Boost algorithm

Nicola Arpino, Gabriele Motta, Michele Panariello

Contents

1 Introduction

1.1 Goal of the project

The problem that our project aims to solve is the blind source and tampering identification of images taken by a specific device.
In order to accomplish this requirement, we will need an unique mark for each device and embed this mark into the images taken by that device.

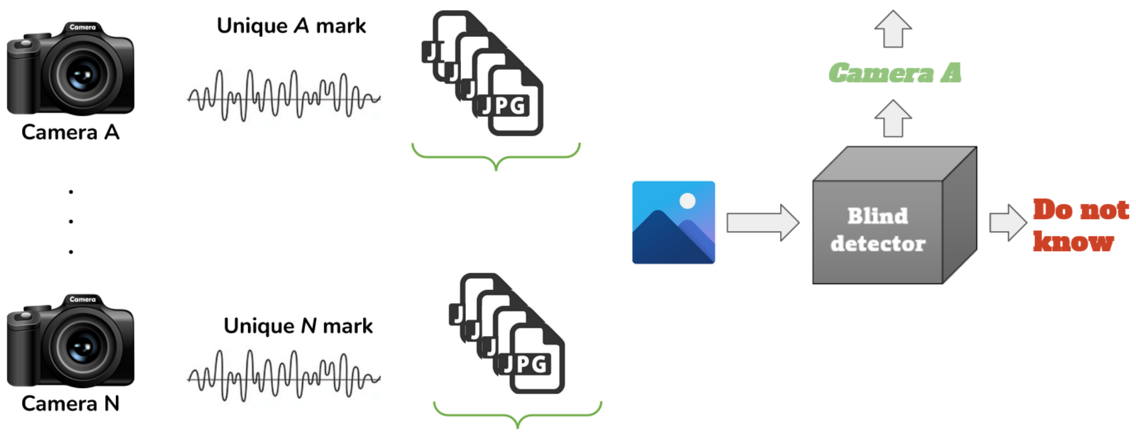


Figure 1: Problem schema

From the problem statement we have identified two major points.

1) Marks must be unique and different from each other.

2) Detector must be blind: only the watermarked image is provided

1.2 Project settings

The setting for this project are: `5 devices with 20 images each`.
Of these 100 images only a few will be embedded and/or attacked. This will allow us to collect all the necessary data for our analysis such as true positive rate (TPR), false positive rate (FPR) and accuracy. We will further explain this fact in the results 4.1

$$TPR = \frac{TP}{TP+FP},$$
$$FPR = \frac{FP}{(FP+TN)},$$
$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}.$$

2 Fair Mark Generation

2.1 Formal Description

In this section we describe in details the method that we used to generate the marks and the purpose of it.

Defintion 1. Let n be an integer. We call mark of length n an elements of the set $\{0, 1\}^n$

Therefore, marks are binary strings. Roughly, our objective is:

1. Embed one of the five marks corresponding to the five devices,
2. Possibly attack the image,
3. Extract the mark,
4. Identify with which of the five cameras the photo was taken, this is done by comparing the extracted mark with the five marks.

The comparison we do in the step 4. is based on how many matching bits we have between the two marks. Let us define formally

Defintion 2. Let m and $\overline{m}$ be two marks of length n. We say that there is a match in the position i-th if $m_i = \overline{m}_i$.

To compare the marks in their full length we use the accuracy metric

Defintion 3. Let m and $\overline{m}$ be two marks of length n. Then the accuracy between m and $\overline{m}$ is the number of matches over n

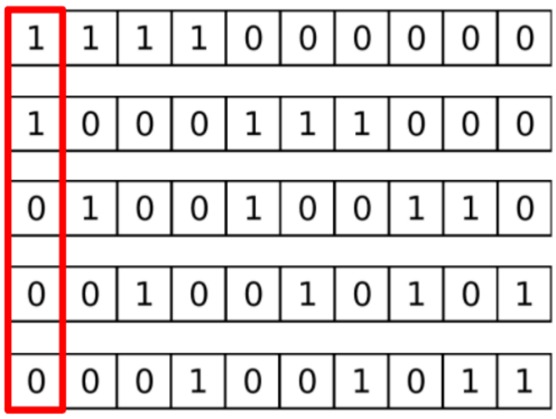
$$Acc(m, \overline{m}) = \frac{\sum_{i=1}^{n} 1}{n} \tag{1}$$

In order to maximize the TPR and minimize the FPR, we want to generate marks that are dissimilar to each other.

