UCLouvain

École polytechnique de Louvain

3 Layers Images Protection and Authentication

Author: **Vincent Wuillaume**
Supervisors: **Benoit Macq, Rony Darazi**
Readers: **François-Xavier Standaert, Benoit Macq, Rony Darazi**
Academic year 2025–2026
Master [120] in Computer Science

Table of contents

[1]Source: https://en.wikipedia.org/wiki/Random_forest

[2]Source: https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/

[3]Source: https://www.geeksforgeeks.org/machine-learning/understanding-scikit-learns-svc-decision-function-and-predict/

[4]https://www.geeksforgeeks.org/machine-learning/how-to-choose-the-best-kernel-function-for-svms/

[5]https://www.geeksforgeeks.org/machine-learning/radial-basis-function-kernel-machine-learning/

[6]https://www.geeksforgeeks.org/machine-learning/voting-classifier/

ABSTRACT

Images are everywhere. On social media, newspaper, on television, … everywhere. Lately, the IA are generating more realistic content than ever. The issue is precisely that is becomes more and more difficult for a human to distinguish which content has been generated from the one that is authentic.

The goal of this Master Thesis is then to propose a method that will help at proving the authenticity of a given image.
The proposed method allows to first secure an image and then to later prove its authenticity by checking the elements that has been introduced in the securing process.

The proposed framework relies on image's metadata, blind watermarking and classification task resulting in an authenticity score to an image. As part of the process, the method also computes a RASH signature, a hash signature that is unique to an image, up to a benign transformation. The computation of the signature is based on the features points detected in the image such that it improves the robustness of the original technique.

According to the experiments carried out, the RASH signature is robust when it should be, and it also detects malicious manipulation. The metadata is also a robust mean to embed data since it has always been successfully retrieved in the performed experiments. Watermark is a robust data method but it shows some sign of weakness when some attacks has been performed of the original images. Finally, the classifier shows good performance when the prediction is done on unseen images from the training dataset. However, if the prediction is done on a completely new dataset, the performance of the classifier decreases significantly. The classification task is then dataset-dependent.

With everything together, the proposed framework has given satisfactory results since the performed set of tests have always return desired results.

Keywords: *RASH, metadata, blind watermark, classification, feature points*

INTRODUCTION AND CONTEXT

Nowadays, everyone sees images on social media, in newspapers, magazine, everywhere. Due to the big rise of the generative models of Artificial Intelligence, it becomes more and more difficult for a human to distinguish what is real and what is not. It has larger implications than just viewing fake content that can make everyone laugh. It also has real societal implications since nowadays, people base their decisions on what they see everyday on social media [1].

To achieve this goal, the proposed method should detect malicious manipulations such as
- add/remove elements from original image
- detect generated images

but the method should also be robust to benign transformation such as
- JPEG compression
- slight blurring
- rotation
- slight cropping

The goal of this Master Thesis is then to provide authenticity proof for a given image. To achieve that goal, a framework in two part will be proposed as end-product of this work. The first part is designed to secure an image while the second will act as a detector and will attribute an authenticity score.

As part of the proposed framework, three Layers trying to guarantee the security of an image have been implemented.
Layer 1 concerns the image metadata, while Layer 2 deals with invisible watermarking. Finally, Layer 3 is a classifier task that tries to distinguish real images from generated ones.
The first part of the proposed method, to secure an image, deals with the first two Layers, where the RASH signature is embedded, respectively in the metadata or as a blind watermark. This part of the proposed framework does not use the Layer 3, as the classifier is purely a detection tool, not a preventive one. Conversely, the second part of the proposed method, which attributes an authenticity score to a given image, uses the 3 Layers.

In the following report, Section 1 will describe the reviewed papers that have been useful to understand the building blocks of the proposed method. A few words will support each of the relevant paper used for this Master Thesis.

Section 2 is the main part of the thesis. It will describe in details all the tools and mechanisms used to construct the proposed method, showing an understanding of the methods used. It will eventually explain how those mechanisms are working together. Figure 1 and Figure 2 outlines the proposed framework in a glance.

Section 3 will discuss the performed tests, obtained results and will give an interpretations about those results.

Section 4 highlights the limitations of the proposed framework.

Finally, Section 6 will present a conclusion and some perspectives for future works in this domain, because it is more than ever a hot topic.

In the first part of the proposed framework, the security information (RASH) related to an image can be putted at two distinct places:

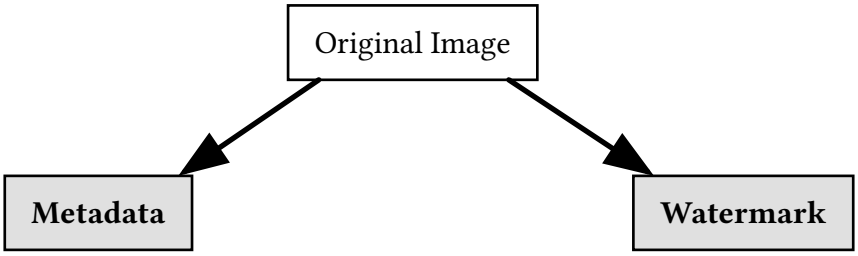


Figure 1: Conceptual Schema for embedding signature

On reception of an image, the three following layers are tested in order to compute the final authenticity score:

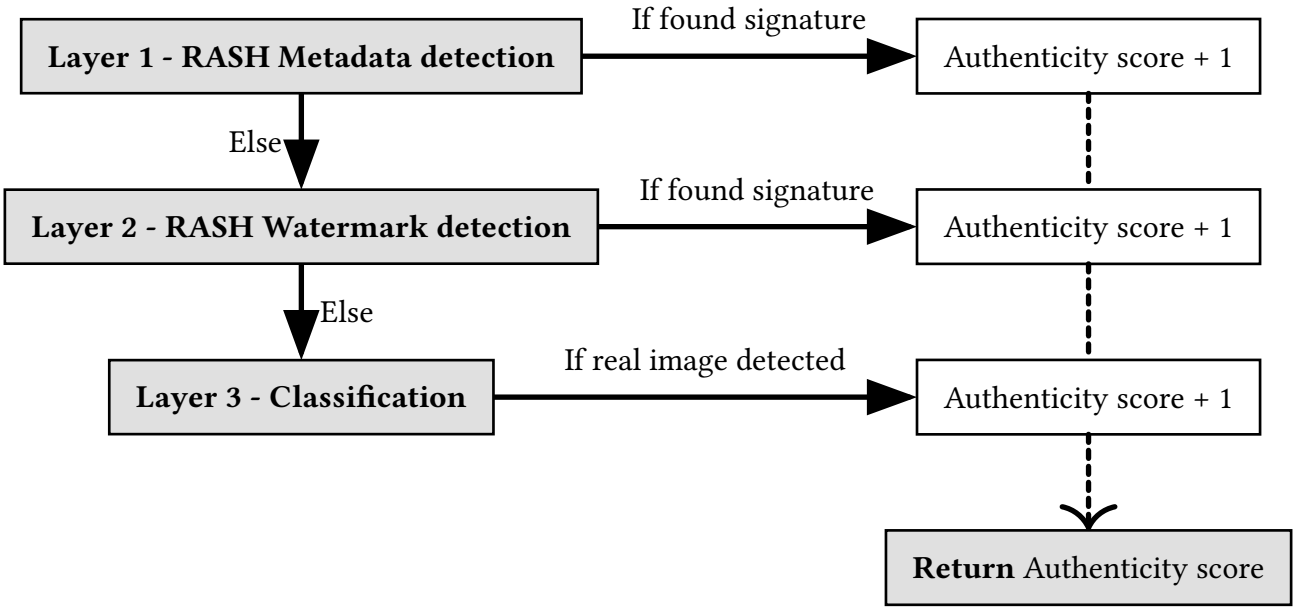


Figure 2: Conceptual Schema for detecting signature

1. RELATED WORK

1.1. Introduction

This first chapter is dedicated to review the relevant papers that have been used in order to first, understand the different methods, then select what will be kept and what will be updated for the proposed framework described in the next chapter.

1.2. Image Signature Computation

Introduction: As part of the proposed framework, the RASH signature discussed in this document is involved in the first two layers, the metadata and the watermark.

Paper: Robust Video Hashing Based on Radial Projections of Key Frames, de Roover et al., 2005

This paper [2] has been the most central part of the proposed framework. It describes, among other things, how a RASH signature is built.

First, a bit of context about the framework's requirements is given:

In computer science, a Hash function is a function that maps a random length input into a fixed length output. It exists multiple type of hash function. Among them, the cryptographic hash, which have the following properties:
- One way function (infeasible to invert)
- Deterministic
- Unique output for a given input

In the context of witnessing the authenticity of an image, these assumptions are a bit too hard, as an image can receive benign transformation such as slight cropping, rotation, blurring, sharpening, compression... In order to solve this issue, De Roover et al. [2] have introduced a new signature that is unique to an image, up to benign transformations (slight cropping, compression, ...), the RASH signature . The RASH name is given for Radial HASH and it acts as an image robust hashing technique based on radial projections.

The algorithm works in 2 steps:
- The radial projections of the image pixels build a RAdial Variance (RAV) vector
- The 40-low frequency discrete cosine transform (DCT) coefficients of the RAV vector are quantized to define the RASH

Definition of the RAV vector

- $\Gamma(\Phi)$ denotes the set of pixels (x, y) that are located on the projection line corresponding to the angle Φ ($0° \leq \Phi < 180°$, so we have 180 projection lines in total).

(x', y') defines the coordinates of the central pixel. $(x, y) \in \Gamma(\Phi)$ if and only if

$$-\frac{1}{2} \leq (x - x').\cos\Phi + (y - y').\sin\Phi \leq \frac{1}{2} \tag{1}$$

- $I(x, y)$ defines the luminance value for the pixel located at (x, y). The projected feature vector $P(\Phi)$, $0° \leq \Phi \leq 180°$ is defined as

$$P(\Phi) = \frac{\sum_{x,y\in\Gamma(\Phi)} I^2(x,y)}{\#\Gamma(\Phi)} - \left(\frac{\sum_{x,y\in\Gamma(\Phi)} I(x,y)}{\#\Gamma(\Phi)}\right)^2 \quad (2)$$

$P(\Phi)$ is thus the variance of the pixels luminance on the line passing through the center of the image and whose orientation is given by the Φ angle.

- The RAV feature vector, $R(\Phi)$, is defined as the central and the normalized version of the projected vector $P(\Phi)$. Let μ and σ defines the mean and the standard deviation of the projected vector. We have

$$R(\Phi) = \frac{P(\Phi) - \mu}{\sigma}, 0° \leq \Phi < 180° \quad (3)$$

Quantization of the RAV vector

- To obtain a compact image digest digest based on the RAV vector, the DCT is used to decorrelate the RAV feature samples. The goal is then to keep only the components of the RAV vector that contain the most of its energy, to preserve discriminating capabilities. After observation, De Roover et al. [2] conclude the energy is more compacted in the low-frequency DCT coefficients.
- Paper defines the proposed image hash, RASH, based on the 40 low-frequency DCT coefficients of the RAV feature vector, denoted $D(n)$, $1 \leq n \leq 40$, the RASH images digest coefficients are defined by

$$D(n) = \sqrt{\frac{2}{N}} . \sum_{\Phi=0}^{N-1} \left(R(\Phi) . \cos\left(\frac{\pi . (2\Phi + 1) . n}{2N}\right)\right) \quad (4)$$

where
- $1 \leq n \leq 40$,
- $0 \leq \Phi < 180$,
- $N = 180$

Each coefficients is quantized on 8 bits so the quantization noise is negligible in front of the noise due to image processing manipulations. It results in a 320 bits RASH image digest.

To be able to consider that 2 signatures correspond to the same image, the correlation between the 2 signatures has to be performed. If the correlation between the 2 signatures overtakes 87%, the 2 signatures are said to be corresponding to the same image. The 87% is explained by multiple experiment described in [2] and authors conclude that this 87% threshold is the optimal value to decide whether two images are visually similar or not.

1.3. Detecting important image's features points

Introduction: As the RASH signature in his original form shows signs of weakness to cropping, it is combined with feature points of the images, which are the points identified as the most important one of the image. This allows to make abstraction of the image background.

Paper: Geometrically Invariant Watermarking Using Feature Points, Bas et al., 2002

The presented paper [3] is intended to propose a new watermarking embedding method, as the "classical watermarking scheme" were not robust to geometrical distortion. The technique presented in the paper relies on the content of the image directly to embed the watermark.

