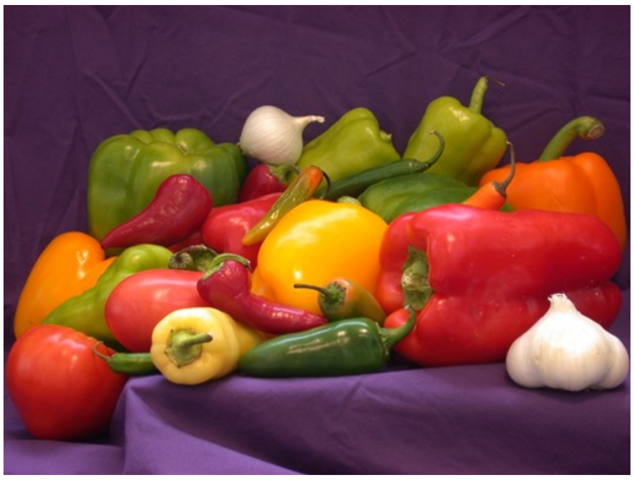


Illustration 1: original "peppers.png"

The image shown in illustration 1 is the original MatLab "*peppers.png*" picture used in all the following steps of the exercise.

The color histogram of the picture is the following:

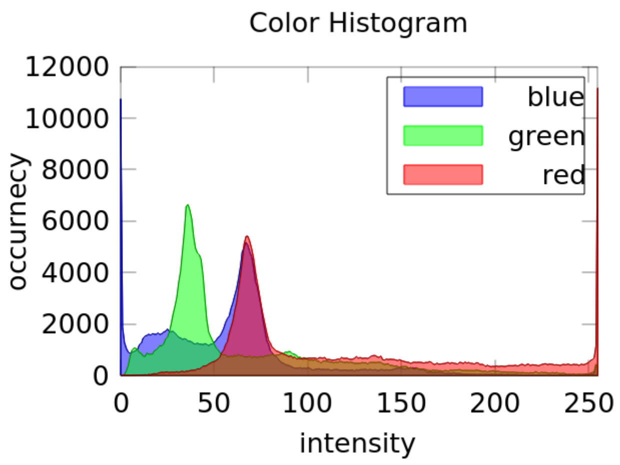


Illustration 2: Color histogram of "pepper.png"

In the illustration 2 is possible to see that red channel is qute dominating the picture (~11000 pixels with 255 as value) while the blue is the less represented (~11000 pixels with 0 as value), this is also qualtiatvely possible to see this color distribution.

In the following illustration I show the image in grey scale and it's luminosity histogram:

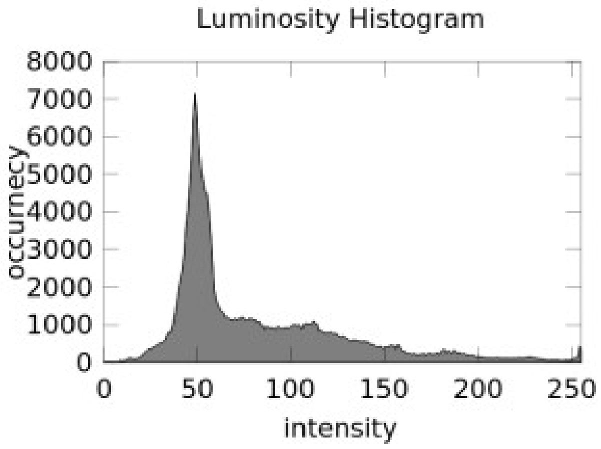
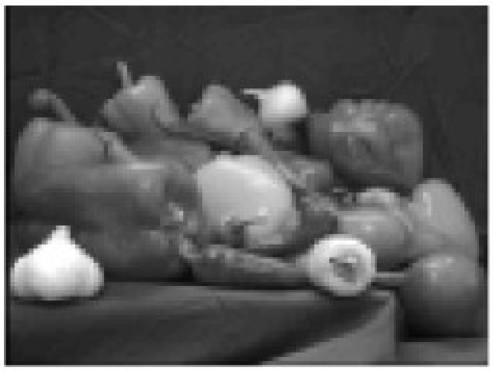