Defintion 4. Let M be a set of marks of same length n. Then the maximum accuracy is defined as

$$max(Acc(m, \overline{m}))_{m, \overline{m} \in M} \tag{2}$$

In other words, we would like to find a set of marks with the *least maximum accuracy.* Let us see an example.



1	1	1	1	0	0	0	0	0	0
1	0	0	0	1	1	1	0	0	0
0	1	0	0	1	0	0	1	1	0
0	0	1	0	0	1	0	1	0	1
0	0	0	1	0	0	1	0	1	1

In this example with five marks, taking any pair of marks, the resulting accuracy will always be 0.4.
Moreover, we can see that for each i-th position we have two 1's and three 0's, this pattern was used to create the five marks with the above property.
For example, in the highlighted rectangle we have the mark 1 and the mark 2 set to 1, in the second position the mark 1 and the mark 3 set to 1 and so on. Intuitively this is the best case scenario.
Now, we are ready to give the following definition

Defintion 5. Given a set M of n different marks of same length. We say that M is fair if and only if the accuracy between any pair of marks m_1 and m_2 are all equal.

The following theorem generalizes the result we saw in the previous example with only five marks. Note that if m is odd $\frac{m}{2}$ is not an integer, so throughout the enunciate and the proof we are going to use the rounded down result.

Theorem 1. *Given a set of $m > 2$ marks of length n. The maximum accuracy is at least*

$$\frac{\binom{m-2}{\frac{m}{2}-2} + \binom{m-2}{\frac{m}{2}}}{\binom{m}{\frac{m}{2}}} \tag{3}$$

Proof. Step 1. We want to prove that a fair set has the smallest maximum accuracy. Assume that M is a fair set of marks. Let i be an index less or equal to n. We choose two marks in M, without loss of generality, say $m_1, m_2 \in M$.

Now, let us consider the i-th bit of both marks and consider flipping the i-th bit of one of the two marks, let us call the set with the flipped marks $\overline{M}$. If $m_1(i) \neq m_2(i)$ then the accuracy $Acc(m_1, m_2)$ would be greater if the bits are flipped. On the other hand, if $m_1(i) = m_2(i)$ then if we flip one of them since $m > 2$, there is at least one mark whose accuracy is greater.

In both cases the maximum accuracy of $\overline{M}$ is greater than the maximum accuracy of M. Moreover, any other set of marks is obtained from M by flipping a certain number of bits in the m marks. Since every flip can only increase or return to the maximum accuracy of M, we have proven that a fair set of marks has the smallest maximum accuracy.

Step 2. We want to prove a fair set has maximum accuracy equal to (3). The maximum accuracy of a fair set is exactly given by (3), this is just an application of basic probability theory. Indeed, in every i-th position half of the marks are 1 and the other half is equal to 0. Therefore we have $\binom{m}{\frac{m}{2}}$ possible combinations, that is indeed the denominator of (3), those are all the possible cases. On the other hand, consider a pair of marks, $m_i, m_j \in M$. The favorable cases is the sum of all the cases where $m_i(k) = m_j(k)$ for $k \in [1, ..., n]$. Set $m_i(k) = m_j(k) = 1$, then we have $\frac{m}{2}$ other marks equal to 0 over a total of $m-2$ marks that are not fixed. Hence, the total combinations are

$$\binom{m-2}{\frac{m}{2}-2}. \tag{4}$$

The case $m_i(k) = m_j(k) = 0$ is analogue, in this case the combinations are

$$\binom{m-2}{\frac{m}{2}}. \tag{5}$$

The favorable cases is the sum of (4) and (5). □

2.2 Advantages and Disadvantages

The method has the following advantages

- The true positive rate is truly maximized.
- The mark generated with this method have 40% of 1's. So it is much easier to spot the right embedded mark w.r.t a random mark (with 50% of 1's)
- It is extensible to larger mark length if more information is needed.
- Any fair mark set for any m has maximum accuracy less than 0.5, indeed, the quantity (1) tends to 0.5 for m that goes to $+\infty$.

On the other hand

- Since the minimum amount of bits needed to have a set of fair marks is $\binom{m}{\frac{m}{2}}$ we need a minimum amount of bit space to have fair marks, this number grows very fast with m, hence if the number of devices to identify is very large the method can not be applied in practice (even though a suboptimal version of it might be a good idea).

- For the same reason, in order to get the fairness the length of the marks must be a multiple of $\binom{m}{\frac{m}{2}}$.
- Not extensible if another device is added to the set.

As final remark, if we need larger mark size than $\binom{m}{\frac{m}{2}}$, we can take the generated base fair marks for a fixed m and we concatenate those to get new marks of arbitrary length. Moreover, in the process described above, we can permute the base generated marks that we concatenate in order to have more variety in the marks.