To embed the watermark, paper follows this pattern:
1. detect robust feature points in an image
2. generate a triangular tessellation of the image based on the set of feature points;
3. map a triangular spread sequence (watermark) into each triangle of the tessellation via affine transform;
4. add the mapped sequence on each triangle.

To retrieve the watermark back, the paper follows this sequence:
1. reconstruct the tessellation;
2. map each triangle to the shape of the original triangular watermark;
3. compute the correlation of each mapped triangle with the original watermark;
4. accumulate the correlations to detect the watermark over the whole image

The feature points detection method that has been selected at that time was the *Harris* Detector. Briefly, this is how this method works[7]

1. **Convert image to grayscale**

2. **Compute image gradients**

The method analyzes how the intensity of a small window of the image changes when the window is slightly shifted. It computes gradients (derivatives) to analyzes the changes in horizontal and vertical direction. I_x and I_y respectively indicate the change in the x-direction and the y-direction. Those derivatives are approximated using Sobel operators.

3. **Construct the structure tensor matrix M**

The gradients obtained at the previous step are used to produce the matrix M which indicates the local intensity variation. This matrix is represented as follows:

$$M = \begin{pmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{pmatrix} \tag{5}$$

4. **Calculate the Harris response**

The response is given by

$$R = \det(M) - k * \operatorname{tr}(M)^2 = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{6}$$

where
- λ_1 and λ_2 are the eigenvalues of the structure tensor matrix M. They are measuring intensity variation around a point. λ_1 indicates the variation in one principal direction and λ_2 represents the variation in the other (orthogonal) principal direction
- $\det(M)$ measures the variation in the 2 directions
- k is an empirically determined constant, typically $[0.04, 0.06]$

[7] https://en.wikipedia.org/wiki/Harris_corner_detector

- $\text{tr}(M)$ represents overall intensity change.

5. **Interpretation**

- If R is small: this is likely a flat region
- If R is negative: this is an edge
- If R is a large positive (both λ_1 and λ_2 are large): this is a feature point

As the other methods used in this paper are not relevant for our proposed framework, we will not look in details about these methods.

1.3.1. Harris feature points detection improvement

As the Harris method is now quite old, we wanted to find another feature point detection method, which is more up to date. After research, the Shi-Tomasi feature points detection method, which is quite similar to the Harris feature point detection, has been found similar to the Harris one but presents some improvements. The Harris method is not obsolete but the Shi-Tomasi one is often prefered in practice as this method is better because it is more robust.

It differs by the response function, which is defined in the Shi-Tomasi method by

$$R = \min(\lambda_1, \lambda_2) \tag{7}$$

The idea here is to consider a corner only when the smaller eigenvalue is sufficiently large, when it is larger than a threshold value T, which is specified by the user. It rejects points where one direction is weak (an edge), which can make it more reliable.

1.4. Blind Watermarking foundation

Introduction: The second layer of the proposed framework is dedicated to the retrieval of an invisible watermark in the pixels of the images. This paper introduces an interesting watermarking embedding and detection method that fits the need of the proposed method.

Paper: A DWT, DCT AND SVD based watermarking technique to protect the image piracy, Rahman, 2013

The DWT-DCT-SVD watermarking technique is an excellent choice for the proposed framework as it proposes 2 capital requirements:

- imperceptibility
- robustness

Those 2 essential points naturally led to chose this technique in order to embed and extract the watermarks.

Next, an individual description of the 3 methods will be given. Finally, paper [4] shows how the 3 methods, DWT, DCT and SVD, collaborate to get the final result.

1.4.1. Prerequisites

A. DWT: Discrete Wavelet Transform

The Discrete Wavelet Transform (DWT) is a technique used to compress images by separating important information from details. It works by applying low-pass and high-pass filters

(conceptually, small group of pixels like 2x2 block) to the image, first across rows and then columns.

This produces 4 subbands:

- LL: a low-resolution approximation of the image (low frequencies)
- LH: horizontal details
- HL: vertical details
- HH: diagonal details

The LL subband contains most of the image's energy, while the others capture fine details, which can often be compressed more.

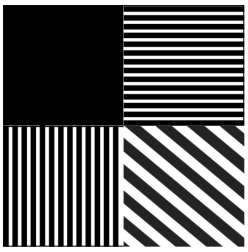


Figure 3: DWT illustration

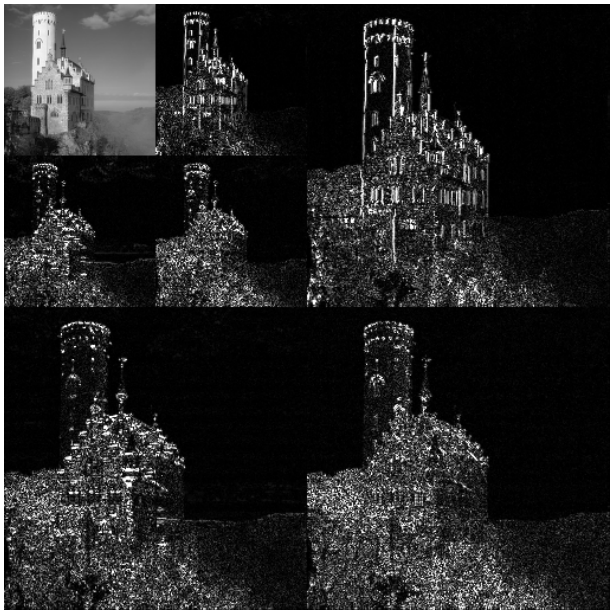


Figure 4: DWT Example[8]

B. DCT: Discrete Cosine Transform

Discrete Cosine transform is a technique used in image compression. It converts image data from the spatial domain (pixels) into the frequency domain. One of its main property is to concentrate energy of the image in the low frequency coefficients. This is briefly how it works:

- Divide the image by square of 8x8 blocks, with the luminance of each of the 64 pixels in that square
- Center each luminance value around 0

[8]Source: https://en.wikipedia.org/wiki/Discrete_wavelet_transform

- DCT transform: Each block is transformed into 64 coefficients, representing the weight of each cosine waves. Usually, higher frequency cosine waves contribute very small to the image, representing details which low frequency cosine waves represents overall shape.
- Quantization: Coefficients are divided by quantization values from the quantization table.
- Encoding: The quantized coefficients are reordered and compressed using using Huffman encoding.

C. SVD: Singular Value Decomposition

The Singular Value Decomposition is a factorization technique that allows to decompose a single matrix into the product of 3 matrices.

Let $X \in \mathbb{R}^{m \times n}$

$$X = U * \Sigma * V^T \tag{8}$$

- $U \in \mathbb{R}^{m \times m}$ is an orthogonal matrix, ($U^T \times U = I$)
- $V \in \mathbb{R}^{n \times n}$ is an orthogonal matrix, ($V^T \times V = I$)
- $\Sigma \in \mathbb{R}^{m \times n}$ with non-negative entries

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & ... & 0 \\ 0 & \sigma_2 & ... & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & ... & ... & \sigma_r \end{pmatrix} \tag{9}$$

where

- $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_m \geq 0$
- *r = rank(X)*

Interpretation

- The columns of U are the left singular values
- The columns of V are the right singular values
- σ_i are singular values, measuring how much each direction is stretched

The singular values indicate the contribution of each rank-1 component to the matrix.

1.5. The DWT-DCT-SVD method

The DWT-DCT-SVD method combines the 3 of the previously described methods.

1.5.1. Embed the watermark

Schematically, an encoding of the watermark follows this flow:

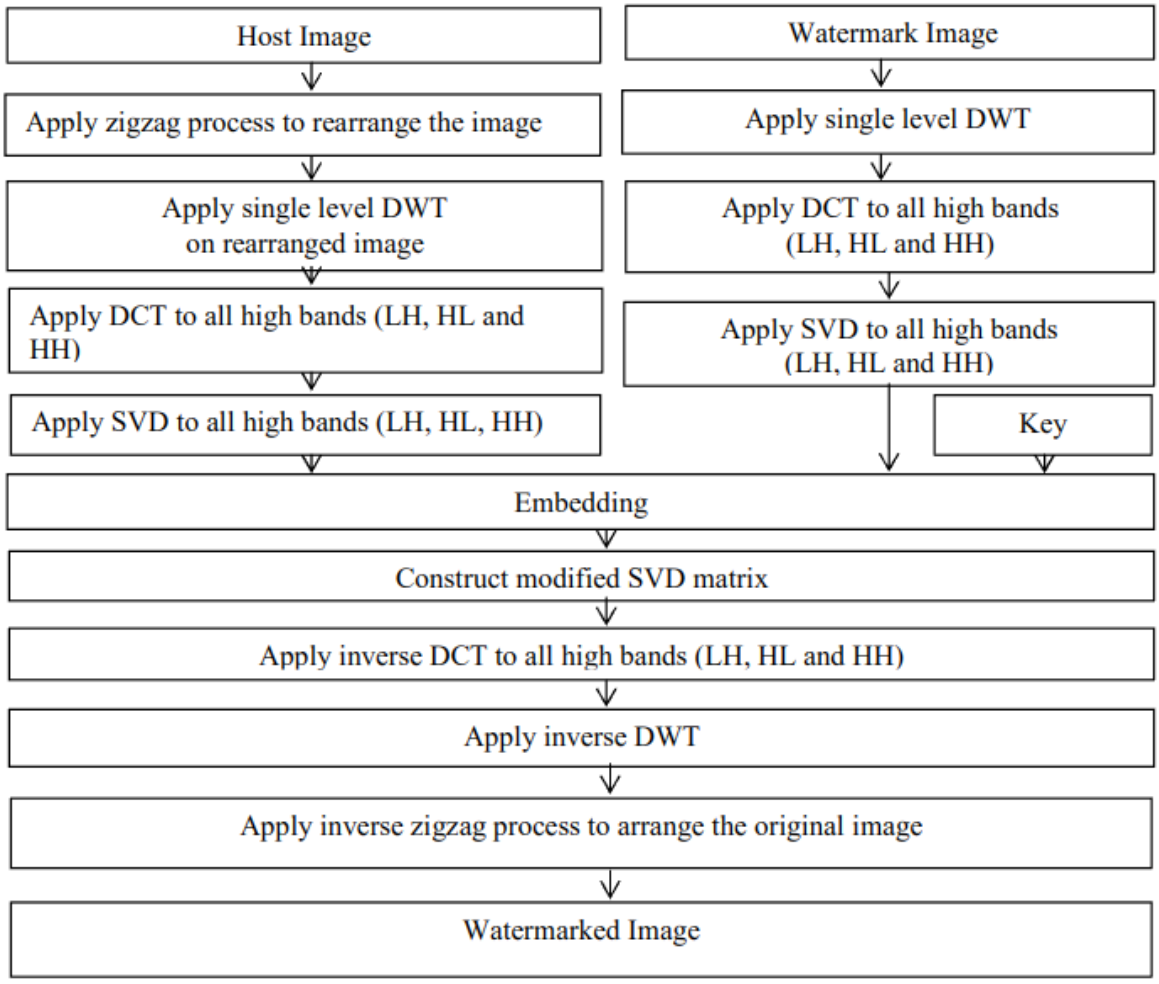


Figure 5: DWT-DCT-SVD embedding schematic[9]

1. Apply **DWT** to Host image and Watermark
2. Apply **DCT** to high frequency bands (LH, HL, HH) of Host image and Watermark
3. Apply **SVD** on High Frequency (LH, HL, HH) for both images
4. Modify SVD values by combining Host and Watermark respective SVD values

$$S_{wi} = S_i + \alpha S_w, i \in [1, 3] \tag{10}$$

$$X = USV^T \Rightarrow X' = US_{wi}V^T \tag{11}$$

with $\alpha \equiv$ strength of the watermark

5. Construct modified SVD matrices
6. Apply inverse DCT to high bands SVD matrices

[9]Source: A DWT, DCT AND SVD BASED WATERMARKING TECHNIQUE TO PROTECT THE IMAGE PIRACY, Md. Maklachur Rahman, 2013