Illustration 3: grayscale picture and its luminosity histogram

From its histogram it's possible to see that the picture is well exposed, as there are just few pixels over exposed (I suppose near the garlic) and close to none under exposed. The global mean and variance of the gray scale picture are:

$$\mu=81.66 \qquad \sigma^2=2169.2$$

In the last step of the exercise we were asked to compute and show the local mean and variance of the grayscale version of "*peppers.png*", the window size used is 5x5 pixels:



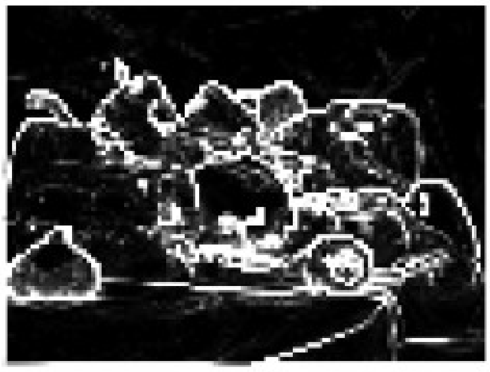


Illustration 4: local average and local variance of gray scale "peppers.png"

As expected the local average image results in a sort of smaller size of the original image, while the local variance image show that the variance increase greatly in the edges of the objects.

2 Noise

In this second exercise we will study how different types of noise will affect both the image quality and the metrics of it. In particular we will implement Gaussian and Salt & Pepper noise as well as the Mean Squared Error and the Peak Signal to Noise Ratio metrics.

As base picture I used again the “*peppers.png*”.

The results of Gaussian noise and Salt & Pepper noise on the image is shown in the following illustration:

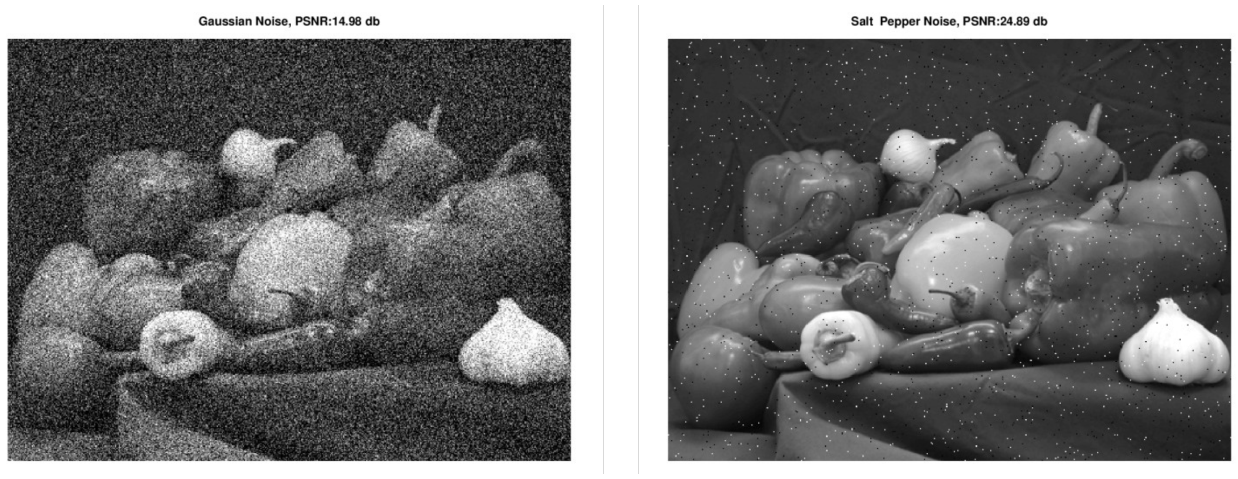

Gaussian Noise, PSNR:14.98 db
Salt Pepper Noise, PSNR:24.89 db

Illustration 5: 15 db PSNR Gaussian noise and 25 db PSNR Salt & Pepper noise

After implementing the metrics I was able to compare first a “double” version of the image with the original “uint8” version with the MSE metrics, this gave me as result an Mean Squared Error of 8768, this due the fact that my MSE implementation doesn’t take in account the different information representation.

