

FaceGuard: Proactive Deepfake Detection

Yuankun Yang^{1*}, Chenyue Liang^{2*}, Hongyu He³, Xiaoyu Cao³, Neil Zhenqiang Gong³

¹Fudan University, 17307110068@fudan.edu.cn

²Chinese Academy of Sciences, llcy_cheryl@outlook.com

³Duke University, {hongyu.he, xiaoyu.cao, neil.gong}@duke.edu

Abstract

Existing deepfake-detection methods focus on *passive* detection, i.e., they detect fake face images via exploiting the artifacts produced during deepfake manipulation. A key limitation of passive detection is that it cannot detect fake faces that are generated by new deepfake generation methods. In this work, we propose *FaceGuard*, a *proactive* deepfake-detection framework. FaceGuard embeds a watermark into a real face image before it is published on social media. Given a face image that claims to be an individual (e.g., Nicolas Cage), FaceGuard extracts a watermark from it and predicts the face image to be fake if the extracted watermark does not match well with the individual’s ground truth one. A key component of FaceGuard is a new deep-learning-based watermarking method, which is 1) robust to normal image post-processing such as JPEG compression, Gaussian blurring, cropping, and resizing, but 2) fragile to deepfake manipulation. Our evaluation on multiple datasets shows that FaceGuard can detect deepfakes accurately and outperforms existing methods.

1 Introduction

As deep learning becomes more and more powerful, deep learning based *deepfake generation methods* can produce more and more realistic-looking deepfakes [8, 18, 19, 20, 30, 35, 41, 42, 51, 56]. In this work, we focus on fake faces because faces are key ingredients in human communications. Moreover, we focus on *manipulated* fake faces, in which a deepfake generation method replaces a target face as a source face (known as *face replacement*) or changes the facial expressions of a target face as those of a source face (known as *face reenactment*). For instance, in the well-known Trump-Cage deepfakes example [34], Trump’s face (target face) is replaced as Cage’s face (source face). Fake faces can be used to assist the propagation of fake news, rumors, and disinformation on social media (e.g., Facebook, Twitter, and Instagram). Therefore, fake faces pose growing concerns to the integrity of online information, highlighting the urgent needs for deepfake detection.

Existing deepfake detection mainly focuses on *passive* detection, which exploits the artifacts in fake faces to detect them after they have been generated. Specifically, given a face image, a passive detector extracts various features from it and classifies it to be real or fake based on the features. The features can be manually designed based on some heuristics [2, 14, 22, 23, 27, 50] or automatically extracted by a deep neural network based feature extractor [1, 6, 11, 14, 28, 29, 37, 38, 48, 54]. Passive detection faces a key limitation [7], i.e., it cannot detect fake faces that are generated by new deepfake generation methods that were not considered when training the passive detector. As new deepfake generation methods are continuously developed, this limitation poses significant challenges to passive deepfake detection.

Our work: In this work, we propose *FaceGuard*, a *proactive* deepfake-detection framework. FaceGuard addresses the limitation of passive detection via proactively embedding watermarks into real face images before they are manipulated by deepfake generation methods. Figure 1 illustrates the difference between passive detection and FaceGuard. Specifically, before posting an individual’s real face image on social media, **FaceGuard embeds a watermark (i.e., a binary vector in our work) into it.** The watermark is human imperceptible, i.e., a face image and its watermarked version look visually the same to human eyes. For instance, the watermark can be embedded into an individual’s face image using the individual’s smartphone. Suppose a face image is claimed to be an individual, e.g., the manipulated

*The first two authors made equal contributions. They performed this research when they were remote interns in Gong’s group.

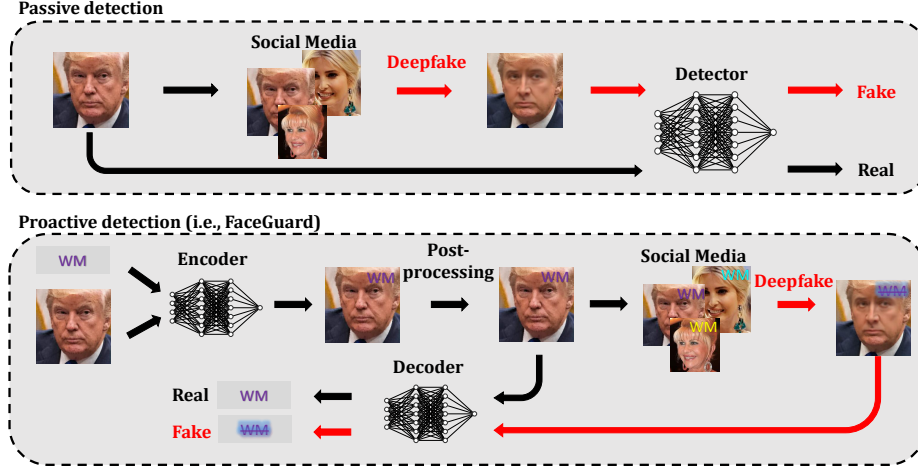


Figure 1: FaceGuard v.s. passive detection.

face is claimed to be Nicolas Cage in the Trump-Case deepfakes example. FaceGuard extracts a watermark from the face image and predicts it to be fake if the fraction of matched bits between the extracted binary-vector watermark and the individual’s ground truth one is smaller than a threshold.

The key components of FaceGuard include an *encoder* and a *decoder*. Both the encoder and decoder are neural networks, but they may have different architectures and parameters. The encoder takes a face image and a watermark as input, and outputs a watermarked face image. The decoder takes a (watermarked) face image as input and outputs a binary-vector watermark. FaceGuard aims to achieve two key properties. On the one hand, a watermarked face image may be post-processed, e.g., via **JPEG compression, Gaussian blurring, cropping, and resizing**. For instance, during transmission on the Internet, images are often JPEG compressed to reduce their sizes. Our watermarks in FaceGuard should be robust to such post-processing, i.e., our decoder can still extract an individual’s watermark from its post-processed watermarked face images. Therefore, FaceGuard can still predict such post-processed watermarked real face images as real. On the other hand, an attacker downloads a watermarked face image from a social media, uses a deepfake generation method to manipulate it, and tries to re-post the fake face image on the social media. Our watermarks should be fragile to deepfake manipulation, i.e., our decoder cannot extract the individual’s watermark from a fake face image claiming as the individual. Therefore, the social media can detect the fake face images using FaceGuard.

