Politecnico di Torino

Energy management for IoT

Project

Energy impact of image watermarking in serial bus transmission

Students:
Matteo Fragassi (s317636)

Contents

List of Figures

CHAPTER 1

Introduction

The importance of media content in increasingly more fields comes with the need of security and authentication of such content. A possible solution is **Digital watermarking**. It is a method to provide protection from tampering and alteration through the insertion of a secret information (i.e. the watermark) into the sensitive data. The watermark is then extracted to prove the ownership of the data.

With the adoption of digital watermarking in IoT devices, it is necessary to consider the possible negative impacts of this technique. Since energy is one of the main concerns of the IoT world, the objective of this project is to analyze the difference of energy consumption of images transmitted with a serial protocol when a watermark is applied.

1.1 Watermarks classification

Digital watermarking techniques can be classified according to different parameters [1]:

- **Characteristics/Robustness**

  The watermarking is `robust` if it can withstand some attacks, meaning that, even after being altered, the watermark can still be completely retrieved. Different watermarking techniques can resist to some extent to different manipulations of the digital content. This property is especially important when the watermark is used to prove the ownership of the data of interest.

- **Perceptibility**

  The watermarking is `perceptible` if the embedding process modifies the original media content in a perceivable way (e.g. an image with a visible watermark applied on it). When this doesn't happen the technique is `imperceptible`.

- **Domain**

  Watermarking techniques can be divided in `spatial` and `frequency` domain. For digital images, spatial domain techniques directly change the value of the pixels thus modifying the visual perception of the image. Some of the most known algorithm are LSB (least significant bit), color manipulation and Patchwork method. On the other hand, frequency domain watermarking transforms the image in the frequency domain and then manipulates the transform function coefficients in order to embed the watermark. DCT (Discrete Cosine Transform), DFT (Discrete Fourier Transform), and DWT (Discrete Wavelet Transform) are only some of the possible techniques used in this domain. A mixed approach is also possible, thus making an algorithm work in the `space-frequency` domain [3].

CHAPTER 2

Implementation

2.1 Overview

All the code is stored in different files inside the `/src` folder:

- `header.py`: Header file with all the main Python packages and the macros useful to manage the simulation environment.
- `main.py`: Main file of the project. It contains all the simulation code.
- `plot.py`: File used to analyze the simulation results, one at a time or many in parallel, by computing meaningful metrics and plotting them.
- `bit_replace.py`: Code of the `Bit Replacement` library 2.2.
- `func.py`: File containing utility functions. The most important is `energyConsumption`. This function computes the energy consumption of an image transmitted on a serial line as number of transitions between consecutive bits. The transmission protocol for images is considered to be one channel at a time in row-major (i.e. one row at a time).

All other directories can be changed by modifying the appropriate macro in `header.py`. By default `/images` contains the simulation images, `/wm` is the watermarks folder and `/simres` stores the simulation output files.

2.2 Watermark algorithms

The project has been implemented using three libraries that exploit different watermarking techniques:

- **Blind Watermark**

  This frequency domain watermarking algorithm uses DCT and SVD (Singular Value Decomposition) to embed a watermark image or string in another image.

  The watermark embedding capacity is limited to 1 byte every 64 bytes of the modified image and, in case an image is used as watermark, its quality is severely reduced. In fact, the watermark is read in grayscale and then turned into a boolean matrix by turning each pixel value into 'True' and 'False' depending on whether it is higher or lower and equal than 128 (line 41 of `blind_watermark.py`). The embedded watermark is the result of this operation so when it is extracted it's inevitably different from the original one 2.1. In some cases this problem is worse

because the image has smoother color changes 2.2. However, despite this problem, the embedding phase works correctly, so the library fits the requirements of the project.

Unfortunately the only apparently modifiable parameter of the algorithm (i.e. `block_shape`) cannot be manipulated without changing the source code of the library, since it is instantiated in the constructor but never used.

The file `blind_watermark.py` needs to be modified before running `main.py` with an image as watermark by substituting lines 102-103 with the following code:

```
    if out_wm_name is not None:
        cv2.imwrite(out_wm_name, wm)

```

This code removes the necessity to write on a file the extracted watermark during a debug procedure. In case this is not a problem it is enough to add the following parameter when calling the `extract` method.

```
    bw_wm = bwm.extract(...,out_wm_name=<path of the output file>,...)

