FACEBOOK AI

Robust Image Watermarking

Watermarking Images in Self-Supervised Latent Spaces

Pierre FERNANDEZ - Research Internship

Supervisors: Matthijs DOUZE & Hervé JÉGOU

Facebook AI Research

Master MVA

May-September 2021

Résumé Opérationnel

Ce stage s'inscrit dans le cadre du traçage de contenu et du tatouage numérique. L'objectif est de modifier une image de manière invisible de sorte que l'on puisse extraire le message ou la marque que l'on y a déposé. Des applications possibles sont la protection du droit d'auteur, l'authentification de contenu et la recherche d'images par similarité dans un grand jeu de donnée. Alors que les premières méthodes de tatouage permettaient d'être robuste vis-à-vis de certaines transformations spécifiques, des architectures d'apprentissage profond plus récentes ont permis de générer des marques robustes à un large éventail d'augmentations.

L'objectif est d'encore améliorer les performances en utilisant l'apprentissage profond. Plus précisément, nous exploitons les extracteurs de descripteurs émergents de l'apprentissage auto-supervisé, et nous étendons les travaux précédents [62] et [55], qui étaient plus proches du tatouage zero-bit, à du tatouage robuste multi-bit. La méthode que nous développons améliore très notablement la robustesse dans la configuration du tatouage zero-bit. Elle permet également d'obtenir des résultats similaires à ceux de l'état de l'art en matière en multibit ou de dissimulation de données, sans avoir à entraîner de réseau spécifique pour le tatouage.

Executive Summary

This internship falls within the scope of content tracing and watermarking. The goal is to modify an image in an invisible way such that we can later assess the message or the mark we put on it. This has applications in copyright protection, content authentication and large scale similarity search. While previous handcrafted watermarking methods allowed robustness against some specific transformations, more recent end-to-end deep-learning architectures generated watermarks robust to a wider range of transformations.

The objective is to improve performance by leveraging modern neural network learning strategies. More specifically we exploit emerging feature extractors from self-supervised learning, and extend the previous works [62] and [55], that were closer to zero-bit watermarking, into a robust multi-bit watermarking approach. The method we develop very noticeably improves robustness in the zero-bit watermarking setup. It also obtains similar results than state-of-art on multi-bit watermarking or data hiding, without needing to train a network explicitly for the watermarking task.

Acknowledgements

I would first like to thank Hervé and Matthijs, my principal advisors during the internship, for the subject they prepared so well and their precious guidance throughout the 4 months. I am also grateful to Alexandre Sablayrolles who provided excellent research leads and introduced me to running experiments on the FAIR clusters in the best way I could hope.

I would also like to thank my fellow interns Quentin, Timothée for their help and their friendly welcome, and Kenza for sharing a big part of the internship with me and making the work more fun and less lonely.

I am greatly thankful to Zoe and Vasil for our previous conversations, as well as Piotr and Mathilde for giving me the opportunity to take part in the DINO_V2 meetings and present my project to people I admire.

Finally, I would like to express my gratitude to Teddy for the time he spent for me and with whom I look forward to work in the future.

Contents

1 Introduction

Every day, 3.2 billion images are shared online, often being transformed and altered from user to user. This rise in multimedia sharing makes it more and more difficult to effectively authenticate content or comply with intellectual property. To address this goal, watermarking and steganography [15] have been extensively used in the past few years. Both aim to hide a message in an image by slightly modifying it. One can later detect or extract the watermark even if the image has suffered some pixel-value (compression, histogram equalization, etc.) or geometric (cropping, rotation, etc.) transformations. While steganography focuses on the secrecy of the hidden messages, watermarking focuses on their robustness.

Usually, watermarking methods must meet 3 intertwined criteria. The first one is *imperceptibility*: the image distortion induced by the watermark encoding must be invisible. The second one is *capacity*: the process must be able to encode messages that are long enough. The last one is *robustness*: the watermark must be retrieved even if the image has been distorted. Imperceptibility can also be understood in the sense of *secrecy*, i.e. an adversary should not be able to detect the mark or decode the message.

In looking at these criteria under the lens of the marking latent space onto which the mark is added, traditional approaches attempt to meet these criteria in two different ways. Spatial domain methods encode the message by directly modifying the pixel values of the cover image. Transform domain ones (or frequency domain) modify the coefficients of its transforms e.g. discrete Fourier transform (DFT), discrete cosine transform (DCT), or discrete wavelet transform (DWT) and mark on the feature space of these transforms.

More recently, deep learning based methods have emerged as a viable alternative, and showed improved robustness to a broad type of transformations. Implicitly, it allows to mark in the latent space induced by a neural network, which is a priori invariant to all transformations that do not affect the decision of the classifier. Examples include directly marking into the semantic space associated with a given set of classes like ImageNet [62], or create a space explicitly created for robust watermarking. For the latter, the most part consist of building end-to-end adversarial architectures as in HiDDeN [74], or other papers that followed.

Our key insight is to leverage the intrinsic robustness of self-supervised networks and properties of their latent space to watermark images. Indeed, the ideal latent space should create representations invariant to augmentations, without having explicit knowledge of the original pixels localization. This exactly corresponds to the setting some self-supervised methods aim at optimizing [31, 12]. Using such pre-trained networks would allow to take advantage of recent advances in self-supervised learning for visual representation learning and would generate watermarking algorithms "for free", without having to retrain entire end-to-end architectures (like HiDDeN).

Our method falls under the scope of blind image watermarking, meaning that the decoder does not have access to the original image. Further, it provides natural security due to the black-box nature of deep learning models.

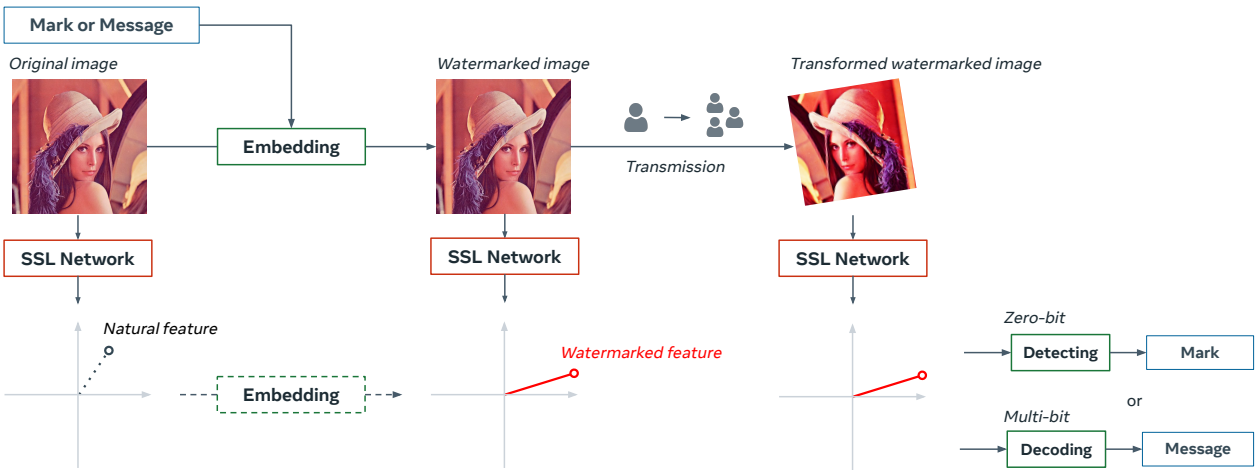


Figure 1: Overview of our method for watermarking in self-supervised-latent spaces. A self-supervised network trained with DINO [12] builds a latent space on which the mark is added by the embedding process. Its effect is to shift the image's feature into a well-specified region of the latent space, such that transformations applied during transmission do not move much the feature. The mark's detection (zero-bit watermarking setup) or message's decoding (multi-bit watermarking setup) is performed in the same latent space.

Our contributions are:

- We adapt the watermarking process developed by Vukotić et al. by adding data augmentation during the image optimization, as well as PSNR clipping that allows to optimize as much as possible under the image quality constraint. We also extend the zero-bit method to multi-bit data hiding.
- We show that self-supervised networks provide excellent latent spaces to watermark images.
- We extensively test our methods against previous methods both in zero-bit and multi-bit setups.

The report is organized as follows:

- We review the watermarking bibliography and related work (section 2)
- We provide theoretical foundations and motivations for zero-bit watermarking with self supervised networks (section 3).
- We extend the method to work in the multi-bit setup, and propose several ways of achieving this (section 4).
- We extensively test our methods and compare it to previous methods (section 5)
- We analyse some aspect of our methods and present some unsuccessful leads (section 6)

2 Related work

2.1 Image watermarking

Image watermarking [15] consists of inserting a mark/message into an image using a secret key. The information should then be retrieved even if the image has undergone transformations such as compression, contrast change, crops, rotations, etc. We usually distinguish zero-bit watermarking, where we encode the presence or absence of the mark, from multi-bit watermarking where the goal is to encode a bitstring message.

It usually aims to optimize three criteria. The first is *capacity*: we want to embed as much information as possible into the cover media. The second is *imperceptibility*: the image distortion must be low. The third is *robustness*: we want the information to be retrieved even if the image has suffered many transformations. (One may also consider robustness to specific attacks that aim at removing the watermarking, which we don't consider in this work.) There is an implicit trade-off between the 3 axes and it is hard to have at the same time a robust watermark with high capacity and low image distortion.

Figure 2: The 3 criteria watermarking algorithms must meet.

Traditional image watermarking methods are usually classified in 2 categories depending on the space on which the watermark is embedded. In *Spatial domain* techniques, the watermark is encoded by directly modifying pixels (such as simply flipping low-order bits of selected pixels) while *frequency domain* (or transform domain) techniques alter some frequency coefficients obtained by transforming the image into the frequency domain.

To cite a few pioneering works in spatial domain watermarking, this is the case of Ni et al. [48] where the algorithm utilizes the zero or the minimum points of the histogram of an image and slightly modifies the pixel grayscale values to embed data into the image (this algorithm is also reversible, meaning that one can also retrieve the original, un-distorted image). Nikolaidis and Pitas [49] casts the watermark by slightly modifying the intensity of randomly selected image pixels, which can be done in such a way that the watermark is resistant to JPEG compression and lowpass filtering. The detection is carried out by comparing the mean intensity value of the marked pixels against that of the pixels not marked using statistical hypothesis testing.

The second category (and the one that usually provides better robustness) is *frequency domain* watermarking [16]. The first step is to use a transformation method such as discrete cosine transform (DCT) to compute coefficients that act as descriptor of the original image. The watermark is then added to these coefficients and mapped back onto the original pixel space to generate the watermarked image. The motivation behind it is that these coefficients provide descriptors invariant to the transformations. Among the transform domains that were used in previous watermarking methods, we can cite Discrete Fourier Transform (DFT) [60], Quaternion Discrete Fourier Transform (QDFT) [50], Discrete Cosine Transform (DCT) [6], SVD-DCT [41], Discrete Wavelet Transform (DWT) [66], DWT-DCT [21] (this list is far from being exhaustive).

We usually distinguish zero-bit watermarking and multi-bit watermarking. In zero-bit watermarking [24], a watermark, carrying no hidden message, is inserted in the content. The watermark detector then checks for the presence of this particular weak signal in content. Among the zero-bit watermarking method, the hypercone detector one is of particular interest as it has been proven to be near to optimal [47, 26]. In this case, the presence of the watermark is assessed by verifying whether the image descriptor lies in a hypercone of the feature space (the method is pictured in Figure 4). In general watermarking also considers the case where the marks carry a number of bits of information [15]: it is called multi-bit watermarking.

2.2 Deep-learning-based watermarking

In the last decade, deep learning-based data hiding models have emerged as a viable alternative to traditional watermarking algorithms. More often than not, they are built as encoder-decoder networks, where an encoder embeds the mark in the image and a decoder tries to extract it. They are trained end-to-end to invisibly encode information while being resilient to a broad type of transformations. They present the advantage of not needing expert knowledge of watermarking algorithms to encode and decode images robustly to transformations.

One of the first deep learning architectures for watermarking [37] was made of two Convolutional Neural Networks (CNN) that acted like auto-encoder and outperformed the best traditional watermarking methods both in robustness and imperceptibility. However, it was non-blind, meaning that it needed the original image to decode the data.

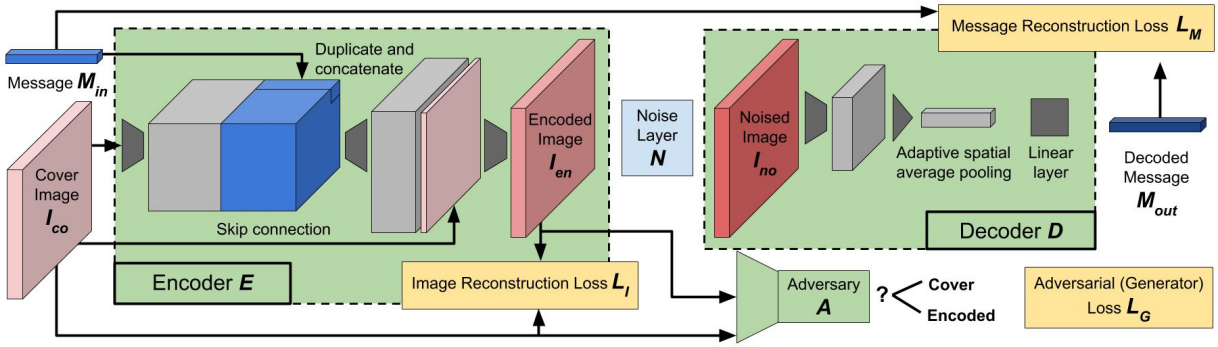


Figure 3: Example model overview: encoder-decoder network from HiDDeN [74]

HiDDeN [74] removed this need and has been a very influential paper since its release. It jointly trains encoder and decoder networks with noise layers that simulate what could happen to an image when transmitted. Inspired by Generative Adversarial Networks (GAN) [29], it also uses an adversarial discriminator to improve visual quality. It is the origin of several extensions that tried to improve its performance and practicality. Among these works, [40] extends the method to arbitrary image resolutions and message lengths, [43] adds adversarial training that brings robustness to unknown transformations and [67], [70] embed the mark with an attention filter further improving imperceptibility. ReDMark [2] added a circular convolutional layer that diffuses the watermark between more image regions. Finally, ROMark [65] uses the architecture of HiDDeN but computes the worst-case attacked image as if an adversary was trying to remove the mark, making the method more robust overall.

For more information on the subject we refer the reader to the work of Byrnes et al. [9]. This very clear and thorough survey describes recent developments in deep learning techniques for data hiding for the purposes of watermarking and steganography.

Alongside the encoder-decoder architectures, Vukotić et al. propose a very different approach that is closer to traditional watermarking methods. Indeed, it utilizes neural networks not for end-to-end training of a watermarking pipeline, but as a fixed transform into a latent space. From this transform, the authors fall back to the hypercone detector for zero-bit watermarking by pushing the image feature onto a hypercone, and

mapping back the watermarked feature onto the image space by using SGD over the pixels (the method is explained with more details in subsection 3.3). The zero-bit watermarking method presented in [61] showed interesting preliminary results, but failed to create watermarks robust to a $30°$ rotation. (Rotation is a good test bench for watermarking robustness since neural networks are usually not invariant to rotation.) The follow-up work [62] aims to increase the inherent robustness of the classification networks by applying progressively harder transformations when training those networks (note that watermarking is not involved in the training process). Even if the results are substantially improved by using a network trained with transformations, relatively small transformations still remove the watermarks with high probability (e.g. $40°$ rotation).

2.3 Steganography

Whereas watermarking focuses on robustness, steganography focuses on the secrecy of the watermarks. It transmits messages that must not be detected or decoded by an adversary. First steganography methods encoded the message in the least significant bits of the image (LSB) [23]. However, it has been showed to be statistically detectable, the full message can even be retrieved in some cases [22]. More recent algorithms are content-adaptive. Usually, they assign a cost of changing each cover element (e.g. pixel or transform coefficient) and then embed a given message while minimizing the expected sum of costs over modified pixels. For instance, HUGO (Highly Undetectable Steganography) [52] computes the distortions from SPAM features for each pixel change by ± 1. The stego message is then embedded onto the pixels with lowest embedding cost. WOW (Wavelet ObtainedWeights) [34] and S-UNIWARD (Universal Wavelet Relative Distortion) [35] extend HUGO by constructing distortions cost from a Daubechies 8-tap wavelet filterbank.

Deep learning techniques have also emerged in the last few years. They present the advantage of being naturally secure assuming the threat does not have access to the embedding and decoding networks. Like watermarking, most of them adopt the encoder-decoder structure with CNNs. HiDDeN [74], already described in subsection 2.2, obtains results similar to HUGO, WOW and S-UNIWARD. [4] focuses on hiding images in other images and UDH [69] aims at the same goal but keeps the image perturbation invariant between all the cover images. SteganoGAN [71] uses a GAN-like architecture to encode arbitrary data and achieves a higher payload than competing deep learning methods. Finally, StegaStamp [58] embeds hyperlinks in images, robustly to the distortions that occur in natural physical photography (and could be considered as a watermarking method more than a steganography one).

2.4 Adversarial examples

Adversarial examples [57, 30] are carefully crafted images with the purpose of confusing neural networks classification of the original image. The generated adversarial images are visually indistinguishable from original images, yet are misclassified with high confidence.

2.5 Self-supervised learning of visual representations

Self-supervised learning (SSL) is a form of unsupervised learning where the data itself provides the supervision, often leveraging its underlying structure. It has widely been used in Natural Language Processing (NLP) tasks and begins to compete with supervised learning in Computer Vision.

Early SSL methods often applied a transformation to the input image and trained the network to predict an information about this transformation or the image For instance, Doersch et al. [18] proposes to predict the relative position of to patches cropped from an image, while [28] predicts a random rotation applied to an image. Deep Cluster [10] proposes another pretext task where the features are clustered and considered as label for the next iterations.

Newer methods, called constrastive ones, are of particular interest. Indeed, instead of predicting a transformation parameter, they directly aim to create features invariant to the transformations. To prevent mode collapse, where the output features are the same for all images, they also aim to separate images from different samples. SimCLR [13], MoCo [33] are good cases in point. However, such methods need to explicitly push aside negative samples, which is a great limitation, since it is impossible to cover the distribution of all the possible pairs.

Non-contrastive methods alleviate the issue by finding new ways to avoid mode collapse. SwAV [11] assigns the representation to a given set of prototype vectors with an optimal transport assignment that prevents collapse on one prototype vector. BYOL [31] and DINO [12] uses a teacher/student architecture where the teacher is a moving average of the student (for more details on DINO, which is used in our work, see

section 5.1). Finally, Barlow Twins [68] uses a statistical prior of feature decorrelation to learn statistically independent features while preventing collapse.

Link have been drawn between contrastive losses and entropy of the latent [7, 20]. Namely, the positive term of the contrastive loss leads to minimization of the conditional differential entropy inside samples of the same category, whereas the negative term of this loss maximizes the entropy of the learned representations. In the same way uniformity [63] provides a theoretical analysis for contrastive representation learning in terms of alignment and uniformity on the hypersphere. This has great importance in watermarking, as it will be explained in subsection 3.2.

3 Zero-bit watermarking with neural networks

Unlike multi-bit watermarking that aims to hide and recover a large number of bits, zero-bit watermarking aims to separate images watermarked with secret key a from non-watermarked ones. Therefore, the zero-bit watermarking decoder is essentially a detector that carries out a statistical hypotheses test between $\mathcal{H}_0$: "the image is not watermarked", and $\mathcal{H}_1$: "the image is watermarked". It gives great importance both to type I and type II errors. Our method builds upon the work of Vukotić et al. [62], but is different with regards to two important components: self-supervised pretrained networks are used to extract the features and a pre-processing step is applied at marking time to impose image constraints and to simulate transformations.

