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:

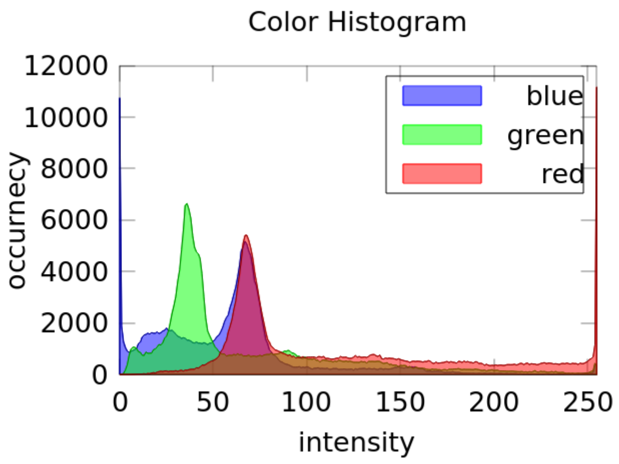


Illustration 2: Color histogram of "pepper.png"

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:

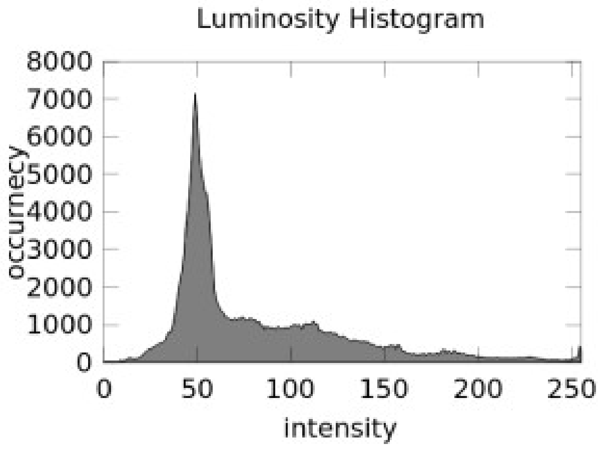
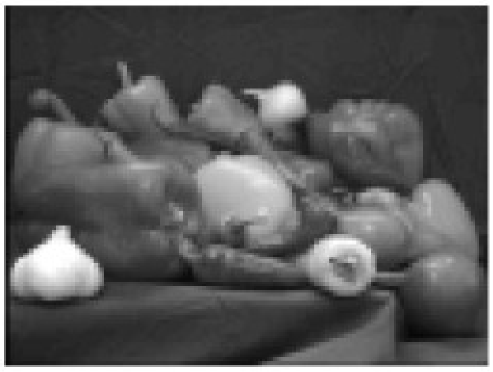


Illustration 3: grayscale picture and its luminosity histogram

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:



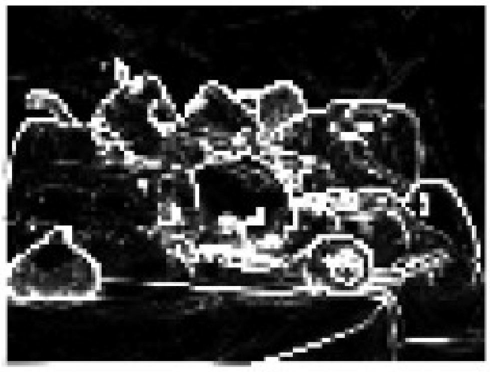


Illustration 4: local average and local variance of gray scale "peppers.png"

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.

In this second part of the 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:

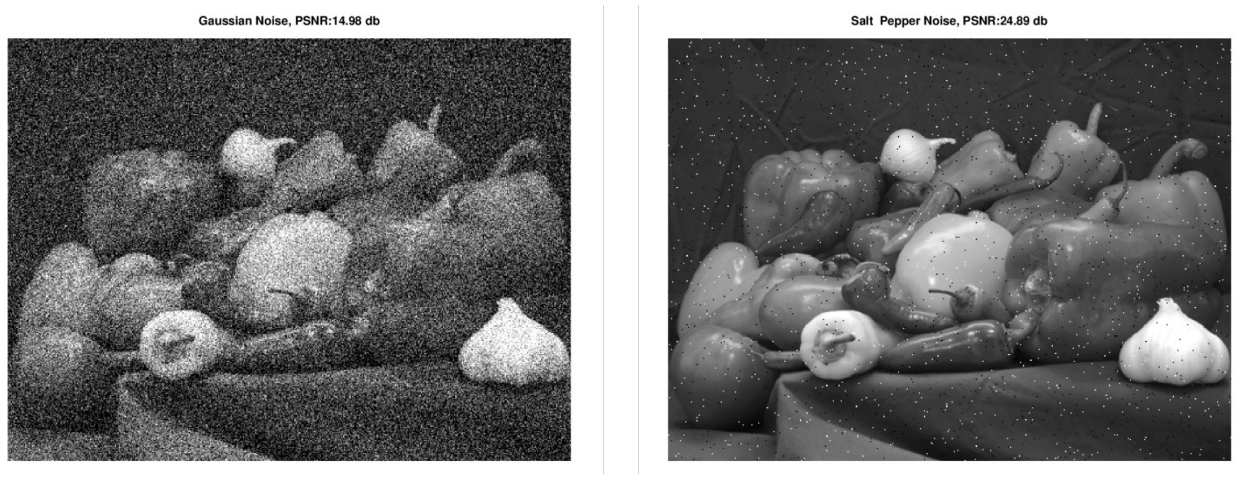

Gaussian Noise, PSNR:14.98 db
Salt Pepper Noise, PSNR:24.89 db

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:

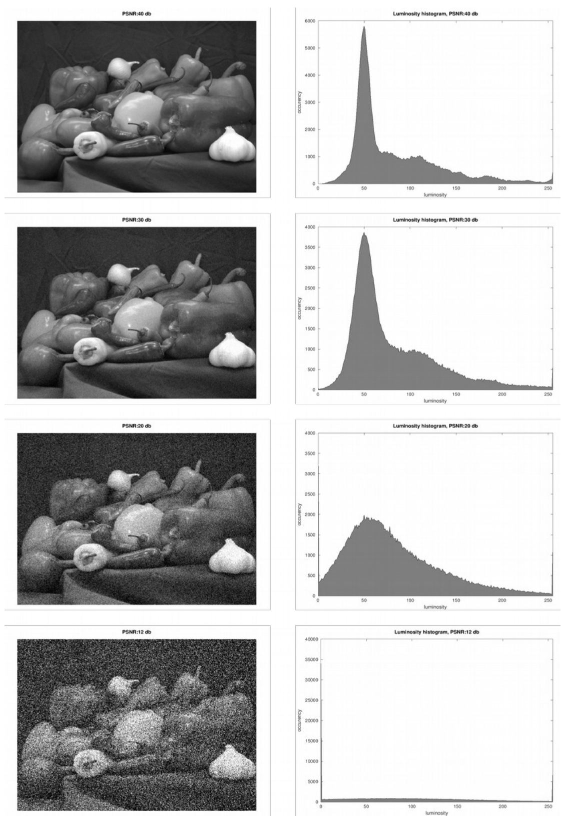


Illustration 6: 4 different level of Gaussian noise

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:

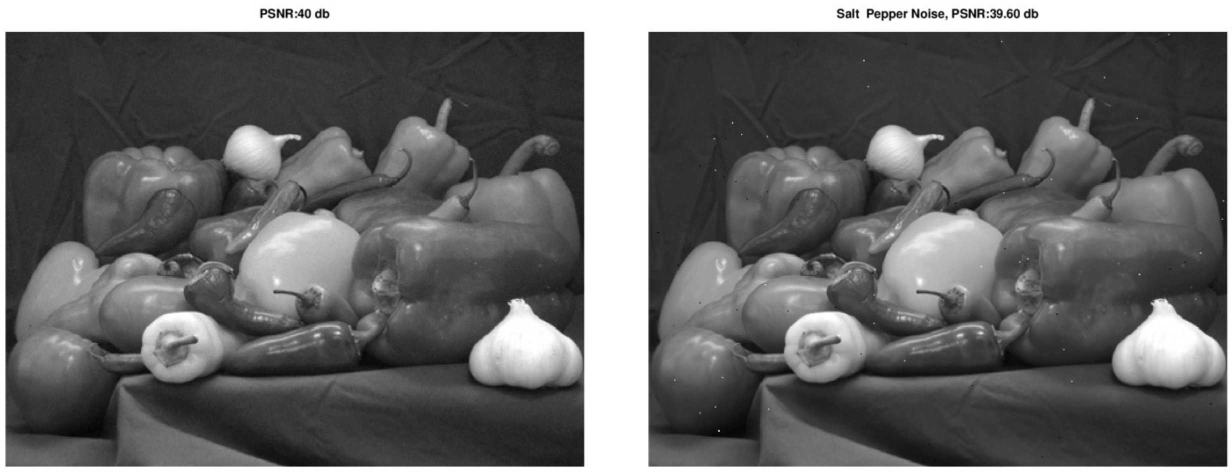


Illustration 7: Gaussian noise and Salt & Pepper noise at a PSNR of 40db

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.

2 Singular Value Decomposition

In this second exercise we will see how the SVD can be applied to an image for both compression and noise “reduction”.

In particular we will decompose the image I in the 3 matrix:

$$I = U \times S \times V^T$$

For compression we will select the first k components of the 3 matrix to represent the original image, more components we use closer it becomes to the original image.

