Compilador de colores utilizando SVD

Joaquín Farias, *Estudiante*

Resumen—En Este informe se presenta una aplicación para compilar los colores de una imagen utilizando SVD. Se explicará la transformación, además de mostrar los resultados de pruebas realizadas. También se hará un análisis del error con distintos valores de porcentaje de valores singulares ocupados.

Palabras clave—Compilador, RGB, SVD, Valores singulares.

1 Introducción

La transformación mediante *Singular Value Decompostion (SVD)* es ampliamente utilizada para comprimir datos, así como también para el *machine learning* y estadística. Esta se ocupa en conjunto con el *análisis de componentes principales (PCA,* por sus siglas en ingles).

El *PCA* es una técnica utilizada para describir un conjunto de datos en términos de componentes que no estén correlacionadas. Estas variables se ordenan mediante la varias original que describen para así darles cierto peso dentro de los datos. Luego, se ocupan las variables con más peso para describir los datos y así lograr una compresión de estos con una representación fidedigna.

.

2 Motivación y descripción del algoritmo de SVD y PCA

El programa realizado es una aplicación para compilar colores de imágenes. La motivación para realizar este proyecto es que el análisis de imágenes para extraer sus características tenga un menor requerimiento computacional, disminuyendo la información dentro de la imagen, pero sin destruir la esencial que nos interesa. Además, también para disminuir la memoria que ocuparía una imagen guardada en un equipo de almacenamiento.

El programa consta de tres funciones principales:

1. **Descomposición con SVD y compresión.**
2. **Descompresión**.
3. **Cálculo de error**.

Estas funciones dependen del valor de entrada del programa, este valor sería el porcentaje de valores singulares a ocupar en la compresión.

2.1 Función de descomposición con SVD y compresión

Esta función descompone las matrices RGB con SVD, así quedan las siguientes matrices que describen ahora a la imagen:

$$IMG_{ORIGINAL} = U \cdot \Sigma \cdot V^t$$

Donde la matriz $IMG_{ORIGINAL}$ es de dimensiones mxn, la matriz U de dimensiones mxm, la matriz V de dimensiones nxn y la matriz Σ de dimensiones min(m,n).
Se nota que la matriz Σ contiene los valores singulares de la matriz $IMG_{ORIGINAL}$ y las otras matrices sus vectores propios.
Ahora, el valor ingresado en pantalla le informa al programa cuando valores singulares debe ocupar para realizar la compresión de la imagen. Esto quiere decir que la matriz Σ se redimensionara a dimensiones kxk, donde k es el valor ingresado en pantalla. Mientras que las matrices U y V también cambiaran sus dimensiones a kxm y kxn, respectivamente.
Con esto témenos la imagen compilada en tres matrices con dimensiones menores a las originales RGB.

2.2 Función de descompresión

Esta función se encarga descomprimir o reconstruir la imagen por medio de las reconstrucciones de las matrices RGB. Luego, estas se concatenan y se forma la imagen comprimida lista para ser mostrada en pantalla.

2.3 Función de cálculo de error de imagen recompilada

Esta función se encarga de calcular el error entre la imagen recompilada y la imagen original. Esto lo logra obteniendo el error relativo en cada vector RGB dentro la matriz de pixeles, es decir, se resta le resta la matriz de la imagen recompilada a la imagen original, luego se divide por 256 (número de bits que tiene cada vector RGB). Luego de esto, se suman los errores de todos los pixeles y se obtiene el promedio aritmético de errores.

También se hará una gráfica que diga exactamente en qué pixeles es más grande el error mediante un mapa de colores.

3 Código

En este capítulo se describirá de manera concisa los códigos principales de cada función ocupada.

El código principal de nuestro programa pide que se ingrese el valor del porcentaje de valores singulares.

3.1 Código de función de compilador de imagen

En la *fig. 1* se puede apreciar el código empleado para esta función. Así, primero luego de hacer la transformación con SVD, se empequeñecen las matrices E, U y V dependiendo del porcentaje de valores singulares que se ingresó. Esto se hacer calculando el número real de valores singulares a ocupar y limitando el rango de las matrices

```
Erc = Er(1:val_sing, 1:val_sing);
Urc = Ur(:,1:val_sing);
Vrc = Vr(:,1:val_sing);

Egc = Eg(1:val_sing, 1:val_sing);
Ugc = Ug(:,1:val_sing);
Vgc = Vg(:,1:val_sing);

Ebc = Eb(1:val_sing, 1:val_sing);
Ubc = Ub(:,1:val_sing);
Vbc = Vb(:,1:val_sing);

U = cat(3, Urc, Ugc, Ubc);
E = cat(3, Erc, Egc, Ebc);
V = cat(3, Vrc, Vgc, Vbc);
```

Fig. 1. Código de la compresión de la imagen.

con este.

