FastStamp: Accelerating Neural Steganography and Digital Watermarking of Images on FPGAs

Shehzeen Hussain[1]*, Nojan Sheybani[1]*, Paarth Neekhara[2]*, Xinqiao Zhang[1], Javier Duarte[3], Farinaz Koushanfar[1]
*Equal contribution
[1]Department of Electrical and Computer Engineering, UC San Diego, [2]Department of Computer Science and Engineering, UC San Diego, [3]Department of Physics, UC San Diego
{ssh028,nsheyban,pneekhar,x5zhang,jduarte,fkoushanfar}@ucsd.edu

ABSTRACT

Steganography and digital watermarking are the tasks of hiding recoverable data in image pixels. Deep neural network (DNN) based image steganography and watermarking techniques are quickly replacing traditional hand-engineered pipelines. DNN based watermarking techniques have drastically improved the message capacity, imperceptibility and robustness of the embedded watermarks. However, this improvement comes at the cost of increased computational overhead of the watermark encoder neural network. In this work, we design the first accelerator platform FastStamp to perform DNN based steganography and digital watermarking of images on hardware. We first propose a parameter efficient DNN model for embedding recoverable bit-strings in image pixels. Our proposed model can match the success metrics of prior state-of-the-art DNN based watermarking methods while being significantly faster and lighter in terms of memory footprint. We then design an FPGA based accelerator framework to further improve the model throughput and power consumption by leveraging data parallelism and customized computation paths. FastStamp allows embedding hardware signatures into images to establish media authenticity and ownership of digital media. Our best design achieves 68× faster inference as compared to GPU implementations of prior DNN based watermark encoder while consuming less power.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Hardware** → **Hardware accelerators**; • **Security and privacy** → **Authentication**; **Tamper-proof and tamper-resistant designs**.

KEYWORDS

Digital watermarking, steganography, FPGA, deep learning

ACM Reference Format:
Shehzeen Hussain[1]*, Nojan Sheybani[1]*, Paarth Neekhara[2]*, Xinqiao Zhang[1], Javier Duarte[3], Farinaz Koushanfar[1]. 2022. FastStamp: Accelerating Neural Steganography and Digital Watermarking of Images on FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22), October 30-November 3, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3508352.3549357

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10.
https://doi.org/10.1145/3508352.3549357

1 INTRODUCTION

Steganography and watermarking techniques aim to embed digital information into visual media in an invisible manner. A watermarking scheme typically consists of two components—an *encoder* that embeds a given digital message in the image pixels and a *decoder* that extracts the message from a watermarked image. Such technology has several applications, such as transmitting secret messages, copyright protection, establishing media ownership, and embedding invisible QR codes into images [41]. Another important use case of digital watermarking lies in media authentication or integrity verification by embedding semi-fragile watermarks into digital media at the source. Semi-fragile watermarks require the property of being fragile to malicious manipulations or tampering while being robust to benign image-processing operations such as image compression, scaling, and color adjustments.

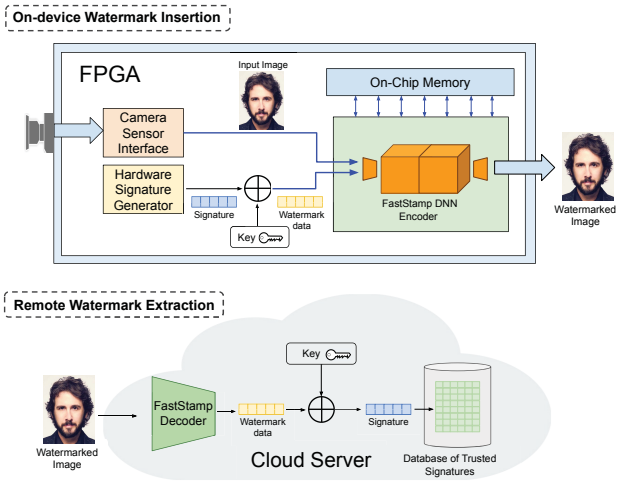

Figure 1: Schematic diagram of FastStamp watermarking pipeline. The pipeline is divided into two steps: watermark insertion using a DNN encoder on FPGA (top) and watermark extraction using a DNN decoder on a cloud server (bottom).

Conventional image watermarking systems use hand engineered pipelines [18, 27, 28, 45] to embed information in the spatial or frequency domain of an image. However, the major limitations of

traditional approaches lie in the higher visibility of the embedded watermarks, limited message capacity, and low robustness to digital compression techniques like JPEG transforms. As a result, these traditional approaches are being quickly replaced with deep learning based techniques, which are achieving state-of-the-art results in image watermarking and steganography tasks.

Deep learning based watermarking techniques relies on encoder and decoder convolutional neural networks (CNNs). These networks are trained end-to-end for the task of embedding and retrieving a given message in an image. While deep learning techniques significantly outperform hand engineered watermarking pipelines, the improvement comes at the cost of increased computational overhead and memory requirement of these models. The best performing neural image watermarking encoders are parameterized by around half a million floating point parameters, which makes it challenging to deploy such systems on resource constrained hardware such as FPGAs or handheld devices. Such techniques have only been implemented at a software level which results in significant latency between image capture and transmission. Our work aims to embed watermarks in images and videos in real-time as they are being captured. Embedding the watermarks at the hardware level can not only reduce the latency of the watermarking process but also enable media authentication and provenance by leveraging unique hardware signatures from PUFs [14] or secure enclaves as the watermarking data.

In this work, we propose FastStamp—a light-weight yet robust neural image watermarking framework to enable real-time watermarking on hardware platforms. Keeping resource constraints in mind, we develop a parameter efficient CNN based watermarking model that can match and even outperform the success metrics of state-of-the-art neural image watermarking models. Our watermarking model leverages efficient neural blocks such as depthwise separable convolutions, spatial upsampling operations, and linear layers to reduce the memory requirement and inference latency without compromising the watermark retrieval accuracy, message capacity, and imperceptibility. We then design and verify an FPGA implementation of our watermarking encoder model to allow image watermarking directly on hardware. Our most optimized design achieves real-time image watermarking and only requires 3 milliseconds to watermark a given image while achieving the same results as the software implementations. To the best of our knowledge, we propose the first framework to accelerate deep neural networks for steganography and watermarking of images using FPGAs. FastStamp can be customized for semi-fragile and robust image watermarking. While semi-fragile watermarking is useful for checking media integrity, robust image watermarking is useful for establishing media ownership and copyright protection.