In particular it is possible to see the value of the MSE between the reconstruction and the original varying the parameter k as shown in the following figure:

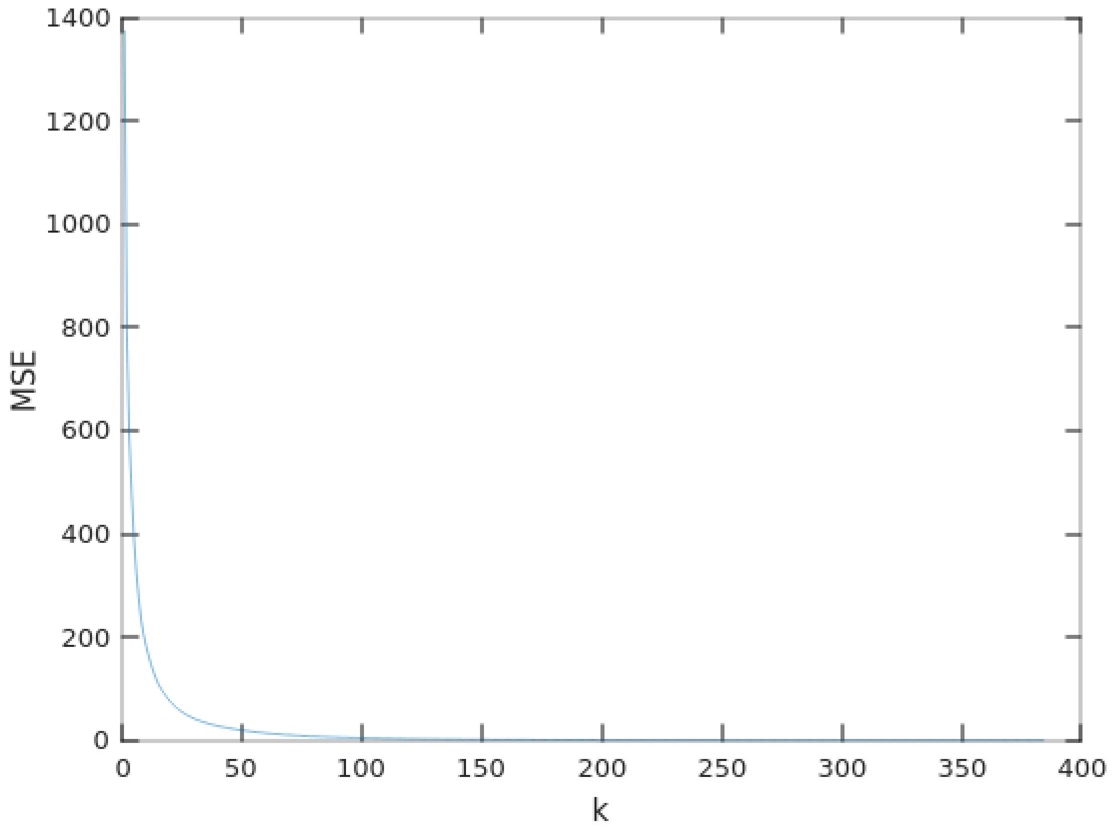


Illustration 8: MSE between the original image and the reconstructed one varying k

To visualize the result of such reconstruction I choose to see the image reconstructed with 15, 30, 50 and 100 components, the pictures are shown in the illustration 9.

As it possible to see in the illustration, already with 50 components the image it’s visually pleasant, and with 100 components become it’s hard to find any visual difference with the original (shown in illustration 3).

