Parte 3:

- **Problema:** El principal problema que presenta el articulo es, como hacer para que, a partir de una base de datos con una cantidad determinada de imágenes, se pueda crear un algoritmo con la capacidad de relacionar una imagen con una cara totalmente diferente a las usadas para entrenar con una que si se encuentra en la base de datos.

- **Solución:** La solución que se plantea en el artículo es tomar el hecho que una imagen se puede representar como una matriz de $m\ x\ n$ pixeles y aplicar una serie de transformaciones y operaciones matriciales para poder comparar una a una las imágenes a identificar con las imágenes de la base de datos y poder dar con la que se parece más, o bien, decidir si no coincide con ninguna en la base de datos.

- **Pasos realizados:** A continuación, se presenta una serie de pasos, los cuales fueron usados para aplicar la solución planteada en el artículo usando el lenguaje de programación de Octave en su versión 5.2.0 para el sistema operativo Windows 10.

    1. **Creación del espacio S:** tomando en cuenta que las imágenes son matrices de tamaño $m\ x\ n = M$ y se tiene una base de datos de N imágenes. El espacio S es una matriz de tamaño $M\ x\ N$. Pero, para poder almacenar cada una de las imágenes en el espacio S primero deben pasar una pequeña transformación en la cual se convierte la imagen de tamaño $m\ x\ n$ en un vector de columna de tamaño $M\ x\ 1$. En la imagen 1 se puede observar cómo se realizó esto en Octave, se hace un doble ciclo for para iterar todas las imágenes en la base de datos, estas se convierten en formato double, luego se convierten en un vector columna usando un algoritmo propio (ver anexo 1) y se almacena cada una en el espacio S.

```
%-----------Creacion del espacio---------------------------------
disp('- Inicio creacion del espacio.')
for x = 1:40
  for y = 1:9
    B = imread(['Database/' int2str(x) '_' int2str(y) '.png']);
    B = im2double(B); %Carga y convierte a double la imagen
    f = img2column(B); %Comvierte la matriz de la imagen en vector
    S(:,(x-1)*9+y) = f;
    fprom = fprom + f;
  endfor
endfor
disp('- Espacio creado.')

%------------Calcula el vector promedio de imagenes------------
fprom = fprom/N;
disp('- Imagen promedio del espacio creada.')
```

Imagen 1. Creación del espacio S en Octave

2. **Cálculo de la imagen promedio:** Para el cálculo de la imagen promedio del espacio S simplemente se hace una media estadística de todos las vectores de imagen del espacio tal que $\frac{1}{N}\sum_{i=1}^{N} f_i$, donde f es un vector de imagen extraído del espacio S. Aprovechando que en que en el paso anterior ya se iteraban todas las imágenes, al momento de agregarlas al espacio, también se incrementaba el valor del vector fprom, el cual es un vector de imagen que almacena la imagen promedio del espacio; para luego de que se iteraban las imágenes de dividía el valor resultante entre el valor de N. Esto también se puede observar en la imagen 1

3. **Calculo de la matriz A y los SVD:** La matriz A es una matriz del mismo tamaño que el espacio S en la cual se almacena cada imagen del espacio S pero restándole el valor de fprom a cada una. Para ello en Octave (como se observa en la imagen 2) se itera el espacio S desde 1 hasta N asignando uno por uno los valores de A. Posteriormente para el calculo de los valores singulares de A se usa la función propia del lenguaje para facilitar el trabajo. Dicha función da como resultado 3 matrices nuevas U, E, V donde U es una matriz ortogonal de tamaño $M\ x\ M$, E es una matriz diagonal de tamaño $M\ x\ N$ y V es una matriz ortogonal de tamaño $N\ x\ N$. Además, se cumple que $A = U * E * V^T$, pero para efectos de este algoritmo solo nos interesa la matriz U, desde ahora llamada “subespacio de caras”.

```
%------------Crea la matriz A---------------------------------
disp('- Inicio creacion de la matriz A = f-fprom.')
for y = 1:N
  A(:,y) = S(:,y) - fprom;
endfor
disp('- matriz A = f-fprom creada.')

%------------Calcula SVD--------------------------------------
disp('- inicio calculo de valores singulares de A')
[U,E,V] = svd(A);
disp('- Valores singulares de A creados')
```