7. Apply inverse DWT to all bands

⇒ obtain watermarked image

1.5.2. Extract the watermark

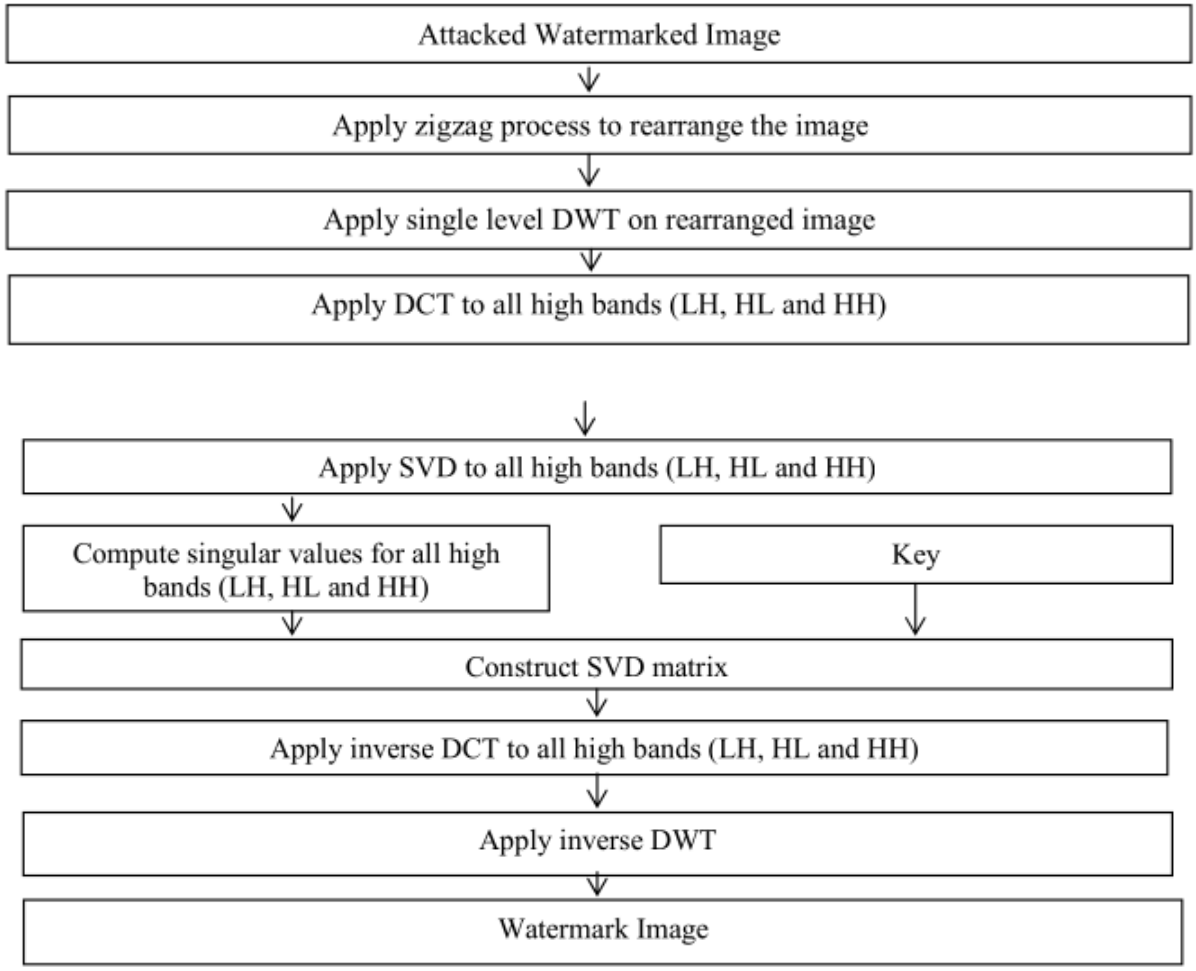


Figure 7: DWT-DCT-SVD extracting schematic[10]

1. Apply DWT to the watermarked image
2. Apply DCT on High Frequency (LH, HL, HH) bands
3. Apply SVD on High Frequency (LH, HL, HH) bands
4. Modify SVD values using

$$S_w = \frac{S_{wi} - S_i}{\alpha}, i \in [1, 3] \tag{12}$$

and construct SVD matrices

5. Apply Inverse DCT to high bands SVD matrices
6. Apply Inverse DWT to all bands

⇒ Get the watermark back

[10]Source: A DWT, DCT AND SVD BASED WATERMARKING TECHNIQUE TO PROTECT THE IMAGE PIRACY, Md. Maklachur Rahman, 2013

Note: the purpose of the zigzag rearranging process is not clearly specified in the paper. An intuition that comes in mind is too avoid to be impacted by a local attack on the watermark.

1.6. Finding the right model to perform the classification task

Introduction: The third layer of the proposed framework is a classifier that tries to detect real from synthetic images. This paper evaluates some classifier models and eventually selects one that is best among the others at performing the task.

Paper: Performance Comparison of Deep Learning Models for Computer Generated Image Detection, Sychandran et al., 2023

This paper [5] is a review about comparing different classifier models trying to distinguish Computer Generated Images (CG) from Photographic Images (PIM).

In the precise working context, it perfectly matches the needs of the proposed framework, as one of the 3 layers is a classifier that have to distinguish CG images from PIM.

The paper evaluates 4 models:
- *ResNet50 with SVM*
- *CNN-R*
- *DenseNet201 with transfer learning*
- *Hybrid Xception Ensemble mode*

on 3 distinct datasets:
- the *Rahmouni* dataset: 1800 PIM and 1800 CG images
- the *Columbia* dataset: 800 PIM and 800 CG images
- the *DSTok* dataset: 4850 PIM and 4850 CG images

The comparison between the models has been done with 4 metrics, namely:

- Precision (fraction of relevant instances among the retrieved instances):

$$P = \frac{T_p}{T_p + F_p} \tag{13}$$

- Recall (the fraction of relevant instances that were retrieved):

$$R = \frac{T_p}{T_p + F_n} \tag{14}$$

- F1 Score (hormonic mean, combines recall and precision[11]):

$$F1 = \frac{2 * P * R}{P + R} \tag{15}$$

- Specificity (the ratio of correctly classified negative instances to the total actual negative instances[12]):

[11] ensure that a classifier makes good predictions of the relevant class (good accuracy) in sufficiently large numbers (good recall), Source: https://fr.wikipedia.org/wiki/F-mesure

[12] Source: https://saimanojyarlagadda.medium.com/understanding-sensitivity-specificity-precision-recall-f1-score-and-confusion-matrix-in-binary-495b66447621

$$S = \frac{T_n}{T_n + F_p} \tag{16}$$

After experiments, authors conclude that the hybrid Xception ensemble model outperforms all tested models for all metrics. This is why the choice of implementing this model for the classification task of the proposed framework has been made.

1.7. Summary

In this chapter, methods about RASH signature computation, feature points detection, watermarking embedding and detection method and classifier models has been presented.

From [2] about the signature computation, the algorithm description of the RASH method has been kept as it fits the need of the desired framework. It is described as robust against benign modification and and this is exactly what the proposed framework want to achieve for the signature computation.

From [3] about the feature points detection, the idea of extracting feature points of the image has been found interesting. Indeed, as during the experiments the RASH implementation has shown some sign of weakness for cropping modification, the idea of computing the RASH only for the most important part of the image has been found highly relevant. However, the paper is using the Harris method for the feature points extraction. Despite this is an old method, the Harris method is not outdated but a more robust method for feature points extraction has been found, namely the Shi-Tomasi method. This feature points extraction method has then been kept for the proposed framework.

From [4] about the watermarking technique, the DWT-DCT-SVD idea is kept. That method will not be re-implemented in the proposed framework because an already implemented Python version of that watermarking embedding and extraction method has been found. Moreover, the choice of using the existing method has been evaluated more relevant, as the method shows some interesting robustness properties.

From [5] about the classifier models, the hybrid Xception ensemble model has been kept. Since it is described as outperforming the other tested model, this is this model that will be implemented in the proposed framework. As a reminder, this classifier is a majority voting classifier that regroups the Random Forest classifier, the KNN one, the SVC with a linear kernel one and finally the SVC with a RBF classifier.

2. PROPOSED FRAMEWORK - 3 Layers Images Protection and Authentication

2.1. Introduction

In this chapter, the main part of this Master Thesis will be described, the proposed framework. It explains step-by-step how the method is built and how it is implemented. More details about the used techniques that were not presented in the Related Work chapter will be given here.

Figure 8 and Figure 9 are showing the steps followed of the proposed framework are presented next as well as the used methods it relies on.
First, the prevention part of the framework is presented with 2 ways of securing an image.

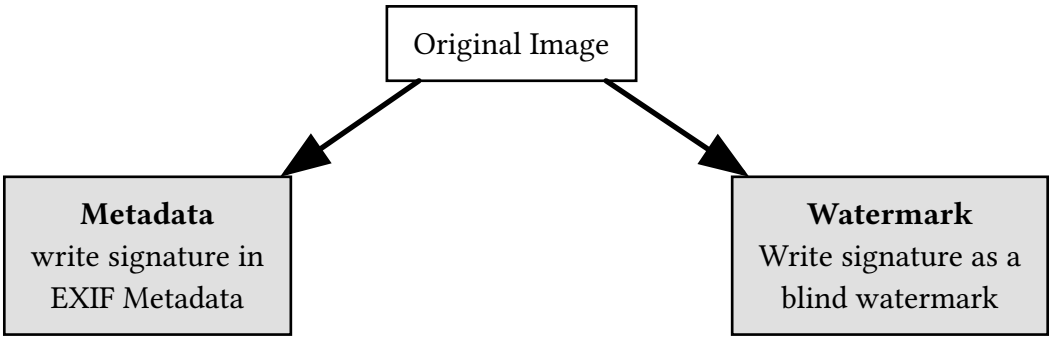


Figure 8: Conceptual Schema for embedding a signature

On reception of an image, the detection part of the proposed framework is built as follow. Each technique used inside a given layer is precised in the corresponding frame of the scheme.

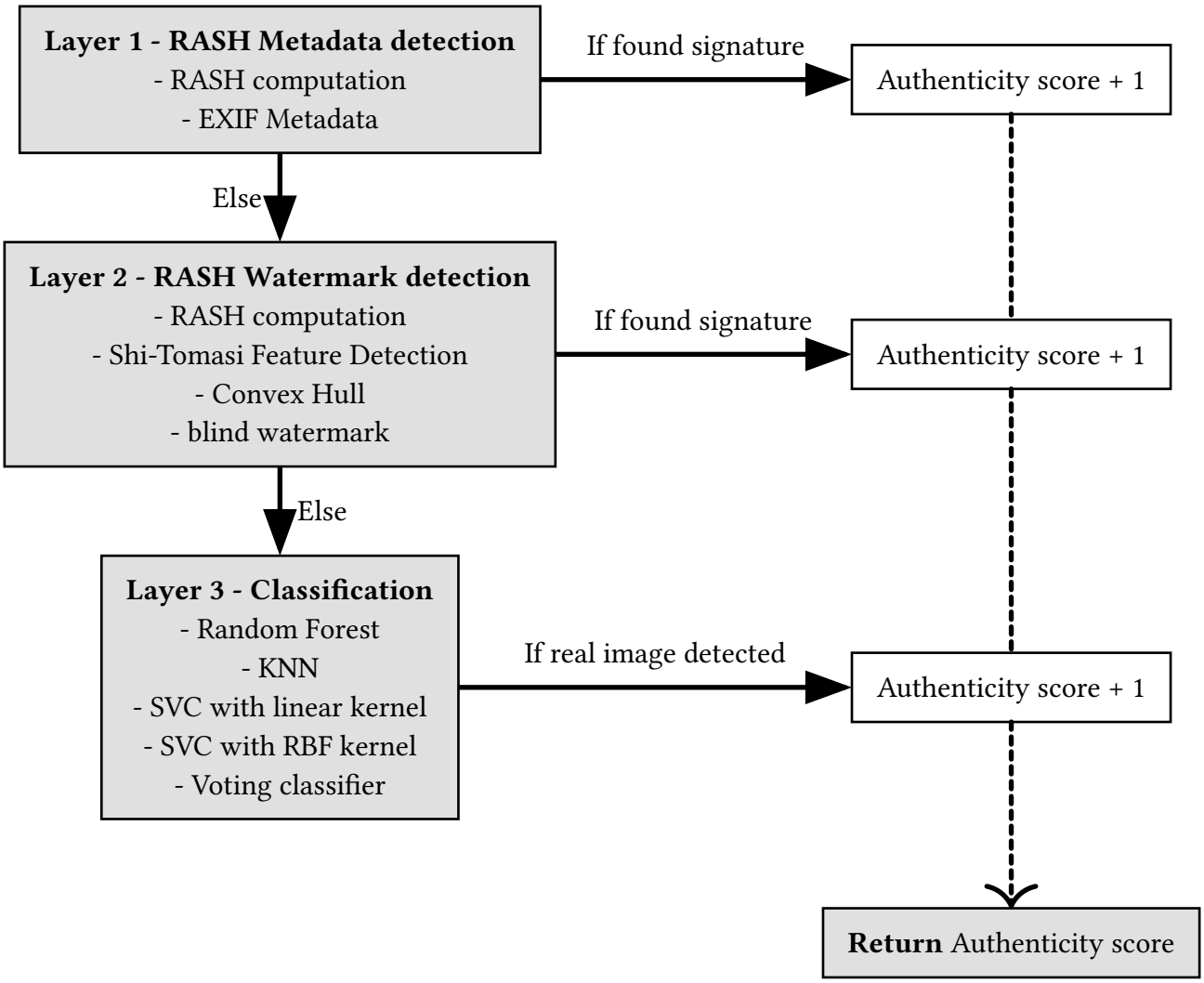


Figure 9: Conceptual Schema for detecting signature

2.2. Proposed framework

This sub-section presents the entire proposed framework as well as the used techniques for each layer.