Summary of Contributions:

- We develop a parameter efficient and computationally inexpensive watermarking model that can embed recoverable watermarks at a significantly lower cost as compared to prior neural watermarking techniques.
- We develop the first FPGA accelerator platform for DNN based image watermarking and steganography. We design reusable and reconfigurable basic blocks pertinent to such models as separable convolutions, 2D upsampling, and skip connections. We deploy the FastStamp encoder model on Xilinx XCVU13P FPGA, which achieves 68× faster inference than prior neural watermarking models.
- Our framework is end-to-end and supports two types of watermarking schemes: *robust* and *semi-fragile*. FastStamp learns to be robust to a wide range of real-world digital image processing operations such as lighting, color adjustments, and compression techniques. Our semi-fragile watermarking scheme learns to be robust to above benign transformations while being fragile to media forgery and local tampering.

2 BACKGROUND

2.1 Digital Watermarking

Digital watermarking techniques broadly seek to generate three different types of watermarks: fragile [9], robust ,[8, 39, 52] and semi-fragile [18, 28, 47]. Fragile and semi-fragile watermarks are primarily used to certify the integrity and authenticity of image data. Fragile watermarks are used to achieve accurate authentication of digital media, where even a one-bit change to an image will lead it to fail the certification system. In contrast, *robust* watermarks aim to be recoverable under several image manipulations to allow media producers to assert ownership over their content even if the video is redistributed and modified. Semi-fragile watermarks combine the advantages of both robust and fragile watermarks and are mainly used for fuzzy authentication of digital images and identification of image tampering [47].

Prior work has proposed hand-engineered pipelines to embed semi-fragile watermark information in the spatial[44] and frequency domain of images and videos. Particularly, in the frequency domain, the watermark can be embedded by modifying the coefficients produced with transformations such as the discrete cosine transform (DCT) [18, 36] and discrete wavelet transform (DWT) [5, 27, 42]. For example, during 2D DWT an image is first decomposed into various frequency channels using a Haar filter. A scaled image is used as the watermark and inserted into mid frequency wavelet channel. Taking 2D inverse DWT of the altered wavelet decomposition produces the watermark embedded image. The major drawbacks of traditional approaches lie in higher visibility of the embedded watermarks, increased distortions in generated images, and low robustness to compression techniques like JPEG transforms as discussed in Section 5.2. Specifically, since digital images shared over social media are highly compressed and undergo various lighting and color adjustments, the watermarks generated by DWT and DCT algorithms break under benign real-world transformations and compression.

More recently, CNNs have been used to provide an end-to-end solution to the watermarking problem. They replace hand-crafted hiding procedures with neural network encoding [4, 16, 32, 34, 41, 43, 49, 52]. These techniques train end-to-end encoder CNNs to embed and decode watermarks, which have resulted in lower imperceptibility and more robust recovery of the watermark data. However, these models are memory intensive and much slower as compared to DCT based algorithms. In our work, we address the performance limitations of neural watermarking systems and propose a light-weight framework for both robust and semi-fragile image watermarking suitable for hardware acceleration platforms.

2.2 FPGA Accelerated Techniques

There have been several efforts to accelerate neural networks on FPGAs [20, 38, 40, 48]. Equipped with the necessary hardware for basic DNN operations, FPGAs are able to achieve high parallelism and utilize the properties of neural network computation to remove unnecessary logic. Some prior efforts have been made in FPGA acceleration of convolutional autoencoder architectures [13, 30, 51]. While these works implement many sub-blocks used in neural network computation, we find that they cannot be directly used for an image watermarking framework because efficient implementations of sub-blocks like 2D upsampling and separable convolutions with skip-connections are absent. Moreover, existing neural architectures for image watermarking are not optimized for hardware-software co-design.

Prior work [15, 17, 24, 25] has made significant efforts in accelerating traditional image watermarking schemes that hide secret information in the frequency domain of images and rely on DCT and DWT based algorithms. As discussed earlier, while such algorithms are vastly popular for hardware applications due to its simplicity and low computational overhead, the resulting watermarked image is often not robust to real-world image transformations. In our work, we develop the first FPGA accelerator platform for DNN based image watermarking and steganography that enables robustness to real-world digital image processing and compression while being selectively fragile to media tampering techniques.

2.3 Countering Media Forgery

With the widespread development of deep learning based image and video synthesis techniques [6, 26, 29, 33], it has become increasingly easier and faster to generate high-quality convincing fake images/videos such as Deepfakes. Such manipulated media can fuel misinformation, defame individuals and reduce trust in media. While considerable research effort has been made in designing CNN based deepfake detectors [2, 10], these techniques have been shown to hold major security vulnerabilities and can be bypassed by attackers [21]. To counter such threats, authors [34, 37, 43] have proposed semi-fragile watermarking as a solution to perform media authentication and distinguish deepfake media from real media by verifying a secret watermark embedded in the media. For both fragile and semi-fragile techniques, a watermark must be inserted when the image is captured, which makes these techniques dependent on both algorithmic and hardware implementation. If watermark information is embedded separately in images and videos after it is captured by a device, this method may fail in situations where tampering is carried out before inserting the signature or watermark. While the proposed solution to media authentication is to add a verifiable digital signature or watermark to an image/video using neural networks, prior works [34, 37, 43] do not actively implement the technology in resource constrained settings or camera hardware. Upon empirical study, we found that it is challenging to fit such off-the-shelf models [34, 41, 52] on FPGAs since they were developed without any attention to hardware-software co-design practices and range from 500 k to 2 million parameters for the encoder model. To this end, we design our own DNN based watermarking system that utilizes depthwise separable convolutions and a parameter efficient message upsampler to reduce computational overhead while preserving required bit recovery accuracy, capacity, and imperceptibility.

3 METHODOLOGY

3.1 Training Framework

Our goal is to develop a learnable, end-to-end model for image steganography and watermarking such that the encoder model can embed a message as a visually invisible perturbation in the image, and the decoder network can extract the message from the watermarked image. We develop two variants of our training framework to generate *robust* and *semi-fragile* watermarks. A robust watermark is designed to be recoverable when real-world image transformations are applied. A semi-fragile watermark is designed to be robust to benign image transformations such as compression, minor color, and contrast adjustments but it should be unrecoverable when malicious image transforms such as image tampering and face-swapping are applied. A semi-fragile watermark is designed to be robust to benign image transformations such as compression and minor color, and contrast adjustments but it should be unrecoverable when malicious image transforms such as image tampering and face-swapping are applied.

