Université de Genève

Multimedia Security and Privacy
14x016

TP 1 : Basic Image Processing

Author: Deniz Sungurtekin

E-mail: Deniz.Sungurtekin@etu.unige.ch

Mars 2020

1 Introduction

The goal of this work is to run through and remind some of the basic images operations with Python libraries skimage, cv2 and numpy. Firstly, we will simply use the principals functions allowing us to read and do some manipulations. Secondly, we will show two types of noises, additive white gaussian noise (AWGN) and salt pepper noise. Those noises will be used to introduce two important metrics used in signal and image processing: the mean squared error (MSE) and the peak signal to noise ration (PSNR). Finally, we will implement a perceptual image hash function which are used to retrieve identical images such as matching an image thumbnail against it original.

2 Basic Image Commands

Using numpy and cv2 libraries we can easily read and display an image and its size. For example, we will read an image of a peacock:

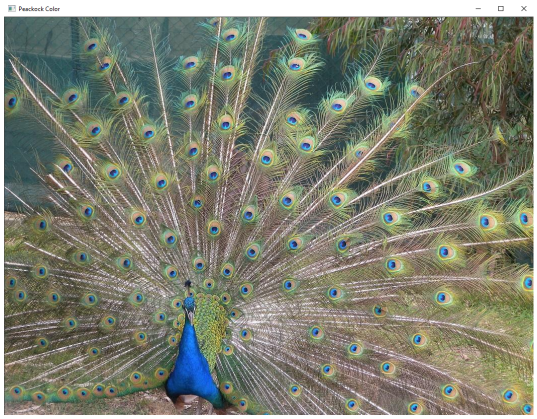

Figure 1: Colored Peacock

```
The image has height:  792
The image has width:  1056
The image has channels:  3
```

Figure 2: Corresponding output for the shape of the colored peacock

As we can see this image has three channels, the red one, the green one and the blue one. For each of this channel, we can plot the corresponding histogram to visualize their distribution in the image:

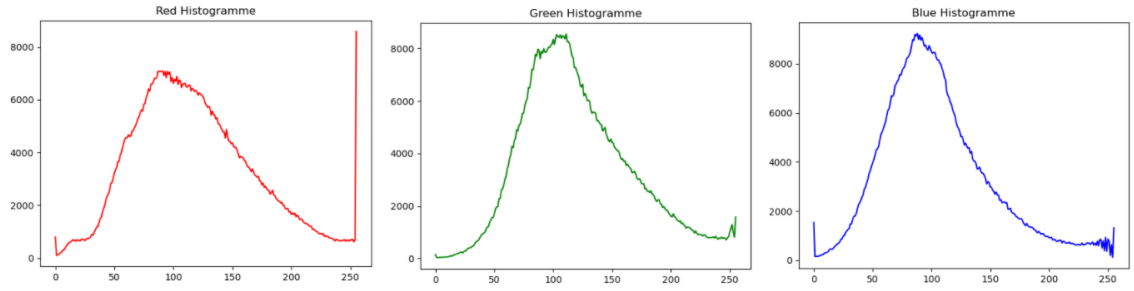

Figure 3: Colors histogram

Indeed, we observe that each color are almost following a Gaussian distribution and that there isn't a strong dominance by a color in the image.

Now we will convert the image to grayscale to simplify our manipulation on it. Then, we will compute the global mean and the global variance of image:

```
Global mean:  118.79164394895929
Global Variance:  2144.2387572253087
```

Figure 4: Global mean and the global variance of pixels value

Using the function skimage.blockreduce from skimage library, we can also compute the local mean and variance with 3x3 blocks:

Figure 5: local mean and variance

We can observe that for the local mean we almost have the grayscale original image with a smaller size because we used a very small block of 3x3 so the value are very similar. However, if we would used a bigger block's size we would have a much blurred image. As regards of the local variance, we can see the white region where the image show a lot of variation in the pixels value.

3 Noise

In this section, we will implement the Salt and pepper noise and White Gaussian Noise and analyse the corresponding MSE and PSNR value between an image and the same noised image.