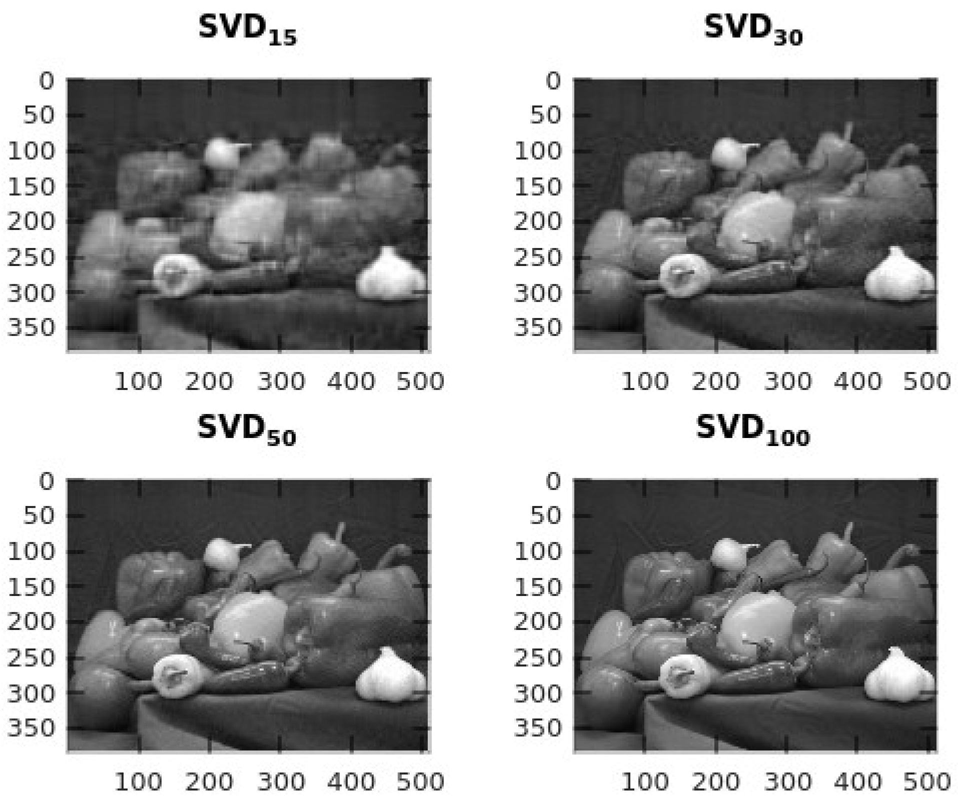


Illustration 9: Reconstruction with 15, 30, 50 and 100 components

We were also asked to create a super noisy version of the *peppers.png* image and to try to see how the SVD decomposition will affect the MSE with the original image.

The plot of the MSE against the number of components is shown in the following illustration:

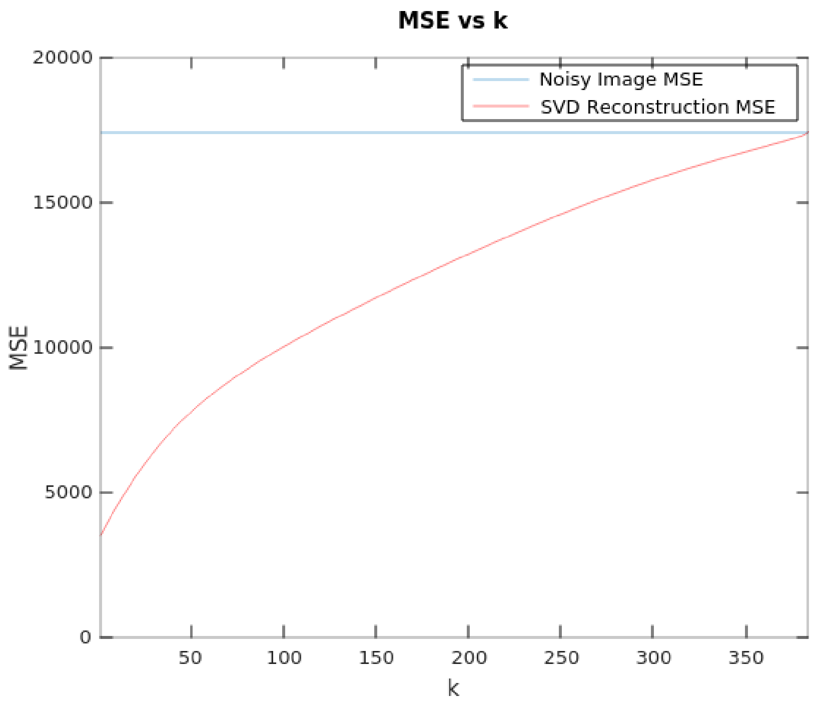


Illustration 10: MSE of super noisy reconstructed image

In this case the trend is opposite to the one before, and more components we have more the error is high and finally match the super noisy image MSE.

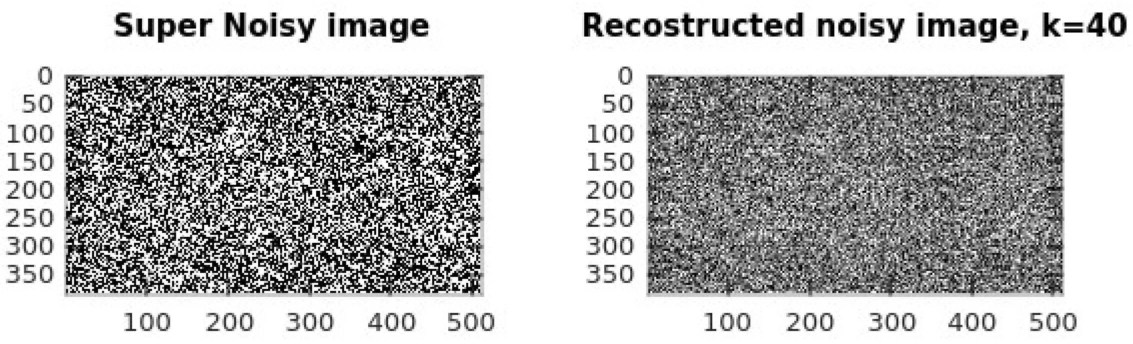


Illustration 11: Super noisy image and it's reconstruction

As it’s possible to see the noisy image has nothing to do with the original *peppers.png* image, however less component we use more the resulting image result flat and some small details of the original image appears (brighter areas for examples).

Probably this because the first components of the SVD contain the strongest part of the information of the image.

3 Noise Visibility Function

In this exercise we will implement the Noise Visibility Function and we will use it to hide a watermark in an image (shown in the following illustration).

