

Detección de rostros usando EigenFaces

Flores-Silva P.

Abstract—Se desarrolló un detector de rostros empleando la técnica más sencilla que consiste en la obtención de EigenFaces para un conjunto de 1071 imágenes de rostros. Para la obtención de las EigenFaces fue necesario realizar la técnica de Análisis de componentes principales para reducir la dimensionalidad de los datos. Esta técnica proporciona una matriz M para la cual al aplicar la descomposición SVD se obtienen los eigenvectores asociados a las eigenFaces. Este método nos permite reconstruir cada una de las imágenes a partir de las eigenFaces. Se encontró que el error mínimo de reconstrucción es aproximadamente de 30 unidades, que se asocia a la perdida de datos debida a PCA. Como complemento se reconstruyeron seis imágenes correspondientes a mamíferos encontrándose un error mínimo de aproximadamente 850 unidades. Finalmente al considerar el error de reconstrucción convencional se obtuvo un rendimiento muy inferior a lo esperado por lo que se propone una nueva función de error que resulta en un mejor desempeño del detector. Si bien es cierto aun con esta nueva función el detector presenta una cantidad considerable de errores que se asocian al cambio de resoluciones entre las imágenes a detectar y el conjunto de datos con el que se trabajó para obtener las eigenFaces. Se propone que todo el proceso se trabajen con imágenes capturadas con la misma cámara fotográfica.
NOTA: La libreta de python desarrollada para este trabajo así como las imágenes resultantes puede encontrarse en la siguiente dirección web: <https://github.com/Pedri0/EigenFaces>.

Index Terms—Visión computacional, Imágenes, EigenFaces, Detector de Rostros, PCA, SVD, Linear Algebra.

1 INTRODUCCIÓN

UNA de las características esenciales del ser humano es el rostro. Debido a la existencia de una enorme variedad de rostros, podemos considerar que dicha característica es aquella que suele ser útil para diferenciarnos de otros seres humanos. Si bien es cierto que existen casos, la mayoría de veces raros, en los que dos o más personas comparten una similitud facial, la identificación de las personas, en la sociedad, tiene dos componentes principales que nos identifica como seres únicos: la primera de ellas es el rostro, mientras que la otra se trata de las huellas digitales. Es por ello que la gran mayoría de documentos oficiales requieren de estas dos características.

Sistemas de seguridad usan al menos una de dichas características. El ejemplo más sencillo puede encontrarse en los smartphones (a partir de la gama media) actuales que implementan ambas para poder acceder a la información del mismo.

De acuerdo a la página web [1] existen al menos dos tipos de sensores dactilares: el óptico y el ultrasónico cuyo funcionamiento, a grandes rasgos, consiste en la detección de una diferencia de potencial electromagnético e una superficie sensible a dichos cambios de potencial. La detección de dicho potencial permite al sensor crear una imagen virtual de la huella dactilar.

El reconocimiento facial consiste en la identificación o verificación de una persona a partir de una imagen o vídeo del rostro. Existen múltiples técnicas para realizar dicha tarea, pero la mayoría de ellas consiste en la comparación de características faciales de una imagen dada con características bien definidas dentro de una base de datos [2].

En el presente trabajo nos enfocaremos en el reconocimiento facial mediante imágenes de rostros, con una de las técnicas

más sencilla para este fin, el cuál emplea técnicas de álgebra lineal para obtener los vectores y valores característicos de una matriz. A esta técnica se le conoce como **EigenFaces**. El reconocimiento facial basado en imágenes contiene una gran cantidad de grados de libertad debido a las propiedades de la imagen, las texturas del rostro entre otros. Esta alta dimensionalidad requiere cantidades considerables de recursos computacionales, que en algunas veces no suelen estar disponibles. Recordemos, también, que una de las propiedades deseadas en cualquier sistema de reconocimiento, la tarea debe realizarse de forma rápida, de esta manera, el hecho de considerar todas las dimensiones de la imagen conduce tiempos relativamente largos de respuesta.

La manera más sencilla de reducir la alta dimensionalidad en un problema en general consiste en la aplicación del **Análisis de componentes principales**.

1.1 Análisis de componentes principales (PCA)

El análisis de componentes principales o PCA por sus siglas del inglés Principal Component Analysis es una de las técnicas más empleadas para aplicaciones como la reducción de la dimensionalidad, compresión de datos con perdida, extracción de características y visualización de datos [3]. A esta técnica se le conoce también como **la transformada de Karhunen-Loéve**.

Existen dos definiciones clásicas equivalentes de PCA: Una de ellas define al PCA como la proyección ortogonal de los datos en un espacio lineal de menores dimensiones, este espacio se conoce como **el subespacio principal**, de tal manera que la varianza de los datos proyectados es máxima. La segunda define al PCA como la proyección lineal que minimiza la media del costo de la proyección, definida como la distancia cuadrática media entre los puntos y sus proyecciones. La figura 1 muestra PCA como la proyección

• P. Flores-Silva estudiante del PCIC en la UNAM.
E-mail: flosipan@ciencias.unam.mx

ortogonal.

Dado que ambas definiciones clásicas son equivalentes, aquí solo se presentará la definición como proyección ortogonal. Cualquier interesado en conocer el segundo método puede consultar [3]

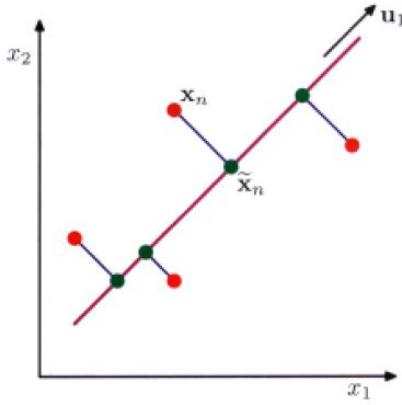


Fig. 1. PCA como proyección ortogonal. Los puntos verdes son los datos proyectados en un espacio lineal de los puntos rojos tales que la varianza es máxima. Tomada de [3].