Firstly, we will read and store an image in uint8 datatype and convert it to double datatype and compute the MSE between the two "same image" to see the importance of knowing which datatype we are using:

```
MSE between the two dataType:  16128.447786022514
```

Figure 6: MSE between the two "same image"

As we can see, the result is not zero which can be explained by the fact that the Uint8 datatype define each pixel's value between 0 and 255. In contrary, the double datatype define each pixel's value between 0 and 1. Consequently, when we do the difference between one pixel of each image, we have a very big value which explain the obtained MSE value between the two same images. In conclusion, its important to use the same datatype or to convert an image in the MSE function.

Now we will add Gaussian noise to an image such that the refactored PSNR (where MSE(x,y) is replaced by the variance of our noise) ratio with the original image is 10dB, 20dB, 30dB and 40dB and analyse the result by displaying the noised image, the corresponding standard deviation used to generate our noise and the histogram of this noised image.

Here are the result for the four differents value of PSNR:

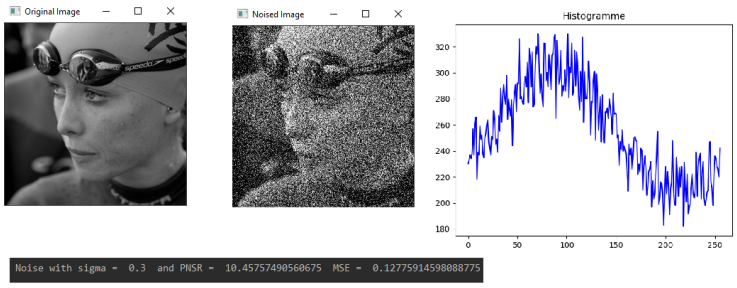

Figure 7: PSNR = 10

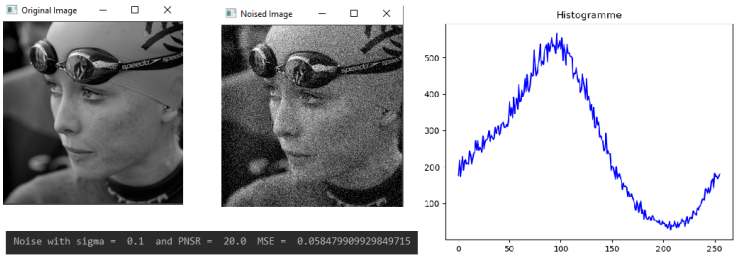

Figure 8: PSNR = 20

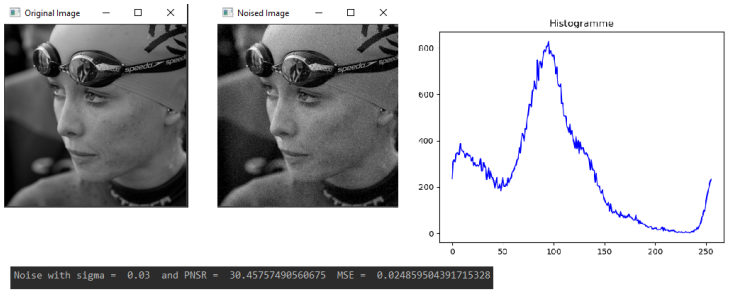

Figure 9: PSNR = 30

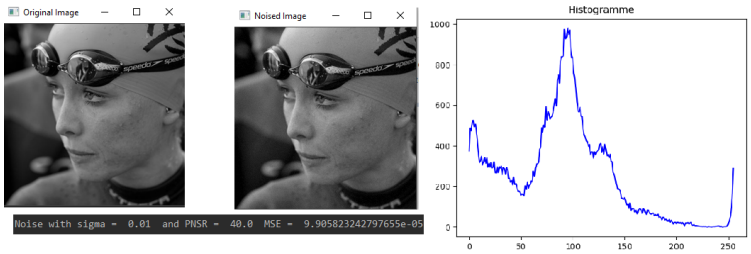

Figure 10: PSNR = 40

We can clearly see that the more the sigma (standard deviation of the noise) is big the more the image is hard to recognize. This noised image is only an addition of two matrix, the image and the Gaussian noise matrix which is simply a matrix where each element follows a centered Gaussian distribution with a given standard deviation. Consequently, the more the noise is big the more the MSE is big and by definition, the more the MSE (or the standard deviation) is big the smaller is the PSNR. Furthermore, we can clearly see the presence of the noise in the histogram where there is a lot of variation if the noise is strong. For example, in the PSNR = 40, the histogram is very similar to the original's one but in the PSNR = 10 we observe a lot of vibrations in the histogram.

