Multimedia Tools and Applications (2022) 81:31219–31243
https://doi.org/10.1007/s11042-022-12880-6

Securing content-based image retrieval on the cloud using generative models

Yong Wang[1] · Fan-chuan Wang[1] · Fei Liu[1] · Xiao-hu Wang[1]

Received: 17 March 2021 / Revised: 8 February 2022 / Accepted: 10 March 2022 /
Published online: 8 April 2022

Abstract
Content-based image retrieval (CBIR) with deep neural networks (DNNs) on the cloud has tremendous business and technical advantages to handle large-scale image repositories. However, cloud-based CBIR service raises challenges in image data and DNN model security. Typically, users who wish to request CBIR services on the cloud require their input images remaining confidential. On the other hand, image owners may intentionally (or unintentionally) upload adversarial examples to the cloud servers, which potentially leads to the misbehavior of CBIR services. Generative Adversarial Networks (GANs) can be utilized to defense against such malicious behavior. However, the GANs model, if not well protected, can be easily abused by the cloud to reconstruct the users' original image data. In this paper, we focus on the problem of secure generative model evaluation and secure gradient descent (GD) computation in GANs. We propose two secure generative model evaluation algorithms and two secure minimizer protocols. Furthermore, we propose and implement Sec-Defense-Gan, a secure image reconstruction framework which can keep the image data, the generative model details and corresponding outputs confidential from the cloud. Finally, We carried out a set of benchmarks over two public available image datasets to show the performance and correctness of Sec-Defense-Gan.

Keywords Content-based image retrieval · Generative adversarial networks (GANs) · Lattice-based homomorphic scheme · secure multiparty computation · Deep neural networks (DNNs)

1 Introduction

Deep neural networks (DNNs) have shown unprecedented superiority for visual object recognition, which may be the most powerful approach for Content-Based Image Retrieval

This work is supported by the National Key Research and Development Program of China (Grant No. 2018YFB0804702)

✉ Yong Wang
cla@uestc.edu.cn

[1] School of Computer Science and Engineering, Center for Cyber Security, University of Electronic Science and Technology of China, Chengdu, China

(CBIR) in recent years [2]. The representative works include robust face detection [3, 21], face mask detection in the environment of COVID-19 [28] and accurate low resolution image classification [4]. With the improvement of these DNN models, CBIR as a DNN-based application is becoming popular in the area of cloud computing. It is a cost-effective way that mobile users outsource their images onto cloud servers which can generate image searching results more intelligently with the help of the modern DNN-based features.

Despite the fact that the cloud-based CBIR has tremendous business and technical advantages to handle large-scale image repositories, it raises users' privacy concerns because they require users to upload and store images onto the cloud servers in most application scenarios. Furthermore, the quality of CBIR services is seriously affected by the well-trained DNN network itself, which has shown its vulnerability to adversarial samples [29]: carefully crafted perturbations added to a legitimate input sample. These disturbances will cause the legal samples to be misclassified even disable the DNN based CBIR services. In reality, users may intentionally (or unintentionally) upload adversarial examples to the cloud servers, which potentially leads to the misbehavior of CBIR services.

Generative Adversarial Networks (GANs) [14] have the powerful expression ability to learn data distribution, which can be widely used to generate adversarial samples [30] or to defense against malicious inputs [27]. Recently, Pouya Samangouei et al.[27] proposed Defense-GAN which is effective against both *white box* and *black box* attacks. Defense-GAN leverages the representative power of GAN to diminish the effect of the adversarial perturbation, by projecting input images onto the range of the GAN's generator prior to feeding them to the classifier. However, the generative model in a GAN framework needs to emulate the data distribution, whose outputs, typically generated images, are obviously sensitive to users' privacy. At the same time, the parameters and intermediate results of a GAN framework are very valuable because of the difficulty of training it. Hence, these data also require to be protected from cloud servers to avoid being commercially exploited. Finally, a GAN framework uses Gradient Descent (GD) to select the appropriate outputs. As was reported in [25, 31], the gradient information can be easily utilized to issue passive and active *white box* inference attacks against deep learning algorithms. Typically, FGSM black-box attack has shown its powerful impacts on DNN algorithms. As a result, the gradient information has to be kept confidential. Theoretically, Fully Homomorphic Encryption (FHE) techniques, such as Lattice-based schemes, can potentially handle such issue. However, it is not efficient because of its design complexity of the operations in a GAN framework. Specifically, it is time-consuming and laborious to support division operations with FHE schemes, which is the core of Gradient Descent algorithms. Secure Multi-party Computation (SMC) can be an alternative solution, but the communication overhead between users and the cloud server will dramatically increase when the DNN layers or its inputs become large.

The contributions of this work are mainly summarized as follows:

– We focus on the problem of secure generative model evaluation and secure gradient descent (GD) computation in GANs. We design two secure de-convolution algorithms to deal with large inputs based on direct convolution and matrix-vector product methods respectively. Furthermore, we design two secure gradient computation and update protocols to accomplish image reconstruction task based on numerical differentiation and chain rule respectively.
– We propose Sec-Defense-Gan, a secure image reconstruction framework, which can diminish the effect of the adversarial perturbation by reconstruction adversarial examples uploaded from the image data owner without leaking any information

to the cloud. Sec-Defense-Gan consists of two major components, i.e., *SecGan* and *SecMinimizer*. *SecGan* provides a secure generative model evaluation functionality while *SecMinimizer* performs secure gradient update.

– We apply Sec-Defense-Gan into secure content-based image retrieval system which assumes that the image data owner may be adversary. The system can protect the image data, the GANs model details (e.g., weight matrix) and the image retrieval results from the cloud, while defending against the adversarial examples which have a different distribution of the training samples used by the NN model.

The rest of the article is organized as follows. We review the related works in Section 2. Section 3 explains the preliminaries. In Sections 4, 5 and 6, we described our framework and implementations in detail. The experimental evaluations are shown in Section 7. Finally, we discuss and draw brief conclusions in Section 8.

2 Related work

Content-based image retrieval (CBIR) has been studied for several decades. CBIR technologies are typically of high computational complexity and storage-consuming because images need to be indexed by their visual contents (features). Hence, it is an inevitable choice to store users' images onto the cloud servers which can further provide CBIR services. Although outsourcing CBIR services to the cloud has great business advantages, privacy concerns arises because users default that the cloud service providers are not trustworthy.

Homomorphic encryption (HE) based techniques can be apply to secure CBIR service. In such schemes, users encrypt images pixel by pixel by utilizing a homomorphic cryptosystem (e.g., Paillier[26], ElGamal[10], or Lattice-based AHE[8]), which allows the cloud to index and process their images in the encrypted domain. Hsu et al.[18] proposed a high-precision CBIR algorithm by adopting Paillier cryptosystem to encrypt private images. This approach is suffered from significative ciphertext expansion, which leads to slow encryption and decryption time and scalability issues. Hu et al. [19] further proposed an efficient scheme for SIFT feature extraction by utilizing the ring-Learn-With-Error (r-LWE) homomorphic cryptosystem[6], different from their previous scheme proposed in [18], their batched secure multiplication protocol is built on Some-What Homomorphic Encryption (SWHE) scheme that enables the two parties to securely compute the products of multiple pairs of private integers simultaneously, with computation and communication costs greatly reduced. A variant work proposed by Zheng et al.[33] replaced Paillier ciphertexts with pointers to a ciphertext table. It reduced the number of encryption operations and minimized ciphertext expansion. Li et al.[22] proposed a double-decryption SIFT feature extraction scheme based on the BCP cryptosystem, which is an additively homomorphic scheme with two independent decryption algorithms.

Although HE-based schemes allow the cloud server to process and index their encrypted images, which is semantically secure. Unfortunately, they present much higher time and space complexity[11]. More importantly, these schemes naturally are facing with ciphertext expansion and noise growth problems[5, 7–9, 13, 24]. These have potentially negative effects on the scalability and accuracy. For example, schemes in [19, 22, 33] can only deal with the integer values of SIFT vectors and accept limited additive homomorphic operations. It is hardly applicable when considering CBIR with deep features, such as features extracted by convolutional neural networks (CNNs), because these schemes perform very poorly due to the large multiplicative depth in a CNN. The last but not the least, these schemes naturally assume that image owners are honest. It requires design of HE based de-convolution operations when apply GAN to defense against such malicious users.

In this work, we consider the secure reconstruction of the generative model in a GAN framework by utilizing the homomorphic encryption in conjunction with multi-party computation.

3 Preliminaries

3.1 Generative adversarial networks (GANS)

GANs include two neural networks, generative model G and discriminative model D. $G : \mathbb{R}^k \mapsto \mathbb{R}^n$ maps a low-dimensional latent space to high-dimensional sample space of x. D is a binary neural network classifier. In the training phase, G and D are typically learned in an adversarial fashion using actual input data samples x and random vectors z. An isotropic Gaussian prior is usually assumed on z. While G learns to generate outputs $G(z)$ that have a distribution similar to that of x, D learns to discriminate between "real" samples x and "fake" samples $G(z)$. D and G are trained in an alternating fashion to minimize the following $min - max$ loss:

$$\begin{aligned}\min_G \quad \max_D V(D, G) = \\ \mathbb{E}_{x\sim\rho_{data}(x)}[\log(D(x))] + \mathbb{E}_{z\sim\rho_z(z)}[\log(1 - D(G(z)))].\end{aligned} \tag{1}$$

It was shown that the optimal GAN is obtained when the resulting generator distribution $\rho_g = \rho_{data}(x)$ [14]. Because GANs are difficult to train in practice [16], many alternative formulations have been proposed. Wasserstein GANs (WGANs) [16] have shown the stability of their training methods, which use the Wasserstein distance in a loss function:

$$\begin{aligned}\min_G \quad \max_D V_W(D, G) = \\ \mathbb{E}_{x\sim\rho_{data}(x)}[D(x)] - \mathbb{E}_{z\sim\rho_z(z)}[D(G(z))].\end{aligned} \tag{2}$$

One of WGANs application scenario is to use a WGAN trained on legitimate training samples to defense against adversarial samples.