3.1 Hypercone detector

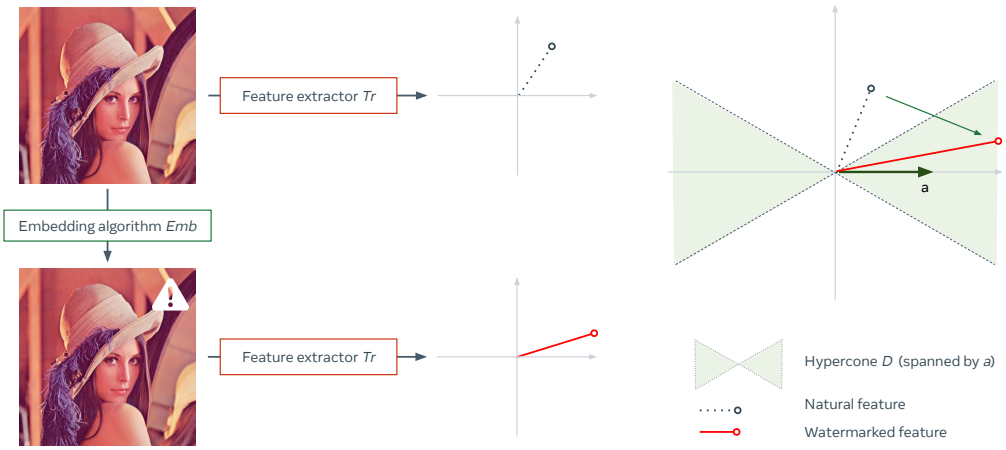


Figure 4: Hypercone detector method. A feature extractor Tr builds the latent space on which the mark is added by the embedding algorithm Emb. Its effect is to shift the image's feature into a hypercone $\mathcal{D}$ of the latent space, spanned by a carrier a. The mark's detection is performed by checking the presence of the feature in the hypercone.

Method overview Our method comprises 3 main components:

- a transformation ϕ that extracts features x from an image $\mathcal{I}$ (i.e. a 3 dimensional tensor with dimensions h, w, c)
- an acceptance region $\mathcal{D}$ that tells whether or not the image is marked based on the test $\phi(\mathcal{I}) = x \in \mathcal{D}$. $\mathcal{D}$ depends on a carrier a.
- an embedding algorithm Emb that adds the watermark on the images: $\mathcal{I}_o \xrightarrow{Emb} \mathcal{I}_m$. It implicitly depends on the transformation ϕ and the acceptance region $\mathcal{D}$.

In the case of the hypercone detector, as the name suggests, the acceptance region is a dual hypercone of semi-angle $\theta \in [0, \pi/2]$ and axis $a \in \mathbb{R}^d$ with $\|a\| = 1$ (a is also called *carrier*):

$$\mathcal{D}_a^\theta := \left\{x \in \mathbb{R}^d : |x^\intercal a| > \|x\| \cos(\theta)\right\} \tag{1}$$

The same hypercone can alternatively be described as the positive region of the function:

$$R_a^\theta(x) = (x^\intercal a)^2 \rho(\theta) - \|x\|^2 \tag{2}$$

where $\rho(\theta) = 1/\cos^2(\theta)$. The acceptance region is then simplified to $\mathcal{D}_a^\theta := \left\{x \in \mathbb{R}^d : R_a^\theta(x) > 0\right\}$. Intuitively, an image $\mathcal{I}$ is marked if $x = \phi(\mathcal{I})$ is sufficiently pushed in the direction of the carrier a. From now on, for the sake of the simplicity, $\mathcal{D}_a^\theta$ and R_a^θ become $\mathcal{D}$ and R.

Theoretical guarantees on the FPR Let a be chosen uniformly over the unit sphere ($\|a\| = 1$). We are interested in the quantity $\text{FPR}_\theta = \mathbb{P}(x \in \mathcal{D} \mid \text{"x comes from a natural image"})$. Under the assumption that the features output by the extractor have an isotropic distribution (this assumption is discussed later), the above quantity reads $\text{FPR}_\theta = \mathbb{P}\left(|x^\intercal a|/\|x\| > \cos(\theta)\right)$, where x is sampled from an isotropic distribution.

A result from statistics [36, 55] shows that the cosine similarity $c(u, a) = |u^\intercal a|/\|u\|\|a\|$ between a fixed vector a on the unit sphere of $\mathbb{R}^d$ and a vector u chosen with an isotropic distribution follows an incomplete beta distribution with parameters $\alpha = \frac{1}{2}$ and $\beta = \frac{d-1}{2}$. Therefore,

$$\text{FPR}_\theta = 1 - I_{\cos^2(\theta)}\left(\frac{1}{2}, \frac{d-1}{2}\right) \tag{3}$$

where $I_\tau(\alpha, \beta)$ denotes the regularized Beta incomplete function (the full proof is available in the Appendix).

We can now ensure that a random vector lies in the hypercone with probability at most p (typically a very small value) by finding θ such that $\text{FPR}_\theta = p$. Hence we can upper-bound by p the false positive rate $\text{FPR} = \mathbb{P}(x \in \mathcal{D} \mid \text{"}x \text{ comes from a natural image"})$, the probability that the detection algorithm will report the presence of a watermark when none is present.

3.2 Self-supervised network as feature extractor

Motivation Two properties must be met by the ideal feature extractor ϕ:

- invariance to geometric and pixel-value transformations so that the extracted feature is almost the same between a watermarked image and a transformed version of it
- invisible perturbations in the image must be able to sufficiently alter the feature to align it with the carrier. Put differently, the extractor should be very sensitive to well designed adversarial attacks. [3, 44] have brought to light the link between high intrinsic dimensionality and vulnerability to adversarial attacks, hence high intrinsic dimensionality should improve the watermarking process.

Our algorithm uses a pretrained neural network ϕ as feature extractor, so that the watermarking process leverages their inherent robustness to data augmentations. Vukotić et al. aimes to increase this robustness by applying progressively harder transformations when training the marking networks on the ImageNet classification task (note that watermarking is not involved in the training process).

We make the assumption that self-supervised learning (SSL) is much more prone to create good marking networks. In addition to not needing any annotated data, it explicitly aims to create features invariant to data augmentation. Besides, it does not suffer from the *semantic collapse* that happens because of supervised learning. Indeed Doersch et al. pointed out that networks trained for classification lose any information that is not necessary for performing the training task. On the opposite, SSL precisely capture more than just classes by creating pretext tasks. It prevents the supervision collapse and keeps a latent space as large as possible that suits the watermarking task.

DINO pretraining Specifically we use DINO [12]: a self-supervised method rapidly described in subsection 2.5. It uses self distillation with no labels and trains a student network by matching its softmaxed outputs to the ones of a teacher network over different views of the same image. The teacher and the student share the same architecture, but while the student is updated by gradient descent, the parameters of the teacher are updated as an exponential moving average of the student's parameters: $\theta_t \leftarrow \lambda\theta_t + (1-\lambda)\theta_s$, where lambda is close to 1.

The invariance of the representations is ensured by random augmentations of color jittering, Gaussian blur and solarization applied on the views during the training. Further, DINO encourages local-to-global correspondence by feeding only global views to the teacher network, whereas the student network also sees smaller crops of the same image.

In the context of watermarking, one may want to add some frequent geometric transformation to strengthen the robustness of the features. Thus, we added random rotations at training time both in the global and the local views.

Normalization layer Usually the output features of a neural network N do not follow a zero-mean isotropic distribution (a case in point is activation functions like ReLU that create features in $\mathbb{R}^+$). This is problematic in 2 ways: the output features might never be able to reach a given hypercone and the theoretical guarantee on the FPR (see Theoretical guarantees on the FPR) does not apply anymore. To alleviate the issue, the output features are whitened by using PCA-whitening (also called PCA-sphering).

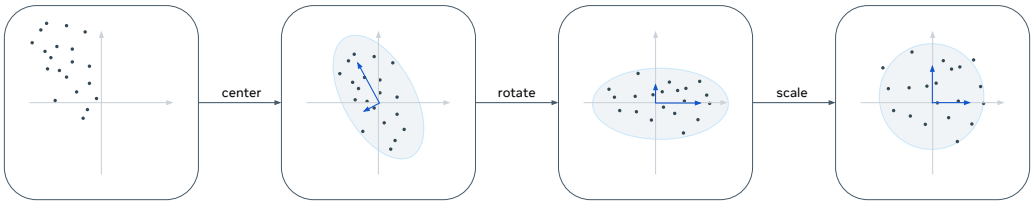


Figure 5: Illustration of PCA-whitening

The chosen set to perform the PCA on is a set X of features extracted from images with approximately the same distribution as the ones we want to watermark. Principal Components Analysis (PCA) computes the empirical mean $\mu = \frac{1}{m}\sum_{i=1}^{m} X_i$ and the empirical covariance matrix $\Sigma = \frac{1}{m}\sum_{i=1}^{m}(X_i - \bar{\mu})(X_i - \bar{\mu})^\intercal$ of the features. From there, the eigendecomposition of $\Sigma = U\Lambda U^\intercal$ allows to create U, an orthogonal matrix composed of the stacked eigenvectors and Λ, a diagonal matrix with the eigenvalues on the diagonal. The whitening transformation is then defined as

$$x \mapsto \Lambda^{-1/2}U^\intercal(x - \bar{\mu})$$

The new set of features output by this transformation have zero mean and identity covariance, therefore the newly generated distribution is isotropic. This transformation can be represented by a linear layer L added to the network N. The final transformation that is used as feature extractor is $\phi = L \circ N$.

3.3 Embedding the mark through data-augmentation and back-propagation

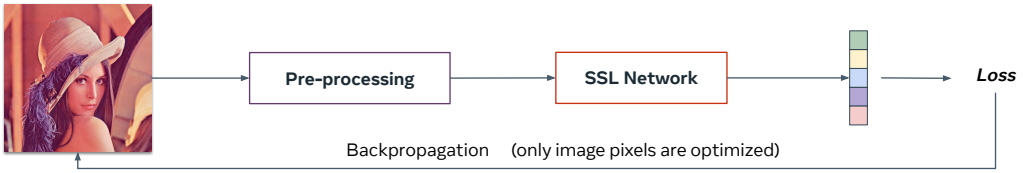


Figure 6: The embedding algorithm. At each iteration, the image is pre-processed to respond to quality constraints and transformed by a random data augmentation. The resulting image is fed to the feature extractor and the loss is computed. Image pixels are then optimized using gradient descent.

Marking process The embedding algorithm Emb aims to add a mark onto $\mathcal{I}_o$ and forge a visually similar image $\mathcal{I}_m$ such that $\phi(\mathcal{I}_m) \in \mathcal{D}$. We employ a technique originally used in the adversarial attacks literature [57, 30]. The optimization problem reads:

$$\min_{\mathcal{I};\, C(\mathcal{I},\mathcal{I}_o);\, \tilde{I}=Tr(I)} \mathcal{L}(\tilde{\mathcal{I}})$$

where $C(\mathcal{I}, \mathcal{I}_o)$ represents the visual constraints imposed to the watermarked image with regards to the original one, and $\tilde{\mathcal{I}}$ is an attacked version of $\mathcal{I}$ (for instance a rotated or cropped version).

Since the feature extractor is differentiable, this is achievable by gradient descent on the pixel values. At each time step, we feed the image $\mathcal{I}_t$ to the pre-processing step, that imposes constraints and attacks the image, and to the network that extract the features. We successively obtain $\tilde{\mathcal{I}}_t$ and $x_t = \phi(\tilde{\mathcal{I}}_t)$. Then, we compute the loss, back-propagate the computation of the gradients through the network and the pre-processing step, and update the pixel values with $\mathcal{I}_{t+1} = \mathcal{I}_t - \eta\nabla\mathcal{L}(I_t)$. In practice, we use the Adam optimizer to update the image's pixels.

Losses $\mathcal{L}$ consists of two terms:

- $\mathcal{L}_{\text{hypercone}}(x) = -(x^\intercal a)^2\rho(\theta) + \|x\|^2$ aims at pushing the feature $x = \phi(\tilde{\mathcal{I}})$ into the *hypercone* $\mathcal{D}$. It is the opposite of R, the objective function defined in (2), therefore minimizing the loss amounts to maximizing R which represents "how much x is into $\mathcal{D}$".
- $\mathcal{L}_{\text{img}}(\mathcal{I}) = \frac{1}{h\times w}\|\mathcal{I} - \mathcal{I}_o\|^2$ controls the *image distortion*. This loss could be replaced by other perceptual image losses such as LPIPS [72] to ensure high perceptual similarity instead of low mean squared error.

At the end:

$$\begin{aligned}\mathcal{L}(\mathcal{I}) &= \mathcal{L}_{\text{hypercone}}(\phi(\tilde{\mathcal{I}})) + \lambda_{img}\mathcal{L}_{\text{img}}(\mathcal{I}, \mathcal{I}_o) \\ &= -(\phi(\tilde{\mathcal{I}})^\intercal a)^2\rho(\theta) + \|\phi(\tilde{\mathcal{I}})\|^2 + \lambda_{img}\frac{1}{h\times w}\|\mathcal{I} - \mathcal{I}_o\|^2\end{aligned} \tag{4}$$

Pre-processing step On top of the marking algorithm used in [61] we add a pre-processing step whose role is twofold. It explicitly imposes constraints on the watermark image, and simulates transformations to allow the robustness of the watermarks.

The optimization process takes into account the constraints $C(\mathcal{I}, \mathcal{I}_o)$ by applying them to the image before sending it to ϕ. The first constraint ensures a fixed (rather low) Peak Signal to Noise Ratio (PSNR) defined as

$$\text{PSNR}(\mathcal{I}, \mathcal{K}) = \log_{10}\left(\frac{255^2}{MSE(\mathcal{I}, \mathcal{K})}\right) = \log_{10}\left(\frac{255^2}{\frac{1}{h\times w}\|\mathcal{I} - \mathcal{I}_o\|^2}\right)$$

To apply this constrain we compute the PSNR between the current image and the original one. If it is lower than the target PSNR, we rescale the difference $\delta = \mathcal{I} - \mathcal{I}_0$ so that the new image has the target PSNR. (If the source PSNR is larger, nothing happens).

To increase the perceptual similarity between the original and the watermarked image, we perform an SSIM heatmap attenuation. SSIM [64] is defined as:

$$\text{SSIM}(\mathcal{I}, \mathcal{K}) = \frac{(2\mu_\mathcal{I}\mu_\mathcal{K} + C_1)(2\sigma_{\mathcal{I}\mathcal{K}} + C_2)}{(\mu_\mathcal{I}^2 + \mu_\mathcal{I}^2 + C_1)(\sigma_\mathcal{I}^2 + \sigma_\mathcal{K}^2 + C_2)} \tag{5}$$

where $\mu_\mathcal{I}$ represents the average grey values of $\mathcal{I}$, $\sigma_\mathcal{I}$ represents its variance, $\sigma_{\mathcal{I}\mathcal{K}}$ represents the covariance between $\mathcal{I}$ and $\mathcal{K}$, and C_1 and C_2 are two constants which are used to prevent unstable results. To create the heatmap we compute the SSIM between $l \times l$ windows around each pixel of each color channel of the image, and the same windows of the original image. Then we apply the heatmap to $\delta = \mathcal{I} - \mathcal{I}_0$ as a Hadamard product ($\delta \leftarrow \text{SSIM}_{heatmap} \odot \delta$).

In the same way, we could project the image back into the l_∞ ball or round its pixels to integral values (even though it is not done in practice).

The pre-processing step also simulates transformations to encourage the watermark to be robust to it. At each step of the marking process, a transformation is chosen at random among Identity, Rotation, Crop, Blur and Resize and applied to the image with random parameters. Other augmentations could be applied, the only requirement is that it should be differentiable in order to back-propagate through it. Adding data augmentations at marking time significantly improves the performance of the method (See 6.1).

3.4 Final overview

To summarize, the major differences between [62] and our method are:

- self-supervision is used to pretrain the backbone models, rather than classification supervision
- differentiable data-augmentation is added at marking time, rather than just relying on the neural network generalization ability
- the PSNR clipping operation in the preprocessing step that allows to continue the optimization even though the target PSNR has been reached, instead of the stopping criterion

The watermarking algorithm is summarized in 1.

Algorithm 1 Zero-bit watermarking with the hypercone detector

Inputs: image: $\mathcal{I}$, carrier $a \in \mathbb{R}^d$, FPR, targeted PSNR
Embedding:
  Compute θ such that the hyperruff detector gives the right FPR
  $\mathcal{I}_m \leftarrow \mathcal{I}$
  For $i = 1, ..., n_{iter}$:
    Apply quality constraints (PSNR clipping, SSIM attenuation and pixel rounding) to $\mathcal{I}_m$
    Choose random transformation among (rotation, crop, resize, blur, identity) and parameters
    $\tilde{\mathcal{I}}_m \xleftarrow{attack} \mathcal{I}_m$
    $x \leftarrow Tr(\tilde{\mathcal{I}}_m)$
    $\mathcal{L} \leftarrow -(x^\intercal a)^2\rho(\theta) + \|x\|^2 + \lambda_{img}\frac{1}{h\times w}\|\mathcal{I}_m - \mathcal{I}\|^2$
    $\mathcal{I}_m \leftarrow \mathcal{I}_m + \eta \times \text{Adam}(\mathcal{L})$
  Return $\mathcal{I}_m$
Detecting:
  $x \leftarrow Tr(\mathcal{I}_m)$
  $R \leftarrow (x^\intercal a)^2\rho(\theta) - \|x\|^2$
  Return $R > 0$

4 Extension to multi-bit data hiding

In the data hiding setup, the detector is replaced by a decoder D. The objective is now to store and be able to retrieve a message under different image alterations. If $\mathcal{I}_m$ is a watermarked image containing the message m, then one should have $D(\tilde{\mathcal{I}}_m) = m$, where $\tilde{\mathcal{I}}_m$ is an altered version of $\mathcal{I}_m$. We propose 3 ways of extending the previous 0-bit watermarking method to multi-bit messages. The results reported in Multi-bit data hiding use the extension presented in Multi-bit watermarking in the hyperruff. Experimental results on the 2 other extensions showed less promising results, which is why they are not presented from an experimental point of view in the rest of the report (although they may be explored in future works).

Preliminary: the hyperruff detector In Vukotić et al.'s paper, watermarking is done by randomly choosing some cone axis $a \in \mathbb{R}^d$ of unit norm and maximizing the objective function $R_a^\theta(x) = (x^\intercal a)^2\rho(\theta) - \|x\|^2$ where x is the extracted feature and θ controls the angle of the cone. Detection is carried out by checking if $R(x) > 0$ (see 3.1 for more details). Multiplying both sides of 2 by $\cos^2(\theta)$ gives an interpretation of $R(x)$ in terms of the norm of x after random projection onto the (1-dimensional) subspace spanned by a:

$$\cos^2(\theta)R(x) = (x^\intercal a)^2 - \|x\|^2\cos^2(\theta) \tag{6}$$

Hence $R(x) > 0$ if and only if $\|\Pi_a(x)\|^2/\|x\|^2 = (x^\intercal a)^2/\|x\|^2 > \cos^2(\theta)$, that is, the projection of x onto the subspace spanned by a retains at least a $\cos(\theta)$ portion of x's magnitude.

