Volumen 5 (2019)
XXV Verano de la Ciencia
ISSN 2395-9797

Compresión y encriptación de información utilizando descomposición en valores singulares.

Grace E. Molina1, David O. Palacios2, Gabriel A. Torres3 y Eduardo Cabal Yépez4

1 [Departamento de Matemáticas, Universidad de Guanajuato, Guanajuato, México] [ge.molinachavez@ugto.mx]
2 [Facultad de Ingeniería, Universidad de San Carlos de Guatemala, Ciudad de Guatemala, Guatemala] [daorpamo96@live.com]
3 [Escuela de Nivel Medio Superior de Moroleón, Moroleón, México] [ga.torresmendoza@ugto.mx]
4 [Departamento de Estudios Multidisciplinarios, Universidad de Guanajuato, Yurira, México] [educabal@ugto.mx]

Desde los métodos antiguos hasta los sistemas modernos, el ser humano siempre ha buscado la privacidad de sus comunicacio- nes. El cifrado de los datos se hace cada vez más importante debido al aumento de los robos de la información que viaja por la red, todo esto ha llevado a buscar una mayor seguridad en la transmisión de la información. La encriptación y compresión de imágenes digitales son técnicas empleadas para proteger y redu- cir el espacio de almacenamiento requerido por los datos. En es- te trabajo se implementó el método de compresión de imágenes propuesto en Salazar et al. (2018) y el método de encriptación de imágenes a color utilizando la descomposición en valores singu- lares (SVD, por sus siglas en inglés), propuesto en Nidhal et al. (2014). En el algoritmo, primero se comprime el archivo, luego se encripta usando una clave aleatoria. En el envío del paque- te de datos al receptor, se adjunta la clave cifrada con la clave aleatoria. Para descifrar el archivo o documento, se realiza el proceso inverso.

SVD | Compresión de imágenes | Criptografía |

Introducción

Una de las cosas que siempre ha acompañado al ser humano ha sido la existencia de conflictos entre personas, ya sea por razones territoriales o de recursos naturales, o por motivos religiosos o ideológicos. En el pasado, esto derivó en el uso de la fuerza y la existencia de ataques, violencia y guerras, donde el más poderoso resultaba ser el vencedor. Por esta ra- zón y con el fin de ganar en estas luchas, desde los métodos antiguos hasta los sistemas modernos, el ser humano siempre ha buscado la privacidad de sus comunicaciones. Las prime- ras técnicas de criptografía o de ocultación de información fueron creadas en este contexto para permitir el envío de in- formación de manera segura entre partes. Las guerras fueron el principal factor que promovió los avances criptográficos en la historia [1]. En la era moderna, la aparición de la infor- mática y el uso masivo de las comunicaciones digitales, han producido un número creciente de problemas de seguridad. En la actualidad, se utilizan distintos métodos basados en la matemática y la informática para ocultar datos ante observa- dores no autorizados. La criptografía actual permite proteger la información de terceras personas, lo que garan-

tiza su con- fidencialidad a la vez que provee mecanismos para asegurar la autenticidad y la integridad [1]. En el cifrado moderno se usan algoritmos simétricos, asimétricos e híbridos. La cripto- grafía simétrica solo utiliza una clave para cifrar y descifrar el mensaje, que tiene que conocer el emisor y el receptor previa-

mente y este es el punto débil del sistema, la comunicación de las claves entre ambos sujetos, ya que resulta más fácil interceptar una clave que se ha transmitido sin seguridad. La criptografía asimétrica se basa en el uso de dos claves: la pú- blica (que se podrá difundir sin ningún problema) y la privada (que no debe de ser revelada nunca). Por otra parte, la criptografía híbrida es un método criptográfico que usa tanto un cifrado simétrico como un asimétrico. Emplea el cifrado de clave pública para compartir una clave para el cifrado simé- trico. El mensaje que se esté enviando en el momento, se cifra usando su propia clave privada, luego el mensaje cifrado se envía al destinatario.

El cifrado de los datos se hace cada vez más importante de- bido al aumento de los robos de la información que viaja por la red. Pero, además de los problemas de seguridad en las co- municaciones, cada día es mayor la necesidad de mejorar la calidad de la información.