Illustration 12: Lena

The Noise Visibility Function tell us how the noise (or a signal) will be visible in a certain area of the image. This depends from the local variance, with higher variance we will have smaller NVF.

Intuitively in uniform region of the image the noise is more noticeable then in rougher regions.

After implementing the function we were asked to add a simple watermark (in this case a Gaussian noise) to the image with and without using the NVF function:

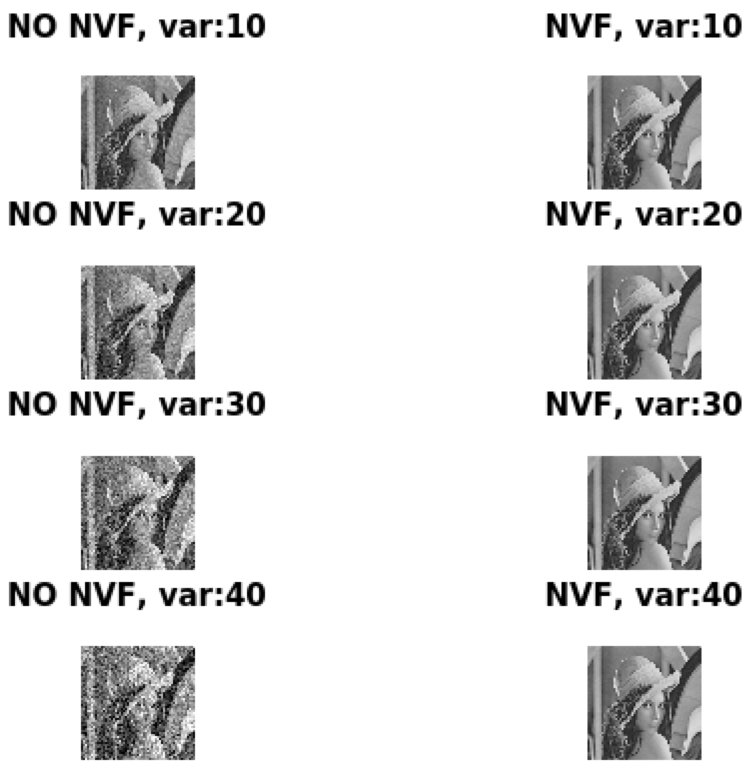


Illustration 13: Watermark with different $\sigma_z^2=40$ *with and without the NVF*

In all the case where we apply the watermark without using the NVF it appear clearly visibile while using the NVF it better hidden, even if with $\sigma_z^2=40$ start to appear even with the NVF

The previous observations are also visible by plotting the MSE of the image with watermark with the original image:

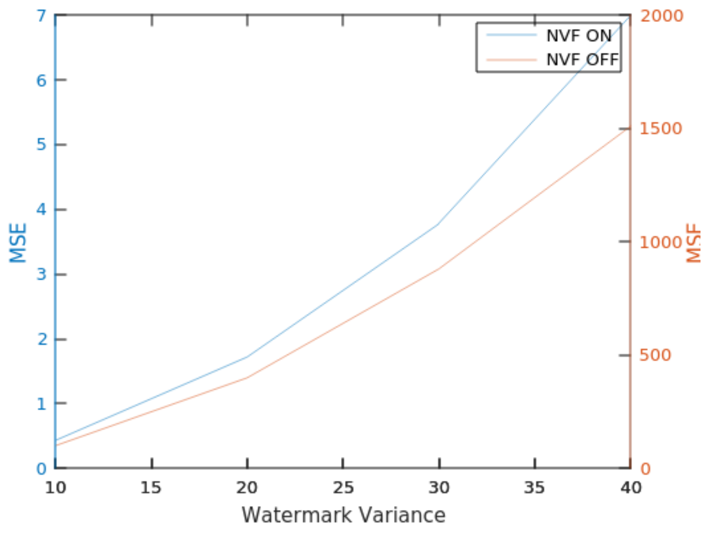


Illustration 14: MSE of the image with watermark with and without NVF

Note that the MSE of the image plus the watermark without NVF is around 300 times higher that the same watermark applied using the NFV.

After some try I find out the the more adapted value of D for the chosen picture is around ~2. This parameter will define the strength of application of the watermark.

Similarly σ_z^2 will modulate the strength of the watermark.