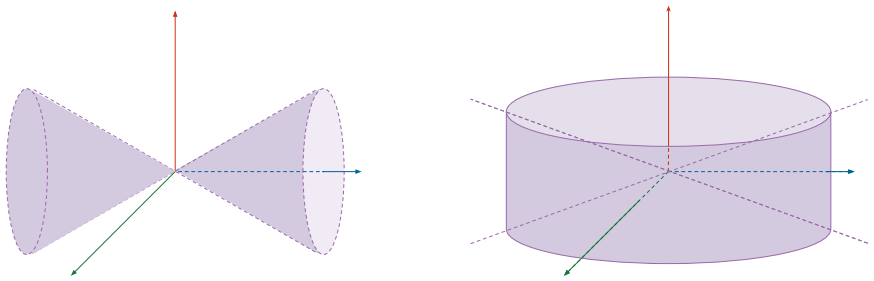


Figure 7: Illustration of the hypercone detection region (left), and the 2-dimensional hyperruff detection region (right) for $d = 3$. In this case, the 2-dimensional hyperruff spanned by a_1, a_2 is the complementary in $\mathbb{R}^3$ of the hypercone supported by a_3.

We now consider projection onto a k-dimensional subspace of $\mathbb{R}^d$, with $k \leq d$. We choose a random orthonormal set A of vectors $a_1, ..., a_k$ and considering the projection Π_A. The detection process can be generalized accordingly by using the objective $R(x) = \|\Pi_A(x)\|^2\rho(\theta) - \|x\|^2$. Since, the norm reads $\|\Pi_a(x)\|^2 = \sum_{i=1}^k (x^\intercal a_i)^2$ by Pythagora's theorem, the marking objective becomes:

$$R(x) = \sum_{i=1}^{k}(x^\intercal a_i)^2\rho(\theta) - \|x\|^2 \tag{7}$$

The previous hypercone detector method corresponds to choosing $k = 1$. The detector is called "k-dimensional ruff" (term coined by Furon) because the detection region looks like a "ruff" collar when $k = 2$ and $d = 3$ (see Figure 7).

The FPR rate of the hypercone detector (see 3.1) can be extended in the case of the hyperruff detector:

$$\text{FPR}_\theta = 1 - I_{\cos^2(\theta)}\left(\frac{k}{2}, \frac{d-k}{2}\right) \tag{8}$$

where $I_\tau(\alpha, \beta)$ denotes the regularized Beta incomplete function (the full proof is available in the Appendix). In the same way as the hypercone detector, this allows to ensure a given FPR by carefully choosing θ.

The hyperruff detector allows to check the presence of the mark on a subspace spanned by more than one carrier. Like for the hypercone detector, this space is also sufficiently large for an image's feature to be detected even if it has undergone some transformations. That will be useful to preserve the notion of detection in the scope of some of the multi-bit watermarking's methods presented bellow. We give an overview of these methods in Figure 8

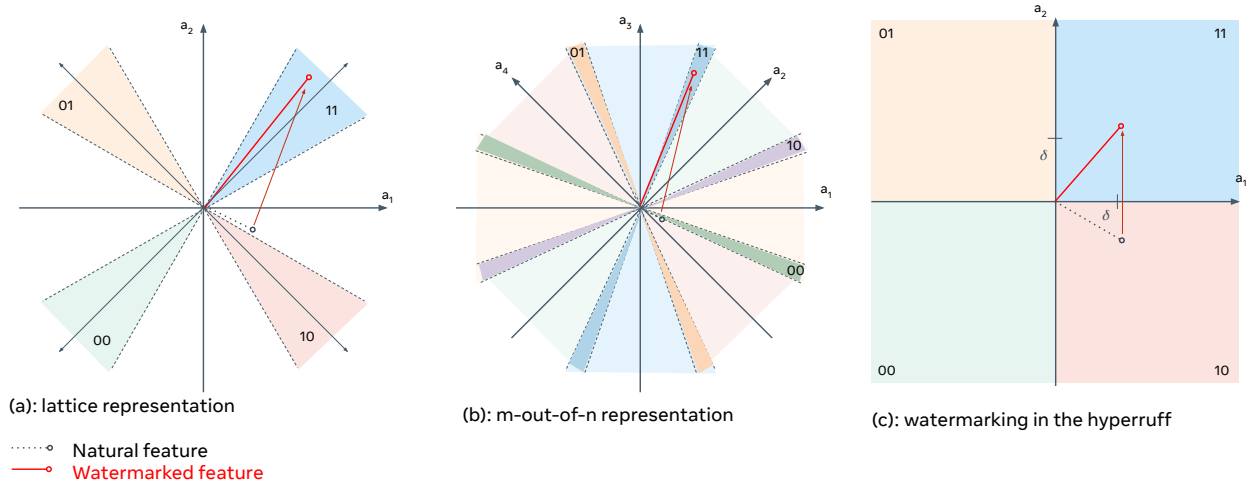


Figure 8: Multi-bit extension for features in $\mathbb{R}^2$ and $k = 2$ bits in the message. We propose 3 extensions. In (a) the message is conveyed by watermarking on a well-chosen carrier, in (b), by watermarking on a set of carriers, in (c) by pushing the feature along several carriers.

4.1 Lattice coding: marking on a well-chosen carrier

Assume that we are able to mark images onto a given carrier a with the zero-bit watermarking method of 3. To encode k bits, choose 2^k unit norm vectors $\mathcal{V} = \{v_1, ..., v_{2^k}\}$ of $\mathbb{R}^d$ then choose to watermark the image $\mathcal{I}$ on one of these carriers. To decode an image $\mathcal{I}_m$ with extracted feature x, we search for the carrier v_i that is closest to x, that is

$$\min_{i=1,...,2^k} \|x - v_i\| \iff \max_{i=1,...,2^k} x^\intercal v_i \tag{9}$$

To create these lattice vectors, choose a random orthonormal set A of vectors $a_1, ..., a_k \in \mathbb{R}^d$. Then, the set is chosen as follows:

$$\mathcal{V} = \{m_1 a_1 + ... + m_k a_k \mid m_1, .., m_k = \pm 1\}$$

which verifies that $|\mathcal{V}| = 2^k$.

The set $\mathcal{V}$ presents interesting properties, namely that the vectors are well spread-out in the feature space and that the optimization problem 9 is easy to solve. Indeed,

$$\max_{v \in \mathcal{V}} x^\intercal v \iff \max_{\tilde{m} \in \{-1,1\}^k} x^\intercal A \tilde{m}$$

and

$$x^\intercal A \tilde{m} = x^\intercal \left(\sum_{i=1}^k \tilde{m}_i a_i\right) = \sum_{i=1}^k \tilde{m}_i (x^\intercal a_i)$$

Hence, this problem is solved by looking at the sign $s_i = \pm 1$ of the projection of x onto each a_i, and choose $\tilde{m}_i = s_i$.

Given a message m and an image $\mathcal{I}$:

- To encode m, we generate $a = \sum_{i=1}^{k} m_i a_i$ then we watermark $\mathcal{I}$ using a as the carrier, and end up with a watermarked image $\mathcal{I}_m$
- To decode $\mathcal{I}_m$, we simply extract its feature x and get the signs of the projections of x onto the a_i.

The previous message decoding step does not imply any detection, which means that every image, even not watermarked ones, would give a message. One may want to keep the detection information brought by the zero-bit watermarking embedding. However, since the detection is oblivious to the original message that was embedded into the watermarked image $\mathcal{I}_m$, the carrier a is unknown. To resolve the issue, we use the hyperruff detector instead, which has knowledge only of the orthonormal set A. Since the watermarking process pushes the feature into an hypercone generated from the a_i, it is also pushes into the hyperruff generated by the a_i.

The corresponding pseudo-code for the algorithm to watermark a message into an image is given in algorithm 2.

Algorithm 2 Multi-bit watermarking with lattice coding

Inputs: image: $\mathcal{I}$, message m, k orthonormal vectors of $\mathbb{R}^d$: $a_1, ..., a_k$, FPR
Encoding:
$a \leftarrow \sum_{i=1}^{k} m_i a_i$
Compute θ such that the hyperruff detector gives the right FPR
Do 0-bit watermarking on $\mathcal{I}$ with carrier a and angle θ
Return $\mathcal{I}_m$
Decoding:
$x \leftarrow Tr(\mathcal{I}_m)$
$R(x) \leftarrow \sum_{i=1}^{k} (x^\intercal a_i)^2 \rho(\theta) - \|x\|^2$
If $R > 0$: watermarked image detected
$m_i \leftarrow (\text{sign}(x^\intercal a_i)+1)/2$, $i = 1, ..., k$
Return m

4.2 m-out-of-n coding: marking on multiple carriers

So far we only considered watermarking on a single carrier a. Now assume that we are able to add up to k marks on the image (it is hard in practice since the hypercones have tiny intersection). Another way to communicate a message is to consider a set of n vectors $\mathcal{A} = a_1, ..., a_n$ (not necessarily orthogonal) and to mark the image on k vectors among $\mathcal{A}$. The corresponding codes are *m-out-of-n codes* [53], called this way because only k bits are activated (set to 1) while all the $n - k$ others are set to 0. They are especially used in the cases of asymmetric binary channels, in which errors of one type, either $1 \to 0$ or $0 \to 1$, occurs much more often than the other (for instance in optical communication where photons may fail to be detected, i.e., $1 \to 0$ crossovers are allowed, but the creation of spurious photons ($0 \to 1$) is impossible). This is our case since we can ensure that false positives occur with small probability.

They are a type of constant weight codes, i.e. codes such that the number of positive bits $k = \sum_{i=1}^{n} m_i$ is constant. For a given parameter choice (n, k), the number of binary codes with k ones is $N = \binom{n}{k}$ and therefore consists of only a subset of the 2^n possible binary codes. The Hamming distance between two constant-weight codes is always even, with a minimum $d_{min} = 2$. Given a codebook $m_1, ..., m_N$ and a code from the codebook that has been corrupted as $\hat{m}$, it can be recovered by selecting the nearest code from the codebook in Hamming distance:

$$m^\star = \text{argmin}_{m \in \{m_1, ..., m_N\}} \sum_{j=1}^{n} |\hat{m}_j - m_j|$$

Several constructions of constant-weight codes have been proposed in the coding literature [56, 45] allowing to detect and correct errors occurring in these cases. The upper bound on the theoretical numbers of bits for several values of k and n are given in Table 1.

To mark on multiple carriers at the same time, the loss function from equation 4 needs to be adapted. Let x be the extracted feature and $b_1, ..., b_k$ the k carriers to mark on. The hypercone loss was defined as

Table 1: Maximum number of bits: $\log_2 \binom{n}{k}$ for a k-out-of-n code

k =	1	2	3	4	5
n = 1024	10	18	27	35	43
2048	11	20	30	39	48
4096	12	22	33	43	53

$\mathcal{L}_a(x) = -R_a(x) = -(x^\intercal a)^2\rho(\theta) + \|x\|^2$. The updated hypercone loss is

$$\mathcal{L}(x) = \max_{i=1,...,k} \mathcal{L}_{b_i} + \sum_{i=1,...,k} \mathcal{L}_{b_i} = -\min_{i=1,...,k} R_{b_i} - \sum_{i=1,...,k} R_{b_i} \tag{10}$$

One could alternatively use $\mathcal{L}(x) = \text{softmax}_{i=1,...,k}\mathcal{L}_{b_i}$. The goal of these losses is to optimize all the carriers at each iteration and to make sure that all of them are activated enough.

The corresponding algorithm to watermark a message into an image is:

Algorithm 3 Multi-bit watermarking with k-out-of-n codes

Inputs: image: $\mathcal{I}$, message m, n orthonormal vectors of $\mathbb{R}^d$: $a_1, ..., a_n$, codebook $M = \{m_1, ..., m_N\}$
(K: number of bits in m, $N = 2^K$: number of codewords in the codebook)
Encoding:
Get $m_i \in M$ corresponding to m
For each $j = 1, ..., n$ such that $m_i^{(j)} = 1$:
Add a_j to the set of active carriers
Watermark $\mathcal{I}$ on the set of active carriers
Return $\mathcal{I}_m$
Decoding:
$x \leftarrow Tr(\mathcal{I}_m)$
$m \leftarrow$ 00..0 (length n)
For each $j = 1, ..., n$:
$R \leftarrow (x^\intercal a_j)^2\rho(\theta) - \|x\|^2$
If $R > 0$: $m^{(j)} \leftarrow 1$
Find $m^\star$ minimizing the Hamming distance in the set M
Return the corresponding K-bit message

4.3 Multi-bit watermarking in the hyperruff

Given an image with feature x and a carrier a, the zero-bit watermark encodes the binary message "Is x marked?" into the magnitude of the projection Π_a. A natural extension to 1-bit watermarking is to use the sign of the projection $\Pi_a(x) = x^\intercal a$ as the message, i.e., $D(x) = (\text{sign}(x^\intercal a) + 1)/2$. Instead of taking only one vector, one can also choose a random orthonormal set A of vectors $a_1, ..., a_k \in \mathbb{R}^d$. Like in the case of the lattice coding, the decoding is done by computing the sign of the k-dimensional vector $x^\intercal A$, i.e, $D(x) = (\text{sign}(x^\intercal A) + 1)/2 \in \{0, 1\}^k$.

The hypercone loss from equation 4 needs to be updated since the goal is not to push into a hypercone anymore, but rather to push towards some directions of the feature space. We split this loss into 2 terms: $\mathcal{L}_{hypercone} \rightarrow \lambda_{msg}\mathcal{L}_{msg} + \lambda_R\mathcal{L}_R$:

- $\mathcal{L}_{msg}$ aims at encoding the bits onto the feature's projection onto the carriers. Let the message be a k-bit vector m. $\mathcal{L}_{msg}$ is defined as a Hinge loss of margin δ on the projections:

$$\mathcal{L}_{msg}(x) = \frac{1}{k}\sum_{i=1}^{k} \max(0, \delta - (x^\intercal a_i).(2m_i - 1)) \tag{11}$$

  Say $m_1 = 1$, the corresponding component of the sum would be $\max(0, \delta - x^\intercal a_1)$. This component is 0 if $x^\intercal a_1 > \delta$, i.e. if the projection is already pushed in the right direction. On the contrary, the loss is non negative as soon as $x^\intercal a_1 < \delta$ and increases linearly, pushing x towards a_1. If $m_1 = 0$, x is now pushed towards $-a_1$ as soon as $x^\intercal a_1 > -\delta$.

- $\mathcal{L}_R$ allows to keep the notion of detection (hence the guarantees on the FPR) by pushing the feature into the hyperruff spanned by A. It is defined as

$$\mathcal{L}_R(x) = -\sum_{i=1}^{k} \text{sign}(m_i)|x^\intercal a_i|(x^\intercal a_i)\rho(\theta) + \|x\|^2 \tag{12}$$

  where θ is computed so that $FPR_\theta = p$ (with p a tiny value). One can note that $\mathcal{L}_R$ is not exactly the inverse of the acceptance function of the hyperruff (see eq. 7). This allows to take into account the fact that the feature should be pushed towards either $\pm a_{1,...,k}$ and not only be co-linear with them.

Algorithm 4 Multi-bit watermarking in the hyperruff

Inputs: image: $\mathcal{I}$, message m, k orthonormal vectors of $\mathbb{R}^d$: $a_1, ..., a_k$, FPR
Encoding:
  Compute θ such that the hyperruff detector gives the right FPR
  Do multi-bit watermarking on $\mathcal{I}$ with carriers $a_1, ..., a_k$, angle θ and the updated losses
  Return $\mathcal{I}_m$
Decoding:
  $x \leftarrow Tr(\mathcal{I}_m)$
  $R(x) \leftarrow \sum_{i=1}^{k}(x^\intercal a_i)^2\rho(\theta) - \|x\|^2$
  If $R > 0$: watermarked image detected
    $m_i \leftarrow (\text{sign}(x^\intercal a_i)+1)/2$, $i = 1, ..., k$
    Return m

Given a k-bit message m, the bit m_i is decoded from the dot product between the extracted feature x and the i-th carrier a_i. We can assimilate this dot product as a Gaussian channel [14]. This link between the dot products and the Gaussian channel is discussed in Channel modeling: Projections onto the carriers as parallel Gaussian channel.

5 Experiments & results

5.1 Experimental setup

Data In all the experiments, unless specified otherwise, we evaluate our method on 1000 random images of the YFCC100M (Yahoo-Flickr Creative Commons 100 Million) dataset [59]. We also use a larger dataset of 100K images from the same dataset to perform the PCA-whitening operation (see Normalization layer). The dataset was not designed for object detection, segmentation, captioning, etc. but rather for the diversity of its content as it includes complex real world scenes.

Another parameter of the utmost importance is the size of the images, since bigger images should allow more *capacity*. The size distribution of the images in the dataset is presented in Figure 9.

To allow a fair comparison with the work of Vukotić et al. we also evaluate our zero-bit watermarking method on the test dataset P of the CLIC challenge 2018 [1]. The dataset is composed of 118 professional high-resolution images collected to be representative for images commonly used in the everyday life. As before, we report the size distribution of the images in Figure 10

Previous deep hiding methods [74][43][2] usually use the COCO dataset [42] resized to 128×128 resolution as benchmark. Therefore we also evaluated our method on 1000 images from the COCO dataset, both in their original resolution (approximately the same distribution as YFCC) and resized to 128×128.

Backbone pre-training We use the ResNet-50 [32] architecture as backbone model. It is trained on the ILSVRC2012 dataset [17] without labels, using self-supervised learning with DINO [12]. We use the default parameters given by the authors and train the networks over 200 epochs, distributed over 16 GPUs. We refer the reader to the original paper for further details. The sole difference lies in data-augmentations applied to local and global crops since we add a rotation of amplitude chosen at random between ± 90 degrees. It must be noted that the data augmentation used in DINO contains color jitter (contrast, brightness, hue, saturation) and Gaussian blur.

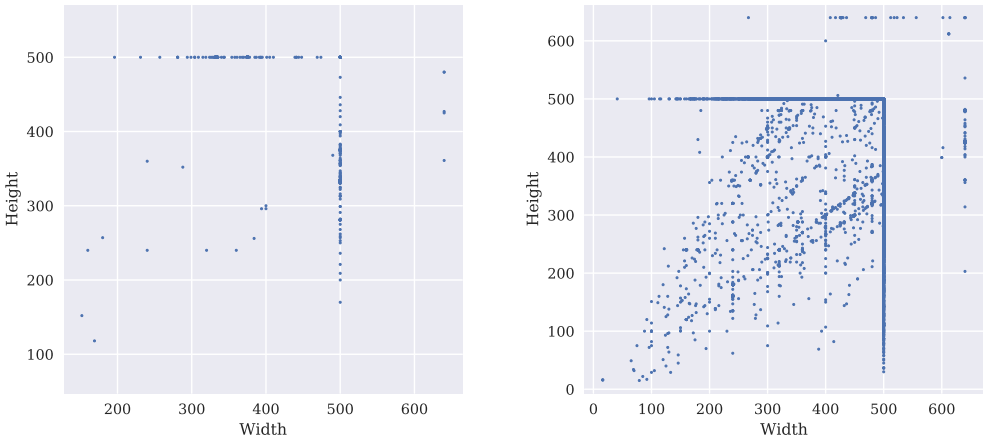


Figure 9: Distribution of the width and height of the images of the YFCC dataset. Left: 1K images used for evaluation, Right: 100K images used for the PCA-whitening operation (3.2)

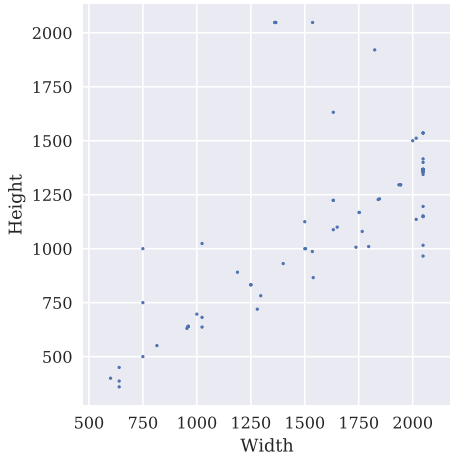


Figure 10: Distribution of the width and height of the images of the CLIC dataset.