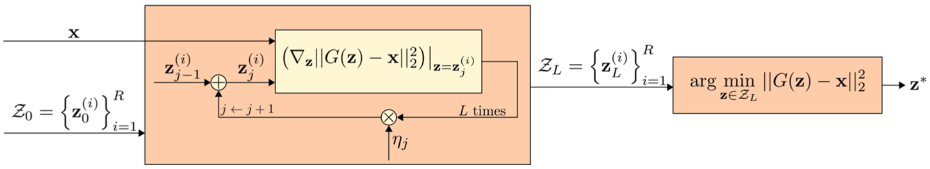
3.2 Defense-GAN algorithm

Defense-Gan is a new framework proposed by samangouei [27] for the purpose of defending against adversarial attacks in classification problems. Defense-Gan employs a novel defense strategy which use a WGAN trained on legitimate (un-perturbed) training samples to "denoise" adversarial examples. At inference time, given a trained GAN generator G and an image x to be classified, z^* is first found to minimize

$$\min_z \|G(z) - x\|_2^2 \tag{3}$$

This non-convex minimization problem is approximated by performing a fixed number of L Gradient Descent (GD) steps using R different random initialization of z. Figure 1 further shows the process of L Gradient Descent (GD) steps in detail:

$G(z^*)$ is then given as an input to the CBIR system. The overview of this application scenario is illustrated in Fig. 2.



$\mathbf{x}$
$\mathcal{Z}_0 = \left\{\mathbf{z}_0^{(i)}\right\}_{i=1}^{R}$
$\mathbf{z}_{j-1}^{(i)}$
$\mathbf{z}_j^{(i)}$
$\left(\nabla_{\mathbf{z}}\|G(\mathbf{z}) - \mathbf{x}\|_2^2\right)\big|_{\mathbf{z}=\mathbf{z}_j^{(i)}}$
$j \leftarrow j+1$
L times
η_j
$\mathcal{Z}_L = \left\{\mathbf{z}_L^{(i)}\right\}_{i=1}^{R}$
$\arg\min_{\mathbf{z}\in\mathcal{Z}_L} \|G(\mathbf{z}) - \mathbf{x}\|_2^2$
$\mathbf{z}^*$

Fig. 1 L steps of Gradient Descent are used to estimate the projection of the image onto the range of the generator. Sign G denotes the generator network and $\mathcal{Z}_0$ denotes a group of random vectors. Given an image x, $\mathcal{Z}_0$ is updated by minimizing the loss iteratively and get the result $\mathcal{Z}_L$. Finally, z^* is selected from the vector group $\mathcal{Z}_L$ by executing the function $argmin$

3.3 Packed additively homomorphic encryption (PAHE)

A general abstraction of packed additively homomorphic encryption (PAHE) schemes supports packing multiple plaintexts into a single ciphertext, performing SIMD homomorphic additions (SIMDAdd), scalar multiplications (SIMDScMult) and permuting the plaintext slots (Perm).

- **SIMDAdd**: Given ciphertexts $[u]$ and $[v]$, SIMDAdd outputs an encryption of their componentwise sum, namely $[u + v]$.
- **SIMDScMult**: Given a ciphertext $[u]$ and a plaintext v, we can output an encryption $[u \circ v]$ (where $\circ$ denotes component-wise multiplication of vectors).
- **Perm**: Given a ciphertext $[u]$ and one of a set of primitive permutations Π defined by the scheme, the Perm operation outputs a ciphertext $[u_\Pi]$, where $[u_\Pi]$ is defined as $([u_{\Pi(1)}], [u_{\Pi(2)}], \ldots, [u_{\Pi(n)}])$, namely the vector u whose slots are permuted according to the permutation Π.

PAHE includes an encryption algorithm, a deterministic decryption algorithm, and a homomorphic evaluation algorithm. The encryption algorithm takes a plaintext message vector $\vec{u}$ from some message space and encrypts it using a private key s_k into a ciphertext denoted as $[\vec{u}]$. The decryption algorithm takes the ciphertext $[\vec{u}]$ and the key s_k and recovers the message vector $\vec{u}$. The homomorphic evaluation algorithm takes as input one or more ciphertexts that encrypt messages $\vec{u}_0, \vec{u}_1, \cdots$, and outputs another ciphertext that encrypts a message $\vec{u} = f(\vec{u}_0, \vec{u}_1, \cdots)$ for some function f. The function f is constructed by using the three basic homomorphic operations: Single Instrument Multiple Data (SIMD) addition

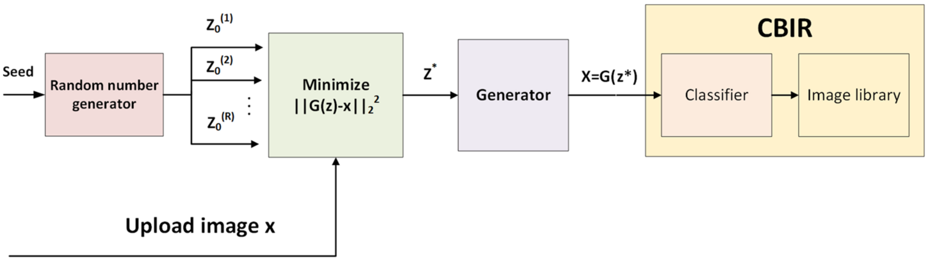


Fig. 2 An application of DEFENSE-GAN in CBIR system. The GAN is trained on the available classifier training dataset in an unsupervised manner. The classifier can be trained on the original training images, their reconstructions $G(z^*)$ using the generator G. As long as the GAN is appropriately trained and has enough capacity to represent the data, original clean images and their reconstructions should not defer much

(*SIMDAdd*), SIMD scalar multiplication (*SIMDScMult*), and permuting the plaintext slots (*Perm*).

Generally, a PAHE schema is parameterized by four constants that are the cyclotomic order m, the ciphertext modulus q, the plaintext modulus p, and the standard deviation σ of a symmetric discrete Gaussian noise distribution (χ). It satisfies: (1) IND-CPA security, which requires that ciphertexts of any two messages $\vec{u}$ and $\vec{u}'$ be computationally indistinguishable; and (2) Function Privacy, which requires that the ciphertext generated by homomorphic evaluation, together with the private key sk, reveals the underlying message, namely the output $f(\cdot)$, but does not reveal any other information about the function f.

For any a linear layer in this paper, which is either $Deconv$ or FC layer, it can be evaluated by homomorphic matrix-vector multiplication and homomorphic convolution. The PAHE scheme ensures that nothing about the linear layer's inputs or outputs will be leaked.

3.4 Secure 2-PC computation

Secure two-party computation (2-PC) is a type of protocols that allow two parties to jointly compute a function $(f_1(x, y), f_2(x, y)) \leftarrow f(x, y)$ without learning each other's input. The Obliv-C [32] is a robust 2-PC garbled circuits implementation. It provides an extensible programming tool for secure computation by exposing the important aspects of data-oblivious computation, while providing a high-level language and the ability to seamlessly integrate with standard C code. Programs typically read, process, and output data in native C code, performing only the secure computation in Obliv-C code.

The main language addition is an *obliv* qualifier, applied to C types and constructs. Typing rules enforce that *obliv* types remain secret unless explicitly revealed. Code within oblivious functions and conditionals cannot modify public data, except within a qualified *obliv* block, in which the code is always executed. These rules allow programmers to reason about data security and develop modular libraries. Recent evaluation of Obliv-C [17] indicates its usability on a range of criteria, including language expressiveness, capabilities of the cryptographic back-end, and accessibility to developers.

For any a non-linear layer in this paper, which typically is $ReLU$ or $Sigmod$, it can securely be calculated by the secure 2-PC computation scheme.

4 The secure GAN reconstruction framework

4.1 Framework overview

As is illustrated in Fig. 3, there are three entities in Sec-Defense-Gan, i.e., the cloud servers (S_1 and S_2), the image owner, and the model provider.

A model provider provides a pre-trained WGAN model to the cloud servers, who treat the pre-trained WGAN model details (e.g., weight matrix) as sensitive and hopes it to be protected from the cloud servers. We assume the model provider is honest because it has no motivation to provide a "fake" pre-trained model. Technically, this assumption could be easily guaranteed by the access control and authorization mechanism.

An image owner generates or collects images and outsources the image data onto the cloud servers. We assume that the image data are sensitive and needs to be protected from the cloud servers. We also assume that the image owner can potentially be malicious, who may intentionally (or unintentionally) upload adversarial examples to the cloud servers.

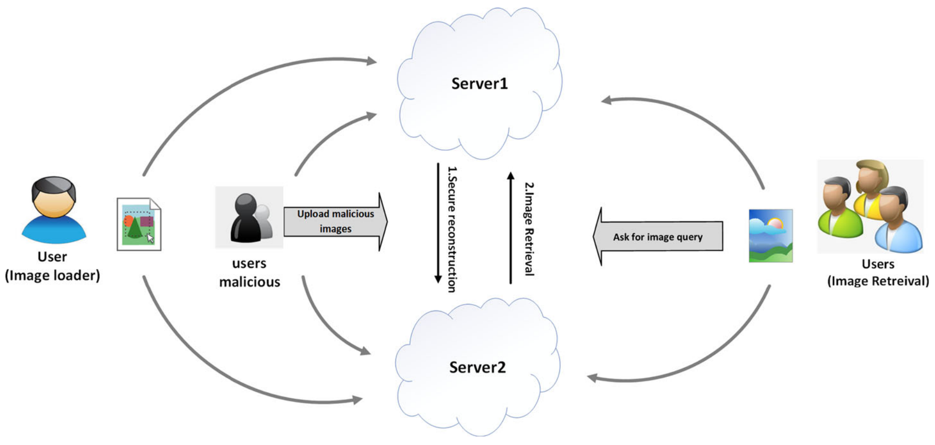


Fig. 3 The overview of Sec-Defense-Gan

These adversarial examples are carefully crafted perturbations added to a legitimate input samples which can cause the legitimate samples to be misclassified at inference time.

The cloud servers (S_1 and S_2) filter out adversarial examples uploaded from the image owners by running the pre-trained WGAN model. We assume that the cloud servers follow the semi-honest adversary model, i.e., they follow the protocol specifications and algorithms exactly, but may attempt to learn additional information by analyzing intermediate computations. In general, secure protocols under the semi-honest model are more efficient than those under the malicious adversary model, and most of the practical SMC protocols are secure under the semi-honest model. We refer the readers to [12] for more details about the security definitions and models. By the semi-honest model, we implicitly assume that the cloud servers do not collude. This model is realistic in the current cloud service market, because S_1 and S_2 could be cloud servers which are provided by legitimate, well-known companies (e.g., Amazon, Google, and Microsoft), collusion between any of them is highly unlikely.