1.2 PCA como proyección ortogonal

Sea $\{\vec{x}_n\}$ un conjunto de datos u observaciones, con $n = 1, 2, \dots, N$ y $\vec{x}_n \in \mathbb{R}^D$. Como ya se mencionó, se requiere proyectar el conjunto de datos sobre un espacio cuya dimensión M es menor a D, de tal manera que se maximice la varianza de los datos proyectados.

Considérese la proyección a un espacio de dimensión uno ($M = 1$). Podemos definir la dirección de este espacio uno-dimensional mediante el vector unitario $\vec{u}_1 \in \mathbb{R}^D$. La proyección de cada dato vendrá dada por el producto escalar $\vec{u}_1^T \cdot \vec{x}_n$, mientras que la proyección promedio se define como $\vec{u}_1^T \langle \vec{x}_n \rangle$, donde $\langle \vec{x}_n \rangle$ es el vector promedio de los datos dado por:

$$\langle \vec{x}_n \rangle = \frac{1}{N} \sum_{n=1}^N \vec{x}_n \quad (1)$$

De esta forma, la varianza de los datos proyectados está dada por:

$$\frac{1}{N} \sum_{n=1}^N (\vec{u}_1^T \vec{x}_n - \vec{u}_1^T \langle \vec{x} \rangle)^2 = \vec{u}_1^T S \vec{u}_1 \quad (2)$$

donde S es la matriz de covarianza de los datos, definida por:

$$S = \frac{1}{N} \sum_{n=1}^N (\vec{x}_n - \langle \vec{x} \rangle) (\vec{x}_n - \langle \vec{x} \rangle)^T \quad (3)$$

Para maximizar la ecuación (2) se introduce un multiplicador de Lagrange que se denota como λ_1 y se hace una maximización sin restricciones de la función

$$\vec{u}_1^T S \vec{u}_1 + \lambda_1 (1 - \vec{u}_1^T \vec{u}_1) \quad (4)$$

Al derivar e igualar a cero se sigue que:

$$S \vec{u}_1 = \lambda_1 \vec{u}_1 \quad (5)$$

Es decir, \vec{u}_1 es un eigenvector de S. Ahora al multiplicar por la izquierda por \vec{u}_1^T :

$$\vec{u}_1^T S \vec{u}_1 = \lambda_1 \quad (6)$$

De esta manera, S será máxima cuando restringimos que \vec{u}_1 sea igual al máximo eigenvalor λ_1 . Este eigenvector se conoce como el primer componente principal.

Si se considera el caso general de proyecciones M-dimensionales, la proyección lineal óptima para la cual la varianza de los datos proyectados es máxima esta definida por los M eigenvectores $\vec{u}_1, \dots, \vec{u}_M$ de la matriz de covarianza S que corresponden a los M eigenvalores $\lambda_1, \dots, \lambda_M$ de valor más alto.

1.3 EigenFaces

Considérese un conjunto de imágenes de rostros de distintas personas. Sean el ancho y el alto de las imágenes representados por w y h respectivamente.

Dado que la técnica de PCA consiste, a grandes rasgos, en encontrar los eigenvectores de la matriz de covarianza de un conjunto de vectores dado. Es de gran utilidad expresar o redimensionar cada una de las imágenes, que en esencia son una matriz, como un vector de dimensiones $w \times h, 1$. Es posible calcular el vector promedio de imágenes redimensionadas usando la ecuación (1). El siguiente paso consiste en calcular la matriz S usando la ecuación (3), sin embargo el calculo de S empleando dicha ecuación es computacionalmente ineficiente ya que de (3) las dimensiones de dicha matriz son de $(w \times h, w \times h)$. Si consideramos que cada imagen tiene dimensión $(64, 64)$, entonces para calcular la matriz S habría que obtenerse 16,777,216 valores. Para evitar el calculo de esta gran cantidad de valores se usa la factorización SVD o descomposición en valores singulares. Sea M una matriz real o compleja de dimensión $m \times n$, entonces su SVD resulta en $M = U \Sigma V^*$, donde U es una matriz unitaria de dimensión $m \times m$, Σ es una matriz rectangular diagonal con valores reales positivos de dimensión $m \times n$ y V es una matriz compleja o unitaria de dimensión $n \times n$. Si M es una matriz real, entonces U y $V^* = V^T$ son matrices reales ortogonales.

Para el caso de las imágenes, consideramos a M como la matriz cuyas dimensiones son $w \times h, N$ donde N es el número de imágenes, para ello, se hace $M = (\vec{x} - \langle \vec{x} \rangle)$.

Una de las propiedades mas interesantes de la descomposición SVD es que las columnas de la matriz U son los vectores propios de la matriz MM^T y los elementos en la diagonal de la matriz Σ corresponden a las raíces cuadradas de los valores propios de los vectores propios de U . Un resultado bien conocido de álgebra lineal es que los valores y vectores propios de una matriz no cambian al multiplicar por una constante, únicamente dichos vectores y valores deben ser reescalados por la misma constante.

De esta manera con SVD es posible obtener hasta N vectores propios ortogonales entre sí y valores propios. Con una menor cantidad de operaciones. Y con ello se aplica la técnica de PCA.

Sean $\{\vec{u}_i\}_{i=1}^N$ el conjunto de N eigenvectores. Entonces, un rostro \vec{x} redimensionado (vector de dimensiones $(w \times h, 1)$) puede ser expresado en el espacio base definido por los N

eigenvectores a través de sus eigenvalores de la siguiente manera:

$$\vec{x} = [\vec{u}_1^T (x - \mu), \dots, \vec{u}_N^T (x - \mu)]^T = [w_1, \dots, w_N]^T \quad (7)$$

Donde esta representación puede ser truncada hasta $k = 1, \dots, N$ eigenvectores. y μ corresponde al vector promedio $\langle \vec{x} \rangle$ calculado con la ecuación (1).