In some cases in the rest of the report, we use networks trained on classification on ImageNet. In this case, we use the default pretrained models of the torchvision library [46], or train new networks with the code used to generate these pre-trained models.

Implementation details of the watermarking process First, given a fixed FPR p, θ is computed by finding a root of the function $\theta \mapsto \mathrm{FPR}_\theta - p$ in the interval $[0, \pi/2]$ using Brent's method [8]. FPR_θ can be defined either via Equation (3) or Equation (8).

For the watermarking process that embeds the mark onto $\mathcal{I}$ and generates $\mathcal{I}_m$, we use the Adam optimizer [38] with learning rate 0.01 and perform 100 gradient descent iterations. At each time step, the preprocessing step clips the PSNR to a target one and perform SSIM attenuation, see Equation (5). SSIM heatmaps are computed with $C_1 = 0.01^2$, $C_2 = 0.03^2$ and from 17×17 windows of the $C = 3$ image's channels, then summed-up and clamped to be non negative, which generates a single heatmap by image.

At each iteration, a transformation is chosen uniformly between Identity, Rotation, Crop, Resize and Blur. Unless specified otherwise, the angle a of the rotation is sampled using a Von Mises distribution with $\mu = 0$, $\kappa = 1$ and multiplied by a_{max}/π where $a_{max} = 90$ degrees (this allows to sample more often smaller rotations which are much more frequent in practice). The crop and resize scales are chosen uniformly in $[0.2, 1.0]$. Besides, the crop is not necessarily squared: its aspect ratio is also randomly chosen between $3/4$ and $4/3$. For blur, the kernel size is randomly selected from all the odd numbers between 1 and 15. Finally, the image is flipped horizontally with probability 0.5.

For zero-bit watermarking, the weight λ_{img}, that controls the importance of the image loss, is set to 1. For

multi-bit watermarking λ_{img} is kept to 1, λ_R that weights the detection loss is set to either 1 or to 0 and λ_{msg} that weights the message loss is set to 5e4. In the case $\lambda_R = 0$, we are only interested in the bit and word accuracy (see section 5.1).

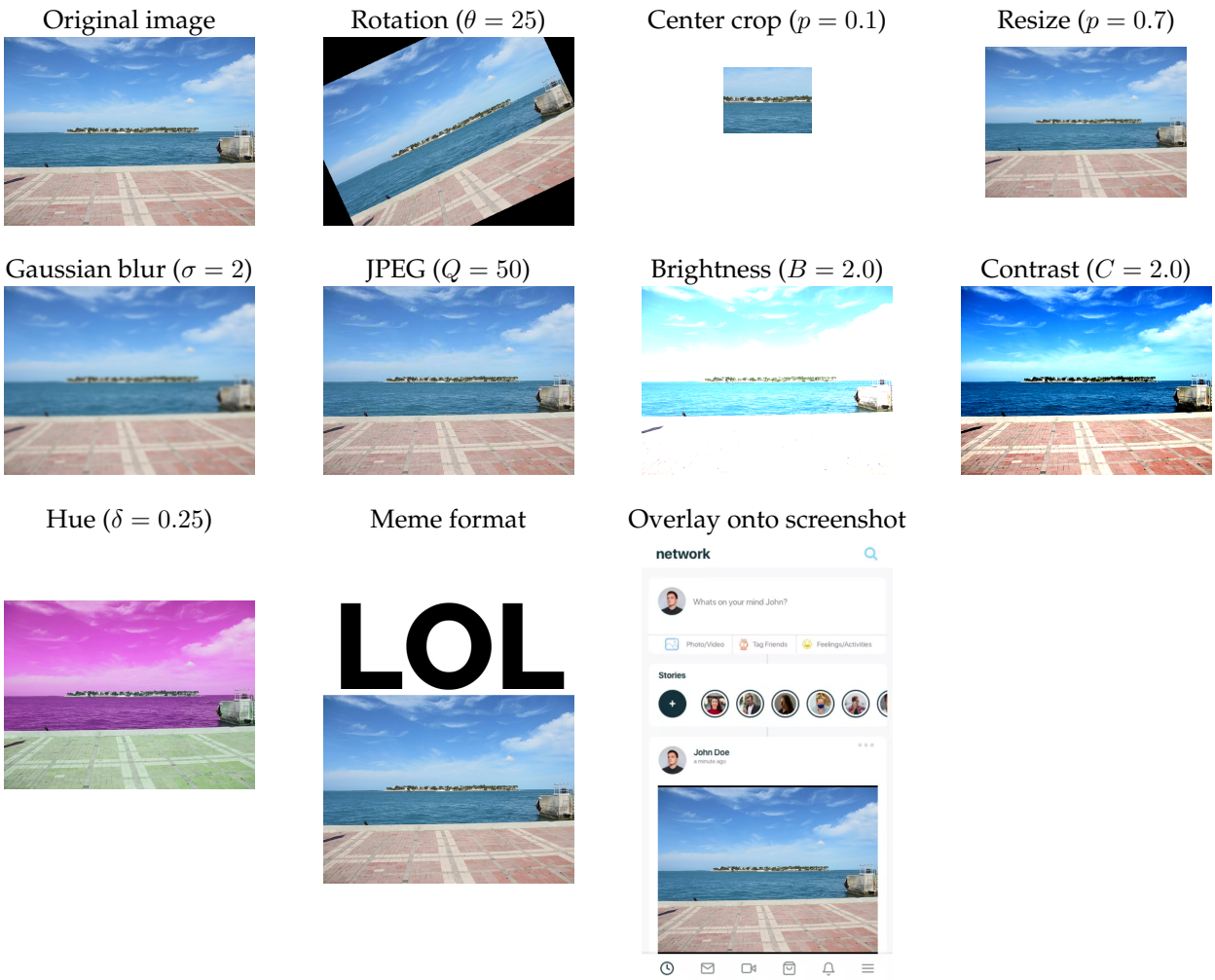


Figure 11: Illustration of the transformations applied at detection time to evaluate the robustness of the watermarks.

Evaluated transformations We evaluate robustness of the method with regards to a broad diversity of transformations (that simulate what happens to an image transmitted between 2 users). We consider:

- *Rotation* of angle θ that rotate the image and keep the rotated image to the same size as the original image by filling with black
- *Gaussian blur* with parameter σ (or equivalently kernel size)
  In the torchvision library, $\sigma = 0.3 * ((\text{kernel\_size} - 1) * 0.5 - 1) + 0.8$.
- *Center crop* that keeps a percentage p of the image.
  We use the center crop transformation and not random crops to have a fixed evaluation of the influence of the parameter p. Since crops are chosen randomly during the watermarking process, our method has no reason to be especially worse if we chose random crops instead of centered ones.
- *Resize* that resizes the image such that the ratio between the area of the resized image and the one of the original is p.
  It can be noted that the Resize transformation is not seen as such during pretraining, but incorporated in the resized crop of local crops that happen in DINO. For instance, the transformation Resize(0.1) almost never happens at pretraining, since the average resolution of an ImageNet image is 469×387 and that local crops are generated from cropping 60% to 95% of the image before resizing it to 96×96.
- *JPEG* compression of quality factor Q

- *Brightness* with parameter B: 0 gives a black image, 1 gives the original image while 2 increases the brightness by a factor of 2
- *Contrast* with parameter C: 0 gives a solid gray image, 1 gives the original image while 2 increases the contrast by a factor of 2
- *Hue* with parameter $\delta \in [-0.5, 0.5]$ that controls the amount of shift in the hue channel
- *"Meme format"* and *"Overlay on screenshot"* augmentations, from the AugLy augmentation library [5]

All transformations but the AugLy ones are controlled by a strength factor (θ, p, $kernel\_size$, etc.). They cover both spatial transformations (crops, rotation, etc.), pixel-value transformations (contrast, hue, etc.) and "everyday life" transformations with the AugLy augmentations. They also cover the harder cases where the decoding algorithm has no information about where the tampering happens nor original size information. Some transformations are seen both at marking time and self-supervised pretraining time (rotation, Gaussian blur), some only at training time (contrast, brightness, hue) and others neither at marking nor at training time (JPEG, meme format, screenshot).

In the experiments, the transformations presented above are applied to watermarked images before detecting or decoding them. Figure 11 illustrates all these augmentations.

Metrics In the zero-bit watermarking setup, we are mostly interested in:

- the False Positive Rate (FPR) defined as FP/N (where FP is the number of false positive, i.e. the number of natural images that are detected, and N the number of natural images). It is the proportion of images that are detected while not watermarked. The FPR is fixed beforehand by the user via the θ parameter, see Equation (3). FPR is related with the scale on which the algorithm must operate. Typically, to process the billion images shared online everyday, the FPR should be lower than 10^-9, otherwise the number of falsely flagged images would be too high.
- the True Positive Rate (TPR) defined as TP/P (where TP is the number of true positive, i.e. the number of watermarked images that are detected, and P the number of watermarked images). It is the proportion of accurately detected watermarks.

Moreover, the detection can also be evaluated with regards to its p-value. Indeed, using Equation 3 or Equation 8 we can retrieve the probability p for a natural feature to have higher correlation with the carrier than the watermarked feature. It means that if we were drawing $O(1/p)$ random secret carriers, on expectation, one of them would give a normalized correlation greater or equal to the original one.

In the multi-bit watermarking setup, we are interested in the number of bits that are accurately decoded. This can be evaluated thanks to the bit accuracy of the decoded bit-strings, defined as the ratio of the Hamming distance between the decoded message and the original one, divided by the number of bits of the message: $\frac{\sum_{i=1}^{k} XOR(m_{orig}, m_{decoded})}{k}$. A random decoding has a bit accuracy of 0.5. Bit error rate (BER) can alternatively be used and is defined as BER $= 1 -$ bit accuracy. These metrics are the most commonly used in the deep data hiding literature.

Previous metrics evaluate the decoding accuracy bit-wise, while communication theory also considers bit-strings as a whole. Therefore, we also consider the word accuracy that is the proportion of words that are perfectly decoded in the set of images. The word accuracy is lower or equal to the bit accuracy, since all the bits must be correct for a word to be counted as accurate.

To evaluate the quality of the watermarked images we use the Peak Signal to Noise Ratio (PSNR) defined as $\text{PSNR}(\mathcal{I}, \mathcal{K}) = \log_{10}\left(\frac{255^2}{MSE(\mathcal{I},\mathcal{K})}\right) = \log_{10}\left(\frac{255^2}{\frac{1}{h\times w}\|\mathcal{I}-\mathcal{I}_o\|^2}\right)$. Like the FPR, the target PSNR is fixed beforehand by the user and the method encodes the watermarks such that generated images have PSNR higher than the target (therefore the observed PSNR can sometimes be higher than the target).

In summary, we fix the reliability of the decoding (via the FPR) and the amount of perturbation of the image (via the PSNR). Then we measure how well the message is recovered via the bit or word accuracy.

5.2 Zero-bit watermarking

Quantitative results We evaluate our method on 1000 images from the YFCC dataset, with PSNR targeted at 40 dB, and FPR set at 10^{-6}. The performance against the strength of some transformations is described in

Figure 12 and with more precise figures in Table 2. We observe that our method allows to create watermarks robust to a wide variety of transforms, while being faithful to the original images and having a low false positive rate. The FPR was checked over 100K natural images from the YFCC dataset, on which no images were flagged, and on the ImageNet dataset on which the detection had an FPR of 6.99e-07 (close to the theoretical value of 1e-6).

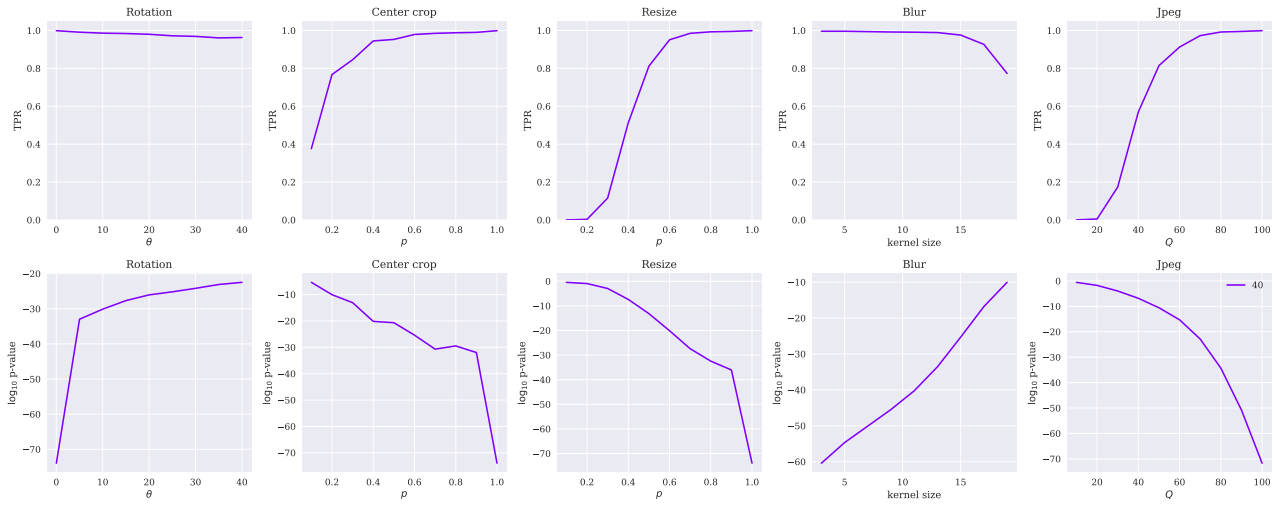


Figure 12: Top row: TPR over 1000 images of the YFCC dataset, against the strength of the transformations applied before detection. Bottom row: average $\log_{10}$ p-value of the detection (see 5.1). The target PSNR is set to 40, i.e. all the watermarked images have PNSR ≥ 40.

Table 2: TPR of the detection of watermarked images over a broad type of transformations applied at detection. The results are averaged over 1000 images of the YFCC dataset. The images are watermarked by our zero-bit watermarking method with FPR set at 10^{-6}.

Transformations	PSNR= 32	36	40	42
Identity	1.000	1.000	0.999	0.998
Rotation (25)	0.997	0.994	0.972	0.942
Crop (0.5)	0.990	0.989	0.953	0.906
Crop (0.1)	0.669	0.583	0.376	0.275
Resize (0.7)	0.994	0.992	0.985	0.967
Gaussian Blur ($\sigma = 2$)	0.996	0.994	0.991	0.990
JPEG (50)	0.944	0.931	0.814	0.613
Brightness (2.0)	0.969	0.963	0.945	0.921
Contrast (2.0)	0.977	0.977	0.964	0.935
Hue (0.25)	0.998	0.998	0.996	0.991
Meme format	0.996	0.995	0.988	0.964
Screenshot	0.921	0.880	0.759	0.678

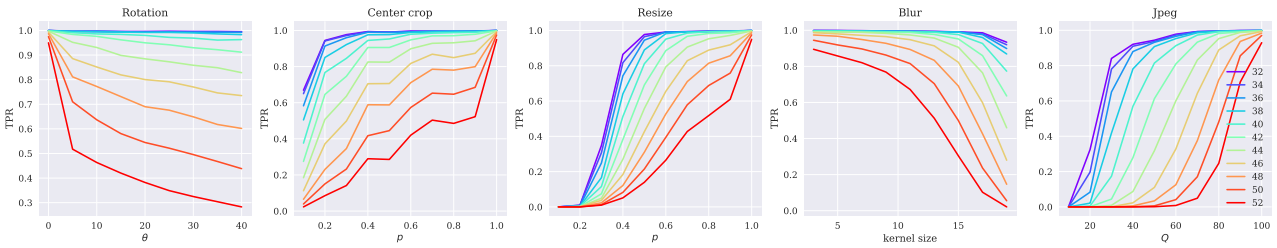


Figure 13: Illustration of the Quality/Robustness trade-off when watermarking at FPR= 10^{-6}. From blue to red the target PSNR decreases and better detection accuracy is obtained for lower PSNR.

Trade-offs The choice of the target PSNR is the result of a trade-off between the quality of the images and the robustness of their watermarks. Indeed, lower quality images tend to encode the watermarks more robustly

at the cost of their invisibility. Table 2 and Figure 13 illustrate this trade-off. To see the quality of watermarked images corresponding to certain PSNR values, we refer the reader to the following paragraph and to Figure 35 in the appendix.

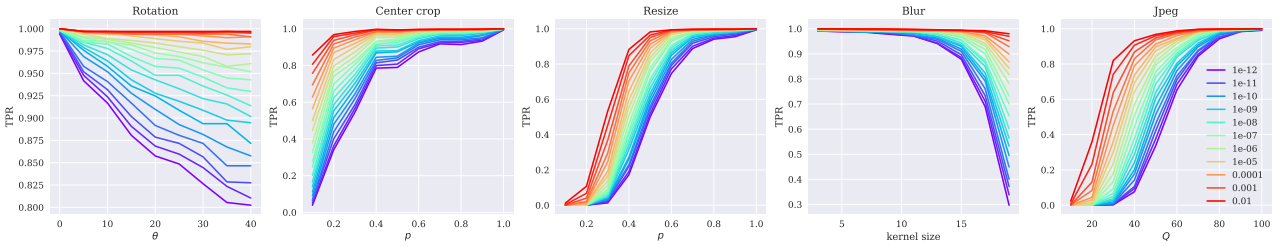


Figure 14: Illustration of the TPR/FPR trade-off when watermarking at PSNR=40. From blue to red the FPR decreases and better TPR is obtained for lower FPR.

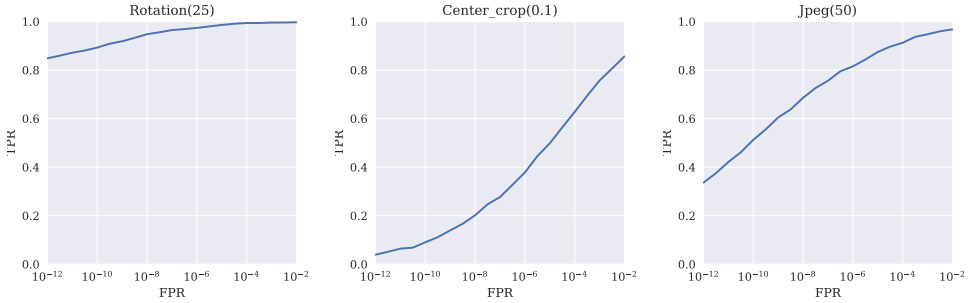


Figure 15: Receiver Operating Characteristic (ROC curve) of the detector against several transformations. (The watermarks were added at PSNR=40.)

In the same way, the choice of the FPR is subject to a trade-off between precision and recall. Lower FPR implies that the method accurately detects more severe transformations, at the cost of detecting more false positives. This is illustrated in figures 14 and 15. In large scale applications, it may be useful to operate at high-FPR low-TPR to avoid the need of human intervention. This is why FPR was set at 10^{-6} in previous experiments.

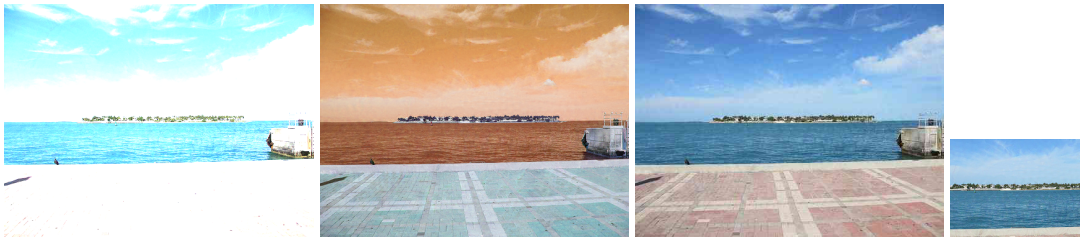
Qualitative results Figure 16 presents some images watermarked at PSNR 40dB, as well as their pixel-wise difference with regards to the original images. At this PSNR value, the watermark is almost invisible even to the trained eye. Besides, it seems to be added in the regions of high contrast of the images. This is mostly due to the perceptual SSIM filter that is applied during the marking process. Figure 35 in the Appendix presents the same images watermarked with different PSNR values.

