We Can Hide More Bits:

The Unused Watermarking Capacity in Theory and in Practice

Aleksandar Petrov[1,†], **Pierre Fernandez**[2], **Tomáš Souček**[2], **Hady Elsahar**[2,†]

[1]University of Oxford, [2]FAIR, Meta, [†]Core contributors

Despite rapid progress in deep learning–based image watermarking, the capacity of current robust methods remains limited to the scale of only a few hundred bits. Such plateauing progress raises the question: *How far are we from the fundamental limits of image watermarking?* To this end, we present an analysis that establishes upper bounds on the message-carrying capacity of images under PSNR and linear robustness constraints. Our results indicate theoretical capacities are orders of magnitude larger than what current models achieve. Our experiments show this gap between theoretical and empirical performance persists, even in minimal, easily analysable setups. This suggests a fundamental problem. As proof that larger capacities are indeed possible, we train Chunky Seal, a scaled-up version of Video Seal, which increases capacity 4× to 1024 bits, all while preserving image quality and robustness. These findings demonstrate modern methods have not yet saturated watermarking capacity, and that significant opportunities for architectural innovation and training strategies remain.

Correspondence: aleksandar@p-petrov.com

Meta

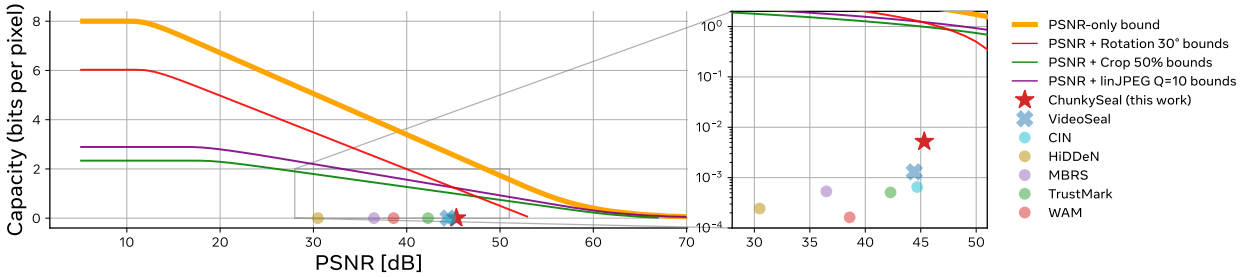

Figure 1 Existing image watermarking models have capacities well under what this paper suggests to be possible. Shown are theoretical bounds on watermarking capacity under a PSNR constraint alone (thick line) and in combination with robustness requirements (thin lines). Recent methods operate far below the achievable bounds, often by orders of magnitude, as seen in the log-scale inset. Our proposed **Chunky Seal (1024 bits)** pushes capacity higher than prior work, but is still very far from the theoretical limits, indicating a large potential for future development.

1 Introduction

Content provenance, the act of identifying the origin and history of media, is increasingly important as generative tools for creating and editing text, audio and video content are becoming ever more accessible and powerful. The most robust and reliable content provenance solutions, including those in the C2PA industry standard (C2PA), combine cryptographically secure metadata, fingerprinting, and invisible watermarking. Invisible watermarking is the key area of interest in both academia and industry. This paper focuses on the image domain, as it is the most mature, despite techniques also existing for text (Kirchenbauer et al., 2023; Dathathri et al., 2024) and audio (Chen et al., 2023; San Roman et al., 2024). Image watermarking is already integrated into commercial models (e.g., Meta AI, Adobe Firefly and Google Imagen) and available for third-party use via open-source models (Zhu et al., 2018; Bui et al., 2023a), and is increasingly being considered for regulatory mandates (The White House, 2023; California State Leg., 2024; European Parliament and Council, 2024).

Invisible image watermarking embeds an *imperceptible* secret message of a *certain capacity* recoverable *under*

a variety of perturbations, leading to an inherent capacity-quality-robustness trade-off. Classic methods used hand-crafted tools, such as the mid-frequencies of the discrete cosine transform (DCT) (Al-Haj, 2007; Navas et al., 2008). The advent of deep learning led to significant improvements in all three dimensions (Bui et al., 2023a; Luo et al., 2020; Tancik et al., 2020; Fernandez et al., 2024).

Yet, despite these techniques, it seems that progress has stagnated. State-of-the-art methods successfully embed around 100−200 bits in a relatively imperceptible way (i.e., Peak Signal-to-Noise Ratio, PSNR, above 40 dB) while robust to perturbations. Improvements in quality and robustness continue, but they are only marginal, leading many to believe we are nearing the limits of what is possible.

Image watermarking indeed may indeed already be a solved problem. Unlike generative or discriminative models that can improve as data and parameters are scaled, watermarking has an inherent performance ceiling. Given an image resolution and a set of robustness constraints, there is a finite amount of information that can be embedded imperceptibly. The existence of this limit and the converging empirical performance of recent models naturally leads to a critical question:

Have we already reached the theoretical ceiling of watermarking performance?

To answer this question, we need to know what this limit actually is and to measure how close our models are to it. We address these challenges in the current paper and offer the following findings:

i. We propose bounds on the capacity of watermarking under a PSNR constraint and robustness to linear augmentations, indicating capacities orders of magnitude larger than seen in practice.
ii. Watermarking models are trained with constraints we cannot directly analyse so we retrain Video Seal (Fernandez et al., 2024), a state-of-the-art image and video watermarking model, to match our simplest theoretical setup: watermarking a single gray image under only a PSNR constraint. Yet, Video Seal fails to encode even 1024 bits, when we successfully encode 2048 bits with a linear model, 32,768 bits by tiling lower-resolution watermarks, and 456,509 bits with a handcrafted model. This indicates severe structural limitations.
iii. With the standard quality and robustness constraints, we train Chunky Seal, a simple scale-up of Video Seal, which embeds 1024 bits while maintaining similar robustness and image quality.

Therefore, our theory and experiments show that

It is possible to achieve much higher capacities than we currently have, although that might require innovation in architectures and training.

2 Related work

Classic principled methods. Early research on image watermarking was dominated by hand-crafted signal processing techniques, grounded in well-understood mathematical and perceptual models. These methods could operate directly in the pixel domain (Van Schyndel et al., 1994; Bas et al., 2002), or in transform domains, most commonly the discrete cosine transform (DCT, Bors and Pitas, 1996; Piva et al., 1997) and the discrete wavelet transform (DWT, Xia et al., 1998; Barni et al., 2001), as well as combinations of the two (Navas et al., 2008; Feng et al., 2010; Zear et al., 2018). A key insight was that perceptually significant frequencies tend to be preserved under transformations (Cox et al., 1997). Other schemes, such as (Ni et al., 2006), introduced perturbations to all pixels with specific values to enable very large payloads. Despite being principled and accompanied by theoretical guarantees, these methods have limited robustness to perturbations and, in some cases, cause noticeable image degradation.

Deep learning based-watermarking. With the development of deep learning techniques for computer vision, it was only natural to extend them to image watermarking. Vukotić et al. (2018) proposed adversarially attacking a fixed image extractors, an idea later built upon by Fernandez et al. (2022) and Kishore et al. (2022). More popular and successful methods train purpose-built neural networks. Early convolutional models, such as those introduced by Mun et al. (2017) and Zhu et al. (2018), established the feasibility of CNN-based watermarking. Subsequently, architectures based on U-Net (Ronneberger et al., 2015) gained prominence, leveraging multi-resolution representations and residual connections to enable greater network depths. Recent work has also explored watermarking in the latent space of diffusion models Bui et al. (2023b). Advances in

perceptual loss design have proven critical: careful loss selection and tuning improved both image quality (Bui et al., 2023a; Fernandez et al., 2024; Xu et al., 2025) and robustness (Tancik et al., 2020; Jia et al., 2021; Pan et al., 2024). Beyond robustness, methods for watermark localization and multi-message embedding have been proposed (Sander et al., 2025; Wang et al., 2024; Zhang et al., 2024), while add-on techniques such as adversarial training (Luo et al., 2020) and attention mechanisms (Zhang et al., 2020) further enhance performance. Although combining image generation with watermarking has also been studied (Fernandez et al., 2023; Wen et al., 2023; Kim et al., 2023; Hong et al., 2024; Ci et al., 2024), such generative approaches are beyond the scope of this work.

Theoretical capacity of watermarking. Most prior attempts for theoretical bounds on the watermarking capacity of images focused on mathematically amenable but detached from reality setups, often assuming Gaussian covers and perturbations. Costa (1983) and Cohen and Lapidoth (2002) showed that under the Gaussian cover and noise assumption, the fact that the decoder does not know the cover image does not affect the capacity: it is determined solely by the strength of the watermark and the channel noise. Chen and Wornell (2002) proposed quantization-based watermarking scheme with guaranteed capacity under Gaussian noise. While Moulin and O'Sullivan (2003) and Moulin and Koetter (2005) are slightly closer to practical setups, i.e., discrete values and distortions other than Gaussian noise, they can only factor robustness to perturbations with small magnitude (similarly to Somekh-Baruch and Merhav (2004)), something that does not apply to most geometric augmentations. Merhav (2005) considers geometric perturbations with large magnitude but only such that can be expressed as a permutation of the pixels. All these works take an information-theory-based approach: power-limited communication over a super-channel with a state that is known to the encoder. Maor and Merhav (2005) studied the connection with compression from an information-theoretic angle but their results cannot be used to obtain practical bounds. By contrast, we take a geometric approach in the current work which we find better suited for studying robustness to more realistic perturbations.

3 Bounds on watermarking capacity

In this section we establish fundamental limits of image watermarking. As watermarking requires every message to be encoded as a distinct image, the problem can be formalized in a geometric framework: images are points on a high-dimensional grid, and capacity is determined by the number of unique such points that do not violate given constraints. Watermarked images must be close to the cover image to be imperceptible and, to remain robust, messages must be represented with sufficient redundancy. Both requirements reduce the number of admissible images available for encoding. We begin with establishing the absolute maximum information that can be represented in an image (Section 3.1). We then add a PSNR constraint in Sections 3.2 and 3.3. We incorporate robustness to transformations such as cropping, rescaling, rotation and JPEG in Section 3.4. Finally, we consider the effect of the data distribution on capacity (Section 3.5).

3.1 Absolute capacity of the image space

Watermarking embeds a message m into an image $\boldsymbol{x}$. Since each message must correspond to a distinct encoded image, the number of unique messages, that is, the watermarking *capacity* is limited by the number of distinct images. An l-bit message requires at least 2^l such images. We represent an image as a vector of length cwh, where c is the number of channels, w is the width and h the height, with each element having 2^k discrete levels when using k-bit colour depth. The tuple (c, w, h, k) defines an *image format*. The set of all possible images in this format is $\mathcal{I} = \{0, 1, \ldots, \rho\}^{cwh}$ with $\rho = 2^k - 1$, which can be thought of as a finite grid[1] of integer points in $\mathbb{R}^{cwh}$. This immediately gives us a trivial upper bound on watermarking capacity: since each message must correspond to a distinct watermarked image, it is not possible to embed more messages than there are distinct images.

Bound 1: Absolute capacity of the image. The capacity of images in the format (c, w, h, k) is

$$\texttt{capacity}[\text{in bits}] = \log_2 |\mathcal{I}| = \log_2 \left((2^k)^{cwh}\right) = cwhk \text{ bits.}$$

[1]The set of valid images is a finite subset of a lattice in $\mathbb{R}^{cwh}$, i.e., $\mathcal{I}{=}[0,\rho]^{cwh}{\cap}\{\sum_{i=1}^{cwh} a_i \boldsymbol{e}_i \mid a_i{\in}\mathbb{Z}\}$ with $\boldsymbol{e}_i$ the i-th unit vector, but we use the term *grid* for readability since no deeper lattice theory is required.

(a) Cube fully inside sphere (low PSNR, Bound 2)

(b) Sphere fully inside cube (high PSNR, Bounds 3 and 4)

(c) Non-trivial intersection (medium PSNR, Bounds 5 and 6)

(d) Sphere at corner, i.e., arbitrary image (Section 3.3)

Figure 2 The box-ball configurations of the PSNR-only constraint. The cube $C_{\mathcal{I}}$ represents the set of all images and the sphere is the PSNR ball centred at the cover. Their intersection determines the set of feasible watermarked images, with the cardinality being the watermarking capacity. **(a)**, **(b)** and **(c)** are the cases with the cover image $\boldsymbol{x}$ at the centre of the cube $C_{\mathcal{I}}$ (gray image, resulting in highest capacity, Section 3.2). **(d)** is the case of the worst-case cover $\boldsymbol{x}$, i.e., at the corner of $C_{\mathcal{I}}$ (Section 3.3).

Bound 1 simply states that the maximum number of embeddable bits is the uncompressed size of the image in bits, i.e., 1.57 Mbits in a 256×256px colour image. We next introduce imperceptibility and robustness constraints to measure their effect on capacity.

3.2 Capacity under a PSNR constraint

In practice, watermarked images must remain visually close to the cover image, ideally imperceptibly so. This limits the set of admissible images for encoding messages. A standard way to quantify distortion is the *peak signal-to-noise ratio (PSNR)*, measured in dB. Requiring a minimum PSNR τ between the cover x and the watermarked image $\tilde{x}$ is equivalent to bounding their ℓ_2 distance (see App. A for the full derivation):

$$\mathrm{PSNR}(x, \tilde{x}) \;\geq\; \tau \quad\Longleftrightarrow\quad \|x - \tilde{x}\|_2 \;\leq\; \epsilon(\tau), \;\text{ with } \epsilon(\tau) = \rho\sqrt{cwh}\, 10^{-\tau/20}. \tag{1}$$

Interpreting PSNR as an ℓ_2-ball constraint gives us an avenue for measuring the message-carrying capacity under it by considering the amount of integer points inside both the cube and this ball. Counting how many such points exist is not trivial, and we analyse the three possible cases (see Figure 2): *i.* the ball is so large that it contains the entire cube (very low τ); *ii.* the ball is small enough to lie fully inside the cube (high τ); *iii.* the ball and cube partially overlap (medium τ). We begin by assuming the cover image $\boldsymbol{x}$ lies at the centre of the admissible range, i.e., $\boldsymbol{x}_g = 2^{k-1}\,\mathbf{1}$ as then the volume of the intersection (and thus the capacity) is maximized. In Section 3.3 we will extend the analysis to arbitrary images.

3.2.1 Cube in ball (low PSNR)

When τ is low, $\epsilon(\tau)$ is large and the ball contains the entire cube $C_{\mathcal{I}} = [0, \rho]^{cwh}$. The PSNR constraint does not rule out any images, so the capacity is just the absolute maximum (Bound 1):

Bound 2: Gray image, PSNR constraint (low PSNR). The capacity of a gray image $\boldsymbol{x}_g$ under a very low minimum PSNR threshold τ is

$$\texttt{capacity}[\text{in bits}] = cwhk.$$

Bound validity: When $\epsilon(\tau) \geq \rho/2\sqrt{cwh}$, or equivalently when $\tau \leq 20 \log_{10} 2 \approx 6.02\,\mathrm{dB}$.

3.2.2 Ball in cube (high PSNR)

When τ is high, the PSNR ball is fully inside the cube. The capacity is the number of integer points inside the ball. This amounts to counting the integer grid points in an n-ball of radius $\epsilon(\tau)$, a problem with no general closed form. In high dimensions cwh and for sufficiently large radii $\epsilon(\tau)$, this is well approximated by the ball volume $\mathrm{Vol}\, B_{cwh}\,[\cdot, \epsilon(\tau)]$. See Section B for details on the validity of this approximation.

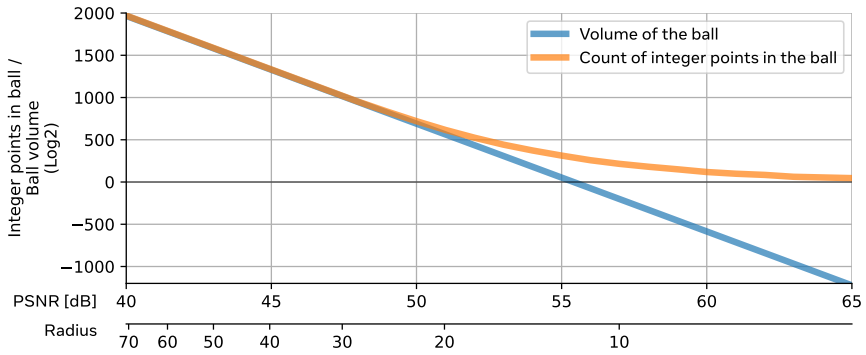

Figure 3 Discrepancy between the volume of a ball and the number of integer lattice points contained in it for small $\epsilon(\tau)$. When the radius is small, the volume-based approximation Bound 3 underesimates the actual number of integer points. In such cases we use the exact count Bound 4 instead. Evaluated for a $16 \times 16 \times 3 = 768$-dimensional ball.

Bound 3: Gray image, PSNR constraint (high PSNR, volume approximation). The capacity of a gray image $\boldsymbol{x}_g$ under a high minimum PSNR threshold τ is approximately

$$\begin{aligned}\texttt{capacity}[\text{in bits}] &\approx \log_2 \operatorname{Vol} B_{cwh}\left[\cdot, \epsilon(\tau)\right] \\ &= \log_2 \frac{\pi^{cwh/2}\,\epsilon(\tau)^{cwh}}{\Gamma\left(\frac{cwh}{2}+1\right)}.\end{aligned}$$

Bound validity: When the ball is fully inside the cube, i.e.$\epsilon(\tau) \leq \rho/2$ (i.e., $\tau \geq 20\log_{10}(2\sqrt{cwh})$) and $\epsilon(\tau)$ large enough for accurate volume approximation (see Bound 8 for small ϵ).

For small radii the volume approximation becomes inaccurate. However, then there are relatively few integer points in the ball and we can explicitly count them, as long as the dimension cwh of the ambient space is not too high. Instead of brute-force enumeration (which scales poorly), we use a method introduced by Mitchell (1966) leveraging symmetries for efficient counting (see Algorithm 2).

Bound 4: Gray image, PSNR constraint (high PSNR, exact count). For small $\epsilon(\tau)$ the capacity is

$$\texttt{capacity}[\text{in bits}] \;=\; \log_2 \texttt{PointsInHypersphereMitchell}(\texttt{dim} = cwh, \texttt{radius} = \epsilon(\tau)).$$

Bound validity: When $\epsilon(\tau) \leq \rho/2$ (i.e., $\tau \geq 20\log_{10}(2\sqrt{cwh})$) and $\epsilon(\tau)$ small enough that exact counting is computationally feasible.

We use Bound 4 whenever we can evaluate Algorithm 2 in reasonable time, and otherwise Bound 3. As shown in Figure 3, the transition between the two regimes is smooth.

3.2.3 Non-trivial intersection (medium PSNR)

For intermediate PSNR values τ, $B_{cwh}\left[\boldsymbol{x}_g, \epsilon(\tau)\right]$ and $C_{\mathcal{I}}$ intersect non-trivially. We can approximate this count by the volume of the intersection, using the same volume-based method as in Bound 3. One can use exact volume computation (see Bound 5 in Section D), though this tends to be numerically unstable. In practice, a simpler upper bound approximates it well by taking the minimum of the two trivial cases:

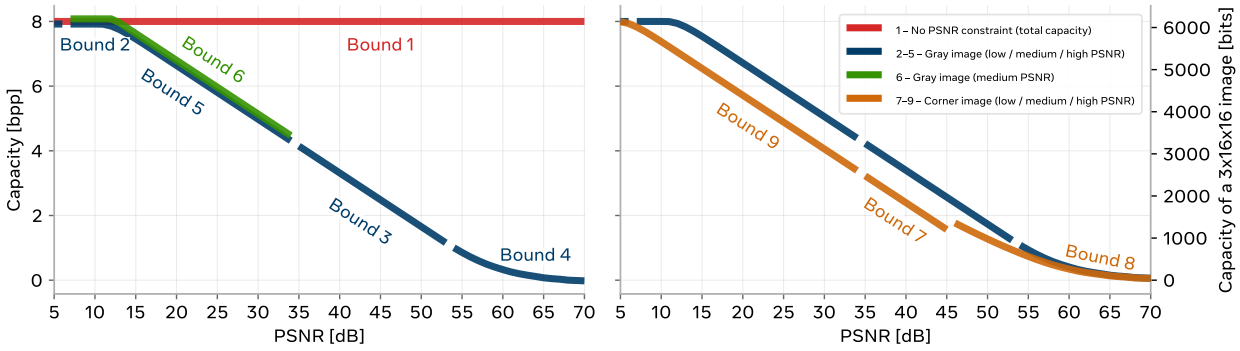

Figure 4 Watermarking capacity under PSNR constraints for a 16×16px 3-channel cover image without robustness requirements. We show the case where the cover lies at the centre of the pixel range (left, Sections 3.1 and 3.2) and at the extreme corner where pixels are saturated (right, Section 3.3). In both cases we plot the family of bounds we established ranging from the trivial total capacity (Bound 1) to volume approximations (Bounds 3 and 7), exact lattice counts (Bounds 4 and 8), and numerical integration for partial overlap (Bounds 5 and 9). In the right panel, the arbitrary cover image case lies below the centred gray image case by at most cwh bits i.e., at most one bit per pixel (1 bpp). The discontinuity between Bounds 7 and 8 is due to the volume approximation undercounting the integer points on the faces of the cube.