Imagen 2. Creación de A y calculo de SVD en Octave

4. **Creación del vector de coordenadas y E_1:** El vector de coordenadas es el encargado de ayudar a determinar, cuál de las caras del espacio es la que más se acerca más a la imagen que queremos identificar para su cálculo se usa una función propia (ver anexo 2), donde se recibe el subespacio de caras y la matriz A ya que el vector de coordenadas se calcula tal que $X = U^T * (f_i - fprom)$, pero como la ultima parte ya fue almacenada en la matriz a solo se itera la matriz A para obtener dichos valores. Una vez que se tiene el vector de coordenadas X se debe calcular el umbral máximo (ε_1) dicho vector indicara la distancia máxima seleccionable para saber si la imagen es una cara o no. Para

realizar este cálculo se programó otra función (ver anexo 3) en donde se aplica la formula $\varepsilon_i = \|X - X_i\|_2$ a cada uno de los valores del vector de coordenadas. En la imagen 3 se observa estos conceptos implementados en Octave.

```
%------------vector de coordenadas------------------------------
disp('- Inicio creacion del vector de coordenadas')
X = coordvector(U, A);
disp('- Vector de coordenadas creado')

%------------E0-------------------------------------------------
disp('- inicio calculo de umbrales')
E1 = threshhold(X);
disp('- E_1 creado')
```

Imagen 3. Creación de Vector de coordenadas y cálculo de umbral

5. **Determinación de la imagen:** Para poder comparar una imagen totalmente nueva con las imágenes de la base de datos primero se debe cargar la imagen, convertirla a double y luego a vector de columna como ya se hizo anteriormente con las imágenes de la base se datos. Una vez que se realiza hay que calcular tres elementos que serán necesarios para la determinación de la imagen, El vector de coordenadas para la nueva imagen con la formula anteriormente usada, luego la proyección del vector el cual se calcula tal que $f_p = U * X_{img}$ y por último el umbral entre la imagen que se comparará y las imágenes de la matriz A (ε_f) para saber si es una matriz, dicho umbral se calcula tal que $\varepsilon_f = \|(f_{img} - f_{prom}) - f_p\|_2$. Esto se puede observar en la imagen 4.

```
numimg = 1;
image = imread(['Comparar/' num2str(numimg) '_10.png']);
image = im2double(image);
f_image = img2column(image);

x_image = U' * (f_image - fprom);

fp = U * x_image;

Ef = norm(((f_image - fprom) - fp),2);
```

Imagen 4. Creación de los parámetros para comparar una nueva imagen.

6. **Determinación de la más parecida:** Para poder determinar la imagen más parecida se deben tomar consideraciones, por ejemplo: si $\varepsilon_f > \varepsilon_1$ entonces la imagen no corresponde a una cara, luego para saber cual es la mas parecida se calcula el umbral para la imagen usando cada valor del vector de coordenadas tal que $\varepsilon_{img} = \|X_{img} - X_i\|_2$, se determina cual es el umbral más pequeño y cuál es el

índice del vector de coordenadas al que pertenece, este índice será el que se usara para extraer la imagen del espacio S y esta será (en teoría) la imagen más parecida.

```
if Ef > El
  disp('No es una cara');
else
  for i = 1:360
    Ei_temp = norm(x_image - X(:,i), 2);
    if Ei_temp < Ei
      index = i;
      Ei = Ei_temp;
    endif
  endfor
```

Imagen 5. Determinación del umbral más pequeño

- **Resultados:**

A continuación, se muestra una a una la comparación de cada una de las 40 imágenes de caras. Es importante destacar que algunas caras no se emparejaron con la persona correcta, sin embargo, las emparejo con facciones similares, esto se podría deber a que como los cálculos usan los valores numéricos de cada pixel por separado y no como un conjunto, en esos casos particularmente, quizá por tener la misma expresión facial o bien por tener mismo color de piel, cabello o forma de la cara al algoritmo le cuesta mas identificar la cara.