Figure 17 presents transformations for which watermarked images are successfully detected, as well as ones that were too severe for the detector. Overall, the method is able to detect watermarks on a broad type of transformations, and on very severe ones.

Comparison with [62] We now compare our method with the baseline method of Vukotić et al. on the test dataset used for the CLIC challenge. Even though their images are extremely similar in nature, it must be noted that the dataset used in our experiments is not the exact same one as the one used by the authors, since they accessed it in June 2018. In this setup, the FPR is set at 10^{-3} and PSNR must be ≥ 42.

The results for the detection accuracy against the strength of some transformations are presented in Figure 18 and in Table 3. Overall, our method gives similar results between the YFCC dataset and the CLIC dataset. The center crop transformation gives better detection performance on the CLIC dataset. One explanation is that its images have higher resolutions (hence more pixels can be used to convey the mark). We observe a strong improvement over the baseline Vukotić et al. [62], especially for great rotations, crops and Gaussian blur where our method gives almost perfect detection. Since JPEG transformation is not used neither at pre-training time nor at marking time, it has no reason to behave better with our method than in [62]. That explains the similarity between both performances for this transformation.

Figure 16: Example of watermarked images at PSNR 40 (Top row). The second row gives the pixel-wise l_1 distance between the images and their watermarked counterpart.



(a) Successfully detected images: Brightness(2), Hue(0.5), JPEG(30), Crop(0.2)

(b) Not detected images: Crop(0.1), JPEG(20), Resize(0.3)

Figure 17: Example of transformed watermarked images as seen by the detector. The target PSNR with regards to the original image is 40dB.

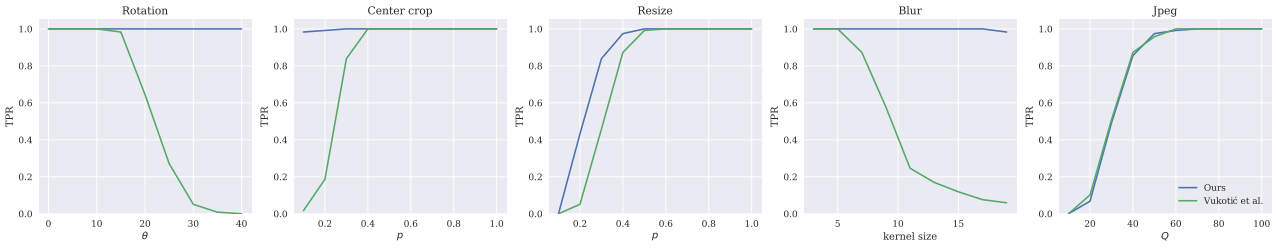


Figure 18: TPR over CLIC test dataset (118 images) against the strength of the transformations applied before detection, for our method and our implementation of the baseline Vukotić et al. [61]. The target PSNR is set to 42, i.e. all the watermarked images have PNSR ≥ 42, and the FPR is set at 10^{-3}.

Remark: comparison with multi-bit methods used as 0-bit watermarking Multi-bit methods can also be used as 0-bit watermarking by using some bits of the message as an identifier. Say for instance that the first k

Table 3: TPR of the detection of watermarked images over a broad type of transformations applied at detection. The results are averaged over 118 images of the CLIC dataset. The images are watermarked with FPR set at 10^{-3} and PSNR $\geq$ 42dB. ($\star$) best results given in [62] (obtained with a VGG-19, harder attacks during training and RMAC aggregation), ($\star\star$) our implementation of [62] (obtained with VGG-19 and default classification pretraining).

Transformations	Ours	[62] ($\star$)	[62] ($\star\star$)
Identity	1.00	1.0	1.00
Rotation (25)	**1.00**	≈ 0.3	0.27
Crop (0.5)	1.00	≈ 0.1	1.00
Crop (0.1)	**0.98**	≈ 0.0	0.02
Resize (0.7)	1.00	-	1.00
Gaussian Blur ($\sigma = 2$)	**1.00**	-	0.25
JPEG (50)	0.97	≈ 1.0	0.96
Brightness (2.0)	0.96	-	0.99
Contrast (2.0)	1.00	-	1.00
Hue (0.25)	1.00	-	1.00
Meme format	1.00	-	0.98
Screenshot	**0.97**	-	0.86

bits are used as an identifier. One could say that an image is watermarked if the first k bits are equal to a fixed message (for instance full of ones, without loss of generality). The FPR in this case would be the probability that the first k bits are one for a natural image. Under the assumption that 0 and 1 are equiprobably output by the decoder, the FPR becomes $(1/2)^k$. For $k = 20$, FPR $\approx 10^{-6}$ and for $k = 30$, FPR $\approx 10^{-9}$.

Now, say the multi-bit hiding method has p bit accuracy on an attack. Denote by $m = m_1 m_2 ... m_k$ the original message encoded in an image and by $\tilde{m}$ the decoded message. The probability for the attacked image to be detected would be

$$\text{TPR} = \mathbb{P}\left(\bigcap_{i=1}^{k} \tilde{m}_i = 1 \mid m_i = 1\right) = \prod_{i=1}^{k} \mathbb{P}(\tilde{m}_i = 1 \mid m_i = 1) = p^k$$

This allows to retrieve what would be a True Positive for multi-bit methods (see Table 4). The idea is to see that very good results on bit-accuracy do not translate to FPR (for a 30-bits message, 0.99 high bit accuracy only give 0.74 word accuracy).

Table 4: TPR of multi-bits methods used as zero-bit, for different targeted FPR values. The case $k = 30$ would correspond to the message length of HiDDeN [74].

k	FPR	p	TPR as a 0-bit method
20	10^{-6}	0.99	0.82
30	10^{-9}	0.99	0.74
20	10^{-6}	0.95	0.36
30	10^{-9}	0.95	0.21

5.3 Multi-bit data hiding

Quantitative results The setup is the same as before, i.e. we evaluate the method on 1000 images of the YFCC dataset, with PSNR targeted at 40dB. The images are watermarked with the multi-bit method described in Multi-bit watermarking in the hyperruff. We set λ_R to 0 meaning that we are only interested in recovering messages without discriminating watermarked images from non watermarked ones (the case $\lambda_R = 1$, that combines zero-bit and multi-bit watermarking setups, is studied in Zero-bit watermarking with the hyperruff detector). For each image, the message to encode is made of $k = 30$ bits chosen at random. This is the typical length used in previous deep data hiding methods [43, 74].

Figure 19 and Table 5 present the bit and word accuracy of the decodings over various transformations applied before detection. The decoding achieves a high bit accuracy over a wide range of geometric (rotation, crops, resize, etc.) and pixel-value (brightness, hue, contrast, etc.) transformations. The rotation and Gaussian blur transformations behave particularly well since the same transformations are seen both at pretraining and at marking time.

However high bit accuracy does not transfer to high word accuracy since all the bits must be perfectly decoded. For instance the 25 degrees rotation has 0.97 bit accuracy but 0.57 word accuracy. To achieve higher word accuracy, the role of error correction is pointed out in subsection 6.6.

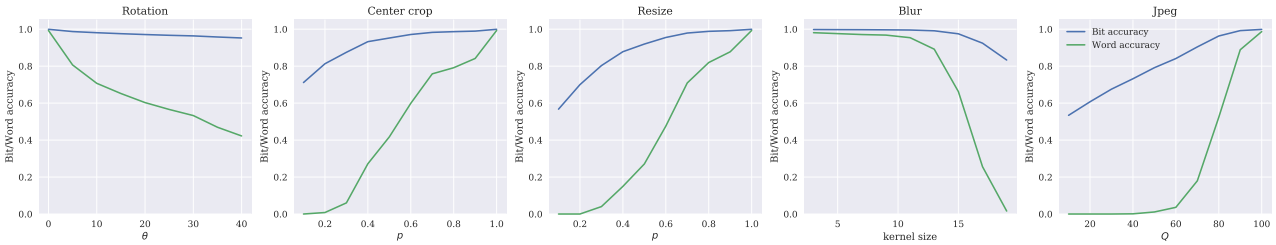


Figure 19: Average bit accuracy (blue) and word accuracy (green) of the decodings, over 1000 images of the YFCC dataset, against the strength of the transformations applied before detection. The target PSNR is set to 40, i.e. all the watermarked images have PNSR ≥ 40, the encoded messages have length 30.

Table 5: Bit and word accuracy of decoded messages over a broad type of transformations applied to images before decoding. The results are averaged over 1000 images of the YFCC dataset. Random 30-bits messages are encoded into the images.

Transformations	Bit accuracy	Word accuracy
Identity	0.999	0.993
Rotation (25)	0.967	0.566
Crop (0.5)	0.952	0.417
Crop (0.1)	0.711	0.000
Resize (0.7)	0.979	0.709
Gaussian Blur ($\sigma = 2$)	0.995	0.954
JPEG (50)	0.792	0.011
Brightness (2.0)	0.913	0.393
Contrast (2.0)	0.916	0.374
Hue (0.25)	0.975	0.684
Meme format	0.936	0.241
Screenshot	0.761	0.000

Trade-offs As before, there are trade-offs between capacity, robustness and imperceptibility of the watermarks. Figure 20 and Figure 21 illustrate these trade-offs.

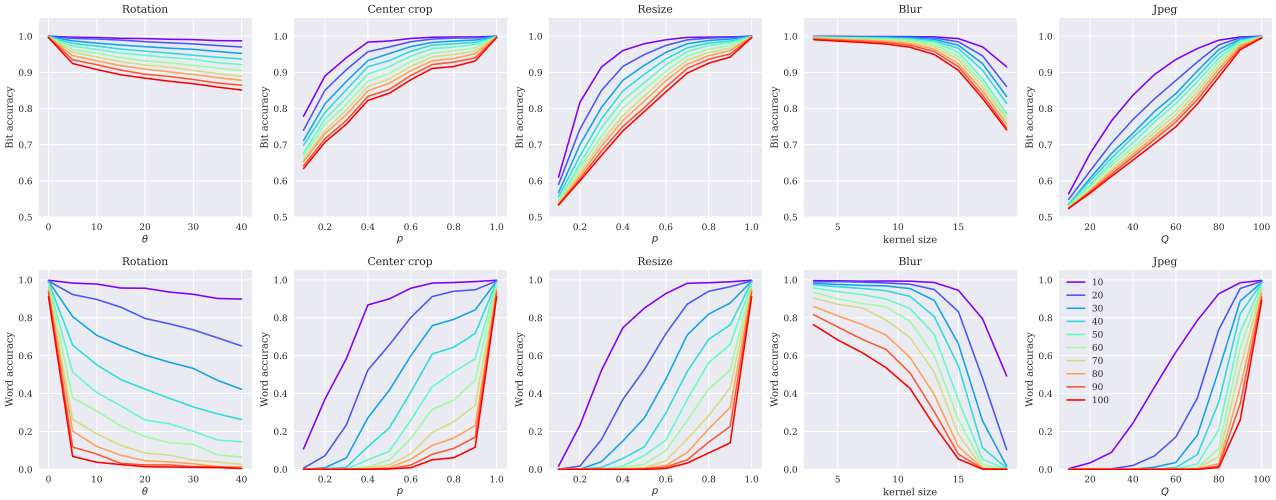


Figure 20: Illustration of the Capacity/Robustness trade-off. The length of the message increases from 10 (purple) to 100 (red): short messages are more robust to transformations than long ones. The target PSNR is set to 40.

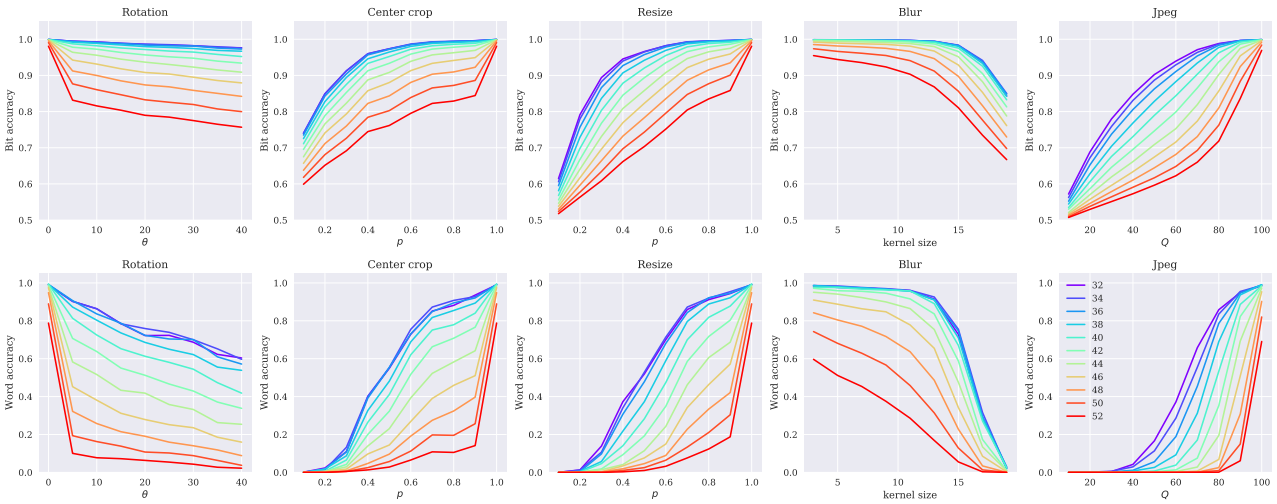


Figure 21: Illustration of the Quality/Robustness trade-off. The PSNR of the watermarked images increases from 32 (purple) to 52 (red): better robustness is achieved for lower quality images. The length of the message is set to 30.

Qualitative results Figure 22 presents some images watermarked at PSNR 40dB, as well as their pixel-wise difference with regards to the original images. As before, the watermark is almost invisible even to the trained eye. Figure 36 in the Appendix presents the same images watermarked with different PSNR values. One interesting difference is noticeable between the 0-bit watermarking results presented in section 5.2 and the ones of the multi-bit watermarking method. At fixed PSNR, the watermark is perceptually less visible for multi-bit watermarking than for 0-bit. One explanation is that the energy put into the image feature is more spread-out across carriers in the multi-bit setup than on the 0-bit one (where the feature is pushed at much as possible towards one carrier).

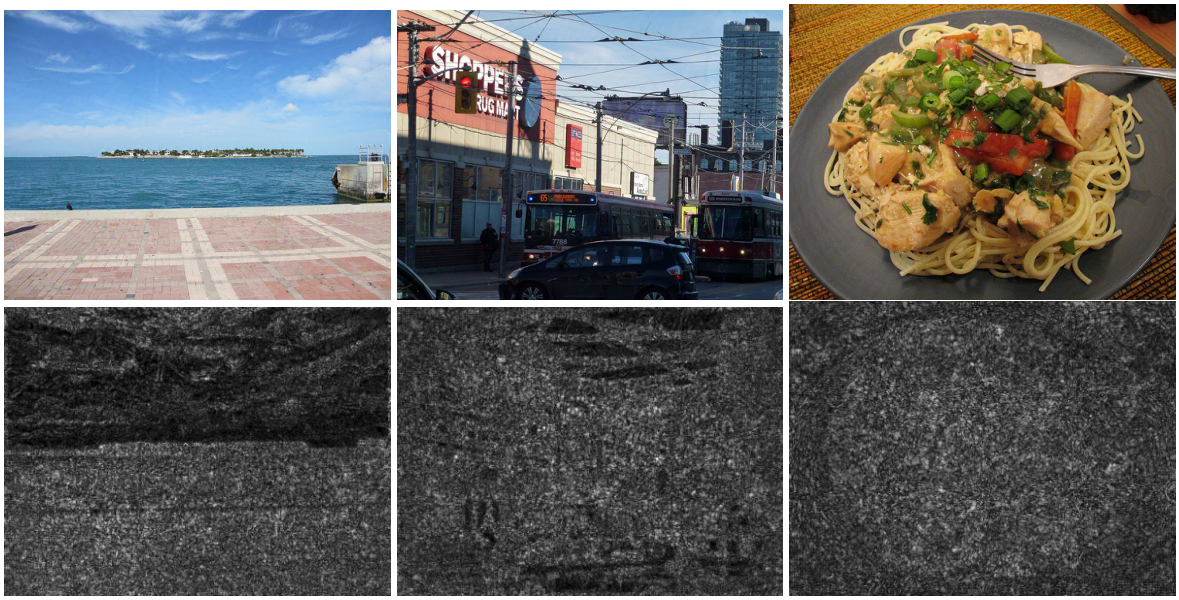


Figure 22: Example of watermarked images at PSNR 40 (Top row). The second row gives the pixel-wise l_1 distance between the images and their watermarked counterpart. The messages to encode are 30 bits long.

Comparison with the state of the art We compare against previous two deep data hiding methods HiDDeN and distortion agnostic [74, 43]. We evaluate our method on 1000 images from the COCO dataset [42], resized to 128×128, and encode 30-bit messages into the images. Since the average PSNR values of the images reported by the authors are around 33dB, we set the target PSNR to 33dB. The results of the average bit accuracy for different transformations are presented in Table 6. Overall, our method gives comparable results to the state-of-the-art, except for the center crop transform where our method fails to achieve high accuracies. It should also be noted that the resize operation is not used as noise layer in neither of [43, 74], which means that our

method should have the advantage. On the contrary, while these methods are trained to be robust to JPEG compression with a differentiable approximation, our method achieves similar performance without specific training.

Table 6: Bit accuracy of the decodings of watermarked images over a broad type of transformations applied at detection. The results are averaged over 1000 images of the COCO dataset. The images are watermarked at PSNR $\geq$ 33dB and the message length is set to 30. The first row uses the original resolutions of the dataset, while the others use a resized version of it (to 128×128). Results for [74, 43] come from [43].

Transformation	Identity	JPEG (50)	Gaussian blur (1)	Crop (0.1)	Resize (0.7)	Hue (0.2)
Ours	1.00	0.96	1.00	0.82	1.00	0.97
Ours, 128×128	1.00	**0.84**	**0.99**	0.55	0.82	**0.94**
Zhu et al. [74]	1.00	0.77	**0.99**	**1.00**	0.85	0.71
Luo et al. [43]	1.00	0.82	0.93	0.98	**0.88**	**0.94**

Furthermore, our method easily scales to higher resolution images, where it achieves higher bit accuracies. The difference between low and high resolutions is pointed out in Figure 23. We tried to reproduce the results given by [43, 74], but in the absence of any pretrained model for [74] or exploitable code in [43], this turned out to be hard. We can make the assumption that their method also scales to higher resolution images. However it would need to retrain all the models at higher resolution. Further, the training process at higher resolutions is more computationally expensive since the message is repeated at each pixel location [74]. From what our experiments showed, it needs a lower batch-size to operate, new values for the hyperparameters, and is harder to train overall.

