RAPPORT

Rémy AUDA

Table des matières

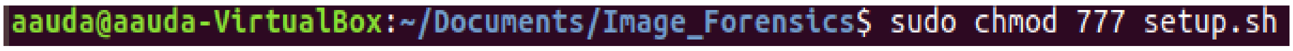
Introduction:

In this project, I implemented various algorithms related to the topic of forensic image. The main goal of forensic image is to detect some manipulations that could be done on an image.

Compilation:

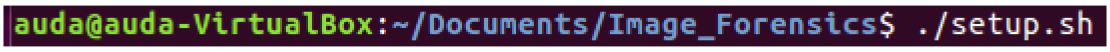
First, go to the root directory of project. After, run the following command line:

sudo chmod 777 setup.sh



```
aauda@aauda-VirtualBox:~/Documents/Image_Forensics$ sudo chmod 777 setup.sh
```

./setup.sh



```
auda@auda-VirtualBox:~/Documents/Image_Forensics$ ./setup.sh
```

This bash could install you automatically all the requirements that you need.

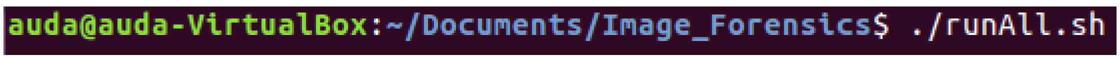
If you experiment some issues, retry to execute the setup.sh.

If the problem persists, open the setup.sh script and try to execute each command line manually.

Next, you have the choice. Either, you can run all demo of different algorithms or you can run each algorithm separately.

To run all demo of different algorithms, execute this following command line to the root directory of project:

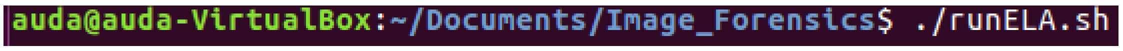
./runAll.sh



```
auda@auda-VirtualBox:~/Documents/Image_Forensics$ ./runAll.sh
```

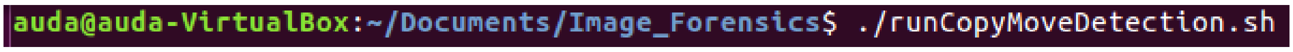
To run all demo of each algorithm separately, execute this following command line to the root directory of project:

./runELA



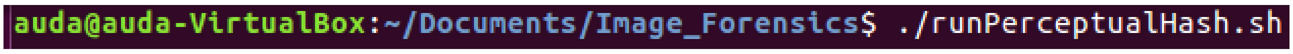
```
auda@auda-VirtualBox:~/Documents/Image_Forensics$ ./runELA.sh
```

./runCopyMoveDetection



```
auda@auda-VirtualBox:~/Documents/Image_Forensics$ ./runCopyMoveDetection.sh
```

./runPerceptualHash

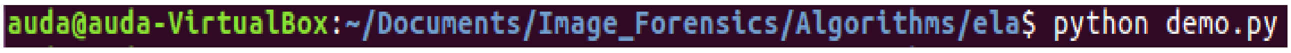


```
auda@auda-VirtualBox:~/Documents/Image_Forensics$ ./runPerceptualHash.sh
```

Of course, you can also run the demo manually with following command:

To execute ELA algorithm, go to the directory « Algorithms/ela » and execute the following command:

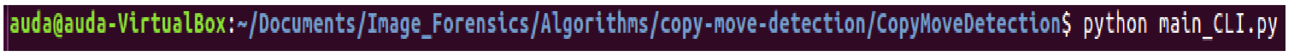
python demo.py



```
auda@auda-VirtualBox:~/Documents/Image_Forensics/Algorithms/ela$ python demo.py
```

To execute Copy-Move-Detection algorithm, go to the directory « Algorithms/copy-move-detection/CopyMoveDetection » and execute the following command:

python main_CLI.py



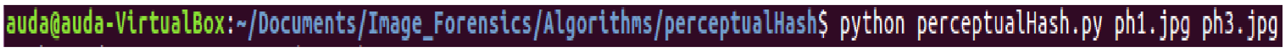
```
auda@auda-VirtualBox:~/Documents/Image_Forensics/Algorithms/copy-move-detection/CopyMoveDetection$ python main_CLI.py
```