Una imagen digital está representada por una matriz, donde cada pequeño elemento de la imagen o "pixel" tiene asig- nado un numero en la escala de grises entre el blanco y el negro (tiene asignado tres números si se trata de una imagen a color) [2]. La imagen podría tener 512 = 29 pixeles en cada fila y 256 = 28 pixeles en cada columna. De modo que la imagen estaría representada por una matriz de 256 512 con 217 entradas; almacenar esta imagen en una computadora no representa ciertamente un problema. Pero cuando se tra- ta de procesar decenas o cientos de imágenes como en una resonancia magnética o tomografía computarizada, entonces si es necesario el uso técnicas de compresión que no pierdan la calidad de la imagen. Otro campo donde es importante la compresión de imágenes es en los sitios de internet. El "peso" de las imágenes insertadas en las páginas web debe ser ligero. Ello contribuirá a que la carga sea más rápida. Por estas razones y por otras más que se podrían mencionar, es necesario un método que sea capaz de reducir el espacio de almacena- miento requerido eliminando información redundante.

En este trabajo se implementó el método de compresión de imágenes propuesto en [3] y el método de encriptación de imágenes a color utilizando la descomposición en valores singulares (SVD, por sus siglas en inglés), propuesto en [4]. La descomposición en valores singulares, ha sido uno de los grandes aportes del álgebra lineal moderna con diferentes aplicaciones en campos como: la visión artificial, recuperación de información de bases de datos, computación lingüís- tica, procesamiento de señales e imágenes, entre otros [3].

Descomposición en valores singulares

Sea $A \in \mathbb{R}^{m\times n}$ una matriz. La matriz A^TA es simétrica y ortogonalmente diagonalizable. Sean $\{\vec{v}_1, ..., \vec{v}_n\}$ una base ortogonal de $\mathbb{R}^n$ formada por los vectores propios de A^TA y $\lambda_1, ..., \lambda_n$ los valores propios correspondientes. Entonces para $1 \le i \le n$ tenemos que

$$\begin{aligned} \|A\vec{v}_i\|^2 &= (A\vec{v}_i)^T A\vec{v}_i \\ &= \vec{v}_i^T A^T A\vec{v}_i \\ &= \vec{v}_i^T(\lambda_i\vec{v}_i) \\ &= \lambda_i. \end{aligned} \qquad (1)$$

De modo que los valores propios de A^TA son todos no negativos. Los **valores singulares** de A (denotados por $\sigma_1, ..., \sigma_n$) son las raíces cuadradas de los valores propios de A^TA, esto es, $\sigma_i = \lambda_i$ para $1 \le i \le n$ [5]. De **(1)** se sigue que los valores singulares de A son las longitudes de los vectores $A\vec{v}_1, ..., A\vec{v}_n$.

La descomposición en valores singulares de la matriz A es una factorización del tipo

$$A = USV^T$$

donde $U \in \mathbb{R}^{m\times m}$ es una matriz ortogonal (esto es, $UU^T = U^TU = I$) cuyas columnas son los vectores propios de AA^T, $V \in \mathbb{R}^{n\times n}$ es una matriz ortogonal cuyas columnas son los vectores propios de A^TA y $S \in \mathbb{R}^{m\times n}$ es una matriz diagonal, donde $S_{ij} = 0$ si $i = j$, $S_{ii} = \sigma_i$ y $\sigma_1 \ge \sigma_2 \ge ... \ge \sigma_n \ge 0$. Las entradas entradas $\sigma_1, ..., \sigma_n$ son los valores singulares de A. Las columnas de U son llamadas vectores singulares izquierdos de A y las columnas de V son llamadas vectores singulares derechos de A.