Bound 6: Gray image, PSNR constraint (medium PSNR, approximation). The capacity of a gray image under minimum PSNR τ is upper-bounded by

$$\text{capacity}[\text{in bits}] \leq \min\left[\text{Bound 2}, \text{Bound 3}\right].$$

Bound validity: When $\rho/2 \leq \epsilon(\tau) \leq \rho/2\sqrt{cwh}$, or equivalently $20\log_{10} 2 \leq \tau \leq 20\log_{10}(2\sqrt{cwh})$.

As shown in Fig. 4 left, this simple upper Bound 6 closely tracks the exact Bound 5. Thus Bound 6 is the practical choice going forward, while Bound 5 is provided in the appendix for completeness. Figure 4 left illustrates all the bounds from this section for a 16×16px image. At 45 dB these bounds give us roughly 2000 bits of capacity (more than 2.5 bpp): orders of magnitude more than the 0.001 bpp we see in practice (Figure 1). For the more reasonable 256×256px watermarking resolution, that maps to roughly 500,000 bits.

3.3 From central gray image to arbitrary cover images

In Section 3.2 we assumed the cover lies at the centre of the pixel range, thereby maximizing the volume of the intersection between the PSNR ball and the cube $C_{\mathcal{I}}$. Real images, however, may be anywhere in $C_{\mathcal{I}}$. Being at the corner of $C_{\mathcal{I}}$ minimizes overlap with the ball and thus provides a lower bound valid for any image. When ϵ is not too large, exactly $1/2^{cwh}$ of the PSNR ball centred at a corner of $C_{\mathcal{I}}$ remains inside $C_{\mathcal{I}}$. Although this may seem drastic, the penalty is in fact modest: at most cwh bits, i.e., one bit per pixel. In Section E we provide the formal bounds for this corner setting. Bound 7 adapts Bound 3, the volume approximation when the ball is fully in the cube. Bound 8 is the analogue of Bound 4, i.e., exact counting for small $\epsilon(\tau)$. Bound 9 parallels Bound 5 for the case when numerical integration is needed. As shown in Figure 4, the gap from the gray-only image bounds is at most 1 bpp, thus:

Watermarking with a PSNR constraint should allow for capacity upwards of 2 bpp and does not explain the low capacities we observe in practice.

3.4 Adding robustness constraints

In practice, watermarking must balance imperceptibility with robustness: the message should survive common processing, like compression, resizing, cropping, rotation, etc. We wish to quantify the capacity reduction due to the robustness constraints. We consider linear transformations, which encompass most transformations used in practice. We also develop LinJPEG, a linearized version of JPEG, allowing us to study the effects of compression in the same setting (see Section F.4 for the construction).

Table 1 Conservative capacity bounds under robustness constraints for PSNR 42 dB. These values are calculated via Bound 13 and are strongly conservative lower bounds on the capacity that is achievable while maintaining robustness to the respective transformations and PSNR under 42 dB.

Augmentation	bpp	Conservative capacity for 16×16px	Conservative capacity for 256×256px
Horizontal Flip	3.064	2,352 bits	602,353 bits
Crop&Rescale 25%	0.107	81 bits	20,958 bits
Crop&Rescale 50%	0.015	11 bits	3,013 bits
Crop&Rescale 75%	0.005	3 bits	904 bits
LinJPEG q=8	0.134	102 bits	26,273 bits
LinJPEG q=10	0.136	104 bits	26,757 bits
LinJPEG q=15	0.137	105 bits	27,020 bits
Rotation 15deg	0.084	64 bits	16,502 bits
Rotation 30deg	0.075	57 bits	14,676 bits
Rotation 45deg	0.083	64 bits	16,401 bits

Take a linear transformation $M \in \mathbb{R}^{cwh \times cwh}$ that maps an image $\boldsymbol{x}$ to a transformed $M\boldsymbol{x}$ and a quantization operation Q (typically an element-wise rounding or floor operation) to map the pixel values of $M\boldsymbol{x}$ to the valid images $\mathcal{I}$. Hence, we have the final transformed image $\boldsymbol{x}' = \mathsf{Q}[M\boldsymbol{x}]$. We need to find the subset of the possible watermarked images under only the PSNR constraint that map to unique valid images after applying M and Q to them. The main complication in this setup is that Q is non-linear.

Heuristic bounds. A simple approach is to take a volumetric approach akin to Bounds 3, 5, 7 and 9. This gives rise to a set of relatively simple heuristic bounds. We factor in how M changes the volume and account for directions compressed by the transformation which destroy capacity as different watermarked images get collapsed together. We also account for directions fully collapsed by M when it is singular. Finally, the stretched directions might result in some watermarked images being outside $C_{\mathcal{I}}$ after the transformation, leading to them being clipped. Bounds 10 to 12 use a heuristic based on the singular values of M to account for the effect on capacity. Refer to Section F.2 for details on their derivation and properties. In Figure 6 we plot these bounds for robustness to rotation, cropping and LinJPEG, showing that even under the most aggressive cropping, we should expect around 0.5 bpp or almost 100,000 bits for 256×256px images.

Conservative bounds. We can show cases where these heuristic bounds under-approximate and cases where they over-approximate the true capacity, e.g., Figures 9 and 10. Thus, the true capacity under linear transformation could be much lower than these bounds predict. To ensure that this is not the case, we develop an actual lower bound: Bound 13. While we reserve the details for Section F.3, this bound is based on over-approximating the set of images that can be quantized by Q to the same image after M is applied to them. As a result, Bound 13 is extremely conservative and unrealistic. We believe that despite Bounds 10 to 12 not being valid lower bounds, they are much closer to the true capacity. Still, we report the conservative bound in Table 1: the most aggressive crop still leaves at least 904 bits for 256×256px images. For the other augmentations, the conservative capacity is much higher. Therefore,

Robustness to geometric transformations and compression significantly reduces the capacity but cannot fully explain the low watermarking capacity of current models.

3.5 From single cover images to datasets and data distributions

In a blind watermarking setup, the decoder must operate without access to the original cover image, creating potential collisions: if multiple natural images (i.e., potential covers) are very close to each other in pixel space, a watermarked version of one cover could be identical to a watermarked version of another. To prevent such ambiguity, the total set of watermarked images within a given region (like the PSNR ball) must be partitioned among all the potential covers it contains. If there are N possible covers, the capacity for each is reduced by $\log_2(N)$ bits. We estimate N using neural compression models like VQ-VAE (Van Den Oord et al., 2017) and VQGAN (Esser et al., 2021), which upper-bound the number of perceptually distinct images. For instance, a 256×256px image can be compressed into a 32×32 latent with a 1024-entry codebook (Muckley

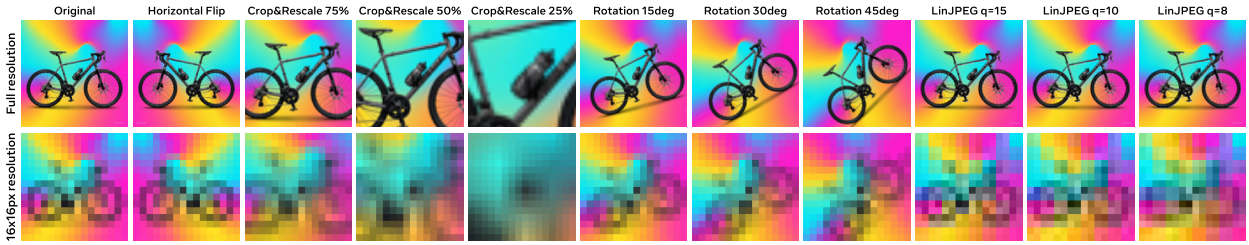

Figure 5 Robustness constraints considered in this paper. The figure illustrates the linear transformations we evaluate: Horizontal Flip, Crop&Rescale, Rotation and Linearlized JPEG compression. We show them both at full resolution and at 16×16px at which we compute the bounds in Section 3.4.

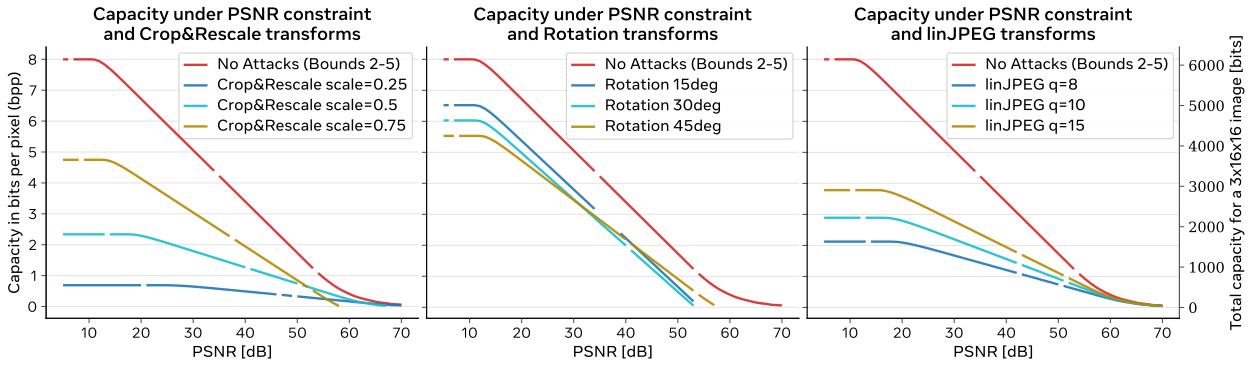

Figure 6 Impact of robustness constraints on watermarking capacity. Capacity (in bits per pixel) under a PSNR constraint is shown for three families of transformations: *Crop&Rescale* (left), *Rotation* (center), and *Linearized JPEG* (right), using the heuristic bounds (Bounds 10 to 12). The red lines show the PSNR-only capacity bounds without robustness constraints Bounds 2 to 5. Each transformation reduces capacity in proportion to its severity: smaller crop scales, larger rotations, or lower JPEG quality factors. Across all cases, robustness constraints reduce but do not eliminate the large theoretical capacity gap with current watermarking methods.

et al., 2023). This representation can express at most $1024^{32\times32} = 2^{10240}$ distinct images. Conservatively assuming all could fall in the PSNR ball of the considered image, capacity is reduced by $10,240$ bits, or about 0.05 bpp, on top of the 1 bpp loss from Section 3.3. Thus, from this perspective,

The data distribution has only a negligible effect on watermarking capacity and cannot explain the low performance of current models.

This aligns with prior findings for Gaussian channels that decoder knowledge of the cover does not affect capacity (Costa, 1983; Chen and Wornell, 2002; Moulin and O'Sullivan, 2003).

4 Empirical performance is much lower than predicted

Section 3 showed that capacities of over 2 bpp at PSNR of 40 dB without robustness constraints, and of 0.5 bpp with robustness, are possible. Even under the very conservative Bound 13 we still would expect capacities of at least 0.01 bpp. However, in practice, the models reported in the literature have significantly lower capacities (less than 0.001 bpp, Figure 1). To understand the cause of this gap, this section asks:

Are existing models significantly under-performing relative to what is possible in practice, or are our bounds too unrealistic?

There are five possible explanations of the large discrepancy between the performance we see in practice (Figure 1) and the bounds in Section 3:

***A*. Real models might be near-optimal if we consider advanced robustness constraints.** Real models are trained with combinations of more complex robustness constraints than what our theory considers. Possibly, if

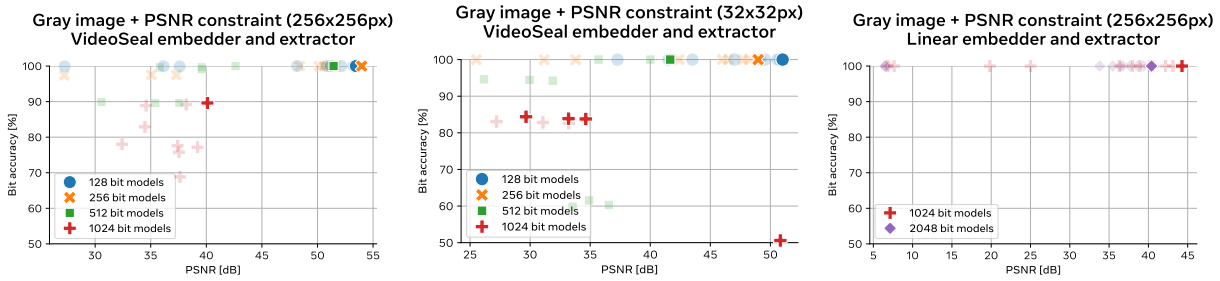

Figure 7 Video Seal fails to learn how to embed 1024 bits into a gray image with only PSNR constraint, whereas a linear embedder and extractor learn how to embed 2048 bits. *(Left.)* Video Seal trained on a single solid gray image with only the detector and MSE losses. It learns to embed up to 512 bits but fails to embed 1024 bits. *(Centre.)* Same setup as the left plot but trained on a reduced 32×32px resolution. The performance is similar; hence, the Video Seal architecture fails to make use of the full resolution. *(Right.)* Replacing the embedder and decoder of Video Seal with a single linear layer each achieves 100% bit accuracy and PSNR above 40 dB for both 1024 and 2048 bits: demonstrating that Video Seal indeed has structural limitations. The results for the sweeps over the learning rate and λ_i are shown, with the best models highlighted.

we could factor these additional robustness constraints, our bounds would also show much lower possible capacity.

B. **Real models might be near-optimal if we consider advanced perceptual constraints.** Real models satisfy advanced perceptual constraints beyond just PSNR, e.g., SSIM (Wang et al., 2004) or LPIPS (Zhang et al., 2018). Possibly, if we could factor these perceptual constraints, our bounds would also show much lower possible capacity.

C. **Real models might be near-optimal if we consider real-world image distributions.** Real models are trained and evaluated on real-world image distributions, not a single fixed image. We argued in Section 3.5 why that should not be a problem, but perhaps we are wrong and the need for disambiguating the cover image reduces the watermarking capacity much more than expected.

D. **Our bounds overestimate capacity and cannot be approached empirically.**

E. **We can do much better and push the Pareto front well beyond the current state-of-the-art.** The architecture and training setups, or compute, we use in practice are not good enough to fully utilise the watermarking capacity of the images, even when factoring in for advanced robustness, perceptual and data distribution constraints.

To understand the cause of the gap between theoretical and real-world performance in image watermarking, we need to find out which of these hypotheses is the underlying cause. If it is **A.**, **B.**, **C.**, **D.**, or a combination of them, then it is possible that, indeed, the best current models are close to what is ultimately possible and we can expect only marginal further improvements. On the other hand, if the cause is **E.**, then that means that there is plenty of space for significant improvements.

4.1 The real-world complexity does not explain the performance gap

Let's first address cases **A.**, **B.**, **C.**, i.e., that our bounds cannot capture the complexity of the robustness, quality and data constraint with which real models are trained. While we cannot bring the real-world complexity to our analytical bounds, we can bring the models to the simplified theoretical setup.

More concretely, we take the simplest of setups: a single gray image with a PSNR constraint, as in Section 3.2. We will use Video Seal as the base for our experiments (Fernandez et al., 2024), originally introduced as an image watermarking model with frame copying that generalizes to video. It was first demonstrated with a 96-bit capacity and was recently extended to a 256-bit open-source version, which we use as the strongest available baseline. To match the setup of Section 3.2, we replace the dataset with a single solid gray image, remove all perceptual constraints but the MSE loss and remove all augmentations. We first retrain it for $n_{\text{bits}} = 128$, 256, 512, and 1024 bits. We have hereby reduced the task to simply find a way to encode n_{bits} into a single fixed image. From Figure 4 we expect capacities of around 600,000 bits at 40 dB in this setup. Thus, the model should easily learn these much lower n_{bits}. We train with AdamW (Loshchilov and Hutter, 2019)

Table 2 Video Seal fails to learn how to embed 1024 bits into a gray image with only PSNR constraint while a linear embedder and extractor learn how to embed 2048 bits. Numerical results for the best-performing runs from Figure 7 and their respective hyperparameters, as well as the handcrafted embedder/decoder from Equation (2) at four representative PSNR values.

	Message size	Message size if tiled to 256x256px	PSNR	Bit acc.	λ_i	lr
VideoSeal (256x256px, 600 epochs)	128 bits		53.45 dB	100.00%	0.5	5e-4
	256 bits		53.98 dB	100.00%	1.0	5e-4
	512 bits		51.45 dB	100.00%	0.5	5e-4
	1024 bits		40.10 dB	89.63%	1.0	5e-5
VideoSeal (32x32px, 600 epochs)	128 bits	8192 bits	51.02 dB	100.00%	1.0	5e-4
	256 bits	16384 bits	48.98 dB	100.00%	1.0	5e-4
	512 bits	32768 bits	41.66 dB	100.00%	1.0	5e-5
	1024 bits	65536 bits	29.66 dB	84.39%	0.1	5e-5
	1024 bits	65536 bits	33.20 dB	83.86%	0.5	5e-5
	1024 bits	65536 bits	34.63 dB	83.78%	1.0	5e-5
	1024 bits	65536 bits	50.83 dB	50.60%	0.5	5e-4
Linear (256x256px, 50 epochs)	1024 bits		44.28 dB	100.00%	20.0	5e-4
	2048 bits		40.40 dB	100.00%	12.0	5e-4
Handcrafted	623232 bits		36.00 dB	100.00%		
	551948 bits		38.00 dB	100.00%		
	456509 bits		42.00 dB	100.00%		
	311616 bits		48.00 dB	100.00%		

with batch size 256 for 600 epochs, 1000 batches per epoch, cosine learning rate schedule with a 20-epoch warm-up, similarly to Video Seal. We sweep over the learning rate (5e-4, 5e-5, 5e-6) and λ_i, the MSE loss weight (0.1, 0.5, 1.0), with LR=5e-5 and $\lambda_i = 0.5$ being the values used for training Video Seal.

The results of training Video Seal on a single gray image can be seen in Figure 7 left and Table 2. There are runs for the 128, 256 and 512 bit models that do achieve 100% bit accuracy and PSNR values above 42dB. However, Video Seal cannot even get to 1024 bits, far from what we expect from the bounds. This is surprising: the model cannot approach the theoretical bounds even after removing the complexities that supposedly make watermarking difficult. This means that neither **A.**, **B.** nor **C.** can explain why we see such a gap between the theoretical and real-world performance.

4.2 Our simplest bounds are achievable, yet models struggle to get near them

Section 4.1 showed that Video Seal cannot match the capacity predicted by the bounds in Section 3.2 even when trained only on a single gray image and with no augmentations. Thus, the complexity of real world watermarking cannot explain the gap between the theoretical and real-world performance. This leaves us with two options: **D.** our bounds are wrong and unachievable, or **E.** our models are under-performing. There are a couple simple experiments that can demonstrate that we can get much closer to the bounds in Section 3.2 and hence **D.** also does not explain the gap.

Linear embedder and extractor. We trained a simple linear embedder and extractor. The embedder gets the 1024 bit message (shifted and scaled to -1 and $+1$ values) and produces a $256\times256\times3$ watermark residual which gets added to the original gray image. Similarly, the decoder is a linear layer from the flattened $256\times256\times3$ image to 1024 outputs, which are thresholded to recover the message. We train only for 50 epochs, with the same learning rate values and $\lambda_i \in \{4, 8, 12, 20\}$.

The results in Figure 7 right and Table 2 show the linear layer learns what Video Seal could not: 100% bit accuracy for 1024 bits with PSNR of 44 dB. We also trained a linear model for 2048 bits which achieved 100% bit accuracy. This shows that capacities beyond 512 bits are possible in practice (at least for a gray image and no robustness) and are learnable via gradient descent. All one needs is the right architecture.

Lower resolution training. The capacity we get from Video Seal in this setting is equivalent to what we would

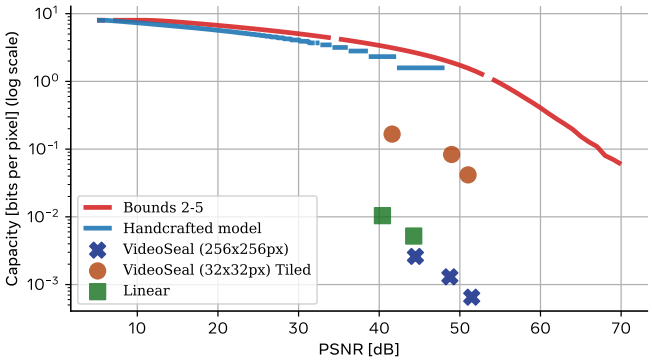

Figure 8 Simple models outperform Video Seal on a gray image with only a PSNR constraint. Experiments from Section 4 compare our theoretical bounds (Section 3.2) against trained models. Video Seal falls well below the predictions, while a linear model performs slightly better and a tiled 32×32px Video Seal is even better. Our handcrafted model nearly matches the bound.

expect at much lower resolutions, namely around 20×20px, indicating that it fails to utilize the available resolution. We check that by training it in the setup of Section 4.1 but at the much lower 32×32px resolution. We sweeped over the same learning rates and λ_i values, and again trained for 600 epochs. As can be seen from Figure 7 centre and Table 2, the performance at 32×32px is very similar to the one at 256×256px: the 512 bit model has 100% bit accuracy with 41.7 dB, despite having 64× less pixels to use. Hence our architecture fails to fully utilize the available capacity.