To execute perceptual Hash algorithm, go to the directory « Algorithms/perceptualHash » and execute the following command:

python perceptualHash.py <image1> <image2>

For example :

python perceptualHash.py ph1.jpg ph3.jpg



```
auda@auda-VirtualBox:~/Documents/Image_Forensics/Algorithms/perceptualHash$ python perceptualHash.py ph1.jpg ph3.jpg
```

1. ELA:

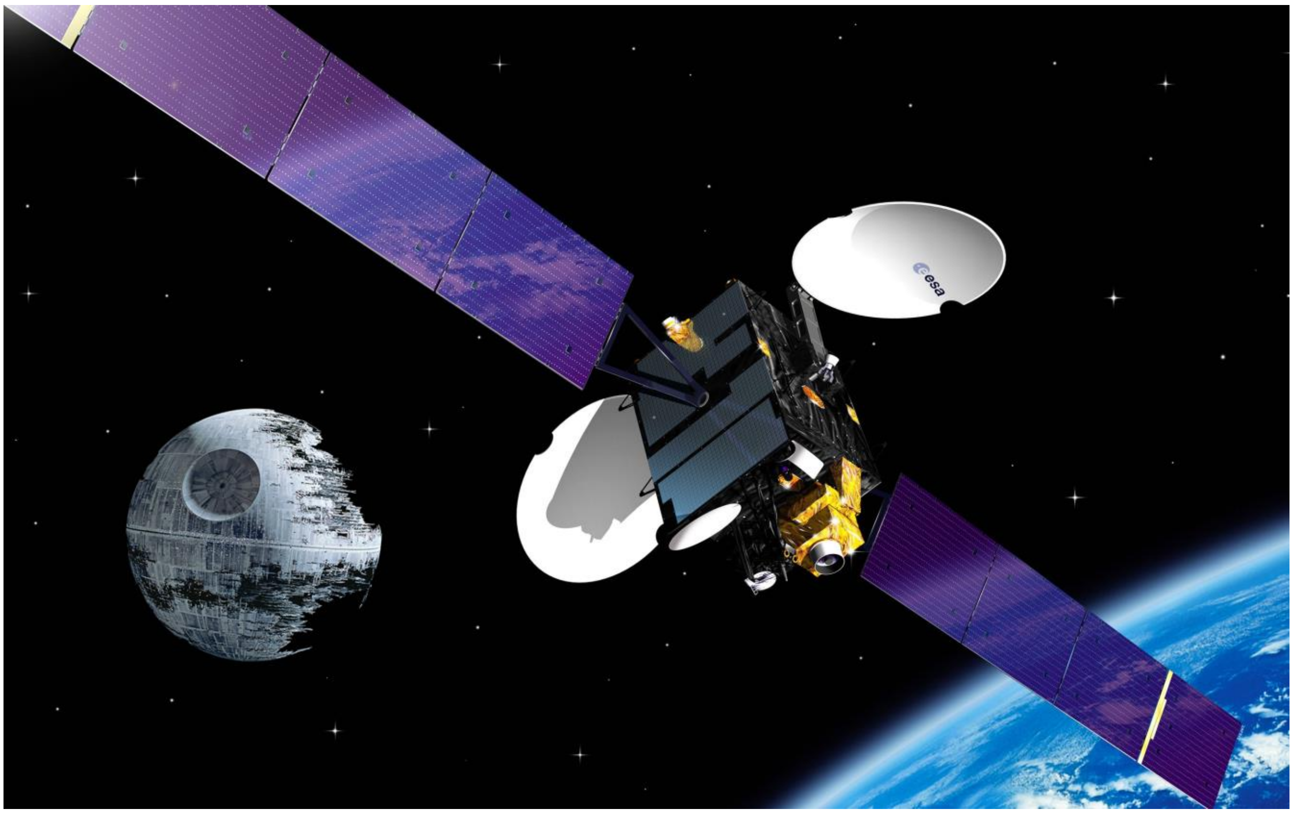
1.1-Description:

This algorithm is an implementation of the « Error Level Analysis algorithm ».