To achieve the two properties above, we jointly train the encoder and decoder using a set of real face images. In particular, we formulate a loss function, which consists of a *reconstruction loss* of the watermark as well as a *pixel loss* and an *adversarial loss* that capture the perceptual difference between a real face image and its watermarked version. We learn an encoder and decoder via minimizing the loss function using stochastic gradient descent. Moreover, we add a *post-processing module* between the encoder and decoder, which applies post-processing operations of interest to the watermarked images produced by the encoder before feeding them into the decoder. However, the post-processing module introduces two key challenges to training. The first challenge is that, JPEG compression, a popular post-processing operation, is non-differentiable, making it impossible to backpropagate gradients from decoder to encoder. The second challenge is how to schedule different post-processing operations for each mini-batch of training images, given multiple post-processing operations.

To address the first challenge, we propose to approximate the quantization/dequantization components of JPEG, which enables us to backpropagate meaningful gradients during training. To address the second challenge, we propose a reinforcement learning based scheduler to determine which post-processing operation should be applied for each mini-batch of training images.

We empirically evaluate FaceGuard using multiple datasets, including FaceForensics [38], Facebook Deepfake Detection Challenge [13], and a Trump-Cage dataset, as well as three deepfake generation methods (two face replacement methods and one face reenactment method). Our results show that FaceGuard can accurately detect fake faces generated by different deepfake generation methods; FaceGuard outperforms passive detection; our watermarking method in FaceGuard outperforms existing ones; and FaceGuard is robust against adaptive deepfakes in which an attacker trains its own encoder/decoder to embed watermarks into its deepfakes.

In summary, our contributions are as follows:

- We propose FaceGuard, a proactive deepfake-detection framework.
- We propose a method to make JPEG compression differentiable and a reinforcement learning based method to schedule post-processing operations when jointly training the encoder and decoder in FaceGuard.
- We extensively evaluate FaceGuard and compare it with state-of-the-art methods.

2 Related Work

2.1 Deepfake Generation

Fake face images can be generated via *face synthesizing* [8, 18, 19, 20, 35, 56] or *face manipulation* [30, 31, 41, 42, 51]. Face synthesizing creates new faces that do not belong to any individual in the world, while face manipulation manipulates the face images of existing individuals. Generally speaking, face manipulation may result in more severe consequences than face synthesizing, as it may target individuals with high social impact, e.g., President of United States and celebrities. Therefore, we focus on face manipulation in this work.

Face manipulation can be further categorized into *face replacement* [31, 51] and *face reenactment* [30, 41, 42]. In face replacement, a target face is replaced as a source face and the generated fake face claims to be the source. For instance, given face images of a target individual and a source individual, FaceSwap [31] trains an autoencoder for each individual, where the encoders of the two autoencoders are shared. When generating fake faces, FaceSwap first encodes a target face image using the shared encoder and then decodes it using the source’s decoder, which transforms the target face into the source face. In face reenactment, the expression, hair color, and/or other properties of the target face are changed as those of a source face. For instance, ICface [44] trains GANs in a self-supervised manner to utilize human interpretable control signals on emotions and head poses. Face replacement changes the identity of the target face to that of the source, while face reenactment preserves the target’s face identity.

2.2 Deepfake Detection

Existing deepfake-detection methods are mainly *passive*. These passive deepfake-detection methods fall into two categories, i.e., *heuristics-based methods* [2, 14, 22, 23, 27, 50] and *deep-learning-based methods* [1, 6, 11, 14, 16, 17, 28, 29, 37, 38, 40, 47, 48, 53, 54]. Heuristic-based methods leverage some heuristic features to detect fake faces. For instance, Li et al. [22] detected fake faces by analyzing eye-blinking of the person in a video. Deep-learning-based methods train binary deep neural network (DNN) classifiers to perform detection, i.e., a binary DNN classifier takes a face image as input and outputs fake or real. Deep-learning-based methods outperform heuristics-based methods. However, a recent study [7] shows that an attacker can evade passive detection via various strategies, e.g., an attacker can use a new deepfake generation method to generate fake faces that will be misclassified as real by a passive detector.

Recently, Wang et al. proposed FakeTagger [46], which aims to track image reuse on social media. The idea is to embed tags to real face images. The tags are robust against both normal post-processing and deepfake manipulations of the tagged face images. FakeTagger predicts a face image as fake if the tag can be extracted from it. However, FakeTagger falsely predicts normally post-processed versions of real face images as fake because the tags are robust against normal post-processing. For instance, if a real face image is downloaded from a social media, JPEG compressed, and re-uploaded to the social media, it will be falsely flagged as fake by FakeTagger. In contrast, our FaceGuard leverages semi-fragile watermarks, which are robust against normal post-processing but are fragile to deepfake manipulations. Therefore, FaceGuard can distinguish fake faces from normally post-processed real ones.

2.3 Watermarks

Watermarking [43] was originally proposed for copyright protection. For instance, a watermark can be embedded into an image as a signature of ownership, which can be extracted later for ownership verification. Conventional watermarks include distortions in both *spatial domain* [43, 45] and *transform domain*, such as discrete cosine transform [5, 10], discrete wavelet transform [4, 15], and discrete Fourier transform [12, 39]. The key limitation of conventional watermarks is that they are not robust enough to common image post-processing [55].

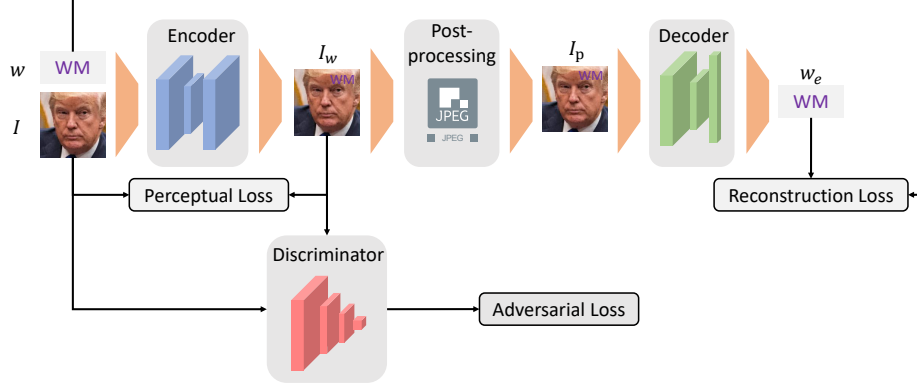


Figure 2: Overview of jointly training the encoder and decoder in FaceGuard.