2.2.1. Embedding side (prevention)

In order to add protection to a new image, it can be achieved with 2 distinct layers, which are the **Metadata** and the **Watermark**. Because an important aspect of this Master Thesis has been to revisit the *RASH* method from [2], which will be explained more in detail later, this signature technique has been chosen as the securing information to look for in the proposed framework.

Layer 1 - Metadata;

- Compute RASH
  - Search for image *Feature Points*: **Shi-Tomasi** Feature Dectector
  - Construct convex hull: **Open CV convexhull()** method
  - Compute **RASH** from pixels inside the convex hull
- Write **RASH** signature into EXIF image metadatas

Layer 2 - Watermark;

- Compute RASH
  - Search for image *Feature Points*: **Shi-Tomasi** Feature Dectector
  - Construct convex hull: **Open CV convexhull()** method
  - Compute **RASH** from pixels inside the convex hull
- Embed RASH inside entire image

2.2.2. Decoding side (detection)

The second aspect of the Master Thesis is an image going through the three following layers, to prove the authenticity of that image. After this step, an authenticity score is given for the image, between 0 and 3, indicating the level of trust a person can have regarding the image authenticity.

Since each of the 3 layers participates equally to the final authenticity score, a single point per layer is given to the authenticity score.

Layer 1 - Metadata;
- Read into EXIF image metadatas and look for **RASH** signature
- Compare a fresh RASH with the one found in the metadata if any

↓

Layer 2 -Watermark;
- Part A
  - Compute a fresh RASH signature
- Part B
  - Extract embedded invisible watermark (if a watermark is found, the framework checks whether it respects the criteria of a RASH signature)
- Part C
  - Compare extracted RASH with freshly computed one

↓

Layer 3 - Classifier;
- Xception Ensemble from Section 1.6
- Random forest classifier
- KNN classifier
- SVC with linear kernel classifier
- SVC with RBF kernel
- Voting classifier (Majority voting)

2.3. Used Datasets

Model	Validation images provenance	Number of training images	Prediction accuracy
Model trained with *DS1*	*DS1*	48000	88.79%
Model trained with *DS1*	*DS2*	48000	58.195%
Model trained with *DS2*	*DS2*	120.000	93.12%

Table 1: accuracy of the watermark detection

This subsection talks about the datasets used to train the proposed framework.

As this Master Thesis is related to images, a dataset with natural and synthetic images had to be found in order to work on it.
The first dataset used in the project has been found on the Kaagle website[13]. The name of the dataset is **"ai-generated-images-vs-real-images"**, *DS1* in Table 1. The choice has been made to use this one as a first dataset for multiple reasons:

- Permissive dataset: this dataset has a MIT license. This indicates that we can do whatever we want with the images, as long as we mention that images comes from this dataset. As part of it, it tells that we can modify images, which is actually the goal of this Master Thesis.
- Big dataset: The primary use of the dataset is to train and test the classifier, which distinguish real images from artificially generated one. For this purpose, this dataset fits exactly what the needs of the work. Moreover, this dataset contains a lot of images from multiple sources, which is desirable to train our classifier.
  - The dataset contains 30,000 real images: 22,500 from Pexels[14] and Unsplash[15], that are free stock of images. The other 7,500 real images comes from WikiArt[16], which is a dataset that can be used only for non-commercial research purpose.
  - The dataset contains also 30,000 artificially generated images: 10,000 with the Stable Diffusion text-to-image generative model, 10,000 with the MidJourney one and 10,000 with the DALL-E one.
  - Partition: The dataset is natively downloadable with a 80% - 20% proportion of training set and testing set, which is in general what is used for a classification task.
- Recent: this dataset is expected to be updated every year, as the Kaagle webpage indicates. So we hope that the newly added feature for image generation by AI will be covered each time the dataset is updated.

We obtained good detection results with this dataset (88.79% of right prediction on validation set), even without fine-tuning the classifier models used, which was a good sign.
In order to confirm the classifier results, the model should also be tested on another dataset.

[13]https://www.kaggle.com/datasets/tristanzhang32/ai-generated-images-vs-real-images
[14]https://www.pexels.com/
[15]https://unsplash.com/
[16]https://www.wikiart.org/

A good candidate to achieve that needs is another dataset found on Kaagle, which is **"CIFAKE: Real and AI-Generated Synthetic Images"**[17], *DS2* in Table 1, which contains 60,000 real images (Krizhevsky & Hinton's CIFAR-10 dataset) and 60,000 artificially generated images (Stable Diffusion version 1.4). The dataset is still used in recent project so it is not an outdated dataset. Additionnaly, it contains a lot of image, which makes the training interesting.

Unfortunately, the performance of the original model trained with *DS1* has not been that good when it has come to perform the prediction task on the test set of the second dataset, with only 58.195% of good prediction, barely more than randomly guessing the output class. However, when training a new model on the training set of this second dataset and then performing a prediction task on the test set, the model gives 93.12% of good prediction. The conclusion has been drawn that one dataset is not the other, and the generating model have their own specificity. As a consequence, it is not relevant to fine-tune the hyperparameters of the classifier to limit overfitting on a single dataset.

[17] https://www.kaggle.com/datasets/birdy654/cifake-real-and-ai-generated-synthetic-images

2.4. Layer 1: Signature Identification

The first layer of the proposed framework is to look for a RASH signature in the image metadata. The metadata standard for images is the EXIF metadata.

This way of dealing with the signature is relevant because the pixels of the image are not modified. Also, metadata give additional details that belongs to an image. As explained in the introduction schematics, the retrieved signature, if so, is compared with a freshly computed RASH signature. In that context, it does not matter if the field that contains the signature is modified since it will not correspond to the fresh signature, and it will not be considered as a valid signature for that image.

2.4.1. Reading *EXIF* metadata

As the EXIF (exchangeable image file format) standard have defined fields such as

- Camera settings: aperture, ISO, focal length, …
- Image metrics: dimensions, …
- Date and time informations
- Location informations
- Descriptions
- Copyright informations
- …

In consequence, we cannot create a new "signature" field. But after analysis, we found that we could put the signature in the ***user_comment*** field. It allows to write in the metadata without removing useful informations about the image. Since it is possible to write, and therefore create false EXIF metadata for an image, the framework cannot only rely on the fact that there is or not metadata.

For example, with this simple library `libimage-exiftool-perl`[18], we are able to create false metadata with this line of code:

```
1 exiftool -Make="NIKON CORPORATION" \
2          -Model="NIKON D850" \
3          -FocalLength="24 mm" \
4          -ExposureTime="1/250" \
5          -ISO=64 \
6          image.jpg
```

As a consequence, we have to be really careful with metadata. It is not enough for an image to prove that it is authentic by the mere presence of metadata. Comparing the found signature, if any, with a freshly computed one is the only relevant information that can be extracted with metadata.

Since the proposed framework is written in Python, it was more easy to find a Python library that allow to write in the metadata. The *exif* Python library[19] is an easy to use library to

[18] https://www.kali.org/tools/libimage-exiftool-perl/

[19] https://pypi.org/project/exif/

manipulate, read and write the metadata. It is with this library that the signature will be written during the protection process as well as read during the authentication process.

2.5. Layer 2: Blind Watermarking Detection

In this step, an invisible watermark is tried to be detected from the image pixels. As a reminder, the watermarking embedding and extraction algorithm of the framework works with the DWT-DCT-SVD method. Reminders about that method can be found in Section 1.4. Thanks to the *blind_watermark* Python library, the watermarks are easily embedded and extracted from the images.

For robustness purpose, it has been decided to not calculate the RASH on all image pixels. Instead, [3] in Section 1.3 has proposed an interesting idea of extracting the feature points from the image, detecting important subjects in it. As a quick reminder, the technique introduced in this paper works in 3 steps:

1. Find feature points in the images thanks to Harris detector
2. Compute triangulation with the extracted feature points thanks to Delauney tesselation
3. Embed a part of the watermark in each triangle of the tesselation

From this technique, the idea of the feature points has been kept, except that the Harris feature point detector has been improved with the Shi-Tomasi feature points detector.

Our invisible watermarking embedding algorithm works like this:

1. Extract feature points of the image with Shi-Tomasi detector

Figure 10: Feature points detected on Lenna photo with the following parameters:
Number of points: 150
quality level: 0.01
minDIstance: 10

2. Compute the convex Hull with the feature points and thanks to the `convexHull()` Python method from OpenCV[20]

[20]https://docs.opencv.org/3.4/d7/d1d/tutorial_hull.html

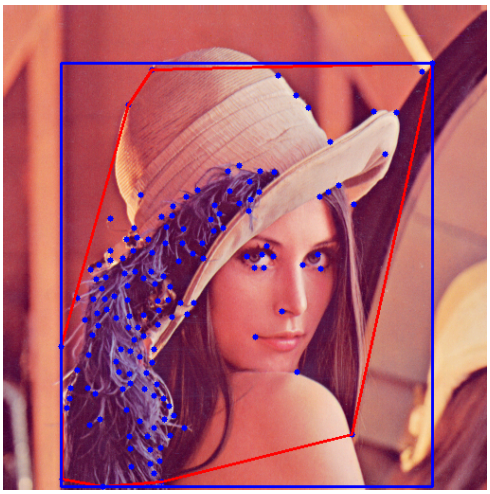


Figure 11: Computation of the Convex Hull and the rectangle around it to find the center which is needed to compute RASH

3. Compute RASH only with pixels that are located inside the Convex Hull (inside red border)
4. Embed the watermark in the entire image for robustness reason, with the Python `blind_watermark` library

In the other way around, to extract and verify the watermark:
1. Extract the watermark thanks to the Python `blind_watermark` library
2. Extract feature points of the image with Shi-Tomasi detector
3. Compute the convex Hull with the feature points and thanks to the `convexHull()` Python method from OpenCV[21]
4. Compute RASH only with pixels that are located inside the Convex Hull
5. Compare extracted and freshly computed RASH

During the framework construction, the Delauney tesselation method has been explored, which is implemented in Python in the Scipy package[22]. But thanks to the Convex Hull, the tesselation is not needed anymore since the convex hull is computed only with the points extracted during the feature points detection.

2.6. Layer 3: Classifier Distinction

As motivated in the Related Works chapter, the hybrid Xception ensemble model from the paper in Section 1.6 is used. As a reminder, it combines 4 existing well-known classifier and apply a majority voting on their respective decisions in order to get the final decision.

We will review a bit more in details the different classifiers in the following subsections.

2.6.1. Prerequisites

A.Random Forest Classifier[23]

The first classifier is the random forest classifier.

[21]https://docs.opencv.org/3.4/d7/d1d/tutorial_hull.html
[22]https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Delaunay.html
[23]Source: https://en.wikipedia.org/wiki/Random_forest

Let's first define a Decision tree[24]. A decision tree in machine learning is a supervised learning approach, used as a predictive model to make conclusions from observations. We can distinguish 2 types of decision trees:

- Classification trees: target variables can take value from a discrete set
  - leaves represent class labels
  - branches represent conjunctions of features that lead to those class labels
- Regression trees: target variable can take continuous values

So the focus is on classification trees since the target variable is just YES/NO, such as an image is a photographic image or not.