Consideremos la matriz

$$B = \begin{bmatrix} 1 & 2 & 2 \\ 2 & 3 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Entonces

$$B^TB = BB^T = \begin{bmatrix} 6 & 10 & 6 \\ 10 & 17 & 10 \\ 6 & 10 & 6 \end{bmatrix}.$$

Los valores propios de B'B son

$$\lambda_1 = 28.86$$

$$\lambda_2 = 0.14$$

$$\lambda_3 = 0$$

y los vectores propios de $B^TB = BB^T$ son

$$\begin{bmatrix} 0.454 \\ 0.542 \\ -1 \end{bmatrix}$$

$$u_1 = v_1 \begin{bmatrix} 0.766 \\ 0.454 \end{bmatrix}, u_2 = v_2 \begin{bmatrix} -0.643 \\ 0.542 \end{bmatrix}, u_3 = v_3 \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Por lo que el conjunto de vectores propios ortonormales de $B^TB = BB^T$ es

$$u_1 = v_1 \begin{bmatrix} 0.454 \\ 0.766 \\ 0.454 \end{bmatrix}, u_2 = v_2 \begin{bmatrix} 0.542 \\ -0.643 \\ 0.542 \end{bmatrix}, u_3 = v_3 \begin{bmatrix} -0.707 \\ 0 \\ 0.707 \end{bmatrix}.$$

Así, tenemos que

$$B = USV^T$$

donde

$$U = \begin{bmatrix} 0.454 & 0.542 & -0.707 \\ 0.766 & -0.643 & 0 \\ 0.454 & 0.542 & 0.707 \end{bmatrix}$$

$$S = \begin{bmatrix} \sqrt{28.86} & 0 & 0 \\ 0 & \sqrt{0.14} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0.454 & 0.766 & 0.454 \\ 0.542 & -0.643 & 0.542 \\ -0.707 & 0 & 0.707 \end{bmatrix}.$$

Obsérvese que el segundo valor singular, $\sqrt{0.14}$, de B es mucho mas pequeño que el primero. Si $\hat{S}$ es la matriz obtenida de S al reemplazar 0.14 por 0. Entonces

$$USV = \begin{bmatrix} 1.11 & 1.87 & 1.11 \\ 1.87 & 3.15 & 1.87 \\ 1.11 & 1.87 & 1.11 \end{bmatrix}$$

que es una aproximación a la matriz B, esta factorización de la matriz B se llama descomposición en valores singulares reducida [6].

Compresión de imágenes utilizando SVD

La idea del algoritmo consiste en encontrar la información esencial dentro de la matriz que representa la imagen y almacenar solo eso. La clave está en los valores singulares. Típicamente, algunos σ's son más significativos y otros son extremadamente pequeños; si se preservan los más significativos y se desechan los más pequeños, entonces se logra reducir el peso de la imagen sin perder calidad [7].

Se escribió un código en Matlab implementando el algoritmo de compresión presentado en [3]. Primero se calcula la SVD para obtener los valores singulares que se encuentran en cada componente de la imagen. Posteriormente, se comprime la imagen mediante la manipulación de los valores singulares.

Algoritmo de compresión

1. Leer la imagen a color

```
Im=imread('C:\compresion\imagen.jpg');
```

2. Almacenar la imagen en formato double.

```
=double(Im);
```

3.Dividir la imagen en sus tres componentes RGB.

```
R=I(:,:,1); G=(:,:,2); B=(:,:,3);
```

4.Calcular la SVD de cada componente.

```
[U1,S1,V1]=svd(R);
[U2,S2,V2]=svd(G);
[U3,S3,V3]=svd(B);
```

5.Guardar en la variable BlockSize el número de valores singulares que se van a utilizar.

```
BlockSize=150;
```

6.Igualar a 0 los valores singulares que se van a descartar.

```
S1(BlockSize+1:m,BlockSize+1:n)=0;
S2(BlockSize+1:m,BlockSize+1:n)=0;
S3(BlockSize+1:m,BlockSize+1:n)=0;
```

7. Comprimir cada una de las componentes.

```
imOutR=U1(:,1:Blocksize)*S1(1:Blocksize
,1:Blocksize)*V1'(1:Blocksize,:);
,1:Blocksize)*V2'(1:Blocksize,:)
imOutB=U3(:,1:Blocksize)*S3(1:Blocksize
,1:Blocksize)*V3'(1:Blocksize,:);
;imOutG=U2(:,1:Blocksize)*S2(1:Blocksize
```

8. Guardar la imagen comprimida.

```
ImComprimida(:,:,1)=imOutR;
ImComprimida(:,:,2)=imOutG;
ImComprimida(:,:,3)=imOutB;
ImagenComprimida=uint8(ImComprimda);
imwrite(ImagenComprimida, 'C:\compresion
\comprimida.jpg')
```

Encriptación de imágenes utilizando SVD

Se escribió un código en Matlab implementando el algoritmo de encriptación de imágenes propuesto en [4]. La idea del al- goritmo consiste en perturbar la información esencial dentro de la matriz que representa la imagen, esto, con el fin de ob- tener una imagen diferente a la original. La encriptación se logra mediante el uso de SVD, una clave (que deberá tener el receptor) y operaciones con matrices. Al final del algoritmo, se obtiene la matriz de la imagen encriptada que será la llave pública y un archivo de texto con la clave para desencriptar que será la llave privada.