To address such limitation, recent works [3, 24, 26, 52, 55] proposed deep-learning-based watermarks. Specifically, in these methods, an *encoder* takes an image and a watermark as input and outputs a watermarked image, while a *decoder* takes a (watermarked) image as input and outputs a watermark. Moreover, given a set of images, they aim to train the encoder and decoder such that the decoder can still extract the correct watermark from a watermarked image even if it has normal post-processing such as JPEG compression.

For instance, HiDDeN [55] proposed a post-processing module between the encoder and decoder during training, where the post-processing module applies post-processing to the watermarked images. In particular, for each mini-batch during training, HiDDeN randomly picks one post-processing from a set of predefined ones uniformly at random and applies it to the watermarked images in the mini-batch. A key challenge of such post-processing module based training is that JPEG compression is not differentiable, which means that the gradients cannot be backpropagated from the decoder to encoder during training. To address this challenge, HiDDeN proposed to remove the non-differentiable quantization/dequantization steps of JPEG compression and simply set high-frequency coefficients in DCT as zero. However, as we will show in experiments, HiDDeN is not robust to JPEG compression.

3 Training Encoder and Decoder in FaceGuard

Figure 2 shows the end-to-end training workflow of our encoder and decoder. **Specifically, we use binary vectors as our watermarks.** Given a real face image I and a binary-vector watermark w , the encoder outputs a watermarked face image I_w . To achieve robustness against normal post-processing, we place a post-processing module in between the encoder and the decoder. This module performs a normal post-processing operation on a watermarked image and produces a post-processed watermarked image I_p . During our training, the module switches between different normal post-processing operations in different mini-batches. In particular, for each mini-batch of the training images, we select one normal post-processing operation and apply it in the module. The decoder extracts a binary-vector watermark w_e from a post-processed watermarked image I_p .

Formulating an optimization problem: We formulate jointly learning the encoder and decoder as an optimization problem. First, the extracted watermark w_e produced by the decoder should be similar to the true watermark w . Therefore, we formulate **a reconstruction loss** $\|w_e - w\|_2^2$. Second, we aim to produce human imperceptible watermarks, i.e., an image I and its watermarked version I_w should be visually similar. To achieve this goal, we formulate a *pixel loss* and an *adversarial loss*. In particular, we use $\|I_w - I\|_2^2$ as a pixel loss, which measures the ℓ_2 -norm of the difference between an image and its watermarked version. Moreover, inspired by [55], we use a discriminator to distinguish between real images and their watermarked versions. Given an image I , the discriminator outputs $D(I)$, i.e., the probability that I is a non-watermarked image. We aim to train the encoder such that its output I_w has high probability $D(I_w)$ predicted by the discriminator. Therefore, we use $\log(1 - D(I_w))$ as an adversarial loss. To summarize, given a discriminator D , we formulate **a loss function** $\mathcal{L} = \|w_e - w\|_2^2 + \lambda_1 \|I_w - I\|_2^2 + \lambda_2 \log(1 - D(I_w))$ for the encoder and decoder, where λ_1 and λ_2 are used to balance the pixel loss and adversarial loss. In our framework, we also jointly learn the discriminator. Formally, we jointly learn the encoder, decoder, and discriminator via solving the following optimization problem:

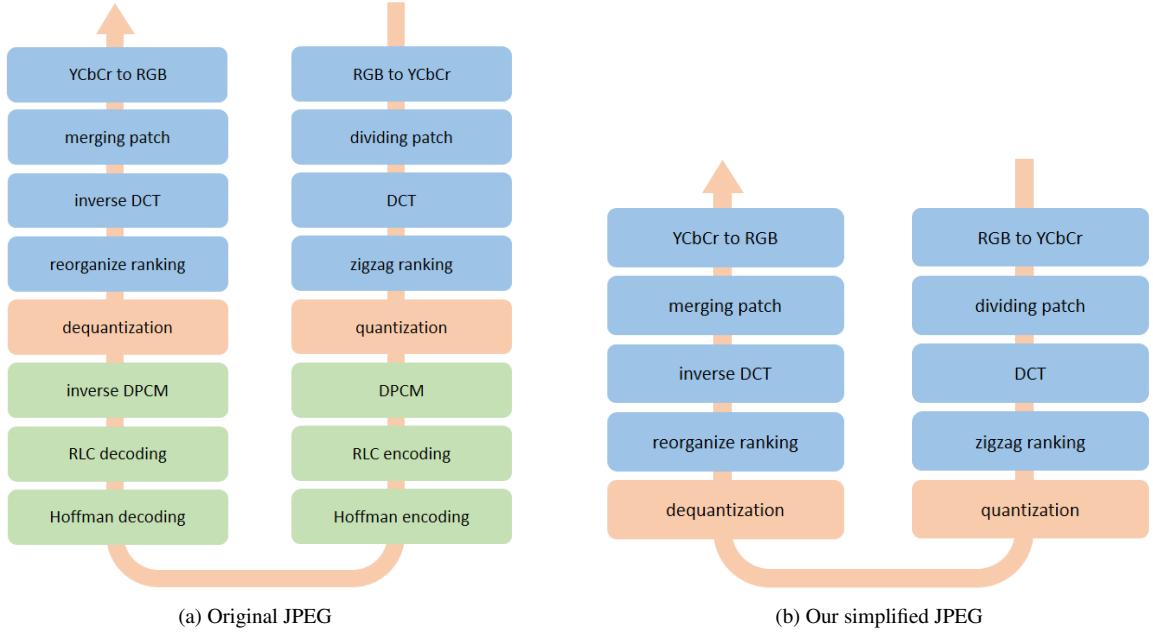


Figure 3: Illustration of the original JPEG compression procedure and our simplified JPEG compression procedure.

$$\min_{\theta, \phi, \psi} \mathbb{E}_{I, w} [\mathcal{L}] + \mathbb{E}_{I, w} [\log(1 - D(I)) + \log(D(I_w))], \quad (1)$$

where θ , ϕ , and ψ are the parameters of the encoder, the decoder, and the discriminator, respectively; and the expectation is taken with respect to the distribution of real face image I and distribution of watermark w . **In experiments, we use a set of real face images and generate random binary-vector watermarks in each mini-batch of training images to calculate the expectation.**

Solving the optimization problem: We alternatively optimize the encoder/decoder and the discriminator. Specifically, given the current discriminator, we update the encoder and decoder via solving Equation (1) using stochastic gradient descent; and given the current encoder and decoder, we update the discriminator via solving Equation (1) using stochastic gradient descent. However, there are two key challenges when updating the encoder and decoder for a given discriminator. **First**, although the post-processing module does not have trainable parameters, it needs to be differentiable for backpropagating gradients from the decoder to encoder. Despite most post-processing operations satisfy the differentiability requirement, JPEG, one of the most common and important post-processing operations, is not differentiable. **Second**, for each mini-batch of the training images, we need to select a post-processing operation to apply. A naive approach is to pick a post-processing operation among a set of predefined ones uniformly at random, as used by HiDDeN. However, this naive scheduling leads to sub-optimal performance as shown in our experiments.