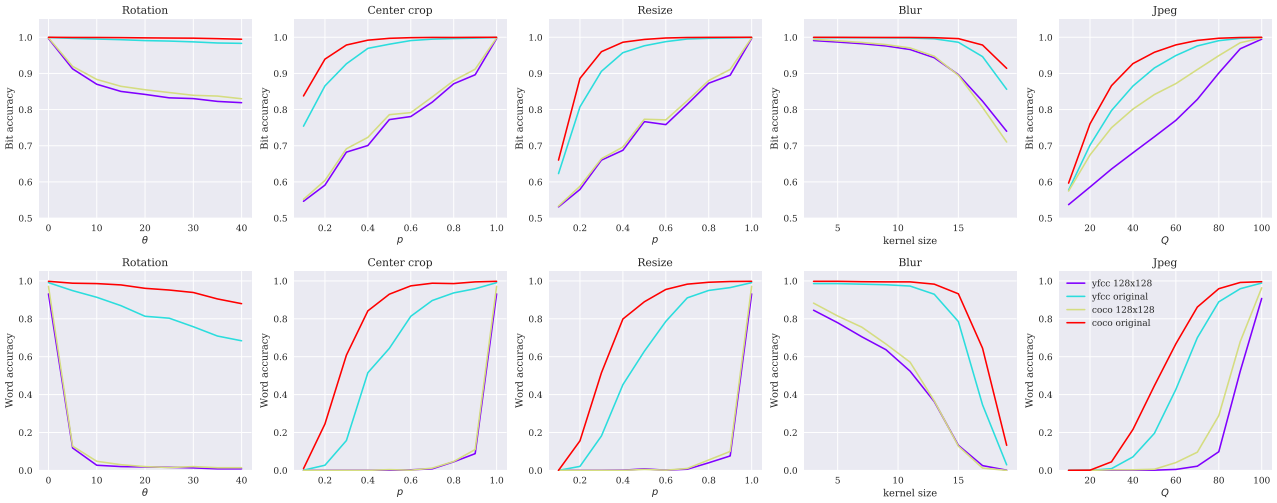


Figure 23: Bit accuracy against the strength of the transformations applied before detection, over 1000 images of the COCO and YFCC datasets, resized to 128×128 resolution or not. The target PSNR is set to 33, i.e. all the watermarked images have PNSR $\geq$ 33, and the message length is set 30.

6 Analyses

In all this section (unless specified otherwise), results are averaged over 1000 images of the YFCC dataset. The zero-bit watermarking method uses a target PSNR of 40dB and an FPR of 1e-6, the multi-bit watermarking one uses the same target PSNR and messages made of 30 bits. The default backbone is the one presented in Backbone pre-training.

6.1 Data-augmentations both at training and marking time

Vukotić et al. showed that networks trained on classification with progressively harder data augmentations improved the robustness of watermarks to these transformations, but it is not enough. Indeed the network learns more invariant features in order to classify the images, but nothing guarantees that the mark that is added will be robust too. By adding data augmentation in the marking process, we ensure that the mark that is added is detected for the image as well as the augmented version of it.

To highlight these statements, we train two ResNet-50 on ImageNet classification, one without rotation augmentation, the other with rotations of angle chosen randomly between ± 90. We do the same for ResNet-50 trained with DINO (see 5.1). Then we compare the zero-bit watermarking method (same setup as 5.2) with or without rotation augmentation at marking time. The results of the detection's robustness are given in Figure 24. Adding rotation at training time indeed improves the robustness of watermarks, even more for the self-supervised networks. As expected, adding rotation augmentation at marking time also improves the results, even more when the network has been pretrained with rotation augmentation. Both are necessary to obtain almost perfect robustness with regards to a transformation.

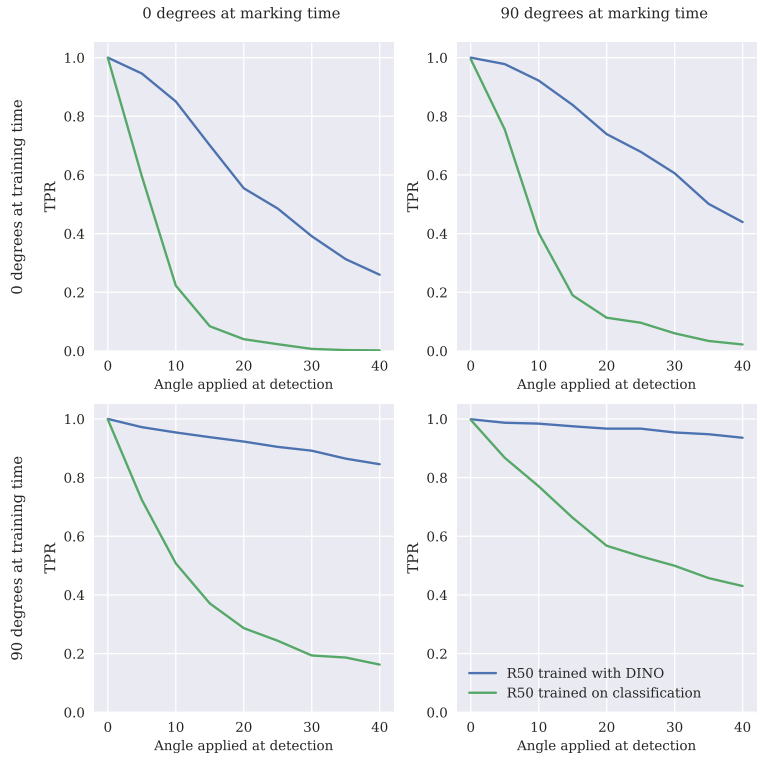


Figure 24: TPR against the angle of the rotation applied before detection. Each row represents a marking network pretrained with different amplitude in the rotation augmentation, each column represents a different amplitude of the rotation augmentation used during the marking process. Rotation augmentation is needed both at pretraining stage and at marking stage to obtain robustness to rotation.

The same observation can be made with the Gaussian blur transformation, as illustrated in Figure 25. However, this improvement is not applicable for all transformations. For example:

- Using crops, the watermarking process only optimizes few pixels of the image at each iteration. Therefore, to be robust to all crops having 1% of the original image, at least 100 iterations would be needed (by paving the original with crops having 1% of the original image). This considerably increases the computational complexity. (This may be a naive approach, and there might be ways to speed it up.)

- Transformations need to be differentiable in order to backpropagate image gradients through it. Typically, this does not cover the JPEG transformation, unless a differentiable approximation is used (as it is done in [74]).

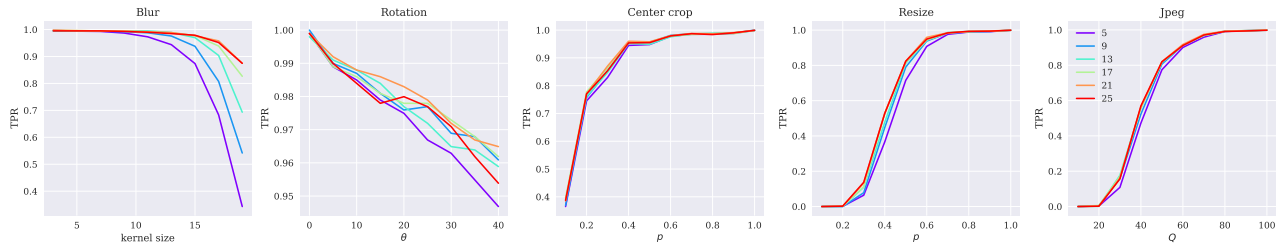


Figure 25: TPR against the strength of the transformations applied before detection. From blue to red the amplitude of the Gaussian blur's kernel size augmentation applied at marking time increases from 5 to 25. Adding more severe blurs at marking time leads to better performance at detection time without affecting the performance on other transformations.

6.2 Performance improvement with self-supervised learning

We hypothesize in 3.2 that self-supervised networks suit better the watermarking task than ones trained on classification. To illustrate this statement, we compare two networks as feature extractor for our zero-bit watermarking method. They are evaluated with the same setup as section 5.2 and the results are presented in Figure 26 and Table 7 (we also refer the reader to Figure 24 of the previous subsection). The improvement brought by the self-supervised pretraining is obvious for all the transformations. It is interesting to note that this improvement also occurs for transformations that were not part of the self-supervised pretraining, like JPEG.

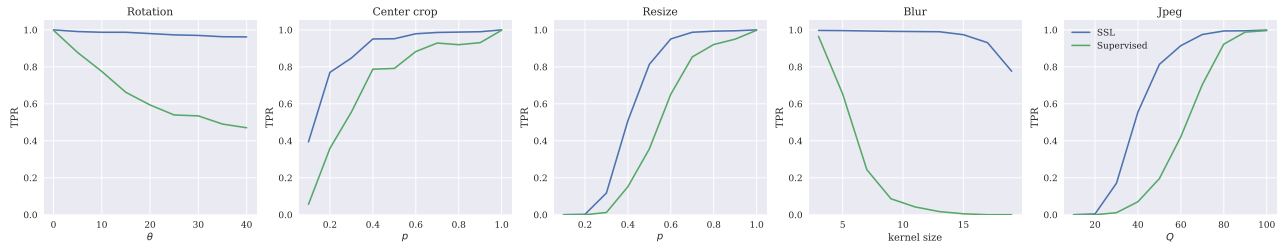


Figure 26: TPR against the strength of the transformations applied before detection, with or without self-supervision in the backbone pretraining.

Table 7: TPR of the detection of watermarked images over a broad type of transformations applied at detection, with or without self-supervision in the backbone pretraining.

Transformations	SSL	Supervised
Identity	1.000	0.998
Rotation (25)	**0.973**	0.540
Crop (0.5)	0.952	0.791
Crop (0.1)	**0.394**	0.056
Resize (0.7)	0.987	0.854
Gaussian Blur ($\sigma = 2$)	**0.991**	0.042
JPEG (50)	**0.813**	0.195
Brightness (2.0)	0.946	0.713
Contrast (2.0)	0.963	0.645
Hue (0.25)	**0.996**	0.463
Meme format	0.988	0.935
Screenshot	**0.756**	0.180

6.3 Analysis of the latent space and entropy regularization

PCA in the embedding space We perform the same analysis as in El-Nouby et al. [20]. In Figure 27 we examine the cumulative energy of the principal components for features from a model trained on ImageNet classification, as well as trained using DINO. We observe that the features after training with the classification task suffer from a collapse in dimensions compared to a DINO-pretrained model, where the cumulative energy spreads across more dimensions. This suggests an ineffective use of the representational capacity of the embedding space, as alignment is favored over uniformity while both are necessary for good representations (see [63]).

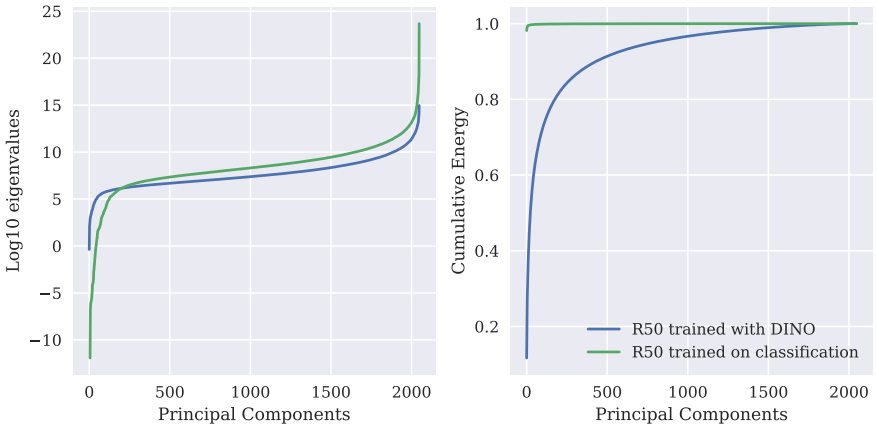


Figure 27: $\log_{10}$ eigenvalues and cumulative energy of the principal components for features extracted using a ResNet-50 backbone. The pretraining is either done with DINO or on classification. The features have collapsed to few dimensions after training on classification, but is reduced by using self-supervision.

Differential entropy regularization Wang and Isola provide a theoretical analysis for contrastive representation learning in terms of alignment and uniformity on the hypersphere. However, how SSL methods like DINO or BYOL behave with regards to uniformity is not clear at the moment. Following this lead, we used differential entropy regularization to improve properties of the latent space (especially uniformity).

Zhang et al. aim a better utilization of the space by spreading out the descriptors through matching first and second moments of non-matching pairs with points uniformly sampled on the sphere. Most related to out method, Sablayrolles et al. propose a differential entropy regularization based on the estimator by Kozachenko & Leonenko [39], in order to spread the vectors on the hypersphere more uniformly, such that it enables improved lattice-based quantization properties.

Therefore, we trained new models with the same parameters as 5.1, but added an entropic regularization loss to the features output by the student network in DINO. Let $x_i, i = 1, ..., n$ be the images of a batch and $f(x_i)$ their extracted features. This loss is either:

- $\mathcal{L}_{KoLeo} = -\frac{1}{n}\sum_{i=1}^{n}\log(\rho_{i,n})$, where $\rho_{i,n} = \min_{j\neq i}\|f(x_i) - f(x_j)\|$. This loss was introduced in [54] and has a satisfactory geometric interpretation: closest points are pushed away, with a strength that is non-decreasing and concave. This ensures diminishing returns: as points get away from each other, the marginal impact of increasing the distance becomes smaller. We refer the interested reader to the original paper for more information.

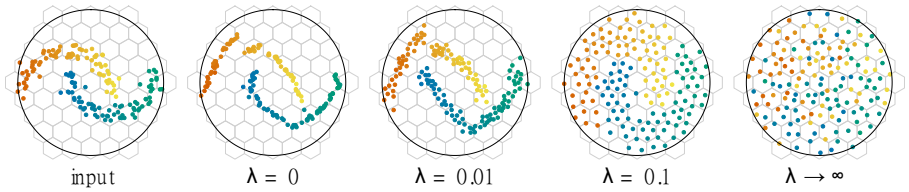


Figure 28: Illustration of the influence of the KoLeo loss taken from [54]. It aims at preserving the neighborhood structure in the input space while best covering the output space (uniformly).

- $\mathcal{L}_{GOR} = M_1^2 + \max(0, M_2 - 1/d)$, where

$$M_1 = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i} f(x_i)^\intercal f(x_j)$$

and

$$M_2 = \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i} (f(x_i)^\intercal f(x_j))^2$$

Here, the loss is called the Global Orthogonal Regularization loss. The first term tries to match the mean of the distributions and the second term tries to make the second moment close to $1/d$. We refer the reader to [73] for more information.

The resulting networks had almost exactly the same performance at the watermarking task, so we don't report any results. However, $\mathcal{L}_{KoLeo}$ noticeably improved the results of the networks on ImageNet classification task, with the KNN evaluation setup (see [12]). These results are presented in Table 8

Table 8: Top-1 accuracy on ImageNet KNN evaluation, for networks trained with or withouta KoLeo term added to the DINO loss with weight 0.1.

Neighbors	ViTs 16		ResNet-50	
	Without	With	Without	With
10	72.33	**73.51**	67.21	**67.92**
20	72.16	**73.45**	67.29	**68.04**
100	70.34	**72.22**	65.58	**66.70**
200	69.30	**71.42**	64.50	**65.70**

6.4 Zero-bit watermarking with the hyperruff detector

The setup is the same as 5.3, i.e. we evaluate the method on 1000 images of the YFCC dataset, with PSNR targeted at 40dB, and encode a 30-bit message onto the images. The images are watermarked with the multi-bit method described in Multi-bit watermarking in the hyperruff. The only difference is that we set λ_R to 1 meaning that we are interested in discriminating watermarked images from non watermarked ones. As in the hypercone detector case, the user can set a theoretical value for the FPR and compute θ accordingly. The effective FPR was checked by setting it to different values and observing the FPR over 100K natural images from the YFCC dataset: setting it to 1e-3 gave 0.963e-03, 1e-4 gave 1.605e-04 and 1e-5 gave 1.003e-05 (very close to the theoretical values).

The results are given in Figure 29. We observe that setting λ_R to 1 allows to retrieve a detection information from the presence of the feature in the hyperruff. However, the observed TPR are lower than the ones of 5.2, and the detection is less robust to transformations. In the same way, the message decodings are less accurate than in 5.3.

6.5 Channel modeling: Projections onto the carriers as parallel Gaussian channel

Gaussian channels As mentioned in Multi-bit watermarking in the hyperruff, given a k-bit message m, the bit m_i is decoded from the dot product between the extracted feature x and the i-th carrier a_i. We can assimilate this dot product as a Gaussian channel [14]. The Gaussian channel is one of the most important continuous alphabet channel. It is a discrete channel with output Y, where Y is the sum of the input $X = m_i$ and the noise Z. The noise Z is drawn i.i.d. from a Gaussian distribution with variance N. Thus,

$$Y = X + Z, \qquad Z \sim \mathcal{N}(0, N) \tag{13}$$

In our case, $Y \approx x^\intercal a_i$ and $X \approx m_i$.

The most common limitation on the input is an energy or power constraint. We assume an average power constraint: for any codeword $x_1, ..., x_k$ transmitted over the channel, we require that $1/k \sum_{i=1}^{k} x_i^2 \geq P$. For instance, if we want to send 1 bit over the channel, one would send one of the signals $X = \pm\sqrt{P}$. The receiver would look at the corresponding Y received and decide that $\sqrt{P}$ was sent if $Y > 0$ and $-\sqrt{P}$ otherwise. (With this scheme, the Gaussian channel has been converted into a discrete binary symmetric channel.)

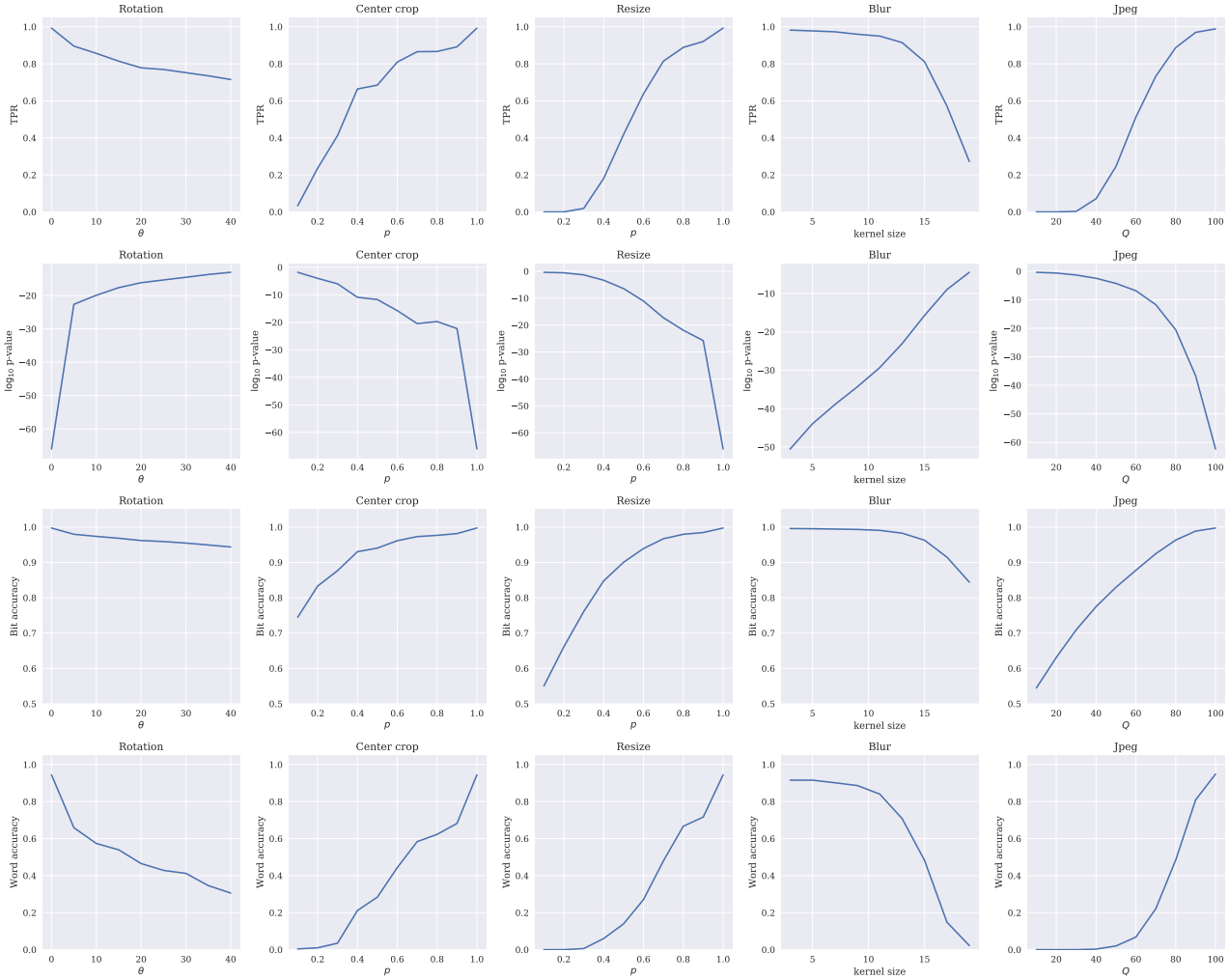


Figure 29: Performance of the multi-bit method with $\lambda_R = 1$ (detection + decoding setup) over a broad type of transformations. From first to 4th row: TPR of the detection, average $\log_{10}$ p-value of the detection, average bit accuracy, word accuracy.

