Métodos Numéricos Avanzados: Trabajo Práctico Nº1

Autovalores y compresión de imágenes

Juan José Marinelli - 50231
Lucas Moscovicz - 52126
Agustín Pagnoni - 52118
Gabriel Zanzotti - 52067

1. Resumen

El trabajo realizado consiste en mostrar los resultados obtenidos al comprimir una imagen de diversas formas. Los métodos utilizados para la compresión son “imagen media” y “PCA”, al cual se le aplican distintas variaciones para analizar y comparar los resultados obtenidos.

Palabras clave: métodos numéricos, método QR, compresión de imágenes, autovalores, autovectores.

2. Introducción

Los temas a tratar en este proyecto son los métodos de imagen media y PCA mediante los cuales se hace la compresión de la figura 1. Las implementaciones de PCA fueron realizadas utilizando las siguientes variantes. La primera realiza el cálculo de los autovalores y autovectores realizando comparaciones de los resultados obtenidos comprimiendo la imagen con distintas cantidades de autovectores. La segunda aplica el método QR con traslaciones para obtener un único autovector utilizándolo para realizar el cálculo. Finalmente se realiza una tercera variante en la cual no se utilizan elementos de punto flotante con el fin de realizar una compresión aún mayor de la imagen usando enteros representados en menor cantidad de bits de los que se usan para la representación en punto flotante.

En la sección de metodología se pueden ver los procedimientos utilizados para los distintos métodos y en la sección de resultados una comparación de las imágenes comprimidas resultantes.

Figura 1 – Imagen utilizada para el proyecto

3. Metodología

3.1. La imagen media

```
a = imread('lena512.bmp');
imshow(a) % muestro la imagen original
b = im2col(a,[16,16],'distinct');
m = mean(double(b.'));
m = m.';
M = repmat(m,1,1024);
d = col2im(M,[16,16],[512,512],'distinct');
meanimg = uint8(round(d));
imshow(meanimg) % muestro la imagen ''media''
```

Figura 2 - código extraído del enunciado para la compresión

```
1  a = imread('lena512.bmp');
2  %imshow(a) % muestro la imagen original
3  b = im2col(a,[16,16],'distinct');
4  m = mean(double(b));
5  m = m.';
6  M = repmat(m,1,256);
7  d = col2im(M',[16,16],[512,512],'distinct');
8  meanimg = uint8(round(d));
9  imshow(meanimg) % muestro la imagen ''media''
```

Figura 3 - código final utilizado para comprimir la imagen

3.1.1 - Pasos que ejecuta el código

1- Lee la imagen y la guarda en la variable ‘a’

2- Como se muestra en la línea 3 de la figura 3 im2col agarra la imagen y la divide en bloques de 16x16 los cuales los ubica como columnas de tamaño 256 en la matriz ‘b’, el parámetro ‘distinct’ indica que antes de separar en bloques se completa la matriz con 0’s para que no haya solapamiento de bloques.

3- Llegamos a la conclusión de que trasponer la imagen como se muestra en la línea 4 de la figura 2 sería incorrecto, ya que la función ‘mean’ hace los promedios por columna, que en caso de trasponer la matriz ‘b’ estaría dando mal ya que la matriz b (256*1024) al trasponer resulta ser de 1024*256, entonces lo que estaría devolviendo su promedio serian en realidad 256 promedios. Pero lo que necesitamos hacer es en realidad el promedio de los 1024 bloques de la imagen por eso esto es modificado en la figura 3.
Al no trasponerlo haría lo que queremos, que es achicar cada bloque por el promedio de sus pixeles.

5- Se traspone la matriz “m” para que los promedios queden separados en filas.

6- Se repiten los valores de cada fila 256 veces para generar cada uno de los bloques iguales.

7- Se traspone M antes de aplicarle col2im ya que el resultado luego de modificar el algoritmo de forma correspondiente será una matriz de 1024x256 en lugar de una de 256x1024.