Además, la base de datos con la que se estaba trabajando no es lo suficientemente robusta como para manejar más opciones, esto también pudo haber sido una posible causa de lo sucedido con esos casos específicos.

De las 4 imágenes analizadas el algoritmo pudo identificar correctamente un total de 37 imágenes, es decir tiene una efectividad de un 92.5% aproximadamente. En conclusión, es muy difícil que un algoritmo de este tipo sea 100 % eficiente, sin embargo, se logró un índice de efectividad muy aceptable.

Imagen a comparar: 1

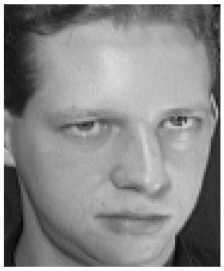


Imagen mas parecida, index = 5

Imagen a comparar: 2

Imagen mas parecida, index = 16

Imagen a comparar: 3

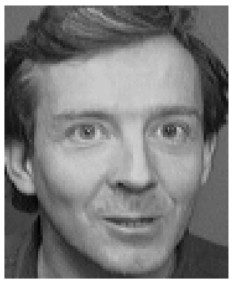


Imagen mas parecida, index = 27

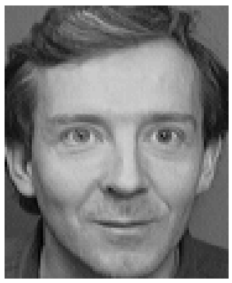


Imagen a comparar: 4

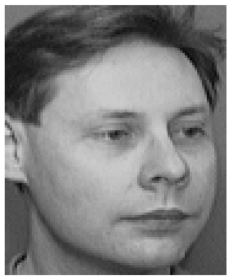


Imagen mas parecida, index = 32

Imagen a comparar: 5

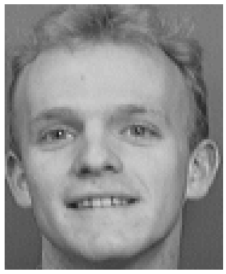


Imagen mas parecida, index = 356

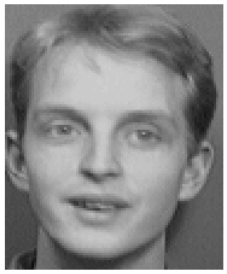


Imagen a comparar: 6

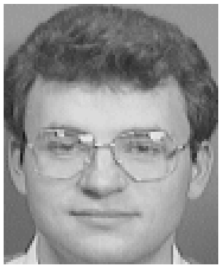


Imagen mas parecida, index = 49

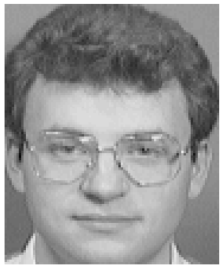


Imagen a comparar: 7

Imagen mas parecida, index = 56

Imagen a comparar: 8

Imagen mas parecida, index = 66

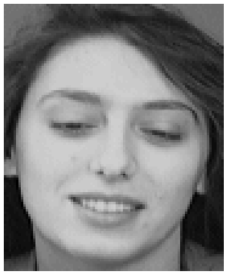


Imagen a comparar: 9

Imagen mas parecida, index = 77

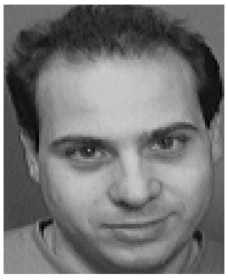


Imagen a comparar: 10

Imagen mas parecida, index = 337

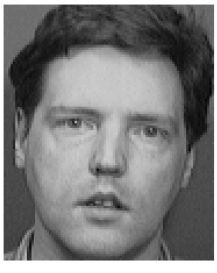


Imagen a comparar: 11

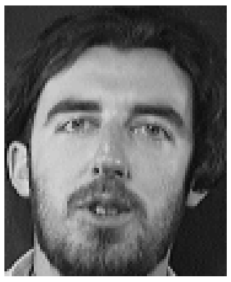


Imagen mas parecida, index = 91

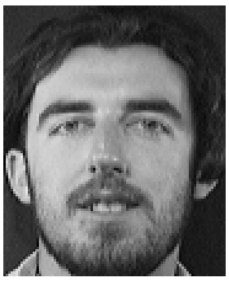


Imagen a comparar: 12