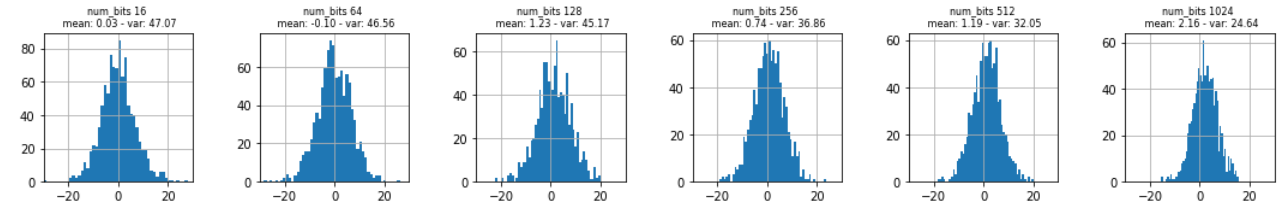
The information capacity of the Gaussian channel with power constraint P is (see [14]):

$$C = \max_{\mathbb{E}X^2 \leq P} I(X;Y) = \frac{1}{2}\log_2\left(1 + \frac{P}{N}\right) \tag{14}$$

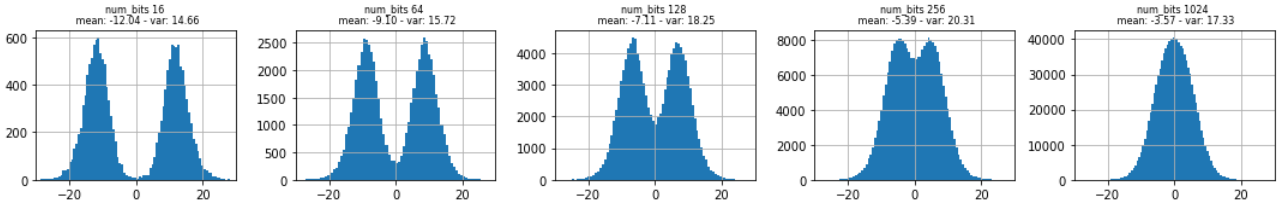
Projections onto the carriers The similarity between projections onto the carriers and Gaussian channels come from their distribution. In our case, the image carries k Gaussian channels ($x^\intercal a_i$ for $i = 1, ..., k$). To illustrate this, we embedded messages of different lengths with the same setup as section 5.2 and observed dot products between the features of the watermarked images and (1) a random vector that was not used as carrier, (2) the k orthogonal carriers that encode the bits. The results are presented in Figure 30. In the case where messages encoded two different bits, 2 clusters can be distinguished corresponding to bit 0 and 1. The mean and variance of these clusters can be estimated using and EM algorithm, and the ones of the first cluster are reported in the same figure.

The distributions can indeed be seen as Gaussian ones (even if it is not perfectly the case, see 8.1). When a bit is encoded in a projection, the dot product is shifted towards left or right depending on its value, as it is the case with the Gaussian channel presented in the previous paragraph. The effect of the transformations applied before detection is to "unshift" the mean and increase the variance, as it can be seen in Figure 30d.

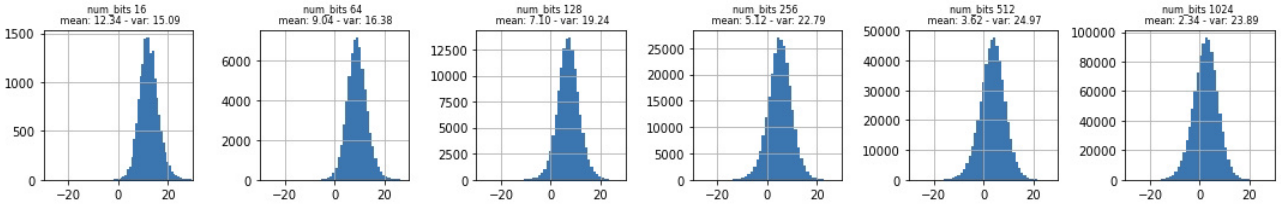
However, the noise of the channel and the power constraint on each of these channels depends on k (the number of bits to encode). For instance, in the case where $k = 16$, the noise is small with regards to the signal that is transmitted and almost all the bits can be recovered. In the case where $k = 256$, the clusters are still distinguishable, but their intersection is important, meaning that errors occur a lot more often. The rest of this section aims to estimate these values, to get back to the capacity of the channels.



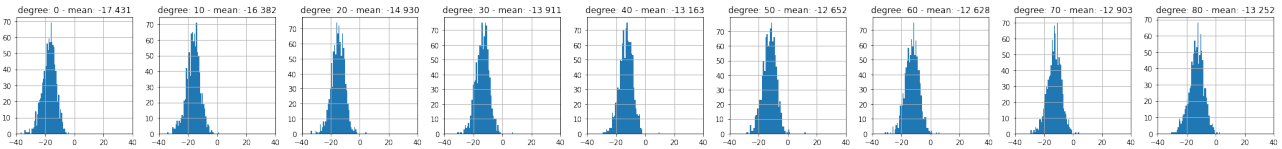
(a) Dot products with a random vector that serves as control, for 1000 decoded images



(b) Dot products across all channels for 1000 decoded images



(c) Dot products across all channels for 1000 decoded images when encoding full-1 messages



(d) Impact of the rotation over the dot products, when encoding the message "1"

Figure 30: Distribution of the dot products $x^\intercal a_i$ between the feature x of watermarked images, and carriers a_i on which bits were encoded.

Estimation of the capacity To estimate C, we use Equation 14. We estimate the power by computing $\mathbb{E}X^2$ where X is the mean of the distribution. We estimate the noise by computing $\mathbb{E}Y^2 - (\mathbb{E}Y)^2$ where Y^2 are the observed dot products. The estimations are made with the experimental setup presented above and by encoding messages full of 1 (to have an accurate estimation of the mean and variance, without relying on the EM algorithm). They are presented in Figure 31 and in Table 9.

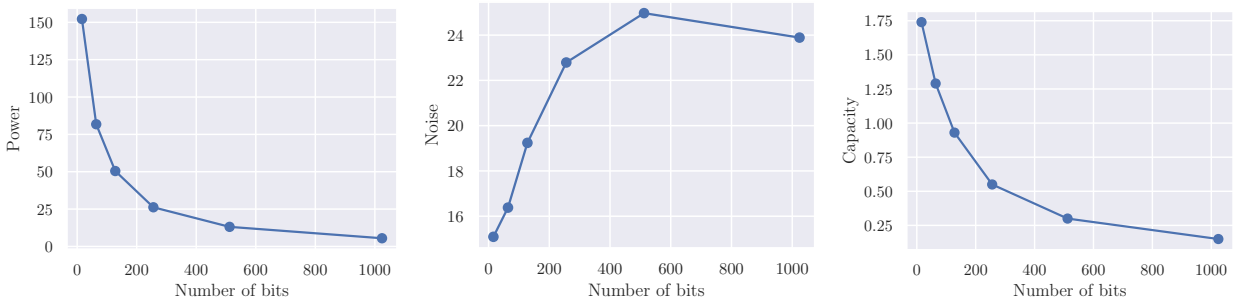


Figure 31: Estimation of the power, the noise and the capacity of the dot products between the watermarked features and the carriers.

We observe that the capacity reaches a plateau at 153.6, meaning that the method is limited on the number of bits it can robustly encode in the image. It should be noted that this is only a theoretical limit on the number of bits that can possibly be encoded with this method. In practice, such rates are hard to achieves and need to carefully design error correction.

Table 9: Estimation of the power, the noise and the capacity of the dot products between the watermarked features and the carriers.

k	16	64	128	256	512	1024
Power	152.23	81.78	50.46	26.22	13.14	5.47
Noise	15.09	16.38	19.24	22.79	24.97	23.89
Capacity	1.74	1.29	0.93	0.55	0.30	0.15
Total capacity	27.8	82.6	119	140	153.6	153.6

6.6 Channel coding

In computing, information theory and coding theory, channel coding (or error correction) is the process of detecting and correcting bit errors during the noisy transmission of a message [14]. Usually it does so by splitting the transmission into 3 stages: encoder, channel and decoder. From the original message (of length k), the encoder generates a codeword (of length $n > k$) that incorporate supplementary information (parity check bits for instance). The decoder detects and/or corrects errors that happen during transmission of the codeword and it generates the received message (ideally the same as the original one). The rate of the ECC indicates the amount of redundant information and is defined as $r = k/n$.

We are particularly interested in *soft-decision* algorithms to decode the modulated signal. Indeed, whereas a hard-decision decoder operates on data that take on a fixed set of possible values (typically 0 or 1 in a binary code), the inputs to a soft-decision decoder may take on a whole range of values in-between. This allows to get the most of the information conveyed by the projections onto the carriers, that are continuous.

Redundancy The most straightforward way of incorporating error correcting codes into our method is to use redundancy. It consists of using the dot products between feature x and $R \times k$ carriers, instead of the k original ones. Say $m = m_1 m_2 ... m_k$ is the original message. The codeword to encode into the image is the same message repeated R times: $m_1 ... m_k ... m_1 ... m_k$. At decoding time, we retrieve bit $\tilde{m}_i$ by summing all dot products encoding for m_i, i.e. $\tilde{m}_i = \left(\sum_{j=1}^{R} x^\intercal a_{i+k*j} > 0 \right)$.

The redundancy method is rather straightforward but not optimal. Indeed, it needs to encode a R times the number of bits while our method is limited in the capacity it can robustly achieve. Therefore, it only improved results noticeably for small messages ($k \leq 20$). In the next paragraph we present another error correcting code.

Low-density parity-check (LDPC) codes LDPC codes are efficient channel coding codes that allow transmission errors to be corrected. They were first described by Gallager in 1960 [27]. They are defined by a sparse parity-check matrix H of dimension $m \times n$, filled with 0 or 1, that is often randomly generated. We say that an LDPC code is a regular when each column of H contains exactly d_v ones and each row dc ones with $d_c > d_v > 1$. Some of the most important properties of regular LDPC codes are the following:

- The amount of 1 follows: $n.d_v = m.d_c$
- The rate of the LDPC follows: $r \geq r_d = 1 - \frac{d_v}{d_c}$

From these properties, one can construct LDPC codes that has a rate higher than a fixed value. Results of section 5.3 highlighted that performance decreases rather fast with the number of bits to encode in the image. Therefore, in our case, we should aim for LPDC with high rates. In Table 10 we describe some configurations of LDPC codes that allow to encode rougly 30 bits messages.

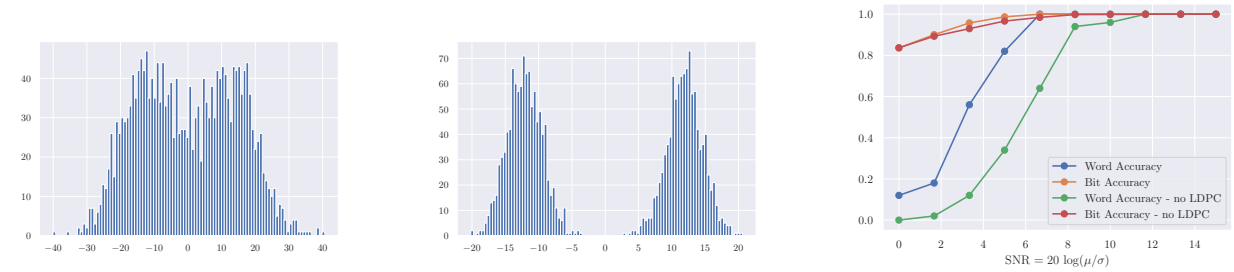
Table 10: LDPC codes configurations. d_v and d_c are the number of 1 per column and per row, k is the length of the message to transmit, n is the lenght of the codeword to encode.

d_v	2					3			
d_c	3	4	5	6	7	4	5	6	7
n	87	60	50	48	42	112	70	60	49
k	30	31	31	33	31	30	30	32	30

A widespread way of decoding LDPC codes is to use belief propagation [51], which is what is used by the library we used to run the following experiments.

Results with LDPC codes First, we generated messages of 30 bits and encoded them using an LDPC code with $(d_v, d_c) = (2, 7)$ (hence $n = 42$ is the length of the message to convey in the image). We then simulated the watermarking process by using Gaussian channels with power and noise set to look like the results presented in subsection 6.5. To study the impact of transformations on the image, we considered different value for the SNR (Signal to Noise Ratio), which is defined as the ratio between the mean and the standard deviation of the distribution (as we have seen, the harder the transformation applied to the image, the lower the SNR of the bit's transmission). This was done by varying the noise used to simulate the Gaussian channel.

Figure 32 describes the influence of LDPC codes on the transmission simulated at different SNR values. The LDPC codes do not improve much the bit accuracy of the decodings, but they do improve the word accuracy.



(a) SNR=3 (b) SNR=13 (c)

Figure 32: Results of LDPC decodings on simulated data. (a) and (b) present the distribution of simulated noise across all the channels at different SNR values. (c) present the bit accuracy and word accuracy of the decodings, against the SNR of the simulated data. LDPC codes allow to greatly improve word accuracy.

From this simulation, we moved on to the same setup as section 5.3 and compared the performance obtained when using LDPC codes or not. In both cases, the original messages are made of 30 bits. This means that with LDPC codes, 42 projections are optimized, while only 30 are in the original case. Bit and word accuracy of the decodings are presented in Figure 33. As expected LDPC codes bring a noticeable improvement in the word accuracy, even if they generate a small drop in bit accuracy.

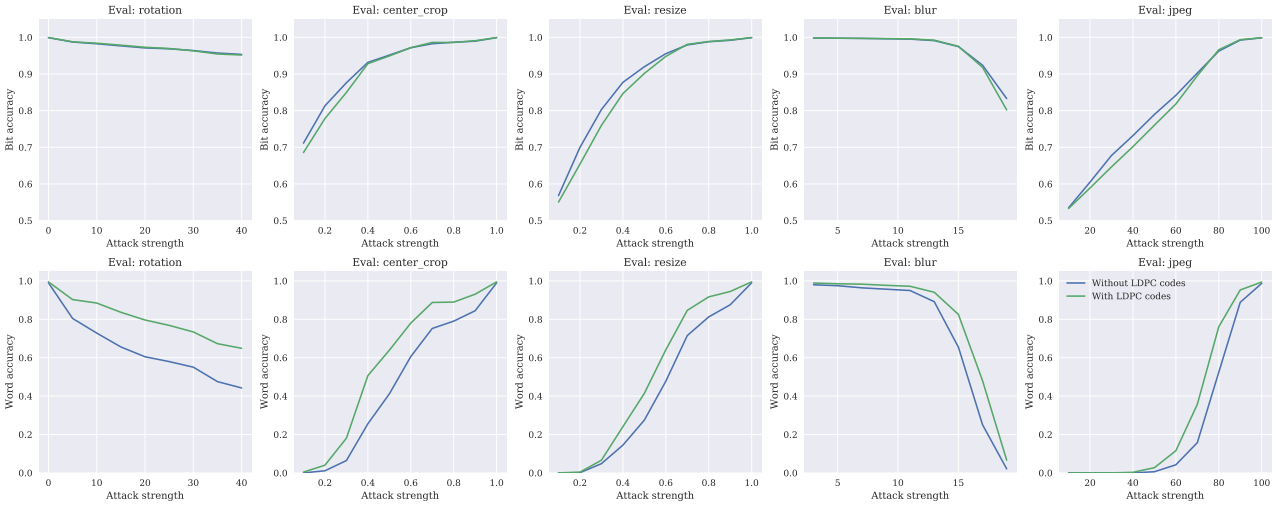


Figure 33: Bit and word accuracy against the strength of the transformations applied before detection, with and without LDPC codes.

7 Conclusion & Discussion

The method proposed in this work is a way to robustly and invisibly embed information - which can either be a mark or a message - into digital images. It uses recent advances in self-supervised learning and the underlying representational properties of the latent spaces generated from it.

By incorporating data augmentation and constraints into the optimization procedure, our zero-bit watermarking method is able to considerably improve performance over the baseline [62]. It achieves high robust-

ness against a wide range of severe transformations, from geometrical ones (rotation, crops, resize, ...) to pixel-value ones (JPEG, contrast, brightness, ...), while keeping high fidelity with regards to the original images, and ensuring a very low false positive rate for the detection. Moreover, extending the method to multi-bit watermarking leads to very promising results, comparable to the state-of-the-art in deep data hiding, and even better with regards to some transformation of the image (namely JPEG compression or blur).

The most interesting (and surprising) aspect of the work is that the networks trained with self-supervision naturally generate spaces suitable for the watermarking task. Indeed, in previous deep data hiding methods such as HiDDeN [74] or Distortion Agnostic [43], architectures are explicitly trained for watermarking and learn representations for that goal. The latent spaces are optimized to be marking spaces above all else. On the contrary, our method considers the latent space as fixed, and yet obtain similar performance (and better in the case of JPEG that has not been seen neither at training or marking time). This could have two implications. First, progress in self-supervised methods would also imply progress in image watermarking, which supports the idea of a universal descriptor learner for vision. Second, it should be possible to finetune the networks and further improve the results.

This present some advantages but also some drawbacks that should be addressed in future works. For instance, even if there is no need to re-train a network for the watermarking task when changing image resolution or message length, the backtracking steps at marking time are computationally expensive. (A rough estimation is 5 minutes on a single GPU to watermark 1000 images if images can be batched, i.e. they have the same size). On the opposite, current deep data hiding, based on end-to-end architectures for the most part, only need one forward pass to watermark images (at the cost of an expansive one-time-training process). An open lead for future works would be to distill our method into a network and to teach it how to watermark images in a single forward pass.

References

[1] Workshop and challenge on learned image compression. URL http://clic.compression.cc/2018/challenge/.

[2] Mahdi Ahmadi, Alireza Norouzi, Nader Karimi, Shadrokh Samavi, and Ali Emami. Redmark: Framework for residual diffusion watermarking based on deep networks. *Expert Systems with Applications*, 2020.

[3] Laurent Amsaleg, James Bailey, Amelie Barbe, Sarah M Erfani, Teddy Furon, Michael E Houle, Miloš Radovanović, and Xuan Vinh Nguyen. High intrinsic dimensionality facilitates adversarial attack: Theoretical evidence. *IEEE Transactions on Information Forensics and Security*, 2020.

[4] Shumeet Baluja. Hiding images in plain sight: Deep steganography. *NeurIPS*, 2017.

[5] Joanna Bitton and Zoe Papakipos. Augly: A data augmentations library for audio, image, text, and video. https://github.com/facebookresearch/AugLy, 2021.

[6] Adrian G Bors and Ioannis Pitas. Image watermarking using dct domain constraints. In *Proceedings of 3rd IEEE International Conference on Image Processing*. IEEE, 1996.