Because deep decision tree often have high variance and can overfit the training data (learn by heart the given data and not be able to apply the learning on fresh datas), random forest is a way to averaging over multiple decision trees, trained on different part of the training set aiming to reduce the variance.

Next, the *bagging* has to be introduced, or *bootstrap aggregating* in other words. It is the algorithm that is used for random forest training. Given a training set $X = x_1, ..., x_n$ with corresponding labels $Y = y_1, ..., y_n$, bagging repeatedly (B times) selects a random sample with replacement of the training set and build trees from these samples.

For b = 1, ..., B:
- Sample n training example from X and corresponding Y, say X_b, Y_b
- train a classification tree, f_b on those X_b, Y_b

Given fresh, unseen data, the prediction can be made by majority voting of all individual trees prediction on this new data, x':

$$\hat{p} = \frac{1}{B} \sum_{b=1}^{B} f_b(x') \tag{17}$$

The parameter B of training sample is a free parameters and can vary from few hundred to several thousand.

It leads to final random forest classifier. It is a special type of bagging that it use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. To motivation for this is to break correlation between the different trees. For example, if one feature is a strong predictor for our target variable, this feature will be selected in many of the B trees. Typically, for a classification problem with p features, $\sqrt{\text{p}}$ (rounded down) features are used in each split.

B.KNN Classifier[25]

KNN stands for K nearest neighbors, and it is also a supervised learning method. Given a new data point, it identifies the k points that are the closest to it and make a prediction based on the majority class of those nearest points.

[24]Source: https://en.wikipedia.org/wiki/Decision_tree_learning

[25]Source: https://www.geeksforgeeks.org/machine-learning/k-nearest-neighbours/

In order to select a good k, a tradeoff has to be found between
- a large k make the prediction more stable if the data has a lot of noise or outliers (errors)
- if k is too large, we can miss important pattern which results in underfitting

$\Rightarrow$ k should be chose carefully according to data

A good way to find a good k value is to use m-fold cross-validation. This means dividing the dataset into m parts. The model is trained on some of these parts and tested on the remaining ones. This process is repeated for each part. The k value that gives the highest average accuracy during these tests is usually the best one to use.

Finally, the metrics used to find the closest points are the more often

1. Euclidian distance: straight-line distance between two points

$$d(x, X_i) = \sqrt{\sum_{j=1}^{n} \left(x_j - X_{ij}\right)^2} \tag{18}$$

or

2. Manhattan distance: total distance you would travel if you could only move along horizontal and vertical lines

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{19}$$

C.SVC Classifier[26]

The next two classifiers are using the same base as a classifier but differs by a key parameters, the kernel. We will first introduce the SVC classifier in general. Then we will review the role of this kernel parameter and finally, we will explain the key difference between those two instances of the kernel parameter for the SVC classifier.

SVC stands for Support Vector Classifier and it is an implementation by Scikit-Learn of SVM (Support Vector Machinie)[27] classification algorithm. Support Vector Machine is a supervised learning model and its goal is to find the best hyperplane (the boundary) separating the points of different classes in a multi-dimensional space. Key concepts of a SVM[28] are

- Support Vector: Closest points to the hyperplane. They play a key role in determining the hyperplane itself as well as the margin
- Margin: The distance between the Support Vector points and the hyperplane boundary. The goal of the SVM is to maximize this margin to obtain the most significant result
- Hard margin: Hyperplane that perfectly separate the data without misclassification
- Soft margin: Allow misclassification by balancing between margin maximization and giving penalties when data is not perfectly separable

[26]Source: https://www.geeksforgeeks.org/machine-learning/understanding-scikit-learns-svc-decision-function-and-predict/

[27]https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[28]https://www.geeksforgeeks.org/machine-learning/support-vector-machine-algorithm/

- C: A regularization term defining the amount of penalty given for a misclassification. Higher C means bigger penalty.
  - A high C indicates an aggressive fit of the training data
  - A small C indicates more regularization
- **Kernel**: A function that maps data to a higher dimension space such that SVM can handle non-linearly separable data

The SVC implementation proposed by Scikit-Learn has 2 interesting methods:

- `decision_function`: it returns a raw measure of distance between each point of the input data and the hyperplane, known as the *decision score*, indicating how far samples (input points) are from the hyperplane.
  - positive value: one class
  - negative value: other class
  - near 0: close to the hyperplane
- `predict`: it returns a class label for each point of the input data. This method is often preferred in the machine learning classification problems.

C1. SVC Linear Classifier[29]

The kernel classifier is the simplest kernel version of the SVC classifier. It is defined as

$$K(x, y) = x \cdot y \tag{20}$$

which is the dot product between x and y, the input features vectors, and measure the similarity between those 2 vectors.

With the linear kernel, we assume that the data is linearly separable. Also, the linear kernel performs well when the data are high-dimensional, when the data is sparse and when classes are approximately linearly separable because otherwise, having more features could imply overfitting.

C2. SVC RBF Classifier[30]

RBF stands for Radial Basis Function. It is one of the most widely used kernel with SVM. It is defined as

$$K(x, y) = \exp(-\gamma * \parallel x - y \parallel^2) \tag{21}$$

with x and y being again the input features vectors, γ is a positive parameters indicating the smoothness of the decision boundary, $\parallel x - y \parallel$ is the Euclidean distance between x and y.

- A high γ value aims at finding localized influence, drawing complex boundary, but it is prone to overfitting.
- A small γ value aims a finding more global influence, with smoother boundary, but it can be prone to overfitting.

[29] https://www.geeksforgeeks.org/machine-learning/how-to-choose-the-best-kernel-function-for-svms/
[30] https://www.geeksforgeeks.org/machine-learning/radial-basis-function-kernel-machine-learning/

Intuitively, it measures the similarity between data points based on their Euclidean distance in the input space.

The biggest advantage of using a RBF classifier is because it allows to obtain nonlinear decision. It also captures complex relationships without prior knowledge of data.

D.Voting Classifier[31]

When using multiple machine learning models, a discriminating criterion is needed between the individual results produced by the multiple models that leads to the final prediction.

We call *ensemble learning* a model that combines multiple base models with the goal of obtaining more accurate prediction. Such ensemble learning aims at reducing the bias and the variation of a single model while increasing overall performance.

The prediction of the voting classifier, which is itself considered as a machine learning model is based on the class with the highest likelihood of becoming the output. That output is based on the largest majority of votes and it averages the prediction of each model that is provided into the voting classifier. The final goal of the voting classifier is to create a single model that learns from multiple model and make a prediction based on the aggregate majority of vote for each individual model.

2 voting classifier can be distinguished:

- Hard voting: the output prediction is the majority of vote between the output of each individual model. For example, from 3 models, let say that 2 classifier say that one image is a natural image and the third one says that the image is generated, the output class will be natural image.
- Soft voting: the average probabilities of the classes determine which one will be the final prediction. For example, from 3 models, let say the probability of being a natural image are [0.88, 0.91, 0.87] and the probability of being a generated image are [0.12, 0.15, 0.07], the average of being a natural image is 0.8867 and the average of being a generated image is 0.1133. We can then say that the class of the soft voting classifier will be natural image.

The *Hybrid Xception Ensemble* from paper [5] in Section 1.6 is presented with the hard voting classifier so this is what has been implemented in the proposed framework.

2.7. Implementation details

2.7.1. Implementing the RASH method

The paper in Section 1.2 [2] has been written in 2005 by Lefèvre et al. As discussed in the corresponding section, it conceptually describe how the *RASH* method is working. As a reminder, it computes a 320 bit-string from an image which is unique for each image up to a benign transformation (rotation, cropping, slight color change, ...).

Although this is an old method, since it has never been implemented, it is still relevant to try to implement it in the proposed framework. Moreover, it perfectly matches the original needs of this Master Thesis.

[31] https://www.geeksforgeeks.org/machine-learning/voting-classifier/

To compare 2 RASH strings, a correlation computation has also been implemented.
As explained in the paper, the 320 bits-string is actually a 40*8 bits string. To compare the 2 RASH strings, they are read in parallel, 8 bits by 8 bits. As each 8 bits are more significant than the next 8 bits, a decreasing weight to each 8 bits segment of the strings is given. The first 8 bits have a weight of 40 while the last 8 bits have a weight of 1. Each 8 bits is next converted into an integer, and the absolute value of the difference between the 2 corresponding 8 bits string is taken. Then a normalized sum of the computed difference is compted, and then converted into a percentage.

1	...	40
11011101	...	00000011
11011100	...	00000010

In this table, the difference between the two first 8 bits string (column 1) is 1. Since this is the most significant part of the 320 bit string, 40*1 are added to the sum of the difference between the two RASH strings. Conversely, the difference between the two last 8 bits string (column 40) is also 1. Since this is the least significant part of the RASH strings, 1*1 is added to the sum of the difference between the two RASH strings.

As a reminder, if the correlation between 2 RASH string is above 87%, the images are considered to be equivalent, while in the opposite case, the method considers that an attack has been performed on the image.

2.7.2. Embedding an invisible watermark

A side note has to be made concerning the DWT-DCT-SVD method. In the paper in Section 1.4, the example schema described in the Related Work chapter shows that the watermark is an image. In the proposed framework case, the watermark to embed is a string (the RASH signature). The issue is solved in the source code of the `blind_watermark` Python library:

```
1  if mode == 'img':
2      wm = cv2.imread(filename=wm_content, flags=cv2.IMREAD_GRAYSCALE)
3      assert wm is not None, 'file "{filename}" not
       read'.format(filename=wm_content)
4
5      self.wm_bit = wm.flatten() > 128
6  elif mode == 'str':
7      byte = bin(int(wm_content.encode('utf-8').hex(), base=16))[2:]
8      self.wm_bit = (np.array(list(byte)) == '1')
```

No matter is the watermark is an image or a string, both are converted in the same format in the used method. The explanation of the paper [4] then still holds for the RASH string case.

2.7.3. *Shi-Tomasi* Feature Points Detection

The Shi-Tomasi Feature Points Detection is a method that detect corners in an image. A corner is a pixel for which, in a small surrounding region, intensity changes cannot be explained by a

single dominant direction (as in an edge), but instead exhibit strong variation along multiple directions. In particular, if the change of intensity have:

- no variation : flat region
- variation in one direction : edge
- variation in two or more direction : corner (what we want to find)

In this Master Thesis, the `cv.goodFeaturesToTrack()` method is used, which is the Shi-Tomasi Feature Points Detection method that is provided by OpenCV with a Python implementation. This OpenCV methods is really easy to use and almost instantaneous. Additionally, we can fine-tune some really interesting parameters making the given feature points more significant.

In the proposed framework, those parameters have been used:

- `maxCorners`: 75
- `qualityLevel`: 0.02
- `minDistance`: 15

This parameters trio has given satisfactory result during the performed experiments.

The `maxCorners` parameter indicate the maximal number of corners we want the method to return. The `qualityLevel` indicates the minimum threshold needed for a point to be considered as a corner. Finally, the `minDistance` parameter indicates the minimum possible Euclidean distance between the returned corners. This is relevant since the given points will be balanced all around the image and not concentrated in a single region if a lot of texture is present in that region.

A good combination of these 3 parameters returns the most significant points from all around the image.

The Shi-Tomasi Feature Points Detection method is an evolution of the Harris Feature Points Detection method and a comparative will be provided in the next section.

2.7.3.1. Comparaison of Shi-Tomasi vs. Harris

As said briefly in the previous section, the Shi-Tomasi Feature Points Detection methods comes after the Harris one. Both methods follow the same principle, but they have a small but significant difference: the Harris method's scoring function, which determines the state of a pixel (corner, flat region, ..) is:

$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2 \tag{22}$$

and in the Shi-Tomasi, the scoring function is the following:

$$R = \min(\lambda_1, \lambda_2) \tag{23}$$

This means that if R is a greater than a threshold value, it is considered as a corner.