The encoder network E takes as input an image x and a bit string $s \in \{0,1\}^L$ of length L, and produces an encoded (watermarked) image x_w. That is, $x_w = E(x, s)$. Depending on our task of either *semi-fragile* or *robust* watermarking, the encoded image goes through the following operations:

(1) ***Robust watermarking:*** In this setting, the image goes through a benign image transformation $g_b \sim G_b$ to produce $x_b = g_b(x_w)$. The benign image is then fed to the decoder network, which predicts the message $s_b = D(x_b)$. For optimizing secret retrieval during training, we use the L_1 distortion between the predicted and ground-truth bit strings. The decoder is encouraged to be robust to benign transformations by minimizing the message distortion $L_1(s, s_b)$. Therefore the secret retrieval error for an image $L_M(x)$ is obtained as follows:

$$L_M(x) = L_1(s, s_b) \tag{1}$$

(2) ***Semi-fragile watermarking:*** In this setting, the watermarked image goes through two image transformation functions—one sampled from a set of benign transformations ($g_b \sim G_b$) and the other sampled from a set of malicious transformations ($g_m \sim G_m$) to produce a benign image $x_b = g_b(x_w)$ and a malicious image $x_m = g_m(x_w)$. The benign and malicious watermarked images are then fed to the decoder network, which predicts the messages $s_b = D(x_b)$ and $s_m = D(x_m)$ respectively. The decoder is encouraged to be robust to benign transformations by minimizing the message distortion $L_1(s, s_b)$; and fragile for malicious manipulations by maximizing the error $L_1(s, s_m)$. Therefore the secret retrieval error for an image $L_M(x)$ is obtained as follows:

$$L_M(x) = L_1(s, s_b) - L_1(s, s_m) \tag{2}$$

The watermarked image is encouraged to look visually similar to the original image by optimizing three image distortion metrics:

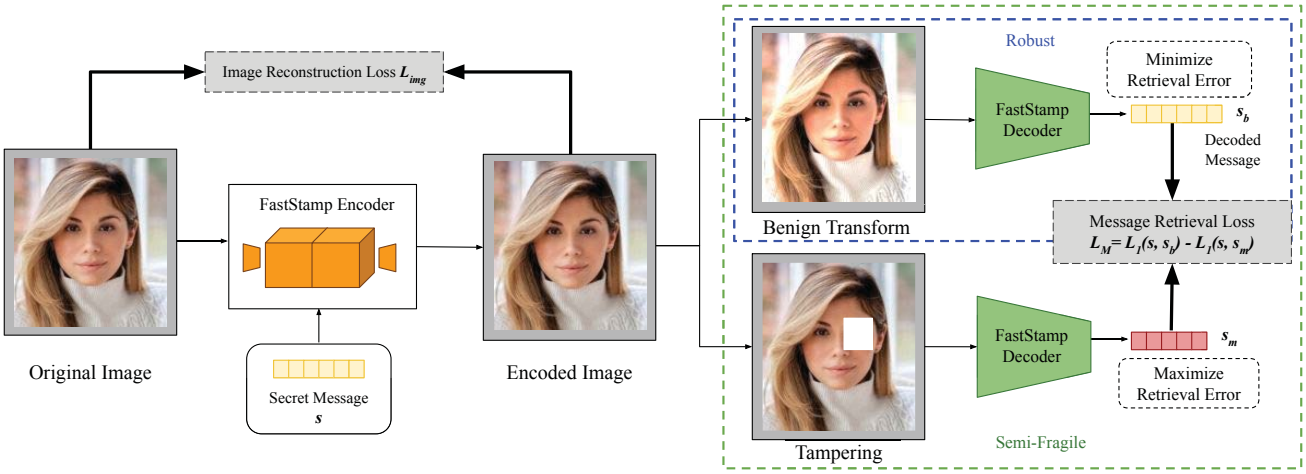

Figure 2: FastStamp Encoder-Decoder Training framework. In both robust and semi-fragile schemes, the encoder model learns to embed a message in a given image by encouraging retrieval from the decoder model under benign transforms. For our semi-fragile design, we additionally encourage fragility to malicious transforms by maximizing message retrieval error on tampered watermark images.

L_1, L_2 and L_{pips} [50] distortions.

$$L_{\text{img}}(x, x_w) = L_1(x, x_w) + L_2(x, x_w) + c_p L_{\text{pips}}(x, x_w) \quad (3)$$

Therefore, the parameters α, β of the encoder and decoder network are trained using mini-batch gradient descent to optimize the following loss over a distribution of input messages and images:

$$\mathbb{E}_{x,s,g_b,g_m}[L_{\text{img}}(x, x_w) + c_M L_M(x)] \quad (4)$$

In the above equations, c_p, and c_M are scalar coefficients for the respective loss terms. We refer the readers to our supplementary material for the values we use for these coefficients and other implementation details.

3.2 Message encoding

The input of our encoder network is a bit string s of length L. This watermarking data includes a secret message or a hardware signature generated by trusted execution environments or PUFs. To further ensure message secrecy, we can encrypt the message using a stream cipher with a secret key that is shared between the encoding and decoding devices. In our work we embed messages of length 128 bits in an image of size 128×128, which allows embedding 2^{128} unique messages in each 128×128 image patch.

3.3 Model Architecture and Optimization

The encoder model takes as input an image x and a message bit-string s to produce a watermarked image x_w. The encoder model of a typical neural watermarking system follows a convolutional U-Net architecture comprising several downsampling and upsampling layers with skip-connections. In prior work [34, 41], the secret message bit-string is first projected using a learnable linear layer reshaped as a matrix to have the same height and width as the input image; and then attached as the fourth channel of the input image. The combined input and secret image then undergo the downsampling and upsampling operations of the U-Net to produce the watermarked image.