[7] Malik Boudiaf, Jérôme Rony, Imtiaz Masud Ziko, Eric Granger, Marco Pedersoli, Pablo Piantanida, and Ismail Ben Ayed. A unifying mutual information view of metric learning: cross-entropy vs. pairwise losses. In *ECCV*, 2020.

[8] Richard P. Brent. Algorithms for minimization without derivatives. 1973.

[9] Olivia Byrnes, Wendy La, Hu Wang, Congbo Ma, Minhui Xue, and Qi Wu. Data hiding with deep learning: A survey unifying digital watermarking and steganography. *arXiv preprint arXiv:2107.09287*, 2021.

[10] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *ECCV*, 2018.

[11] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *Neural information processing systems*, 2020.

[12] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. *arXiv preprint arXiv:2104.14294*, 2021.

[13] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*. PMLR, 2020.

[14] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[15] Ingemar Cox, Matthew Miller, Jeffrey Bloom, Jessica Fridrich, and Ton Kalker. *Digital watermarking and steganography*. Morgan kaufmann, 2007.

[16] Ingemar J Cox, Joe Kilian, F Thomson Leighton, and Talal Shamoon. Secure spread spectrum watermarking for multimedia. *IEEE transactions on image processing*, 1997.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009.

[18] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, 2015.

[19] Carl Doersch, Ankush Gupta, and Andrew Zisserman. Crosstransformers: spatially-aware few-shot transfer. *NeurIPS*, 2020.

[20] Alaaeldin El-Nouby, Natalia Neverova, Ivan Laptev, and Hervé Jégou. Training vision transformers for image retrieval. *arXiv preprint arXiv:2102.05644*, 2021.

[21] Liu Ping Feng, Liang Bin Zheng, and Peng Cao. A dwt-dct based blind watermarking algorithm for copyright protection. In *2010 3rd International Conference on Computer Science and Information Technology*. IEEE, 2010.

[22] Jessica Fridrich. *Steganography in digital media: principles, algorithms, and applications*. Cambridge University Press, 2009.

[23] Jessica Fridrich, Miroslav Goljan, and Rui Du. Detecting lsb steganography in color, and gray-scale images. *IEEE multimedia*, 2001.

[24] Teddy Furon. A constructive and unifying framework for zero-bit watermarking. *IEEE Transactions on Information Forensics and Security*, 2007.

[25] Teddy Furon. Binary tardos codes and zero-bit watermarking. 2018.

[26] Teddy Furon. Watermarking error exponents in the presence of noise: The case of the dual hypercone detector. In *ACM Workshop on Information Hiding and Multimedia Security*, 2019.

[27] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 1962.

[28] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *ICLR*, 2018.

[29] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *NeurIPS*, 2014.

[30] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *ICLR*, 2014.

[31] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *Neural information processing systems*, 2020.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[33] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

[34] Vojtěch Holub and Jessica Fridrich. Designing steganographic distortion using directional filters. In *WIFS*, 2012.

[35] Vojtěch Holub, Jessica Fridrich, and Tomáš Denemark. Universal distortion function for steganography in an arbitrary domain. *EURASIP Journal on Information Security*, 2014.

[36] Ahmet Iscen, Teddy Furon, Vincent Gripon, Michael Rabbat, and Hervé Jégou. Memory vectors for similarity search in high-dimensional spaces. *IEEE transactions on big data*, 2017.

[37] Haribabu Kandi, Deepak Mishra, and Subrahmanyam R.K. Sai Gorthi. Exploring the learning capabilities of convolutional neural networks for robust image watermarking. *Computers & Security*, 2017.

[38] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015. URL http://arxiv.org/abs/1412.6980.

[39] LF Kozachenko and Nikolai N Leonenko. Sample estimate of the entropy of a random vector. *Problemy Peredachi Informatsii*, 1987.

[40] Jae-Eun Lee, Young-Ho Seo, and Dong-Wook Kim. Convolutional neural network-based digital image watermarking adaptive to the resolution of image and watermark. *Applied Sciences*, 2020.

[41] Zhen Li, Kim-Hui Yap, and Bai-Ying Lei. A new blind robust image watermarking scheme in svd-dct composite domain. In *2011 18th IEEE International Conference on Image Processing*. IEEE, 2011.

[42] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*. Springer, 2014.

[43] Xiyang Luo, Ruohan Zhan, Huiwen Chang, Feng Yang, and Peyman Milanfar. Distortion agnostic deep watermarking. In *CVPR*, 2020.

[44] Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi Wijewickrema, Grant Schoenebeck, Dawn Song, Michael E. Houle, and James Bailey. Characterizing adversarial subspaces using local intrinsic dimensionality. *International Conference on Learning Representations*, 2018.

[45] Florence Jessie MacWilliams and Neil James Alexander Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

[46] Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM international conference on Multimedia*, 2010.

[47] Neri Merhav and Erez Sabbag. Optimal watermark embedding and detection strategies under limited detection resources. *IEEE Transactions on Information Theory*, 2008.

[48] Zhicheng Ni, Yun-Qing Shi, Nirwan Ansari, and Wei Su. Reversible data hiding. *IEEE Transactions on circuits and systems for video technology*, 16(3):354–362, 2006.

[49] Nikos Nikolaidis and Ioannis Pitas. Robust image watermarking in the spatial domain. *Signal processing*, 1998.

[50] Junlin Ouyang, Gouenou Coatrieux, Beijing Chen, and Huazhong Shu. Color image watermarking based on quaternion fourier transform and improved uniform log-polar mapping. *Computers & Electrical Engineering*, 2015.

[51] Judea Pearl. Reverend bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the Second AAAI Conference on Artificial Intelligence*. AAAI Press, 1982.

[52] Tomáš Pevnỳ, Tomáš Filler, and Patrick Bas. Using high-dimensional image models to perform highly undetectable steganography. In *International Workshop on Information Hiding*. Springer, 2010.

[53] Tenkasi V Ramabadran. A coding scheme for m-out-of-n codes. *IEEE Transactions on Communications*, 1990.

[54] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Spreading vectors for similarity search. *ICML*, 2019.

[55] Alexandre Sablayrolles, Matthijs Douze, Cordelia Schmid, and Hervé Jégou. Radioactive data: tracing through training. In *ICML*. PMLR, 2020.

[56] Derek H Smith, Lesley A Hughes, and Stephanie Perkins. A new table of constant weight codes of length greater than 28. *the electronic journal of combinatorics*, 2006.

[57] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations*, 2013.

[58] Matthew Tancik, Ben Mildenhall, and Ren Ng. Stegastamp: Invisible hyperlinks in physical photographs. In *CVPR*, 2020.

[59] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. Yfcc100m: The new data in multimedia research. *Communications of the ACM*, 2016.

[60] Matthieu Urvoy, Dalila Goudia, and Florent Autrusseau. Perceptual dft watermarking with improved detection and robustness to geometrical distortions. *IEEE Transactions on Information Forensics and Security*, 2014.

[61] Vedran Vukotić, Vivien Chappelier, and Teddy Furon. Are deep neural networks good for blind image watermarking? In *WIFS*, 2018.

[62] Vedran Vukotić, Vivien Chappelier, and Teddy Furon. Are classification deep neural networks good for blind image watermarking? *Entropy*, 2020.

[63] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *ICML*, 2020.

[64] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 2004.

[65] Bingyang Wen and Sergul Aydore. Romark: A robust watermarking system using adversarial training. *arXiv preprint arXiv:1910.01221*, 2019.

[66] Xiang-Gen Xia, Charles G Boncelet, and Gonzalo R Arce. Wavelet transform based watermark for digital images. *Optics Express*, 1998.

[67] Chong Yu. Attention based data hiding with generative adversarial networks. In *AAAI*, 2020.

[68] Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. *ICML*, 2021.

[69] Chaoning Zhang, Philipp Benz, Adil Karjauv, Geng Sun, and In So Kweon. Udh: Universal deep hiding for steganography, watermarking, and light field messaging. *NeurIPS*, 2020.

[70] Honglei Zhang, Hu Wang, Yuanzhouhan Cao, Chunhua Shen, and Yidong Li. Robust watermarking using inverse gradient attention. *arXiv preprint arXiv:2011.10850*, 2020.

[71] Kevin Alex Zhang, Alfredo Cuesta-Infante, Lei Xu, and Kalyan Veeramachaneni. Steganogan: High capacity image steganography with gans. *arXiv preprint arXiv:1901.03892*, 2019.

[72] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

[73] Xu Zhang, Felix X Yu, Sanjiv Kumar, and Shih-Fu Chang. Learning spread-out local feature descriptors. In *ICCV*, 2017.

[74] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *ECCV*, 2018.

8 Appendix

8.1 Distribution of a dot product

The proof is greatly inspired by [36].

Let u be a random vector uniformly distributed on the unit sphere of $\mathbb{R}^d$ ($\|u\| = 1$) and a fixed vector of the same hypersphere. We are interested in the distribution of the cosine similarity which is in this case $c(u, a) = u^\intercal a$.

Let $G_1, ..., G_d \sim \mathcal{N}(0, 1)$ and $G = (G_1, ..., G_d)^\intercal$ be such that $u = G/\|G\|$. Without loss of generality, we can assume that $a = (\|a\| = 1, 0, ..., 0)$ (up to a change of axes). Therefore,

$$u^\intercal a = \frac{G_1}{\sqrt{\sum_{i=1}^d G_i^2}}$$

Let $\tau \in \mathbb{R}^+$. We have

$$\begin{aligned}\mathbb{P}\left((u^\intercal a)^2 \geq \tau^2\right) &= \mathbb{P}\left(\frac{G_1^2}{\sum_{i=1}^d G_i^2} > \tau^2\right) \\ &= \mathbb{P}\left(\frac{G_1^2}{\sum_{i=2}^d G_i^2} > \frac{\tau^2}{1-\tau^2}\right)\end{aligned}$$

By definition $y = (d-1)\frac{G_1^2}{\sum_{i=2}^d G_i^2}$ is a ratio of Chi-squares distribution and follows a Fisher distribution $F(1, d-1)$. It follows that

$$\mathbb{P}(|u^\intercal a| > \tau) = 1 - I_{\tau^2}(1/2, (d-1)/2)$$

where $I_x(a, b)$ is the regularized incomplete beta function.

Finally setting τ to $\cos\theta$, we get that

$$\mathbb{P}(|u^\intercal a| > \cos\theta) = 1 - I_{\cos^2\theta}(1/2, (d-1)/2)$$

8.2 Distribution of the projection on a subspace

We now consider $\Pi_A(u)$ where A is an set of k orthogonal vectors of the hypersphere of $\mathbb{R}^d$. In the same way as before, we can consider A such that

$$\|\Pi_A(u)\|^2 = \frac{\sum_{i=1}^k G_i^2}{\sum_{i=1}^d G_i^2}$$

and therefore

$$\mathbb{P}(\|\Pi_A\|^2 > \tau^2) = \mathbb{P}\left(\frac{\sum_{i=1}^k G_i^2}{\sum_{i=k+1}^d G_i^2} > \frac{\tau^2}{1-\tau^2}\right)$$

Finally, $y = \frac{d-k}{k}\frac{\sum_{i=1}^k G_i^2}{\sum_{i=k+1}^d G_i^2}$ is a ratio of Chi-squares distribution and follows a Fisher distribution $F(k, d-k)$. Therefore, substituting τ by $\cos\theta$, we get:

$$\mathbb{P}(\|\Pi_A(u)\| > \cos\theta) = 1 - I_{\cos^2\theta}(k/2, (d-k)/2)$$

8.3 More watermarked images

Figure 34: From top to bottom: original images, watermarked images and the difference between the watermarked image and the original image, from the CLIC dataset. The watermark is added with our zero-bit watermarking method at FPR= 10^{-3} and target PSNR 42dB.

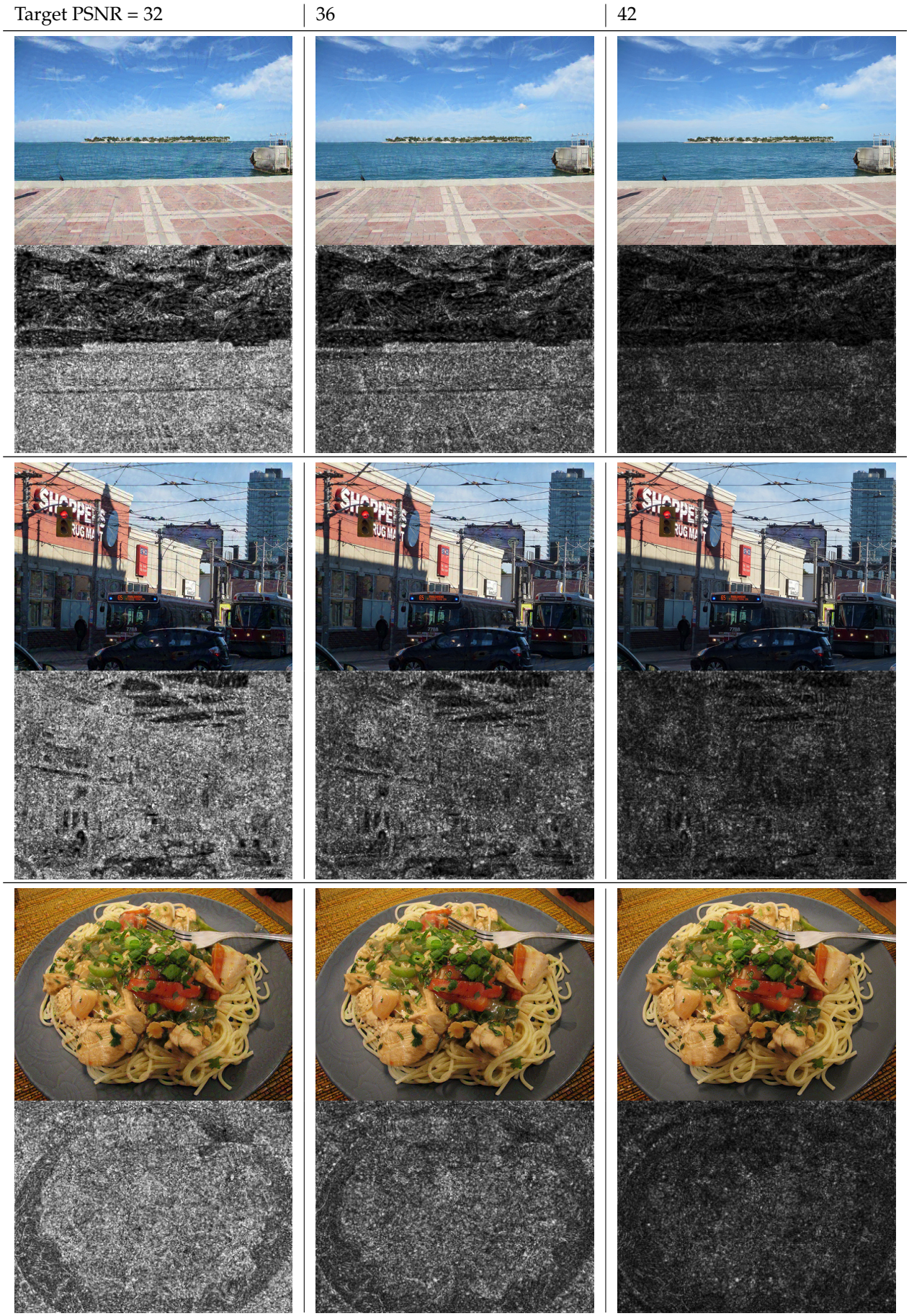


Figure 35: Watermarked images and the difference between the image and the original image. The watermark is added with our zero-bit watermarking method at FPR= 10^{-6} with different values for the target PSNR (with target PSNR $= 32$, the effective PSNR is around 33 for the 3 images).

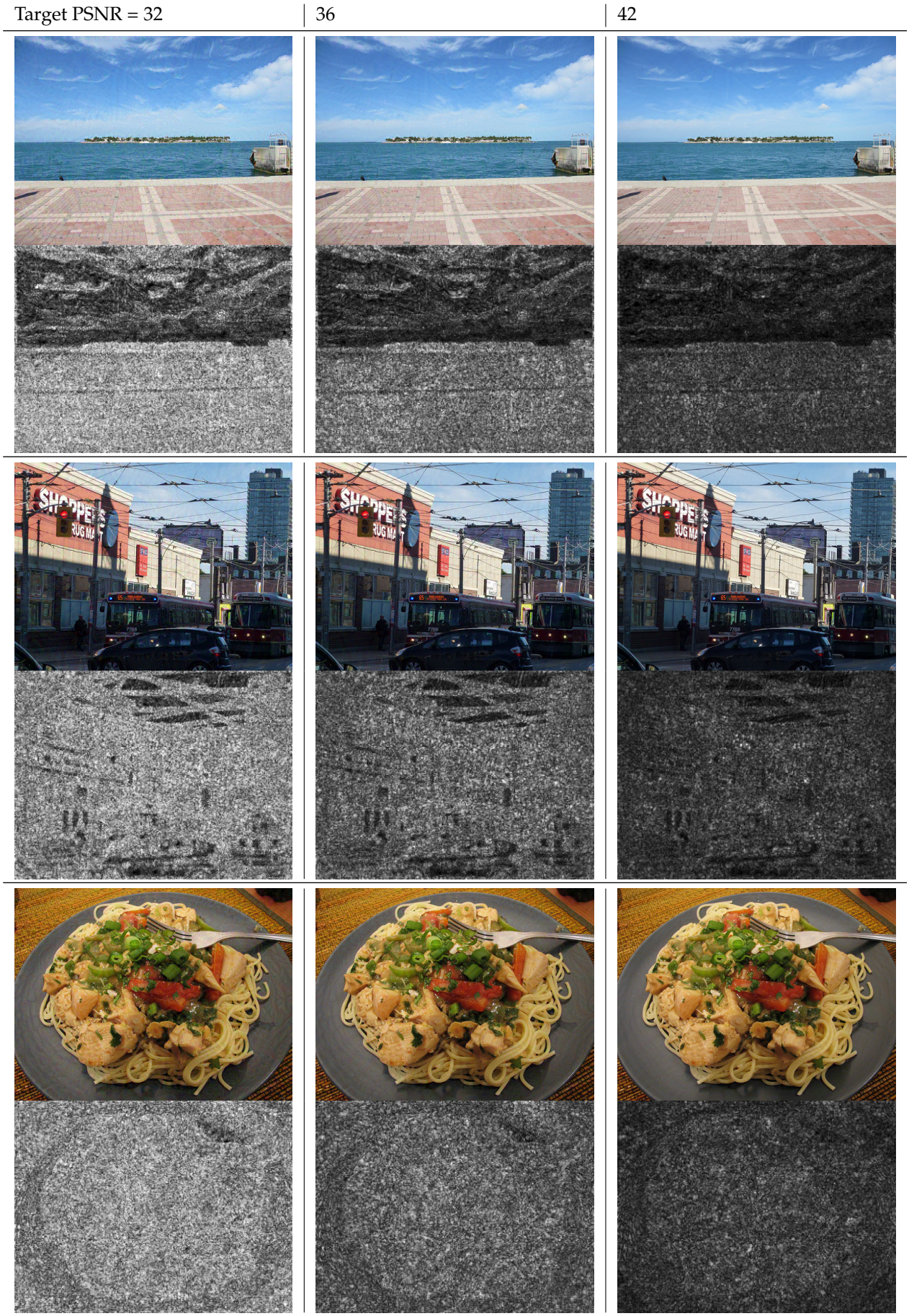


Figure 36: Watermarked images and the difference between the image and the original image. The watermark is added with our multi-bit watermarking method with message length 30 and with different values for the target PSNR (with target PSNR = 32, the effective PSNR is around 33 for the 3 images).