3.2 PCA

3.2.1 Único autovector

```
1  function out = autoimage(autoimg)
2      a = imread('lena512.bmp');
3      b = im2col(a, [16 16], 'distinct');
4      m = mean(double(b));
5      m = m.';
6      M = repmat(m, 1, 256)';
7      ds = double(b)-M;
8      cc = cov(double(b'));
9      [V,D] = eig(cc);
10     % ordeno los autovalores de mayor a menor
11     D = diag(D);
12     [D,i] = sort(D,'descend');
13     D = diag(D);
14     V = V(:,i);
15     % muestro la primera ''autoimagen''
16     c = (V(:,autoimg)-min(V(:,autoimg)))*256/(max(V(:,autoimg))-min(V(:,autoimg)));
17     dv1 = col2im(c,[16,16],[16,16],'distinct');
18     dv1 = uint8(round(dv1));
19     imshow(dv1);
20     % proyecto la primer autoimagen
21     % me queda la proyeccion en cada fila (producto interno)
22     pv1 = V(:,autoimg).'*ds;
23     % si solamente considero la primer autoimagen...
24     d = M;
25     for k = 1:1024
26       d(:,k) = d(:,k) + pv1(k)*V(:,autoimg);
27     end
28     % d es una matriz de 1024x256
29     compimg = col2im(d,[16,16],[512,512],'distinct');
30     compimg = uint8(round(compimg));
31     imshow(compimg)
32 end
```

Figura 4 – Código utilizado tanto para un autovector como para muchos

Haremos un seguimiento del código utilizado en la figura 4 para mostrar su funcionamiento.

1 - ds = double(b)-M;
En esta línea se resta a cada pixel imagen el valor promedio del bloque al que pertenece.

2 - cc = cov(double(b));
Se calcula la matriz covarianza la cual tendrá en cada posición *Xij* donde *Xij* es la covarianza entre el nivel de gris *Xi* y el *Xj.*

3 - [V,D] = eig(cc);
Se obtienen los autovalores y autovectores de la matriz de covarianza en las matrices V y D respectivamente.
La matriz D es diagonal y los elementos de la misma son los autovalores de cc.

4 - D = diag(D);
Se obtienen los autovalores en la diagonal de D como un vector.

5 - [D,i] = sort(D,'descend');
Se ordena el vector D y obtengo en i las posiciones en las cuales quedo cada autovalor luego del ordenamiento (se usa para luego ordenar los autovectores de forma que se correspondan con su respectivo autovalor en orden)

6 - D = diag(D);
Se vuelven a colocar los autovectores como diagonal de la matriz D

7 - V = V(:,i);
Usando i (orden de los autovalores), se ordenan los autovectores de la matriz V de forma que se correspondan, como fue mencionado anteriormente, con su respectivo autovalor.

Las siguientes 4 líneas de código, no afectan al algoritmo en general. Pero son usadas para mostrar la 'autoimagen' correspondiente a un autovector calculado. Se decidió modificar el código para poder parametrizar el autovalor a usar en lugar de que este fuese siempre el primero, el parámetro usado es el que toma el nombre de "autoimg".

8 - c = (V(:,autoimg)-min(V(:,autoimg)))*256/(max(V(:,autoimg))-min(V(:,autoimg)));
9 - dv1 = col2im(c,[16,16],[16,16],'distinct');
10 - dv1 = uint8(round(dv1));
11 - imshow(dv1)

[ver Figuras 5.1 5.2 5.3 5.4 y 5.5]

Dichas 4 líneas toman el autovector deseado de la matriz V, normalizan sus valores entre 0 y 256 (le restan el mínimo valor del autovector a todos los valores del mismo, luego lo dividen por el tamaño del intervalo al cual pertenecen sus valores para que todos los valores queden entre 0 y 1 y finalmente se multiplican los valores resultantes por 256 para expresar dicho autovector como valores en un intervalo [0:256]). Luego se pasan estos 256 valores a una matriz de 16x16 y se redondean. Finalmente se muestran como una imagen de 16x16 bloques de un determinado color.