The above described encoder model architecture in prior work [34, 41, 52] has a large memory footprint and is unsuitable for deployment in resource-constrained settings. To reduce the model size without compromising on the watermarking performance, we propose the following architectural optimizations:

3.3.1 ***Secret Message Upsampler.*** The secret message upsampler projects the message string s to a matrix that gets attached as the fourth channel of the input image. A naïve implementation using a linear layer can result in a large memory footprint since the number of parameters is given by hwL, where h and w are the input image height and width, respectively, and L is the secret message length. An input image size of 128×128 and a message length of 128, would result in more than two million parameters. To optimize the number of parameters, we perform the secret upsampling operation as follows:

(1) Project the message s to a vector s_{proj} of size $h'w'$ using a linear layer.
(2) Reshape s_{proj} to a matrix $s_{\text{proj}M}$ of dimensions (h', w')
(3) Upsample $s_{\text{proj}M}$ to a matrix of size (h, w) using nearest-neighbour upsampling.

The nearest-neighbour upsampling operation is parameter-free and computationally more efficient as compared to matrix-vector multiplication. Through our experiments, we find that using an (h', w') that are much smaller than (h, w) can achieve the same watermarking while being significantly efficient in both time and memory. For an image of size $(128, 128)$ and message length $L = 128$, we use $h' = w' = 16$ thus requiring only 32768 parameters. The upsampled secret gets attached as the fourth channel of the input image and undergoes the U-Net downsampling and upsampling operations described below.

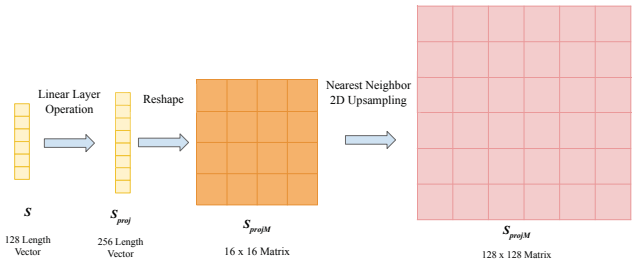

Figure 3: An example of optimized secret message upsampling using linear layer projection followed by nearest neighbor 2D upsampling.

3.3.2 ***U-Net Downsampling.*** The downsampling network in U-Net architecture typically comprises 5 to 8 convolutional blocks. Each block contains a strided convolutional layer, a batch normalization layer, and a non-linear activation like ReLU or leaky ReLU. The number of output channels of each convolutional layer increase with the depth of the network, doubling at each step until a maximum value is reached. To optimize this architecture, we first replace the convolutional layers with depthwise and point-wise separable convolutional layers [7]. Not only does this optimization reduce the number of parameters but also reduces the number of floating point operations required in each layer computation. Next, we optimize a number of output channels of each convolutional block. In our experiments, we perform a design-space exploration to find that we can substantially reduce the number of output channels in each layer without compromising on secret retrieval accuracy and imperceptibility. Our most optimized design uses 5 downsampling layers with 64 output channels in the final separable convolutional layer. Figure 4 details the network architecture and output tensor sizes after each downsampling step.

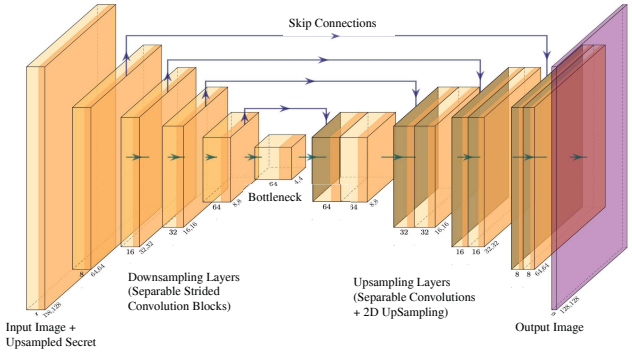

Figure 4: FastStamp Encoder Architecture. Our encoder network takes as input an image x and the output of the secret message upsampler $s_{\text{proj}M}$ and generates the watermarked image

3.3.3 ***U-Net Upsampling.*** The upsampling network in U-Net architecture follows a mirror image of the downsampling network. Instead of a regular convolutional layer, each upsampling block typically contains a transposed convolutional layer. However, transposed convolutional layers have been shown to introduce unwanted visual artifacts in the generated images [35] and we see the same effect empirically in our work. To remove such artifacts, we replace the transposed convolution layer with a separable convolution layer followed up by nearest neighbor 2D upsampling following the recommendations given by past work [35]. At each upsampling step, the output of the corresponding downsampling step is concatenated with the block input to provide skip-connections which are known to improve the performance of encoder-decoder models. The output of the last upsampling layer undergoes a tanh activation function to normalize output values between -1 and 1, which are then scaled between 0 and 1 to produce an RGB image. Figure 4 details our encoder network architecture.

FastStamp's Decoder follows a similar architecture as the encoder but contains 8 downsampling and 8 upsampling layers. After the upsampling U-net, the decoder network follows the inverse architecture of the secret message upsampler to predict the 128-bit message.

4 ACCELERATOR DESIGN

4.1 Design Overview

Figure 5 gives a high-level overview of our FPGA accelerator design. Each neural network layer is treated as a separate dataflow stage. To be low latency and high throughput, all weights and biases are stored on-chip. Complex activation functions are implemented via precomputed lookup tables. The design uses task-level pipelining (i.e., HLS dataflow) for each layer and streams the data between each dataflow stage using first-in-first-out buffers (FIFOs). As FIFOs can only be read once, to implement the skip connections, an additional dataflow stage is used to clone the skip connection data from its input FIFO into two other FIFOs so that it can be read twice for its two datapaths.

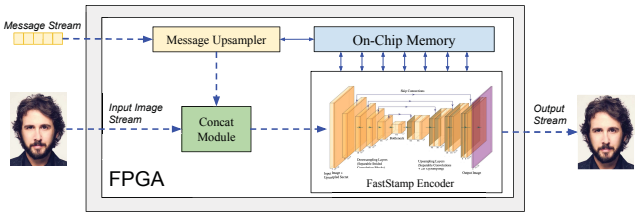

Figure 5: Design overview of FastStamp Accelerator Platform.