Our system goal is to protect the image data, the outsourced WGAN model details (e.g., weight matrix), and the outputs of the WGAN model from the cloud servers S_1 and S_2, while defending against the adversarial examples which have a different distribution of the training samples used by the classifier.

An image owner splits an image I into two additive shares I_a and I_b, w.r.t $I_a = I + I_b$, which are then stored onto S_1 and S_2 respectively. In addition, S_2 holds the pre-trained WGAN model to reduce adversarial noise of the uploaded images. The WGAN model is outsourced by the model provider after the model parameters (e.g., weight matrix) are encrypted with sk. S_2 carries out the homomorphic operations of the linear layers of the WGAN model. S_1 keeps its private key s_k and runs Yao's GC protocols with S_2 to perform the secure computations of the non-linear layers. S_1 and S_2 holds a share, $\{G(z^*)_a, G(z^*)_b\}$, of the final outputs of the WGAN model, which can be treated as the inputs of the classifier (e.g., VGG-16) for the image I.

4.2 The functionalities of sec-defense-Gan

As is illustrated in Fig. 4, the primary functionalities of our system consists two components: Secure **SecGan** and **SecMinimizer**

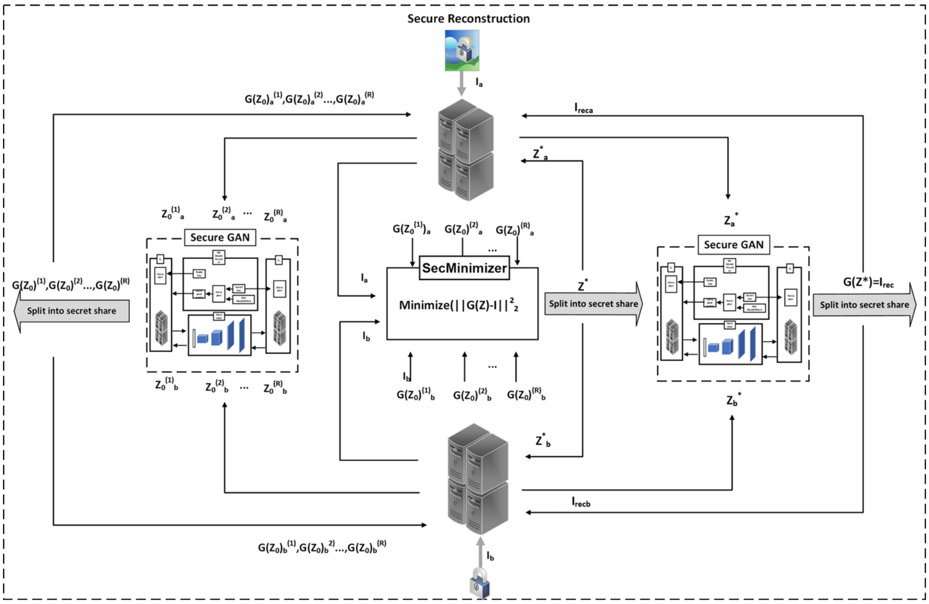


Fig. 4 The functionalities of Sec-Defense-Gan. Given an addition share image (I_a, I_b), servers S_1 and S_2 securely execute image reconstruction using L steps of Gradient Descent, in which $SecGan$ executes the generator forward propagation algorithm and the $SecMinimizer$ executes the updating of Z. The output of Sec-Defense-Gan is the reconstructed image I_{rec}

- $\{G(z)_a, G(z)_b\} \leftarrow SecGan([z])$. Given encrypted vectors $[z]$, it returns a pair of set $G(z)_a, G(z)_b$ by cooperatively performing the secure evaluation of a GAN model, where $G(z)_a$ is held by S_1 and $G(z)_b$ is kept by S_2. For the outputs of the WGAN $G(z)$ that have a distribution similar to that of I, we have $G(z) = G(z)_a - G(z)_b$.
- $z^* \leftarrow SecMinimizer(\{G(z)_a, G(z)_b\}, \{I_a, I_b\})$. Given a pair of set $\{G(z)_a, G(z)_b\}$ from $SecGan([z])$ and the additive share form $\{I_a, I_b\}$ of image I, it finds z^* by cooperatively running a fixed number L of Gradient Descent (GD) steps.

5 The implementation of SecGAN

The generative model used in GANs includes four layers (Fig. 5).

- $FC(m)$ is a fully-connected layer with m outputs. Its input is a 128-dimensional vector z.
- $DeConv(m, k \times k, s)$ is deconvolution with m feature maps, filter size $k \times k$, and stride s.
- $Relu$ is the Rectified Linear Unit activation function.
- $Sigmod$ is the *Smooth activation function*.

The layers used in GANs can be classified into two types: linear layers (e.g., FC and $DeConv$) and non-linear layers (e.g., $Relu$ and $Sigmod$). There are many works that proposed 2-PC secure computation of non-linear layer functions [20]. As for the linear layers, Homomorphic based solutions can achieve good performance. However, considering

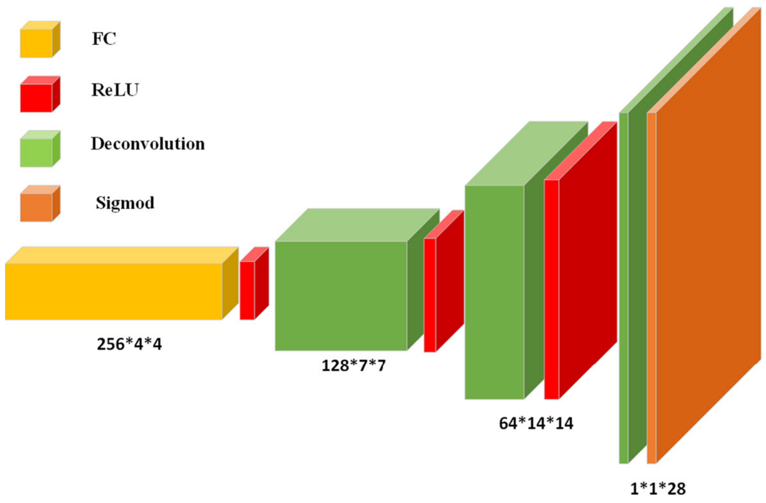


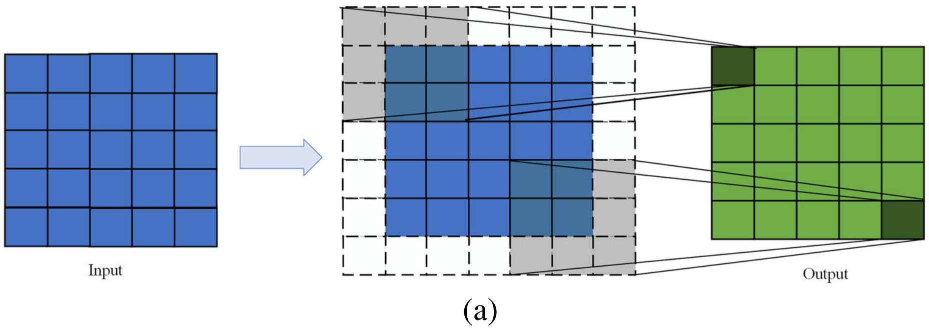
Fig. 5 The generative model in GANs consists of an FC layer, three Relu layers, three De-convolution layers and a Sigmod layer

the de-convolution, a practical solution is needed to achieve both the security and efficiency. On the other hand, we try to convert operations in both linear layers (e.g., FC and $DeConv$) into the same homomorphic operations to deal with linear layers in one shot.

The de-convolution in the $Deconv$ layer is the transpose of convolution, which returns a feature map that has the same shape as the training samples. We can implement it by utilizing Lattice-based PAHE scheme. There are two methods to implement the de-convolution, i.e., direction convolution and matrix-vector product. The remaining problem is to pad the inputs for direct convolution, or convert the $Deconv$ to matrix multiplication and addition operations. For ease of description, the $Deconv$ layer can be defined by 5 parameters: s is used to denote the stride, $k = k_w = k_h$ denotes the filter size (width and height), $i = i_w = i_h$ denotes the input size, $pa = pa_w = pa_h$ denotes the padding size and $o = o_w = o_h$ denotes the output size.

The direct convolution can be thought of as dilating the input (by adding $s - 1$ zeros between adjacent input elements), padding it with k $-$ 1 zeros, and cross-correlating it with the filter. For convolution, we use half (SAME) padding (padding size pa' is $\lfloor k/2 \rfloor$) and unit strides (s' is 1). That is, the output size is the same with the input size of the convolution. For $s = 1, k = 2n+1, n \in \mathbb{N}$, we have $pa = pa'$, its output size is $o = i-(k-1)-2p = i$. Taking Fig. 6(a) as an example, let the input size $i = 5$, $k = 3$ and $s = 1$, we have $pa = 1$, the output size is $o = i = 5$, i.e, we only need to pad the original input (blue entries) with zeros (white entries). As for $s > 1$, the padding size is $pa = k - pa' - 1$ and the output size is $o = s \cdot (i - 1) + k - 2 \cdot pa'$. Taking Fig. 6(b) as another example, let the input size $i = 3$, $k = 3$, $s = 2$ and $pa = 1$, the output size is $o = 5$, it is equivalent to convolute 3×3 filter over a 3×3 input (with $s - 1 = 1$ zero inserted between inputs) padded with a 1×1 border of zeros using unit strides.