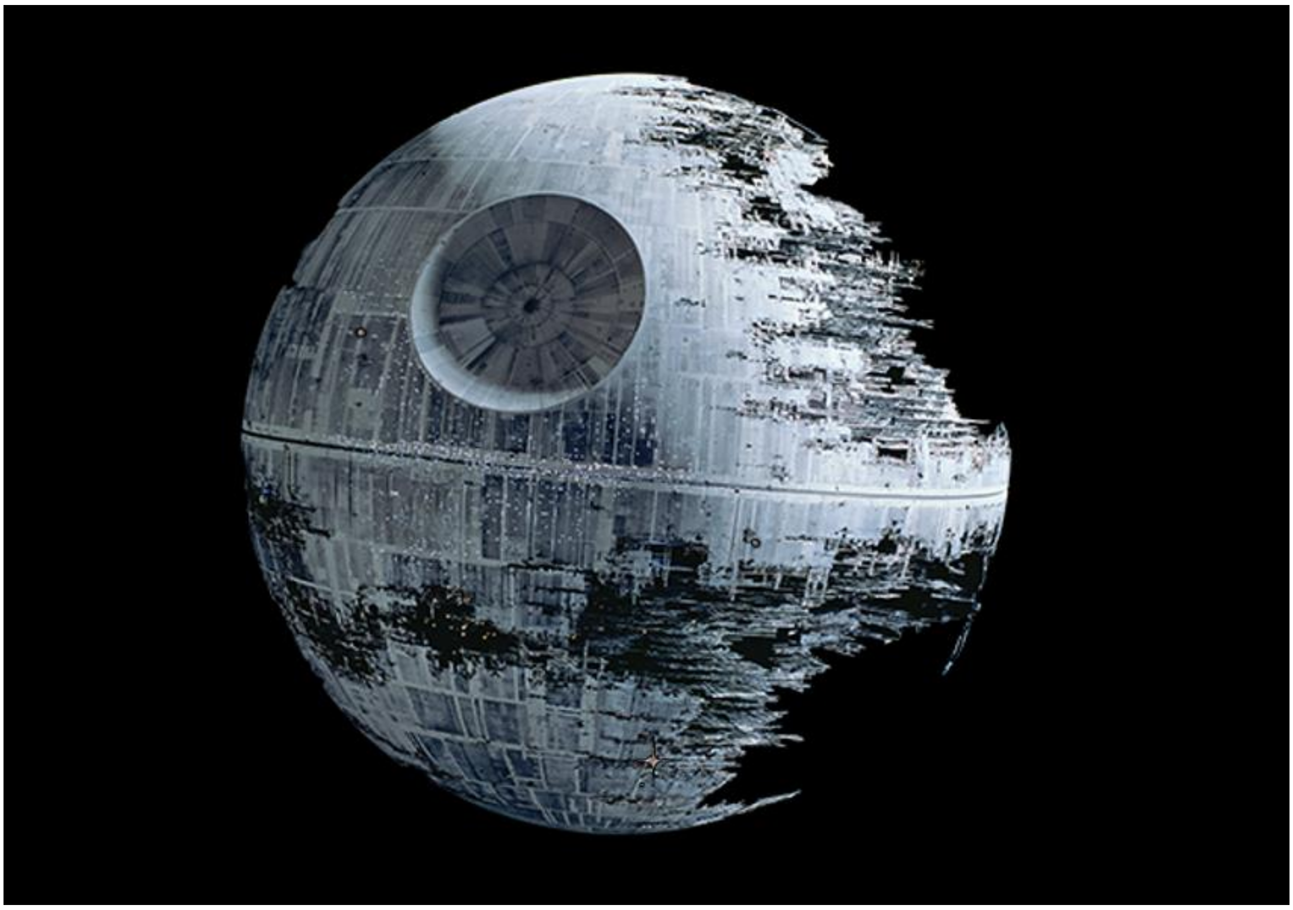
Thus, the algorithm find the different between two images.