This only change makes the 2 methods very similar. However, this small change shows significants performance improvement in the corner detection. Here after, 2 comparison example of the 2 methods are shown[32]:

[32]Source: https://medium.com/pixel-wise/detect-those-corners-aba0f034078b

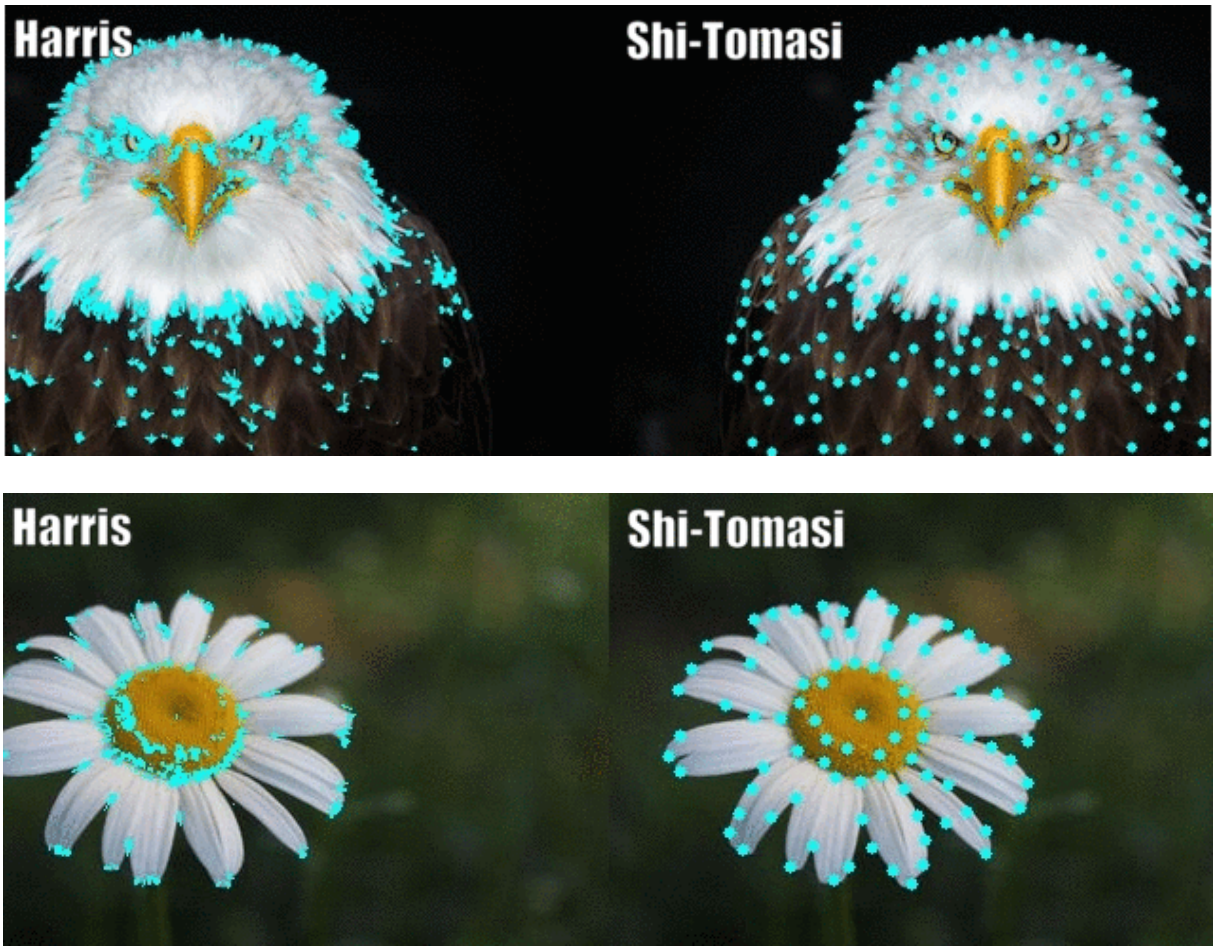


Figure 12: Harris vs. Shi-Tomasi feature points detection comparison

From those example, one can state that the Shi-Tomasi feature points detection method is more robust and accurate. Moreover, it is more customizable in the parameters it takes in the Python method, and this methods is also easier to use.

In term of computation time, no significant difference is notable between the 2 methods. The gain of the Shi-Tomasi method is only notable efficiency and robustness wise.

2.7.4. Implementing the classifier

Since for this classification task, the features of the images on which the model has been trained were of course not available (the primary material was simply free images downloaded from internet), the model has therefore had to learn the natural and synthetic features that characterize the images.

The `img2vec` library from PyTorch[33] implemented in Python has been used in order to extract the features from the images.

[33]https://pypi.org/project/img2vec-pytorch/

The following pseudo-code explain how accuracy score of the model explained in Section 2.3 has been obtained.

```
1 extract features of all images
2 fit the model with training images
3 make prediction on validation images
4 compare with true class of validation images
5 obtain the accuracy score of the model
```

In the context of the proposed framework, similar principle applies for an unseen image.

```
1 extract features of the image
2 make prediction for the class of that image
3 return predicted class
```

For an unseen image, we have no clue to state if the prediction is correct or not since we don't know in advance the true class for that unseen image. We therefore have to rely on the training of the model, and hope that it has been trained as best as possible to obtain the best prediction possible.

2.8. Summary

To wrap up this chapter, a summary of the proposed framework is performed.

The proposed framework is composed of 2 distinct parts.
The first part is designed to protect an image. Step-by-step, it

- computes the RASH of the given image
- embed the computed RASH in the EXIF metadata, in the `UserComment` field
- embed the computed RASH as a blind watermark, with the corresponding `blind_watermark` Python library

The second part is the detection one. It is composed of 3 distinct layers that each performs a verification in order to accumulate points to compute the **authenticity score**, which goes from 0 to 3. First, a fresh RASH from a given image is computed. Then, the 3 layers are:

- Look for RASH signature in the EXIF metadata, in the `UserComment` section, then compute the correlation between the found signature (if so) and the freshly computed RASH
- Look for RASH signature as a watermark with the Python `blind_watermark` library, then compute the correlation between the found signature (if so) and the freshly computed RASH
- Perform a classification task with the hybrid Xception ensemble model. If prediction gives that provided image is a real image, additional point is added to the authenticity score.

Finally, it returns the authenticity score to the user.

3. EXPERIMENTS AND RESULTS

3.1. Introduction

This chapter will present multiple tests that has been done to ensure the robustness of the proposed method. An important part of this section will also be dedicated to the interpretation that can be made with those results.

3.2. Tests performed

3.2.1. Blind Watermark robustness

Before using the `blind_watermark` Python library[34] as the watermark embedding and detection for our proposed framework, a test has to be performed to ensure that this watermark method is robust.

Type of Watermark	Number of image	True detection rate
Filename	6000	99,6%

Table 3: accuracy of the watermark detection

The followed test has been performed:
Given 6000 images, the filename of the image have been embedded as the watermark. The experience has been done with the filename and not directly with the RASH because computing the RASH is a computational heavy task. Instead, the filename takes no time to compute.

For each image of that dataset where the watermark has been embedded, the goal of the test was to extract the watermark from that image and to compare it with the true filename of the image. The result were good, as **99.6%** of the watermarks were correctly matching the true filename of the corresponding images.

According to the official Python documentation of this library[35], the watermarking scheme should be robust against non-malicious transformations, such as

- resizing
- horizontal/vertical cut
- rotation
- cropping

Further experiments to validate those results have been considered in the following table.

[34] https://pypi.org/project/blind-watermark/
[35] https://github.com/guofei9987/blind_watermark/tree/master

Attack	Number of image	True detection rate
blurring 1/10 + JPEG compression 85%	100	100%
JPEG compression 70%	100	100%
JPEG compression 50%	100	93%
Random crop	50	100%
Horizontal cut	50	96%

Table 4: watermark robustness to experimented attacks

A benign transformation that has not been taken into account in that list is the blurring. An experiment has then been made to ensure that the watermarking is resistent to blurring.

Given 100 images, a watermark has been put in the pixels of those images. Then a blurring has been applied on those 100 images. If the blurring is weak (with a radius of 1, indicating the level of blurring), the watermark can be extracted successfully in 100% of the cases. Moreover, the watermarked image were in a PNG format. After the blurring they have been converted in JPG format, with a quality level of 85.

Attack	Number of image	True detection rate
blurring 3/10	100	0%
Rotation 45°	100	0%
Resizing 95%	100	0%

Table 5: watermark robustness to performed attacks

Unfortunately, as shown in that last experiment, the watermark is not robust against all the attacks it should be robust, as stated in the official Python documentation page of the library[36]. However, the strength of the proposed framework stands in the redundancy of the securing information in the 3 layers. Indeed, the next experiments shows that the other layers are robust against those attacks.

3.2.2. EXIF metadata robustness

First of all, first experiments performed with EXIF metadata show that EXIF metadata can only be embedded with JPEG images. If images are in PNG format, a conversion has to be made to JPEG format first.

The robustness of the metadata embedding/extraction method in the proposed framework will be presented here.

Type of metadata	Number of image	True detection rate
Filename	100	96%

Table 6: metadata robustness experiment

This first experiment shows that in 96% of the case, the message that has been put in the metadata is successfully retrieved.

[36]https://pypi.org/project/blind-watermark/

The focus will now be put on showing that the message in the metadata can also be retrieved in the presence of an attack.

Attack	Number of image	True detection rate
blurring 1/10	100	96%
Rotation 90°	100	96%
Resizing 70%	100	96%
JPEG compression 70	100	96%

Table 7: metadata robustness to performed attacks

Experiments show that with Pillow[37], the metadata are discarded when saving the image unless explicitly mentioned. Considering that, after applying the attacks and explicitly rewrite the metadata from the original image, experiments show that metadata are still successfully retrieved afterward.

3.2.3. Complete vs. Resized image RASH correlation

Attack	Number of images	Overall correlation
Resizing - 50%	50	82.7%

Table 8: correlation between original image's RASH and resized image's RASH

In order to reduce the computation time, the resizing of the image has been considered. In the first experiment, images has been resized to 50%. The mean of the RASH correlation over the 50 original images and the same 50 resized images is 82.7%. If we refer to [2], the paper that describes the RASH method, the threshold of 87% is not met. This is not sufficient to be able to use resized images instead of the original images. To compute 100 RASH (50 original + 50 with resized images), the experiment has taken nearly 11 hours of run.

3.2.4. Classifier: image type choice

As a reminder, the machine learning models behind the used classifier is the hybrid Xception ensemble model. It is a multi model classifier composed of a Random Forest classifier, a KNN classifier, a SVC classifier with a linear kernel and a SVC classifier with a RBF kernel classifier. In order to obtain the final class prediction for a given image, the discrimination is done by a hard majority vote, for which the resulting class for the image is simply the class that has obtained the more vote from the 4 previous classifiers.

For the training of the model, multiple experiments have been considered in order to train the most relevant classifier possible. Among these experiments, the training of the classifier has been done first with original images. Then the possibility to train the model on original images on which a DWT has been applied has been evaluated. From the DWT, the classifier has been trained with the LL subbands and with the LH subbands. We thought it was a good idea to explore the Discrete Wavelet Transform (DWT) path since this transformation technique reveals how the signal energy is distributed across the resolutions (i.e., the subbands). Also since we would like to try this path for performance reasons since DWT is a compression technique and its subands are lighter than original image. Therefore this technique has been

[37] https://pypi.org/project/pillow/

used to see if a significant gain in training time could be obtained. The following table shows the difference in time and in accuracy between the 3 resolution of the images.

Technique	Accuracy	Training time
Original image	0.8879	5h28
DWT - LL subband	0.8659	5h25
DWT - LH subband	0.819	1h23

Table 9: Model training comparison with multiple image types

From this table, we can see that there is no difference in time between training the model on the original images or the LL subband of the DWT for original images. However, we can see that training the classifier on those LL subbands results in a less accurate classification. From those metrics, we can put it aside. On the other hand, we can see with the LH subband that the training of the model is way faster (by a factor $\approx$ x3.5) but we can also clearly see that this way of training the model gives us a less accurate classification. For those reasons, the decision of training the model with non-transformed images has been made.

3.2.5. Classification with watermarked image

For the classification layer, the model has been trained on non-watermarked images as the dataset has been downloaded from the internet. In that context, special tests had to be performed to ensure that the prediction class for an unseen image was unchanged whether a watermark was present or not in the image. Since the used watermark is invisible, the prediction is in fact well unchanged before or after a watermark has been included in the image, which is a desired result.

Type of image	Number of images	Prediction accuracy
Real	100	93%
Real + Watermark	100	95%

Table 10: prediction with and without watermark applied

For the needs of this experiments, 100 images have been taken randomly. Both classification have been done with the same set of 100 images, with the difference that in the second prediction, an invisible watermark has been put in the images pixels.

The classifier, which is the layer 3 has also been tested on attacked images.