3.2 Código de función de descomprimido de imagen

En la *fig. 2* se puede apreciar el código empleado para esta función, que es esencialmente la multiplicación de las matrices V, E y U para la reformación de las matrices RGB, luego de concatenan estas matrices para dejar la imagen lista para mostrarse en pantalla.

```
R = U(:,:,1) * E(:,:,1) * (V(:,:,1))';
G = U(:,:,2) * E(:,:,2) * (V(:,:,2))';
B = U(:,:,3) * E(:,:,3) * (V(:,:,3))';

%se calcula la matriz de salida que contiene las matrices U, E y V de las
%matrices RGB.
IMG_RECON = uint8(cat(3, R, G, B));
```

Fig. 2. Código de función de descomprimido de imagen.

3.3 Código de función de cálculo de error de imagen recompilada

En la *fig. 3* se puede apreciar el código empleado para esta función, que básicamente calcula el error relativo en cada posición de la matriz **IMG_RECON,** para luego calcula un promedio de errores de toda la matriz.

Además, se hace un mapa de colores con los errores para ver de forma más ilustrativa estos errores.

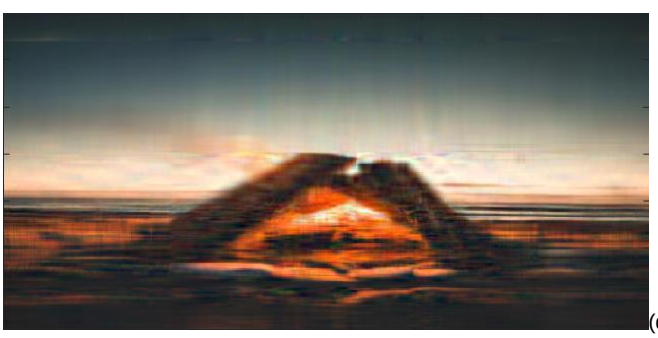
```
%calculo de errores en cada elemento de las matrices RGB
img_restaR = abs(R - Rc);
img_restaG = abs(G - Gc);
img_restaB = abs(B - Bc);
for i=1:size(IMG_ORIGINAL,1)
        for j=1:size(IMG_ORIGINAL,2)
            img_erroresR(i,j) = img_restaR(i,j) / 256;
            img_erroresG(i,j) = img_restaG(i,j) / 256;
            img_erroresB(i,j) = img_restaB(i,j) / 256;
        end
end

%calculo de error promedio en cada vector RGB
for i=1:size(IMG_ORIGINAL,1)
        for j=1:size(IMG_ORIGINAL,2)
            img_prom_erroresRGB(i,j) = img_erroresR(i,j) +
            img_erroresG(i,j) + img_erroresB(i,j);
        end
end
img_prom_erroresRGB(:,:) = img_prom_erroresRGB(:,:)/3;
```

Fig. 3. Código de función de descomprimido de imagen.

(a)

(b)

(c)



(d)

Fig. 4 Imagen recompilada con distintos porcentajes de valores singulares. (a) Imagen original. (b) 10%. (c) 5%. (d) 1%.

4 Pruebas y análisis

Dentro de este capítulo se presentarán los resultados de la aplicación de nuestro programa, el análisis del tiempo de ejecución de las distintas funciones y el análisis de error.

4.1 Pruebas, resultados y ejemplos

Los resultados de pruebas hechas con una imagen se mientras en la *fig. 4* donde se aprecia cómo cambia la imagen con distintos porcentajes de uso de valores singulares. Se nota que a menor el valor menos resolución tienen los colores de la foto y, por lo tanto, más alejada esta de la original.

4.2 Análisis de error

Se nota por el grafico de la *fig. 5* que a medida que disminuye el porcentaje de valores singulares que se ocupa, el error aumenta. Además, la *fig. 6* muestra el cómo varia este error dependiendo de la cantidad de componentes principales que se ocupen para comprimir la imagen.