4 Nuts and bolts

In this exercise we will try to classify some object in a pictures, to do so we will use the property extracted using the function *regionprops.*

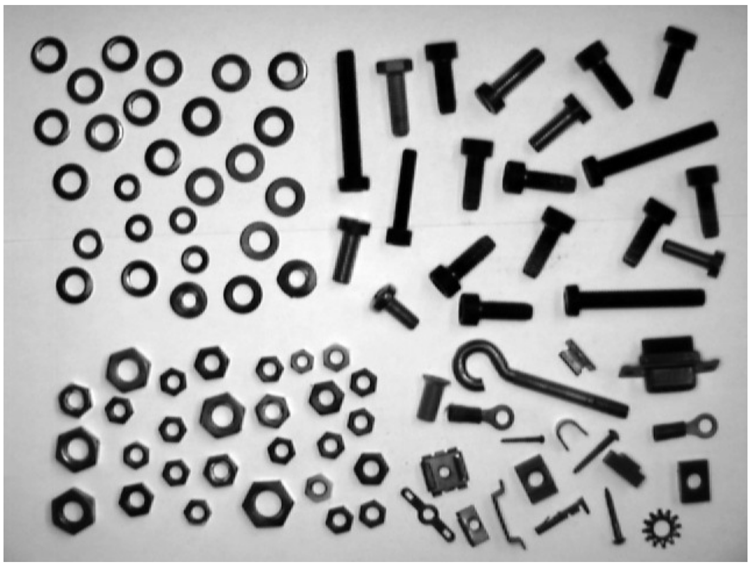


Illustration 15: Original image

The goal is to be able to classify 4 kind of objects from the picture shown in illustration 15.

The first step is to convert it in a B/W image, and to de-noisy it as much as possible, to do so I choose to use a Gaussian blur:

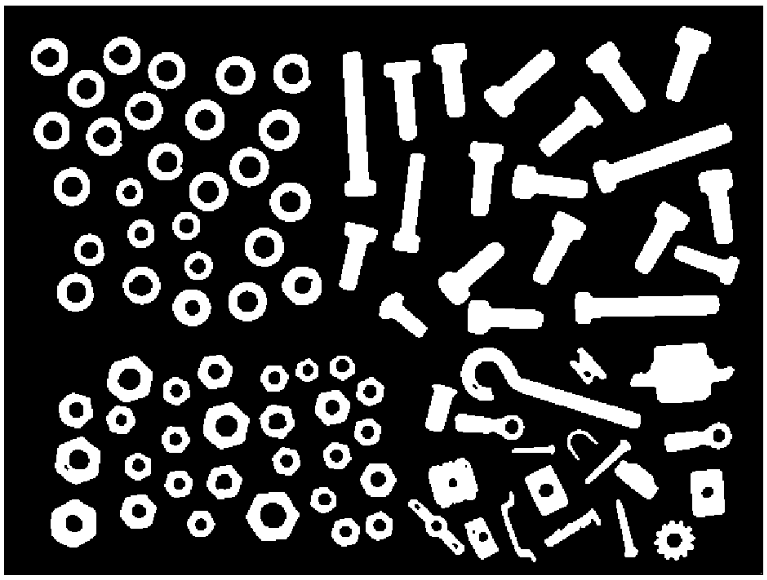


Illustration 16: B/W and blurred image

Then just using the position of each object I manually classified it in 4 groups:

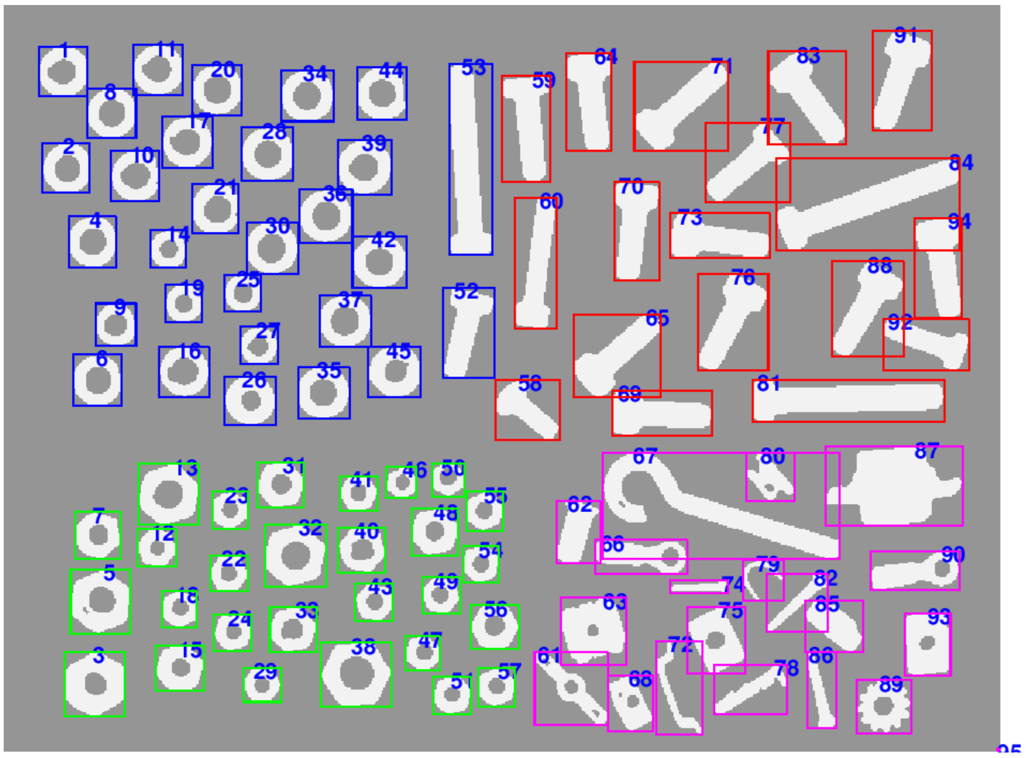


Illustration 17: Manually classified objects

Then I tried to search which properties could be useful to automatically classify this objects, I have chose the E*ccentricity* (that return how elliptical the object is), the *EulerNumber* (that returns the number of objects minus the number of holes) and the *FilledArea* (that returns the area of the object) divided by the *Area* (that returns the area of the bounding box).

The resulting features are shown in the following illustration:

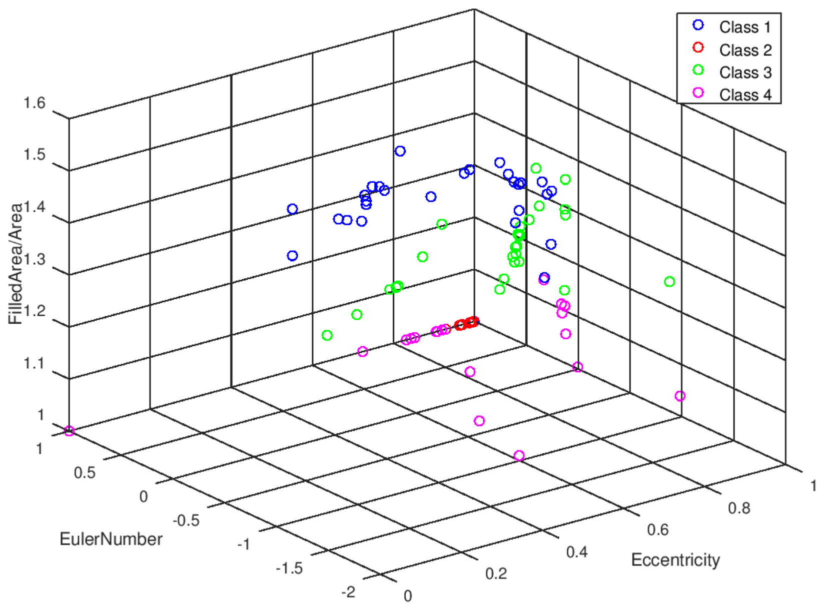


Illustration 18: Eccentricty, EulerNumber and FilledArea/Area of the objects

Using the *kmeans* unsupervised classification algorithm over the extracted features I obtained the following classification:

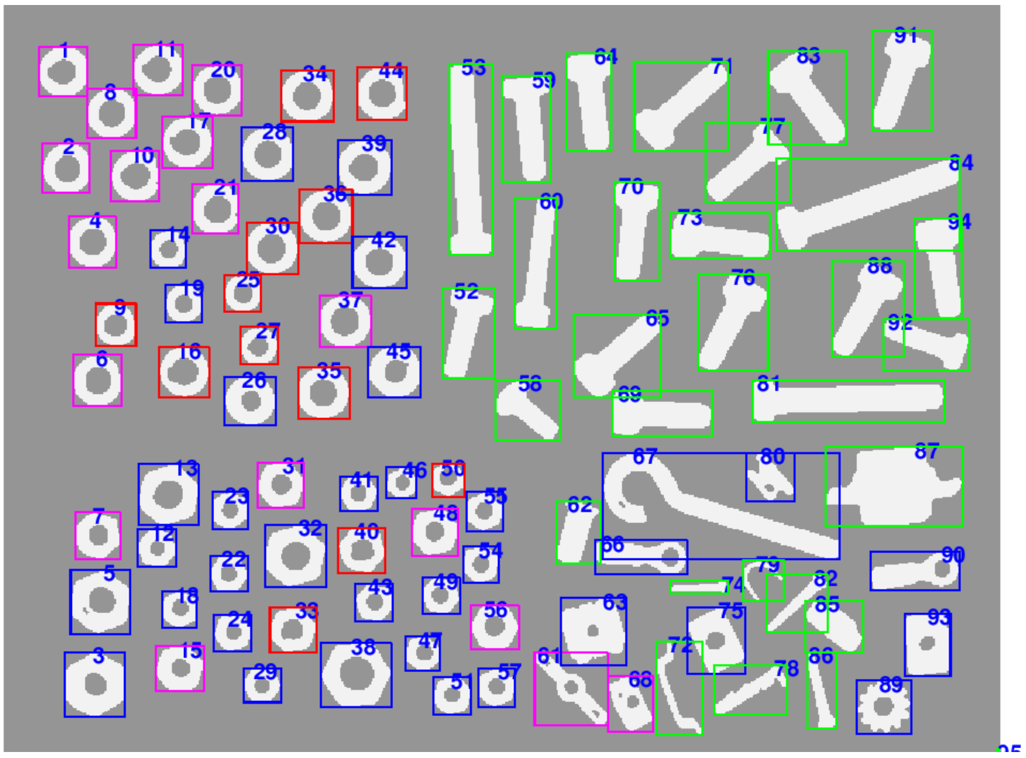


Illustration 19: Automatic classification

The classification is able to identify very well the bolt (class 2 of the manual classification) and it's able to recognize the objects with circular shape, however it fails to distinguish between the round and the hexagons.