To address the first challenge, we propose an approximation of JPEG compression that makes backpropagation possible. Moreover, we leverage reinforcement learning to schedule the different post-processing operations to address the second challenge. Next, we discuss the details.

3.1 Approximating JPEG Compression for Backprop

Figure 3a illustrates how JPEG compression works. Most of the steps are differentiable, except the quantization/dequantization ones. For simplicity, we omit the steps between quantization and dequantization as they are invertible operations, whose combination is equivalent to the identity mapping. Our simplified JPEG compression workflow is shown in Figure 3b.

In the quantization step, the input is element-wisely divided by a quantization matrix and rounded to the nearest integers. In the dequantization step, the rounded values are multiplied by the same quantization matrix to reconstruct

the initial input. Specifically, let x denote the input to the quantization step and $g(x)$ denote the output of the dequantization step. Formally, $g(x)$ is defined as follows in JPEG:

$$g(x) = \begin{cases} x - (x\%1), & x\%1 < 0.5 \\ x + 1 - (x\%1), & x\%1 \geq 0.5, \end{cases} \quad (2)$$

where $x\%1$ represents the **decimal** part of x . Due to the rounding operation in quantization, different inputs to the quantization step may lead to the same output after dequantization. Moreover, the rounding operation makes the gradient through the quantization-dequantization operations meaningless. Specifically, the gradient $\nabla_x g(x)$ is zero everywhere except when the decimal part of x is 0.5, where $g(x)$ is non-differentiable. Formally, we have:

$$\nabla_x g(x) = \begin{cases} 0, & x\%1 \neq 0.5 \\ \text{none}, & x\%1 = 0.5, \end{cases} \quad (3)$$

To address the aforementioned challenge, we approximate the gradient of the quantization-dequantization steps by explicitly characterizing the changes in x . In particular, we approximate the quantization-dequantization steps as a linear transformation $g(x) = kx$, where k is as follows:

$$k = \begin{cases} \frac{x - x\%1}{x}, & x\%1 < 0.5 \wedge x \neq 0 \\ \frac{x + 1 - x\%1}{x}, & x\%1 \geq 0.5 \\ 0, & x = 0. \end{cases} \quad (4)$$

Here, k is the ratio between the output and the input of the quantization-dequantization steps, which quantifies the change to x introduced by these steps. Therefore, we use it to approximately represent the gradient. Specifically, in the forward pass, we calculate the value of $g(x)$ following Equation (2). However, in the backward pass, instead of using Equation 3, we treat k as a constant and consider it as our gradient, i.e., we have $\nabla_x g(x) = k$. Our approximation makes it possible to obtain meaningful end-to-end gradients in our training workflow, which enables us to learn the encoder and decoder via stochastic gradient descent.

3.2 Scheduling Post-processing via Reinforcement Learning

Given multiple commonly used post-processing operations, the post-processing module selects one for each mini-batch of training images. The *random scheduler*, which picks a post-processing operation uniformly at random, achieves sub-optimal performance because the difficulty of fitting different post-processing operations varies. For instance, assume we have two post-processing operations p_1 and p_2 . p_1 is easy to fit, while p_2 is hard. A random scheduler essentially switches between p_1 and p_2 with equal probability. At some point during training, the encoder/decoder fits p_1 well, while still underfits p_2 due to the different difficulty levels. Beyond this point, ideally the scheduler should focus on p_2 . However, a random scheduler will still vacillate between the two, leading to overfit p_1 and/or underfit p_2 .

Therefore, we design an advanced scheduler using reinforcement learning, which learns to adaptively switch among different post-processing operations. Specifically, we consider the choices of different post-processing operations as our states, and the switches between different choices as the set of actions. Assuming there are N post-processing operations. We denote the set of states as $\mathcal{S} = \{s^{(i)} | i = 1, 2, \dots, N\}$ and the set of actions as $\mathcal{A} = \{a^{(i)} | i = 1, 2, \dots, N\}$, where $a^{(i)}$ means the state will switch to $s^{(i)}$ for the next mini-batch. For an input image, we define its *bitwise accuracy* as the fraction of bits in the extracted binary-vector watermark w_e that match those in w . For a mini-batch B , we define its average bitwise accuracy f_t as the mean of the bitwise accuracy of images in the mini-batch. We further design our reward function as follows:

$$r(s_t, a_t) = \beta[f_t - h_{s_{t+1}}] + f_t + b_{s_{t+1}}. \quad (5)$$

In the reward function, β is a constant weight and s_{t+1} is the state for the $(t + 1)$ -th mini-batch, which depends on the action a_t . \mathbf{h} is an N -dimensional vector that records the previous bitwise accuracy for each state. \mathbf{b} is a constant N -dimensional vector that represents a bias for each state in the reward function based on the domain knowledge. $h_{s_{t+1}}$ and $b_{s_{t+1}}$ are the s_{t+1} -th entry in \mathbf{h} and \mathbf{b} , respectively.

Algorithm 1: Reinforcement-Learning-based Scheduler

Input: $\mathcal{S}, \mathcal{A}, r, \mathbf{h}, \mathbf{b}, \epsilon_t, s_t, B_t$, and Q .