Ahora bien, la reconstrucción $\hat{\vec{x}}$ de una imagen \vec{x} a partir de los primeros k eigenvectores y eigenvalores se obtiene con:

$$\hat{\vec{x}} = \mu + \vec{u}^T \vec{w} \quad (8)$$

donde $\vec{u}^T = [u_1, \dots, u_k]$ y $\vec{w}^T = [w_1, \dots, w_k]$. Una representación visual de la reconstrucción se muestra en la figura 2.



Fig. 2. Representación visual de la reconstrucción de un rostro a partir de los eigenvectores y eigenvalores. El primer sumando corresponde a μ , mientras que el segundo sumando corresponde al producto interno de los primeros siete eigenvectores. Tomada de [4].

El error de reconstrucción se define como la norma l_2 de la diferencia entre \vec{x} y $\hat{\vec{x}}$:

$$E = \|(\vec{x} - \hat{\vec{x}})\|_2 \quad (9)$$

2 PROCEDIMIENTO EXPERIMENTAL Y RESULTADOS

El conjunto de imágenes de rostros empleado puede obtenerse en la siguiente pagina web: http://conradsanderson.id.au/lfwcrop/lfwcrop_grey.zip. De acuerdo a la carpeta raíz, todo el conjunto de imágenes es un extracto de la base de datos *Labeled Faces in the Wild*, donde cada una de dichas imágenes mantienen únicamente la región central de cada imagen original y en la gran mayoría de las imágenes el fondo es omitido. De esta manera no es necesario realizar ningún preprocessamiento a las mismas. Las dimensiones de cada imagen son de (64, 64) y están almacenadas en un formato .pgm, el dataset consiste de 1071 imágenes.

El dataset se guarda en una carpeta llamada famousFace y cada imagen es leída y almacenada en una lista llamada imágenes, mediante la siguiente instrucción

```
1 import cv2
2 import glob
3 imágenes = [cv2.imread(file,-1) for file in glob.
4     glob("famousFace/*.pgm")]
5
```

Listing 1. script para leer las 1071 imágenes del rostro.

Posteriormente, creamos una función auxiliar que nos permite visualizar k imágenes en forma de enrejado. Además por simplicidad se transforma la lista imágenes a un array de numpy. El código se muestra a continuación

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 imágenes = np.array(imágenes)
4 #funcion auxiliar para mostrar un sample del data
5 set
6 def muestraImagenes(imagen,alto,ancho,filas,columnas):
7
```

```
6 plt.figure(figsize=(3.0 * columnas, 3.0 * filas))
7
8 for i in range(columnas * filas):
9     plt.subplot(filas, columnas, i + 1)
10    plt.imshow(imagen[i].reshape((alto, ancho)),
11               cmap='gray')
12    plt.title('Famos@ %d' % i)
13    plt.xticks(())
14    plt.yticks(())
```

Listing 2. script para visualizar k imágenes.

Al ejecutar la función para 16 imágenes seleccionadas de forma aleatoria del conjunto de datos obtenemos la figura 3.



Fig. 3. Conjunto de 16 imágenes seleccionadas al azar del conjunto total de datos

Para realizar el PCA de las imágenes mediante la descomposición SVD tal y como se describió en la introducción se desarrollo una función que recibe como argumentos el conjunto de datos o imágenes y el numero de eigenvectores deseados. Esta función redimensiona las imágenes, calcula el vector μ y realiza la descomposición SVD de la matriz M . Regresa el vector de proyecciones, los eigenvectores, el vector μ y la matriz M . El código es como sigue.

```
1 def PCA(Datos,numcomponentes):
2     #Convertimos cada imagen a un vector de N^2 x 1
3     Gamma = imágenes.reshape(imágenes.shape[0],
4                               imágenes.shape[1]*imágenes.shape[2])
5     #calculamos la media (step 3)
6     Psi = np.mean(Gamma, axis=0)
7     #calculamos phi
8     Phi = Gamma - Psi
9     #calculamos la SVD de Gamma (steps 4 to 7)
10    U, Sigma, V = np.linalg.svd(Phi)
11    componentes = V[:numcomponentes]
12    proyeccion = U[:, :numcomponentes] * Sigma[: numcomponentes]
13
14    return proyeccion, componentes, Psi, Phi
```

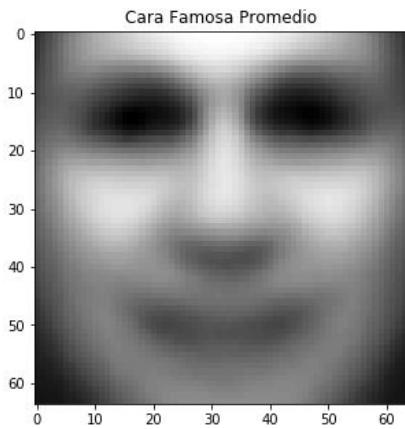
Listing 3. Función que obtiene el PCA y regresa vectores necesarios para realizar la reconstrucción.

Al aplicar la función PCA para 50 eigenvectores sobre todo el conjunto de datos, redimensionarlos y pasarlos por la función muestraImagenes obtenemos las primeras 50 eigenFaces (ver figura 4)



Fig. 4. Primeras 50 eigenFaces

Con un proceso similar es posible obtener la imagen del vector μ . Esta imagen se muestra en la figura 5

Fig. 5. Imagen del vector promedio μ

Para el proceso de reconstrucción se desarrolló una función llamada reconstruye cuyos parámetros son la matriz M, los eigenvectores, el vector promedio, las dimensiones de la imagen y un índice que es auxiliar para iterar sobre todas las imágenes. Esta función aplica sobre dichos vectores o matrices las ecuaciones (7) y (8) y regresa la imagen reconstruida. La función se muestra a continuación.