And features are distributed between the classes as following:

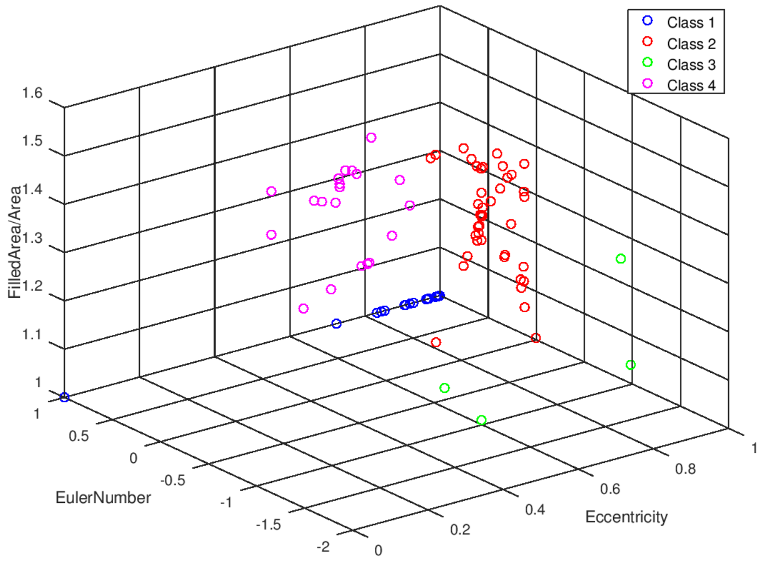


Illustration 20: Automatic classification using extracted features

I believe that with a better choice of features or more powerful features extraction method it would be possible to automatize the classification.

However trying to classify only two classes of objects works very well as shown in the following figure:

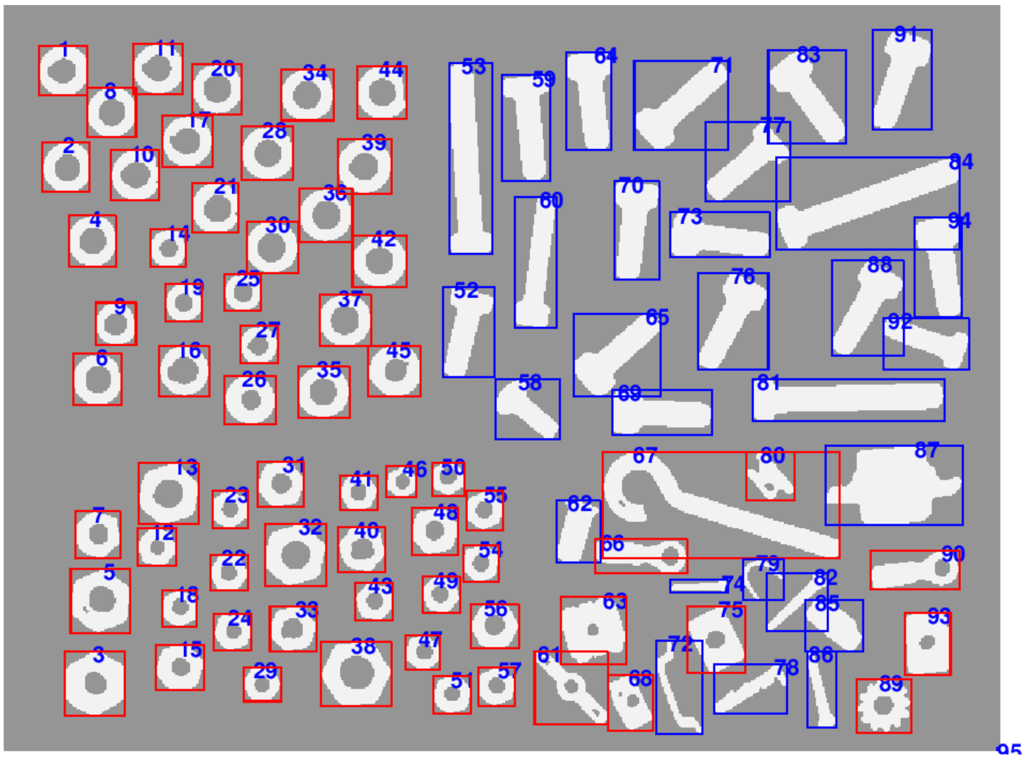


Illustration 21: 2 classes classification