```

- **Invisible Watermark**

  This library offers three methods to embed a watermark into an image:

  - `DWT and DCT`

    Its implementation has severe capacity problems, not allowing to embed more than 1 byte of information every 58710 bytes (i.e. 21 bytes is the maximum embedding capacity for a 1024x1024 image).

  - `DWT, DCT and SVD`

    This algorithm allows to embed more information that the previous one (i.e. 1 byte every 517 bytes) but it also performs the embedding/extraction three times slower. This is the algorithm used for the project.

    The algorithm seems to be modifiable through some parameters (i.e. `scales` and `block`) during the embedding phase.

  - `Machine learning based`

    Even though the neural network seems really promising, the uncertainty on the result variation with different datasets lead not to consider this approach as a viable solution.

  The library methods accept watermarks in many formats (i.e. ipv4, uuid, bit strings, byte strings and Base16) but they do not process directly images and some kind of strings. For these reason the watermark is transformed into a byte array before being fed to the `encoder` object.

- **Bit Replacement**

  This is a simple spatial domain algorithm that acts at the bit level by substituting some bits in a certain position of the original image with some bits of the watermark. The position of the substituted bit is the same in every byte and the modified bytes do not need to be consecutive (i.e. the stride can be more than one). The code for this algorithm was created by taking inspiration from the `VideoDigitalWatermarking` library [6].

  With this technique it is possible to embed at most 1 byte every 8 bytes of the original image (1 bit for each byte). This is the scenario where the watermark bits are embedded in consecutive bytes of the image. If the algorithm stride (i.e. `BYTE_STRIDE`) is not 1, it is possible to embed 1 byte every 8*`BYTE_STRIDE` bytes.

It is also possible to choose the position of the bits that are going to be replaced using the macro REPLACED_BIT and the channel of the original image that is going to be modified with BRCH (the algorithm modifies only one channel). It is better to select bits near to the LSB, in order to avoid changing the pixel value too much, and the channel 0 (i.e. blue in the BGR color space), since the human eye is less sensitive to changes of the blue component [7].

In terms of watermarking features, the Bit Replacement algorithm cannot resist any kind of attack. Even if a bit is lost, the watermark is not going to be extracted correctly. The other two algorithms are more robust when it comes to security but they have a lower embedding capacity, as already mentioned.

The algorithms are activate and deactivated in the simulation through the BIT_REPLACEMENT, BLIND_WM and INV_WM macros. It is also possible to activate the debug mode, when the corresponding algorithm is selected, by setting DEBUG_BR, DEBUG_BW and DEBUG_IW.

2.3 Watermark handling

Watermarks have to be placed under the path specified by WM_PATH. Two types of watermark are supported, as reported by the WM_TYPES macro:

- **Strings**

  These watermarks are listed, one per line, in the file WM_STR_FILE. The string used for the current simulation is specified by WM_LINE. If the string is too big to be embedded in at least one image of the provided dataset, an error message is printed on the terminal and the simulation is stopped.

- **Images**

  While the transmitted images are treated like colored images in the BGR color space, watermarks are handled in grayscale. This is done not only for ease of use (one channel to embed instead of three), but also because Blind Watermark and Invisible Watermark read the watermark in grayscale anyway. Images used as watermark may also need to be resized before being used due to the capacity constraints of the watermarking algorithms. The resized images are placed under WM_PATH.

  Images file names have a specific format specified by the WM_NAME_FULL macro for the basic image and by WM_NAME_RESIZED for the resized version. The macros WM_NAME, WM_EXTENSION and WM_DIM are used to create the name. In particular, WM_DIM is the maximum number of pixels per dimension.

The type of watermark for a simulation can be changed by appropriately setting WM_TYPE and the maximum watermark dimension in bytes is manually recorded in WM_BSIZE, together with WM_SIZE, after executing a preliminary simulation step 2.5.

Independently from the type of watermark, the program works with data types that are byte sized. This means that the pixels of an image cannot be represented by more than 8 bits per channel. It also means that utf-16 characters, are decomposed in two bytes. While this is a real limit for images, many kind of string can be used as long as they are encoded in the original format when the watermark is extracted.

The simulation results in SIM_RES_PATH have been produced using four different watermarks, two images and two strings. One of the images has simple shapes and color palette while the other one is colorful, with each color that fades smoothly into the next one. The strings are different too since one of them is 32 bytes long while the other one occupies 2025 bytes. However, they only have utf-8 characters.