```

1 def reconstruye(phi, componentes, media, alto, ancho,
2 , indice):
3     numeroFotos, tamanoFotocuadrado = phi.shape
4     pesos = np.dot(phi, componentes.T)
5     vectorCentrado=np.dot(pesos[indice, :], componentes)
6     imagenReshapeada=(media+vectorCentrado).reshape(
7         alto, ancho)
8     return imagenReshapeada

```

Listing 4. Función que reconstruye los rostros y regresa la imagen del rostro.

Para reconstruir las imágenes con k eigenvectores y k eigenvalores es necesario correr la función PCA con dicho valor de k y posteriormente ejecutar la función reconstruye con los parámetros provistos de la función PCA. Para reconstruir todas las imágenes del conjunto y mostrarlas se usa el siguiente:

```

1 imagenesReconstruidas=[reconstruye(phi, componentes,
2     psi, imagenes.shape[1], imagenes.shape[2],
3     contador) for contador in range(len(imagenes))]
4 imagenesReconstruidas = np.array(
5     imagenesReconstruidas)

```

```

3 muestraImagenes(imagenesReconstruidas[z], imagenes.
4 shape[1], imagenes.shape[2], 4, 4)

```

Listing 5. reconstrucción de todas las imágenes con los parámetros previos calculados por PCA para k eigenvectores.

Al variar el valor de k para 10, 50, 150, 450, 1071 obtenemos las figuras 6, 7, 8, 9 y 10 respectivamente. Note que las 16 imágenes que se muestran en cada figura corresponden a las reconstrucciones de las imágenes mostradas en 3.

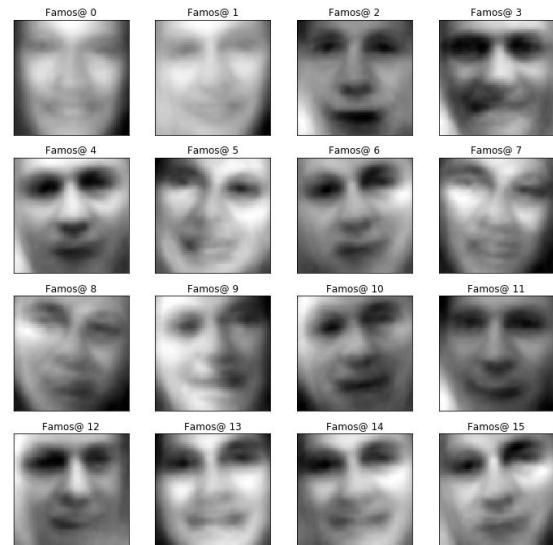


Fig. 6. Reconstrucción con las primeras 10 EigenFaces

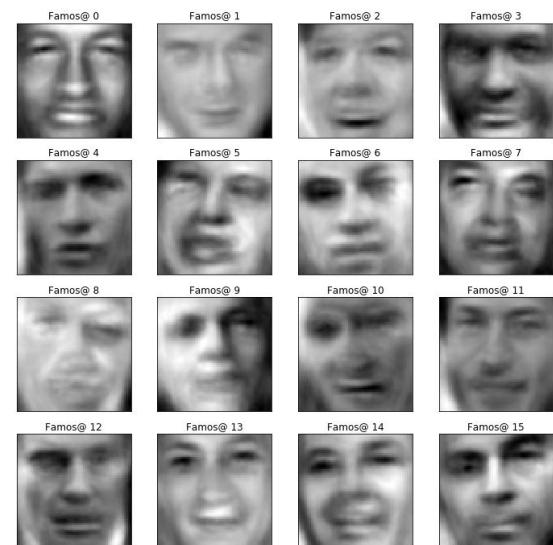


Fig. 7. Reconstrucción con las primeras 50 EigenFaces

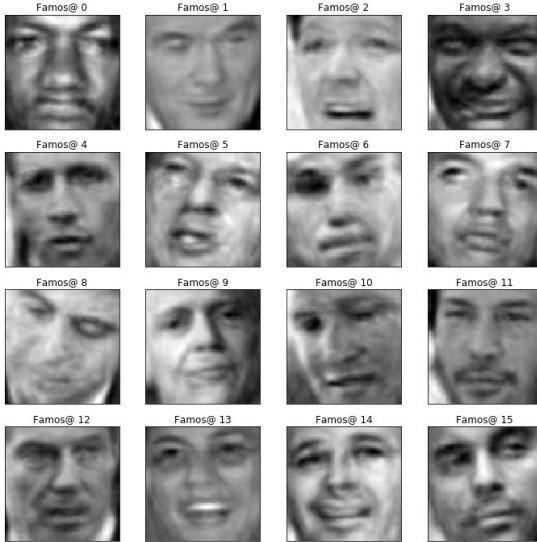


Fig. 8. Reconstrucción con las primeras 150 EigenFaces