2.3 Random Mark Generation Comparison

Instead of methodically generate the marks, one could think to generate the marks randomly. It has almost exactly the pros and cons reversed, indeed it has

- More scalability.
- Extensibility, since we can randomly generate another mark.
- No bounds on the mark length (no lower bounds and not necessarily a multiple of a number).

but it is less precise and efficient

- True positive rate not maximized.
- Very far from optimality, especially if mark length is small.
- Maximum accuracy between pairs can be a lot larger than 0.5.

3 Boost algorithm

3.1 Embedding

Blind identification means that we have only the watermarked image without any other information.
The embedding strategy chosen will need to use something that can be preserved like the greatest value between two DCT spots.

Our method is divided into four steps:

1) Calculate the optimal size of the chunks via **adaptive approach**.
2) Calculate the portion of the mark that goes into each image layer.
3) Retrieve all the chunks from each layer and select a part of them.
4) Cycle over chunks of each layer and follow the **partitioning** and **embedding rule**.

3.1.1 Adaptive Approach

To improve embedding quality and robustness, the chunk resolution should be related to the image resolution. Given an image with a given resolution and embedding ratio (how invasive the mark should be), the adaptive approach takes care of selecting the best resolution for each chunk. This approach calculates chunk sizes by looking for the best divisor factors of the given image size. A divisor factor is a number such that the division between an image dimention and it produces a remainder of zero.
To calculate the best factors we need to perform three steps:

1) Calculate the total chunks -> `(imageX * imageY) / (randomXFactor * randomYFactor)`.
2) Check if the total chunks are more than the mark size multiplied by the embedding ratio.
3) Check if the distance between randomXFactor and randomYFactor is the smallest between every other pair of random factors that confirmed the first check.

Note that the minimum chunk size required is 8x8.

3.1.2 Calculate the quantity of mark per layer

The algorithm can work both with *greyscale* or *colored* RGB images. Grey images have only one layer, while RGB images three.
If we are in the RGB case we can choose to use one, two or three layer to embed our mark. Consequently the quantity of mark that will be inserted in every layer will be equal to `(markSize / channelsToUse)`

3.1.3 Select the chunks

Once we have the resolution of each chunk we can calculate the number of chunk using the following formula `(imageX * imageY / chunkX * chunkY)`.
After we have calculated the total number of chunks, we can move on to selecting the chunks that will be used to embed the mark. Thanks to the *embedding ratio*, we can calculate an offset between each chunk such that each embedding portions of the mark will be inserted with a certain distance from each other. This allows us to spread the mark throughout the image and not just on a portion of it. The quoted offset is calculated using the following formula: `(totalChunks - (totalChunks mod(markSize)) // markSize`.
Where "//" means get the quotient of the euclidean division and `x mod(y)` means take the remainder from the division of x by y.

3.1.4 Embed each chunk

Now that we have the areas in which to insert our mark we can move on to embedding. In each chunk we are going to embed a **single bit of the mark** using two rules, **partitioning** and **embedding rule** .
The partitioning rule consist into dividing the mark in 3 portions and assign at each part of the mark a pair of DCT coefficients: [(2, 6), (3, 5)], [(5, 3), (6, 2)] and [(0, 7), (7, 0)].
The embedding rule consist into increment the first spot if we want to embed a ’0’, else increment the second

one if we want to embed a ’1’. To such a spot is also added the absolute difference between the two, because we have to be sure that the spot increased will be the greatest. Due to the fact that the increment greatly increases the difference between the two spots, the mark will be very difficult to damage because this would require a specific attack to make the smaller spot become the larger one.
The energy of the increment is defined by the **boost** parameter that can be set as the embedding rate to increase the robustness or invisibility of the mark.

3.2 Detection

The detection system will extract the mark following the same steps of the embedding method:

1) Calculate the optimal size of the chunks via **adaptive approach**.
2) Calculate the portion of the mark that goes into each image layer.
3) Retrieve all the chunks from each layer and select a part of them.
4) Cycle over chunks of each layer and follow the **partitioning** and **extraction rule**.

The only difference is in the last step in which we are going to use the **extraction rule**. The extraction rule consists in checking which is the greatest between the two spots selected by the partitioning rule, if the first spot is the greatest then we extract a zero, if not a one.
After the mark extraction is done, we can perform the main task thanks to the **identify function**.