Original image :

Forgered image :



The algorithm find the different between the images:



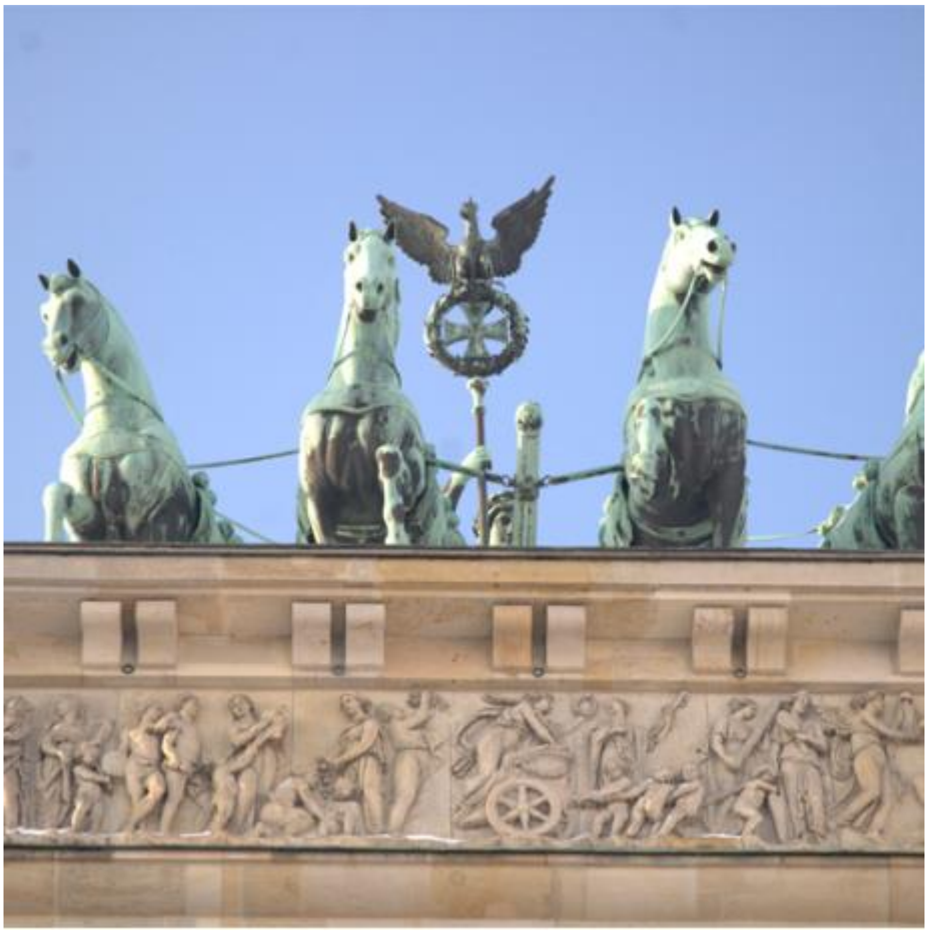
2. Copy-Move-Detection:

This algorithm is an implementation of the « Copy-move-Detection ».

Thus, this python script is an implementation to detect a copy-move manipulation attack on digital image based on Overlapping Blocks.