Fig. 10. Reconstrucción con las primeras 1071 EigenFaces



Fig. 9. Reconstrucción con las primeras 450 EigenFaces



Fig. 11. Imágenes originales de 4 gatos y 2 perros

Una vez obtenidos los eigenvectores y eigenvalores. Es posible reconstruir caras de mamíferos, claramente dado que la cara humana y de cualquier otro mamífero pueden ser muy diferentes, la reconstrucción, aun al considerar los 1071 eigenvectores, tendrá mucho ruido con respecto a las imágenes originales. Para observar esta diferencia se consideraron 6 imágenes de 4 gatos y dos perros y se les aplicó el mismo proceso de reconstrucción considerando diferentes valores de k . Las imágenes originales se muestran en la figura 11, mientras que las reconstrucciones para 1071, 450, 150 y 80 eigenvalores se muestran en las figuras 12, 13, 14 y 15 respectivamente.

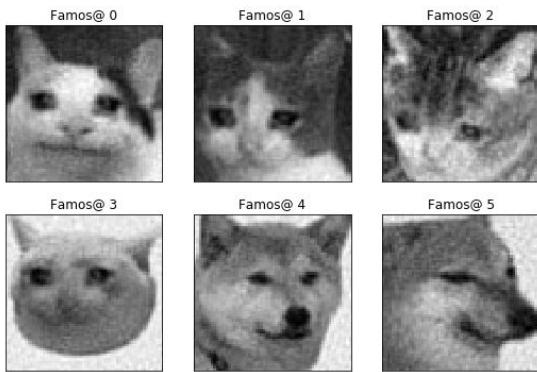


Fig. 12. Reconstrucción de las imágenes de gatos y perros con las primeras 1071 EigenFaces

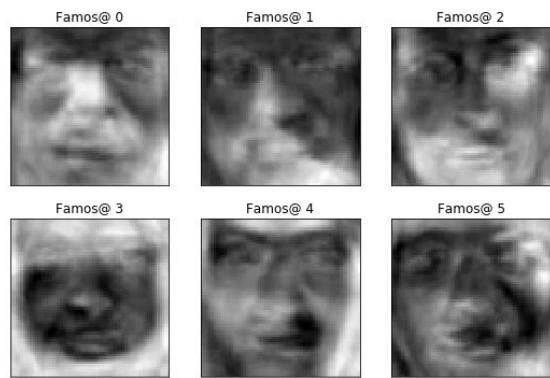


Fig. 15. Reconstrucción de las imágenes de gatos y perros con las primeras 80 EigenFaces

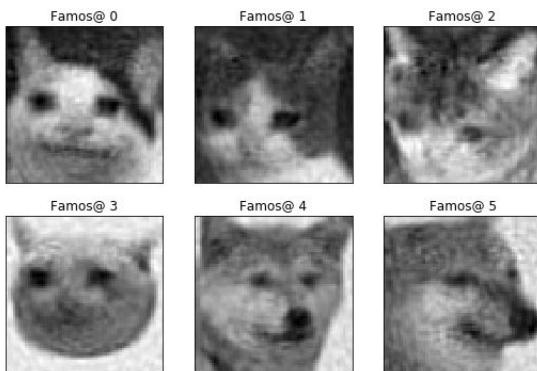


Fig. 13. Reconstrucción de las imágenes de gatos y perros con las primeras 450 EigenFaces

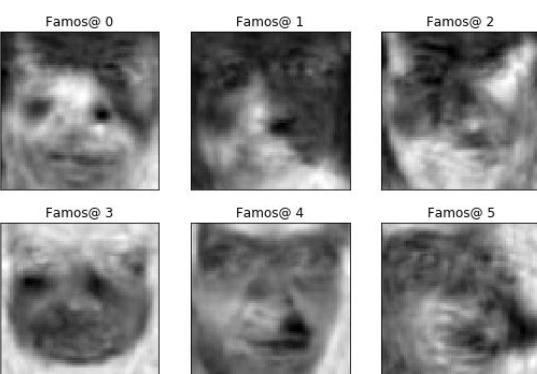


Fig. 14. Reconstrucción de las imágenes de gatos y perros con las primeras 150 EigenFaces

Finalmente para obtener el error de reconstrucción para una sola imagen dada, se escribió el siguiente código. Note que los parámetros de dicha función son los parámetros arrojados por PCA, la imagen del vector μ y la imagen a reconstruir. Dentro de la misma función se realiza todo el proceso y se calcula el error tal y como se define en la ecuación (9).