The disadvantage of direct convolution is its inefficiency. It will increase addition and multiplication operations in the ciphertext when padding zeros to the input. Let's consider convolution process again, for any input size $i = i_h = i_w$, filter size $k = k_h = k_w$, stride



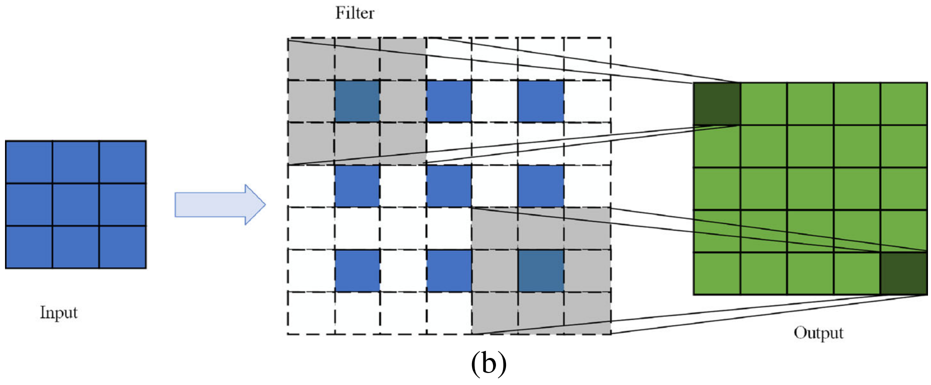


Fig. 6 Examples of direct convolution. (a) denotes the size invariant convolution. (b) denotes the convolution with different in/output shapes

s, and output size $o = o_h = o_w$. We have $o = \frac{i+2p-k}{s} + 1$, $pa_h = s \times (o_h - 1) + k_h - i_h$ and $pa_w = s \times (o_w - 1) + k_w - i_w$, where pa_h and pa_w is the height and the width of the padding size respectively. Hence, we have

$$\begin{aligned} pa_{top} &= pa_h/2 \\ pa_{down} &= pa_h - pa_{top} \\ pa_{left} &= pa_w/2 \\ pa_{right} &= pa_w - pa_{left} \end{aligned} \tag{4}$$

where pa_{top}, pa_{down}, pa_{left} and pa_{right} are the number of elements required in the four direction of the input matrix: up, down, left and right.

We use matrix-vector product to represent convolution process. First, we initialize a $(o_h * o_w) \times (i_h * i_w)$ sparse matrix C with zeros. Then, we use a matrix with the same shape as the input, where each value is set to the index number of the input elements, i.e., 1, 2, ..., $i_h \times i_w$. We pad the matrix with zeros according to the (4). We slide the filter with stride s on the padded matrix. Each time that the filter slides, we record the $k_h \times k_w$ filter window. Finally, we fill each value of the filter into the element with non-zero index number in the corresponding sparse matrix C.

Consider a de-convolution with $i = 4$, $k = 3$ and $s = 3$, according to the (4), we have $pa_{top} = 1$, $pa_{down} = 1$, $pa_{left} = 1$ and $pa_{right} = 1$. Let's take the first slide as an example in Fig. 7, we record the 3×3 filter window, and the non-zero positions are 1, 2, 5 and 6. Hence, we fill each value of the filter (i.e., $w_{1,1}$, $w_{1,2}$, $w_{2,1}$ and $w_{2,2}$) into them. Similarly,

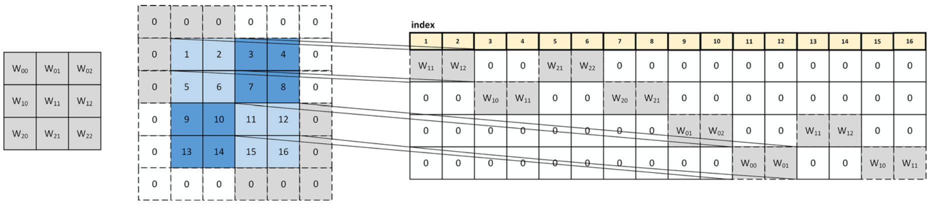


Fig. 7 An example of matrix-vector product with the sparse matrix. Filter size is 3 × 3, sparse matrix size is 4 × 16, and we set $i = 4, k = 3, s = 3$

we can fill the non-zero positions in the sparse matrix by sliding the filter window with stride $s = 3$.

Single channel de-convolution. The sparse matrix C can be derived directly. Figure 8 shows an example of single channel de-convolution that is parameterized as $i = 2$, $k = 2$, $s = 1$, $pa = 0$ and $o = 3$. Considering the corresponding forward convolution process, its input size is $i' = 3$ and its output size is $o' = 2$. The sparse matrix C is of size 4 × 9. We flatten the input X into a 4-dimensional vector. Hence, the de-convolution can be easily computed by linear operation that is the matrix-vector product $Y = C^T X$, where C^T is the transpose of C.

Multiple channel de-convolution. The sparse matrix C can be derived by matrix splicing. We use $c_o \times c_i \times k_h \times k_w$ denotes the filter size and $c_i \times i_h \times i_w$ denotes the input. Figure 9 shows an example of multiple channel de-convolution with $c_i = 3$, $c_o = 2$, $i = 2$, $k = 2$, $s = 1$ and $pa = 0$. The input X is of size 3 × 2 × 2, the filter is of size 2 × 3 × 2 × 2 and the output Y is of size 2 × 3 × 3. Considering the corresponding forward convolution process, its input is of size 2 × 3 × 3 and its output is of size 3 × 2 × 2. Its filter is of size 3 × 2 × 2 × 2 which is obtained by reversing input and output channels of de-convolution, i.e., $c_o \times c_i \times k_h \times k_w \rightarrow c_i \times c_o \times k_h \times k_w$. For each output channel corresponding to c_i filters, we have c_i sparse matrices. We horizontally join these c_i sparse matrices and vertically join the c_o output channels. Hence, we obtain the sparse matrix C with the size of $n_o \times n_i = (c_o * o_w * o_h) \times (c_i * i_w * i_h) = 18 \times 12$. Finally, we perform linear operation $Y = C^T X$ that takes the 3 × 2 × 2 size of input matrix flattened as a 12-dimensional vector and produces an 18-dimensional vector. The latter is reshaped as the 2 × 3 × 3 size of output matrix.

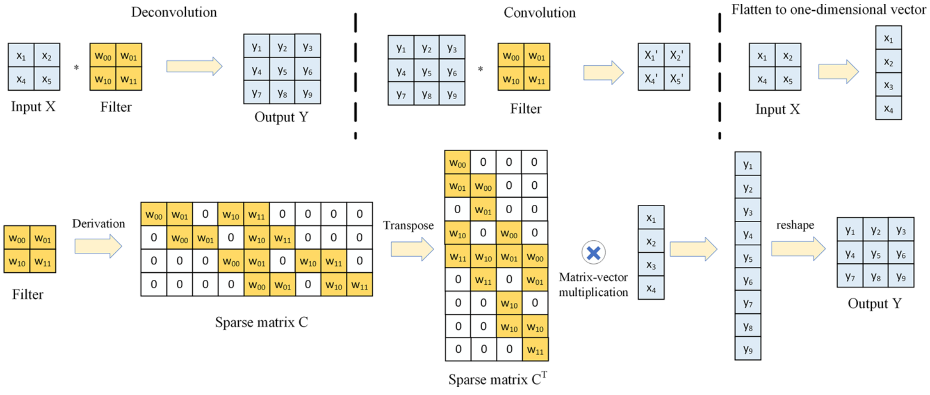


Fig. 8 An example of single channel de-convolution

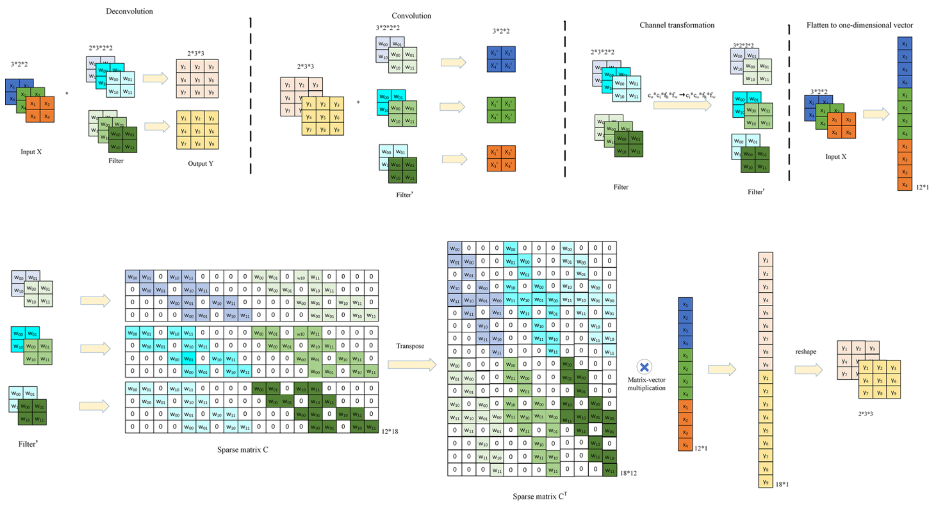


Fig. 9 An example of multiple channel de-convolution

Secure De-convolution with large inputs. The lattice-based PAHE scheme can be used to evaluate matrix-vector product over ciphertext, such as the libraries provided by Gazelle [20]. In the PAHE scheme, the input is packed into a single n-slots ciphertext so that it allows us better utilization of SIMD and control the noise growth. However, the packing methods require that the input size n_i and the output size n_o are smaller than n. Let's consider the noise in the ciphertext. In the lattice-based PAHE scheme, the noise is bounded by the coefficients of the sampled error polynomials, the plaintext size and the number of operations (addition and multiplication). We refer readers to [1] for details. In our framework, the size of a plaintext is a single machine word (64 bits). The coefficients of the sampled error polynomials are pre-defined constants. The number of homomorphic operations is determined by the size of the sparse matrix C, which is $n_o \times n_i$. The matrix-vector product results can be correctly decrypted if $\eta < q/(2p)$ where η is the overall at most noise growth. We have $\eta = n_i \cdot p \cdot \sqrt{n} \cdot \eta_c$ where η_c is the noise growth of the element wise multiplication between C and X. Hence, the maximum size of the input, n_i, should satisfy $n_i < q/(2 \cdot p^2 \cdot \sqrt{n} \cdot \eta_c)$. In practice, when n_i is larger than n, a straight forward way is to divide the input into small blocks and pack each of them [20, 23]. Figure 10(a) shows that the original input X and C are split into $n \times n$ sized blocks that are to be packed independently. However, the homomorphic additions of these blocks are required to achieve the final matrix-vector products. When n_i becomes particularly large, there will be too many number of blocks involved in the homomorphic additions. Consequently, the noise level introduced by these operations may cause the decryption failed.

