

# Multimedia Security and Privacy

## TP1: Basic Image Processing

Prof. SVIATOSLAV VOLOSHYNOVSKIY,  
ROMAN CHABAN <[roman.chaban@unige.ch](mailto:roman.chaban@unige.ch)>,  
YURY BELOUSOV <[yury.belousov@unige.ch](mailto:yury.belousov@unige.ch)>,

Stochastic Information Processing Group

March 2, 2023

## Submission

Please send your submissions in either PDF or Jupyter Notebook to “Assignments/TP1: Basic Image Processing” on <https://moodle.unige.ch> before **Wednesday, March 16 2023, 08:00 AM**. Note, the assessment is mainly based on your report, which should include your answers to all questions and the experimental results.

## 1 Introduction

This exercise will run through some of the basic image commands from Python libraries `skimage`, `cv2`, `numpy`.

### Exercise

1. Read in the image `'peacock.jpg'`. The image is present in Moodle.
2. Display the size and the histogram of the image.
3. Convert the image to grayscale.
4. Determine the global mean and the global variance of the image.
5. Determine the local mean and variance of an image and display them. You could find how to properly use Python-numpy functions `numpy.reshape`, `numpy.mean`, `numpy.var`, or the Python-skimage function `skimage.measure.block_reduce` any other library/function.

## 2 Noise

This exercise will showcase two noise types, *Additive White Gaussian Noise* (AWGN) and *salt & pepper* noise.

### Additive White Gaussian Noise

AWGN is a type of *channel* in which the discrete channel output  $Y_i$  at some time event with index  $i$  is the sum on the input  $X_i$  and noise  $Z_i$ , where  $Z_i$  is independently and identically-distributed (i.i.d.) from a zero-mean Gaussian distribution with standard deviation  $\sigma$  (or variance  $\sigma^2$ ). Formally:

$$\begin{aligned} Z_i &\sim \mathcal{N}(0, \sigma) \\ Y_i &= X_i + Z_i \sim \mathcal{N}(X_i, \sigma) \end{aligned}$$

### Exercise

1. Write a function that generates an array of size  $N \times M$  with Gaussian Noise, i.e. samples drawn from the distribution  $\mathcal{N}(\mu, \sigma)$ . See the Python function `numpy.random.randn` or Matlab function `randn`.

### Salt & pepper Noise

*Salt & pepper* noise can be defined as follows. Let there be a  $N \times M$  grayscale image whose datatype supports a value range of  $\{s_{min} \dots s_{max}\}$ . Let  $\mathbf{y}$  denote the noisy image and  $\mathbf{x}$  the original image. Then the observed gray level in image  $\mathbf{y}$  at pixel location  $(i, j)$  is given by:

$$y_{i,j} = \begin{cases} s_{min} & \text{with probability } p \\ s_{max} & \text{with probability } q \\ x_{i,j} & \text{with probability } 1 - p - q \end{cases} \quad (1)$$

### Exercise

1. Implement a function that generates *salt & pepper* noise with parameters  $p$  and  $q$ .

### Metrics

We will now define and implement two important metrics used in signal and image processing: the Mean Squared Error (MSE) and the Peak Signal to Noise Ratio (PSNR).

#### Mean Squared Error

The Mean Squared Error (MSE) between two images  $\mathbf{x}$  and  $\mathbf{y}$  is defined as:

$$\text{MSE} = \frac{1}{N \cdot M} \sum_{i=1}^N \sum_{j=1}^M (y[i, j] - x[i, j])^2,$$

where  $N$  and  $M$  are the width and the height of image  $\mathbf{x}$  and  $\mathbf{y}$ .

### Exercise

1. Write a function that determines the Mean Squared Error (MSE) between two images  $\mathbf{x}$  and  $\mathbf{y}$ .
2. Read in a new copy of the image `peacock.jpg`, keep it in its original datatype and range, i.e. Python/Numpy `uint8` and `{0..255}`.
3. Now read in a second copy of the image `peacock.jpg` but map it to `double` and `{0..1}`. You could use Python `skimage.img_as_float`. Compare the two images using the MSE. Can you explain the result?