```

1 def lossy(imagen,phi,componentes,psi,imagenPromedio)
2     :
3     lista = []
4     loss = []
5     salto = []
6     imagenPromedio = imagenPromedio.reshape(
7         imagenPromedio.shape[0]*imagenPromedio.shape[1])
8     imagen = imagen.reshape(imagen.shape[0]*imagen.
9         shape[1])
10    centro = imagen-psi
11    for j in range(0,componentes.shape[0]):
12        lista.append(componentes[j].T@centro)
13    pesos = np.array(lista)
14
15    for i in range(0,1072,3):
16        reconstrucion = imagenPromedio +
17            componentes[:i].T@pesos[:i]
18        err = imagen - reconstrucion
19        err = err.T @ err
20        loss.append(err)
21        salto.append(i+1)
22
23    lossys = np.array(loss)
24    lossys = np.sqrt(lossys)
25    lossys = np.column_stack((salto, lossys))

```

Listing 6. Funcion que calcula el error de reconstrucción para la variacion de k y una sola imagen.

Las gráficas de error para una imagen aleatoria del conjunto de rostros humanos se muestra en la figura 16, mientras que el error de reconstrucción para una imagen aleatoria del conjunto de perros y gatos se muestra en la figura 17.

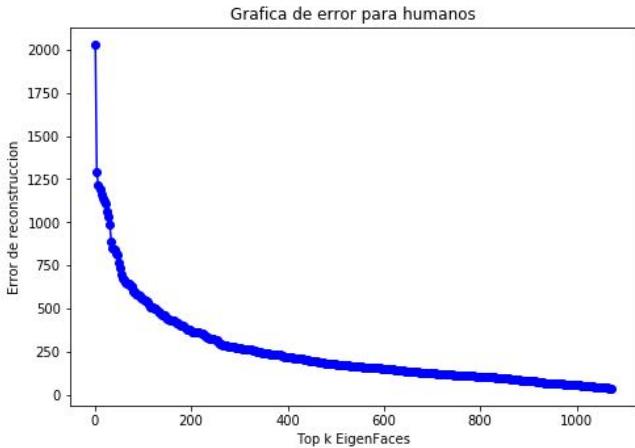


Fig. 16. Error de reconstrucción para diferentes valores de k y una imagen aleatoria del conjunto de rostros

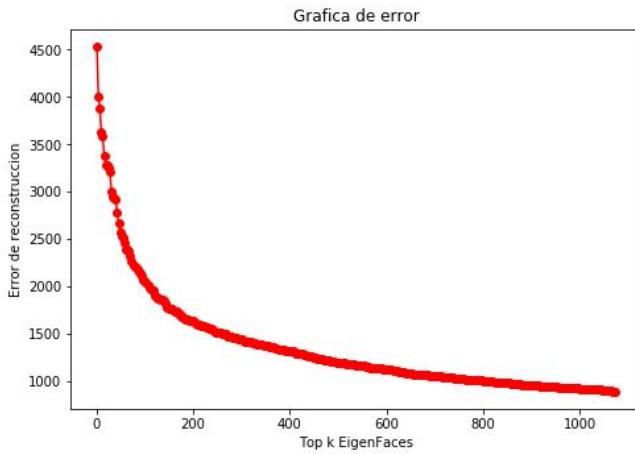


Fig. 17. Error de reconstrucción para diferentes valores de k y una imagen aleatoria del conjunto de perros y gatos

2.1 Reconocimiento de rostros

Dada una imagen para la cual aparece un conjunto de personas cuyo rostro es se encuentra frente a la cámara. Se va quiera reconocer dichos rostros usando los eigenvectores y eigenvalores previamente calculados. Note que la imagen puede ser de cualquier dimensión bajo la restricción de que se va a recorrer una ventana sobre la misma imagen de las mismas dimensiones de nuestro dataset, esto es, la ventana va a ser de dimensiones (64, 64). Para cada una de estas ventanas se va a calcular la matriz M dada por $M = V - \mu$, donde V representa la ventana de la imagen de interés. Posteriormente se va a calcular el vector \hat{x} usando la ecuación (8) sabiendo que $\vec{w} = [\vec{u}_1^T M, \dots, \vec{u}_k^T M]$. Para la detección de rostros se va a usar $k = 1071$, es decir todos los eigenvectores. Ya que la gráfica de error nos dice que el error es mínimo cuando se emplean todos los eigenvectores. Una vez conocido \hat{x} se procede a calcular el error de reconstrucción de la ecuación (9) sustituyendo \hat{x} por M , es decir, se va a calcular $E = \|M - \hat{x}\|_2$. Si $E < T$, donde T es un valor umbral definido, diremos que en la ventana se esta detectando un rostro y se procede a pintar la ventana de blanco indicando que aquella área dentro del

mismo se trata de un rostro según el criterio de error.

Dado que el conjunto de datos considera únicamente rostros por separado y todo el conjunto de datos fue empleado para obtener los 1071 eigenvectores es inverosímil considerar una imagen compuesta de dichos rostros para detectar rostros. Por lo que se optó por usar dos imágenes de bandas de metal. Cabe aclarar que la resolución del dataset y las dos nuevas imágenes a emplear puede variar significativamente, además, dado que las ventanas se recorren sobre toda la imagen sin translaparse no se asegura que se este capturando todo el rostro si no parte de él (si es el caso), es decir, es muy probable que las ventanas no se traten del caso ideal del rostro completo tal y como lo es para el dataset, y pueden suceder al menos tres casos: a) que la ventana no contenga ninguna clase de rostro, b) que la ventana capture partes del rostro y otras correspondientes al fondo o ropa y c) que la ventana capture la totalidad del rostro. El caso c) es el mas improbable ya que las dos imágenes empleadas se tomaron al vuelo como se encontraron en internet y debido a las posiciones de los sujetos no hay garantía de que el caso c) suceda. Debido a estas variaciones no se conoce a priori el valor T por lo que se va a variar dicho valor y se mostrara cada imagen con los resultados obtenidos para dicho valor de T .

El ventaneo requiere de la lectura de las dos imágenes a emplear dos ciclos for y un condicional del valor umbral T . El código se muestra a continuación.

```

1 #ahora probemos con otra imagen de black
2 black = cv2.imread('gaal.jpg',0)
3 #imprimimos sus dimensiones
4 print(black.shape)
5 #creamos los ciclos for para iterar sobre la imagen
6 # y pintar la imagen si se detecta una cara
7 for filas in range(32,black.shape[0]+1,64):
8     for columnas in range(32,black.shape[1]+1,64):
9         ventana = black[filas-32:filas+32,columnas
10            -32:columnas+32]
11         error = lossy2(ventana,phi,componentes,psi,
12                         caraPromedio)
13         if (error <= 100):
14             black[filas-31,columnas-31:columnas
15 +31]=255
16             black[filas+31,columnas-31:columnas
17 +31]=255
18             black[filas-31:filas+31,columnas-31]=255
19             black[filas-31:filas+31,columnas+31]=255
20 plt.figure(figsize = (16,16))
21 plt.imshow(black,cmap='gray')
22 plt.show()
23 #guardamos la imagen
24 cv2.imwrite('black100Usual.png',black)
```

Listing 7. script para la detección de rostros.

El código presenta la función lossy2 definida como el error E descrito arriba, el código es el siguiente.

```

1 #definimos nuesta funcion de error que simplemente
2 # es la norma 12
3 # del vector reconstruido con el vector imagen (de
4 # tamano N X 1)
5 # esta distancia se calcula para los 1071
6 # eigenvectores
7 def lossy2(imagen,phi,componentes,psi,imagenPromedio
8 ):
9     lista = []
10    imagenPromedio = imagenPromedio.reshape(
11        imagenPromedio.shape[0]*imagenPromedio.shape[1])
12    imagen = imagen.reshape(imagen.shape[0]*imagen.
13        shape[1])
```