We can use this 32×32px model to demonstrate that much higher capacities are indeed possible. As we do not consider robustness to any geometric or valuemetric transformations, simply tiling the lower-resolution watermarks can produce a watermark with much higher capacity at the same PSNR. Therefore, if we take the 256×256px resolution as reference, we get 64× the capacity at 32×32px. Hence, if we tile the 512 bit model which achieved 100% bit accuracy at 41.7 dB, we get a model with the impressive 32,768 bits bits of capacity and still 41.7 dB (as PSNR is resolution-independent). Therefore, we are getting effective capacity of 32,768 bits by simply tiling the 32×32px model. That is much closer to our bound of roughly 600,000 bits but still only about 0.167 bpp.

Handcrafted embedder and extractor. We can do even better by manually crafting an embedder and extractor. The key observation is that mapping a hypercube to binary messages is easy. Take the ball of radius $\epsilon(\tau) = \rho\sqrt{cwh}\ 10^{-\tau/20}$ from Equation (1). The half-side of the largest cube that can fit in this ball is $d = \epsilon(\tau)/\sqrt{cwh} = \rho\ 10^{-\tau/20}$. We have that each edge of the box contains a cwh-dimensional grid of $q = 2\lfloor d \rfloor + 1 = 2\lfloor 2^k\ 10^{-\tau/20} \rfloor + 1$ points per side. Hence, that gives us total capacity in bits of

$$\log_2 \left[\left(2 \left\lfloor 2^k\ 10^{-\tau/20} \right\rfloor + 1 \right)^{cwh} \right] = cwh \log_2 \left[2 \left\lfloor 2^k\ 10^{-\tau/20} \right\rfloor + 1 \right] = cwh \log_2 q, \tag{2}$$

or $\log_2 q$ bits per pixel. See Figure 8 for a plot of that for different PSNR values. For 42 dB, and images of 256×256px that gives us a capacity of 456,509 bits (see Table 2) almost 14× what we could embed with the 32×32px tiling approach. Moreover, it gets us close to the theoretical bound. This scheme is easy to encode and decode: convert the binary string to q-nary representation and then map each digit in this representation to one of the axes of the box. The decoding is just as easy: get each digit by measuring where the watermark stands along each dimension, combine into a q-nary representation and convert back to binary.

Therefore, we can get much closer to the boundary, at least in the solid gray image case with PSNR constraint and no robustness requirements. Thus, case **D.**, that our bounds are wrong and impossible to achieve, is unlikely. This leaves us with one possible explanation as to why models in practice do not exhibit performance anywhere near what our theory predicts. That would be option **E:**

Our models are likely significantly underperforming relative to what is possible in practice.
We likely can do much better and push the Pareto front well beyond the current state-of-the-art.

The architectures we use might have the wrong inductive biases or poor hyperparameter choices, our datasets might not be big enough, we might not be training long enough, or our models might be too small. Whatever the specific cause, image watermarking models clearly have significant room for performance improvement.

	Chunky Seal (ours)	Video Seal 256bits
Capacity	1024 bits 0.0052 bpp	256 bits 0.0013 bpp
Embedder size	1022.7M	11.0M
Extractor size	773.7M	33.0M
PSNR ↑	45.32±2.16	44.42±2.21
SSIM ↑	0.995±0.006	0.996±0.003
MS-SSIM ↑	0.997±0.002	0.997±0.001
LPIPS ↓	0.0085±0.0067	0.0019±0.0011
Bit acc. Identity	99.74±0.28%	99.90±0.21%
Bit acc. Flip	99.65±0.34%	99.89±0.24%
Bit acc. Rotate (≤10°)	98.27±2.10%	98.84±1.10%
Bit acc. Resize (71–95%)	99.74±0.28%	99.90±0.21%
Bit acc. Crop (77–95%)	98.25±1.75%	98.04±1.57%
Bit acc. Brightness (0.5–1.5×)	98.99±1.87%	98.67±2.67%
Bit acc. Contrast (0.5–1.5×)	99.54±0.51%	99.56±0.45%
Bit acc. JPEG (Q 50–80)	98.79±0.75%	99.74±0.47%
Bit acc. Gaussian Blur (k≤9)	99.74±0.28%	99.90±0.22%
Bit acc. Overall	99.15±0.63%	99.31±0.60%

Table 3 Chunky Seal performance on images from SA-1B (Kirillov et al., 2023) at their original resolution. Chunky Seal has much higher capacity (1024 bits) than Video Seal while preserving its image quality and robustness on a wide variety of transformations. The improvement is driven by scaling the model size and its training. Extended results on SA-1B (Kirillov et al., 2023) and COCO (Lin et al., 2014) as well as qualitative results, are reported in Section I.1

5 Better performance in practice is possible: Chunky Seal

While it remains possible that current models approach a theoretical limit under robustness and quality constraints, training a watermarking model with comparable quality and robustness but with substantially higher capacity would decisively rule this out. We take Video Seal (Fernandez et al., 2024) as the base model and train it for 1024 bits. We increased the embedding dimension to 2048, the U-Net channel multipliers from $[1, 2, 4, 8]$ to $[4, 8, 16, 32]$, and enabled watermarking in all three channels, not just the luma (Y) channel. This results in an embedder 90× larger than the original Video Seal embedder. The ConvNeXt (Liu et al., 2022) extractor was similarly scaled: we increased the depths for each stage from $[3, 3, 9, 3]$ (as in ConvNeXt-tiny) to $[3, 3, 27, 3]$ (as in ConvNeXt-base), with their dimensions increased from $[96, 192, 384, 768]$ to $[256, 512, 1024, 2048]$. The stride of the first layer was reduced from 4 to 2. This results in an extractor that is 23× larger than the original Video Seal extractor. Due to its significantly increased size, we name this model Chunky Seal. We train it at the original 256×256px resolution. We apply gradient clipping with a maximum norm of 0.01, which proved critical for stabilizing training.

As shown in Table 3, Chunky Seal shows image quality and robustness comparable to Video Seal across a wide range of distortions, while providing a **4× higher message capacity** (1024 vs. 256 bits). Despite its much larger capacity, Chunky Seal maintains nearly identical image quality across all metrics, and only slightly higher LPIPS. The robustness results further confirm that Chunky Seal sustains high bit-accuracy across transformations such as rotation, resizing, cropping, brightness and contrast changes, JPEG compression, and blurring, closely matching Video Seal. We emphasize that these results were achieved *without hyperparameter tuning*, whereas Video Seal was extensively optimized for quality and robustness.

Achieving 4× the capacity per pixel with comparable robustness and quality through simple scaling strongly suggests that substantially higher capacities are within reach using improved architectures and training strategies.

6 Discussion

Given the theoretical and empirical results outlined in the previous sections and the nuanced interpretations, we would like to discuss some of the implications and the limitations of our findings.

Why maximise capacity? One could argue that we don't need kilobytes of watermarking capacity; the roughly 256 bits we currently have is plenty. However, higher capacities can open up new avenues for content provenance. For example, rather than using watermarks to encode a hash of a C2PA manifest so that we can recover

it from a trusted third-party database (Collomosse and Parsons, 2024), we can instead encode the whole manifest itself, removing the need for registries.

Still, the implications of this paper are not limited to increasing the capacity of watermarking. Depending on the application, one might have a fixed length payload and robustness constraints and might want to maximise image quality; or fixed capacity and image quality and wanting to maximise robustness. We did not explicitly look at these cases but the inherent trade-offs between capacity, robustness and quality imply that if we succeed at significantly improving one of these axes, we can trade that improvement for the other two. In other words, by showing that we should be able to achieve orders of magnitude higher capacities, keeping the image quality and robustness fixed, it is not inconceivable that one would then be able to significantly increase image quality and/or robustness while maintaining capacity.

Implications on the nature of image data. In this work, we argue that the intrinsic capacity for embedding messages in images is significantly higher than current methods suggest. This proposition has two important implications. First, it confirms that the intrinsic dimensionality of the natural image data manifold is, in fact, low. Second, it suggests that neural compression techniques could achieve much higher compression rates than what is currently possible. Furthermore, our finding that watermarking capacity is an order of magnitude larger than even the highest-capacity state-of-the-art models helps explain the feasibility of applying and detecting multiple watermarks on a single image, as demonstrated in (Petrov et al., 2025).

Why doesn't deep learning get us anywhere close to the bounds? Despite Chunky Seal obtaining much higher capacities than previous watermarking models, it nevertheless is still very far from the bounds for even the most aggressive augmentations, as illustrated in Figure 1. This raises the question of why is it so difficult to get close to the bounds. From the controlled setup experiments in Section 4 we saw that the causes are likely not the data distribution (Video Seal on a single gray image did not get anywhere close to the bounds), the resolution (Video Seal performed similarly at lower resolutions), or the augmentations (Video Seal performed similarly when augmentations were removed). That left the model architecture or other aspects of the training as unsuitable. Chunky Seal showed that scaling the model size might be part of the solution, but even that had only marginal success. The fact that simple linear embedder and extractors outperform Video Seal in the simple setup of Section 4.2 indicates that we do have an architecture problem. This is not surprising as the embedder-extractors pair is effectively learning an identity map, something notoriously difficult for neural networks (He et al., 2016; Hardt and Ma, 2017). Therefore, we need more radical innovation in architectures, losses and training pipelines.

Towards principled watermark development, testing and benchmarking. Section 4 illustrated some unexpected phenomena in Video Seal which likely also exist in other models. Namely, increasing the watermarking resolution would not enable higher capacity, nor would removing the augmentations. Hence, we propose a set of sanity checks that we believe a new generation of *principled* watermarking methods should pass.

When trained on solid gray images with no augmentations, a *principled* watermarking method should have its capacity in bits increasing linearly with the number of pixels, should decrease linearly with increasing PSNR and should reach at least the performance of the linear and the handcrafted baselines from Section 4. When trained on gray images with a single augmentation, its capacity should decrease as predicted with stronger augmentations, e.g., cropping to $1/4$ of the area should lead to a 4-fold reduction in capacity and LinJPEG with quality 15 should lead to a 2-fold reduction. These properties are not a guarantee for good performance of watermarking in practice but necessary conditions to be (close to) Pareto-optimal. We hope that by aiming to satisfy these properties, we, as a community, will produce new generation of watermarking models with orders of magnitude higher capacity and/or significantly better image quality.

Limitations. In spite of our attempt to be as comprehensive as possible, a few limitations remain in the analysis and experiments in the present paper. For one, we only consider image watermarking. While our results likely extend to video (which might have even larger capacities due to the temporal dimension), it is not clear what the bounds on audio or text watermarking would be. Our theoretical bounds in Section 3 were limited to the setups we could study analytically and thus miss many aspects used in practice, such as combinations of transformations, advanced image quality techniques and accurate assessment of the effect of data distribution and the model size. Therefore, this leaves room for more advanced theoretical bounds that can be developed and that would be of significant value. Some of our bounds require numerical integration in very high dimensions and hence cannot be accurately evaluated at higher resolutions. While we resorted

to using bits per pixel values computed at the manageable 16×16px resolution, better numerical methods could be beneficial. Our main robustness bounds are also heuristics rather than formal upper bounds. While we do also have valid upper bounds they are extremely conservative and unrealistic, leaving plenty of space for improvement. Finally, while Chunky Seal does outperform Video Seal, it is also much larger and induces significant latencies which make it difficult to deploy at scale or on-device. Therefore, we need to achieve better watermarking performance, ideally without needing significant scaling up of our models.

Acknowledgments

This work would not have been possible without the feedback and support of Alex Mourachko, Sylvestre-Alvise Rebuffi, Tuan Tran and Valeriu Lacatusu. We would like to especially thank Yoshinori Aono, for answering all our questions and assisting with implementation of some of his results. We thank Matthew Muckley and Karen Ullrich for early discussions about this project. The authors are also grateful to Teddy Furon for pointing us to some relevant literature. AP acknowledges partial support by the EPSRC Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (EP/S024050/1).

References

Ali Al-Haj. 2007. Combined DWT-DCT digital image watermarking. *Journal of Computer Science*, 3(9):740–746.

Matthias Althoff, Olaf Stursberg, and Martin Buss. 2010. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear analysis: Hybrid systems*, 4(2):233–249.

Yoshinori Aono and Phong Q Nguyen. 2017. Random sampling revisited: Lattice enumeration with discrete pruning. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 65–102.

Mauro Barni, Franco Bartolini, and Alessandro Piva. 2001. Improved wavelet-based watermarking through pixel-wise masking. *IEEE transactions on image processing*, 10(5):783–791.

Patrick Bas, J-M Chassery, and Benoit Macq. 2002. Geometrically invariant watermarking using feature points. *IEEE transactions on image Processing*, 11(9):1014–1028.

Adrian G Bors and Ioannis Pitas. 1996. Image watermarking using DCT domain constraints. In *ICIP*.

Tu Bui, Shruti Agarwal, and John Collomosse. 2023a. Trustmark: Universal watermarking for arbitrary resolution images. *arXiv preprint arXiv:2311.18297*.

Tu Bui, Shruti Agarwal, Ning Yu, and John Collomosse. 2023b. RoSteALS: Robust steganography using autoencoder latent space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*.

California State Leg. 2024. Amendment to California Assembly bill 3211. California State Legislature. Amended in Assembly.

Brian Chen and Gregory W Wornell. 2002. Quantization index modulation: A class of provably good methods for digital watermarking and information embedding. *IEEE Transactions on Information theory*, 47(4):1423–1443.

Guangyu Chen, Yu Wu, Shujie Liu, Tao Liu, Xiaoyong Du, and Furu Wei. 2023. WavMark: Watermarking for audio generation. *arXiv preprint arXiv:2308.12770*.

Hai Ci, Pei Yang, Yiren Song, and Mike Zheng Shou. 2024. RingID: Rethinking tree-ring watermarking for enhanced multi-key identification. *arXiv preprint arXiv:2404.14055*.

Coalition for Content Provenance and Authenticity (C2PA). 2025. Technical specification 2.2.

Aaron S Cohen and Amos Lapidoth. 2002. The Gaussian watermarking game. *IEEE Transactions on Information Theory*, 48(6):1639–1667.

John Collomosse and Andy Parsons. 2024. To authenticity, and beyond! Building safe and fair generative AI upon the three pillars of provenance. *IEEE Computer Graphics and Applications*.

Denis Constales. 1997. Solution to "The volume of the intersection of a cube and a ball in N-space" posed by Liqun Xu. *SIAM Review (Problems and Solutions)*, 39.4:779–786.

Max Costa. 1983. Writing on dirty paper. *IEEE Transactions on Information Theory*, 29(3):439–441.

I.J. Cox, J. Kilian, F.T. Leighton, and T. Shamoon. 1997. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing*, 6(12):1673–1687.

Sumanth Dathathri, Abigail See, Sumedh Ghaisas, Po-Sen Huang, Rob McAdam, Johannes Welbl, Vandana Bachani, Alex Kaskasoli, Robert Stanforth, Tatiana Matejovicova, et al. 2024. Scalable watermarking for identifying large language model outputs. *Nature*, 634(8035):818–823.

Patrick Esser, Robin Rombach, and Bjorn Ommer. 2021. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*.

European Parliament and Council. 2024. Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828 (Artificial Intelligence Act).

Liu Ping Feng, Liang Bin Zheng, and Peng Cao. 2010. A DWT-DCT based blind watermarking algorithm for copyright protection. In *2010 3rd International Conference on Computer Science and Information Technology*, volume 7, pages 455–458.

Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. 2023. The stable signature: Rooting watermarks in latent diffusion models. In *International Conference on Computer Vision*.

Pierre Fernandez, Hady Elsahar, I Zeki Yalniz, and Alexandre Mourachko. 2024. Video Seal: Open and efficient video watermarking. *arXiv preprint arXiv:2412.09492*.

Pierre Fernandez, Alexandre Sablayrolles, Teddy Furon, Hervé Jégou, and Matthijs Douze. 2022. Watermarking images in self-supervised latent spaces. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Carl Friedrich Gauss. 1837. De nexu inter multitudinem classium, in quas formae binariae secundi gradus distribuuntur, earumque determinantem. In *Werke: Band 2*, pages 269–291.

Antoine Girard. 2005. Reachability of uncertain linear systems using zonotopes. In *Proceedings of the 8th International Conference on Hybrid Systems: Computation and Control*.

Moritz Hardt and Tengyu Ma. 2017. Identity matters in deep learning. In *International Conference on Learning Representations*.

G. H. Hardy. 1915. On the expression of a number as the sum of two squares. *Quarterly Journal of Mathematics*, 46:263–283.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Seongmin Hong, Kyeonghyun Lee, Suh Yoon Jeon, Hyewon Bae, and Se Young Chun. 2024. On exact inversion of DPM-solvers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Zhaoyang Jia, Han Fang, and Weiming Zhang. 2021. MBRS: Enhancing robustness of DNN-based watermarking by mini-batch of real and simulated JPEG compression. In *Proceedings of the 29th ACM international conference on multimedia*.

Changhoon Kim, Kyle Min, Maitreya Patel, Sheng Cheng, and Yezhou Yang. 2023. Wouaf: Weight modulation for user attribution and fingerprinting in text-to-image diffusion models. *arXiv preprint arXiv:2306.04744*.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *International Conference on Machine Learning*.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. 2023. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026.

Varsha Kishore, Xiangyu Chen, Yan Wang, Boyi Li, and Kilian Q Weinberger. 2022. Fixed neural network steganography: Train the images, not the network. In *International Conference on Learning Representations*.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. 2014. Microsoft COCO: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. 2022. A Convnet for the 2020s. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11976–11986.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

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

Alina Maor and Neri Merhav. 2005. On joint information embedding and lossy compression. *IEEE Transactions on Information Theory*, 51(8):2998–3008.

Neri Merhav. 2005. An information-theoretic view of watermark embedding-detection and geometric attacks. *Proceedings of WaCha, 2005, First Wavila Challenge*.

WC Mitchell. 1966. The number of lattice points in a k-dimensional hypersphere. *Mathematics of Computation*, 20(94):300–310.

Pierre Moulin and Ralf Koetter. 2005. Data-hiding codes. *Proceedings of the IEEE*, 93(12):2083–2126.

Pierre Moulin and Joseph A O'Sullivan. 2003. Information-theoretic analysis of information hiding. *IEEE Transactions on information theory*, 49(3):563–593.

Matthew J. Muckley, Alaaeldin El-Nouby, Karen Ullrich, Herve Jegou, and Jakob Verbeek. 2023. Improving statistical fidelity for neural image compression with implicit local likelihood models. In *Proceedings of the 40th International Conference on Machine Learning*, pages 25426–25443.

Seung-Min Mun, Seung-Hun Nam, Han-Ul Jang, Dongkyu Kim, and Heung-Kyu Lee. 2017. A robust blind watermarking using convolutional neural network. *arXiv preprint arXiv:1704.03248*.

K. A. Navas, Mathews Cheriyan Ajay, M. Lekshmi, Tampy S. Archana, and M. Sasikumar. 2008. DWT-DCT-SVD based watermarking. In *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pages 271–274.

Zhicheng Ni, Yun-Qing Shi, N. Ansari, and Wei Su. 2006. Reversible data hiding. *IEEE Transactions on Circuits and Systems for Video Technology*, 16(3):354–362.

Minzhou Pan, Yi Zeng, Xue Lin, Ning Yu, Cho-Jui Hsieh, Peter Henderson, and Ruoxi Jia. 2024. JIGMARK: A black-box approach for enhancing image watermarks against diffusion model edits. *arXiv preprint arXiv:2406.03720*.

Aleksandar Petrov, Shruti Agarwal, Philip HS Torr, Adel Bibi, and John Collomosse. 2025. On the coexistence and ensembling of watermarks. *arXiv preprint arXiv:2501.17356*.

Alessandro Piva, Mauro Barni, Franco Bartolini, and Vito Cappellini. 1997. DCT-based watermark recovering without resorting to the uncorrupted original image. In *Proceedings of international conference on image processing*, volume 1, pages 520–523. IEEE.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-Net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241.

Robin San Roman, Pierre Fernandez, Hady Elsahar, Alexandre Défossez, Teddy Furon, and Tuan Tran. 2024. Proactive detection of voice cloning with localized watermarking. In *Proceedings of the 41st International Conference on Machine Learning*.

Tom Sander, Pierre Fernandez, Alain Oliviero Durmus, Teddy Furon, and Matthijs Douze. 2025. Watermark anything models: Localized deep-learning image watermarking for small areas, inpainting, and splicing. *International Conference on Learning Representations*.

Rolf Schneider. 2013. *Convex bodies: the Brunn–Minkowski theory*, volume 151. Cambridge University Press.

Anelia Somekh-Baruch and Neri Merhav. 2004. On the capacity game of public watermarking systems. *IEEE Transactions on Information Theory*, 50(3):511–524.

Matthew Tancik, Ben Mildenhall, and Ren Ng. 2020. StegaStamp: Invisible hyperlinks in physical photographs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

The White House. 2023. Executive order on the safe, secure, and trustworthy development and use of artificial intelligence.

Aaron Van Den Oord, Oriol Vinyals, et al. 2017. Neural discrete representation learning. *Advances in Neural Information Processing Systems*, 30.

Ron G Van Schyndel, Andrew Z Tirkel, and Charles F Osborne. 1994. A digital watermark. In *Proceedings of 1st international conference on image processing*, volume 2, pages 86–90. IEEE.

Igor G Vladimirov. 2015. Quantized linear systems on integer lattices: A frequency-based approach. *arXiv preprint arXiv:1501.04237*.