Since we have two semi-honest servers (S_1 and S_2) in our framework, the sever S_1 keeps the private key sk and decrypts the encrypted intermediate results before performing non-linear layer computations based on 2-PC secure computation. S_2 performs homomorphic evaluation of the linear layers. Obviously, the success of decryption will clear the noise introduced by homomorphic operations. Therefore, we combine PAHE and 2-PC plaintext addition techniques. The idea is to divide the input (X and C) into blocks to perform homomorphic matrix-vector product separately on S_2. After adding a uniform random vector $\vec{r}$ to each intermediate ciphertext homomorphically, S_2 sends them to S_1, the latter performs the

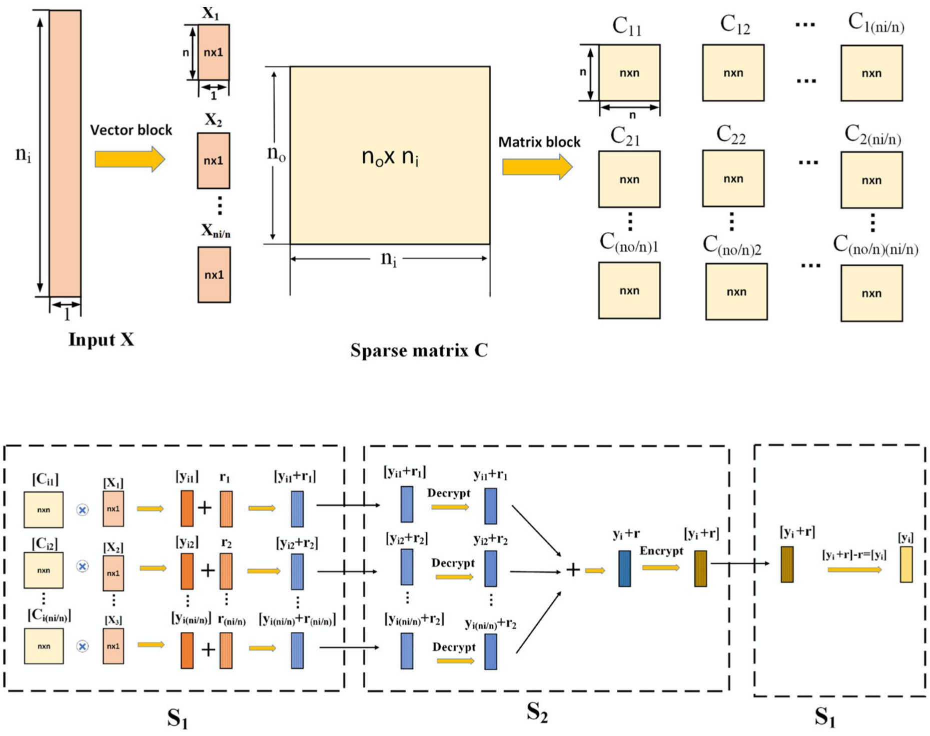


Fig. 10 Examples of divide and conquer technique

decryption. After the additions in plaintext, S_1 encrypts the results with s_k and sends it back to S_2. S_2 gets the matrix-vector product results by subtracting $\vec{r}$ homomorphically. In this way, it can accept large number of inputs and correctly perform homomorphic matrix-vector product (homomorphic additions and multiplications) with low memory consumption (see Section 7). Figure 10(b) takes a group of column block summation as an example, S_2 obtains ciphertext $[y_{ij}]$ by performing homomorphic matrix-vector product on each block $[C_{ij}]$ and $[X_j]$, where $i = 1, ..., n_o/n$ and $j = 1, ..., n_i/n$. Then, S_2 gets $[y_{ij} + \vec{r}_j] = [y_{ij}] + \vec{r}_j$ by homomorphically adding uniform random vector $\vec{r}_j$ to $[y_{ij}]$, which are sent to S_1. The latter decrypts them with the private key sk to get the plaintexts $y_{ij} + \vec{r}_j$. After which, the summation of the plaintexts $\sum_{j=1}^{n_i/n} y_{ij} + \vec{r}_j$ is encrypted again by S_1, denotes as $[y_i + \vec{r}]$, and sent back to S_2. S_2 removes the random vector $\vec{r}$ by homomorphically subtracting $[y_i] = [y_i + \vec{r}] - \vec{r}$ to get the correct ciphertexts of the matrix-vector product results, i.e., $[y_i]$, where $i = 1, 2, ...n_o/n$.

6 The implementation of *SecMinimizer*

The implementation of $SecMinimizer$ includes three components:

– $(Z_{0a}, Z_{0b}) \leftarrow GenRandVec(R, d)$. It returns a pair of d-dimensional vector sets $(Z_{0a} = \{z_{0a}^{(i)}\}_{i=1}^{R}, Z_{0b} = \{z_{0b}^{(i)}\}_{i=1}^{R}$, each of which has R vectors. S_1 (S_2) generates the d-dimensional vector set Z_{0a} (Z_{0b}) with Gaussian distribution, respectively.

Then, S_1 encrypts Z_{0a} and sends the ciphertext $[Z_{0a}]$ to S_2. The latter gets the initial vector set Z_0 in the encrypted form by performing homomorphic addition, i.e., $[Z_0] = [\{z_{0a}^{(i)}\}_{i=1}^{R}] + [\{z_{0b}^{(i)}\}_{i=1}^{R}]$. $[Z_0]$ is used as the initial input of $SecGan$.

- $(Z_a^*, Z_b^*) \leftarrow Minimizer(X_a, X_b)$. Given the secret shares X_a and X_b of input X, it returns the secret shares Z_a^* and Z_b^* of the optimal vector set Z^*, where $z_i^* = z_{ai}^* + z_{bi}^*$, $z_i^* \in Z^*, z_{ai}^* \in Z_a^*, z_{bi}^* \in Z_b^*$. Z_a^* is held by S_1 and Z_b^* is kept by S_2. We use mean squared error (MSE) $\mathcal{L}(X, Z) = ||G(Z) - X||_2^2$ as the loss function, where $G(Z) = G(Z)_a + G(Z)_b$ is the outputs of the WGAN by invoking the function $SecGan$. We use L iterations of gradient descent (GD) to find the optimal vector set Z^*.
- $(X_a', X_b') \leftarrow Reconstruction(Z_a^*, Z_b^*)$. Given the optimal vector set Z_a^* holding by S_1, and Z_b^* holding by S_2, S_1 and S_2 cooperatively invoke $argmin$ and the function $SecGan$ to obtain the final reconstructed input X', which is split into two shares X_a' holding by S_1 and X_b' holding by S_2.

The overall view of the secure minimizer is illustrated in Algorithm 1. It includes secure gradient computation and secure gradient update based on the garbled circuits.

Algorithm 1 Secure Minimizer protocol.

```
Input:
  • $S_1$: The partial vector $Z_a$, sub-image $X_a$
  • $S_2$: The partial vector $Z_b$, sub-image $X_b$, $\vec{r}$
  • loss function:$\mathcal{L}(Z, X)$
  • $R, L, \eta$
Output:
  • $S_1$: $Z_a^* = Z^* + \vec{r}$
  • $S_2$: $Z_b^* = \vec{r}$
1: $(Z_{0_a}, Z_{0_b}) \leftarrow GenRandVec(R, d)$
2: for l=0;l<L;l++ do
3:     $(\nabla_Z \mathcal{L}(Z, X)_a, \nabla_Z \mathcal{L}(Z, X)_b) \leftarrow$ secure gradient computing
4:     $S_1$ computes $x = z_{l_a} + \eta \times \nabla_z \mathcal{L}(z, X)_a$
5:     $S_2$ computes $y = z_{l_b} + \eta \times \nabla_z \mathcal{L}(z, X)_b$
6:     $Z_l \leftarrow sub(x, y)$
7:     $S_1$ get $Z_{l_a} \leftarrow Z_l + \mathbf{r}$
8:     $S_2$ get $Z_{l_b} \leftarrow \mathbf{r}$
9: end for
```

6.1 Secure gradient computation: approximation

For the sake of simplicity, we assume $R = 1$. Given the loss function $\mathcal{L}(X, z) = ||G(z) - X||_2^2$, a straight forward computation is using the numerical differentiation to approximate the true value of the gradient by choosing a small h, such as symmetric difference quotient. Hence, the finite difference approximation of the gradient for the vector z is:

$$\nabla_z \mathcal{L}(z, X)|_{z=z^{(i)}} = \lim_{h \to 0} \frac{\mathcal{L}(z^{(0)}, ..., z^{(i)} + h, ...) - \mathcal{L}(z^{(0)}, ..., z^{(i)} - h, ...)}{2h} \tag{5}$$

where z is a vector of size 128, and X is a given input of the size $i_h \times i_w$.

Note that S_1 and S_2 keep the additive shares $G(z)_a$, $G(z)_b$ of the $SecGan$ outputs and X_a, X_b of the inputs respectively, hence, we have

$$\begin{aligned}\mathcal{L}(z, X) = ||G(z) - X||_2^2 \\ &= ||(G(z)_a + G(z)_b) - (X_a + X_b)||_2^2 \\ &= ||(G(z)_a - X_a) + (G(z)_b - X_b)||_2^2\end{aligned} \tag{6}$$

The gradient descent update is computed recursively as follows:

$$\begin{aligned}&z_1 = z_0 + \eta \times \nabla_z \mathcal{L}(z, I)|_{z=z_0} \\ &z_2 = z_1 + \eta \times \nabla_z \mathcal{L}(z, I)|_{z=z_1} \\ &... \\ &z_L = z_{L-1} + \eta \times \nabla_z \mathcal{L}(z, I)|_{z=z_{L-1}}\end{aligned} \tag{7}$$

Let's take first iteration as an example. Recall that S_1 keeps the additive share z_{0_a} and $\nabla_z \mathcal{L}(z, X)_a|_{z=z_0}$, S_2 keeps another share z_{0_b} and $\nabla_z \mathcal{L}(z, X)_b|_{z=z_0}$, we have

$$\begin{aligned}z_1 &= (z_{0_a} - z_{0_b}) + \eta \times (\nabla_z \mathcal{L}(z, X)_a|_{z=z_0} - \nabla_z \mathcal{L}(z, X)_b|_{z=z_0}) \\ &= (z_{0_a} + \eta \times \nabla_z \mathcal{L}(z, X)_a|_{z=z_0}) - (z_{0_b} + \eta \times \nabla_z \mathcal{L}(z, X)_b|_{z=z_0})\end{aligned} \tag{8}$$

Accordingly, we let S_1 and S_2 independently compute $x = G(z)_a - X_a$ and $y = G(z)_b - X_b$. Then, they compute the loss function $\mathcal{L}(z, X)$ further the gradient $\nabla_z \mathcal{L}(z, X)$ using 2-PC secure computation based on the garbled circuits. The inputs of the circuits include x from S_1, y and randomly generated number r from S_2. The outputs are r denoted as $\nabla_z \mathcal{L}(z, X)_b$ to S_2 and $\nabla_z \mathcal{L}(z, X) + r$ denoted as $\nabla_z \mathcal{L}(z, X)_a$ to S_1. According to the (8), S_1 and S_2 can independently compute $x = z_{0_a} + \eta \times \nabla_z \mathcal{L}(z, X)_a|_{z=z_0}$ and $y = z_{0_b} + \eta \times \nabla_z \mathcal{L}(z, X)_b|_{z=z_0}$. Similarly, the gradient can be updated by using garbled circuits. The garbled circuits can be implemented using sum, multiplication and division gates.