```

8 centro = imagen-psi
9 for j in range(0,componentes.shape[0]):
10     lista.append(componentes[j].T@centro)
11 pesos = np.array(lista)
12 reconstruccion = componentes.T@pesos
13 err = centro - reconstruccion
14 err = err.T @ err
15 err = np.sqrt(err)
16 return err

```

Listing 8. Funcion de error convencional.

Las dos imágenes a emplear se muestran en las figuras 18 y 19.



Fig. 18. Primera imagen a detectar rostros



Fig. 19. Segunda imagen a detectar rostros. La señal no representa nada y no va dirigida a alguien en particular

Al aplicar la función de ventaneo y definir el error $E = \|\mathbf{M} - \hat{\mathbf{x}}\|_2$ para la detección de rostros con los valores de $E \leq 100$, $E \leq 400$ y $E \leq 8000$ se obtienen las figuras 20, 21 y 22 respectivamente.

Fig. 20. Detección de rostros de la figura 18 para un umbral de $E = 100$ y la función de Error definida como $E = \|\mathbf{M} - \hat{\mathbf{x}}\|_2$ Fig. 21. Detección de rostros de la figura 18 para un umbral de $E = 400$ y la función de Error definida como $E = \|\mathbf{M} - \hat{\mathbf{x}}\|_2$



Fig. 22. Detección de rostros de la figura 18 para un umbral de $E = 8000$ y la función de Error definida como $E = ||M - \hat{x}||_2$



Fig. 23. Detección de rostros de la figura 18 para un umbral de $\hat{E} = 8000$ y la función de Error definida como $\hat{E} = ||\mu - \hat{x}||_2$

Notemos que la figura 20 todos los casos detectados no corresponden a ninguno de los cuatro rostros. La detección de rostros es mejor para cuando $E = 400$ (figura 21) pero únicamente detecta a una de las cuatro personas y resulta en demasiados errores. Finalmente la figura 22 detecta toda la imagen como rostros algo que claramente tiene muchos errores.

Debido a que se obtenían resultados malos para la segunda imagen usando esta función de error los resultados no se presentan. La figura 20 indica que probablemente que la función de error tal y como se define no esta representando en su totalidad un rostro. Por esta razón se procede a proponer una nueva función de error mucho mas sencilla y con mejor rendimiento.

Una inspección rápida del resultado que se obtiene al restar el vector M con la reconstrucción da como resultado algo muy parecido a un rostro, es por esta razón por la que se cometen demasiados errores aun cuando la ventana no contiene ninguno de ellos. Al definir la función de error como $\hat{E} = ||\mu - \hat{x}||_2$ nos conduce a un rostro mucho menos definido para el caso en el que la región de interés o ventana no contiene ninguna parte del rostro humano. De esta forma procedemos a realizar el ventaneo de la figura 18 considerando la nueva función de error \hat{E} . Para valores de umbral de $\hat{E} = 8000$, $\hat{E} = 9000$ y $\hat{E} = 11000$ se obtienen las figuras 23, 24 y 25 respectivamente.



Fig. 24. Detección de rostros de la figura 18 para un umbral de $\hat{E} = 9000$ y la función de Error definida como $\hat{E} = ||\mu - \hat{x}||_2$

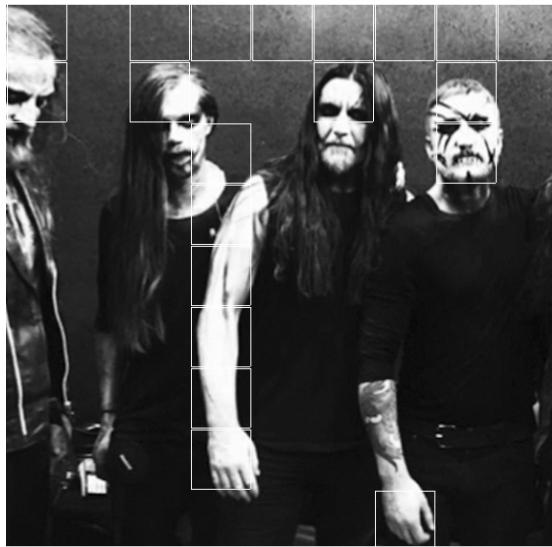


Fig. 25. Detección de rostros de la figura 18 para un umbral de $\hat{E} = 11000$ y la función de Error definida como $\hat{E} = ||\mu - \hat{x}||_2$



Fig. 27. Detección de rostros de la figura 19 para un umbral de $\hat{E} = 9500$ y la función de Error definida como $\hat{E} = ||\mu - \hat{x}||_2$

Dado que se obtienen resultados mejores, se procede a analizar la segunda imagen con los siguientes valores de umbral: $\hat{E} = 1500$, $\hat{E} = 9500$, $\hat{E} = 10000$ y $\hat{E} = 11500$. Las imágenes obtenidas se muestran en la figuras 26, 27, 28 y 29 respectivamente.



Fig. 26. Detección de rostros de la figura 19 para un umbral de $\hat{E} = 1500$ y la función de Error definida como $\hat{E} = ||\mu - \hat{x}||_2$



Fig. 28. Detección de rostros de la figura 19 para un umbral de $\hat{E} = 10000$ y la función de Error definida como $\hat{E} = ||\mu - \hat{x}||_2$



Fig. 29. Detección de rostros de la figura 19 para un umbral de $\hat{E} = 11500$ y la función de Error definida como $\hat{E} = \|\mu - \hat{x}\|_2$

3 DISCUSIÓN

3.1 EigenFaces

A pesar de que en la muestra de imágenes se observan rostros con alguna orientación no central, el hecho de que la imagen promedio o vector promedio si tenga dicha orientación nos da pauta a considerar que efectivamente la gran mayoría de rostros preservan dicha orientación y no es necesario realizar algún tipo de preprocesamiento.

Por otro lado, la visualización de las eigenFaces nos da una pista sobre la información dominante. Los primeros eigenvectores contienen información de la forma del rostro a diferentes niveles de gris. Conforme se va avanzando, la información dominante son una combinación de los niveles de gris. Puede observarse que a partir de la eigenFace 100 ya las formas llegan a distorsionarse hasta el punto de que dichas formas se asemejan a una onda en dos dimensiones. Este cambio en la dominación entre las formas y los niveles de gris es claro, las componentes principales para detectar un rostro son las formas seguido de la iluminación. Notemos que al usar 150 eigenvectores para reconstruir las 16 caras de muestra empleadas (figura 8) los rostros se aprecian muy bien definidos y puede distinguirse entre cada famoso perfectamente. El hecho de que las imágenes se muestren borrosas se debe en su mayor parte a los niveles de gris contenidos en las primeras 150 eigenFaces. Obsérvese también que para el caso de las 10 primeras eigenFaces (fig 6) los rostros aparecen tener muy pocos niveles de gris lo cual provoca que sean casi indistinguibles entre sí. Al inspeccionar visualmente la reconstrucción producida en 10 la similitud entre las imágenes reales y reconstruidas es muy alta. Sin embargo, la gráfica de error encuentra que existe al menos un error asociado mayor a cero (figura 16). Este error asociado se debe a la descomposición PCA ya que recordemos el efecto de dicho proceso contendrá perdida de la información.