4.2 Implementation Details

To implement FastStamp on FPGA, we began with using the standard *hls4ml* [1, 11, 12] framework and added additional libraries to this to support the necessary functionalities such as depthwise separable convolutions, and nearest-neighbor 2D upsampling, concatenation layers (i.e., skip connections) which were previously not supported. We create custom sub-blocks and modify existing implementations with optimizations that better support our model. Our design is composed of the following 6 main architectural sub-blocks:

4.2.1 ***Linear Layer.*** The linear layer in the *Secret Message Upsampler* module is implemented using efficient matrix-vector multiplication based on product and sum tree. In order to optimize the design and make efficient use of the DSP blocks, we use a parallelized approach to convert layer computations into multiple MAC operations. Multiple rows of the weight matrix are processed simultaneously by dividing it into chunks. In each round, a chunk of the weights matrix is copied to one of the weight buffers while the other weight buffer is fed into the *dot product* modules together with a copy of the input vector. The iterations end when all rows of the weight matrix have been processed. Then each *dot-product* function partitions its input vectors into chunks and concurrently executes MAC operations over the partitioned subsets. The accumulated results of the subsets are then added together within the *reduce_sum* function to compute the final output. The *reduce_sum* module performs a tree-based reduction algorithm which further reduces latency.

4.2.2 ***Nearest-neighbor 2D Upsampling.*** 2D upsampling layers are used in the secret *Secret Message Upsampler* and the *U-Net Upsampling layers*. To upsample a matrix, we iterate through every element in the matrix and start building the new resized matrix. We use nearest neighbour as our interpolation method, as it is the least hardware intensive computation available for resizing purposes. We do not utilize pipelining in this sub-block, as it would cause a drastic increase in resource utilization for this resizing task while not having a substantial effect on latency. Due to these problems, pipelining makes this sub-block unscalable. Instead, loop unrolling is utilized to reduce loop iterations, giving us latency reductions with negligible impact on resource utilization.

4.2.3 ***Skip Connections.*** Concatenate layers are used in FastStamp to implement skip connections between upsampling and downsampling layers. These layers contributed towards some of the highest resource utilization in our design. To realize FastStamp on hardware, efficient support for concatenation of different sized inputs was implemented. We also distribute resource utilization to BRAM within these layers due to a large allocation of LUTs. Alongside this, we are able to minimize latency by pipelining the operations with complete unrolling in these layers.

4.2.4 ***Separable Convolution.*** Due to resource limitations, traditional convolution techniques in machine learning settings are often not feasible on hardware. Recent works [3, 46] have shown that utilizing separable convolutions significantly reduces the number of multiplications needed. Instead of traditional convolution, separable convolution layers are used in both U-Net downsampling and upsampling layers. A separable convolution comprises of a depthwise separable convolution and point-wise convolution operation. The depthwise separable convolution is performed independently across the input channels and results in the same number of output channels. A point-wise convolution operation reduces to matrix-vector multiplication along the channel axis for each spatial cell of the input. We also use a streaming implementation of depthwise convolution utilizing pipelining with complete loop unrolling, array partitioning, and loop flattening to achieve minimal latency. We optimize standard point-wise convolution in a similar manner as our linear layer.

4.2.5 ***Batch-normalization.*** Batch-normalization [22] layers are used in U-Net downsampling and upsampling layers. These layers are used after the separable convolution layers to stabilize the network. Rather than using this optimization, pipelining, loop unrolling, and array partitioning are enforced to accelerate the normalization layer. This allows for the batch-normalization output to be used as an input to skip connections, a feature that cannot be done when fusing layers. Our optimizations achieve low latency and minimal resource utilization.

4.2.6 ***Non-linear activations.*** ReLU and tanh are the two non-linear activation functions are used in FastStamp. The ReLU function, which is used in the U-net downsampling and upsampling layers computes the following function for each input x, $y = \max(0, x)$. Due to the simplicity of the ReLU activation function, we use loop unrolling to optimize latency for this operation. The tanh activation function computes the following function for each input x, $y = (e^x - e^{-x})/(e^x + e^{-x})$. We use the default tanh implementation in *hls4ml* which is performed using a pre-computed lookup table to reduce latency.

5 EXPERIMENTS AND RESULTS

5.1 Dataset

We conduct experiments on the CelebA dataset [31] which is a large database of over 200,000 face images of 10,000 unique celebrities. We set aside 1000 images for testing the watermarking models and split the remaining data into 80% training and 20% validation. All FastStamp models are trained using images of size 128×128, which are obtained after center-cropping and resizing the CelebA images. We conduct experiments with message bit length $L = 128$.

5.2 Evaluation Metrics

For the evaluation of our watermarking techniques, we investigate the following metrics based on prior works [15, 19].

1. Imperceptibility: We compute **peak signal to noise ratio (PSNR)** and **structural similarity index (SSIM)** between the watermarked and original images. A higher value of both these metrics indicates a more imperceptible watermark.

2. Capacity: Capacity measures the amount of information that can be embedded in the image. We use **bits per pixel (BPP)** which is calculated as $L/(HWC)$ where L is the message length and H, W, C indicate the height, width, and channels of the image. Higher BPP values indicate higher capacity.

3. Bit-Recovery Accuracy (BRA): BRA calculates the recovery accuracy of the bit string s. For robustness, we aim to have a high BRA when benign or intended transformations such as JPEG compression or color and contrast adjustments are applied. For semi-fragile watermarking systems, the goal is to have a low BRA when image tampering operations such as local tampering or FaceSwap are applied while maintaining robustness against benign transformations.

We compare our watermarking framework against three prior works on image watermarking—a DCT based semi-fragile watermarking system [45] and two robust neural image watermarking systems HiDDeN [52] and StegaStamp [41]. To evaluate the robustness and fragility of our watermarking systems, we perform the following transformations that are unseen during training:

	Model Size	*Capacity*			*Imperceptibility*		*BRA (%) – Benign*			*BRA (%) – Tampering*
Method	# Params	*H*,*W*	*L*	BPP	PSNR	SSIM	None	JPG-75	Filtering	FaceSwap
DCT (Semi-Fragile) [18]	—	128	256	5.2×10^{-3}	22.49	0.871	99.81	56.65	94.62	85.51
HiDDeN (Robust) [52]	411 k	128	30	6.1×10^{-4}	27.57	0.934	97.06	72.71	94.52	—
StegaStamp (Robust) [41]	528 k	400	100	2.0×10^{-4}	29.39	0.925	99.92	99.91	99.84	—
FastStamp (Robust)	45 k	128	128	2.3×10^{-3}	30.65	0.942	100.00	99.84	99.78	—
FastStamp (Semi-Fragile)	45 k	128	128	2.3×10^{-3}	30.64	0.940	100.00	99.74	99.72	51.11

Table 1: Capacity, imperceptibility, and BRA metrics of different watermarking systems. H, W indicates the height and width of the input image. For both robust and semi-fragile watermarking systems a high BRA is desirable for benign transforms. For semi-fragile watermarking systems, a low BRA is desirable for tampering transforms.