Vedran Vukotić, Vivien Chappelier, and Teddy Furon. 2018. Are deep neural networks good for blind image watermarking? In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*.

Arnold Walfisz. 1957. *Gitterpunkte in mehrdimensionalen Kuglen*, volume 33 of *Monografie Matematyczne*.

Gregory K. Wallace. 1991. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44.

Guanjie Wang, Zehua Ma, Chang Liu, Xi Yang, Han Fang, Weiming Zhang, and Nenghai Yu. 2024. MuST: Robust image watermarking for multi-source tracing. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612.

Yuxin Wen, John Kirchenbauer, Jonas Geiping, and Tom Goldstein. 2023. Tree-ring watermarks: Fingerprints for diffusion images that are invisible and robust. *arXiv preprint arXiv:2305.20030*.

Wikipedia. 2025. JPEG — Wikipedia, the free encyclopedia. [Online; accessed 30-July-2025].

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

Rui Xu, Mengya Hu, Deren Lei, Yaxi Li, David Lowe, Alex Gorevski, Mingyu Wang, Emily Ching, and Alex Deng. 2025. InvisMark: Invisible and robust watermarking for ai-generated image provenance. In *2025 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*.

Aditi Zear, Amit Kumar Singh, and Pardeep Kumar. 2018. A proposed secure multiple watermarking technique based on DWT, DCT and SVD for application in medicine. *Multimedia tools and applications*, 77(4):4863–4882.

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

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. 2018. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Xuanyu Zhang, Runyi Li, Jiwen Yu, Youmin Xu, Weiqi Li, and Jian Zhang. 2024. EditGuard: Versatile image watermarking for tamper localization and copyright protection. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*.

Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. 2018. HiDDeN: Hiding data with deep networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Appendix

A Equivalence of PSNR and the ℓ_2 ball constraint

In the main text we stated that imposing a minimum PSNR τ is equivalent to requiring the watermarked image to remain within an ℓ_2 ball around the cover image. Here we give the short derivation and clarify how the radius is defined in terms of both ℓ_2 and MSE distances.

PSNR is defined from the mean squared error (MSE) between two images:

$$\mathrm{PSNR}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = 10 \log_{10} \frac{(\texttt{max possible pixel value})^2}{\mathrm{MSE}(\boldsymbol{x}, \tilde{\boldsymbol{x}})}.$$

The MSE measures the *average* squared pixel difference, while the squared ℓ_2 norm measures the *total* squared difference across all cwh pixels. The two are linked by

$$\mathrm{MSE}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) = \frac{1}{cwh} \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2^2.$$

Thus, a PSNR threshold on MSE can be rephrased as a maximum allowable ℓ_2 distance between $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$. The connection between the two is:

$$\begin{aligned}
\mathrm{PSNR}(\boldsymbol{x}, \tilde{\boldsymbol{x}}) \geq \tau &\iff 10 \log_{10}\Big(\frac{\rho^2}{\mathrm{MSE}}\Big) \geq \tau \\
&\iff \mathrm{MSE} \leq \frac{\rho^2}{10^{\tau/10}} \\
&\iff \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2^2 \leq cwh \cdot \frac{\rho^2}{10^{\tau/10}} \\
&\iff \|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|_2 \leq \rho\sqrt{cwh}\, 10^{-\tau/20}.
\end{aligned}$$

We will define

$$\epsilon(\tau) = \rho\sqrt{cwh}\, 10^{-\tau/20}.$$

In other words, $\epsilon(\tau)$ specifies the largest ℓ_2 distance (equivalently, the largest total squared pixel error) allowed by the PSNR constraint.

B Volume-based estimation of the number of grid points in hyperspheres

Calculating watermarking capacity under a PSNR constraint reduces to counting how many valid images (grid points in the pixel space) lie inside an ℓ_2 ball of radius ϵ around the cover image. This is equivalent to asking how many integer grid points fall inside such a ball: a problem without a general closed-form solution. In two dimensions this becomes the well-known *Gauss circle problem* (Gauss, 1837; Hardy, 1915). In higher dimensions, and particularly for large radii, the count can be well-approximated by the volume of the n-dimensional ball:

$$\mathrm{Vol}(B_n[\cdot, r]) = \frac{\pi^{n/2}}{\Gamma\left(\frac{n}{2}+1\right)} r^n. \tag{3}$$

We can approximate the number of integer points inside the ball with its volume. Simply, each grid point corresponds to a unit cube in space, so counting grid points is almost the same as measuring volume. The only difference comes from the boundary of the ball: some cubes are cut by the surface, so they are only partially inside. The absolute error grows as $\mathcal{O}(r^{(n-1)/2})$ (Walfisz, 1957; Mitchell, 1966). Since the volume itself grows as $\mathcal{O}(r^n)$, the relative error decreases as $\mathcal{O}(r^{-(n+1)/2})$. Thus for large radii the volume approximation is quite accurate. For small radii though, this error is significant, as shown in Figure 3, hence we will use exact counting (Section C) for these cases.

C Exact counting grid points in hyperspheres for small radii

In Section 3.2 we need to calculate how many integer points are in the interior of small ℓ_2 balls for which approximating the number of integer points with the volume of the ball as in Section B is inaccurate. Naively, we can iterate through the points in the smallest hypercube with integer coordinates that contains our ball and check which points would fall inside the ball. See Algorithm 1 for one implementation of this.

Algorithm 1: Brute Force Count of Lattice Points in a Hypersphere

Function `BruteForceCount`(radius : $\mathbb{R}_{\geq 0}$, dim : $\mathbb{Z}^+$)**:**
    rsq $\leftarrow$ radius^2;
    single_axis_points $\leftarrow \{-\lfloor \text{radius} \rfloor, \ldots, \lfloor \text{radius} \rfloor\}$;
    counter $\leftarrow 0$;
    foreach $\boldsymbol{p} \in$ product(single_axis_points, repeat = dim) **do**
        if $\sum_{p=1}^{dim} \boldsymbol{p}_i^2 \leq rsq$ **then**
            counter $\leftarrow$ counter $+ 1$;
    return counter;

Obviously, Algorithm 1 does not scale beyond very low dimensions because it has complexity that is exponential in the dimension. Luckily, one can leverage symmetries to reduce the number points to be checked. We take the method by (Mitchell, 1966) with pseudo-code presented in Algorithm 2. Note that this can also be further sped up by caching the calls to S.

Algorithm 2: Count Lattice Points in a Hypersphere (Mitchell's Method)

Function `PointsInHypersphereMitchell`(dim : $\mathbb{Z}^+$, radius : $\mathbb{R}_{\geq 0}$)**:**
    return `S`(dim, radius^2, ∞);

Function `S`($m : \mathbb{Z}_{\geq 0}$, $Z : \mathbb{R}$, $J : \mathbb{Z} \cup \{\infty\}$)**:**
    if $m = 0$ **then**
        if $Z \geq 0$ **then**
            return 1;
        else
            return 0;
    $N \leftarrow \left\lfloor \sqrt{Z/m} \right\rfloor$;
    $r \leftarrow (2N+1)^m$;
    for $i \leftarrow 1$ **to** $m - 1$ **do**
        $\text{MIN}_i \leftarrow \left\lfloor \min\left(\sqrt{Z/i},\ J-1\right) \right\rfloor$;
        for $J_m \leftarrow N+1$ **to** MIN_i **do**
            $r \leftarrow r + \binom{m}{i} \cdot 2^i \cdot$ `S`$(m-i,\ Z - iJ_m^2,\ J_m)$;
    return r;

Unfortunately, Algorithm 2 is not applicable if we want to compute the number of lattice points in the intersection between the hypersphere and a hypercube, as often needed when computing the number of valid images available for watermarking in the present paper. In such cases, we can use the following simple algorithm which is faster than the naive Algorithm 1 but slower than Algorithm 2. Note that, again, this can be significantly sped up by caching the calls to `IterativeCountWithBounds`.

Algorithm 3: Count Lattice Points in a Hypersphere with Bounds

Function `PointsInHypersphereWithBounds`(dim : $\mathbb{Z}^+$, radius : $\mathbb{R}_{\geq 0}$, bounds : $(\mathbb{R} \times \mathbb{R})^{\text{dim}}$)**:**
    int_bounds $\leftarrow$ $(\lceil b_0 \rceil, \lfloor b_1 \rfloor)$ for each (b_0, b_1) in bounds;
    return `IterativeCountWithBounds`(radius^2, dim, int_bounds);

Function `IterativeCountWithBounds`(rsq : $\mathbb{R}_{\geq 0}$, dim : $\mathbb{Z}_{\geq 0}$, bounds : $(\mathbb{Z} \times \mathbb{Z})^{dim}$)**:**
    if dim $= 0$ **then**
        return 1 if rsq ≥ 0, else 0;
    count $\leftarrow 0$;
    $M \leftarrow \lfloor \sqrt{\text{rsq}} \rfloor$;
    lb $\leftarrow \max(-M, \text{bounds}[0][0])$;
    ub $\leftarrow \min(M, \text{bounds}[0][1])$;
    for $i \leftarrow$ lb **to** ub **do**
        if $i^2 \leq$ rsq **then**
            count $\leftarrow$ count+ `IterativeCountWithBounds`(rsq $- i^2$, dim $- 1$, bounds[1 :]);
    return count;

D Bounds for gray image in the non-trivial intersection case

We expand on the exact bound on the capacity under only PSNR constraint (no robustness) in the non-trivial intersection case, i.e., for medium PSNR values, as discussed in Section 3.2.3. We use the volume-based approximation approach, reducing the problem to finding the volume of an intersection of a hypercube and a hypersphere. Unfortunately, there is no closed-form solution. However, with some care for the numerical precision, we can compute these intersections with numerical integration. First, observe that we can express the volume of the intersection of an arbitrary ball and hypercube as the volume of the intersection of appropriately transformed hypercube with the unit ball:

$$\operatorname{Vol}\left[\prod_{j=1}^{n}[\alpha_j,\beta_j]\ \cap\ B_n\left[\boldsymbol{x},r\right]\right] = r^n \operatorname{Vol}\left[\prod_{j=1}^{n}\left[\frac{\alpha_j-\boldsymbol{x}_j}{r},\frac{\beta_j-\boldsymbol{x}_j}{r}\right]\ \cap\ B_n\left[\mathbf{0},1\right]\right]. \tag{4}$$

The right-hand side of Equation (4) can be represented as an infinite sum, which, in practice, can be approximated via truncation. We will use the following result originally due Constales (1997) and generalized by Aono and Nguyen (2017, Theorem 4):

Theorem 1 (Volume of the intersection of a cube and a ball)**.** *Let $S(x)=\int_0^x \sin(t^2)\,dt$ and $C(x)=\int_0^x \cos(t^2)\,dt$ be the Fresnel integrals.*[2] *Let $\alpha_j \le \beta_j$ for $1\le j\le n$ and $\ell=\sum_{j=1}^n \max(\alpha_j^2,\beta_j^2)$. Then:*

$$\operatorname{Vol}\left[\prod_{j=1}^{n}[\alpha_j,\beta_j]\cap B_n\left[\mathbf{0},1\right]\right] = \begin{cases} K & \text{if } \ell\le 1 \\ \left(\frac{1}{2}-\frac{\sum_{j=1}^n \alpha_j^2+\beta_j^2+\alpha_j\beta_j}{3\ell}+\frac{1}{\ell}+\frac{1}{\pi}\operatorname{Im}\sum_{k=1}^{\infty}\frac{\Phi(-2\pi k/\ell)}{k}e^{2i\pi k/\ell}\right)K & \text{if } \ell>1 \end{cases}$$

where $K=\prod_{j=1}^n|\beta_j-\alpha_j|$ and Φ is defined as

$$\Phi(\omega)=\prod_{j=1}^{n}\frac{\left(C(\beta_j\sqrt{|\omega|})-C(\alpha_j\sqrt{|\omega|})\right)+i\ \operatorname{sign}(\omega)\left(S(\beta_j\sqrt{|\omega|})-S(\alpha_j\sqrt{|\omega|})\right)}{(\beta_j-\alpha_j)\sqrt{|\omega|}}.$$

A number of speed-ups and numerical performance optimization tricks can be used when computing the terms in Theorem 1. For example, when all $\alpha_j=\alpha_1$ and $\beta_j=\beta_1$ for all $1\le j\le n$ and when $\alpha_j=-\beta_j$, which is often the case in the setups we consider. We provide such optimized implementation in Algorithm 4.

Theorem 1 directly gives us a bound on the capacity for the non-trivial intersection case:

Bound 5: Gray image, PSNR constraint (medium PSNR, numerical integration). The capacity of a gray image $\boldsymbol{x}_g$ under a minimum PSNR constraint τ in the ambient space $\mathcal{A}$ is upper-bounded by

$$\begin{aligned}\texttt{capacity}[\text{in bits}] &\approx \log_2\left[\epsilon^{cwh}\operatorname{Vol}\left[\prod_{j=1}^{cwh}\left[-\frac{\rho/2}{\epsilon(\tau)},\frac{\rho/2}{\epsilon(\tau)}\right]\cap B_{cwh}\left[\mathbf{0},1\right]\right]\right] \\ &= cwh\log_2\epsilon(\tau)+\log_2\operatorname{Vol}\left[\prod_{j=1}^{cwh}\left[-\frac{\rho/2}{\epsilon(\tau)},\frac{\rho/2}{\epsilon(\tau)}\right]\cap B_{cwh}\left[\mathbf{0},1\right]\right],\end{aligned}$$

with the volume computed by truncating the sum in Theorem 1.
Bound validity: If $B_{cwh}\left[\boldsymbol{x}_g,\epsilon(\tau)\right]\not\subset C_{\mathcal{I}}$ and $C_{\mathcal{I}}\not\subset B_{cwh}\left[\boldsymbol{x}_g,\epsilon(\tau)\right]$, which happens when $\rho/2\le\epsilon(\tau)\le\rho/2\sqrt{cwh}$ or, equivalently from Equation (1), when $20\log_{10}2\le\tau\le 20\log_{10}(2\sqrt{cwh})$. Assuming cwh not too large (otherwise numerical evaluation of the bound becomes intractable).

[2] Note that some software libraries provide alternative (normalized) definitions for the Fresnel integrals: $\tilde{S}(x)=\int_0^x\sin(\pi x^2/2)$ and $\tilde{C}(x)=\int_0^x\cos(\pi x^2/2)$. The two are related as such: $S(x)=\sqrt{\pi/2}\ \tilde{S}(\sqrt{2/\pi}x)$ and $C(x)=\sqrt{\pi/2}\ \tilde{C}(\sqrt{2/\pi}x)$.

The main limitation of Bound 5 is that it becomes computationally intractable to compute it for large cwh. In our implementation, we could evaluate it up to cwh of several hundred. However, as can be seen in Figure 4 left, Bound 5 can be well-approximated by simply considering the minimum of Bound 2 and Bound 3. Therefore, for large resolutions we will use Bound 6.

Algorithm 4: Computes the volume of the intersection between a hypersphere and a box

Function ApplyWithGrouping(f : function, args, mode : (product, sum)):
    // Applies a function f to arguments, grouping them for stability.
    pairs $\leftarrow$ list(zip($*$args));
    $\mathcal{C} \leftarrow$ Counter(pairs) ; // Count unique argument tuples
    if mode = product **then**
        result $\leftarrow 1$;
    else if mode = sum **then**
        result $\leftarrow 0$;
    foreach $p \in \mathcal{C}$.keys() **do**
        $c \leftarrow \mathcal{C}[p]$;
        $r \leftarrow f(p)$;
        if mode = product **then**
            result $\leftarrow$ result $\cdot\, r^c$;
        else if mode = sum **then**
            result $\leftarrow$ result $+ c \cdot r$;
    return result;

Function $\Phi_{\text{inner}}(\alpha : \mathbb{R}, \beta : \mathbb{R}, \omega : \mathbb{R})$:
    $\omega' \leftarrow \sqrt{|\omega|}$;
    if $\alpha = -\beta$ **then**
        $T_1 \leftarrow 2 \cdot \text{fresnelc}(\beta\omega')$;
        $T_2 \leftarrow i \cdot \text{sgn}(\omega) \cdot 2 \cdot \text{fresnels}(\beta\omega')$;
    else
        $T_1 \leftarrow \text{fresnelc}(\beta\omega') - \text{fresnelc}(\alpha\omega')$;
        $T_2 \leftarrow i \cdot \text{sgn}(\omega) \cdot (\text{fresnels}(\beta\sqrt{\omega_{abs}}) - \text{fresnels}(\alpha\omega'))$;
    return $(T_1 + T_2)/((\beta - \alpha)\omega')$;

Function $\Phi(\omega : \mathbb{R},\ \boldsymbol{\alpha},\ \boldsymbol{\beta})$:
    $f_{\text{inner}} \leftarrow ((\alpha, \beta) \mapsto \Phi_{\text{inner}}(\alpha, \beta, \omega)\ )$;
    return ApplyWithGrouping(f_{inner}, $[\boldsymbol{\alpha}, \boldsymbol{\beta}]$, product);

Function BallCubeIntersection($R : \mathbb{R}_{>0}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, $N_{sum} : \mathbb{Z}^+$):
    // Trim scaled bounds to the unit ball range $[-1, 1]$
    $\boldsymbol{\alpha}' \leftarrow \text{elementwise\_max}(-1, \boldsymbol{\alpha}/R)$;
    $\boldsymbol{\beta}' \leftarrow \text{elementwise\_min}(1, \boldsymbol{\beta}/R)$;
    $\ell \leftarrow$ ApplyWithGrouping($(a, b) \mapsto (\max(-a, b))^2$, $[\boldsymbol{\alpha}', \boldsymbol{\beta}']$, sum);
    $E_X \leftarrow {}^1\!/_3$ ApplyWithGrouping($(a, b) \mapsto a^2 + b^2 + ab$, $[\boldsymbol{\alpha}', \boldsymbol{\beta}']$, sum);
    $V_{\text{scale}} \leftarrow$ ApplyWithGrouping($(a, b) \mapsto \log_2(b - a)$, $[\boldsymbol{\alpha}', \boldsymbol{\beta}']$, sum);
    $T_1 \leftarrow {}^1\!/_2$; $T_2 \leftarrow {}^{E_X}\!/_\ell$; $T_3 \leftarrow {}^1\!/_\ell$;
    $S_4 \leftarrow 0$;
    for $k \leftarrow 1$ **to** N_{sum} **do**
        $\omega_k \leftarrow -2\pi k/\ell$;
        $S_4 \leftarrow S_4 + \frac{1}{k} \cdot\ \Phi(\omega_k, \boldsymbol{\alpha}', \boldsymbol{\beta}')\ \cdot \text{expjpi}(2k/\ell)$;
    $T_4 \leftarrow \text{Im}(S_4)/\pi$;
    $V_{\text{norm}} \leftarrow T_1 - T_2 + T_3 + T_4$;
    return $\log_2(V_{\text{norm}}) + V_{\text{scale}} + \text{len}(\boldsymbol{\alpha}') \cdot \log_2(R)$;

E Bounds for arbitrary cover images

In Section 3.3 we extended the analysis from centred covers to arbitrary images. Here we go into detail about how the corresponding bounds were derived.

When pixels are saturated, the cover may lie on the boundary of the cube of cube $C_{\mathcal{I}}$. The most adverse case is when all pixels are saturated,i.e., the cover at a corner. This minimizes the overlap between the PSNR ball and the grid and hence provides a lower bound on capacity for any image.

By symmetry, the ball is evenly divided among the 2^{cwh} orthants of $\mathbb{R}^{cwh}$, so only a fraction $1/2^{cwh}$ of its volume lies within the grid. In two dimensions this corresponds to a quarter of a circle inside a square corner, and in three dimensions to one eighth of a sphere inside a cube corner. Although this seems drastic, the effect on capacity is limited: at most cwh bits, i.e. one bit per pixel.

We now provide the detailed derivations of Bounds 7 to 9, which are the analogues of the centered bounds from Section 3.2, adapted to the corner case.

Bound 7: Arbitrary image, PSNR constraint (high PSNR, volume approximation, analogous to Bound 3). The capacity of an image $\boldsymbol{x}$ under a minimum PSNR constraint τ in the ambient space $\mathcal{A}$ is upper-bounded by

$$\begin{aligned}\texttt{capacity}[\text{in bits}] &\approx \log_2\left[\frac{\operatorname{Vol} B_{cwh}\left[\cdot, \epsilon(\tau)\right]}{2^{cwh}}\right] \\ &= \frac{cwh}{2}\log_2 \pi + cwh \log_2 \epsilon(\tau) - \frac{\ln \Gamma\left(\frac{cwh}{2}+1\right)}{\ln 2} - cwh.\end{aligned}$$

Bound validity: If $\epsilon(\tau) \leq \rho/2$ or, equivalently from Equation (1), if $\tau \geq 20\log_{10}(2\sqrt{cwh})$. Assuming $\epsilon(\tau)$ not too small for the volume approximation to be valid (see Bound 8 for small $\epsilon(\tau)$).

Bound 8: Arbitrary image, PSNR constraint (high PSNR, exact count, analogous to Bound 4). The capacity of an image $\boldsymbol{x}$ under a minimum PSNR constraint τ in the ambient space $\mathcal{A}$ is upper-bounded by

$$\texttt{capacity}[\text{in bits}] \approx \log_2 \texttt{PointsInHypersphereWithBounds}(cwh, \epsilon(\tau), (0, 2^k)),$$

with `PointsInHypersphereWithBounds` defined in Algorithm 3.
Bound validity: If $\epsilon(\tau) \leq \rho/2$ or, equivalently from Equation (1), if $\tau \geq 20\log_{10}(2\sqrt{cwh})$. Assuming ϵ small (otherwise the numerical evaluation becomes intractable, see Bound 7 for large ϵ).