Output: Selected post-processing operation for B_t .

```
1:  $u \leftarrow \text{Uniform}(0, 1)$ 
2: if  $u < 1 - \epsilon_t$  then                                     // exploitation
3:    $a_t \leftarrow \arg\max_{a \in \mathcal{A}} Q(s_t, a)$ 
4: else                                                         // exploration
5:    $a_t \leftarrow$  a random action  $a \in \mathcal{A}$ 
6: end if
7: Calculate average bitwise accuracy  $f_t$ 
8: Update the Q-table based on Equation (6)
9: Decay  $\epsilon_t$ 
10:  $h_{s_{t+1}} \leftarrow f_t$                                      //update h
11: return  $a_t$ 
```

We train our scheduler following Q-learning [49] in an online learning manner, i.e., we keep updating our scheduler at the same time we train our encoder and decoder. Specifically, we keep a $N \times N$ matrix of Q-values called Q-table. Each row in the matrix represents a state, while each column represents an action. The Q-value at location (i, j) indicates the benefit of taking action $a^{(j)}$ at state $s^{(i)}$. At the beginning, we initialize the Q-table Q with arbitrary values (e.g., all zeros), initialize the vector \mathbf{h} as all zeros, and start with a random state. Later, for each mini-batch B_t , we decide which post-processing to apply to B_t based on the Q-table. Considering the exploitation-exploration trade-off, with probability $1 - \epsilon_t$, we select the action with the largest Q-value at state s_t . With probability ϵ_t , we randomly pick an action to take. We set ϵ_t to be large at the beginning and decay it gradually. We perform the selected post-processing in between the encoder and the decoder, and compute the average bitwise accuracy f_t . With f_t , we update the history bitwise accuracy vector \mathbf{h} and calculate the reward function r_t following Equation (5). Moreover, we use r_t to update the Q-table following the Q-learning update rule, i.e., we have:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_t + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)], \quad (6)$$

where α is the learning rate and γ is a constant discount factor. Our scheduler is summarized in Algorithm 1.

4 Experiments

4.1 Experimental Setup

Datasets: We use face images from three popular datasets, i.e., FaceForensics++ [38], Facebook Deepfake Detection Challenge (DFDC) [13], and Trump-Cage [33].

FaceForensics++. FaceForensics++ consists of YouTube videos that have faces in them. We randomly pick two videos and extract face images from them. We consider one of them as the target and the other as the source. For each video, we split it into frames and extract face images from the frames using the publicly available package Dlib [21]. We follow [38] to enlarge the face regions around the located center by $1.3\times$. We also resize the face images to 299×299 pixels. In total, we have 460 target face images and 693 source face images.

DFDC. DFDC contains videos made by paid actors. Like FaceForensics++, we randomly select two videos from DFDC, each containing face images of one individual; we extract face images from them; and we enlarge the face regions and resize the face images to 299×299 pixels. In total, we have 300 face images of the target and 300 face images of the source.

Trump-Cage. The Trump-Cage dataset contains face images of Donald Trump and Nicolas Cage, which are crawled from the Internet. Specifically, there are 376 face images of Trump (target) and 318 face images of Cage (source).

For each of the three datasets, we use 60% of face images as the training dataset, 20% as the validation dataset, and the remaining 20% as the test dataset.



Figure 4: Examples of JPEG compression with quality factor 10, Gaussian blurring with variance 1.0, cropping with size 0.6, and resizing with ratio 0.6.

Deepfake generation methods: We consider three deepfake generation methods, i.e., *FaceSwap1* [31], *FaceSwap2* [32], and *ICface* [44]. Both *FaceSwap1* and *FaceSwap2* are face replacement methods, while *ICface* is a face reenactment method.

For *FaceSwap1* and *FaceSwap2*, we use the target faces to train the target decoder, the source faces to train the source decoder, and both target and source faces to train the shared encoder. We generate fake faces by feeding the target faces into the encoder and then decoding them using the source decoder. For *ICface*, we randomly select one target face from each dataset. Then for each source face in the dataset, we generate one fake face. Specifically, we extract the expression from the source face and use it to reenact the target face with the pre-trained model published by the authors.

Post-processing operations: In our experiments, we consider 4 commonly used post-processing operations, i.e., **JPEG, Gaussian blurring, cropping, and resizing**. For JPEG, we consider quality factors ranging from 10 to 100 with a step size 10. For Gaussian blurring, we consider variances ranging from 0 to 1.0 with a step size 0.1. For cropping, we consider crop sizes ranging from 0.6 to 1.0 with a step size 0.1. For resizing, we consider resizing ratios ranging from 0.3 to 1.0 with a step size 0.1. We do not consider crop sizes smaller than 0.6 and resizing ratios smaller than 0.3 because we found they degrade the image quality substantially and are not used in practice. Figure 4 shows an example of each post-processing operation. For each watermarked real face image, we apply these post-processing operations with different parameters. The real face images and their post-processed versions are treated as *negative* examples, while the generated fake faces are treated as *positive* examples.

Evaluation metrics: We use Accuracy (ACC), False Positive Rate (FPR), and False Negative Rate (FNR) as our evaluation metrics. ACC is the fraction of test positive and negative examples that are correctly classified. We sample a balanced test set, i.e., the same number of testing positive examples and negative examples, to calculate ACC to avoid the bias introduced by unbalanced test set. FPR is the fraction of negative examples that are falsely detected as positive (i.e., fake). FNR is the fraction of positive examples that are incorrectly classified as negative (i.e., real).

Compared methods: We compare FaceGuard with the following methods.

Passive detector. We consider a state-of-the-art passive detector [38], which trains an Xception network [9] as a detector, i.e., replaces the last classification layer of an Xception net to be a binary classifier. Following [38], we initialize the Xception network with the pre-trained ImageNet weights. We first freeze all the weights except the last layer and train for 3 epochs using the training negative examples (including both real face images and their post-processed versions) and training positive examples generated by *FaceSwap1*. Then, we train the entire network for another 15 epochs. We use the validation dataset to choose the model with the highest validation accuracy during training. We use *FaceSwap1* to train the passive detector to show that its effectiveness degrades for deepfakes generated by other methods not considered during training, following [7].

Conventional watermarking method (FaceGuard-ConvWM). Watermarking is a key component of FaceGuard. Our FaceGuard uses a deep-learning-based watermarking method. FaceGuard-ConvWM is a variant of FaceGuard, in which we replace our deep-learning-based watermarking method as a conventional one (both encoder and decoder). Specifically, we consider a conventional blind watermarking method [36]. In this watermarking method, the watermark is an image. We predict a face claiming as an individual to be fake if the *structural similarity index measure (SSIM)* between the extracted watermark and the individual’s ground-truth one is smaller than a threshold.