Attack	Number of images	Prediction accuracy
Horizontal cut	100	98%
Resize - 95%	100	96%
Resize - 50%	100	96%
Random crop	100	98%
JPEG compression - 70%	100	95%
Resize - 50%	100	94%
Blur - 1/10	100	94%
Rotation - half 15°, half 45°	200	95%

Table 11: prediction with and without watermark applied

3.2.6. Showing the robustness of the RASH method

This subsection is dedicated to showing the robustness of the RASH method. As a reminder, the desired behavior for that signature is to be robust again benign transformation (rotation, slight cropping, compression, ...) and to be different if malicious transform has been applied to the original image.

The second reminder is that the RASH is not computed over the all the pixels of the image. Instead, the feature points are extracted and the convex hull is drawn thanks to those feature points. The RASH is then computed over the pixels inside the convex hull.

For the following tests, 2 images will be shown: the left one is the original image and the right one is the modified image.

The used image has been taken from Pexels, which is an open source image stock[38]

[38]https://www.pexels.com/photo/classical-vintage-fiat-car-parked-on-grass-near-street-25349389/

Same image


Correlation: 100%

As expected, the correlation between those 2 images is 100% as the 2 images are exactly the same.

Different images


Correlation: 37.2%

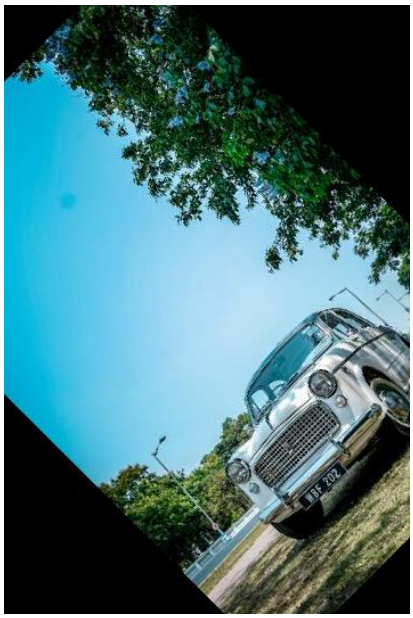
This result is also expected since the 2 images are completely different.

Rotation


Correlation: 98.9%

Since the RASH method is robust to rotation due to the way it works (described in Section 1.2), this high correlation is an expected result.


Correlation: 75.6%

Correlation: 67.9%

Correlation: 72.7%

More surprising and unwanted results were concerning the rotation since that experiment show that the RASH signature computation is not robust to rotation, except the 180 degree rotation.

Cropping


Correlation: 93.1%

The second image has been cropped on the right and left side. Thanks to the feature points extraction, the RASH computation is not impacted that much by this modification since there was no important points on the cropped part.

Slight color change

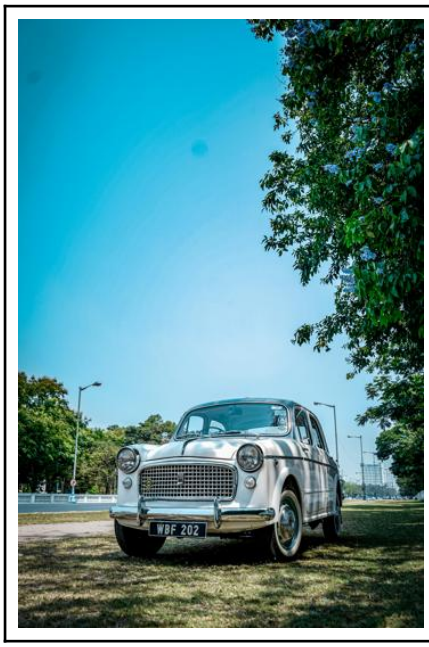
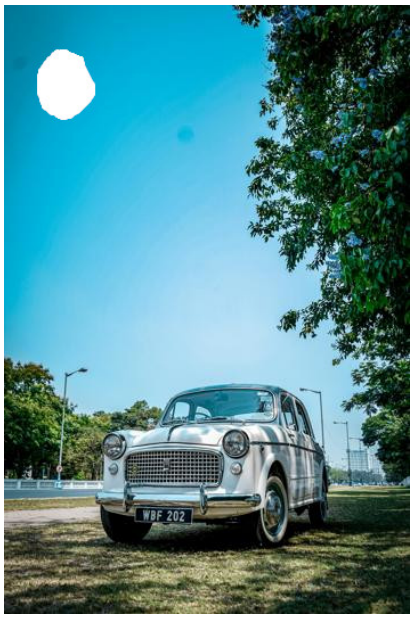

Correlation: 98.4%

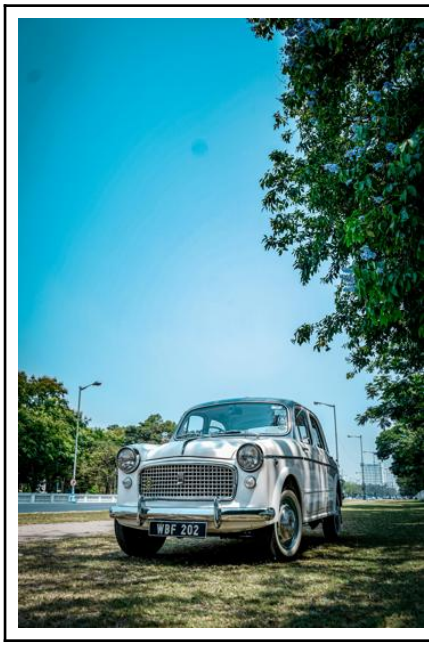
Correlation: 99.1%

The 2 previous results are also expected since the image are the same, there is just change in the saturation of the colors.

Removing an element

Correlation: 97.2%




Correlation: 75.5%

This experiment is interesting as it shows that removing a little part in the side of the image (no feature points were present there) does not impact that much the RASH computation. However, removing a central part of the image make the correlation between the 2 RASH dropping significantly, which is a desired result.

Adding an element


Correlation: 68.4%

Finally, adding some objects on the image drop significantly the RASH too, which is once again a desired result.

3.3. Demonstration of the end-product

To illustrate how the full project is working, this subsection is dedicated to showing the output of the 2 proposed methods.

3.3.1. secure.py

First, given a new image, this image can be secured by adding the RASH in the metadata and in the watermark respectively. To achieve that, the following command line can be used in order to launch the first part of the proposed framework.

```
1  python3 secure.py {image_name}
```

- `python3` is to call the python compiler
- `secure.py` is the name of the file which hosts the securitization code
- `image_name` is the filename of the image you want to authenticate, to be replaced by the user with actual filename

During the execution, the user has the possibility to see the step that is currently performed. The following image shows the result at the end of the execution, showing also all the intermediate steps:

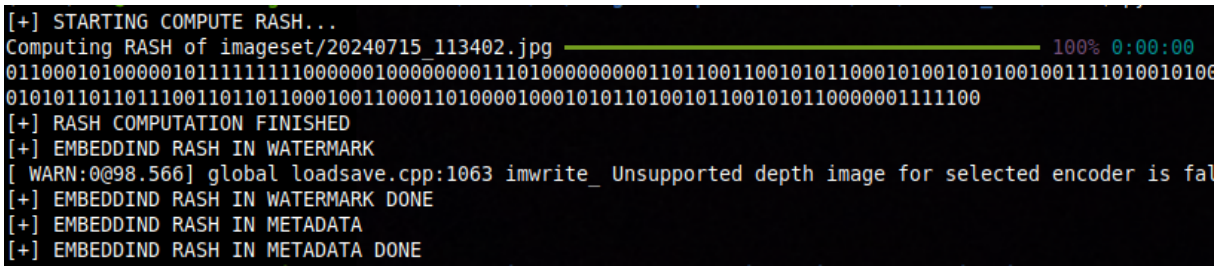


Figure 13: Execution of the secure.py program

During the computation of the RASH, user is presented with a progress bar showing estimated remaining computation time as well as the percentage of the execution that has already been performed.

Each step (RASH computation, metadata embedding, watermark embedding) is presented to the user with the moment it starts:

```
1 [+] EMEBDDING ...
```

and the moment it has finished

```
1 [+] ... DONE
```

3.3.2. main.py

The main part of this Master Thesis is the detection part, and the way it works will be shown here.

In order to launch the program, the same principle than `secure.py` applies. This simple following command is needed to start the detection phase.

```
1 python3 main.py {image_name}
```

Here again, the user is proposed with the live details about the current execution of the code. The following screenshot shows the result at the end of the execution, showing also all the intermediate steps:

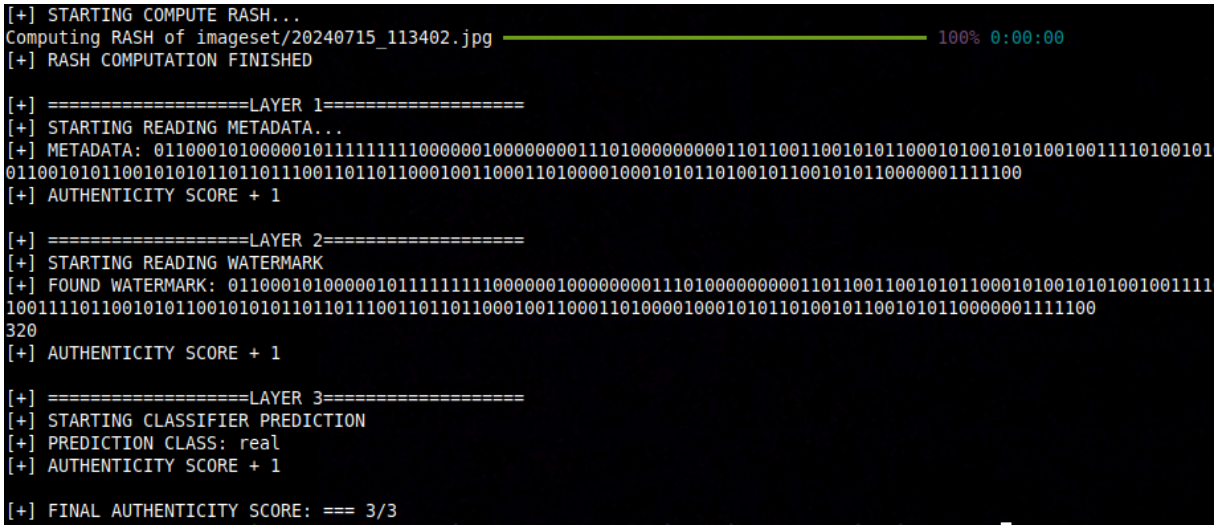


Figure 14: Execution of the main.py program

During the computation of the RASH, user is presented with a progress bar showing estimated remaining computation time as well as the percentage of the execution that has already been performed.

Each step (RASH computation, metadata embedding, watermark embedding, classification task) is presented to the user with the moment it starts:

```
1  [+] STARTING ...
```

It then shows the result of the corresponding phase:
- layer 1: shows if the RASH is found in the metadata + it prints the found RASH
- layer 2: shows if the RASH is found in the watermark + it prints the found RASH
- layer 3: it prints the predicted class

At each step, it also tell whether it adds one point to the total authenticity score or not.

At the end of the execution, user is finally presented with the final authenticity score is has obtained with the provided image.

3.4. Testing with the proposed framework

Due to the time it takes to computes the RASH, as well as the embedding of the watermark, it has not been possible to perform an experiment on thousands of images. However, in order to demonstrates the framework, 20 real images and 20 synthetic images have been randomly selected from multiple datasets. The following table will show the result of the experiments (authenticity score) as well as the pre-treatment they have received. For the real images, when all the layers does not show satisfactory results, the layer that is missing is indicated in the “Authenticity score” column, next to the score.