Imagen mas parecida, index = 108

Imagen a comparar: 13

Imagen mas parecida, index = 113

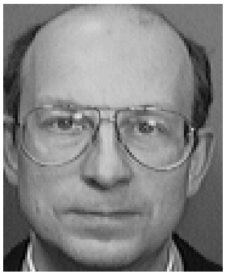


Imagen a comparar: 14

Imagen mas parecida, index = 126

Imagen a comparar: 15

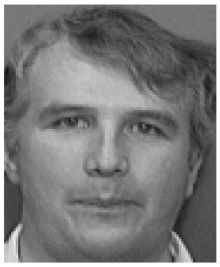


Imagen mas parecida, index = 128

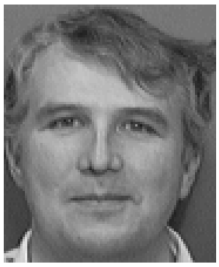


Imagen a comparar: 16

Imagen mas parecida, index = 138

Imagen a comparar: 17

Imagen mas parecida, index = 151

Imagen a comparar: 18

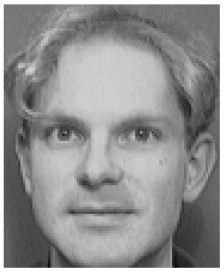


Imagen mas parecida, index = 161

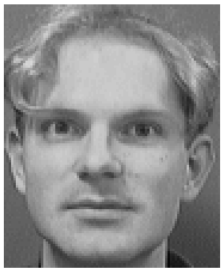


Imagen a comparar: 19

Imagen mas parecida, index = 163

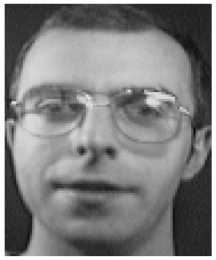


Imagen a comparar: 20

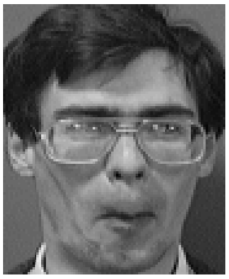


Imagen mas parecida, index = 173

Imagen a comparar: 21

Imagen mas parecida, index = 188

Imagen a comparar: 22

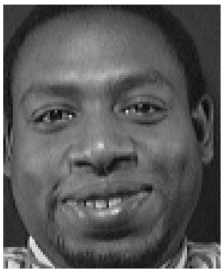


Imagen mas parecida, index = 191

Imagen a comparar: 23

Imagen mas parecida, index = 199

Imagen a comparar: 24

Imagen mas parecida, index = 216

Imagen a comparar: 25

Imagen mas parecida, index = 219

Imagen a comparar: 26

Imagen mas parecida, index = 233

Imagen a comparar: 27

Imagen mas parecida, index = 235

Imagen a comparar: 28

Imagen mas parecida, index = 246

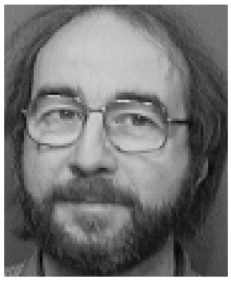


Imagen a comparar: 29

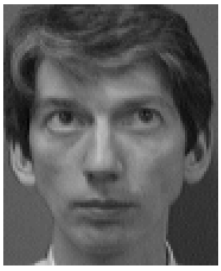


Imagen mas parecida, index = 255

Imagen a comparar: 30

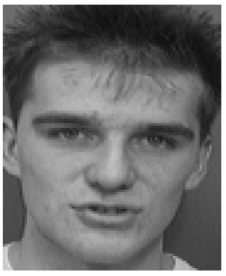


Imagen mas parecida, index = 262

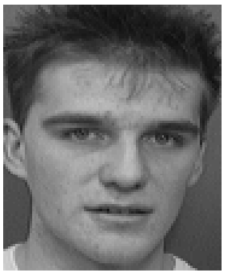


Imagen a comparar: 31

Imagen mas parecida, index = 275

Imagen a comparar: 32

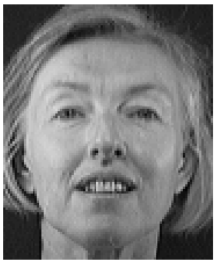