12 - pv1 = V(:,autoimg).'*ds;

La siguiente línea realiza la proyección del autovector seleccionado sobre cada columna de la matriz ds la cual representa un bloque al cual se le resto el color promedio del mismo, es decir, cada elemento de dicha columna tendrá como valor la diferencia entre su propio color y el color promedio del bloque al que pertenece. Al ser una proyección, en pv1 obtendremos un vector en el cual tendremos en cada posición un

número que representará el producto interno entre el autovector y el bloque sobre el cual lo estamos proyectando.

13 - d = M;
14 - for k = 1:1024
15 - d(:,k) = d(:,k) + pv1(k)*V(:,autoimg);
16 - end

Estas líneas tienen como función agregar a cada columna de d (en la cual están todos los elementos de un bloque los cuales contienen como valor el color promedio de dicho bloque) la proyección del autovector elegido sobre dicho bloque multiplicada por el autovector.

17 - compimg = col2im(d,[16,16],[512,512],'distinct');
18 - compimg = uint8(round(compimg));
19 - imshow(compimg)

Finalmente, se vuelve a representar la imagen como una matriz de 512x512 pixeles, se redondean los valores y se muestra la imagen final.

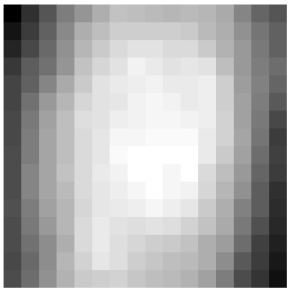

Figura 5.1 - autoimagen 1

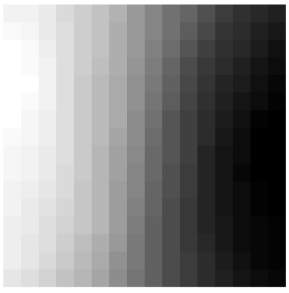

Figura 5.2 - autoimagen 2

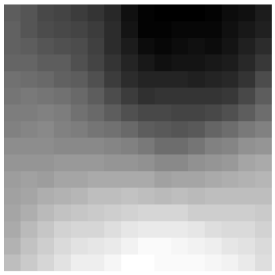

Figura 5.3 - autoimagen 3

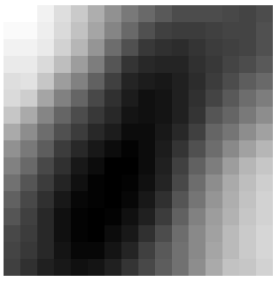

Figura 5.4 - autoimagen 4

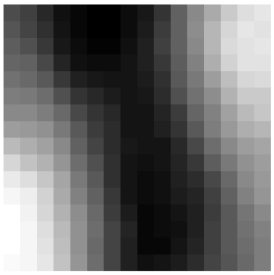

Figura 5.5 - autoimagen 5

3.2.2 Múltiples autovectores

Para esta implementación se cambió la siguiente porción de código:

```
14 - for k = 1:1024
15 -     d(:,k) = d(:,k) + pv1(k)*V(:,autoimg);
16 - end
```

por:

```
14 - for k = 1:1024
15 -     for j = autoimg
16 -         pv1 = V(:,j).'*ds;
17 -         d(:,k) = d(:,k) + pv1(k)*V(:,j);
18 -     end
19 - end
```

Esto se utilizó para poder agregar a cada bloque la proyección de más de un autovector sobre dicho bloque.
Cada autovector que se agrega provee mayor nivel de detalle a la imagen ya que es equivalente a tomar un conjunto generador más completo. Como una nota al margen, pudimos observar que al tomar todos los 256 autovectores (es decir, la base de R 256), la imagen resultante fue idéntica a la imagen provista por la cátedra.