6.2 Secure gradient computation: chain rule

The numerical differentiation requires repeatedly invoking the function $SecGan$ to get the outputs. For example, when z is a 128-dimensional vector, it needs call 256 times of the function $SecGan$ to compute $\nabla_z \mathcal{L}(z, X)|_{z=z^{(i)}}$. Obviously, its efficiency is very low. On the other hand, its accuracy is dependent on h and the Linear property of the function $SecGan$, which may cause the results unacceptable.

We can find derivative of the composite functions by the chain rule. The derivative of the composition equals the derivative of the outside function with respect to the inside, and times the derivative of the inside function. The number of functions that make up the composition determines how many differentiation steps are necessary.

Let's consider the loss function again, we have

$$\nabla_z \mathcal{L}(z, X) = \frac{\partial \mathcal{L}(z, X)}{\partial G(z)} \cdot \frac{\partial G(z)}{\partial z} \tag{9}$$

where $\mathcal{L}(z, X)$ is a scalar and $G(z)$ can be treated as a matrix. Hence, $\frac{\partial \mathcal{L}(z,x)}{\partial G(z)} = Mean(2 \times (G(z) - X))$ has the same shape as $G(z)$. For the $\frac{\partial G(z)}{\partial z}$, we can compute $\frac{\partial G(z)}{\partial z}$ by the chain rule. Taking the WGAN network as an example in Fig. 5, we use o_f to denote the output of the FC layer, o_r^i denotes the output of the i-th $ReLU$ layer, o_d^i denotes the output of the i-th $DeConv$ layer and o_σ denotes the output of the $Sigmod$ layer. Let's say we have the composite function $G(z) = o_\sigma(o_d^3(o_r^3(o_d^2(o_r^2(o_d^1(o_r^1(o_f(z))))))))$. We give the mathematical derivation as follows:

$$\frac{\partial G(z)}{\partial z} = \frac{\partial o_\sigma}{\partial o_d^3} \cdot \frac{\partial o_d^3}{\partial o_r^3} \circ \frac{\partial o_r^3}{\partial o_d^2} \cdot \frac{\partial o_d^2}{\partial o_r^2} \circ \frac{\partial o_r^2}{\partial o_d^1} \cdot \frac{\partial o_d^1}{\partial o_r^1} \circ \frac{\partial o_r^1}{\partial o_f} \cdot \frac{\partial o_f}{\partial z} \tag{10}$$

where '$\cdot$' denotes the matrix-vector product and '$\circ$' denotes component-wise product between vectors. If we can compute the gradient of each layer independently, we can convert the (10) to the two type of operations. Consequently, we can utilize the PAHE scheme instead of 2-PC secure computation to securely calculate the gradient of the WGAN, which will make our implementation simpler and more efficient.

Let's take a close look at each layer:

- **Derivative of FC and $DeConv$ layer.** The FC layer performs the matrix-vector product. Hence, given the inputs x, we have $\frac{\partial o_f}{\partial x} = W$, where W is the weight matrix used by FC layer. Recall that we can transform the de-convolution into matrix-vector product as well, similarly, given the inputs x, we have $\frac{\partial o_d^i}{\partial x} = C^i$, where C^i is the sparse matrix used in the i-th $DeConv$ layer.
- **Derivative of $ReLU$ layer.** Given the inputs x, the outputs of the i-th $Relu$ layer is defined as $o_r^i(x) = max(0, x)$. The derivative of the i-th $ReLU$ layer is:

$$o_r^{i'}(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

  Hence, we have $\frac{\partial o_r^i}{\partial x} = R^i$, where R^i has the same shape of the input x.
- **Derivative of $Sigmod$ layer.** Given the inputs x, the outputs of the $Sigmod$ layer is defined as $o_\sigma(x) = \frac{1}{1+e^{-x}}$. The derivative of the $Sigmod$ layer is

$$o'_\sigma(x) = o_\sigma(x)(1 - o_\sigma(x))$$

  Hence, we have $\frac{\partial o_\sigma}{\partial x} = o'_\sigma(x)$, which is a vector with the same shape of the inputs x.

Based on the gradient calculations above, the (10) can be transformed to:

$$\nabla_z \mathcal{L}(z, x) = \mathcal{L} \circ \mathcal{G} \cdot C^3 \circ R^3 \cdot C^2 \circ R^2 \cdot C^1 \circ R^1 \cdot W \tag{11}$$

where $\mathcal{G}$ denotes $\frac{\partial o_\sigma}{\partial o_d^3}$, $\mathcal{L}$ is $\frac{\partial \mathcal{L}(z,x)}{\partial G(z)}$.

Now, we can use chain rule to calculate the gradient of the WGAN by only invoking the function $SecGan$ once. It is very fortunate that the operations involved in (11) are matrix-vector product and component-wise vector product. Therefore, we can use the PAHE scheme combined with the 2-PC plaintext addition to implement the secure gradient computation, which is the same as the secure de-convolution computation technique proposed in Section 5. Considering the last layer of the example in Fig. 5, $[\mathcal{L}]$ is an encrypted vector of size 784×1. Hence, $[\mathcal{L}] \circ [\mathcal{G}]$ is an encrypted vector of size 784×1. $[C^3]$ is an encrypted matrix of size 12544×784, so $[\mathcal{L}] \circ [\mathcal{G}] \cdot [C^3]$ is an encrypted vector of size 12544×1. This process continues for each layer, and we finally get the encrypted vector, $[\nabla_z \mathcal{L}(z, x)]$, of size 128×1. In Section 7, we will further show the detailed information and performance of each layer.

7 Experimental evaluation

Dataset Mnist. We select the Mnist dataset of handwritten digits to measure the feasibility of our framework. The MNIST dataset has a training set of 60,000 examples and a test set of 10,000 examples. The training set is made up of numbers written by 250 different people and the value of each label is an integer between 0 and 9. Each image consists of 28 $\times$ 28 pixels, each pixel is represented by a gray value. We randomly selected 5,000 images in our experiments. They are grouped by content into 10 categories, each of which contains 500 images and is corresponding to an integer number of 0 to 9. In the experiments, we selected distinct collections of images, containing 500, 1,000, 1,500,..., and 5,000 distinct images, respectively.

Dataset coral. The publicly available Coral database has been used for the CBIR task. The database consists of 1,000 test images which are divided into 10 different classes. Each class consists of 100 images. The classes are created based on the different objects available in the images. The classes are African peoples, elephant, beach, rose, building, horse, bus, mountain, dinosaur, and food dish.

The attacking model. We take Fast Gradient Sign Method (FGSM) under black-box attack model to evaluate the effectiveness of our framework. Given an image x and its corresponding true label y, the FGSM under black-box attack sets the perturbation δ to:

$$\delta = \epsilon \cdot sign(\nabla_x J(x, y)). \tag{12}$$

It has been shown that FGSM under black-box attack [15] is extremely fast rather than optimal. It simply uses the sign of the gradient at every pixel to determine the direction with which to change the corresponding pixel value.

The FGSM under black-box attack assumes that adversaries have no access to the neural network parameters, neither do they have access to a large training dataset. The adversary trains a substitute model using a very small dataset augmented by synthetic images. Adversarial samples are then found by applying the FGSM method on the substitute model.

Evaluation metrics. We evaluate the performance of the implementation of the functions *SecGan* and *SecMinimizer* respectively, which includes the time consumption (termed as *time cost*), the memory space consumption (termed as *memory cost*), and the overall communication overhead between the cloud servers S_1 and S_2 (termed as *communication cost*). The *communication cost* refers to the size of the intermediate data in bytes exchanged between the servers. The functions include online and offline phases. The online phase begins with the cloud servers S_1 and S_2 sharing their inputs and the generation of random vector z. Then, S_1 and S_2 execute the input reconstruction task by cooperatively running L steps of GD. The offline phase includes the pre-process of the inputs and the circuits garbling. The classification accuracy (termed as *accuracy*) with adversary samples is also measured to show the correctness of our implementation.

The *communication cost* is measured with the tool NetHogs 0.8. Please note that we only measured the time consumed by each server, and the communication delay is ignored.

Implementation details. Our framework needs a packed additive homomorphic encryption (PAHE) scheme and a two-party secure computation (2-PC) scheme. Parameters for both schemes are selected for a 128-bit security level. The PAHE scheme was implemented

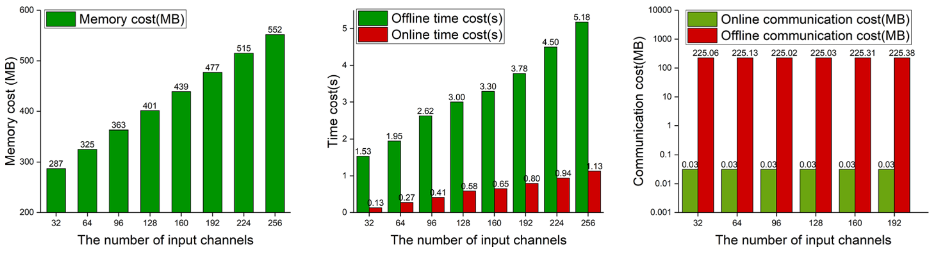


Fig. 11 Performance of direct convolution v.s. the number of input channels

by using Gazelle, where the plaintext modulus p is set to 22 bits and the ciphertext modulus q is chosen to be a 60-bit psuedo-Mersenne prime. The 2-PC scheme was implemented by using Obliv-C. The circuits garbling phase is counted into the offline phase because its independent of the users' inputs.

7.1 Micro-benchmarks

Secure de-convolution benchmarks. We first benchmark the impact of the direct convolution on the secure de-convolution performance. Because we use the channel packing technique, the input is packed and encrypted c_i/c_n ciphertexts and the output contains c_o/c_n ciphertexts, where c_n represents the number of channels that fit in a single ciphertext. Hence, we need $c_o \times c_i \times k_h \times k_w \times i_h \times i_w$ *SIMDAdd* and *SIMDScMult* calls. We measured the *memory cost*, *time cost* and *communication cost* under different number of input channels and output channels, which are presented in Figs. 11 and 12, respectively. In our benchmark, for each channel, the input size is set to 4×4, the output size is set to 8×8, the filter size is set to 5×5 and the stride is set to 2. Thus, we have $c_n = 32$.