Note that Bound 7 slightly *under-approximates* capacity, since only half of the lattice points lying on the faces of the hypercube are captured by the volume approximation. This explains the discontinuity between Bound 7 and Bound 8 visible in Figure 4 right.

For the other two cases: non-trivial intersection and the cube being fully in the ball, we can directly apply Theorem 1 with appropriate change of bounds. We simply need to adjust the bounds in Bound 5 from $[-\rho/2, \rho/2]$ to $[0, \rho]$:

Bound 9: Arbitrary image, PSNR constraint (medium PSNR, numerical integration, analogous to Bound 5). The capacity of an image $\boldsymbol{x}$ under a minimum PSNR constraint τ in the ambient space $\mathcal{A}$ is upper-bounded by

$$\begin{aligned}\texttt{capacity}[\text{in bits}] &\approx \log_2 \left[\epsilon(\tau)^{cwh} \operatorname{Vol}\Big(\prod_{j=1}^{cwh} \Big[0, \tfrac{2^k-1}{\epsilon(\tau)}\Big] \cap B_{cwh}\left[\mathbf{0}, 1\right] \Big) \right] \\ &= cwh \log_2 \epsilon + \log_2 \operatorname{Vol}\Big(\prod_{j=1}^{cwh} \Big[0, \tfrac{2^k-1}{\epsilon(\tau)}\Big] \cap B_{cwh}\left[\mathbf{0}, 1\right] \Big),\end{aligned}$$

with the volume computed by truncating the sum in Theorem 1.
Bound validity: If $\epsilon(\tau) \geq \rho/2$ or, equivalently from Equation (1), when $\tau \leq 20 \log_{10}(2\sqrt{cwh})$. Assuming cwh not too large (otherwise the numerical evaluation becomes intractable).

Unlike the centred case, here the symmetry condition $\alpha_j = -\beta_j$ no longer holds, so the simplifications of Theorem 1 cannot be applied. Bound 9 is therefore numerically stable only at relatively low resolutions. Nevertheless, the per-pixel capacity (bpp) is resolution-invariant, so we compute these bounds at 16×16px.

Bounds 7 to 9 extend the centred-case analysis to arbitrary cover images. Across all three regimes, the penalty of being at the corner of the grid is at most one bit per pixel. Thus, the observed gap between theoretical capacity and practical watermarking performance cannot be explained by image position within the grid.

F Bounds for capacity under robustness constraints

In practice, watermarking requires balancing perceptual quality with robustness. That is, minor modifications to the watermarked image should not prevent the watermark from being extractable. Such modifications might arise in the normal processing of the image (a social media website might compress uploaded images) or might be malicious (to strip provenance information). Typically, one considers a set of transformations against which a watermarking method should be robust.

Robustness comes at a cost: it reduces capacity. To quantify this trade-off we study how robustness constraints reduce the number of images which we can use for watermarking, quantifying the corresponding reduction in capacity. We will focus on robustness to linear transformations, which, though seemingly restrictive, covers or approximates most practical transformations. Linear transformations are also compositional, simplifying the creation of complex transformations from basic ones. Standard augmentation can be directly represented as linear transformations. Section G shows how to represent colour space changes, rotation, flipping, cropping and rescaling, as well as a number of intermediate operators that can be used to construct the linear operators corresponding to other transformations.

F.1 Robustness to linear transformations

A linear transformation applied to the ball $B_{cwh}[\mathbf{0}, \epsilon]$ of possible watermarked images can turn it into an ellipsoid (if the transformation has more than one unique singular value) can scale it (if they are all the same but not 1) and can also project it into a lower-dimensional space if the transformation is not invertible. Let's take a linear transformation $M \in \mathbb{R}^{cwh \times cwh}$ that maps an image $\boldsymbol{x}$ to a transformed image $M\boldsymbol{x}$. Note that we further need to apply a quantization operation to map the pixel values of $M\boldsymbol{x}$ to the valid images $\mathcal{I}$. Hence, we have the final transformed image $\boldsymbol{x}' = \mathsf{Q}[M\boldsymbol{x}]$ with Q being a quantization operator, typically an element-wise rounding or floor operation. The main complication in this setup is that Q is non-linear. To establish the capacity under a linear transformation, we need to find the subset of the possible watermarked images under only the PSNR constraint that map to unique valid images after applying M and Q to them.

F.2 Heuristic bounds

A simple approach is to consider the volumetric approach for calculating capacity that we used for Bounds 3, 5, 7 and 9. A linear operator M would change the volume of $B_{cwh}[\mathbf{0}, \epsilon]$ by a factor of $\det M = \lambda_M^1 \times \cdots \times \lambda_M^{cwh}$ (the product of M's eigenvalues) if M is not singular. Note that if $\det M > 0$, then we can also express it as $\det M = \sigma_M^1 \times \cdots \times \sigma_M^{cwh}$, the product of singular values of M. If M is singular, then $MB_{cwh}[\mathbf{0}, \epsilon]$ has 0 cwh-dimensional volume as some eigenvalues (singular values) would be 0. However, it will have a non-zero $\operatorname{pdet} M = \prod \left\{\lambda_M^i \mid \lambda_M^i \neq 0,\ i = 1, \ldots, cwh\right\}$ ($\operatorname{rank} M$)-dimensional volume, with $\operatorname{pdet} M$ being the *pseudo-determinant* of M.

The change in volume governs the reduction in capacity. However, it is not as simple as just calculating this volume and taking that to be the capacity because of the quantizer Q. Take for example

$$M = \begin{bmatrix} 2 & 0 \\ 0 & 1/2 \end{bmatrix}. \tag{5}$$

The determinant is $\det M = 1$ indicating no volume change and, thus, if we ignore the quantization, no capacity change. However, one of the dimensions is squeezed by a factor of 2, hence we should lose about half of the capacity along this axis. The other axis, that is stretched by a factor of 2, does not create capacity because pairs of these points would have the same preimage. Therefore, assuming we are far from the boundaries of the cube, we should see half the original capacity, if we want to have robustness to this augmentation. Following this observation, we provide an heuristic for the reduction ξ of capacity due to the linear operator and quantization:

$$\xi_M = \prod_{\sigma_M^i \in \Sigma_M : \sigma_M^i > 0} \min(\sigma_M^i, 1), \tag{6}$$

where Σ_M are the singular values of M. ξ_M captures the combined effect of all singular values of M. Each $\sigma_i < 1$ represents a compression that reduces capacity proportionally due to the quantization, while $\sigma_i > 1$ is

capped at 1 since stretching cannot create capacity. The product accounts for all $\operatorname{rank} M$ dimensions, so ξ_M reflects the total fraction of capacity that remains after accounting for all reductions.

If, for a moment, we ignore that we need to clip the pixel values in their valid range, we get the following capacity under an *invertible* linear transformation M for ϵ large enough so that the volume-based approximation is applicable:

$$\begin{aligned}\texttt{capacity under } M[\text{in bits}] &= \texttt{capacity}[\text{in bits}] + \log_2 \xi_M \\ &\approx \log_2 \operatorname{Vol} B_{cwh}\left[\cdot, \epsilon(\tau)\right] + \log_2 \xi_M.\end{aligned}$$

If M is singular, however, then we need to compute the capacity under the lower-dimensional projection of $B_{cwh}\left[\cdot, \epsilon(\tau)\right]$ in order to account for the collapsed dimensions:

$$\texttt{capacity under } M[\text{in bits}] \approx \log_2 \operatorname{Vol} B_{\operatorname{rank} M}\left[\cdot, \epsilon(\tau)\right] + \log_2 \xi_M.$$

Note that the radius $\epsilon(\tau)$ is still computed in the ambient cwh-dimensional space.

A further complication caused by the $\sigma_i > 1$ singular values is the possibility of the sphere going out of the bounds of the cube after the transform. Looking again at the example in Equation (5), the stretching along the first dimension might result in some of the watermarked images being clipped and hence mapped to the same image after applying the transformation. We can factor this in by adjusting the bounds corresponding to the cube boundaries in Bounds 3 to 5 accordingly. This results in the heuristic bounds in this section.

First, let's look at the setting where the ball is fully inside the cube *before and after* the transformation, and its radius $\epsilon(\tau)$ is large enough for us to approximate the number of images in it with its volume. Taking into account M being possibly singular, we have:

Bound 10: Gray image, Linear transformation (Heuristic), PSNR constraint (high PSNR, volume approximation). The capacity of a gray image $\boldsymbol{x}_g$ under a low minimum PSNR constraint τ and a linear transformation $M \in \mathbb{R}^{cwh \times cwh}$ in the ambient space $\mathcal{A}$ is upper-bounded as such:

$$\begin{aligned}\texttt{capacity}[\text{in bits}] &\approx \log_2 \xi_M \ \operatorname{Vol} B_{\operatorname{rank} M}\left[\cdot, \epsilon(\tau)\right] \\ &= \log_2 \xi_M \ \frac{\pi^{(\operatorname{rank} M)/2} \ \epsilon(\tau)^{\operatorname{rank} M}}{\Gamma\left(\frac{\operatorname{rank} M}{2} + 1\right)} \\ &= \frac{\operatorname{rank} M}{2} \log_2 \pi + (\operatorname{rank} M) \log_2 \epsilon - \frac{\ln \Gamma\left(\frac{\operatorname{rank} M}{2} + 1\right)}{\ln 2} + \sum_{\sigma_M^i \in \Sigma_M : \sigma_M^i > 0} \min(\log_2 \sigma_M^i, 0).\end{aligned}$$

Bound validity: If $\epsilon(\tau) \leq \rho/2\mu$ with $\mu = \max\{\sigma_1, \ldots, \sigma_{cwh}, 1\}$ or, equivalently when $\tau \geq 20 \log_{10}(2\mu\sqrt{cwh})$. Assuming $\epsilon(\tau)$ not too small (see Bound 11 for small $\epsilon(\tau)$).

Similarly to Bounds 4 and 8, when the radius ϵ is too small, the volume approximation is poor and we resort to exact counting instead:

Bound 11: Gray image, Linear transformation (Heuristic), PSNR constraint (high PSNR, exact count). The capacity of a gray image $\boldsymbol{x}_g$ under a low minimum PSNR constraint τ and a linear transformation $M \in \mathbb{R}^{cwh \times cwh}$ in the ambient space $\mathcal{A}$ is upper-bounded as such:

$$\begin{aligned}\texttt{capacity}[\text{in bits}] &\approx \log_2 \xi_M \ \texttt{PointsInHypersphereMitchell}\left(\operatorname{rank} M, \epsilon\right) \\ &= \log_2 \texttt{PointsInHypersphereMitchell}\left(\operatorname{rank} M, \epsilon\right) \ + \sum_{\sigma_M^i \in \Sigma_M : \sigma_M^i > 0} \min(\log_2 \sigma_M^i, 0)\end{aligned}$$

with PointsInHypersphereMitchell as described in Algorithm 2.
Bound validity: If $\epsilon(\tau) \leq \rho/2\mu$ with $\mu = \max\{\sigma_1, \ldots, \sigma_{cwh}, 1\}$ or, equivalently when $\tau \geq 20 \log_{10}(2\mu\sqrt{cwh})$. Assuming $\epsilon(\tau)$ is small (otherwise, computationally intractable, see Bound 10 for large $\epsilon(\tau)$).

And finally, we have the non-trivial intersection case, where we need to account for the clipping of the watermarked images to $C_{\mathcal{I}}$, the cube of all possible images:

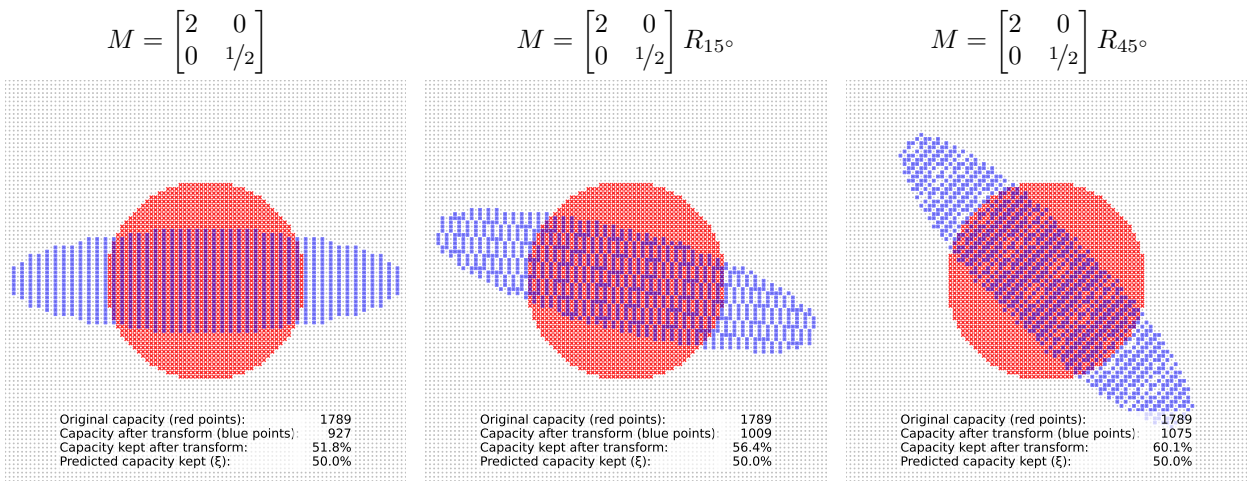

Figure 9 Quantization can result in higher capacities than predicted by Equation (6). Showing how rotations of a matrix with singular values smaller and larger than 1 can affect the capacity of a linear transform due to the quantization effects. Despite Equation (6) predicting factor $\xi = 0.5$ for all three cases, for this M we observe larger factors of up to 0.60 in the case of 45° rotation.

Bound 12: Gray image, Linear transformation (Heuristic), PSNR constraint (medium PSNR, numerical integration). The capacity of a gray image $\boldsymbol{x}_g$ under a low minimum PSNR constraint τ and a linear transformation $M \in \mathbb{R}^{cwh \times cwh}$ in the ambient space $\mathcal{A}$ is upper-bounded as such:

$$\begin{aligned}
\mathsf{capacity}[\text{in bits}] &\approx \log_2 \left[\xi_M \, \epsilon^{\operatorname{rank} M} \operatorname{Vol} \left[\prod_{\sigma_M^i \in \Sigma_M : \sigma_M^i > 0} \left[-\frac{\rho}{2\epsilon \max(1, \sigma_M^i)}, \frac{\rho}{2\epsilon \max(1, \sigma_i)} \right] \cap B_{\operatorname{rank} M}[\mathbf{0}, 1] \right] \right] \\
&= (\operatorname{rank} M) \log_2 \epsilon(\tau) \\
&\quad + \log_2 \operatorname{Vol} \left[\prod_{\sigma_M^i \in \Sigma_M : \sigma_M^i > 0} \left[-\frac{\rho}{2\epsilon(\tau) \max(1, \sigma_M^i)}, \frac{\rho}{2\epsilon(\tau) \max(1, \sigma_M^i)} \right] \cap B_{\operatorname{rank} M}[\mathbf{0}, 1] \right] \\
&\quad + \sum_{\sigma_M^i \in \Sigma_M : \sigma_M^i > 0} \min(\log_2 \sigma_M^i, 0),
\end{aligned}$$

with the volume computed by truncating the sum in Theorem 1.
Bound validity: If $\epsilon(\tau) > \rho/2\mu$ with $\mu = \max\{\sigma_1, \ldots, \sigma_{cwh}, 1\}$ or, equivalently when $\tau \leq 20 \log_{10}(2\mu\sqrt{cwh})$. Assuming rank M not too large (otherwise numerical evaluation of the bound becomes intractable).

We would like to stress that Bounds 10 to 12 are *heuristics* and are near-exact only for axis-aligned transformations, i.e., where M has at most one non-zero value in each row or column.

Higher capacities than predicted by Bounds 10 to 12 are possible. Take $M_\theta = MR_\theta$ as in Equation (5) but multiplied with a rotation matrix R_θ:

$$R_\theta = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

The rotation does not affect the singular values hence the scaling factor $\xi_M = \xi_{M_\theta}$ and Bounds 10 to 12 are unaffected. Nevertheless, the exact count of images that remain after M_θ and Q (and thus the capacity) is much higher in the rotated case. Figure 9 shows that while ξ_{M_θ} is 0.5 for all θ, the actual capacity factor at for $\theta = 45°$ is 0.6, i.e., 20% higher. Thus Bounds 10 to 12 are not upper bounds on the capacity under a linear transformation.

Unfortunately, Bounds 10 to 12 are also not lower bounds on the capacity. Take the case of transforming with just R_θ. R_θ has a determinant 1 for all θ, hence the scaling factor ξ_{R_θ} is also 1 and therefore, the capacity should be unchanged. However, as can be seen in Figure 10, the empirical $\hat{\xi}_{R_{45°}}$ is just 0.837 when we rotate the disk by $\theta = 45°$. While this particular case has been studied analytically by Vladimirov (2015, Theorem

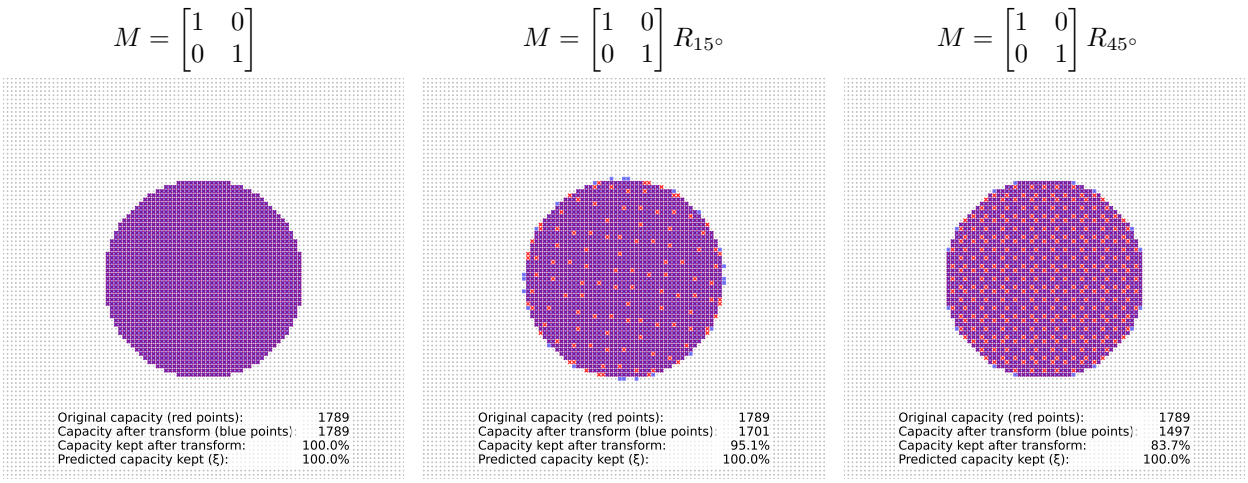

Figure 10 Quantization can result in lower capacities than predicted by Equation (6). Showing how rotations of a disk can affect the capacity of a linear transform due to the quantization effects. Despite Equation (6) predicting factor $\xi = 1$ for all three cases, for rotations we observe larger factors of as little as 0.837 in the case of 45° rotation.

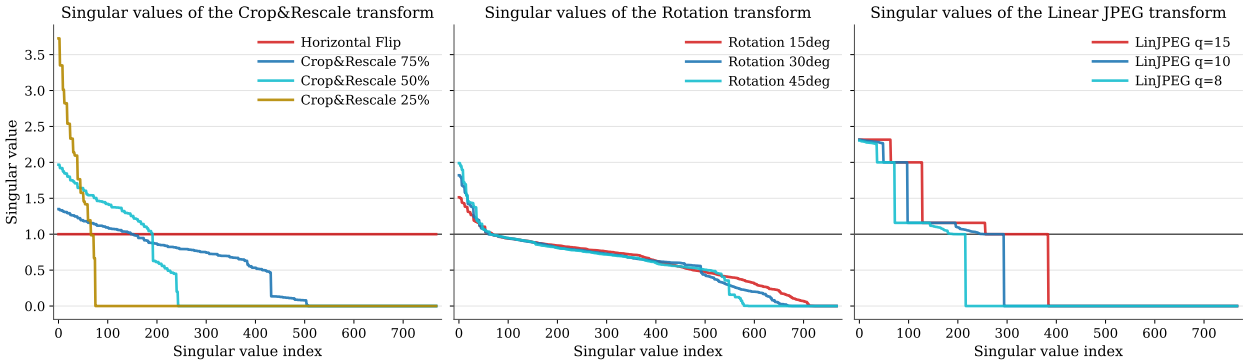

Figure 11 Singular values of the linear transformations. Plotted are the singular values of the linear transformations, sorted in descending order.

19) who demonstrated that for large enough disks this reduction is $\hat{\xi}_{R_\theta} = 1 - (\cos\theta + \sin\theta - 1)^2$, convenient results for general linear operators M are unlikely to be possible.