Algoritmo de encriptación. El algoritmo se puede aplicar tanto a imágenes a colores como a imágenes a escala de gri- ses. Se debe implementar el siguiente método de encriptación a cada una de las componentes de la imagen a color (rojo, azul y verde).

1. Leer la imagen.

```
Im=imread('C:\encriptacion\imagen.jpg');
```

2.Almacenar la imagen en formato double.

```
A=double(Im);
```

3.Crear tres claves (key1, key2, key3) de tal forma que cumplan las siguientes condiciones:

$$[7 < key1 < 10]$$
$$[0.9 < key2 < 3] \text{ y } |key2 * 100\int f = 150$$
$$[0 < key3 < 3] .$$

4.Usar las claves key1 y key2 para mezclar los valores de la matriz A y crear nuevas matrices.

```
A1=key1*max(A)-A;
A2=key1*max(A1)-A1;
B1=A1-A2;
B2=key2*B1+A2;
```

5. Calcular la SVD de las últimas dos matrices obtenidas en el paso anterior (B1 y B2).

```
[UB1, SB1, VB1] = svd(B1);
[UB2, SB2, VB2] = svd(B2);
```

6.Construir dos nuevas matrices intercambiando los va- lores singulares (SB1) de B1 con los valores singulares (SB2) de B2.

```
C1 = UB1*SB2*VB1';
C2 = UB2*SB1*VB2';
```

7.Para mejorar la calidad de la encriptación se repetirán los pasos anteriores. Primero, se deben mezclar los va- lores de las matrices C1 y C2 usando la clave key3 para obterner dos matrices nuevas (D1 y D2).

```
D1=C1-C2;
D2=key3*D1+C2;
```

Luego, calcular la SVD a la matrices resultantes (D1 y D2) y construir dos nuevas matrices intercambian- do los valores singulares (SD1) de D1 con los valores singulares (SD2) de D2.

```
[UD1, SD1, VD1] = svd(D1);
[UD2, SD2, VD2] = svd(D2); E1=UD1*SD2*-
VD1'; E2=UD2*SD1*VD2';
```

8.Combinar E1 y E2 para crear una matriz aumentada.

```
F=[E1 E2];
```

9. Aplicar la siguiente transformación lineal a la matriz F:

$$FF = F - MI$$
$$MA \quad MI$$

donde MA es el numero máximo en la matriz F y MI es el numero mínimo en la matriz F.

10. Escoger 8 entradas de la matriz FF de forma aleatoria.

11. Guardar en Posicion1 el valor de FF en la primera en- trada que se escogió de forma aleatoria en el paso an- terior. Si el signo de MI es positivo, entonces guardar el numero 0.0 dicha entrada, sino entonces guardar el numero 0.1.

```
if MI>0
Posicion1=FF(Entradax1,Entraday1); FF(En-
tradax1,Entraday1)=0.0;
else
Posicion1=FF(Entradax1,Entraday1); FF(En-
tradax1,Entraday1)=0.1;
end
```

12.Guardar en Posicion2 el valor de FF en la segunda en- trada que se escogió de forma aleatoria en el paso 10. Guardar en dicha entrada el siguiente valor:

$$\frac{,|| MI |\int ,}{\frac{255}{10000}}$$

```
Posicion2=FF(Entradax2,Entraday2); FF(En-
tradax2,Entraday2)=(floor((floor(
abs(MI))/255)))/10000;
```