3.3 Identify

The identify function is the core of the project, this function aims to recognize the device through a double threshold mechanism that goes to choose the best accuracy between the extracted mark and each mark in the database. In order to perform this task the identify function need to perform four steps:

1) Compare the extracted mark pairwise with each other available original mark in the database using **accuracy** as metric.
2) Save the comparison between the highest and the lowest accuracy.
3) Compare those accuracies with the upper-bound and lower-bound thresholds.
4) Choose the accuracy that has the maximum distance from his threshold.

3.3.1 Compare the extracted mark

The first step is to compare the extracted mark with all the other original marks into the database, with this operation we can obtain a vector of accuracies where the accuracy is defined as the number of the equal bits divided the number of the total bits of the mark.

[57.28, 47.96, 49.71, 52.23, 47.18, 47.48]

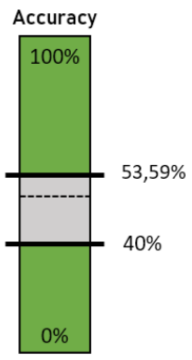
3.3.2 Save the highest and the lowest accuracy

The second step consist into select the highest and the lowest accuracy from the vector and save them for a successive operation.

[57.28, 47.96, 49.71, 52.23, 47.18, 47.48]

3.3.3 Compare the two best accuracies with their relatives thresholds

Now that we have both the highest and the lowest accuracy, we can compare them with our thresholds.



At this point we have three cases:

1) The highest accuracy is higher than the upper-bound threshold **and** the lowest accuracy is lower than the lower-bound threshold.
2) The highest accuracy is higher than the upper-bound threshold **or** the lowest accuracy is lower than the lower-bound threshold.
3) The highest accuracy is lower than the upper-bound threshold **and** the lowest accuracy is higher than the lower-bound threshold.

If we are in the first case we proceed to the fourth step, if we are in the second case we guess the corresponding device, if we are in the third step we cannot say anything about the identification and the algorithm return "no devices found".

3.3.4 Choose the best accuracy

If we have arrived at this step it means that both, the highest and the lowest accuracy have passed the thresholds check. In this case we have to choose the most suitable accuracy by looking at the distance between the accuracies and their relative thresholds. For example:

[57.59, 47.96, 49.71, 52.23, 37.00, 47.48]

In this case we choose the highest accuracy because the distance between the highest accuracy and the upper-bound threshold is 4, while the distance between the lowest accuracy and the lower-bound threshold is 3.

3.3.5 Tampering detection

The identify function can also be used for the tampering detection, infact if one of the accuracies is greater or equal to 97% we can say that the image was not attacked.

Note that this only works if the image is embedded, in fact, having an accuracy greater than 97% without embedding is probabilistically impossible.

4 New metric: SSIM vs WPSNR

In this section we would like to explain the reasons behind the adoption of *SSIM*[1] *(Structural similarity index measure)* in the place of the WPSNR[3].
Above methods are commonly used in order to measure the difference in terms of image quality between two images; the very last one is the one that has been used in the course and that was our first adoption. In this project we had to manage not only ***gray-scale*** images but also ***colored*** ones, typically in RGB format. In this scenario we found out the usage of WPSNR impractical because of the following reasons.

1. RGB images are made up of three layers: how to address the case in which not all the layers contains a piece of additional information? (the watermark)
2. A simple solution might calculate the `average` of the three layers between the the original and the watermarked image. This solution unfortunately does not work. The WPSNR value between two layers that are identical increases too much the average: this leads to high values of the metric that do not represents the true image quality.
3. Even if with some effort we would have been fix the calculation of values in above scenario, the WPSNR implementation we were using has a not negligible cost in terms of time:
   40 seconds to process a single image

SSIM has then been adopted since, for the purpose of this work, is almost identical to WPSNR but provides some practical improvements:

- Default *multi-channel* images support
- Up to ***3x*** speedup
- Does not require any "weights" table to be calculated

4.1 How it works

SSIM is a *perception-based* metric, it exploits some characteristics of the human-visual-system and therefore it is based on 3 features:

1. **Luminance**: it is measured by averaging over all the pixel values.
2. **Contrast**: It is measured by taking the standard deviation (square root of variance) of all the pixel values.
3. **Structure**: The structural comparison is done by using a consolidated formula but in essence, we divide the input signal with its standard deviation so that the result has unit standard deviation which allows for a more robust comparison.

$$\text{SSIM}(\mathbf{x},\mathbf{y}) = [l(\mathbf{x},\mathbf{y})]^{\alpha} \cdot [c(\mathbf{x},\mathbf{y})]^{\beta} \cdot [s(\mathbf{x},\mathbf{y})]^{\gamma}$$