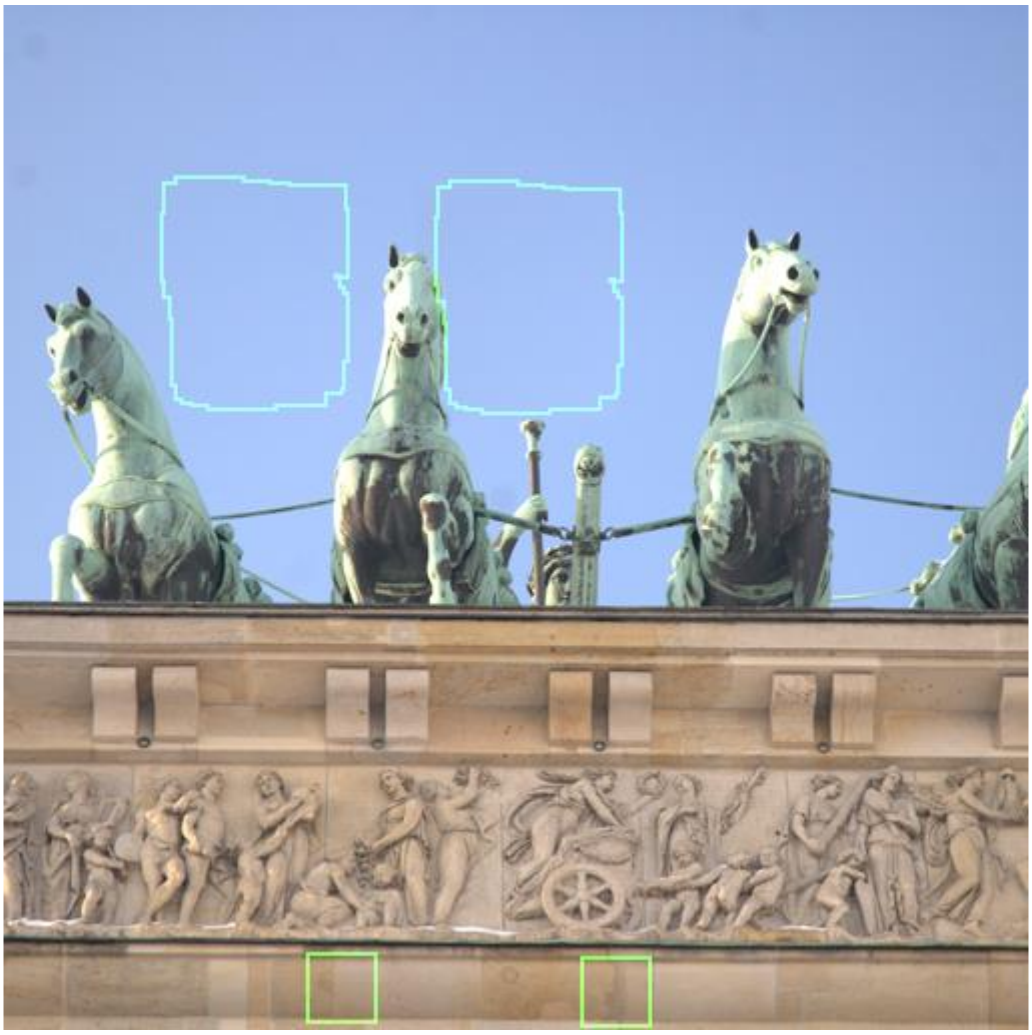
The resulting images are in the following sub-folder: « Algorithms/copy-move-detection/CopyMoveDetection/testcase_result »

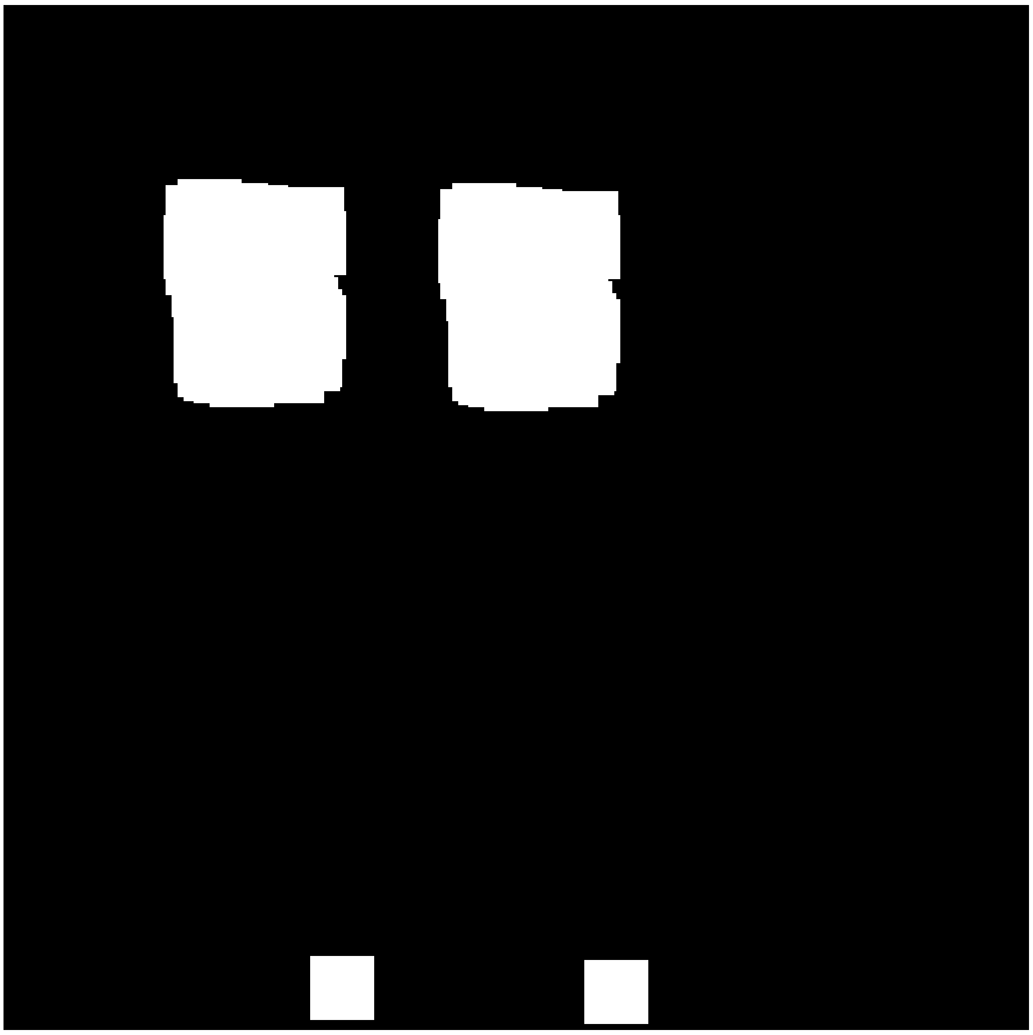
Original image:



Forgered image :

Result :





3. Perceptual Hash:

This algorithm is an implementation of the « perceptual Hash ». This implementation of perceptual hash is based on Neal Krawetz's dHash algorithm.

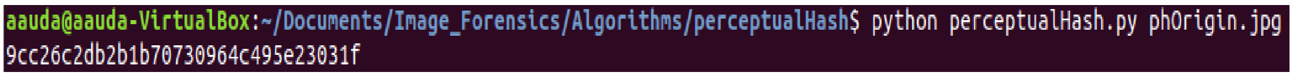
Thus, this python script is an implementation to generate a "difference hash" for a given image. So, the perceptual hash is good for finding exact duplicates and near duplicates, for example, the same image with slightly altered lighting, a few pixels of cropping, or very light photoshopping.

In fact, the resulting hash won't change if the image is scaled or the aspect ratio changes. Increasing or decreasing the brightness or contrast, or even altering the colors won't dramatically change the hash value. Even complex adjustments like gamma corrections and color profiles won't impact the result.

Moreover, the algorithm is very efficient and fast.
The hash values represent the relative change in brightness intensity. To compare two hashes, just count the number of bits that are different (Hamming distance). A value of 0 indicates the same hash and likely a similar picture. A value greater than 10 is likely a different image, and a value between 1 and 10 is potentially a variation.

The algorithm allows therefore to use perceptual hash to generate the difference hash for a specific image. Also, we can compute the bit delta between two images.

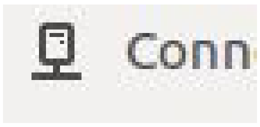
Example of hash computation:



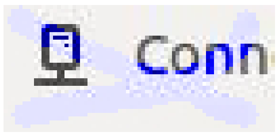
```
aauda@aauda-VirtualBox:~/Documents/Image_Forensics/Algorithms/perceptualHash$ python perceptualHash.py phOrigin.jpg
9cc26c2db2b1b70730964c495e23031f
```

First example of hash comparison:

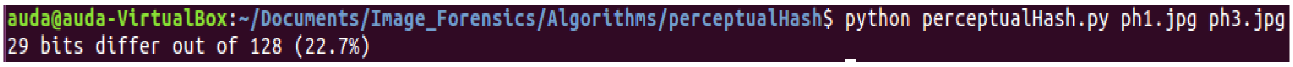
First image:



Second image :



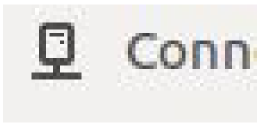
Result :



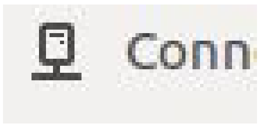
```
auda@auda-VirtualBox:~/Documents/Image_Forensics/Algorithms/perceptualHash$ python perceptualHash.py ph1.jpg ph3.jpg
29 bits differ out of 128 (22.7%)
```