(1) **JPEG Compression:** Digital images are usually stored in a lossy format such as JPEG. We compress the watermarked images using JPEG-75 compression and measure the decoding BRA.
(2) **Filtering:** We apply a set of real-world image filtering operations using the Pilgram library [23] that simulates photo editing filters that are common on social media. These include color, contrast, and lighting adjustments.
(3) **Face Swapping:** For evaluating semi-fragile watermarking systems, we simulate image tampering by performing face swapping using the open source implementation of FaceSwap [26]. A low BRA is desirable against this transform to detect tampering.

5.3 Training and Architecture Optimization

Our encoder model follows the depthwise separable convolutional U-Net architecture as discussed in Section 3.3. To find the optimum architecture, we create different-sized versions of the baseline U-Net architecture by reducing the number of channels in each layer by a factor of 2, 4 and 8. We find that reducing the number of channels by a factor of 4 does not compromise model performance while being significantly lighter than the base U-Net model.

We train two variants of this optimized design in the robust and semi-fragile settings using the training technique described in Section 3.1. In the robust setting, we simulate differentiable JPEG compression, Gaussian blur, color, and contrast adjustment as the benign transforms g_b during training. In the semi-fragile setting, in addition to the benign transforms, we simulate differentiable localized tampering as the malicious transform g_m during training. We train our models for 200 k mini-batch iteration with a fixed learning rate of 1.5×10^{-4} using Adam optimizer. Table 1 compares our optimized models *FastStamp (Robust)* and *FastStamp (Semi-Fragile)* against prior neural and DCT based image watermarking frameworks. As compared to prior neural image watermarking and steganography models, FastStamp is significantly smaller and achieves a similar BRA with slightly improved imperceptibility as compared to StegaStamp [41] and HiDDeN [52]. The improvement in imperceptible metrics is achieved by using nearest neighbor 2D upsampling in the U-Net architecture as opposed to transposed convolutions in prior work.

5.4 Design Space Exploration

We implement the individual submodules described in Section 4.2 using Vivado HLS for the Xilinx XCVU13P FPGA board. First, we perform a search over the bit-width of the fixed point representation of the network weights and intermediate outputs. Our goal is to find the lowest bit-width that does not compromise on message recovery and imperceptibility. Figure 6 indicates the BRA and PSNR of different bit-width implementations of FastStamp. Based on this analysis, we use a 16-bit representation with 6 bits for the integer and 10 bits for the decimal representation. Next, we explore pipelining and loop unrolling options in our Vivado HLS implementation of various submodules. Complete loop unrolling in submodule implementations was an infeasible design choice for FastStamp since it exceeded the available resources on the device. We found that partial loop unrolling with factors of 8 and pipelining loops that were completely unrolled were the most effective optimizations for FastStamp.

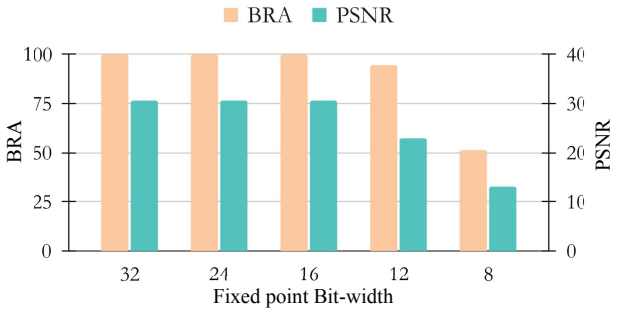

Figure 6: Watermarking success metrics for different fixed-point representations. A high value for both BRA and PSNR is desirable for accurate message recovery and imperceptibility.

While effective pipelining and loop unrolling resulted in a significant reduction in resource utilization, the LUT requirement of our design still exceeded the available resources on the device. Reducing the loop unrolling factor was an effective strategy to reduce LUT utilization but resulted in significant increases in encoding latency. To avoid latency increase, the next strategy we applied was distributing variables, such as model weights, that were initially all implemented as LUTs, to the BRAM on our board. We also force

	Resource Utilization (%)				*Performance*			*Correctness*		
Design	BRAM	FF	LUT	DSP	Clock	Latency	Throughput	BRA	PSNR	SSIM
Available Resources	94 Mb	3456K	1728K	12288	Period (ns)	(# Cycles)	(Hz)			
FixedPoint-32	>100%	51%	>100%	>100%	—	—	—	100.0	30.67	0.942
FixedPoint-16	89%	18%	>100%	54%	—	—	—	100.0	30.64	0.941
FixedPoint-16-Optimized	59%	14%	72%	53%	5	596823	335	100.0	30.64	0.941

Table 2: Design-space exploration for FPGA implementation of FastStamp on Xilinx XCVU13P FPGA board. We report the resource utilization, timing performance, and implementation correctness for various designs. Our optimized 16-bit fixed point implementations fit within the available resources while maintaining the same correctness metrics as the 32-bit implementation.

operations, such as multiplication in the separable convolution into DSP blocks, which results in lower LUT utilization. We utilize per layer reuse factor to tune the inference latency versus utilization of FPGA resources and enable parallelization. This allows us to process multiple MAC operations at every unit of time. Using all of these optimizations, we are able to store our model entirely in the on-chip memory of the FPGA and perform low latency encoding while avoiding communication with off-chip memory. Table 2 lists the resource utilization, performance, and correctness of some of the design choices that led to our most optimized design, *FixedPoint-16-Optimized*. Figure 7 shows sample outputs of this optimized FPGA implementation and compares them with the GPU implementation in PyTorch.