Above formula is a compact way to show how contributions of the three key features are calculated. They represents respectively the luminance, contrast and structure comparison functions. Values range are between 0 and 1: the higher the value, more similar are the compared images. We conducted many tests in order to compare the "old" WPSNR metric and SSIM. The original target was to provide a framework whose embedding process does not degrade the image very much e.g WPSNR >= 60db. The analogous value for the new metric is `0.995` whereas we consider an image degraded with a value $<$ `0.9`.
More information can be found here[2]

5 Results and Conclusions

5.1 Roc curves

ROC has been calculated considering all the images. Each has been attacked 20 times with all the available attacks configured as follows:

- Gaussian Noise: (AWGN) sigma = 300
- Blur: sigma = 6
- Sharpening = sigma = 10; alpha = 10
- Resizing: 0.10
- Jpeg compression: QF = 13

The attacks implementations has been taken from the public available Python libraries `skimage` and `scipy`. Some of them were are different from the ones used in in class since they were not supporting multi-layer images. We put some effort in adapting attacks to work both with gray-scale and colored images.

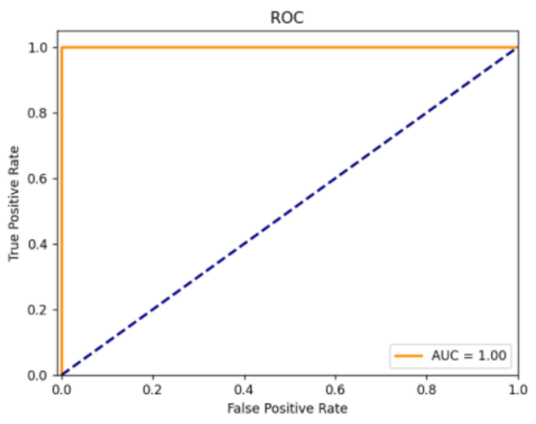


Figure 2: Roc without attacks

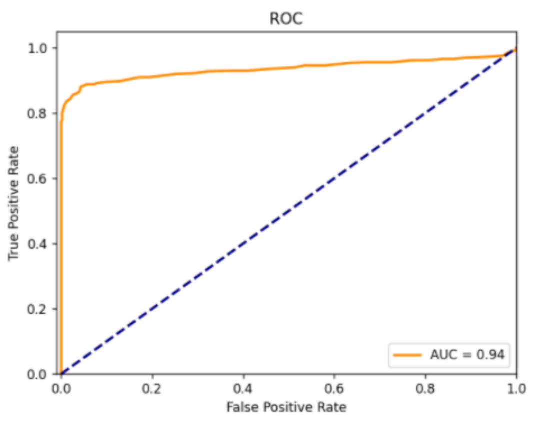


Figure 3: Roc with attacks

As you can see when there are no attacks, the framework classifier is identical to the one of the "ideal" case in which there are no false alarms. In case of attacks the result we got is the following:
`For an FPR = 0.05 corresponds TPR = 0.87`

5.2 Confusion tables

In order to validate our work, in addition to ROC curves, we provide also confusion tables that are meant to describe how the classification framework works. In the above figure are depicted on the rows the actual

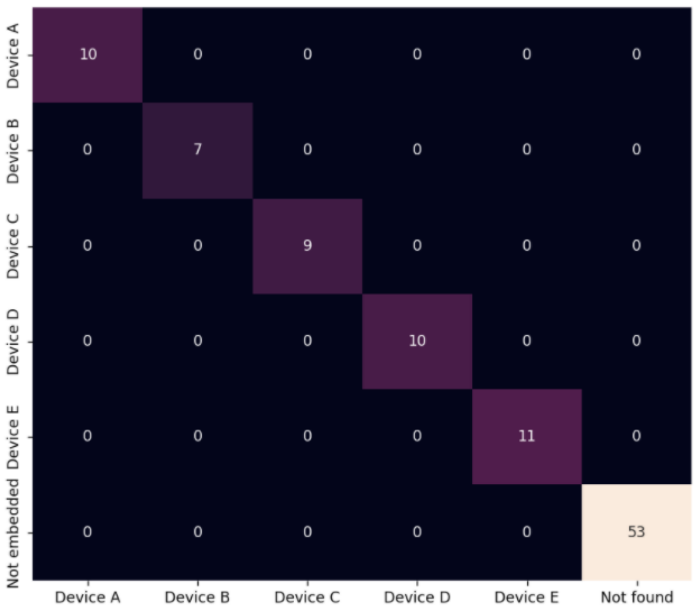
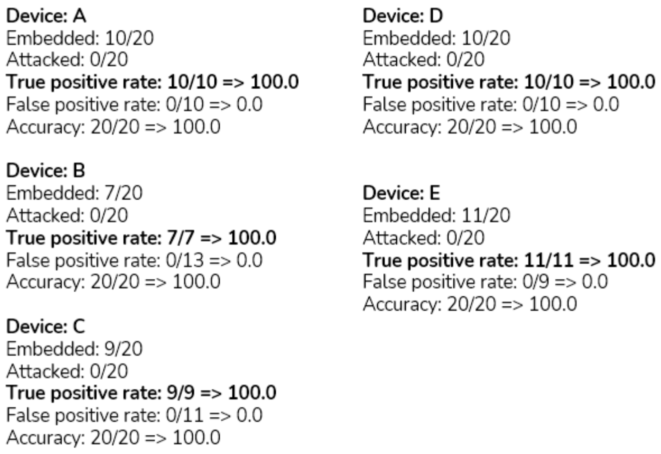