13.Guardar en Posicion3 el valor de FF en la tercera en- trada que se escogió de forma aleatoria en el paso 10. Guardar en dicha entrada el siguiente valor:

$$\frac{residuo(\ MI\ ,\ 255)}{10000}$$

14.Guardar en Posicion4 el valor de FF en la cuarta en- trada que se escogió de forma aleatoria en el paso 10. Guardar en dicha entrada el siguiente valor:

$$|\ MI\ | - || MI |\int$$

```
Posicion4=FF(Entradax4,Entraday4); FF(En-
tradax4,Entraday4)=abs(MI)-floor(
abs(MI));
```

15.Guardar en Posicion5 el valor de FF en la quinta entra- da que se escogió de forma aleatoria en el paso 10. Si el signo de MA es positivo, entonces guardar el nume- ro 0.0 dicha entrada, sino entonces guardar el numero 0.1.

16. Guardar en Posicion6 el valor de FF en la sexta entrada que se escogió de forma aleatoria en el paso 10. Guar- dar en dicha entrada el siguiente valor:

$$\frac{,|| MA |\int ,}{\frac{255}{10000}}$$

17. Guardar en Posicion7 el valor de FF en la séptima en- trada que se escogió de forma aleatoria en el paso 10. Guardar en dicha entrada el siguiente valor:

$$\frac{residuo(\ MA\ ,\ 255)}{10000}$$

18. Guardar en Posicion8 el valor de FF en la octava en- trada que se escogió de forma aleatoria en el paso 10. Guardar en dicha entrada el siguiente valor:

$$|\ MA\ | - || MA |\int$$

19. Guardar la imagen (matriz) obtenida en el paso 18. La matriz obtenida en el paso anterior es la matriz de la imagen encriptada.

20. Guardar en un archivo de texto las claves key1, key2 key3, Entradax1, Entraday1, Entradax2, Entraday2, Entradax3, Entraday3, Entradax4, Entraday4, Entra- dax5, Entraday5, Entradax6, Entraday6, Entradax7, Entraday7, Entradax8, Entraday8, Posicion1, Posi- cion2, Posicion3, Posicion4, Posicion5, Posicion6, Posicion7, Posicion8.

Algoritmo de desencriptación.

1. Leer la imagen encriptada.

```
Im=imread('C:\desencript\imagen.jpg');
```

2. Almacenar la imagen en formato double.

```
FF1=double(Im);
```

3.Leer el archivo de texto que se guardó en el paso 20 del algoritmo de encriptación.

4. Recuperar usando las claves del paso anterior, el nu- mero máximo (MA) y el numero mínimo (MI) de la matriz F del paso 8 en el algoritmo de encriptación.

```
if FF1(Entradax1,Entraday1)==0.0
   signo1=1;
else
   signo1=-1;
end

MI1=signo1*(((FF1(Entradax2,Entraday2)
    *10000)*255)+(FF1(Entradax3,
    Entraday3)*10000)+FF1(Entradax4,
    Entraday4));

if FF1(Entradax5,Entraday5)==0.0 signo2=1;
else
signo2=-1; end

MA=signo2*(((FF1(Entradax6,Entraday6)
*10000)*255)+(FF1(Entradax7,

Entraday7)*10000)+FF1(Entradax8,

Entraday8));
```

5.Recuperar la matriz FF del paso 8 del algoritmo de en- criptación.

```
FF1(Entradax1,Entraday1)=Posicion1;
FF1(Entradax2,Entraday2)=Posicion2;
FF1(Entradax3,Entraday3)=Posicion3;
FF1(Entradax4,Entraday4)=Posicion4;
FF1(Entradax5,Entraday5)=Posicion5;
FF1(Entradax6,Entraday6)=Posicion1;
FF1(Entradax7,Entraday7)=Posicion7;
```

```
FF1(Entradax8,Entraday8)=Posicion8;
```

6.Recuperar la matriz F del paso 8 del algoritmo de en- criptación.

```
F1=(FF1*(MA1-MI1))+MI1;
```

7.Recuperar la matriz E1 y la matriz E2 del paso 7 del algoritmo de encriptación. Recuérdese que de acuerdo al paso 8 del algoritmo de encriptación, las matrices E1 y E2 forman la matriz F.