After implementing the PSNR metrics I was able to compare different noise configuration:

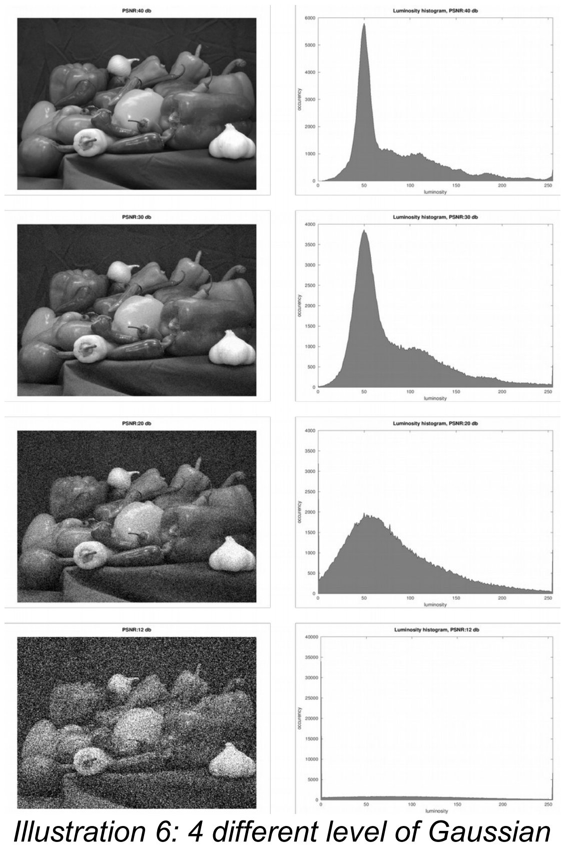


Illustration 6: 4 different level of Gaussian noise

In this illustration is possible to see the effect of the Gaussian noise on the luminosity histogram and on the picture itself.

The PSNR pass from 40db in the first picture to 10db in the last one. While the image is still recognizable in all the 4 cases the histogram change complitly.

In the first case (40db) and in the second case (30db) the histogram is very similar to the original one (in illustration 3). In the other two cases (20db and 10db) the histogram become more and more flat with close to any resemblance to the original one.

This is due to the fact that the noise energy has become higher than the signal itself.

Finally we were asked to compare the Gaussian and Salt & Pepper noise at the same PSNR of 40 db:

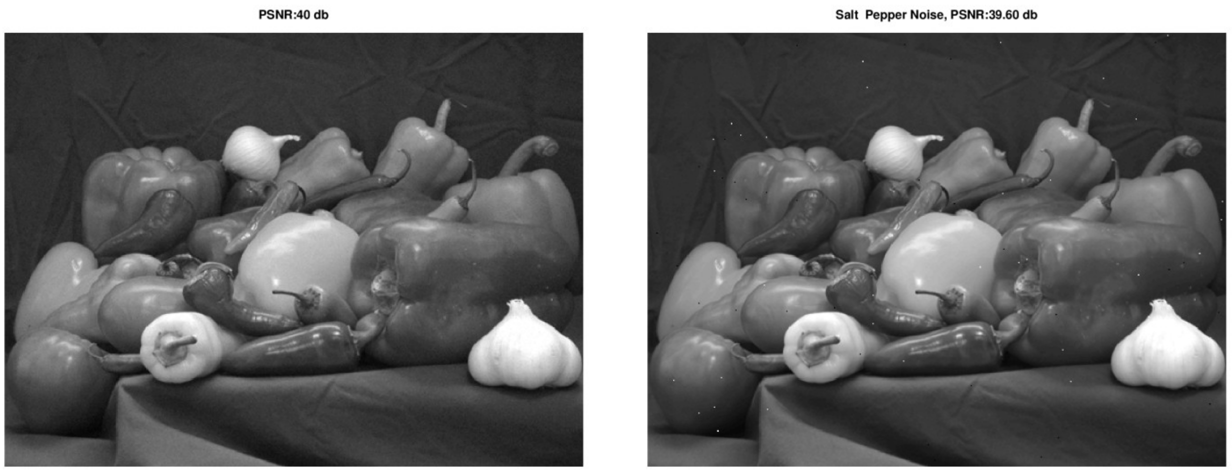


Illustration 7: Gaussian noise and Salt & Pepper noise at a PSNR of 40db

In this illustration is possible to see how the salt & pepper noise is visually much more disagreeably than the Gaussian noise, in the fact where the Gaussian noise add a small random value to all the pixels while the salt & pepper totally delete (saturating or obscuring) some pixels information.

3 Identification

In this last exercise we were asked to implement a simple image hash algorithm for 256x256 pixels images.

This perceptual hash will return a 64 bits key, where each bit represents a 32x32 pixels window. The value of the bit is 1 if the window luminosity average is higher than the global average , otherwise 0.

After implementing this simple algorithm and hashing 200 images we were asked to compare the hashes of this images using the hamming distance and the probability of error previously implemented.

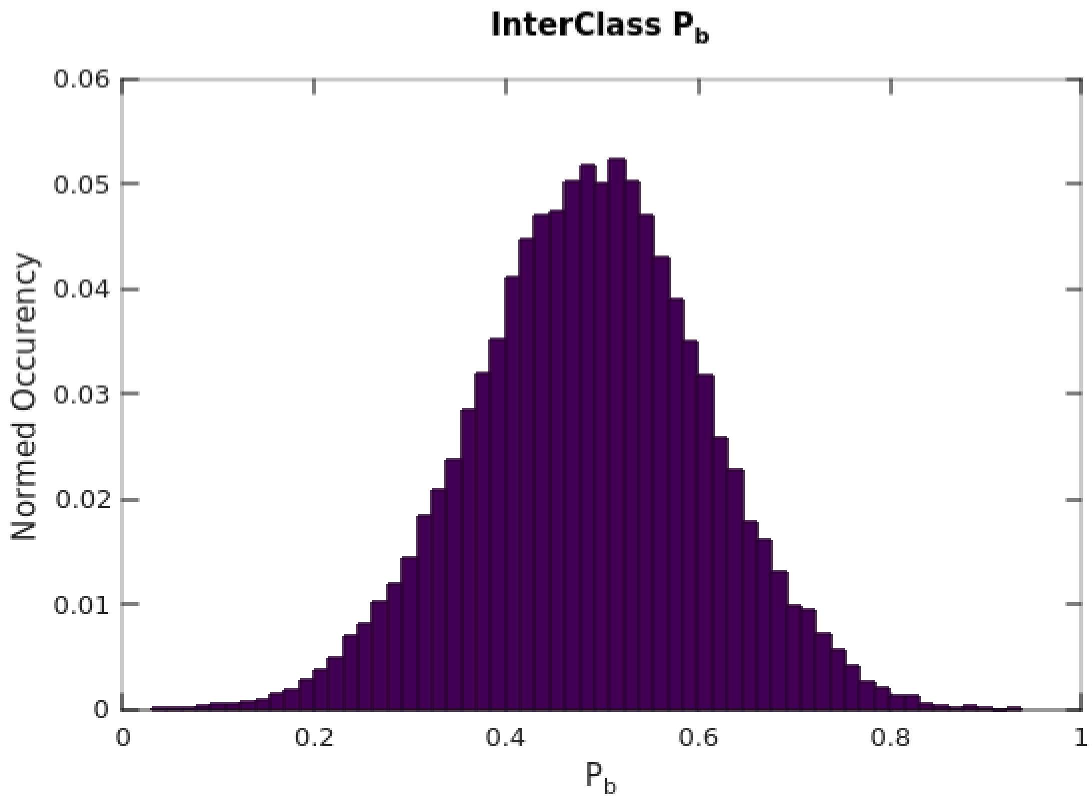


Illustration 8: Probability of Error of the 200 pictures

In the illustration 7 is shown the distribution of the inter-class probability of error. As predictable the distribution is centered in 0.5 (means half of the bits of the hash are different) and it follows a Gaussian distribution.