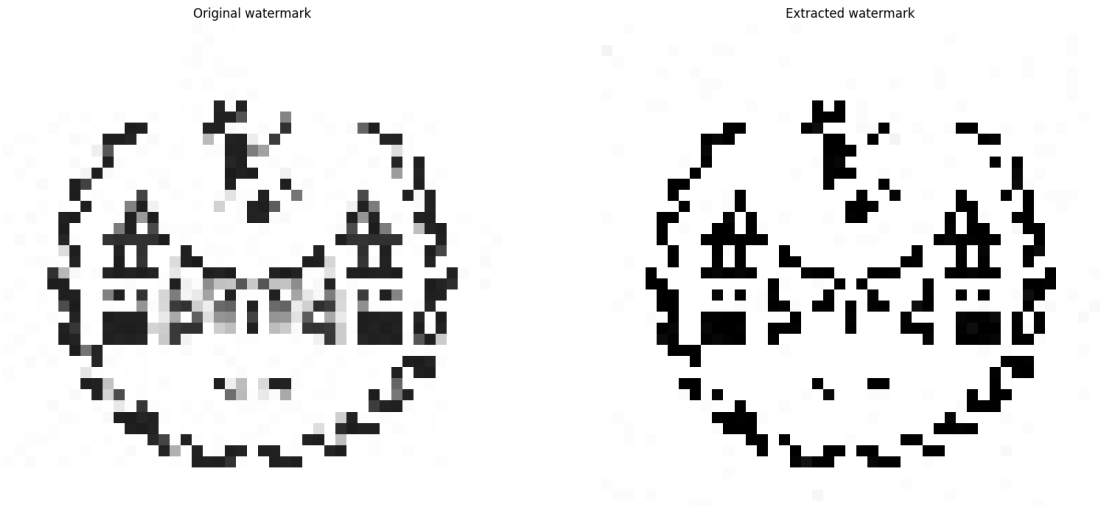

Figure 2.1: Original grayscale watermark vs extracted watermark [Blind Watermark]

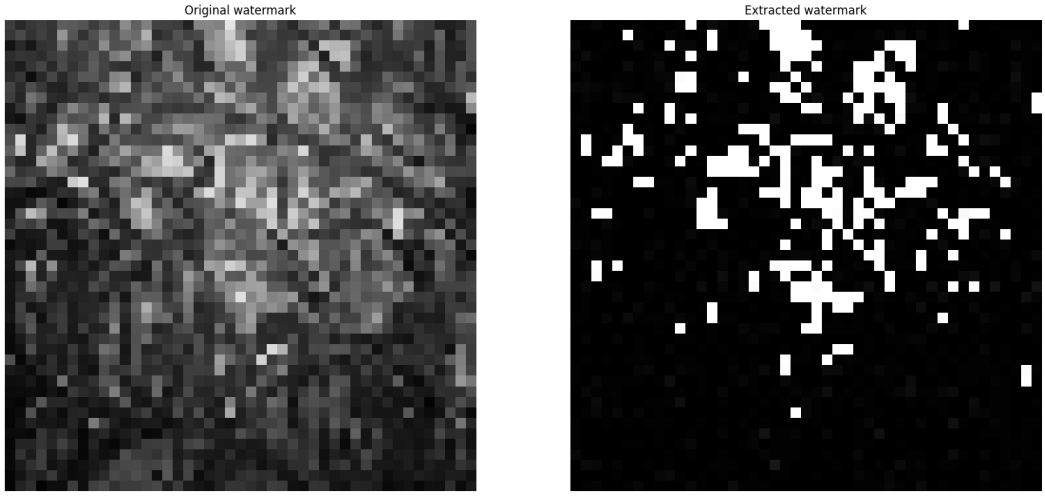

Figure 2.2: Original vs extraced watermark [Blind Watermark]

2.4 Simulation details

2.4.1 Datasets

The images "to be transmitted" used for the simulation are under `IMGS_PATH`. They are only a subset of two much larger datasets:

- **DOTA**[8]: Aerial images taken from Google Earth
- **NuImages**[9]: Autonomous driving images (i.e. car cameras)

2.4.2 Results record

At the end of a simulation a JSON file can be found inside `SIM_RES_PATH`. The file name has a different format, always specified by `DATA_FILENAME`, depending on whether the watermark is an image of a string. A simulation with the former type of watermark produces an output file that contains the watermark file name. This means that a simulation with a watermark will write on the previously generated file, no matter the size of the watermark. The same may happen with string watermarks but, in this case, because the watermark used for a certain simulation is identified by the first six characters of the string.