#### Peak Signal to Noise Ratio

The Peak Signal to Noise Ratio (PSNR) is defined as:

$$\text{PSNR} = 10 \log_{10} \left( \frac{\alpha^2}{\text{MSE}(\mathbf{x}, \mathbf{y})} \right),$$

where  $\alpha$  is the maximum value possible with the `type` of  $\mathbf{x}$  and  $\mathbf{y}$ . The unit of the PSNR is dB.

### Exercise

1. Write a function that implements the PSNR function.

### Testing

We will test how various noise strengths influence the PSNR value between the original image and the noisy copy.

### Exercise

1. Refactor the PSNR definition such that the PSNR is expressed as a function of the noise variance  $\sigma_z^2$ . You may assume that  $\sigma_z^2 = \text{MSE}(\mathbf{x}, \mathbf{y})$ .
2. Add Gaussian noise to an image such that the PSNR ratio with the original image is 10dB, 20dB, 30dB and 40dB.
3. Show the noisy images on the screen. How do they look?

4. Show the histograms for these noisy images, can you explain what you see?
5. Add salt & pepper Noise to an image until the PSNR ratio between the original and the noisy image is 40 dB. Visually compare it to the 40dB noisy image to which Gaussian noise was added. What can you conclude?



**Figure 1** – All these noisy images have approximately the same MSE value between them and the original, about 0.0135

### 3 Identification

In this mini project we will design and test a perceptual image hash function. Perceptual hash functions generate a short binary descriptor vector from an image. They are designed so that minor changes in the image itself, do not lead to a different descriptor value. They thus differ completely from their cryptographic counterparts who are designed to flip half of the output bits if a single one input bit changes.

Perceptual hashes are used, for example, to retrieve identical images, such as matching an image thumbnail against its original. See the reverse-image search engine <http://www.tineye.com/>.

Three principle things will be done

- Design and implementation of the perceptual hash function.

- Testing the uniqueness of the generated descriptor vectors for a small collection of images.
- Testing the influence of noise on the image on its resulting descriptor vector.

You will find 200 tiff images, each  $256 \times 256$  on Moodle.

## Perceptual Hash

The basic algorithm can be implemented as follows:

### Exercise

1. Read an image and convert it to grayscale.
2. Determine the global mean of the image.
3. For each  $32 \times 32$  subimage non-overlapping block, determine the local mean.
4. For each subimage block determine if the local mean is larger than the global mean. If so, the hash value for this particular block is 1 and 0 otherwise. The resulting descriptor is thus a 64 bit binary vector.

### Hamming distance

Our algorithm outputs 64-bit binary descriptor vectors. They can thus be compared with the Hamming distance. In information theory, the Hamming distance  $h$  between two strings of equal length  $N$  is the number of positions at which the corresponding symbols are different. It can be implemented as:

$$h = \sum_{i=1}^N \oplus(a[i], b[i])$$

The probability of error  $P_b$  between two strings of equal length  $N$  is:

$$P_b = \frac{h}{N};$$

### Exercise

1. Implement a function that determines the Hamming distance  $h$  and probability of error  $P_b$  between two binary vectors.
2. Implement a function that can read in 200 images sequentially and stores the 200 resulting binary descriptor vectors. See the Python `os.walk` or `pathlib.Path.iterdir()`.

## Inter and intra class distance

The inter-class distance is the distance, following some metric, between members of a non-identical class. In this case, all 200 images form their own class, then completed with one slightly distorted copy. To ascertain if our perceptual hash can distinguish all 200 images we will test all distances between them. If different images give the same descriptor vector, a so-called *collision*, their difference will be zero. Consequently, these images can not be distinguished based on our hash function. Secondly, if the difference in hash value between two different images is very small, it might not be possible to distinguish an original image from a completely different image and an identical but slightly distorted copy. See Figure 2.

The intra-class distance is the distance between members of the same class. In our case, this is the distance between the descriptor value from an image and a noise-distorted copy. We will thus measure how much the descriptor of an image changes if that image is corrupted by noise. See Figure 3.