Now we will add salt and pepper noise to the image to have a PSNR = 40dB noisy image:

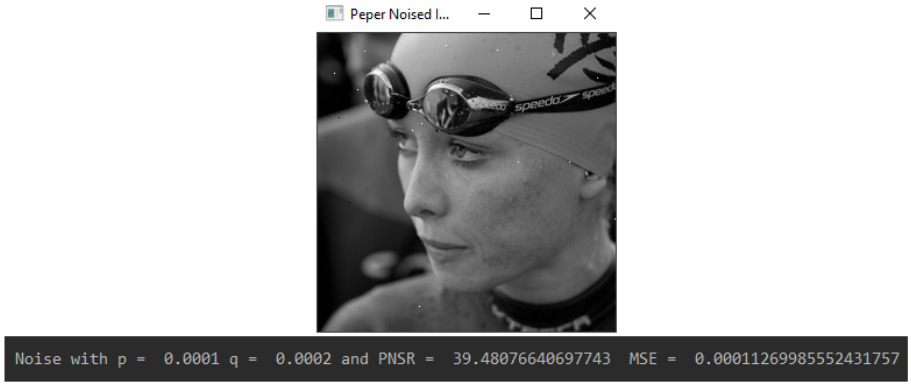

Figure 11: Pepper noise PSNR = 40

Here, we can visually see the difference between the noised image and the original one because there is some white and black point on the image. We can conclude that the Gaussian noise is less detectable by the human visual system because it's distributed inside all the pixels of the image. In contrary, the salt and pepper noise is only located at some specifics pixels but their value are extreme so its immediately visible by our eyes if the neighborhood of those pixels aren't completely black or white.

4 Identification

Here, we will design and test a perceptual image hash function, Those functions generate a short binary descriptor vector from an image such that even minor changes in the image lead to a different descriptor value. The idea is to work with 200 images for which we will compute the corresponding descriptor value. Then, for each of those images, we will compute the Gaussian noised image with PSNR = 35 and their descriptor value. With the descriptors value we can compute the probability error between them, this probability is given by:

$$P_b = \frac{h}{N} \tag{1}$$

Where h is the hamming distance between the two vectors and N is the length of the binary descriptor vector.

With the probability error of each pair in the set, we can analyse the inter-class and the intra-class distance. In total there is 40'000 possible pairs because we have 200 originals images and 200 noised images but 200 of those pairs are between image which belongs to the same class. In other words, 200 pairs are made of the descriptor vector from an image and the descriptor vector of the corresponding noised image. Consequently, the goal is to build and display a normalized histogram of all found P_b between 39800 pairs to visualize the probability mass function of the inter-class distances and between 200 pairs to visualize the PMF of the intra-class distances.

We can observe my result for the inter-class distance:

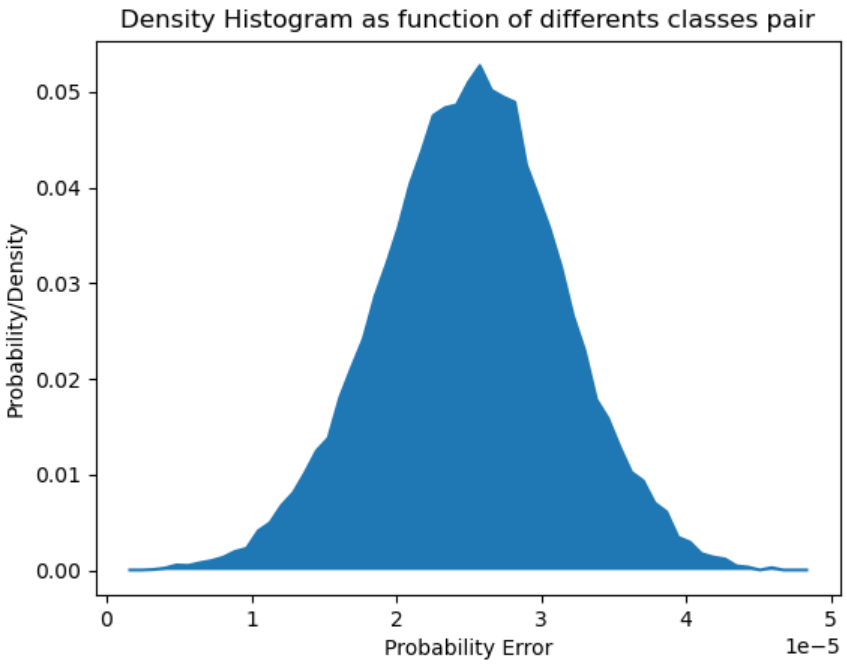

Figure 12: Inter-class PMF

We can clearly see that it follow a Gaussian distribution, it seems pretty good because we can see that there is only few collisions so we can easily distinguish two image of different classes. A collision occurs when we have for two vectors a probability error of 0. However for the intra-class the result seems bad because almost all of my probability error is equal to 0. I have 95% of 0 and only 0.5% of 0.1 so the histogram is only made of two bins:

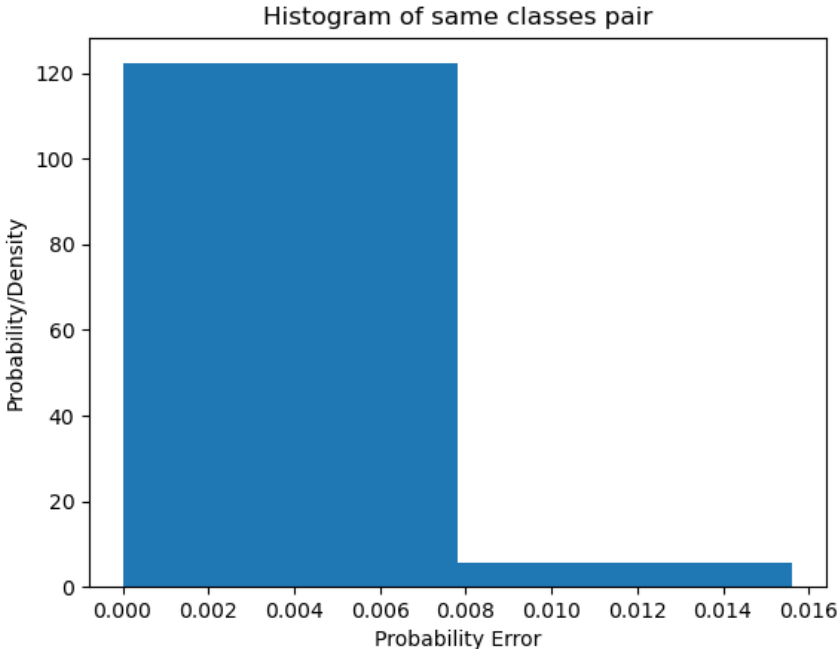

Figure 13: Intra-class PMF

Furthermore, the plt.hist from matplotlib doesn't seems to work properly because despite the fact that I specified a density histogram, I have value bigger than one in the y-axe. (After some research a lot of people seems to have the same problem but i guess there is always a way to do it correctly) That's why I explicitly specified the distribution I had.

More importantly, if my implementation is correct it seems that our perceptual hash is not a good way to distinguish two images from the same class with a noised image of PSNR = 35. As I said, almost every probability error equals 0 so we have a lot of collision. Consequently, it's hard to distinguish which image is the original in the pair because they have almost every time the same descriptor vector.