Figure 4: Confusion table without attacks

"true" cases, on the columns instead there are the method's predictions. On the diagonal there are the number of times each device has been recognized as itself e.g the classification was correct. As we did for the ROC curves the whole set of images has been used (100 images, 20 per 5 different devices). Such a validation has been made realistic trough the random decisions about *embedding* and *attacking*: not all the images were embedded and attacked. In this way we provide cases in which the method can produce both *false-positives* (e.g the image has not been embedded but the framework claims for a match with a device) and *false-negatives* (the image has been embedded but no match is found). As you can see in the figure there is a case `Not embedded - Not found` that represent the scenario in which the image *has not been embedded* and it was correctly classified as such e.g no mark is found. In short, there are no cases in which the classification has been erroneous. We expected such a behaviour since the ROC without attacks was almost perfect.



Device: A
Embedded: 10/20
Attacked: 0/20
True positive rate: 10/10 => 100.0
False positive rate: 0/10 => 0.0
Accuracy: 20/20 => 100.0

Device: B
Embedded: 7/20
Attacked: 0/20
True positive rate: 7/7 => 100.0
False positive rate: 0/13 => 0.0
Accuracy: 20/20 => 100.0

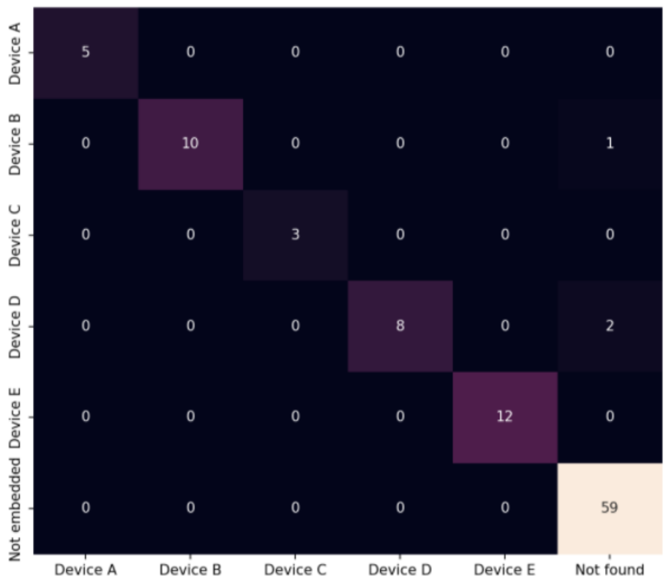
Device: C
Embedded: 9/20
Attacked: 0/20
True positive rate: 9/9 => 100.0
False positive rate: 0/11 => 0.0
Accuracy: 20/20 => 100.0

Device: D
Embedded: 10/20
Attacked: 0/20
True positive rate: 10/10 => 100.0
False positive rate: 0/10 => 0.0
Accuracy: 20/20 => 100.0

Device: E
Embedded: 11/20
Attacked: 0/20
True positive rate: 11/11 => 100.0
False positive rate: 0/9 => 0.0
Accuracy: 20/20 => 100.0

Figure 5: Devices results of confusion table

The Figure 5 provide further data that can be analyzed in addition to the confusion table. In such a table *Device A* has been correctly recognized 10 times. It means that only 10 images over 20 have been embedded. This intuition is confirmed by the actual data we report in the above figure. As you can see the *TPR* is maximum for each device.
The metrics you see are calculated with formulas we introduced in section 1.2