Parameter	Value
# epochs	40,000
batch size $ B $	30
length of watermark $ w $	30
initial learning rate η	0.001
loss weights λ_1, λ_2	0.7, 0.001
reward weight β	10
bias vector \mathbf{b} [original, JPEG, Gaussian blurring, cropping, resizing]	[-0.001, 0.001, 0, 0, 0]
initial exploration probability ϵ_0	1.0
exploration probability decay	$2.5 \times 10^{-4}/\text{epoch}$
discount factor γ	0.5
scheduler learning rate α	0.2

Table 1: Parameter settings.

FaceGuard-HiDDeN. In this variant, we replace our deep-learning-based watermarking method as HiDDeN [55] in FaceGuard. We use the implementation from the authors.

FaceGuard-ConvWM (or FaceGuard-HiDDeN or FaceGuard) predicts a face image to be fake if the SSIM (or *bitwise accuracy*) between the extracted watermark and the corresponding ground truth one is smaller than a threshold. Bitwise accuracy is the fraction of matched bits between an extracted watermark and the ground-truth one. We determine the threshold of each method using the negative examples in the validation dataset. We select a threshold for a method so the method can detect as many fake faces as possible while falsely classifying a small fraction of real faces as fake. Specifically, we choose the largest threshold for a method such that the method achieves at most 1% FPR on the validation dataset.

Parameter settings: We use convolutional neural networks as our encoder, decoder, and discriminator. Their architectures are the same as those in HiDDeN and are shown in the Appendix. We set $\lambda_1 = 0.7$, $\lambda_2 = 0.001$, batch size $|B| = 30$, and watermark length $|w| = 30$. We use Adam as optimizer with initial learning rate η as 0.001, and we use CosineAnnealingLR [25] with default parameter settings to adjust the learning rate. For our reinforcement learning based scheduler, we set $\beta = 10$ and $\mathbf{b} = [-0.001, 0.001, 0, 0, 0]$, where the numbers represent the bias for no post-processing, JPEG, Gaussian blurring, cropping, and resizing, respectively. We set $\alpha = 0.2$ and $\gamma = 0.5$. We initialize the Q-table as all zeros. We initialize the exploration probability $\epsilon_0 = 1.0$ and keep it unchanged in the first 8,000 epochs to explore all possibilities. After that, we decay ϵ_t by 2.5×10^{-4} per epoch. We train for 40,000 epochs in total. Table 1 summarizes the parameters.

4.2 Experimental Results

FaceGuard outperforms the compared methods: Table 2 shows the results of different methods. We observe that FaceGuard achieves the highest ACC in all the three datasets. Passive detector has high FNRs for deepfakes generated by methods (i.e., FaceSwap2 and ICface) not considered during training in most cases. FaceGuard-ConvWM achieves low ACC because the ConvWM method is not robust against post-processing. In particular, post-processed face images have small SSIMs between the extracted watermarks and the ground truth ones in ConvWM, and thus fake faces are also classified as real when selecting the threshold that achieves at most 1% FPR on the validation dataset. FaceGuard-HiDDeN achieves smaller ACC because HiDDeN is not robust against JPEG compression. Figure 5 in Appendix shows that HiDDeN achieves much lower bitwise accuracy than FaceGuard for the extracted watermarks when JPEG compression is used. We found that HiDDeN and FaceGuard achieve similar bitwise accuracy for the other three post-processing operations.

Impact of the two key components of FaceGuard: FaceGuard has two key components, i.e., differentiable JPEG approximation and reinforcement learning based scheduler. We evaluate their impact. Table 3 shows the ACC of different variants of FaceGuard on DFDC dataset. FaceGuard w.o. JPEG uses the original JPEG operation instead of our approximation. As a result, the encoders are not updated for the mini-batches with JPEG compression because the gradients are 0. FaceGuard w.o. RL uses the random scheduler instead of our reinforcement learning based one. FaceGuard with both components achieves the highest ACC.

Adaptive deepfakes: An attacker can adapt its deepfakes after FaceGuard is deployed. The encoder and decoder

Method	ACC	FPR	FNR		
			FaceSwap1	FaceSwap2	ICface
Passive Detector	66.6	0.0	0.5	100.0	100.0
FaceGuard-ConvWM	49.4	1.2	100.0	100.0	100.0
FaceGuard-HiDDeN	86.0	1.2	0.0	40.5	40.2
FaceGuard	99.2	1.7	0.0	0.0	0.0

(a) FaceForensics++

Method	ACC	FPR	FNR		
			FaceSwap1	FaceSwap2	ICface
Passive Detector	96.1	0.0	0.0	21.8	1.8
FaceGuard-ConvWM	49.5	1.1	100.0	100.0	100.0
FaceGuard-HiDDeN	72.2	1.3	0.0	72.3	90.7
FaceGuard	99.5	1.1	0.0	0.0	0.0

(b) DFDC

Method	ACC	FPR	FNR		
			FaceSwap1	FaceSwap2	ICface
Passive Detector	92.9	0.0	0.7	0.0	41.7
FaceGuard-ConvWM	49.6	0.8	100.0	100.0	100.0
FaceGuard-HiDDeN	74.0	1.4	10.7	83.9	57.1
FaceGuard	97.7	1.5	1.0	8.1	0.0

(c) Trump-Cage

Table 2: ACC, FPR, and FNR (%) of different methods.

Method	ACC
FaceGuard w.o. JPEG	91.2
FaceGuard w.o. RL	98.2
FaceGuard w.o. RL/JPEG	92.4
FaceGuard	99.5

Table 3: ACC (%) of different variants of FaceGuard.

Dataset	ACC
FaceForensics++	99.2
DFDC	93.9
Trump-Cage	97.6

Table 4: ACC (%) of FaceGuard for adaptive deep-fakes.

Dataset	SSIM
FaceForensics++	0.94
DFDC	0.94
Trump-Cage	0.86

Table 5: Average SSIM between the real face images and their watermarked versions in FaceGuard for each dataset.

parameters as well as watermarks in FaceGuard are secret information and not available to an attacker. However, we assume an attacker has access to the architectures of the encoder and decoder as well as the training face images. Therefore, the attacker can train its own encoder and decoder using our training workflow. The attacker extracts a watermark from a watermarked real face image using its own decoder and embeds the corresponding extracted watermark into its fake face image using its own encoder. Table 4 shows the ACC of FaceGuard against such adaptive deepfakes on the three datasets. We observe that FaceGuard still achieves high ACC against such adaptive deepfakes.