Secure matrix-vector product benchmarks. We further benchmark the impact of the matrix-vector product on the secure de-convolution performance. The de-convolution operation is transformed into matrix-vector product by deriving the sparse matrix C with the size of $n_o \times n_i = (c_o * o_w * o_h) \times (c_i * i_w * i_h)$. Hence, we need n_i *SIMDScMult* and $n_i - 1$ *SIMDAdd* calls. Obviously, the number of homomorphic operations is dropped compared with the direct convolution. We used the same input parameters and measured the *memory cost*, *time cost* and *communication cost* under different number of input channels and output channels, which are presented in Figs. 13 and 14, respectively. Because we divide the input into blocks to perform homomorphic matrix-vector product separately when the number of input channels is larger than 128, the *communication cost* increases. Compared with

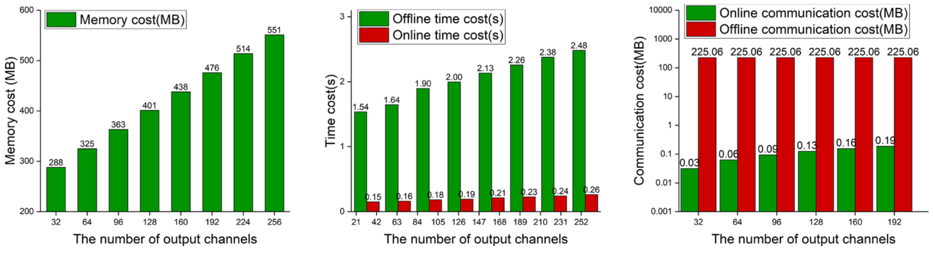


Fig. 12 Performance of direct convolution v.s. the number of output channels

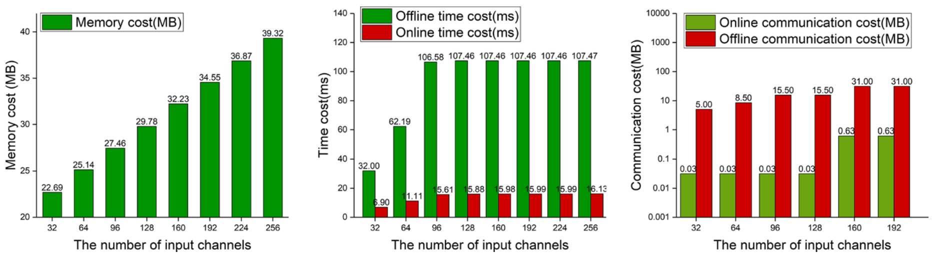


Fig. 13 Performance of matrix-vector product v.s. the number of input channels

the results presented in Figs. 11 and 12, both the *time cost* and the *memory cost* are dropped, which shows that matrix-vector product is 30× faster than the direct convolution. It only costs about 0.03 seconds when the number of the input channels is 256, while the direct convolution consumes about 1.1 seconds.

7.2 The *SecGan* benchmarks

We report runtimes and network bandwidth for the secure evaluation of the generator in the WGAN network. We compare the two de-convolution methods that are composed into the evaluation. The results are shown in Table 1.

7.3 The *SecMinimizer* benchmarks

We benchmark the two methods that are used during the secure SGD computation in the *SecMinimizer*. Table 2 shows the *time cost* and the *communication cost* of the two methods for one gradient update. Because the numerical differentiation method needs repeatedly calling the generator in the WGAN network, the *time cost* is very heavy. In shark contrast, the chain rule method is 400× faster. The online time of the one gradient update in the WGAN network is about 1.6 seconds.

7.4 The overall benchmarks

We compose the *SecGan* and *SecMinimizer* from the previous section and evaluate the complete WGAN network over Mnist dataset. We set $L = 1$, $R = 1$. Table 3 compares the runtimes and bandwidth of our method and that of Samangouei's [27] in plaintext. Our

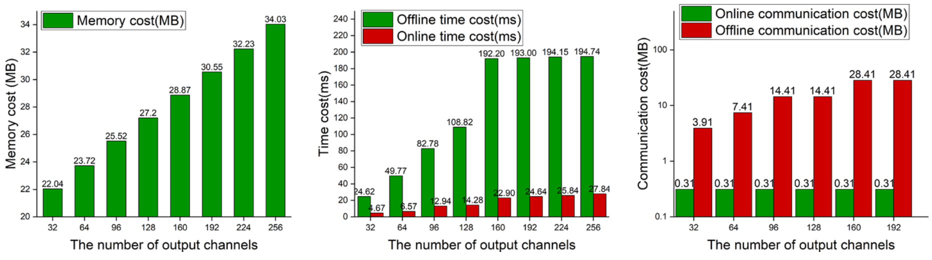


Fig. 14 Performance of matrix-vector product v.s. the number of output channels

Table 1 The benchmarks of the *SecGan* for the WGAN network

	Runtime (s)			Communication (MB)			Memory
	offline	online	total	offline	online	total	(MB)
Direct convolution	17.55	3.64	21.19	225.25	1	226.25	1468
Matrix vector product	18.52	2.50	21.02	448	10	458	1539

secure solution is 150× slower than that over plaintext, which is about 34.2 seconds. Please note that the online time is 5.3 seconds, which is about 20× slower than that over plaintext.

7.5 Correctness with adversary samples

Since our work focus on comparing the accuracy differences rather than the accuracy between our method and Samangouei's, the selection of NN network is not important. We refer to the model structure of Samangouei [27] and choose model A {a four-layer convolutional neural network with two $Conv$ layers and two FC layers} as our classifier and model B {a four-layer convolutional neural network with three $Conv$ layers and one FC layer} as our substitute network. When the original images are used to train the classifier and defense-GAN is used, we refer to it as *Defense-GAN-Ori*. Under the *Defense-GAN-Ori*, we compare Sec-Defense-Gan with defense-GAN proposed by Samangouei [27] under FGSM black-box attacks as well as under no attack. Figure 15 shows the classification accuracy of the classifiers using Sec-Defense-Gan with various number of iterations L ($R = 10$, $\epsilon = 0.3$) on the Mnist dataset. We use simple parameter discretization method in the experiments, i.e., we represent real numbers with a fixed precision of n decimal points by directly scale up real numbers by 10n times and converting them into 64-bit integers. Obviously, the larger n will lead to the better accuracy, but may cause result overflow when performing the homomorphic addition. Therefore, there is a certain loss of accuracy in the experiments even the results show that our solution performs consistently well under the FGSM black-box attack.

7.6 Correctness evaluation of CBIR

In our experiment, we use Mnist and Coral dataset. The images of different classes are divided into training and testing set. For the training of the classifier model, 80% images

Table 2 The benchmarks of *SecMinimizer* for one gradient update iteration

	Runtime(s)			Communication(MB)			Memory
	Offline	Online	Total	Offline	Online	Total	(MB)
Chain rule	11.369	1.648	13.017	452.155	8.75	460.905	1522
Numerical differentiation	4492.8	640	5132.8	225.25	1	226.25	1468

Table 3 the overall performance under Plaintext and Ciphertext

	Runtime(s)			Communication(MB)			Memory(MB)
	offline	online	total	offline	online	total	
Ciphertext	28.919	5.288	34.207	452.155	8.75	460.905	1522
Plaintext	–	–	0.279	–	–	–	3053.41

are used and 20% images are used as testing set denoted as $Test\_Ori$. A total of 800 and 200 images are used for training and testing respectively. In addition, we use FGSM algorithm to generate the variant of testing set, called $Test\_FGSM$. We use our framework to reconstruct $Test\_FGSM$, call $Test\_Rec$.

In the experiment, the two-layer CNN model is used. For the first layer of the CNN model, 32 feature maps of $3 \cdots 3$ are used for convolution with an input image to derive the features. After the convolution, $2 \cdots 2$ max pooling is used to reduce the feature size. For the second layer, 16 feature maps of $3 \cdots 3$ are used for the convolution with the output of the first layer to derive the features. After the convolution, $2 \cdots 2$ max pooling is used to reduce the feature size. After the max pooling, flattening is used to convert feature maps into a single column vector, which will be passed to the neural network. The performance of the CBIR can be measured by the precision rate. Precision Rate is defined as the ratio of the total number of relevant images retrieved to the total number of images retrieved:

$$Precision = \frac{Number\ of\ relevant\ images\ retrived}{Total\ number\ of\ images\ retrived} \tag{13}$$

Table 4 and 5 shows the classification precision of the CBIR on Mnist and Coral datasets.

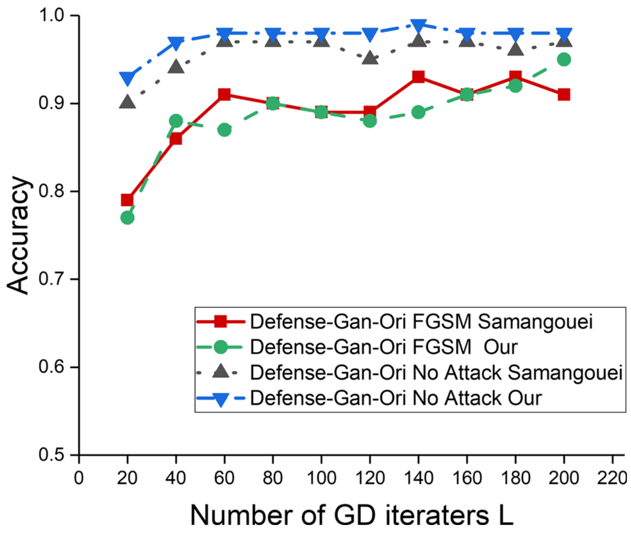


Fig. 15 Classification accuracy comparison of Model A using Defense-GAN with different L ($R = 10$)

Table 4 Precision of CBIR on different datasets

	Mnist		
	Mnist_Test_Ori	Mnist_Test_FGSM	Mnist_Test_Rec
Precision	94%	16%	92%

Finally, Fig. 16 shows the content-based image retrieval results on Mnist dataset where the most relevant images have been successfully retrieved at top ranks. In the figure, the first column is the provided query image. The remaining columns are the retrieved images sorted according to their similarity (the Euclidean distance, see [23] for details) to the query image.