The JSON data structure contained in the output file contains the energy consumption data of the different watermarking algorithms other than the basic energy consumption of the image. The keys are managed using the macro `DATA_KEYS` and can be changed to some extent. In fact, the code inside `plot.py` works as long as the key ordering is:

```
{'algorithm1':[],'algorithm2':[],...,'algorithm3':[],'BaseImg':[],'Img':[]}
```

To change the data structure it is necessary to modify both the dictionary at the end of `main.py` and the keys list `DATA_KEYS`.

2.5 Code usage

All the necessary packages to run the project are listed in `requirements.txt`. Use the following command to install them:

```
pip install -r requirements.txt
```

If you want to use the same package versions the project was developed with just use `requirements_stable.txt` instead of `requirements.txt`.

The simulation flow can be tuned through the environment variables in `header.py` as explained in the previous sections. Here is a summary of the most important ones:

- `IMGS_PATH`: root folder of the images used in the simulations;
- `WM_SIZE_COMP`: 'True' to compute the maximum watermark size in bytes, 'False' otherwise;
- `WM_BSIZE`: maximum watermark size in bytes;
- `WM_PATH`: root folder of the watermarks used in the simulations;
- `WM_TYPES`: allowed watermarks types;
- `WM_TYPE`: currently used watermark type;
- `WM_STR_FILE`: string watermarks file (one per line);
- `BIT_REPLACEMENT`, `BLIND_WM` and `INV_WM`: 'True' to consider that algorithm in the simulation, 'False' otherwise;
- `DEBUG_BR`, `DEBUG_BW` and `DEBUG_IW`: 'True' to debug a specific algorithm (that algorithm has to be active);
- `SIM_RES_PATH`: root folder of the simulation results;
- `DATA_KEYS`: entries of the data structure the simulation results are recorded into; it can be found at the end of `main.py`.

Before starting a real benchmark, it is necessary to compute the maximum watermark size by setting `WM_SIZE_COMP=True` and executing:

```
cd src
python3 main.py
```

Be sure to follow the instructions on the terminal.

The real benchmark can be started by executing `main.py` after setting `WM_SIZE_COMP=False`. The simulation progress will be reported on the terminal and, at the end, the result is going to be recorded as explained before 2.4.2. At this point the results can be analyzed by executing:

```
python3 plot.py
```

This will start an interactive menu to select the appropriate file between the ones inside the simulation results folder.

CHAPTER 3

Results

By analyzing the simulation results in `SIM_RES_PATH` it is possible to notice that the energy consumption with the watermarks applied is not so different from the basic energy consumption 3.1. In fact, on average, the energy consumption when the watermarks are embedded in the transmitted images is never 0.22% higher than the basic images energy consumption. In most cases it's not even close to 0.1%.

The maximum positive (i.e. that consumes more energy) variation from the basic image energy consumption is 5.5%. This value is reached by the Invisible Watermark algorithm when 2.1 (left figure) is embedded into 3.2. A possible explanation is that this image is mostly dark while the embedded watermark has a white background. However, this observation is purely qualitative since the variation is highly dependent on the watermarking algorithm, the watermark and the basic image. In fact, the Blind Watermark algorithm produces its maximum variation when 2.2 (left image) is embedded into 3.3. In this case is more difficult to explain why there is such a high variation since similar images, like 3.4, have an energy consumption closer to the one of the basic image.

In some cases, the embedding causes the transmission to consume less energy 3.6. The maximum negative (i.e. that consumes less energy than the basic image) variation is -2.7% and it is achieved with the Blind Watermark algorithm by embedding 2.1 (left image) to 3.5. This case is another exception to the initial observation since the image is mostly dark while the watermark has a white background.

In general, the Bit Replacement algorithm is the less impacting than the other algorithms in terms of energy consumption. In fact, the provided simulations show an energy consumption variations lower then 0.01%. The other tested techniques usually have an energy consumption variation slightly higher than the Bit Replacement one, with some rare exceptions.