8.Calcular la descomposición en valores singulares de las matrices E_1 y E_2 y usar la SVD para recupe- rar las matrices D1 y D2 del paso 7 del algoritmo de encriptación.

```
[UE_1,SE_1,VE_1]=svd(E_1);
[UE_2,SE_2,VE_2]=svd(E_2);
D_1=UE_1*SE_2*VE_1'; D_2=UE_2*SE_1*VE_2';
```

9.Recuperar las matrices C1 y C2 del paso 6 del algorit- mo de encriptación.

```
C_2=D_2-key3*D_1;
C_1=D_1+C_2;
```

10.Calcular la descomposición en valores singulares de las matrices C_1 y C_2 y usar la SVD para recupe- rar las matrices B1 y B2 del paso 4 del algoritmo de encriptación.

```
[UC_1,SC_1,VC_1]=svd(C_1);
[UC_2,SC_2,VC_2]=svd(C_2);
B_1=UC_1*SC_2*VC_1';
B_2=UC_2*SC_1*VC_2';
```

11.Recuperar las matrices A1 y A2 del paso 4 del algorit- mo de encriptación.

```
A_2=B_2-B_1*key2;
A_1=B_1+A_2;
```

12.Recuperar la matriz A del paso 2 del algoritmo de en- criptación.

```
AA=((key1*min(A_1))/(key1-1))-A_1;
```

13. Guardar la imagen (matriz) obtenida en el paso 12. La matriz obtenida en el paso anterior es la matriz de la imagen desencriptada.

Resultados

Se hicieron pruebas del algoritmo de compresión y para la validación de los resultados, se compararon los tamaños en bytes de las imágenes de entrada y de salida.

En la Fig. 1 y Tabla I se muestran los resultados del algoritmo de compresión con una imagen de 480 750 píxeles. La Fig. 1(a) muestra la imagen original la cual tiene un tamaño de almacenamiento de 121727 bytes. En primer lugar, se hizo la compresión de la imagen original tomando solo la cantidad de 80 valores singulares, la imagen resultante se muestra en la Fig. 1(b)), las imágenes mostradas en las Fig. 1(c) y 1(d)están comprimidas tomando la cantidad de 170 y 300 valores singulares respectivamente.

(a) Original **(b) BlockSize = 80**

(c) BlockSize = 170 **(d) BlockSize = 300**

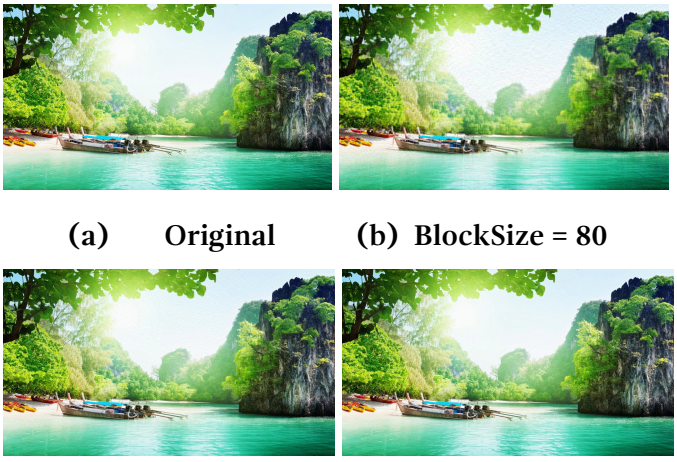
Fig. 1. Imagen de 480×750 pixeles.

Al usar 80 valores singulares, la calidad de la imagen disminuye de manera considerable, como se muestra en la Fig. 1(b)), con un tamaño de tan solo 86777 bytes, mientras que al usar 170 y 300 valores singulares, la calidad de la imagen mejora. En todos los casos disminuyó el tamaño en bytes. En la Fig. 2 y Tabla II se muestran los resultados del algoritmo de compresión con una imagen de 1080 x1920 píxeles. La imagen original se muestra en la Fig. 2(a) y

Tamaño original	BlockSize	Tamaño de salida
121727 bytes	80	86777 bytes
	170	99965 bytes
	300	108056 bytes

Tabla I. Tamaño en bytes de imágenes comprimidas.

tiene un tamaño de almacenamiento de 553521 bytes. La Fig. 2(b) muestra la imagen resultante de comprimir la imagen original tomando solo la cantidad de 80 valores singulares, las imágenes mostradas en las Fig. 2(c) y 2(d) son el resultado de comprimir la imagen original tomando la cantidad de 170 y 300 valores singulares respectivamente.