Second example of hash comparison:

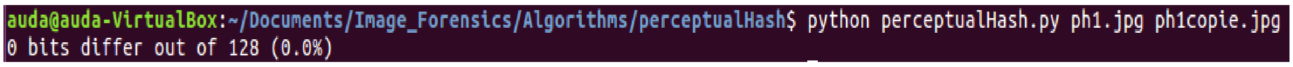
First image:



Second image :



Result :



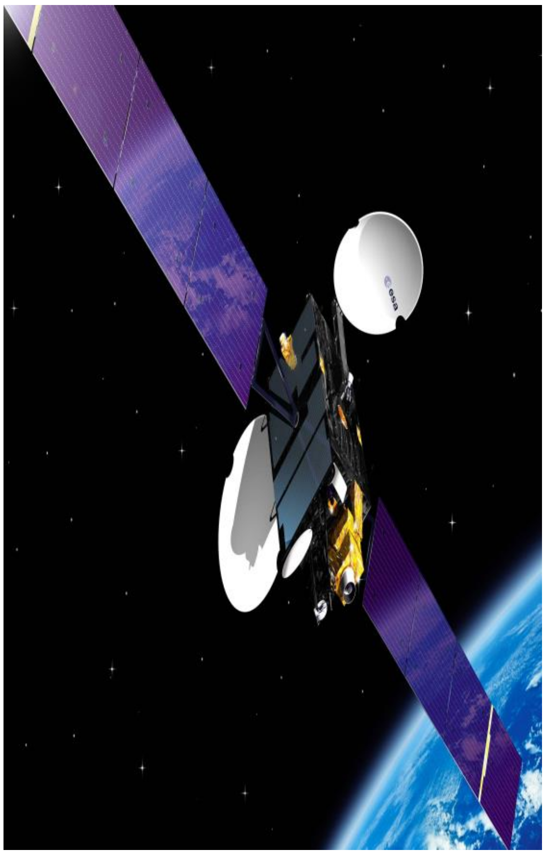
```
auda@auda-VirtualBox:~/Documents/Image_Forensics/Algorithms/perceptualHash$ python perceptualHash.py ph1.jpg ph1copie.jpg
0 bits differ out of 128 (0.0%)
```

Third example of hash comparison:

First image:

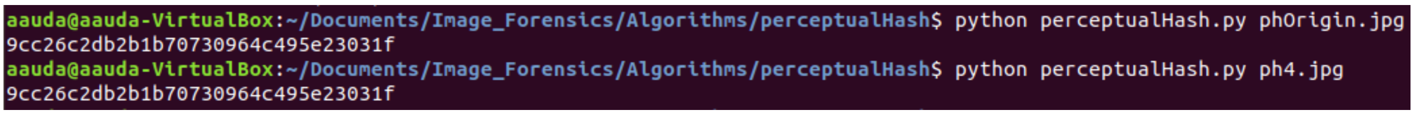
Second image:

(The image is the same as the first but it has been resize and degrade in quality)



Result:

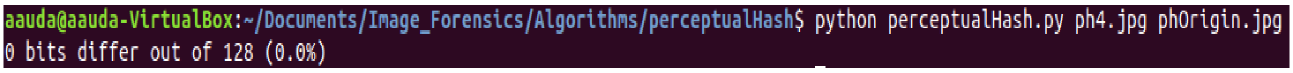
We can compute the hash for the two different images:



```
aauda@aauda-VirtualBox:~/Documents/Image_Forensics/Algorithms/perceptualHash$ python perceptualHash.py phOrigin.jpg
9cc26c2db2b1b70730964c495e23031f
aauda@aauda-VirtualBox:~/Documents/Image_Forensics/Algorithms/perceptualHash$ python perceptualHash.py ph4.jpg
9cc26c2db2b1b70730964c495e23031f
```

So, we find that the hash is the same and therefore, the image represent the same thing.

We can confirm that by the computation of the number of bits that are different (Hamming distance):



```
aauda@aauda-VirtualBox:~/Documents/Image_Forensics/Algorithms/perceptualHash$ python perceptualHash.py ph4.jpg phOrigin.jpg
0 bits differ out of 128 (0.0%)
```