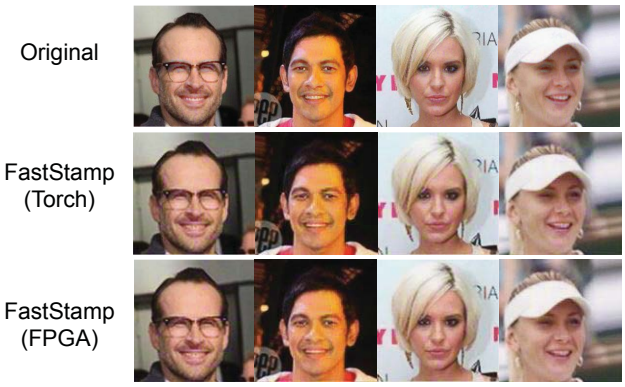

Figure 7: Sample image outputs of FastStamp optimized design and PyTorch implementation with the original image

5.5 Performance and Power Analysis

Table 3 compares the inference time and power requirement of our optimized FPGA implementation with the highly optimized CPU and GPU implementation of FastStamp and the open source implementations of prior neural watermarking systems. We benchmark the optimized PyTorch implementation of FastStamp on the Nvidia Tesla V100 GPU. The CPU implementation is a NumPy inference program written by us and optimized fully. We measure the power consumption for the GPU benchmarks using the Nvidia power measurement tool (*Nvidia-smi*) running on *Linux* operating system, which is invoked during program execution. For our FPGA implementations, we synthesize our designs using Xilinx Vivado 2020.1. We then integrate the synthesized modules accompanied by the corresponding peripherals into a system-level schematic using the Vivado IP Integrator. The frequency is set to 200 MHz, and power consumption is estimated using the synthesis tool. Our FPGA implementation achieves ~68× faster speed against prior work's GPU implementation and ~10× faster speed against FastStamp's GPU implementation at a 3× lower power requirement.

Implementation	Time (ms)	Power (W)
StegaStamp GPU [41]	205	76
HiDDeN GPU [52]	234	65
FastStamp CPU	326	—
FastStamp GPU	30	59
FastStamp FPGA	3	19

Table 3: Power consumption and wall-clock time (in milliseconds) required to generate a single watermarked image per implementation.

6 CONCLUSION

We propose an efficient image watermarking model that matches or even outperforms prior neural image watermarking and steganography models while utilizing significantly fewer parameters. By leveraging an efficient secret message upsampling module, depth-wise separable convolutions, and 2-D upsampling, we were able to train a smaller model while preserving success metrics for watermarking tasks. Finally, we implement our encoder model on an FPGA to achieve 68× higher throughput as compared to prior GPU implementations. Our implementation allows watermark embedding directly at the hardware source, which not only secures the image capture and transmission pipeline but also reduces latency in embedding the watermark. In the process of this implementation, we develop reconfigurable sub-modules which can accelerate convolutional downsampling and upsampling networks on hardware.

7 ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation (NSF) Trust-Hub under award number CNS2016737, NSF TILOS under award number CCF-2112665, NSF Cooperative Agreement OAC-2117997 (a3d3.ai), DoD UCR W911NF2020267 (MCA S-001364), and ARO MURI under award number W911NF-20-S-0009. Thanks to the Fast ML collective (fastmachinelearning.org) for support.

REFERENCES

[1] Thea Aarrestad et al. 2021. Fast convolutional neural networks on FPGAs with hls4ml. *MLST* 2, 4 (2021), 045015. https://doi.org/10.1088/2632-2153/ac0ea1 arXiv:2101.05108 [cs.LG]

[2] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. 2018. Mesonet: a compact facial video forgery detection network. In *2018 IEEE International Workshop on Information Forensics and Security (WIFS)*. IEEE.

[3] Lin Bai, Yiming Zhao, and Xinming Huang. 2018. A CNN Accelerator on FPGA Using Depthwise Separable Convolution. *IEEE Transactions on Circuits and Systems II: Express Briefs* (2018).

[4] Shumeet Baluja. 2017. Hiding images in plain sight: Deep steganography. *NIPS* (2017).

[5] Oussama Benrhouma, Houcemeddine Hermassi, and Safya Belghith. 2015. Tamper detection and self-recovery scheme by DWT watermarking. *Nonlinear Dynamics* (2015).

[6] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. 2020. StarGAN v2: Diverse Image Synthesis for Multiple Domains. In *CVPR*. 8185. https://doi.org/10.1109/CVPR42600.2020.00821

[7] François Chollet. 2017. Xception: Deep Learning with Depthwise Separable Convolutions. In *CVPR*. 1800. https://doi.org/10.1109/CVPR.2017.195

[8] Ingemar J Cox, Joe Kilian, F Thomson Leighton, and Talal Shamoon. 1997. Secure spread spectrum watermarking for multimedia. *IEEE Trans. Image Process.* (1997).

[9] Ferdinando Di Martino and Salvatore Sessa. 2019. Fragile watermarking tamper detection via bilinear fuzzy relation equations. *J. Ambient Intell. Humaniz. Comput.* (2019).

[10] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. 2020. The DeepFake Detection Challenge (DFDC) dataset. (2020). arXiv:2006.07397

[11] Javier Duarte et al. 2018. Fast inference of deep neural networks in FPGAs for particle physics. *JINST* 13, 07 (2018), P07027. https://doi.org/10.1088/1748-0221/13/07/P07027 arXiv:1804.06913 [physics.ins-det]

[12] Farah Fahim et al. 2021. hls4ml: An Open-Source Codesign Workflow to Empower Scientific Low-Power Machine Learning Devices. In *tinyML Research Symposium 2021*. arXiv:2103.05579 [cs.LG]

[13] Ekaterina Govorkova et al. 2022. Autoencoders on field-programmable gate arrays for real-time, unsupervised new physics detection at 40 MHz at the Large Hadron Collider. *Nat. Mach. Intell.* 4 (2022), 154. https://doi.org/10.1038/s42256-022-00441-3 arXiv:2108.03986

[14] Chongyan Gu, Neil Hanley, and Máire O'neill. 2017. Improved reliability of FPGA-based PUF identification generator design. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* (2017).

[15] Mohamed Ali Hajjaji, Mohamed Gafsi, Abdessalem Ben Abdelali, and Abdellatif Mtibaa. 2019. FPGA implementation of digital images watermarking system based on discrete Haar wavelet transform. *Security and Communication Networks* (2019).

[16] Jamie Hayes and George Danezis. 2017. Generating steganographic images via adversarial training. In *NIPS*.

[17] Sambaran Hazra, Sudip Ghosh, Sayandip De, and Hafizur Rahaman. 2018. FPGA implementation of semi-fragile reversible watermarking by histogram bin shifting in real time. *Journal of Real-Time Image Processing* (2018).