La reconstrucción de los perros y gatos usando todos los eigenvectores resulta en imágenes ruidosas con un error mayor a 800 unidades (17), este error es fácil de explicar

al considerar que el conjunto de eigenFaces reconoce formas e intensidades inherentes al ser humano y dado que los animales tienen formas muy distintas así como sus intensidades en sus caras varían bastante con respecto a la cara humana, este efecto de ruido se produce al tratar de reconstruir las formas de dichos animales. Este hecho es muy claro al observar los resultados obtenidos en las figuras 14 y 15 donde aprecia claramente una combinación de formas pertenecientes al rostro humano con iluminaciones no homogéneas en los mismos. Por lo que este proceso de reconstrucción parte de una cara humana y se distorsiona de tal manera que se obtienen formas de los animales. Esta distorsión consiste en la combinación lineal de los mismos eigenvectores.

3.2 Reconocimiento de rostros

El hecho de considerar la función de error convencional nos llevó a una gran cantidad de errores que si este sistema se deseara implementar en la vida real su desempeño sería bastante malo. Sin embargo, cabe aclarar que este error puede deberse a la diferencia de resolución entre las imágenes del dataset y las dos imágenes y que cada ventana recorrida sobre la imagen no garantiza que capture el rostro totalmente.

Al cambiar la función de error por la propuesta, el desempeño del detector incrementa pero aun así encontramos una variedad de errores. Una inspección visual de dichos errores nos lleva a pensar que la función de error propuesta es muy sensible a las intensidades de gris que tienden a los valores de 255 unidades. Y a texturas con las mismas tonalidades. Esta sensibilidad se explica al observar la imagen del vector promedio μ . Ya que se considera la detección de una cara como aquel conjunto de vectores que comparten una tonalidad cercana al blanco. Este hecho se observa perfectamente en la imagen 25 donde el detector clasifica el brazo de la persona como un rostro, al igual que el fondo ruidoso que contiene niveles de gris similares al blanco. Si bien es cierto esta función de error tiene valores mucho mayores a los considerados por la función original en realidad la magnitud no nos interesa en sí, si no más bien la comparación entre los rostros almacenados en los eigenvectores y una clasificación más acertada. Que se logra al considerar la segunda función de error.

Se encontró que los valores de umbral funcionan como un hiperparámetro que varía de imagen a imagen. Esto nuevamente puede solucionarse al considerar imágenes capturadas con el mismo detector o cámara en lugar de emplear imágenes cuya resolución puede variar demasiado con respecto a las imágenes a detectar y las imágenes empleadas para obtener los eigenvectores.

4 CONCLUSIONES

Se empleó la técnica de PCA sobre un conjunto de imágenes de rostros cuya resolución de cada imagen era de (64, 64) en escala de grises. Mediante dicha técnica fue posible obtener los eigenvectores y eigenvalores de la matriz M mediante la descomposición SVD. Dichos eigenvectores y eigenvalores fueron útiles para realizar la reconstrucción de todas las imágenes en el conjunto de datos. A partir de la variación

de los eigenvectores para la reconstrucción fue posible obtener una gráfica del error de reconstrucción en función de dicha variación de los eigenvectores. Con dicha gráfica fue posible obtener una medida de la perdida de datos producida por la técnica de PCA, que para el caso de las imágenes de los rostros es relativamente baja con un valor aproximado de 30 unidades. Como complemento a esto se reconstruyeron seis imágenes de cuatro gatos y dos perros empleando los eigenvectores previamente calculados. Al igual que el caso anterior se obtuvo una gráfica del error de reconstrucción obteniéndose un valor mínimo aproximado de 850 unidades. Este valor no debe ser interpretado como el error producido por PCA ya que para este caso no se están empleando imágenes del mismo conjunto.

Una vez obtenido el conjunto de eigenvectores se procedió a analizar dos imágenes con resolución mayor al conjunto de datos empleado. Esto con el fin de detectar los rostros presentes en ambas imágenes. Para este fin se encontró que la función de error convencional ofrece resultados no deseados y de baja calidad. Por lo que al considerar una nueva función de error se obtienen mejores resultados que claramente son mejorables. Para el segundo caso se encontró que la función de error es sensible a intensidades cercanas a las 255 unidades.

Finalmente, el hecho de obtener cantidades apreciables de errores en ambos casos se asocia al hecho de que ambas imágenes no coinciden en resolución con respecto al conjunto de imágenes empleadas para obtener los eigenvectores y que el análisis de ambas imágenes consiste en ventanear cada imagen con las mismas dimensiones de las imágenes originales sin translaparse, por lo que no se asegura que dichas ventanas no aseguran cubrir todo el rostro tal y como sucede en las imágenes originales. Con ello en mente sería interesante analizar imágenes capturadas por el mismo detector para así comprobar si la hipótesis aquí planteada es verdadera o falsa.

El caso del ventaneo con translape no fue considerado debido al tiempo de ejecución.

REFERENCES

- [1] M. Santos, *Sensores de huellas dactilares: cómo funcionan y por qué no son tan seguros*, Redacción para Hardzone <https://hardzone.es/2018/04/20/sensores-huellas-dactilares/>. Consultado el 23.05.20
- [2] Wikipedia, *Facial recognition system*, https://en.wikipedia.org/wiki/Facial_recognition_system. Consultado el 23.05.20
- [3] M. Bishop, *Pattern Recognition and Machine Learning*, Springer, primera Ed., 2006.
- [4] C. Nikou, *Image Analysis: PCA and Eigenfaces*, Diapositivas de clase http://www.cs.uoi.gr/~cnikou/Courses/Image_Analysis/2010-2011/10_PCA_and_Eigenfaces_2spp.pdf. Consultado el 23.05.20