3.2.3 Método QR con traslaciones

```
function T=tridiagonalize(A)
  dim = length(A(1,:))
  H = A;
  for i=1:dim-2
    x = A(:,i);
    first_n = eye(dim,i+1)*ones(i+1,1);
    y = x.*first_n;
    s = sign(x(i+1)) * norm(x(i+1:dim));
    y(i+1) = s;
    w = (x-y)/norm(x-y);
    P = eye(dim) - 2*w*w';
    A = P*A*P;
  end
  T = A
end

```

Figura 6 – Función de tridiagonalización

```
function [V,D]=qr_with_shifts(A)
  T = tridiagonalize(A);
  dim = length(A(1,:));
  tol = 0.001;
  Qavec = eye(dim);
  D = zeros(dim);

  for m=dim:-1:2
    while(A(m-1,m) > tol)
      [Q,R] = qr(A - A(m,m)*eye(m));
      A = R*Q + A(m,m)*eye(m);
    end
    Qavec(1:m, 1:m) = Qavec(1:m,1:m)*Q;
    D(m,m) = A(m,m);
    A = A(1:m-1,1:m-1);
  end
  D(1,1) = A(1,1);
  V = Qavec;
end

```

Figura 7 – Función del método QR con traslaciones

La función "tridiagonalize" de la figura 6 fue creada para obtener la matriz tridiagonal y la implementación sigue los pasos de Householder. Una vez con la matriz tridiagonal, pasamos a obtener la descomposición QR con traslación utilizando el código de la figura 7, en la cual en cada paso va acumulando los autovalores, y además opera con la submatriz de Q para que al finalizar, en las columnas de Q queden los autovectores. Finalmente se utilizó nuestro método para comprimir la imagen la cual queda sin diferencias aparentes con respecto a cuándo se utiliza la función "eig".

3.2.4 Método sin punto flotante

Para esta sección del proyecto, detallaremos cómo realizar la compresión de una imagen y luego comunicar la misma sin utilizar números de punto flotante.

El primer paso del proceso es realizar la compresión básica de la imagen usando 'la imagen media', proceso realizado en la primer parte de este informe.
Una vez hecho esto, se obtiene la matriz de covarianza de los niveles de gris y sus autovalores y autovectores al igual que en la segunda parte de este informe.
Los elementos de estos autovalores y autovectores están en formato punto flotante. La idea principal de esta sección será comunicar los 5 autovectores principales sin necesidad de usar números de punto flotante.
Para lograr esto, se realizó un mapeo de todos los valores de cada autovector a un número entero en el intervalo 0-255 para poder enviar cada valor usando un byte en lugar de 4 como en el caso del punto flotante.

Para reconstruir cada autovector se necesita el valor máximo y mínimo del mismo. Y dado que estos estaban representados también en punto flotante, se decidió aplicar un segundo algoritmo para representarlos como números enteros sin una pérdida considerable de precisión.
El algoritmo elegido fue elevar dichos valores al cuadrado, calcular su inversa y redondearlos a enteros de 16 bits. En este caso se usaron 16 bits para una menor pérdida de precisión ya que solamente se debían calcular 2 valores por cada autovector y no es una cantidad de espacio considerable respecto del autovector en sí.

El algoritmo usado para ello se puede ver en la figura 8:

```
X = V(:, autoimg);
A = zeros(length(autoimg), 2);
for i = 1:length(X(1,:))
    A(i, 1) = max(X(:,i));
    A(i, 2) = min(X(:,i));
    X(:,i) = uint8((X(:,i)-A(i,2)).*255./((A(i,1))-A(i,2)));
end

% Modificamos los maximos y minimos de cada autovector para pasarlos
% como enteros de 16 bits. Se elevan al cuadrado para minimizar el error
% de redondeo.
A = int16(1./(A.^2)).*sign(A);
```

Figura 8 – Función utilizada para realizar la compresión sin punto flotante

Para la reconstrucción de la imagen, se aplica el proceso inverso primero a los máximos y mínimos de cada autovector, es decir, se toma la raíz cuadrada de la inversa de dichos valores.

Luego, usando los máximos y mínimos calculados se convierten los autovectores a una aproximación de su rango original.