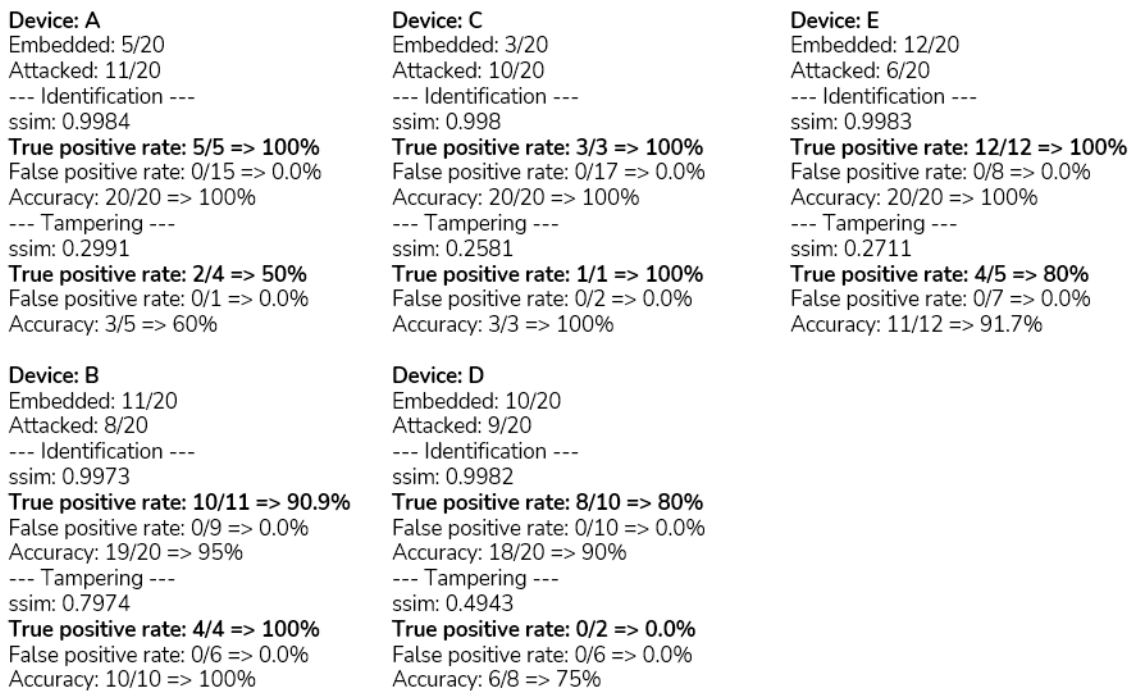
	Device A	Device B	Device C	Device D	Device E	Not found
Device A	5	0	0	0	0	0
Device B	0	10	0	0	0	1
Device C	0	0	3	0	0	0
Device D	0	0	0	8	0	2
Device E	0	0	0	0	12	0
Not embedded	0	0	0	0	0	59

Figure 6: Confusion table with attacks

A confusion table (Figure 6) has been calculated also in case of attacks applied to images. This scenario is useful to test the robustness of the framework. Each image has been attacked exactly once with attacks chosen randomly from the pool below:

1. Gaussian noise: sigma = 150
2. Blur: sigma = 4
3. Sharpening: sigma = 8, alpha = 8
4. Median: kernel = 21
5. Resizing: scale = 0.15
6. Jpeg compression: QF = 15

As you might have noticed the attacks are weaker respect to the ones used for ROC calculation. The main reason is that we wanted to show another scenario in which attacks were "realistic" in terms of image visual quality. The attacks used for ROC calculation were highly destructive and still the identifier has good performances. Below (Figure 9) you can find and example of how much difference there is between Gaussian noise attacks we used.



```
Device: A
Embedded: 5/20
Attacked: 11/20
--- Identification ---
ssim: 0.9984
True positive rate: 5/5 => 100%
False positive rate: 0/15 => 0.0%
Accuracy: 20/20 => 100%
--- Tampering ---
ssim: 0.2991
True positive rate: 2/4 => 50%
False positive rate: 0/1 => 0.0%
Accuracy: 3/5 => 60%

Device: B
Embedded: 11/20
Attacked: 8/20
--- Identification ---
ssim: 0.9973
True positive rate: 10/11 => 90.9%
False positive rate: 0/9 => 0.0%
Accuracy: 19/20 => 95%
--- Tampering ---
ssim: 0.7974
True positive rate: 4/4 => 100%
False positive rate: 0/6 => 0.0%
Accuracy: 10/10 => 100%

Device: C
Embedded: 3/20
Attacked: 10/20
--- Identification ---
ssim: 0.998
True positive rate: 3/3 => 100%
False positive rate: 0/17 => 0.0%
Accuracy: 20/20 => 100%
--- Tampering ---
ssim: 0.2581
True positive rate: 1/1 => 100%
False positive rate: 0/2 => 0.0%
Accuracy: 3/3 => 100%

Device: D
Embedded: 10/20
Attacked: 9/20
--- Identification ---
ssim: 0.9982
True positive rate: 8/10 => 80%
False positive rate: 0/10 => 0.0%
Accuracy: 18/20 => 90%
--- Tampering ---
ssim: 0.4943
True positive rate: 0/2 => 0.0%
False positive rate: 0/6 => 0.0%
Accuracy: 6/8 => 75%

Device: E
Embedded: 12/20
Attacked: 6/20
--- Identification ---
ssim: 0.9983
True positive rate: 12/12 => 100%
False positive rate: 0/8 => 0.0%
Accuracy: 20/20 => 100%
--- Tampering ---
ssim: 0.2711
True positive rate: 4/5 => 80%
False positive rate: 0/7 => 0.0%
Accuracy: 11/12 => 91.7%
```