[18] Chi Kin Ho and Chang-Tsun Li. 2004. Semi-fragile watermarking scheme for authentication of JPEG images. In *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.* IEEE.

[19] Alain Hore and Djemel Ziou. 2010. Image quality metrics: PSNR vs. SSIM. In *2010 20th international conference on pattern recognition*. IEEE.

[20] Shehzeen Hussain, Mojan Javaheripi, Paarth Neekhara, Ryan Kastner, and Farinaz Koushanfar. 2021. FastWave: Accelerating Autoregressive Convolutional Neural Networks on FPGA. In *ICCAD*.

[21] Shehzeen Hussain, Paarth Neekhara, Malhar Jere, Farinaz Koushanfar, and Julian McAuley. 2021. Adversarial Deepfakes: Evaluating Vulnerability of Deepfake Detectors to Adversarial Examples. In *WACV*.

[22] Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on Machine Learning*, Francis Bach and David Blei (Eds.). PMLR. https://proceedings.mlr.press/v37/ioffe15.html

[23] Akiomi Kamakura. 2019. pilgram https://github.com/akiomik/pilgram.

[24] Rohollah Mazrae Khoshki, Sami Oweis, Shumei Wang, George Pappas, and Subramaniam Ganesan. 2014. FPGA hardware based implementation of an image watermarking system. *Int. J. Adv. Res. Comput.* (2014).

[25] S Kiran, KV Nadhini Sri, and J Jaya. 2013. Design and implementation of FPGA based invisible image watermarking encoder using wavelet transformation. In *2013 International Conference on Current Trends in Engineering and Technology (ICCTET)*. IEEE.

[26] Marek Kowalski. 2018. FaceSwap https://github.com/MarekKowalski/FaceSwap/.

[27] Chunlei Li, Aihua Zhang, Zhoufeng Liu, Liang Liao, and Di Huang. 2015. Semi-fragile self-recoverable watermarking algorithm based on wavelet group quantization and double authentication. *Multimedia tools and applications* (2015).

[28] Eugene T Lin, Christine I Podilchuk, and Edward J Delp III. 2000. Detection of image alterations using semifragile watermarks. In *Security and Watermarking of Multimedia Contents II*. International Society for Optics and Photonics.

[29] Ming Liu, Yukang Ding, Min Xia, Xiao Liu, Errui Ding, Wangmeng Zuo, and Shilei Wen. 2019. STGAN: A Unified Selective Transfer Network for Arbitrary Image Attribute Editing. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[30] Shuanglong Liu, Hongxiang Fan, Xinyu Niu, Ho-cheung Ng, Yang Chu, and Wayne Luk. 2018. Optimizing CNN-based segmentation with deeply customized convolutional and deconvolutional architectures on FPGA. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* (2018).

[31] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep Learning Face Attributes in the Wild. In *ICCV*.

[32] Xiyang Luo, Ruohan Zhan, Huiwen Chang, Feng Yang, and Peyman Milanfar. 2020. Distortion agnostic deep watermarking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

[33] Yisroel Mirsky and Wenke Lee. 2021. The Creation and Detection of Deepfakes: A Survey. *ACM Comput. Surv.* (2021).

[34] Paarth Neekhara, Shehzeen Hussain, Xinqiao Zhang, Ke Huang, Julian McAuley, and Farinaz Koushanfar. 2022. FaceSigns: Semi-Fragile Neural Watermarks for Media Authentication and Countering Deepfakes. (2022). arXiv:2204.01960

[35] Augustus Odena, Vincent Dumoulin, and Chris Olah. 2016. Deconvolution and Checkerboard Artifacts. *Distill* (2016). https://doi.org/10.23915/distill.00003

[36] RO Preda and DN Vizireanu. 2015. Watermarking-based image authentication robust to JPEG compression. *Electronics Letters* (2015).

[37] Amna Qureshi, David Megías, and Minoru Kuribayashi. 2021. Detecting Deepfake Videos using Digital Watermarking. In *2021 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*.

[38] Mohammad Samragh, Mohammad Ghasemzadeh, and Farinaz Koushanfar. 2017. Customizing Neural Networks for Efficient FPGA Implementation. In *Field-Programmable Custom Computing Machines (FCCM), 2017 IEEE 25th Annual International Symposium on*. IEEE.

[39] Abdulaziz Shehab, Mohamed Elhoseny, Khan Muhammad, Arun Kumar Sangaiah, Po Yang, Haojun Huang, and Guolin Hou. 2018. Secure and robust fragile watermarking scheme for medical images. *IEEE Access* (2018).

[40] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. 2016. Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM.

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

[42] R Tay and JP Havlicek. 2002. Image watermarking using wavelets. In *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002.* IEEE.

[43] Run Wang, Felix Juefei-Xu, Meng Luo, Yang Liu, and Lina Wang. 2021. Faketagger: Robust safeguards against deepfake dissemination via provenance tracking. In *Proceedings of the 29th ACM International Conference on Multimedia*. 3546.

[44] Jun Xiao and Ying Wang. 2008. A semi-fragile watermarking tolerant of Laplacian sharpening. In *2008 International Conference on Computer Science and Software Engineering*. IEEE.

[45] Hengfu Yang, Xingming Sun, and Guang Sun. 2009. A semi-fragile watermarking algorithm using adaptive least significant bit substitution. *J. Inf. Technol.* (2009).

[46] Byeongheon Yoo, Yongjun Choi, and Heeyoul Choi. 2018. Fast Depthwise Separable Convolution for Embedded Systems. In *Neural Information Processing*, Long Cheng, Andrew Chi Sing Leung, and Seiichi Ozawa (Eds.). Springer International Publishing, Cham.

[47] Xiaoyan Yu, Chengyou Wang, and Xiao Zhou. 2017. Review on semi-fragile watermarking algorithms for content authentication of digital images. *Future Internet* (2017).

[48] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. ACM.

[49] Ru Zhang, Shiqi Dong, and Jianyi Liu. 2019. Invisible steganography via generative adversarial networks. *Multimedia tools and applications* (2019).

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

[51] Wei Zhao, Zuchen Jia, Xiaosong Wei, and Hai Wang. 2018. An FPGA implementation of a convolutional auto-encoder. *Applied Sciences* (2018).

[52] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. 2018. HiDDeN: Hiding Data With Deep Networks. In *ECCV*. 682. https://doi.org/10.1007/978-3-030-01267-0_40