Imagen mas parecida, index = 285

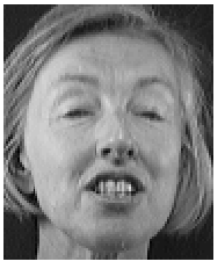


Imagen a comparar: 33

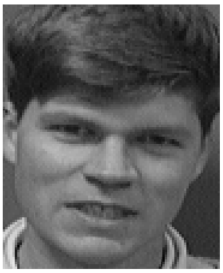


Imagen mas parecida, index = 290

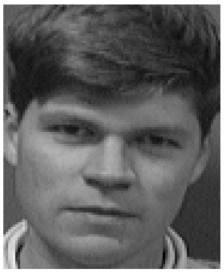


Imagen a comparar: 34

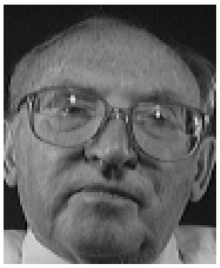


Imagen mas parecida, index = 306

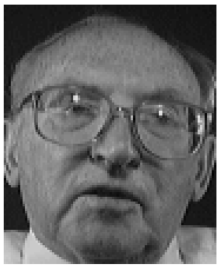


Imagen a comparar: 35

Imagen mas parecida, index = 311

Imagen a comparar: 36

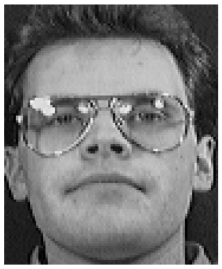


Imagen mas parecida, index = 321

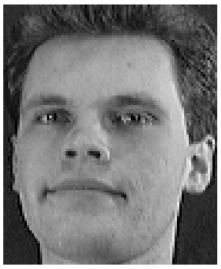


Imagen a comparar: 37

Imagen mas parecida, index = 333

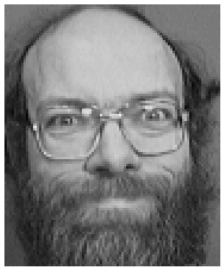


Imagen a comparar: 38

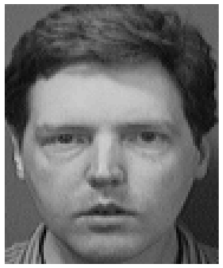


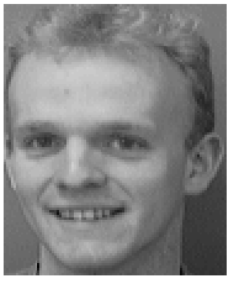
Imagen mas parecida, index = 338

Imagen a comparar: 39

Imagen mas parecida, index = 348

Imagen a comparar: 40

Imagen mas parecida, index = 37



- **Anexos:**

```
function Y = img2column(A)
  [m,n] = size(A);
  Y = zeros(m*n,1);
  for x = 1:m
    for y = 1:n
      Y(n*(x-1)+y) = A(x,y);
    endfor
  endfor
```

```
function Y = column2img(A,m,n)
  Y = zeros(m,n);
  for x = 1:m
    for y = 1:n
      Y(x,y) = A(n*(x-1)+y);
    endfor
  endfor
```

Anexo 1. Conversión de imagen a vector columna y viceversa.

```
function X = coordvector(U, A)
  [m, n] = size(A);
  X = zeros(m,n);
  for i = 1:n
    X(:,i) = (U' * A(:,i));
  endfor
endfunction
```

Anexo 2. Creación del vector de coordenadas.

```
function Y = threshhold(A)
  [m,n] = size(A);
  Y = Y_temp = 0;
  for i = 1:n
    Y_temp =  norm(A-A(:,i),2);
    if Y_temp > Y
      Y = Y_temp;
    endif
  endfor
endfunction
```

Anexo 3. Calculo del umbral.