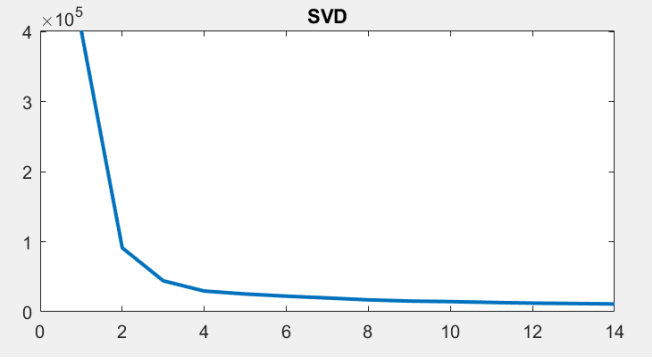


Fig. 5. Grafica del error respecto a los valores singulares ocupados para la compresión de la imagen.

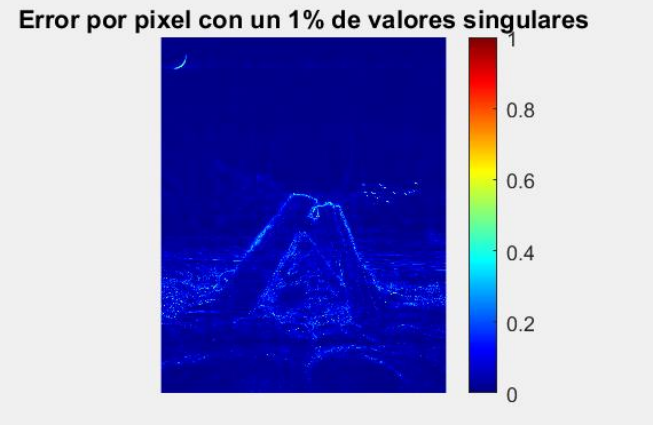
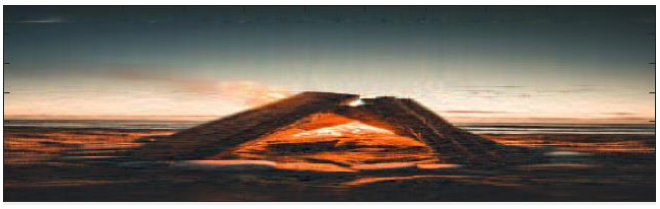


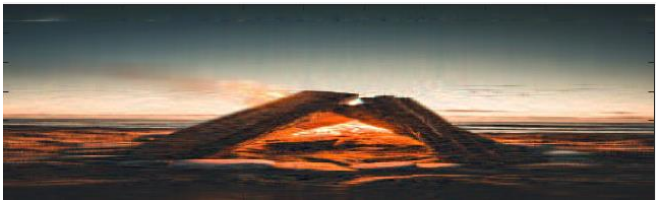
Fig. 3. Gráfico de mapeo de colores que representa el error con 1% de valores singulares ocupados.

4.3 Análisis con otro formato de colores

Se analiza cómo reacciona el programa si se ocupa otro formato de colores, en este caso CIElab. En la *fig. 9* se puede ver que los resultados entre ocupar RGB o CIElab son muy parecidos y no varía mucho el error de las imágenes compiladas.



(a)



(b)

Fig. 3. Comparación de compilación de imágenes RGB y LAB con SVD. (a) Imagen RGB comprimida. (b) Imagen LAB comprimida.

5 Conclusiones

Los resultados obtenidos están dentro de lo esperado de la ejecución de esta aplicación. Además, se nota que se puede ocupar este método de compresión de imágenes para distintos formatos de colores. Y que es bastante beneficioso a la hora de comprimir, ya que con muy pocos valores singulares se puede llegar a una buena solución, es decir ocupando matrices con menor rango y menos datos.

Referencias

[1] Boyd, S., & Vandenberghe, L. (2018). Introduction to applied linear algebra: vectors, matrices, and least squares. Cambridge university press.

[2] Norvig, P., & Russell, S. (2004). Inteligencia artificial. Editora Campus, 20.)

Joaquín Farias Muñoz Estudiante de ingeniería civil electrónica, Pontificia Universidad catolica de Valparaíso.