Nevertheless, for practical purposes, we believe that Bounds 10 to 12 are mostly lower bounds. For instance, when we have both "squish" and "stretch" axes, i.e., singular values both smaller and larger than 1 —which is the case for the transformations we consider— then the true capacity can be much higher than predicted due to the rounding operation filling the space better as seen in Figure 9. Therefore, in general, we would consider this bound to be a good heuristic.

To evaluate the effect of Bounds 10 to 12 on the watermarking capacity, we considered horizontal flipping, cropping and rescaling, rotation (around the centre of the image), as illustrated in Figure 5. The construction of the respective matrices is described in Sections G.3.3, G.3.5 and G.3.6. As can be seen in Figure 11 they all, except for the horizontal flip, have singular values above and below 1 and are rank deficient. Figure 6 has Bounds 10 to 12 for the Crop&Rescale and the Rotation transformations and compares them with the bounds without robustness constraints from Section 3.2. Rotations have a roughly 2 bpp decrease in capacity mostly driven by the loss of capacity at the corners and the effects of the interpolation. As expected, aggressive crops reduce the capacity significantly. At 40 dB, cropping to 0.25 of the image results in about 0.5 bpp, down from more than 3 bpp without the robustness constraint. Still, 0.5 bpp implies capacity of 98, 304 bits for a 256×256px image, considerably larger capacity than what we observe in practice. Therefore, robustness to augmentations does not seem to explain the much lower capacities that we see in practice.

F.3 Conservative bounds

While we are confident in the heuristic Bounds 10 to 12, it is nevertheless possible that, in practice, capacities suffer from a similar problem as the rotation R_θ (Figure 10) and hence are lower than predicted by the heuristic bounds. Unfortunately, due to the non-linear nature of the quantizer Q and the curse of dimensionality, providing an actual lower bound for ξ_M proves difficult. Still, we outline one approach here which, while extremely conservative, is a valid lower bound.

The idea of this conservative bound is to find an upper bound for the cardinality of the preimage of any transformed image, that is, an upper bound to how many images would be mapped to the same transformed image under M and Q. Every untransformed point (image) $\boldsymbol{x}$ in $C_\mathcal{I} \cap B_{cwh}[\boldsymbol{x}_g, \epsilon] \cap \mathbb{Z}^{cwh}$ maps to a point $\boldsymbol{x}'$ in $C_\mathcal{I} \cap \mathbb{Z}^{cwh}$ after being transformed by $\mathsf{Q}[M\boldsymbol{x}]$. However, multiple images can map to the same image by the transformation, i.e., $|M^+\mathsf{Q}^{-1}[\boldsymbol{x}'] \cap C_\mathcal{I} \cap B_{cwh}[\boldsymbol{x}_g, \epsilon] \cap \mathbb{Z}^{cwh}|$ might be more than 1. Here M^+ is the (Moore–Penrose) pseudo-inverse of M and $\mathsf{Q}^{-1}[\boldsymbol{x}']$ is the set of points that Q maps to $\boldsymbol{x}'$, thus $M^+\mathsf{Q}^{-1}[\boldsymbol{x}']$ are all points $\boldsymbol{x}$ such that $\mathsf{Q}[M\boldsymbol{x}] = \boldsymbol{x}'$. Define N to be the number of points available for watermarking without the robustness constraint (i.e., $2^{\text{capacity[in bits]}}$) and n to be the number of unique images that are robust to the transformation $\mathsf{Q}M$:

$$\begin{aligned} N &= \left|C_\mathcal{I} \cap B_{cwh}[\boldsymbol{x}_g, \epsilon] \cap \mathbb{Z}^{cwh}\right|, \\ n &= \left|\left\{\boldsymbol{x}' \in C_\mathcal{I} \cap \mathbb{Z}^{cwh} \mid \exists \boldsymbol{x} \in C_\mathcal{I} \cap B_{cwh}[\boldsymbol{x}_g, \epsilon] \cap \mathbb{Z}^{cwh} \text{ such that } \boldsymbol{x}' = \mathsf{Q}[M\boldsymbol{x}]\right\}\right|. \end{aligned} \tag{7}$$

In other words, n is the capacity that we can achieve while being robust to the linear transformation M. Note that the scaling factor ξ_M is precisely n/N. Section 3.2 was concerned with finding N, here we will provide an upper bound to n. While obtaining n directly is difficult, if we know that every transformed point has at most K preimage points, i.e.:

$$|M^+\mathsf{Q}^{-1}[\boldsymbol{x}] \cap C_\mathcal{I} \cap B_{cwh}[\boldsymbol{x}_g, \epsilon(\tau)] \cap \mathbb{Z}^{cwh}| \leq K, \ \forall \boldsymbol{x}$$

then we know that $n \geq \lceil N/K \rceil$. In other words we have that the scaling factor must be lower-bounded as $\xi_M \geq 1/K$. So the problem of finding a lower bound to ξ_M has reduced to obtaining K, an upper bound to the number of preimage points that any transformed image can have.

The quantization operation Q maps a hypercube of side 1 to a given image, regardless of whether the rounding or floor quantization is used. The maximum number of points that can fit in the preimage under M of such a unit cube is the K we are looking for. The preimage of a hypercube under a linear operation M, when restricted to a hypersphere of radius ϵ, can be over-approximated with a zonotope (with proof in Section H):

Theorem 2. *Given a hypercube $C = \boldsymbol{b} + [0,1]^n \subset \mathbb{R}^n$, $\boldsymbol{b} \in \mathbb{R}^n$ and a possibly singular matrix $M \in \mathbb{R}^{n\times n}$ giving rise to the map $f_M(\boldsymbol{x}) = M\boldsymbol{x}$, the supremum of the volume of the preimage of C under M when intersected with a hypersphere of radius r is upper-bounded as:*

$$\sup_{\boldsymbol{c}\in\mathbb{R}^n} \operatorname{Vol}\left[f_M^{-1}(C) \cap B_n[\boldsymbol{c}, r]\right] \leq r^n \operatorname{Vol}\left[\prod_{i=1}^{n}\left[-\frac{\beta_i}{r}, \frac{\beta_i}{r}\right] \cap B_n[\boldsymbol{0}, 1]\right], \tag{8}$$

with

$$\boldsymbol{\beta} = \operatorname{abs}\left[\frac{1}{2}\Sigma^+ U^\top\right]\mathbf{1}_n + \left[\mathbf{0}^\top_{\text{rank}[M]}, r\mathbf{1}^\top_{n-\text{rank}[M]}\right]^\top, \tag{9}$$

where $U\Sigma V^\top$ is the SVD decomposition of M and Σ^+ is the pseudo-inverse of the diagonal matrix Σ (i.e., the reciprocal of the non-zero elements of Σ). The volume of the box-ball intersection can be computed with Theorem 1.

Now, Theorem 2 gives us an upper-bound for K, the number of images in a PSNR ball that are "collapsed" onto the same image after a linear transformation M.

Bound 13: Gray image, Linear transformation (Conservative), PSNR constraint. The capacity of a gray image $\boldsymbol{x}_g$ under a linear transformation $M \in \mathbb{R}^{cwh \times cwh}$ in the ambient space $\mathcal{A}$ is lower-bounded as

$$\begin{aligned}
\texttt{capacity under } M[\text{in bits}] &= \log_2 n \\
&\geq \log_2 \frac{N}{K} \\
&= \log_2 N - \log_2 K \\
&\geq \texttt{capacity}[\text{in bits}] - cwh \log_2 \epsilon - \log_2 \operatorname{Vol}\left(\prod_{j=1}^{cwh} \left[-\frac{\beta_j}{\epsilon}, \frac{\beta_j}{\epsilon} \right] \cap B_{cwh}\left[\mathbf{0}, 1\right] \right),
\end{aligned}$$

where β is computed as in Theorem 2, the volume of the intersection as in Theorem 1, and `capacity`[in bits] is the capacity without the linear robustness constraint.
Bound validity: Assuming cwh not too large (otherwise numerical evaluation of the bound becomes intractable).

This bound can be numerically unstable and is tractable only for low dimensions cwh, high numerical precision and large amount of sum terms kept when evaluating the intersection volume. Nevertheless, even with this extremely conservative bound, which restricts the capacity significantly more than the heuristic Bounds 10 to 12, we still observe 0.005 bpp for the most aggressive crop transform (see Table 1). This might seem small but it still amounts to over 900 bits for a 256×256px image. For the other transformations we get capacities in excess of 3,000 bits. Therefore, even in this strongly conservative setup we still see capacities significantly larger than what we observe in practice.

Bound 13 relies on a severe over-approximation of a zonotope with an axis-aligned box, meaning that it is extremely conservative. The product of singular values heuristic Bounds 10 to 12 are probably much closer to the true capacity (and in fact, possibly also conservative, as observed in the rotated ellipsoid case, Figure 9). Therefore in general we will use the heuristic Bounds 10 to 12.

F.4 Robustness to compression

Beyond geometric transformations, watermarks should also be robust to compression. Compression happens during normal image processing, even in the absence of attacks, and tends to strip a lot of information from an image. Hence, it is possible that it imposes a very strong reduction on the capacity of watermarking. The problem with compression methods, though, is that they are highly non-linear and difficult to study analytically. Here, we analyse JPEG, arguably the most widely used image compression method. While JPEG is non-linear, it has only one non-linear step. We can linearize this step without deviating too much from the behaviour of classic JPEG, resulting in LinJPEG.

The standard JPEG compression and decompression consists of the following steps (Wallace, 1991; Wikipedia, 2025):

1. Convert the colour space from `RGB` to `YCbCr`. Linear and invertible. Section G.2.2.
2. Downsample the `Cb` and `Cr` channels, typically by a factor of 2 in both dimensions. Linear but rank-deficient. Section G.2.5.
3. Divide each channel into 8×8px tiles and apply steps 4–6 to each tile individually. The `Y` channel would have 4× the tiles of the chroma channels because of the downsampling. Section G.2.9.
4. Perform Discrete Cosine Transform (DCT) over each tile Linear and invertible. Section G.2.7.
5. Do element-wise multiplication with a quantization matrix (matrix values depend on the quality setting). Linear and invertible.
6. **Round the pixel values to integers**. [non-linear, due to the rounding]
7. Perform Inverse DCT over each tile. Linear and invertible. Section G.2.8.
8. Upsample the `Cb` and `Cr` channels. Linear and invertible. Section G.2.6.

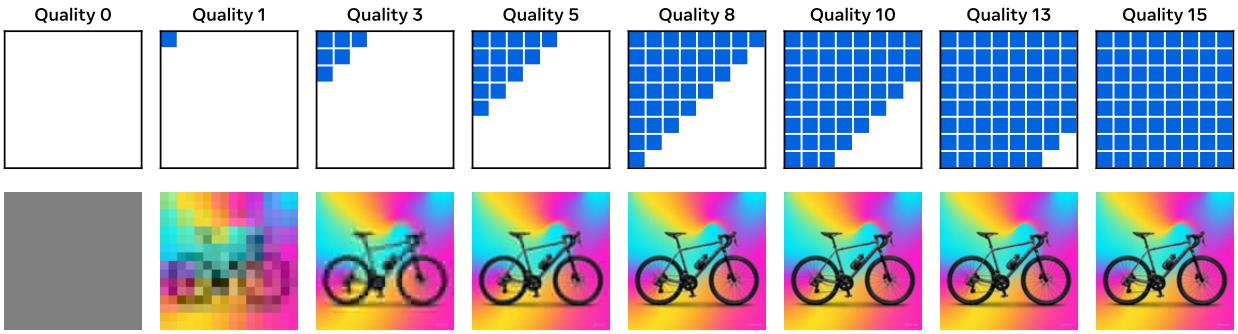

Figure 12 Illustration of LinJPEG at various strengths. We show the effect of compressing an image with LinJPEG, our linearised variant of JPEG compression, at various quality settings. The quality refers to the number of DCT diagonals kept for each 8×8px tile. Quality 0 means no DCT coefficients are kept, while quality 15 means that the image is unchanged. Quality 10 and above produces little visual artifacts and is almost indistinguishable from the original image.

9. Convert the color space from `YCbCr` to `RGB`. Linear and invertible. Section G.2.3.

The compression comes from step 2, which downsamples the chroma channels, and step 6 which efficiently encodes the frequencies which have been rounded down to 0 via entropy coding. The lossless compression after the quantization does not change the pixel values and does not affect capacity, hence we ignore it. Step 6 is the only non-linear step which prevents applying Bounds 10 to 13 to JPEG compression. Let's take a closer look at a single tile (we take the example from (Wikipedia, 2025)). When converted to DCT space (G), the higher frequencies are represented in the bottom and right sides of the matrix. As high-frequency components are less perceptible, the quantization matrices Q are designed to attenuate them more, resulting in more of them becoming 0 after rounding:

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.12 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.87 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix}$$

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

$$Q\ \text{round}(G/Q) = \begin{bmatrix} -416 & -33 & -60 & 32 & 48 & -40 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & -24 & -56 & 19 & 26 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -42 & 13 & 80 & -24 & -40 & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ -42 & 17 & 44 & -29 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ 18 & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}$$

Therefore, the compression mechanism of Step 6 acts by producing zeros in the lower right corner of the DCT components G. That gives us a simple way to linearize Step 6: simply drop the highest frequencies, regardless of the value of their components. This is equivalent to a linear operator with a diagonal matrix with 1s and 0s on the diagonal and is explicitly constructed in Section G.2.10. As all the other steps are linear,

we can compose them with our alternative to Step 6 to obtain LinJPEG: a linear operator that is a very close approximation to JPEG. LinJPEG is formally constructed in Section G.2.12. This is a similar strategy to JPEG-Mask proposed by (Zhu et al., 2018) as a way to make JPEG differentiable. However, rather than differentiating through the compression, here we are interested in its singular values.

The way we have implemented this is to map a quality factor that is an integer from 0 to 15 inclusive to the number of diagonals that are being zeroed out. See Figure 12 for examples. While classic JPEG is designed to have fixed perceptual quality and variable file size, our LinJPEG instead has variable perceptual quality and fixed file size.

As can be seen from the capacity plot Figure 6 right for LinJPEG, the higher the compression rate, the lower the capacity, as expected. At first, it may seem strange that quality 15, where no frequency components are dropped, still has a 50% reduction in capacity. However, that is due to the downsampling of the chroma channels, which reduces the effective number of pixels we have for watermarking by half. This can also be seen by half of the singular values being 0 (Figure 11 right). Interestingly, again, even with relatively high compression rates (quality 8) we still observe capacities of more than 1 bpp at 40 dB. Finally, as can be seen in Table 1, even the conservative bound from Section F.3 gives us 0.13 bpp or 25,559 bits for a 256×256px image. Therefore, although (Lin)JPEG compression removes a lot of information from the image, it still leaves plenty of watermarking capacity.

G Augmentations as linear transformations

This section describes a series of linear transformations used for the robustness bounds in Section 3.4. Each transformation is defined by a matrix A and a bias vector $\boldsymbol{b}$, and acts on an input vector $\boldsymbol{x}$ as $f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$. Some transformations are compositions of others, as described below. We will use the pipe operator | for left-to-right composition, rather than the classical $\circ$ operator for right-to-left composition.

G.1 Generic linear transformation

A generic linear transformation has the form $f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$, where A is a matrix and $\boldsymbol{b}$ is a bias vector. This transformation can be applied to vectors or images (flattened as vectors, with each third corresponding to one channel).

$$f(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$$

G.2 LinJPEG and its building blocks

G.2.1 Pixel-wise transformation

The pixel-wise transform applies a given linear transformation $g(\boldsymbol{p}) = B\boldsymbol{p} + \boldsymbol{c}$ independently to each pixel across an image. If the base transformation maps c_{in} input channels to c_{out} output channels (i.e., $B \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}}}$) the overall matrix A is constructed as a block matrix, where each block is an identity matrix of size $w \times h$ (the number of pixels), scaled by the corresponding entry in the base transformation's matrix. The bias $\boldsymbol{b}$ is the base bias $\boldsymbol{c}$ repeated for every pixel.

$$A = \begin{bmatrix} B_{1,1} I_{wh} & \dots & B_{1,c_{\text{in}}} I_{wh} \\ \vdots & \ddots & \vdots \\ B_{c_{\text{out}},1} I_{wh} & \dots & B_{c_{\text{out}},c_{\text{in}}} I_{wh} \end{bmatrix} \qquad \boldsymbol{b} = \left.\begin{bmatrix} \boldsymbol{c} \\ \vdots \\ \boldsymbol{c} \end{bmatrix}\right\} wh \text{ times}$$

G.2.2 RGB to YCbCr conversion

Color-space transformations are pixel-wise transformations. To convert the RGB color space to the YCbCr color space we have the base matrix:

$$B = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331364 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix}$$

The bias is $\boldsymbol{c} = [0, 128, 128]$. We can use the pixel-wise transform we defined above to obtain the linear transform for applying this across the image:

$$(A, \boldsymbol{b}) = \texttt{Pixel-wise Transformation}[B, \boldsymbol{c}].$$

G.2.3 YCbCr to RGB conversion

To convert the YCbCr color space back to RGB, we need the composition of two transforms: First, subtract 128 from Cb and Cr channels, then apply the matrix:

$$B = \begin{bmatrix} 1 & 0 & 1.402 \\ 1 & -0.344136 & -0.714136 \\ 1 & 1.772 & 0 \end{bmatrix}$$

Thus, we have the bias

$$\boldsymbol{c} = B[0, -128, -128]^\top.$$

To get a transformation for the whole image, we again use the pixel-wise transform we defined above:

$$(A, \boldsymbol{b}) = \texttt{Pixel-wise Transformation}[B, \boldsymbol{c}].$$

G.2.4 Take rows and columns transformation

This transformation selects specific rows and columns from an input image, effectively downsampling or upsampling. The matrix A is constructed such that each output pixel corresponds to a specific input pixel, with a 1 at the appropriate position and 0 elsewhere. The bias $\boldsymbol{b}$ is zero. Assume the indices are provided as two lists `row indices` and `col indices`. Then we can set the non-zero elements of A as:

```
for ir, r in enumerate(row indices):
    for ic, c in enumerate(col indices):
        for chan in range(channels):
            A[
                chan * (len(row indices) * len(col indices)) + ir * len(col indices) + ic,
                chan * w * h + r * w + c,
            ] = 1
```

G.2.5 Downsampling transformation

The downsample transformation reduces the resolution of an image by a fixed factor. The matrix A selects every k-th row and column. The bias $\boldsymbol{b}$ is zero.

$$(A, b) = \texttt{TakeRowsAndColumns}\big[\texttt{row\_indices} = [1, 1{+}k, \ldots, h{-}k{+}1], \texttt{col\_indices} = [1, 1{+}k, \ldots, w{-}k{+}1]\big].$$

G.2.6 Upsampling transformation

The upsampling transformation increases the resolution by repeating rows and columns k times. The matrix A selects every row and column k times. The bias $\boldsymbol{b}$ is zero.

$$(A, b) = \texttt{TakeRowsAndColumns}\big[\texttt{row\_indices}{=}[\underbrace{1, \ldots, 1}_{\times k}, \ldots, \underbrace{h \ldots, h}_{\times k}],\ \texttt{col\_indices}{=}[\underbrace{1, \ldots, 1}_{\times k}, \ldots, \underbrace{w \ldots, w}_{\times k}]\big].$$

G.2.7 Discrete Cosine Transform (DCT) for 8x8 blocks

The Dct8x8 transformation applies the DCT to a single-channel 8×8px block. It first subtracts 128 from each pixel so that the values are centred at 0, then applies the DCT matrix. The subtraction can be done with:

$$g(\boldsymbol{x}) = I_{64}\boldsymbol{x} - 128 \cdot \mathbf{1}.$$

The DCT matrix is constructed using the four-dimensional tensor $D \in \mathbb{R}^{8\times8\times8\times8}$:

$$D_{x,y,u,v} = \cos\left(\frac{(2x+1)u\pi}{16}\right)\cos\left(\frac{(2y+1)v\pi}{16}\right)$$
$$D' = \text{reshape}(0.25\ \boldsymbol{\alpha}\boldsymbol{\alpha}^\top, (1, 64))\ \odot\ \text{reshape}(D, (64, 64)),$$

where $\boldsymbol{\alpha}_1 = 1/\sqrt{2}$, $\boldsymbol{\alpha}_i = 1$ for $2 \le i \le 8$ and $\odot$ is an element-wise multiplication with broadcasting. This results in

$$h(\boldsymbol{x}) = D'\boldsymbol{x}.$$

The overall transformation is:

$$f(\boldsymbol{x}) = (g \ \mid\ h)(\boldsymbol{x}).$$

G.2.8 Inverse DCT for 8x8 blocks

The iDct8x8 transformation applies the inverse DCT to a single-channel 8×8px block. The matrix is constructed similarly to the DCT, but with the roles of x, y and u, v swapped. After the transformation, 128

is added to each pixel to restore the original range.

$$
\begin{aligned}
g(\boldsymbol{x}) &= I_{64}\boldsymbol{x} + 128 \cdot \mathbf{1}, \\
D_{x,y,u,v} &= \cos\left(\frac{(2u+1)x\pi}{16}\right)\cos\left(\frac{(2v+1)u\pi}{16}\right), \\
D' &= \text{reshape}(0.25\ \boldsymbol{\alpha}\boldsymbol{\alpha}^\top, (1, 64))\ \odot\ \text{reshape}(D, (64, 64)), \\
h(\boldsymbol{x}) &= D'\boldsymbol{x}, \\
f(\boldsymbol{x}) &= (h\ \mid\ g)(\boldsymbol{x}).
\end{aligned}
$$