(a) Original **(b) BlockSize = 80**

(c) BlockSize = 170 **(d) BlockSize = 300**

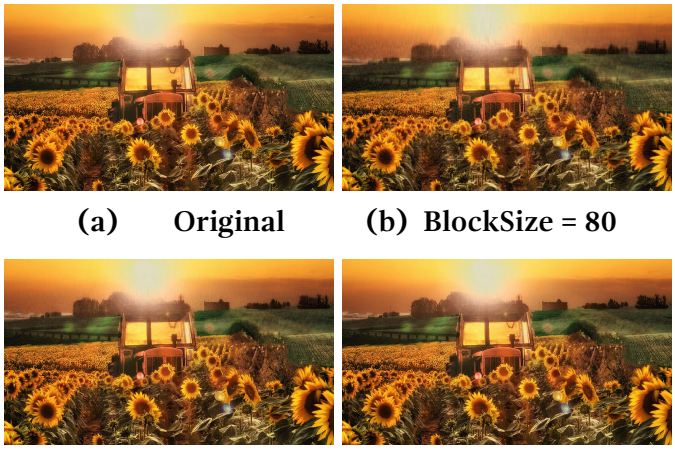
Fig. 2. Imagen de 1080×1920 pixeles.

Tamaño original	BlockSize	Tamaño de salida
553521 bytes	80	302023 bytes
	170	348055 bytes
	300	373764 bytes

Tabla II. Tamaño en bytes de imágenes comprimidas.

Como en el caso de la imagen mostrada en la Fig. 1, al usar 80 valores singulares, la calidad de la imagen disminuyó de manera considerable (Fig. 2(b)), así como su tamaño en bytes (302023 bytes). Por otro lado, al usar 170 y 300 valores singulares, la calidad de la imagen mejora. Es importante destacar que en todos los casos disminuyó el tamaño en bytes.

En la Fig. 3 y Tabla III se muestran los resultados del algoritmo de compresión con una imagen de 1080 1920 píxeles. La imagen original está mostrada en la Fig. 3(a) y tiene un tamaño de almacenamiento original de 713914 bytes. Las imágenes mostradas en las Fig. 3(b), 3(c) y 3(d) están comprimidas tomando en cuenta la misma cantidad de valores singulares de las imágenes anteriores (Fig. 1 y Fig. 2).
Al usar 80 valores singulares, la calidad de la imagen disminuyó de manera considerable (Fig. 3(b)) y quedó con un tamaño de 295307 bytes, mientras que al usar 170 y 300.



(a) Original **(b) BlockSize = 80**

(c) BlockSize = 170 **(d) BlockSize = 300**

Fig. 3. Imagen de 1080×1920 pixeles.

Tamaño original	BlockSize	Tamaño de salida
713914 bytes	80	295307 bytes
	170	333480 bytes
	300	348632 bytes

Tabla III. Tamaño en bytes de imágenes comprimidas.

valores singulares, la calidad de la imagen mejoró y en todos los casos disminuyó el tamaño en bytes.

Se hicieron pruebas del algoritmo de encriptación con varias imágenes a colores. La Fig. 4 muestra los resultados obtenidos. Primero, se comprimió la imagen original con un BlockSize de 300 (columna 2) luego, se encriptó la imagen comprimida (columna 3) y por último, siguiendo los pasos del algoritmo de desencriptación se desencriptó la imagen de la columna 3.

En todos los casos la encriptación visual fue buena y la ima- gen de salida del algoritmo de desencriptación no perdió in- formación perceptible al ojo humano de la imagen que se

encriptó.

Conclusiones

En este trabajo se implementó un algoritmo de compresión y encripatación de imágenes digitales usando como base la descomposición en valores singulares SVD.

En la comparativa visual de los resultados, se observó que la imagen encriptada es, visualmente, muy diferente a la imagen original y que el algoritmo de encriptación y desencriptación trabaja muy bien con imágenes a colores.