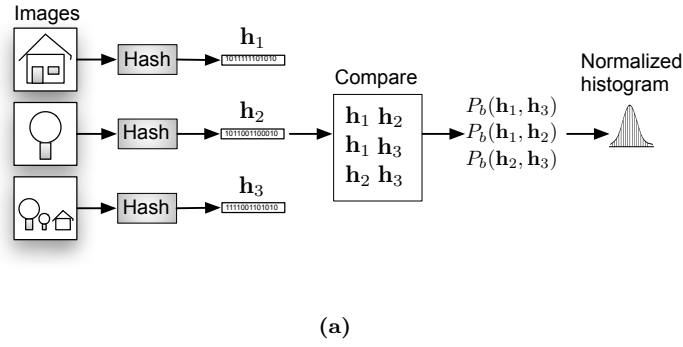


Figure 2 – Inter-class test schematic

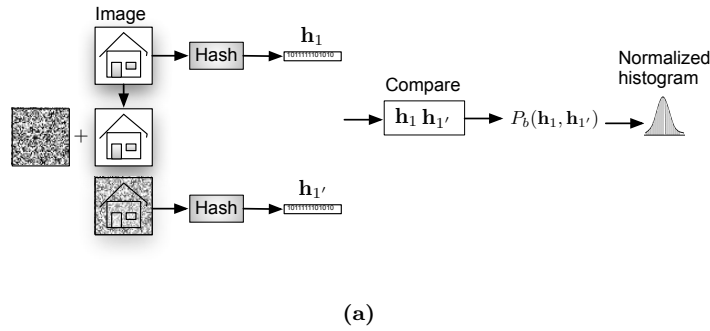


Figure 3 – Intra-class test schematic

### Implementation

All distances can be determined computationally efficiently using matrix multiplication. Let matrix  $\mathbf{X} \in \mathbb{R}^{L \times M}$  be the matrix with column vectors  $\{\mathbf{x}^1, \dots, \mathbf{x}^M\}$  where each vector  $\mathbf{x}^m \in \mathbb{R}^L$  is a descriptor vector of length  $L$  from image  $m$ ,  $1 \leq m \leq M$ . Similarly, let matrix  $\mathbf{Y}$  be the matrix formed from descriptor vectors, but originating from distorted images. In this particular TP column vector  $\mathbf{x}^1$  is the 64-bit hash value from image 1, and  $\mathbf{y}^1$  is the 64-bit hash value from the distorted image 1.

Intra and inter class distances can be ascertained as follows:

- Convert  $\mathbf{X}$  and  $\mathbf{Y}$  values from  $\{0, 1\}$  to  $\{-1, 1\}$
- Determine the *confusion* matrices:  $\mathbf{Z} = \mathbf{X}^T \mathbf{Y}$  and  $\mathbf{Z}' = \mathbf{X}^T \mathbf{X}$ .
- Convert to traditional hamming distances:  $\forall z \in \mathbf{Z}$  or  $\mathbf{Z}' \mid z := (z - L) ./ -2$  (for each row subtract  $L$  and divide by  $-2$ ).
- The intra class distances can now be read of the diagonals of matrices  $\mathbf{Z}$  and  $\mathbf{Z}'$ , the off diagonals elements form the inter class distances.

### Exercise

1. Implement a function that takes all images one by one, makes a copy and distorts this copy with AWGN resulting in a PSNR of 35dB.
2. Determine the descriptor of both the original image and the distorted copy. Do this for all 200 images.

3. Implement a function that determines the probability of error  $P_b$  between all pairs of two from different classes among the 200.
4. Build and show a *normalized* histogram of all found  $P_b$  values. Normalized means that all values from all histogram buckets should sum up to 1. You are thus estimating a probability mass function (PMF) of the inter-class distances.
5. What can you conclude from the histogram?
6. Implement a function that determines the probability of error  $P_b$  between all pairs of two from the same class among the 200.
7. Build and show a *normalized* histogram of all found  $P_b$  values. You are thus estimating a probability mass function (PMF) of the intra-class distances.
8. What can you conclude from the histogram?
9. Compare the estimated probability density graphs from the intra-class and inter-class distances. Do they overlap? What can you conclude from this?

**Hints**

- Start with testing with 20 images while debugging.
- Write out intermediate results to disk.
- Test your normalized histogram on samples from a known distribution first.