Figure 7: Confusion table data with attacks

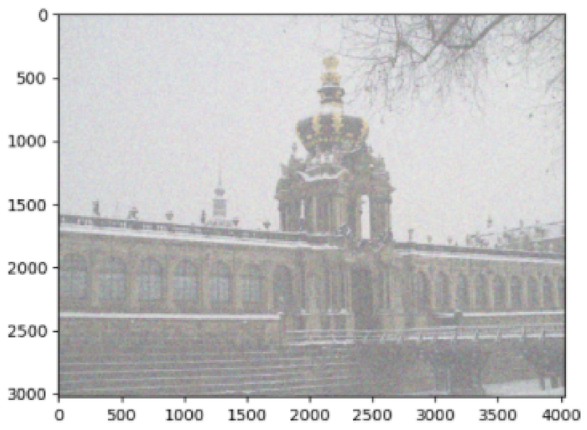


Figure 8: Gaussian noise sigma 200

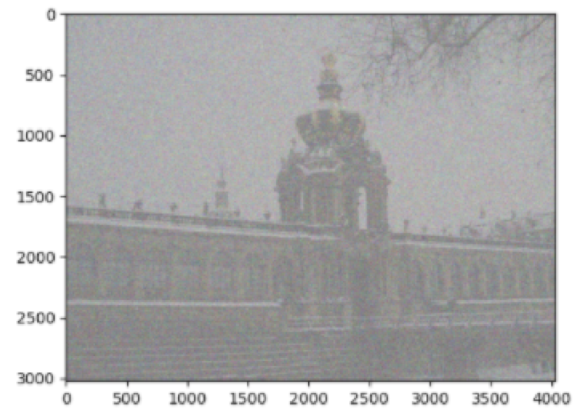


Figure 9: Gaussian noise sigma 450

5.3 Conclusions & future works

In short, we are satisfied by the results obtained but there is still margin to do better. Some of the future works that we would explore are:

1. Tests with combined standard attacks
2. Tests with geometric attacks
3. Tests with a larger and more heterogeneous dataset
4. Implementation of conditions that can identify devices into the upperbound-lowerbound thresholds space e.g *gray-area*
5. Embedding using bigger chunks to select the area with more gray pixels

We already started to explore the very last point. We conducted many "empirical" tests and we were able to find some attacks patterns that could be exploited in order to increase the number of times the identifier declares a match case.
The main idea is the following: consider the whole pairs of collected accuracies instead of the best candidates only. Generally speaking *"Exploit of the information given by all accuracies pairs using mathematical tools like* ***average*** *and* ***standard deviation."***

5.3.1 Gray area analysis example

Assume:
Device embedded: Device A
Idea: If the maximum accuracy is greater than the accuracies average excluded the greatest + 5, select the greatest accuracy.
Greatest accuracy: 52.58
Accuracies avg: 47,352 (excluded the greatest)

52,58 > 47,352 + 5 → Claim the device associated to the greatest accuracy pair as the matching device.
As you can see this intuition that comes from empirical observation is able to lower the upperbound of the accuracy metric.

References

[1] Zhou Wang - Alan Conrad Bovik. *Image Quality Assessment: From Error Visibility to Structural Similarity.* 2004. URL: `https://www.cns.nyu.edu/pub/eero/wang03-reprint.pdf` (visited on 02/27/2022).

[2] Zhou Wang - Alan Conrad Bovik. *The SSIM Index for Image Quality Assessment: materials.* 2004. URL: `https://www.cns.nyu.edu/~lcv/ssim/` (visited on 02/27/2022).

[3] Sebastian Bosse Johannes Erfurt Christian R. Helmrich. "A Study of the Perceptually Weighted Peak Signal-To-Noise Ratio (WPSNR) for Image Compression". In: (2019).