De acuerdo a la comparativa cualitativa de los resultados del algoritmo de compresión, podemos concluir que con la ma- nipulación de los valores singulares se puede comprimir un buen porcenaje de la imagenes sin reflejar distorsiones vi- suales significativas. Finalmente, el algoritmo de desencrip- tación propuesto logró desencriptar las imágenes sin perder calidad e información de las imágenes encriptadas.

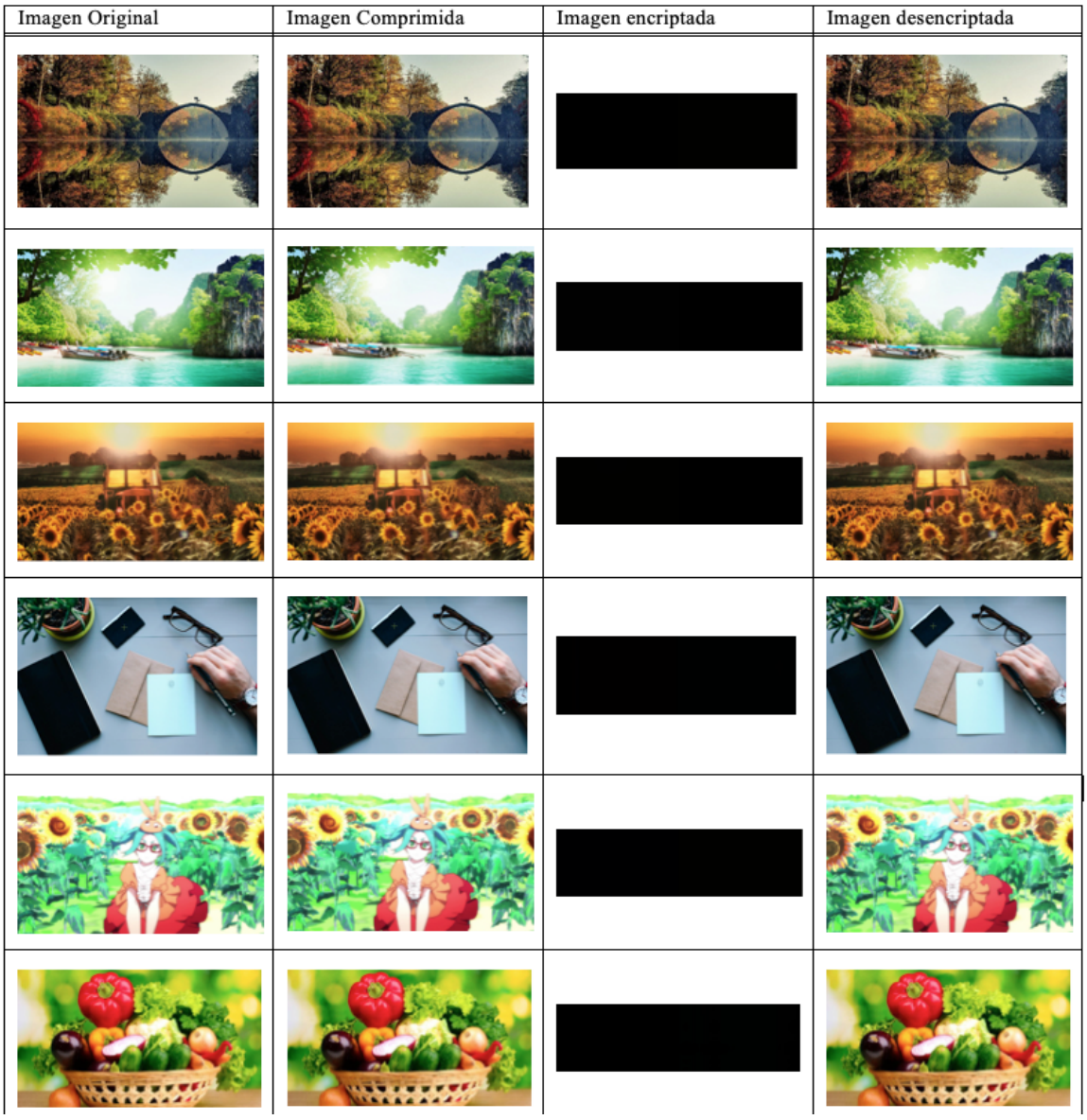


Fig. 4. Pruebas de compresión y encriptacion.