G.2.9 Tiling transformation

The tiling transformation divides an image into tiles and applies a linear transform to each tile. This operates over a single channel. Define the linear transformation for a tile as $g(\boldsymbol{t}) = B\boldsymbol{t} + \boldsymbol{c}$, with $B \in \mathbb{R}^{\texttt{ts}^2 \times \texttt{ts}^2}, \boldsymbol{t}, \boldsymbol{c} \in \mathbb{R}^{\texttt{ts}^2}$, `ts` being the size of the tile (typically 8 for our use-cases). We expect that the `ts` divides the width w and height h of the image. The matrix A is constructed by placing the base transform's matrix along the diagonal for each tile, so that each tile is transformed independently. The bias $\boldsymbol{b}$ is constructed by tiling the base bias for each tile. The A and $\boldsymbol{b}$ of the resulting transformation can then be computed using this pseudo-code:

```
hor_tiles = width // ts
ver_tiles = height // ts

tiles = split(B, ts, axis=0)
tiles = [split(s, ts, axis=1) for s in tiles]  # ts lists of ts tiles of ts x ts size

per_n_rows = block_matrix(
   [[block_diag([tile] * hor_tiles) for tile in htiles] for htiles in tiles]
)

A = block_diag([per_n_rows] * ver_tiles)

bias_tiles = split(transform.bias, tile_side, axis=0)
bias = concatenate([tile(bias_tile, hor_tiles) for bias_tile in bias_tiles])
b = tile(bias, ver_tiles)
```

G.2.10 JPEG filter

This filter transformation drops the lowest frequencies in an 8×8 block according to a quality factor. This is the linearized replacement for the quantization operation in the standard JPEG compression algorithm, as discussed in Section F.4. The quality factor q designates the number of diagonals kept: from $q = 0$ for the worst quality where all diagonals are masked off, to $q = 15$ where all diagonals are kept. The resulting $A \in \mathbb{R}^{64\times 64}$ can be constructed with the following pseudo-code:

```
matrix = triu(ones(8, 8), k=16 - 8 - q)[:, ::-1]
A = diag(matrix.flatten())
```

G.2.11 Per-Channel transformation

The Per-Channel Transform applies a separate linear transformation to each channel of an image. The matrix A is block-diagonal, with each block being the matrix for the corresponding channel. The bias $\boldsymbol{b}$ is the concatenation of the biases for each channel. Given c channels, each with its own linear transformation $A_i, \boldsymbol{b}_i$, $1 \le i \le c$, the resulting linear transform has:

$$
\begin{aligned}
A &= \text{diag}(A_1, \ldots, A_c), \\
\boldsymbol{b} &= \begin{bmatrix} \boldsymbol{b}_1 \\ \vdots \\ \boldsymbol{b}_c \end{bmatrix}.
\end{aligned}
$$

G.2.12 LinJPEG transform

The LinJPEG transformation approximates JPEG compression with quality q as a sequence of linear transforms, as explained in Section F.4. In a nutshell, we need to convert the colour space from `RGB` to `YCbCr` and then downsample the `Cb` and `Cr` channels by a factor of 2. We then need to apply the DCT, filter and iDCT operations on 8×8px tiles of each channel. Then we need to upsample the `Cb` and `Cr` channels to restore them to the original resolution and convert the colour space back to `RGB`. We can build a single linear transform with its A and $\boldsymbol{b}$ doing all that by composing the building blocks defined above:

$$
\begin{aligned}
(A_Y, \boldsymbol{b}_Y) &= \texttt{Tile}\,[\texttt{Dct8x8} \;\mid\; \texttt{JpegFilter}[q] \;\mid\; \texttt{iDct8x8}, \texttt{width} = w, \texttt{height} = h] \\
(A_C, \boldsymbol{b}_C) &= \texttt{Downsample}[k = 2] \\
&\quad \mid \texttt{Tile}\big[\texttt{Dct8x8} \;\mid\; \texttt{JpegFilter}[q] \;\mid\; \texttt{iDct8x8}, \texttt{width} = w/2, \texttt{height} = h/2\big] \\
&\quad \mid \texttt{Upsample}[k = 2] \\
(A, \boldsymbol{b}) &= \texttt{RGBToYCbCr} \;\mid\; \texttt{PerChannelTransform}[(A_Y, \boldsymbol{b}_Y), (A_C, \boldsymbol{b}_C), (A_C, \boldsymbol{b}_C)] \;\mid\; \texttt{YCbCrToRGB}.
\end{aligned}
$$

G.3 Pixel mapping transforms

A number of transforms can be defined via a map $\mu(x, y) \mapsto (u, v)$ that says which point (u, v) in the original image corresponds to a pixel (x, y) in the transformed image. Note while x and y are integer coordinates, u and v need not be. Thus, some sort of interpolation will be needed. Here we consider both nearest neighbour and bilinear interpolation. It is interesting that in both of these case, given a fixed map μ, its application with interpolation is a linear operation. If μ is itself parameterized (e.g., the angle of rotation), then the transformation generally is not linear in the parameter. That is why we consider μ with different parameters to be distinct transformations.

G.3.1 Nearest neighbour pixel mapping

The matrix A is binary, with $A_{i,j} = 1$ if output pixel i maps to input pixel j. Below we show the construction for a pixel map `mu` and a single channel. For multi-channel images, the same transformation can be applied to each channel using `PerChannelTransform`.

```
mesh_x, mesh_y = meshgrid(range(width), range(height))
mesh_x = flatten(mesh_x)
mesh_y = flatten(mesh_y)
mapped_x, mapped_y = vectorize(mu)(mesh_x, mesh_y)
mapped_x = clip(round(mapped_x), 0, width - 1)
mapped_y = clip(round(mapped_y), 0, height - 1)

A = zeros(width * height, width * height)
target_indices = mesh_x + mesh_y * width
source_indices = mapped_x + mapped_y * width
A[target_indices, source_indices] = 1
```

G.3.2 Bilinear pixel mapping

The matrix A is constructed so that each output pixel is a weighted sum of the four nearest input pixels, with weights determined by the mapping. Below we show the construction for a pixel map `mu` and a single channel. For multi-channel images, the same transformation can be applied to each channel using `PerChannelTransform`.

```
mesh_x, mesh_y = meshgrid(range(width), range(height))
mesh_x = flatten(mesh_x)
mesh_y = flatten(mesh_y)
mapped_x, mapped_y = vectorize(mu)(mesh_x, mesh_y)

def get_corners(mapped: array, range: int) -> tuple[array, array]:
```

```
    l = floor(mapped)
    u = ceil(mapped)
    # we need to ensure that the lower and the upper bounds are not the same
    u = where(u == l, l + 1, u)
    return l, u

mapped_x_l, mapped_x_u = get_corners(mapped_x, width)
mapped_y_l, mapped_y_u = get_corners(mapped_y, height)

denom = (mapped_x_u - mapped_x_l) * (mapped_y_u - mapped_y_l)
w11 = (mapped_x_u - mapped_x) * (mapped_y_u - mapped_y) / denom
w12 = (mapped_x_u - mapped_x) * (mapped_y - mapped_y_l) / denom
w21 = (mapped_x - mapped_x_l) * (mapped_y_u - mapped_y) / denom
w22 = (mapped_x - mapped_x_l) * (mapped_y - mapped_y_l) / denom

# Create the matrix for a single channel
target_indices = mesh_x + mesh_y * width

mapped_x_l = clip(mapped_x_l, 0, width - 1)
mapped_x_u = clip(mapped_x_u, 0, width - 1)
mapped_y_l = clip(mapped_y_l, 0, height - 1)
mapped_y_u = clip(mapped_y_u, 0, height - 1)

indices_11 = mapped_x_l + mapped_y_l * width
indices_12 = mapped_x_l + mapped_y_u * width
indices_21 = mapped_x_u + mapped_y_l * width
indices_22 = mapped_x_u + mapped_y_u * width

A = zeros(width * height, width * height)
A[target_indices, indices_11] += w11
A[target_indices, indices_12] += w12
A[target_indices, indices_21] += w21
A[target_indices, indices_22] += w22
```

G.3.3 Horizontal flip transformation

The pixel mapping is $\mu : (x, y) \mapsto (\texttt{width} - 1 - x, y)$. The transformation can be applied with either the nearest neighbour or the bilinear interpolation methods, by default we use nearest neighbour but bilinear should give the exact same result here.

G.3.4 Vertical flip transformation

The pixel mapping is $\mu : (x, y) \mapsto (x, \texttt{height} - 1 - y)$. The transformation can be applied with either the nearest neighbour or the bilinear interpolation methods, by default we use nearest neighbour but bilinear should give the exact same result here.

G.3.5 Centre crop and rescale transformation

This transformation crops the centre of an image to a given scale and rescales it to the original size. The pixel mapping μ for a fixed scale s is:

$$(x, y) \mapsto \left(\left(x - \frac{\texttt{width}}{2}\right) \cdot s + \frac{\texttt{width}}{2},\ \left(y - \frac{\texttt{height}}{2}\right) \cdot s + \frac{\texttt{height}}{2}\right).$$

The transformation can be applied with either the nearest neighbour or the bilinear interpolation methods, by default we use bilinear.

G.3.6 Rotation transformation

This rotation transformation rotates an image around its centre by a fixed angle θ. The pixel mapping μ is:

$$(x, y) \mapsto \Bigg(\left(x - \frac{\texttt{width}}{2}\right) \cos\theta - \left(y - \frac{\texttt{height}}{2}\right) \sin\theta + \frac{\texttt{width}}{2}, \\ \left(x - \frac{\texttt{width}}{2}\right) \sin\theta + \left(y - \frac{\texttt{height}}{2}\right) \cos\theta + \frac{\texttt{height}}{2} \Bigg).$$

The transformation can be applied with either the nearest neighbour or the bilinear interpolation methods, by default we use bilinear.

H Proof of Theorem 2

Let $\boldsymbol{b}$ be a point in $\mathbb{R}^n$ and $C = \boldsymbol{b} + [0,1]^n$ be the hypercube with a corner at $\boldsymbol{b}$. Given $\boldsymbol{x} \in C$, the preimage of f_M can be obtained using the Moore-Penrose pseudo-inverse M^+ of M:

$$f_M^{-1}(\boldsymbol{x}) = \left\{ M^+\boldsymbol{x} + [I - M^+M]\boldsymbol{w} \;\middle|\; \boldsymbol{w} \in \mathbb{R}^n \right\}.$$

Hence:

$$\begin{aligned} f_M^{-1}(C) &= \left\{ M^+\boldsymbol{b} + M^+\boldsymbol{p} + (I - M^+M)\boldsymbol{w} \;\middle|\; \boldsymbol{w} \in \mathbb{R}^n, \boldsymbol{p} \in [0,1]^n \right\} \\ &= \left\{ M^+\boldsymbol{b} \right\} \;\oplus\; M^+[0,1]^n \;\oplus\; (I - M^+M)\mathbb{R}^n \\ &= \left\{ M^+\left(\boldsymbol{b} + \frac{1}{2}\right) \right\} \;\oplus\; \frac{1}{2}M^+[-1,1]^n \;\oplus\; (I - M^+M)\mathbb{R}^n, \end{aligned} \tag{10}$$

where $\oplus$ is the Minkowski sum. Take $U\Sigma V^\top = M$ to be the SVD decomposition of M. U and V are orthonormal matrices, while Σ is a diagonal matrix with non-negative entries. We will assume that the singular values on the diagonal are sorted in descending order, hence the first $\operatorname{rank}[M]$ values on the diagonal are non-zero and the rest are zero. The pseudo-inverse of M can be conveniently expressed as $M^+ = V\Sigma^+U^\top$. We can use that the pseudo-inverse of diagonal matrices can be constructed by taking the reciprocals of the non-zero elements on the diagonal, leaving the zeros unchanged. Thus we have:

$$\begin{aligned} (I - M^+M)\mathbb{R}^n &= (I - V\Sigma^+U^\top U\Sigma V^\top)\mathbb{R}^n \\ &= (I - V\Sigma^+\Sigma V^\top)\mathbb{R}^n && (U^\top U = I \text{ as } U \text{ is orthonormal}) \\ &= (VV^\top - V\Sigma^+\Sigma V^\top)\mathbb{R}^n && (VV^\top = I \text{ as } V \text{ is orthonormal}) \\ &= V(I - \Sigma^+\Sigma)V^\top\mathbb{R}^n \\ &= V(I - \operatorname{diag}[\mathbf{1}_{\operatorname{rank}[M]}, \mathbf{0}_{n-\operatorname{rank}[M]}])V^\top\mathbb{R}^n \\ &= V \operatorname{diag}[\mathbf{0}_{\operatorname{rank}[M]}, \mathbf{1}_{n-\operatorname{rank}[M]}]V^\top\mathbb{R}^n \\ &= \tilde{V}\tilde{V}^\top\mathbb{R}^n && (\tilde{V} = V_{[:,\ \operatorname{rank}[M]+1:]} \in \mathbb{R}^{n\times(n-\operatorname{rank}[M])}) \\ &= \tilde{V}\mathbb{R}^{n-\operatorname{rank}[M]} && (\tilde{V}^\top\mathbb{R}^n = \mathbb{R}^{n-\operatorname{rank}[M]}). \end{aligned}$$

Thus, combining with Equation (10), we have that the set of images that would be mapped by M to images in the cube C, i.e., after quantization, is the following polytope:

$$f_M^{-1}(C) = \left\{ M^+\left(\boldsymbol{b} + \frac{1}{2}\right) \right\} \;\oplus\; \frac{1}{2}V\Sigma^+U^\top[-1,1]^n \;\oplus\; \tilde{V}\mathbb{R}^{n-\operatorname{rank}[M]}.$$

The volume of the intersection in Equation (8) is maximized when the ball centre $\boldsymbol{c}$ coincides with the polytope centre $M^+(\boldsymbol{b} + 1/2)$. Furthermore, because the volume of the intersection is invariant to shifts, we can simplify the left-hand side of Equation (8) to:

$$\begin{aligned} \sup_{\boldsymbol{c}\in\mathbb{R}^n} \operatorname{Vol}\left[f_M^{-1}(C) \;\cap\; B_n\left[\boldsymbol{c}, r\right]\right] &= \operatorname{Vol}\left[\left(\frac{1}{2}V\Sigma^+U^\top[-1,1]^n \;\oplus\; \tilde{V}\mathbb{R}^{n-\operatorname{rank}[M]}\right) \cap B_n\left[\mathbf{0}, r\right]\right] \\ &= \operatorname{Vol}\left[\underbrace{\left(\frac{1}{2}V\Sigma^+U^\top[-1,1]^n \;\oplus\; r\tilde{V}[-1,1]^{n-\operatorname{rank}[M]}\right)}_{Z} \cap B_n\left[\mathbf{0}, r\right]\right], \end{aligned}$$

where we use the fact that the intersection with a ball of radius r needs to be contained within the hypercube $[-r, r]^n$, that $\tilde{V}$ has orthonormal columns and that the two sets making up the sum in Z are orthogonal.[3]

Computing the volume of this intersection in the general case is computationally intractable. However, if we over-approximate the left-hand side of the intersection (Z) with a (rotated) box, then we can use our previous results on the volume of box-ball intersections, in particular Theorem 1.

[3] Follows from the properties of the pseudo-inverse: $(M^+M)^\top = M^+M$ and $M^+MM^+ = M^+$.

To upper-bound Z with a (rotated) box we first observe that it is the Minkowski sum of two zonotopes and hence is a zonotope itself. A zonotope is defined as

$$\left\{ \boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} = \boldsymbol{c} + \sum_{i=1}^{p} \xi_i \boldsymbol{g}_i, \quad \xi_i \in [-1, 1] \quad \forall i = 1, \ldots, p \right\},$$

where $\boldsymbol{c} \in \mathbb{R}^n$ is its centre and $G = \{\boldsymbol{g}_i\}_{i=1}^p$, $\boldsymbol{g}_i \in \mathbb{R}^n$ is the set of its generators. A zonotope is, equivalently the Minkowski sum of line segments. Thus, the Minkowski sum of zonotopes is also a zonotope. Its centre is found by adding together the centres of the original zonotopes, and its set of generators is just all the generators from both shapes combined (Schneider, 2013). Now, it is clear that Z is a zonotope as it is the Minkowski sum of two other zonotopes.

The V and $\tilde{V}$ matrices simply rotate the resulting zonotope. As the ball $B_n[\mathbf{0}, r]$ is rotation invariant, we can ignore this rotation. That will help us with tightening the box approximation as the $n - \operatorname{rank}[M]$ dimensions of the second zonotope will now be automatically axis-aligned. Thus we have (up to a rotation):

$$Z = \left[\tfrac{1}{2}\Sigma^+ U^\top, \quad r I_{n[:, \operatorname{rank} M + 1:]}\right] [-1, 1]^{2n - \operatorname{rank} M} = G\, [-1, 1]^{2n - \operatorname{rank} M},$$

with $G \in \mathbb{R}^{n \times (2n - \operatorname{rank} M)}$ being its $2n - \operatorname{rank} M$ n-dimensional generators.

A zonotope with generators G is contained in an axis-aligned box $\prod_{i=1}^n [-\beta_i', \beta_i']$, where $\boldsymbol{\beta}$ is the sum of absolute values of G across the generators: $\boldsymbol{\beta}' = \operatorname{abs}[G]\, \mathbf{1}_{2n - \operatorname{rank}[M]} = \operatorname{abs}[\tfrac{1}{2}\Sigma^+ U^\top]\mathbf{1}_n + [\mathbf{0}_{\operatorname{rank}[M]}^\top, r\mathbf{1}_{n - \operatorname{rank}[M]}^\top]^\top$ (Girard, 2005; Althoff et al., 2010).

Note that this over-approximation can be extremely loose: this is the step that makes Bound 13 so conservative. However, with this, we can now apply Equation (4) and Theorem 1 for the box-ball intersection:

$$\begin{aligned}
\sup_{\boldsymbol{c} \in \mathbb{R}^n} \operatorname{Vol}\left[f_M^{-1}(C) \cap B_n[\boldsymbol{c}, r]\right] &= \operatorname{Vol}\left[\left(\frac{1}{2} V \Sigma^+ U^\top [-1, 1]^n \oplus r\tilde{V}[-1, 1]^{n - \operatorname{rank}[M]}\right) \cap B_n[\mathbf{0}, r]\right] \\
&= \operatorname{Vol}\left[\left(G\, [-1, 1]^{2n - \operatorname{rank}[M]}\right) \cap B_n[\mathbf{0}, r]\right] \\
&\leq \operatorname{Vol}\left[([-\beta_1, \beta_1] \times \cdots \times [-\beta_n, \beta_n]) \cap B_n[\mathbf{0}, r]\right] \\
&= r^n \operatorname{Vol}\left[\left(\left[-\frac{\beta_1}{r}, \frac{\beta_1}{r}\right] \times \cdots \times \left[-\frac{\beta_n}{r}, \frac{\beta_n}{r}\right]\right) \cap B_n[\mathbf{0}, 1]\right].
\end{aligned}$$

I Comprehensive results for ChunkySeal

I.1 Extended evaluation

Table 4 Extended results of Chunky Seal on SA-1B (Kirillov et al., 2023).