8 Conclusions

In this paper, we design two secure de-convolution algorithms and two secure gradient computation and update protocols to accomplish image reconstruction task. With these build blocks, we propose and implement Sec-Defense-Gan and apply it to secure CBIR system. Sec-Defense-Gan uses a judicious combination of PAHE scheme and garbled circuit based two-party computation, both of which satisfy IND-CPA security, i.e., ciphertexts of any two messages are computational indistinguishable. Sec-Defense-Gan guarantees the input/output image data and GANs model details confidentiality while providing the ability to defense against adversarial image examples. The application of Sec-Defense-Gan to CBIR over datasets shows that it can successfully retrieve similar images from the datasets without leaking input image data, intermediate results and retrieved image data to the cloud.

There are several ways in which Sec-Defense-Gan can be improved. First, PAHE scheme supports SIMD operations over ciphertext domain, however, the time-consuming is still very heavy because $Deconv$ or FC layer has homomorphic operations quadratic in the input size. It is a straightforward way to consider the sparsity of the weight matrix in the $Deconv$ or FC layer. The weight matrix can be 'compressed' before it is packed and encrypted. Another important direction is to focus more on specific adversarial image type rather than general examples, which may reduce L iterations of gradient descent (GD) to find the optimal vector set Z^*. More importantly, our work did not investigate theoretical bounds on the information leakage of each function in our framework, which would remain as a topic of

Table 5 Precision of CBIR retrieved results on Coral dataset

	Coral		
	Coral_Test_Ori	Coral_Test_FGSM	Coral_Test_Rec
Precision	94.5%	15%	91%

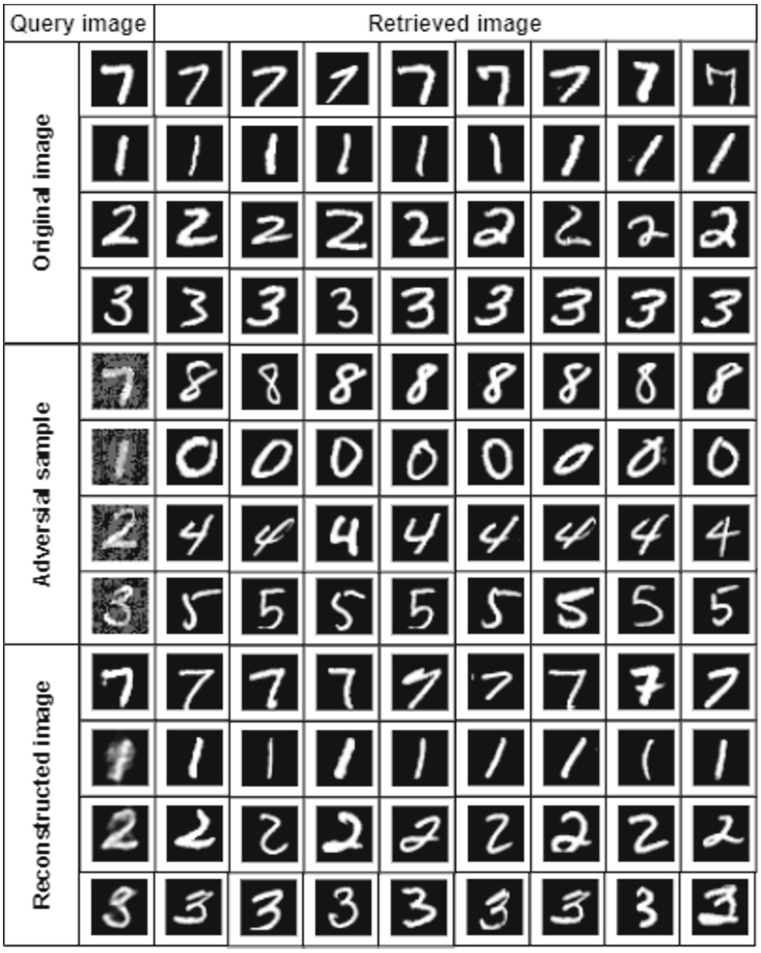


Fig. 16 Samples of retrieved similar images

future research. The last but not the least, it is necessary to design the secure training algorithms of GANs model over ciphertext domain because of the confidentiality requirement of image dataset.

Funding This work is supported by the National Key Research and Development Program of China (Grant No. 2018YFB0804702).

Declarations

Conflict of Interests The authors declare no competing interests.

References

1. Agrawal S, Freeman DM, Vaikuntanathan V (2011) Functional encryption for inner product predicates from learning with errors. In: Lee DH, Wang X (eds) Advances in cryptology – ASIACRYPT 2011. Springer Berlin Heidelberg, Berlin, Heidelberg, pp 21–40
2. Bansal M, Kumar M, Kumar M (2021) 2d object recognition: a comparative analysis of sift, surf and orb feature descriptors. Multimedia Tools and Applications 80(12):18839–18857
3. Bansal M, Kumar M, Kumar M, Kumar K (2021) An efficient technique for object recognition using shi-tomasi corner detection algorithm. Soft Computing 25:4423–4432
4. Bansal M, Kumar M, Sachdeva M, Mittal A (2021) Transfer learning for image classification using vgg19: Caltech-101 image data set. Journal of Ambient Intelligence and Humanized Computing
5. Brakerski Z, Vaikuntanathan V (Oct 2011) Efficient Fully Homomorphic Encryption from (Standard) LWE. In: 2011 IEEE 52nd Annual Symposium on Foundations of Computer Science, pp 97-106
6. Brakerski Z, Vaikuntanathan V (2011) Fully homomorphic encryption from ring-lwe and security for key dependent messages. In: Proceedings of the 31st annual conference on advances in cryptology, ser. CRYPTO'11. Springer-Verlag, Berlin, Heidelberg, pp 505–524
7. Chillotti I, Gama N, Georgieva M (2020) TFHE: Fast Fully Homomorphic Encryption Over the Torus. J Cryptol 33:34–91
8. Chillotti I, Gama N, Georgieva M (2016) Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In: Cheon JH, Takagi T (eds) Advances in cryptology – ASIACRYPT 2016. Springer, Berlin, Heidelberg, pp 3–33
9. Dowlin N, Gilad-Bachrach R, Laine K, Lauter K, Naehrig M, Wernsing J (Feb 2016) CryptoNets: applying neural networks to encrypted data with high throughput and accuracy. Microsoft Research, Tech. Rep. MSR-TR-2016-3
10. Elgamal T (July 1985) A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory 31(4):469–
11. Ferreira B, Rodrigues J, Leit?o J, Domingos H (Sept 2015) Privacy-Preserving Content-Based Image Retrieval in the Cloud. In: 2015 IEEE 34th Symposium on Reliable Distributed Systems (SRDS), pp 11–20
12. Goldreich O (2004) Foundations of cryptography. Basic Applications, vol 2. Cambridge University Press, New York
13. Goldwasser S, Micali S, Rackoff C (1989) The knowledge complexity of interactive proof systems. SIAM J Comput 18(1):186–208
14. Goodfellow IJ, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial networks
15. Goodfellow I, Shlens J, Szegedy C (2015) Explaining and harnessing adversarial examples. arXiv:1412.6572
16. Gulrajani I, Ahmed F, Arjovsky M, Dumoulin V, Courville A (2017) Improved training of wasserstein gans
17. Hastings M, Hemenway B, Noble D, Zdancewic S (May 2019) SoK: General Purpose Compilers for Secure Multi-Party Computation. In: 2019 IEEE Symposium on Security and Privacy (SP), pp 1220–1237
18. Hsu CY, Lu CS, Pei SC (2012) Image feature extraction in encrypted domain with privacy-preserving sift. IEEE Transactions on Image Processing 21(11):4593–4607
19. Hu S, Wang Q, Wang J, Qin Z, Ren K (2016) Securing sift: privacy-preserving outsourcing computation of feature extractions over encrypted image data. IEEE Transactions on Image Processing 25(7):3411-3425
20. Juvekar C, Vaikuntanathan V, Chandrakasan A (2018) GAZELLE: A low latency framework for secure neural network inference. In: 27th USENIX security symposium (USENIX security 18). Baltimore, MD: USENIX Association, pp 1651-1669
21. Kumar A, Kumar M, Kaur A (2021) Face detection in still images under occlusion and non-uniform illumination. Multimedia Tools and Applications, vol. 80
22. Li P, Li T, Yao Z-A, Tang C-M, Li J (Aug 2017) Privacy-preserving outsourcing of image feature extraction in cloud computing. Soft Computing 21(15):4349–4359
23. Liu F, Wang Y, Wang F, Zhang Y, Lin J (2019) Intelligent and secure content-based image retrieval for mobile users. IEEE Access 7:119209–119222
24. Mao Q, Wang L, Tsang IW (May 2017) A unified probabilistic framework for robust manifold learning and embedding. Mach Learn 106(5):627–650

25. Nasr M, Shokri R, Houmansadr A (May 2019) Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In: 2019 IEEE Symposium on Security and Privacy (SP), pp 739–753
26. Paillier P (1999) Public-key cryptosystems based on composite degree residuosity classes. In: IN ADVANCES IN CRYPTOLOGY - EUROCRYPT 1999. Springer-Verlag, pp 223–238
27. Samangouei P, Kabkab M, Chellappa R (2018) Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models. ArXiv, 1805.06605. access date 2018.05.18
28. Singh S, Ahuja U, Kumar M, Kumar K, Sachdeva M (2021) Face mask detection using yolov3 and faster r-cnn models: Covid-19 environment. Multimedia Tools and Applications, vol. 80
29. Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, Fergus R (2013) Intriguing properties of neural networks
30. Wang X, He K, Song C, Wang L, Hopcroft JE (2019) At-gan: An adversarial generator model for non-constrained adversarial examples
31. Wang Z, Song M, Zhang Z, Song Y, Wang Q, Qi H (2019) Beyond inferring class representatives: User-level privacy leakage from federated learning. In: IEEE INFOCOM 2019 - IEEE conference on computer communications, pp 2512–2520
32. Zahur S, Evans D (2015) Obliv-C: A language for extensible data-oblivious computation. Cryptology ePrint Archive, Report 2015/1153. https://eprint.iacr.org/2015/1153
33. Zheng P, Huang J (2013) An efficient image homomorphic encryption scheme with small ciphertext expansion. In: Proceedings of the 21st ACM International conference on multimedia, ser. MM ’13. New York, NY, USA: ACM, pp 803–812