Perceptual quality of watermarked face images: One goal of our watermark is that it should not sacrifice the perceptual quality of the face images. Therefore, we evaluate the perceptual quality of watermarked face images using the widely used SSIM as a metric. Specifically, for each real face image and its watermarked version in FaceGuard, we calculate their SSIM. Table 5 shows the average SSIM for each dataset. The large average SSIMs indicate that the perceptual quality loss introduced by our watermark is small.

5 Conclusion

In this work, we propose FaceGuard, a proactive deepfake detection framework. FaceGuard proactively embeds a watermark into a real face image using a deep-learning-based watermarking method and detects a fake face image if the extracted watermark does not match well with the ground truth one. Moreover, we design a differentiable approximation of JPEG compression and a reinforcement learning based scheduler to jointly train the encoder and decoder in FaceGuard. Our evaluation results show that FaceGuard can effectively detect deepfakes and outperform existing methods. **An interesting direction for future work is to combine passive detection and proactive detection.**

References

- [1] Darius Afchar, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Mesonet: a compact facial video forgery detection network. In *WIFS*, 2018.
- [2] Shruti Agarwal, Hany Farid, Yuming Gu, Mingming He, Koki Nagano, and Hao Li. Protecting world leaders against deep fakes. In *CVPR Workshops*, 2019.
- [3] Mahdi Ahmadi, Alireza Norouzi, Nader Karimi, Shadrokh Samavi, and Ali Emami. Redmark: Framework for residual diffusion watermarking based on deep networks. *Expert Systems with Applications*, 2020.
- [4] Ali Al-Haj. Combined dwt-dct digital image watermarking. *Journal of computer science*, 2007.
- [5] Mauro Barni, Franco Bartolini, Vito Cappellini, and Alessandro Piva. A dct-domain system for robust image watermarking. *Signal processing*, 1998.
- [6] Belhassen Bayar and Matthew C Stamm. A deep learning approach to universal image manipulation detection using a new convolutional layer. In *IH & MM Sec*, 2016.
- [7] Xiaoyu Cao and Neil Zhenqiang Gong. Understanding the security of deepfake detection. In *SecureComm*, 2021.
- [8] Yunje Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *CVPR*, 2018.
- [9] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017.
- [10] Wai C Chu. Dct-based image watermarking using subsampling. *IEEE transactions on multimedia*, 2003.
- [11] Davide Cozzolino, Giovanni Poggi, and Luisa Verdoliva. Recasting residual-based local descriptors as convolutional neural networks: an application to image forgery detection. In *IH & MM Sec*, 2017.
- [12] Frederic Deguillaume, Gabriela Csurka, Joseph JK O’Ruanaidh, and Thierry Pun. Robust 3d dft video watermarking. In *Security and Watermarking of Multimedia Contents*, 1999.
- [13] Brian Dolhansky, Joanna Bitton, Ben Pfau, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge dataset. *arXiv preprint arXiv:2006.07397*, 2020.
- [14] Joel Frank, Thorsten Eisenhofer, Lea Schönherr, Asja Fischer, Dorothea Kolossa, and Thorsten Holz. Leveraging frequency analysis for deep fake image recognition. In *ICML*, 2020.
- [15] Emir Ganic and Ahmet M Eskicioglu. Robust dwt-svd domain image watermarking: embedding data in all frequencies. In *MM&Sec*, 2004.
- [16] Yang He, Ning Yu, Margret Keuper, and Mario Fritz. Beyond the spectrum: Detecting deepfakes via re-synthesis. In *IJCAI*, 2021.
- [17] Ziheng Hu, Hongtao Xie, YuXin Wang, Jiahong Li, Zhongyuan Wang, and Yongdong Zhang. Dynamic inconsistency-aware deepfake video detection. In *IJCAI*, 2021.
- [18] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. In *ICLR*, 2018.

- [19] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019.
- [20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *CVPR*, 2020.
- [21] Davis E. King. Dlib-ml: A machine learning toolkit. *Journal of Machine Learning Research*, 2009.
- [22] Yuezun Li, Ming-Ching Chang, and Siwei Lyu. In icu oculi: Exposing ai generated fake face videos by detecting eye blinking. *arXiv preprint arXiv:1806.02877*, 2018.
- [23] Yuezun Li and Siwei Lyu. Exposing deepfake videos by detecting face warping artifacts. In *CVPRW*, 2019.
- [24] Yang Liu, Mengxi Guo, Jian Zhang, Yuesheng Zhu, and Xiaodong Xie. A novel two-stage separable deep learning framework for practical blind watermarking. In *ACMMM*, 2019.
- [25] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. In *ICLR*, 2017.
- [26] Xiyang Luo, Ruohan Zhan, Huiwen Chang, Feng Yang, and Peyman Milanfar. Distortion agnostic deep watermarking. In *CVPR*, 2020.
- [27] Falko Matern, Christian Riess, and Marc Stamminger. Exploiting visual artifacts to expose deepfakes and face manipulations. In *WACVW*, 2019.
- [28] Huy H Nguyen, Fuming Fang, Junichi Yamagishi, and Isao Echizen. Multi-task learning for detecting and segmenting manipulated facial images and videos. In *BTAS*, 2019.
- [29] Huy H Nguyen, Junichi Yamagishi, and Isao Echizen. Use of a capsule network to detect fake images and videos. *arXiv preprint arXiv:1910.12467*, 2019.
- [30] Yuval Nirkin, Yosi Keller, and Tal Hassner. Fsgan: Subject agnostic face swapping and reenactment. In *ICCV*, 2019.
- [31] Online. Faceswap1. <https://github.com/deepfakes/faceswap>, Last accessed August 2021.
- [32] Online. Faceswap2. https://github.com/joshua-wu/deepfakes_faceswap, Last accessed August 2021.
- [33] Online. Trump-cage dataset. https://github.com/joshua-wu/deepfakes_faceswap, Last accessed August 2021.
- [34] Online. Trump-cage example. <https://www.youtube.com/watch?v=l3E7rOt9--g>, Last accessed August 2021.
- [35] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.
- [36] Md. Maklachur Rahman. **A dwt, dct and svd based watermarking technique to protect the image piracy.** *International Journal of Managing Public Sector Information and Communication Technologies*, 2013.
- [37] Nicolas Rahmouni, Vincent Nozick, Junichi Yamagishi, and Isao Echizen. Distinguishing computer graphics from natural images using convolution neural networks. In *WIFS*, 2017.
- [38] Andreas Rössler, Davide Cozzolino, Luisa Verdoliva, Christian Riess, Justus Thies, and Matthias Nießner. Face-Forensics++: Learning to detect manipulated facial images. In *ICCV*, 2019.
- [39] JJKO Ruanaidh, WJ Dowling, and Francis M Boland. Phase watermarking of digital images. In *ICIP*, 1996.
- [40] Nishant Subramani and Delip Rao. Learning efficient representations for fake speech detection. In *AAAI*, 2020.
- [41] Justus Thies, Michael Zollhöfer, and Matthias Nießner. Deferred neural rendering: Image synthesis using neural textures. *ACM Transactions on Graphics (TOG)*, 2019.
- [42] Justus Thies, Michael Zollhofer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *CVPR*, 2016.