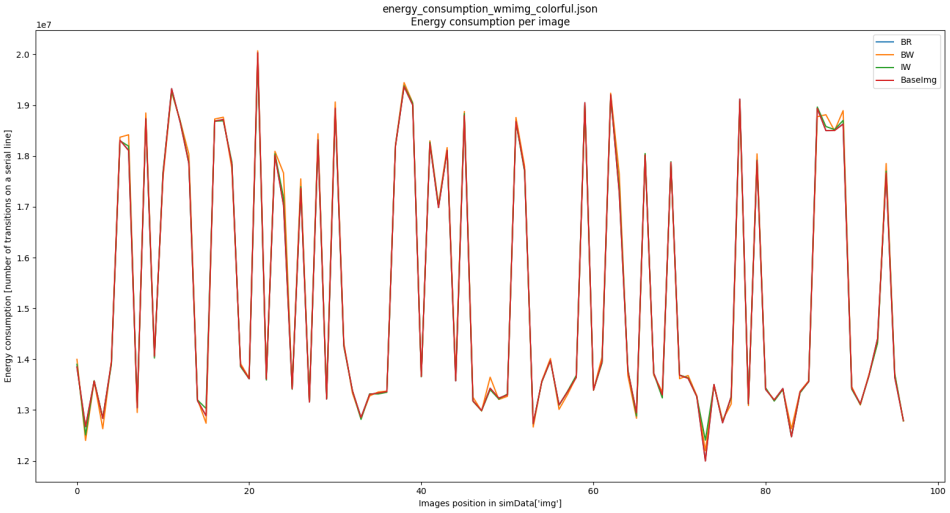

Figure 3.1: Energy consumption when "colorful_45x45.jpeg" is used as watermark

Figure 3.2

Figure 3.3

Figure 3.4

Figure 3.5

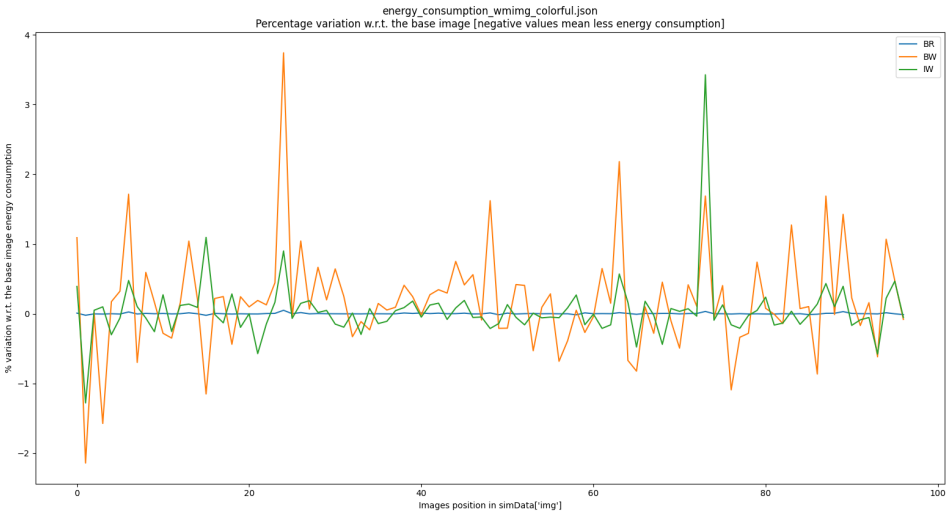

Figure 3.6: Energy consumption variation with "colorful_45x45.jpg" as watermark

Bibliography

[1] Shweta Wadhera, Deepa Kamra, Ankit Rajpal, Aruna Jain and Vishal Jain (2022) *A Comprehensive Review on Digital Image Watermarking*, https://arxiv.org/abs/2207.06909

[2] Mahbuba Begum and Mohammad Shorif Uddin (2020) *Digital Image Watermarking Techniques: A Review*, https://www.researchgate.net/publication/339348660_Digital_Image_Watermarking_Techniques_A_Review

[3] Rani Kumari and Abhijit Mustafi (2022) *The spatial frequency domain designated watermarking framework uses linear blind source separation for intelligent visual signal processing*, https://www.frontiersin.org/articles/10.3389/fnbot.2022.1054481/full

[4] Blind Watermark, https://pypi.org/project/blind-watermark/

[5] Invisible Watermark, https://pypi.org/project/invisible-watermark/

[6] Video Digital Watermarking, https://github.com/piraaa/VideoDigitalWatermarking

[7] https://www.sciencedirect.com/science/article/pii/S1319157816301483?via%3Dihub#s0010

[8] DOTA, https://captain-whu.github.io/DOTA/

[9] NuImages, https://www.nuscenes.org/nuimages