4. Resultados

Las imágenes de la figura 9 corresponden a la compresión usando 1, 2, 3, 4 y 5 autovectores correspondientemente.
Se observa que al usar mayor número de autovectores mejora la calidad de la imagen, y habiendo usado los primeros 5 autovectores (es decir, los correspondientes a los mayores autovalores), la imagen resulta muy similar a la original ya que estos son los que mayor cantidad de información contienen sobre la misma.

Figuras - 9.1, 9.2, 9.3, 9.4, 9.5

Para la compresión usando el método QR se obtuvo la figura 10.

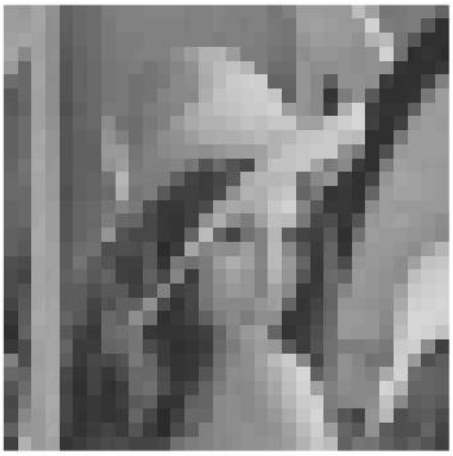

Figura 10 – Imagen comprimida usando el método QR con traslaciones

Con respecto a la variante de PCA sin usar punto flotante, en la figura 11 se ve la imagen usada durante el informe luego de su compresión, truncado y reconstrucción usando los 5 autovectores principales (aquellos con mayor autovalor respectivamente).

Figura 11 – Imagen comprimida con PCA sin punto flotante

5. Conclusiones

Una observación importante al usar una cantidad de autovectores variable es la forma en la que agregar autovectores hace variar la imagen resultante.
Luego de diferentes tipos de experimentos, tomando tanto autovectores correspondientes a los autovalores mayores en módulo como a los menores, se observó que al usar los correspondientes a los mayores autovalores se lograba una imagen más aproximada a la imagen original.
Esto se puede explicar de la siguiente manera:
Los 256 autovectores calculados son una base de R256, luego, usando todos ellos puedo reconstruir la matriz de covarianza y aplicarla a la imagen media para obtener nuevamente la imagen original.
Si en lugar de tomar los 256 autovectores calculados se tomasen los más representativos, es decir, los correspondientes a los autovalores más grandes, se podría también reconstruir, al menos aproximadamente, cualquier columna de la matriz de covarianza dado que estos autovectores son los 'más representativos' de dicha matriz.
Por lo tanto, tanto tomar más autovectores como seleccionar los de mayor autovalor correspondiente, generan una aproximación más precisa a la matriz de covarianza y, por ende, a la reconstrucción de la imagen original.

Hablando del método QR con traslaciones, supuestamente utilizar la descomposición QR con corrimiento permitiría comprimir información de los autovalores menos significativos en los autovalores con mayor información, logrando así tener menos pérdida al comprimir utilizando el autovector.
Con respecto a la compresión PCA sin punto flotante, se puede calcular el porcentaje de compresión como el tamaño luego de la compresión sobre el tamaño original de la imagen.

Tamaño original:

$$(512*512) \text{ Bytes} = 262{,}444 \text{ Bytes}$$

Tamaño luego de la compresión y truncado:

- Tamaño de la imagen media =1024 Bytes (1 Byte por cada bloque de la 'imagen media' representando el nivel de gris de dicho bloque de 0 a 255)
- Tamaño de los 5 autovectores enviados luego de la compresión = 5 * 256 (Un byte por cada valor de cada autovector)
- Tamaño de los máximos y mínimos enviados para la reconstrucción de los autovectores = 5 * 2 * 2 (2 Bytes por cada máximo y mínimo de cada autovector)

$$\text{Total} = 1024+1280+20 \text{ Bytes} = 2324 \text{ Bytes}.$$

Luego la compresión de la imagen fue de un 99,11%.

6. Bibliografía

http://people.maths.ox.ac.uk/wendland/teaching/mt201011/part3.pdf | Método QR con traslaciones