	Chunky Seal (ours) 1024bit, 256px	Video Seal 256bit, 256px	Video Seal 96bit, 256px	HiDDeN	MBRS	TrustMark	WAM
Capacity	1024 bits	256 bits	96 bits	48 bits	256 bits	100 bits	32 bits
PSNR	45.32 dB	44.42 dB	53.19 dB	30.41 dB	45.54 dB	42.29 dB	38.19 dB
SSIM	0.9945	0.9963	0.9995	0.9299	0.9962	0.9941	0.9842
MS-SSIM	0.9966	0.9972	0.9993	0.9062	0.9967	0.9944	0.9877
LPIPS	0.0085	0.0019	0.0028	0.2021	0.0044	0.0028	0.0446
Embedding Time	0.27 s	0.06 s	0.06 s	0.08 s	0.08 s	0.07 s	0.15 s
Extraction Time	0.05 s	0.01 s	0.01 s	0.01 s	0.01 s	0.01 s	0.02 s
Bit Acc.	99.74%	99.90%	97.92%	92.19%	98.74%	99.81%	100.00%
Bit Acc. (Horizontal Flip)	99.65%	99.89%	97.20%	64.06%	50.63%	99.81%	100.00%
Bit Acc. (Rotate 5°)	99.29%	99.37%	94.79%	80.99%	50.21%	56.88%	98.83%
Bit Acc. (Rotate 10°)	97.26%	98.31%	90.26%	72.22%	50.42%	48.23%	75.00%
Bit Acc. (Rotate 30°)	49.56%	51.30%	54.93%	50.09%	51.50%	49.65%	51.82%
Bit Acc. (Rotate 45°)	50.01%	50.18%	50.20%	46.88%	51.50%	50.96%	50.91%
Bit Acc. (Rotate 90°)	51.37%	81.07%	49.08%	49.57%	50.50%	49.42%	50.78%
Bit Acc. (Resize 32%)	99.75%	99.89%	97.92%	90.36%	98.24%	99.81%	100.00%
Bit Acc. (Resize 45%)	99.73%	99.90%	97.88%	91.06%	98.51%	99.81%	100.00%
Bit Acc. (Resize 55%)	99.73%	99.90%	97.88%	91.23%	98.60%	99.81%	100.00%
Bit Acc. (Resize 63%)	99.74%	99.90%	97.92%	91.67%	98.65%	99.81%	100.00%
Bit Acc. (Resize 71%)	99.73%	99.90%	97.96%	91.75%	98.69%	99.81%	100.00%
Bit Acc. (Resize 77%)	99.74%	99.89%	97.88%	91.75%	98.68%	99.81%	100.00%
Bit Acc. (Resize 84%)	99.74%	99.90%	97.92%	92.01%	98.68%	99.81%	100.00%
Bit Acc. (Resize 89%)	99.74%	99.90%	97.92%	91.84%	98.69%	99.81%	100.00%
Bit Acc. (Resize 95%)	99.74%	99.90%	97.92%	92.01%	98.71%	99.81%	100.00%
Bit Acc. (Crop 32%)	49.71%	50.24%	50.84%	48.70%	49.95%	49.65%	79.30%
Bit Acc. (Crop 45%)	65.22%	50.70%	51.52%	48.35%	50.59%	51.73%	94.14%
Bit Acc. (Crop 55%)	86.90%	52.73%	63.58%	57.03%	50.20%	51.58%	96.22%
Bit Acc. (Crop 63%)	93.42%	66.70%	79.13%	64.06%	49.77%	51.81%	97.79%
Bit Acc. (Crop 71%)	95.85%	87.34%	86.74%	69.44%	50.26%	56.42%	98.83%
Bit Acc. (Crop 77%)	97.13%	95.33%	91.83%	74.39%	50.57%	92.85%	99.35%
Bit Acc. (Crop 84%)	97.77%	98.31%	93.47%	79.86%	50.59%	99.88%	99.61%
Bit Acc. (Crop 89%)	98.81%	98.97%	94.83%	82.47%	50.38%	99.92%	99.22%
Bit Acc. (Crop 95%)	99.29%	99.56%	95.03%	82.90%	50.93%	99.92%	99.22%
Bit Acc. (Brightness 10%)	83.62%	83.17%	82.69%	51.13%	61.76%	80.04%	96.48%
Bit Acc. (Brightness 25%)	99.57%	98.93%	95.43%	56.25%	84.39%	97.19%	100.00%
Bit Acc. (Brightness 50%)	99.74%	99.76%	97.48%	75.52%	95.01%	99.54%	100.00%
Bit Acc. (Brightness 75%)	99.73%	99.84%	98.04%	86.81%	97.91%	99.62%	100.00%
Bit Acc. (Brightness 125%)	99.22%	98.91%	94.47%	94.36%	95.42%	97.00%	100.00%
Bit Acc. (Brightness 150%)	97.26%	96.18%	87.78%	93.23%	91.03%	92.42%	100.00%
Bit Acc. (Brightness 175%)	95.74%	94.48%	83.13%	93.06%	88.79%	90.35%	99.22%
Bit Acc. (Brightness 200%)	95.10%	92.85%	80.17%	92.88%	86.93%	88.31%	99.48%
Bit Acc. (Contrast 10%)	99.46%	97.88%	84.21%	51.65%	74.01%	74.96%	95.31%
Bit Acc. (Contrast 25%)	99.67%	99.76%	95.91%	56.25%	89.35%	95.19%	100.00%
Bit Acc. (Contrast 50%)	99.74%	99.82%	97.56%	75.95%	96.21%	99.19%	100.00%
Bit Acc. (Contrast 75%)	99.74%	99.85%	97.92%	86.89%	98.08%	99.54%	100.00%
Bit Acc. (Contrast 125%)	99.58%	99.59%	95.15%	94.70%	96.63%	98.65%	100.00%
Bit Acc. (Contrast 150%)	99.11%	98.96%	92.35%	96.09%	93.69%	95.23%	100.00%
Bit Acc. (Contrast 175%)	98.52%	97.49%	89.58%	96.09%	91.50%	92.15%	99.87%
Bit Acc. (Contrast 200%)	97.86%	95.88%	86.98%	96.61%	89.21%	90.08%	99.74%
Bit Acc. (Hue -0.2)	98.37%	99.40%	83.25%	60.68%	95.39%	97.31%	95.18%
Bit Acc. (Hue -0.1)	99.66%	99.56%	94.63%	73.09%	97.28%	98.92%	99.87%
Bit Acc. (Hue 0.1)	99.68%	99.06%	95.47%	80.82%	97.12%	98.54%	100.00%
Bit Acc. (Hue 0.2)	99.28%	99.04%	81.57%	59.46%	95.70%	97.50%	98.70%
Bit Acc. (JPEG 40)	97.18%	99.41%	94.91%	91.32%	98.35%	99.50%	100.00%
Bit Acc. (JPEG 50)	98.35%	99.64%	96.47%	91.58%	98.84%	99.62%	100.00%
Bit Acc. (JPEG 60)	98.62%	99.76%	96.15%	91.49%	98.54%	99.62%	100.00%
Bit Acc. (JPEG 70)	98.90%	99.74%	96.39%	91.49%	98.42%	99.62%	100.00%
Bit Acc. (JPEG 80)	99.31%	99.84%	97.40%	91.84%	98.60%	99.69%	100.00%
Bit Acc. (JPEG 90)	99.60%	99.85%	97.32%	92.10%	98.66%	99.69%	100.00%
Bit Acc. (Gaussian Blur 3)	99.74%	99.90%	97.92%	91.67%	98.66%	99.81%	100.00%
Bit Acc. (Gaussian Blur 5)	99.74%	99.90%	97.96%	90.97%	98.47%	99.81%	100.00%
Bit Acc. (Gaussian Blur 9)	99.74%	99.89%	97.88%	89.24%	98.15%	99.81%	100.00%
Bit Acc. (Gaussian Blur 13)	99.73%	99.85%	97.96%	87.15%	97.67%	99.77%	100.00%
Bit Acc. (Gaussian Blur 17)	99.70%	99.79%	98.00%	84.98%	96.83%	99.73%	100.00%

Table 5 Extended results of Chunky Seal on COCO (Lin et al., 2014).

	Chunky Seal (ours) 1024bit, 256px	Video Seal 256bit, 256px	Video Seal 96bit, 256px	HiDDeN	MBRS	TrustMark	WAM
Capacity	1024 bits	256 bits	96 bits	48 bits	256 bits	100 bits	32 bits
PSNR	44.29 dB	44.94 dB	53.33 dB	30.51 dB	45.81 dB	42.72 dB	38.73 dB
SSIM	0.9917	0.9953	0.9992	0.8469	0.9944	0.9921	0.9803
MS-SSIM	0.9968	0.9975	0.9988	0.9203	0.9976	0.9931	0.9891
LPIPS	0.0061	0.0022	0.0033	0.1850	0.0035	0.0015	0.0295
Embedding Time	0.03 s	0.01 s	0.01 s	0.01 s	0.01 s	0.01 s	0.03 s
Extraction Time	0.04 s	0.01 s	0.01 s	0.00 s	0.00 s	0.00 s	0.01 s
Bit Acc.	99.66%	99.92%	97.64%	92.40%	98.70%	99.90%	100.00%
Bit Acc. (Horizontal Flip)	99.52%	99.87%	97.16%	61.83%	49.87%	99.87%	99.97%
Bit Acc. (Rotate 5°)	97.11%	99.06%	94.69%	79.85%	50.04%	65.31%	97.84%
Bit Acc. (Rotate 10°)	94.46%	97.56%	91.53%	72.65%	49.92%	51.89%	77.16%
Bit Acc. (Rotate 30°)	50.57%	50.83%	57.08%	53.37%	49.64%	50.05%	51.00%
Bit Acc. (Rotate 45°)	49.98%	50.53%	50.55%	49.54%	50.15%	50.93%	50.75%
Bit Acc. (Rotate 90°)	56.36%	83.44%	50.58%	51.23%	49.90%	49.84%	49.28%
Bit Acc. (Resize 32%)	94.69%	97.69%	97.53%	71.19%	90.57%	99.81%	99.81%
Bit Acc. (Resize 45%)	98.60%	99.56%	97.70%	80.50%	96.16%	99.86%	100.00%
Bit Acc. (Resize 55%)	99.31%	99.79%	97.64%	84.88%	97.28%	99.87%	100.00%
Bit Acc. (Resize 63%)	99.53%	99.86%	97.66%	87.17%	97.80%	99.90%	100.00%
Bit Acc. (Resize 71%)	99.63%	99.89%	97.67%	87.92%	98.11%	99.88%	100.00%
Bit Acc. (Resize 77%)	99.66%	99.91%	97.69%	88.71%	98.21%	99.90%	100.00%
Bit Acc. (Resize 84%)	99.66%	99.91%	97.68%	89.23%	98.34%	99.90%	100.00%
Bit Acc. (Resize 89%)	99.67%	99.93%	97.67%	89.38%	98.34%	99.91%	100.00%
Bit Acc. (Resize 95%)	99.66%	99.92%	97.70%	89.75%	98.43%	99.89%	100.00%
Bit Acc. (Crop 32%)	49.84%	50.25%	50.75%	49.27%	49.54%	49.95%	73.88%
Bit Acc. (Crop 45%)	60.64%	49.73%	50.36%	50.40%	50.04%	50.79%	91.47%
Bit Acc. (Crop 55%)	82.26%	50.59%	59.70%	56.02%	49.99%	49.79%	95.72%
Bit Acc. (Crop 63%)	89.82%	58.23%	74.25%	61.19%	50.64%	51.19%	97.19%
Bit Acc. (Crop 71%)	93.68%	81.43%	85.59%	67.94%	49.80%	54.77%	97.22%
Bit Acc. (Crop 77%)	94.71%	92.29%	89.15%	73.06%	49.70%	87.79%	98.19%
Bit Acc. (Crop 84%)	96.04%	97.64%	92.94%	78.83%	49.66%	99.95%	99.12%
Bit Acc. (Crop 89%)	97.17%	98.96%	94.32%	81.27%	49.84%	99.98%	98.69%
Bit Acc. (Crop 95%)	97.88%	99.48%	95.21%	83.31%	50.82%	99.96%	99.22%
Bit Acc. (Brightness 10%)	85.06%	81.38%	81.43%	52.33%	62.51%	83.43%	96.12%
Bit Acc. (Brightness 25%)	98.76%	98.82%	94.93%	55.85%	84.75%	97.85%	99.75%
Bit Acc. (Brightness 50%)	99.53%	99.84%	97.06%	74.29%	95.43%	99.74%	100.00%
Bit Acc. (Brightness 75%)	99.65%	99.92%	97.50%	86.44%	97.98%	99.87%	100.00%
Bit Acc. (Brightness 125%)	99.16%	99.02%	96.24%	95.04%	95.24%	98.56%	100.00%
Bit Acc. (Brightness 150%)	98.47%	97.78%	94.73%	95.56%	92.20%	96.45%	100.00%
Bit Acc. (Brightness 175%)	97.34%	96.13%	92.64%	95.63%	89.16%	94.62%	100.00%
Bit Acc. (Brightness 200%)	95.83%	94.18%	89.48%	94.92%	86.91%	91.97%	99.97%
Bit Acc. (Contrast 10%)	97.77%	98.29%	83.32%	52.31%	77.41%	78.54%	93.69%
Bit Acc. (Contrast 25%)	99.43%	99.68%	95.15%	55.79%	90.76%	96.47%	99.78%
Bit Acc. (Contrast 50%)	99.62%	99.87%	97.11%	73.81%	96.71%	99.68%	100.00%
Bit Acc. (Contrast 75%)	99.66%	99.92%	97.50%	86.27%	98.19%	99.87%	100.00%
Bit Acc. (Contrast 125%)	99.23%	99.19%	95.83%	94.67%	95.67%	98.57%	100.00%
Bit Acc. (Contrast 150%)	98.55%	97.84%	93.30%	96.00%	92.83%	95.83%	99.97%
Bit Acc. (Contrast 175%)	97.75%	96.42%	91.00%	96.58%	90.38%	93.65%	99.97%
Bit Acc. (Contrast 200%)	96.89%	95.07%	89.08%	96.85%	88.58%	90.87%	99.88%
Bit Acc. (Hue -0.2)	97.37%	98.35%	82.19%	59.62%	95.41%	99.03%	96.91%
Bit Acc. (Hue -0.1)	99.41%	98.77%	94.92%	72.27%	97.34%	99.73%	100.00%
Bit Acc. (Hue 0.1)	99.37%	99.05%	95.53%	82.04%	97.63%	99.76%	99.97%
Bit Acc. (Hue 0.2)	97.70%	98.64%	78.98%	61.85%	96.64%	99.54%	98.06%
Bit Acc. (JPEG 40)	65.86%	97.79%	72.64%	87.98%	95.44%	98.34%	97.28%
Bit Acc. (JPEG 50)	72.47%	98.74%	80.22%	88.21%	96.22%	99.05%	98.31%
Bit Acc. (JPEG 60)	76.97%	99.26%	85.39%	88.44%	97.28%	99.38%	98.84%
Bit Acc. (JPEG 70)	82.93%	99.55%	89.42%	88.77%	97.50%	99.64%	99.53%
Bit Acc. (JPEG 80)	88.89%	99.79%	93.34%	89.31%	97.99%	99.73%	99.56%
Bit Acc. (JPEG 90)	93.21%	99.87%	96.06%	90.50%	98.25%	99.81%	99.84%
Bit Acc. (Gaussian Blur 3)	99.63%	99.89%	97.70%	86.23%	97.75%	99.87%	100.00%
Bit Acc. (Gaussian Blur 5)	99.29%	99.86%	97.75%	80.40%	96.09%	99.86%	100.00%
Bit Acc. (Gaussian Blur 9)	97.95%	99.74%	97.55%	71.00%	90.90%	99.86%	99.88%
Bit Acc. (Gaussian Blur 13)	93.88%	99.57%	97.12%	65.04%	84.27%	99.85%	99.62%
Bit Acc. (Gaussian Blur 17)	84.65%	99.11%	96.64%	60.94%	77.49%	99.41%	99.38%

I.2 Image examples

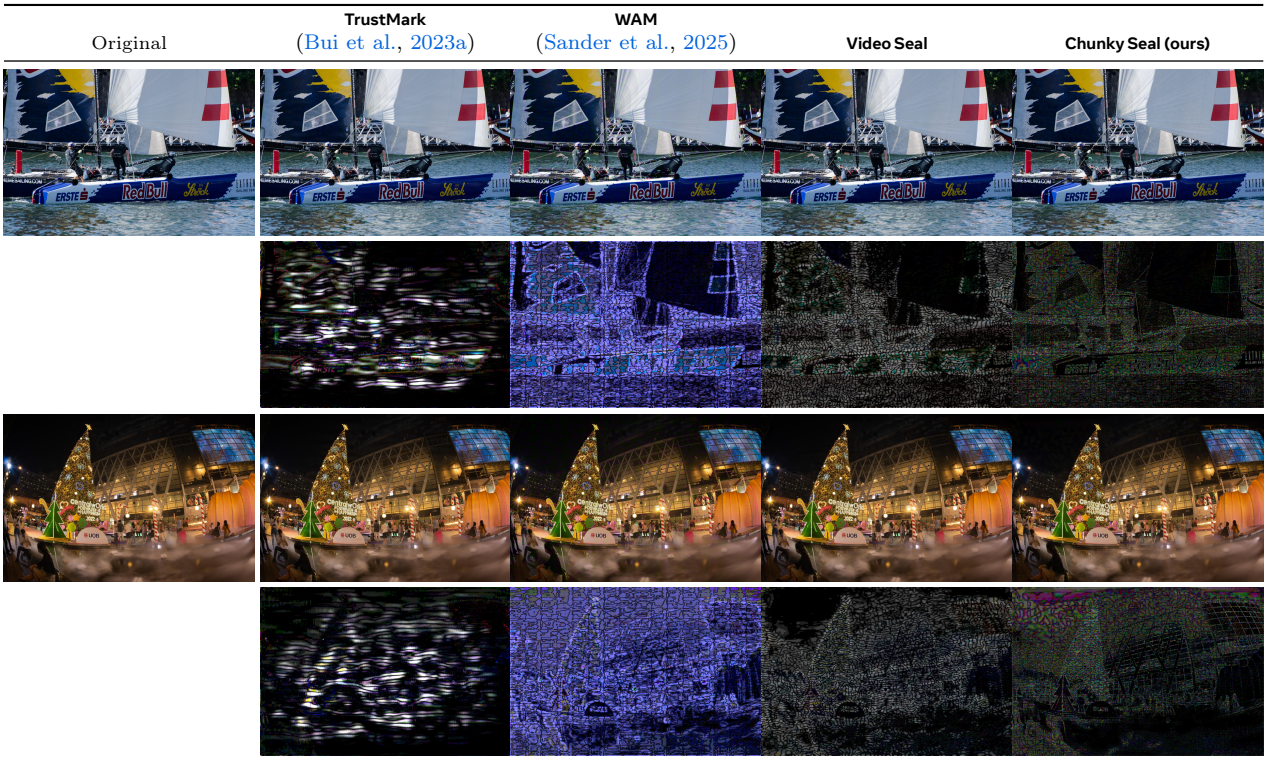

Figure 13 Qualitative results for the different watermarking methods on images taken from the SA-1b dataset at their original resolutions. We show the original images, the watermarked ones, and the watermark distortions brightened for clarity.

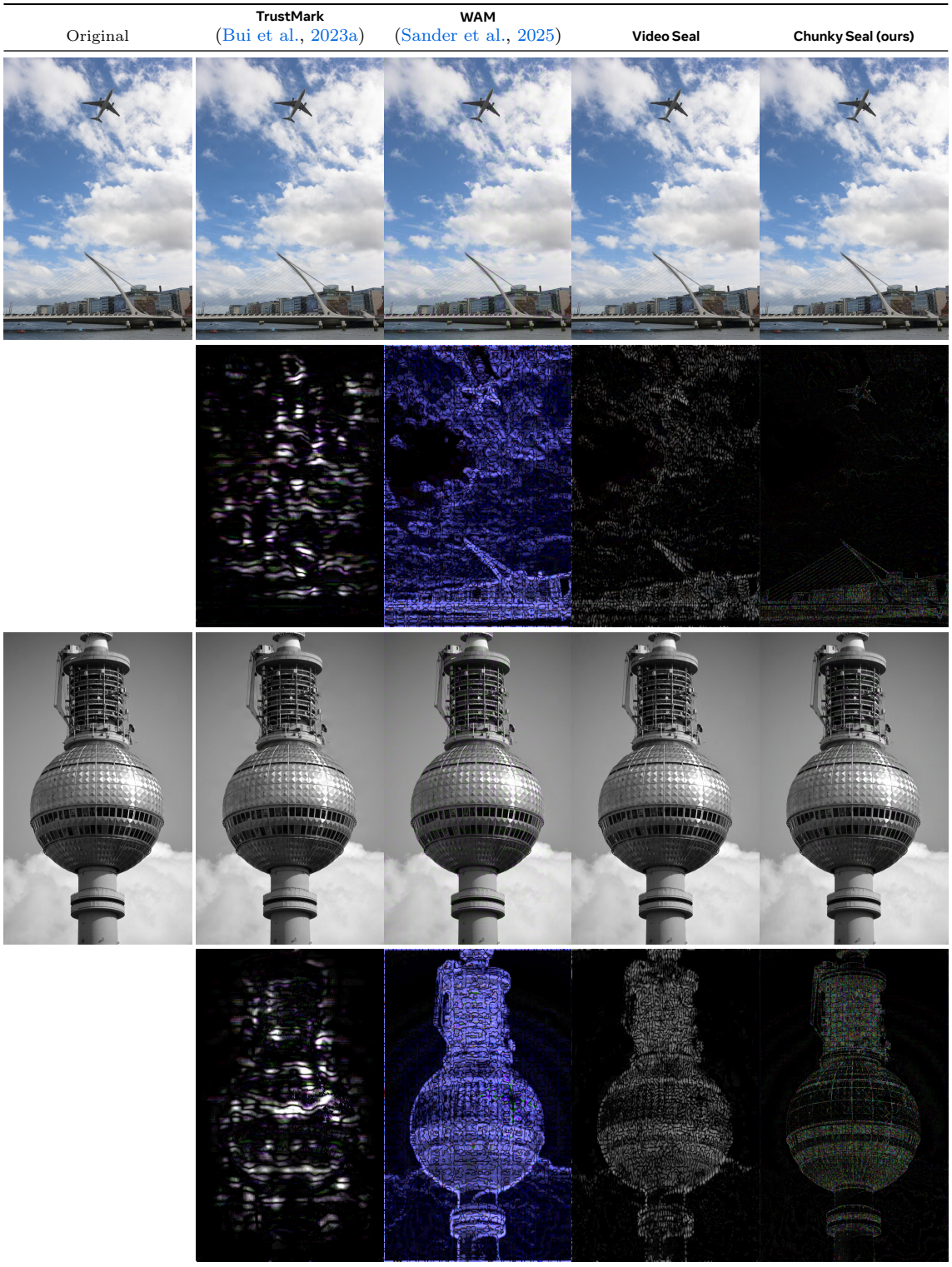

Figure 14 Qualitative results for the different watermarking methods on images taken from the SA-1b dataset at their original resolutions. We show the original images, the watermarked ones, and the watermark distortions brightened for clarity.