Filename	Dataset	True class	Secured?	Authenticity score
20240715_113402.jpg	Personal dataset	Real	Y	3/3
20240716_150516.jpg	Personal dataset	Real	Y	3/3
20250713_144856.jpg	Personal dataset	Real	Y	3/3
DSC03332.jpg	Personal dataset	Real	Y	2/3 - layer 3
IMG_8105.jpg	Personal dataset	Real	Y	2/3 - layer 3
0025.jpg	ai-generated-images-vs-real-images[39]	Real	Y	3/3
0037.jpg	//	Real	Y	3/3
0039.jpg	//	Real	Y	3/3
0071.jpg	//	Real	Y	3/3
0270.jpg	//	Real	Y	2/3 - layer 2
0015.jpg	//	Fake	N	0/3
0079.jpg	//	Fake	N	0/3
0118.jpg	//	Fake	N	0/3
0153.jpg	//	Fake	N	0/3
0199.jpg	//	Fake	N	0/3
animals_107.jpg	AI vs Real Images Dataset[40]	Real	Y	2/3 - layer 2
city_116.jpg	//	Real	Y	2/3 - layer 2
food_117.jpg	//	Real	Y	3/3
people_107.jpg	//	Real	Y	3/3
nature_12.jpg	//	Real	Y	2/3 - layer 2
fake_89.png	10000 Real vs Fake Faces (StyleGAN3)[41]	Fake	N	0/3
fake_7444.png	//	Fake	N	0/3
fake_6.png	//	Fake	N	0/3
fake_61.png	//	Fake	N	0/3
fake_29.png	//	Fake	N	0/3

[39] https://www.kaggle.com/datasets/tristanzhang32/ai-generated-images-vs-real-images
[40] https://www.kaggle.com/datasets/rhythmghai/ai-vs-real-images-dataset?resource=download
[41] https://www.kaggle.com/datasets/troykueh/real-vs-fake-faces-stylegan3?select=Real+faces

Filename	Dataset	Class	Has been secured	Authenticity score
image-2.png	AI vs Real Images Dataset	Fake	N	1/3
image-14.png	//	Fake	N	0/3
image-24.png	//	Fake	N	0/3
image-26.png	//	Fake	N	0/3
image-33.png	//	Fake	N	0/3
animals-100.jpg	//	Real	Y	2/3 - layer 2
animals-105.jpg	//	Real	Y	1/3 - layer 2,3
animals-109.jpg	//	Real	Y	2/3 - layer 2
animals-111.jpg	//	Real	Y	2/3 - layer 2
animals-115.jpg	//	Real	Y	2/3 - layer 2
0_20241130135552.jpg	Detect AI-Generated Faces: High-Quality Dataset[42]	Fake	N	0/3
102_20241130141854.jpg	//	Fake	N	0/3
115_20241130142148.jpg	//	Fake	N	0/3
116_20241130142214.jpg	//	Fake	N	0/3
123_20241130142417.jpg	//	Fake	N	0/3

Table 13: Execution result of the complete proposed framework

Interpretation: From this experiment, multiple conclusions can be drawn. First, experiment shown that layer 1, which is the metadata is highly robust since when the RASH is put in it, it has always be retrieved. Second, because the RASH is also embedded as an invisible watermark within the image pixels of secured images, and because the correlation between the newly computed RASH and the RASH extracted from the metadata has been validated at 100%, it can be concluded that the watermark embedding process does not alter the RASH signature computation. Third, experiment also show that the layer 2 is a bit fragile for some datasets. This is why it is important to have layer 1 and layer 3 in backup to still be able to obtain a good authenticity score. Finally, this experiment has also permit to see that in the case of fake image, where no securing process has been applied, the authenticity score is almost always of 0/3 which means that the classifier output a fake prediction for those images.

3.5. Summary

This chapter has shown thanks to the performed experiments (RASH, watermark, classifier) that the desired results have been achieved. It has also shown the complete behavior of the complete proposed framework, with the corresponding output for each of the 2 programs.

[42]https://www.kaggle.com/datasets/shahzaibshazoo/detect-ai-generated-faces-high-quality-dataset

4. LIMITATIONS

The main limitation of the proposed framework in this Master Thesis is related to the evolution of Artificial Intelligence. During this project, a tool that seems resilient to AI today has been built, but who knows what AI will be capable of doing tomorrow ? We all know that the generated content is more and more realistic and it is more and more difficult to detect it.
In the previous section about the datasets used, it has been shown that a dataset of AI generated images is not the other and the models generating the images are different, such that a classifier is not able to detect AI with high precision generated images from another dataset that are not generated with the same model.
In the part concerning the metadata, it has been shown that it is possible to include handcrafted metadata in an image, and it is also true for a generated image, even if generated image are not intended to have metadata because it is nonsense to have aperture, ISO value, shutter speed and all the content related to camera model and settings since this image is generated by a computer. But since it is possible to add those value for any image (including generated image), the mere fact that metadata are present in a image is not sufficient to prove authenticity of this image.

Since none of the 3 layers it sufficient on its own to prove entire authenticity of an image, 3 distinct layers have been developed in a single tool. The reason of building 3 such layers is to be redundant with the computed RASH signature. For example, since it has been shown that data can be overwritten, we can not only rely on this layer to state that an image is synthetic if the signature is not present on the metadata. This is why the RASH signature can also be putted in the image pixels directly, as an invisible watermark. Moreover, if the securing part of the process has not been applied, the proposed framework still offers the possibility to give the user a hint about whether the image is real or synthetic by performing a classification task.

Another major limitation of the proposed framework is the time taken to compute a RASH signature, depending on the image resolution. In the original implementation of the algorithm, presented in [2], the program makes multiple pass on each pixel of the given image. Depending on the image resolution, passing on every pixel of the image can be a heavy task. In consequence, it has not been possible to test the final proposed framework on thousands of images.

A last noticed limitation during the realization of the proposed framework is that metadata are deleted when images are sent via social media such as WhatsApp, Messenger or Discord for privacy reasons, but they are kept when images are sent via email. This is important to keep that aspect in mind when manipulating and sending images in order to compute their authenticity score.

5. NOTES ABOUT USE OF AI

During the realization of this Master Thesis, AI has been used in a constructive way. It means that it has only been used to get more precise details and explanation about the already existing used method. No code used in the proposed framework has been generated by AI. None, or very few, of the AI was used to reformulate this master's report either. As a work about the proof of authenticity in the context of images, I also wanted to let this report authentic.

6. CONCLUSION AND PERSPECTIVES

To conclude, a tool with the goal of proving the authenticity of an image has been proposed. In consists in three different layers:

- Layer 1 is dealing with metadata
- Layer 2 is dealing with blind watermark
- Layer 3 performs a classification task to distinguish real images from synthetic ones

The results obtained from the different experiments done were mostly good, despite a few unexpected results, described in each corresponding section.

As shown in the previous chapter, the proposed framework built for this Master Thesis is not perfect, and depending on the evolution of AI models, will never be. However, under the current circumstances, the proposed framework could receive some improvements.

Among those improvements, an improvement concerning the layer 3, the classification task, can be envisaged. To improve the overall framework, layers could be ponderated according to the given source image. Given the fact that classifier is not performing very well with image from a new dataset, the framework could analyze the content of the dataset during training such that if a new image is given to the framework, if the subject of that image is new to the framework, the ponderation of the classification task could be lowered. To be more explicit, a very simplified example could be the following: if the model has been trained only on cats and dogs image, if the framework is provided with an image of a house, the classification task could not perform well. As a consequence, the ponderation of the classifier could be lowered such that the trust given to that layer is limited.

Another issue with the classifier is that it is trained once, then it produces a model and only that model is used to states whether an image is real or synthetic. An evolution that can be bring to the model to face this issue could be to somehow feed the model with new data such can it can be updated and learn new AI tendance.

A final idealistic perspective concerning Layer 3 could be the following. Since the classifier is not performing well on unseen images from another dataset, a new handcrafted dataset, the biggest possible that that brings together a maximum of diversity among the various datasets, could be created. It would help by training the classifier on the most diverse images possible.

As already stated earlier, the RASH signature computation is time expensive. An optimization to the computation algorithm could help the framework by improving the time of the computation of the signature.

Another feature that could be explored is the following: it seems that some people found an invisible watermark that Google put in every GEMINI generated image. The project can be found on the following link https://github.com/aloshdenny/reverse-SynthID. However, a reserve has to be made concerning this method. First, it covers only the GEMINI generated images. Second, we have to ensure that the Gemini watermark is not destroyed when the invisible watermark of the proposed framework is inserted in the pixels of the image.

7. ACKNOWLEDGEMENTS

I would like to pronounce my gratitude to Professor Benoit Macq and Rony Darazi for their help, suggestion, motivation and constructive feedback all year long but also to have proposed this Master Thesis subject. It has been a subject on which I have appreciated to work on but also knowing that the subject, which is the detection of the generated content as well as content modified in a malicious fashion, is a major concern in the actual society, has kept my motivation up to propose an useful tool for the society.

I would also like to express my gratitude to Professor François-Xavier Standaert, the reader and jury member of this Master Thesis, for the time and expertise he has given at evaluating this thesis.

I would also express my gratitude to Emile Van Gysel for the time he has spent reading this thesis and helping me solving misspellings .

A big thank you also to my parents for their unwavering support, motivation and investment during this academic journey and more broadly since the very first day of my life. Thanks for giving me the opportunity of doing such studies. Their belief in my abilities, support through challenging times and the values they have given me has helped me to give the better version of myself.

Finally, I also want like to thanks all friends I met along the way at UCLouvain during those years, for their technical help but even more so for their human qualities.

Bibliography

[1] S. Taborosi, A. Kovačević, and B. Maljugić, "THE ROLE OF SOCIAL MEDIA IN THE DECISION-MAKING PROCESS," 2022, p. .

[2] C. De Roover, C. De Vleeschouwer, F. Lefebvre, and B. Macq, "Robust video hashing based on radial projections of key frames," *IEEE Transactions on Signal Processing*, vol. 53, no. 10, pp. 4020–4037, 2005, doi: 10.1109/TSP.2005.855414.

[3] P. Bas, J.-M. Chassery, and B. Macq, "Geometrically invariant watermarking using feature points," *IEEE Transactions on Image Processing*, vol. 11, no. 9, pp. 1014–1028, 2002, doi: 10.1109/TIP.2002.801587.

[4] M. M. Rahman, "A dwt, dct and svd based watermarking technique to protect the image piracy," *CoRR*, 2013, [Online]. Available: http://arxiv.org/abs/1307.3294

[5] S. C. S and S. R, "Performance Comparison of Deep Learning Models for Computer Generated Image Detection," in *2023 International Conference on Control, Communication and Computing (ICCC)*, 2023, pp. 1–5. doi: 10.1109/ICCC57789.2023.10164874.

[6] M. Deng, F. Yang, Q. Chen, J. Wang, S. Sun, and B. Qi, "A multi-scale refinement corner detection algorithm based on Shi-Harris," *Digital Signal Processing*, vol. 161, p. 105137, 2025, doi: https://doi.org/10.1016/j.dsp.2025.105137.

[7] R. Roy *et al.*, "A Comprehensive Dataset for Human vs. AI Generated Image Detection." [Online]. Available: https://arxiv.org/abs/2601.00553

[8] P. Neekhara, S. Hussain, X. Zhang, K. Huang, J. McAuley, and F. Koushanfar, "FaceSigns: Semi-Fragile Neural Watermarks for Media Authentication and Countering Deepfakes." [Online]. Available: https://arxiv.org/abs/2204.01960

[9] Z. M. Hazza and N. A. Aziz, "Detecting Computer Generated Images for Image Spam Filtering," in *2012 International Conference on Advanced Computer Science Applications and Technologies (ACSAT)*, 2012, pp. 313–317. doi: 10.1109/ACSAT.2012.38.

[10] C. De Roover, C. De Vleeschouwer, F. Lefebvre, and B. Macq, "Robust image hashing based on radial variance of pixels," in *IEEE International Conference on Image Processing 2005*, 2005, p. III–77. doi: 10.1109/ICIP.2005.1530332.

[11] A. Shehab *et al.*, "Secure and Robust Fragile Watermarking Scheme for Medical Images," *IEEE Access*, vol. 6, no. , pp. 10269–10278, 2018, doi: 10.1109/ACCESS.2018.2799240.

[12] W. H. Alshoura, Z. Zainol, J. S. Teh, and M. Alawida, "An FPP-resistant SVD-based image watermarking scheme based on chaotic control," *Alexandria Engineering Journal*, vol. 61, no. 7, pp. 5713–5734, 2022, doi: https://doi.org/10.1016/j.aej.2021.10.052.

[13] R. Sinhal, D. K. Jain, and I. A. Ansari, "Machine learning based blind color image watermarking scheme for copyright protection," *Pattern Recognition Letters*, vol. 145, pp. 171–177, 2021, doi: https://doi.org/10.1016/j.patrec.2021.02.011.

[14] P. Milosav, Z. Banjac, T. Unkasevic, and M. Mostafa, "Overview and Classification of Digital Watermarking Algorithms," 2019, pp. 537–545. doi: 10.15308/Sinteza-2019-537-545.

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**