This result is good, as it means that the generated hash identify well the pictures.

After this first inter-class comparison we were asked to compare the hashes of the original images with the one of the degraded images (using PSNR of 35db and Gaussian noise).

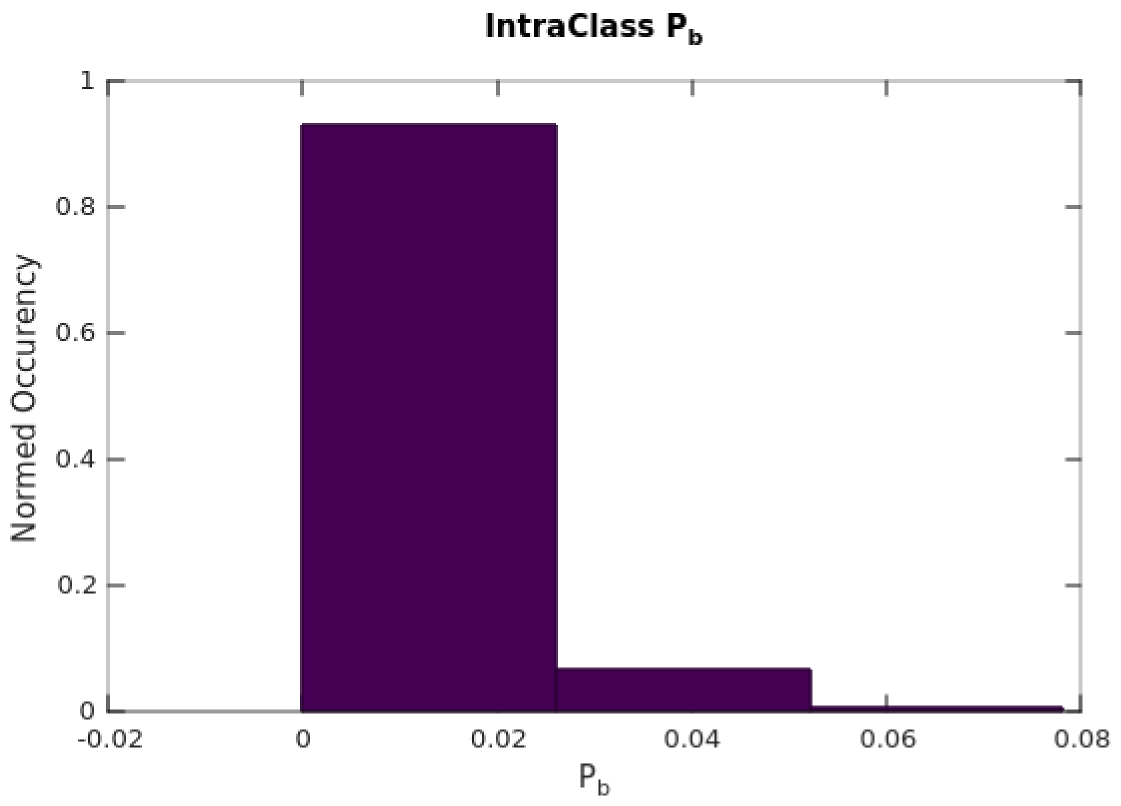


Illustration 9: Probability of Error of the intra-class analysis

This results is also very good, the histogram shows that big majority of the comparisons of the original pictures with their degraded version has 0 probability of error (meaning all the bits of the hash are the same).

This is due that the average and global mean of the pictures are statistically affected in the same way from the Gaussian noise, and so the ratio between the local average and the global average remains constant.

In conclusion this simple perceptual hash is performing well against noise (and compression), as is capable of recognizing the degraded images, however still it has many limitations:

- Image resolution changes
- Rotations and deformations
- Exposition changes

I believe that more sophisticated hashes algorithms could address this limitations.