- [43] Anatol Z Tirkel, GA Rankin, RM Van Schyndel, WJ Ho, NRA Mee, and Charles F Osborne. Electronic watermark. *Digital Image Computing, Technology and Applications (DICTA'93)*, 1993.
- [44] Soumya Tripathy, Juho Kannala, and Esa Rahtu. Icfac: Interpretable and controllable face reenactment using gans. *arXiv preprint arXiv:1904.01909*, 2019.
- [45] Ron G Van Schyndel, Andrew Z Tirkel, and Charles F Osborne. A digital watermark. In *ICIP*, 1994.
- [46] Run Wang, Felix Juefei-Xu, Meng Luo, Yang Liu, and Lina Wang. Faketagger: Robust safeguards against deepfake dissemination via provenance tracking. In *ACMMM*, 2021.
- [47] Run Wang, Felix Juefei-Xu, Lei Ma, Xiaofei Xie, Yihao Huang, Jian Wang, and Yang Liu. Fakespotter: A simple yet robust baseline for spotting ai-synthesized fake faces. In *IJCAI*, 2020.
- [48] Sheng-Yu Wang, Oliver Wang, Richard Zhang, Andrew Owens, and Alexei A Efros. Cnn-generated images are surprisingly easy to spot...for now. In *CVPR*, 2020.
- [49] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.
- [50] Xin Yang, Yuezun Li, and Siwei Lyu. Exposing deep fakes using inconsistent head poses. In *ICASSP*, 2019.
- [51] Egor Zakharov, Aliaksandra Shysheya, Egor Burkov, and Victor Lempitsky. Few-shot adversarial learning of realistic neural talking head models. In *ICCV*, 2019.
- [52] Chaoning Zhang, Philipp Benz, Adil Karjauv, Geng Sun, and In So Kweon. Udh: Universal deep hiding for steganography, watermarking, and light field messaging. In *NeurIPS*, 2020.
- [53] Daichi Zhang, Chenyu Li, Fanzhao Lin, Dan Zeng, and Shiming Ge. Detecting deepfake videos with temporal dropout 3dcnn. In *IJCAI*, 2021.
- [54] Peng Zhou, Xintong Han, Vlad I Morariu, and Larry S Davis. Two-stream neural networks for tampered face detection. In *CVPRW*, 2017.
- [55] Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *ECCV*, 2018.
- [56] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017.

Appendix

<i>Layer</i>	<i>Type</i>	<i># Input channel</i>	<i># Output channel</i>	<i>Kernel size</i>	<i>Stride</i>	<i>Padding</i>	<i>With BN</i>	<i>Activation</i>
1	Conv.	3	64	(3,3)	(1,1)	(1,1)	True	ReLU
2	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
3	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
4	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
5	Conv.	97	64	(3,3)	(1,1)	(1,1)	True	ReLU
6	Conv.	64	3	(1,1)	(1,1)	None	False	None

Table 6: Model architecture of the encoder.

Layer	Type	Input size	Output size	Kernel size	Stride	Padding	With BN	Activation
1	Conv.	3	64	(3,3)	(1,1)	(1,1)	True	ReLU
2	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
3	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
4	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
5	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
6	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
7	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
8	Conv.	64	30	(3,3)	(1,1)	(1,1)	True	ReLU
9	GAP	30	30	—	—	—	—	—
10	Flatten	—	—	—	—	—	—	—
11	FC	30	30	—	—	—	False	None

Table 7: Model architecture of the decoder. Input/output size indicates the number of channels for convolution layers and the global average pooling (GAP) layer, while indicating the number of neurons for the fully-connected (FC) layer.

Layer	Type	Input size	Output size	Kernel size	Stride	Padding	With BN	Activation
1	Conv.	3	64	(3,3)	(1,1)	(1,1)	True	ReLU
2	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
3	Conv.	64	64	(3,3)	(1,1)	(1,1)	True	ReLU
4	GAP	64	64	—	—	—	—	—
5	Flatten	—	—	—	—	—	—	—
6	FC	64	1	—	—	—	False	None

Table 8: Model architecture of the discriminator. Input/output size indicates the number of channels for convolution layers and the global average pooling (GAP) layer, while indicating the number of neurons for the fully-connected (FC) layer.

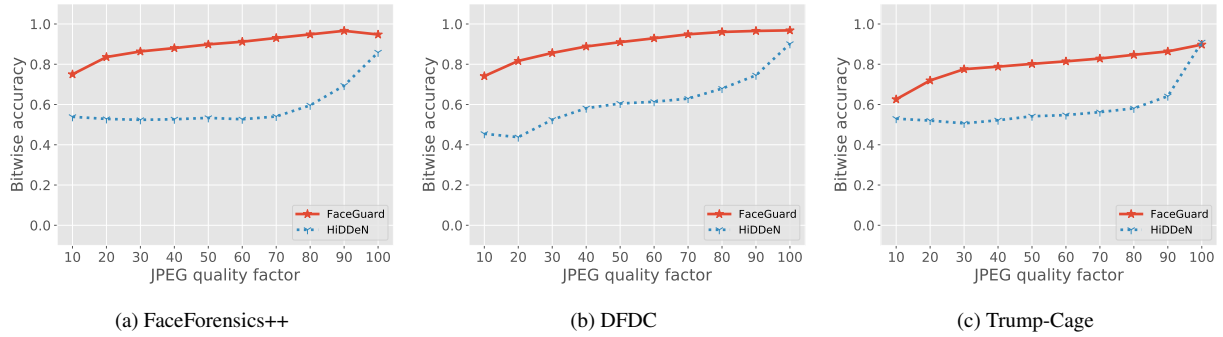


Figure 5: Bitwise accuracy of the extracted watermarks for the real face images post-processed by JPEG compression with different quality factors on the three datasets.