

Proyecto Final: Reconocimiento de Rostros

Universidad Nacional Autónoma de México

Facultad de Ingeniería

Reconocimiento de Patrones (757-1)

Basile Álvarez Andrés José

Keller Ascencio Rodolfo Andrés

Shen Shuai

*

Resumen—A lo largo de este documento se presentarán los resultados y análisis de nuestro proyecto final de la materia de Reconocimiento de Patrones, donde se estará trabajando con la implementación de la distintos modelos que sean capaces de realizar el reconocimiento de rostros a partir de imágenes, entre estos métodos hacemos uso del Análisis de Componentes Principales PCA para la reducción de la dimensionalidad de características de las imágenes, así como el uso de modelos SVC, MLP y CNN para el reconocimiento de rostros. Finalmente, se hace una comparativa entre estos modelos y se pone a prueba una aplicación práctica a partir del uso de Redes Convolucionales, la biblioteca DeepFace y el modelo VGG-Face.

I. INTRODUCCIÓN

I-A. Análisis de Componentes Principales

Los conjuntos de datos grandes son cada vez más comunes y a menudo resultan difíciles de interpretar. El análisis de componentes principales (PCA) es una técnica para reducir la dimensionalidad de dichos conjuntos de datos, aumentando la interpretabilidad pero minimizando la pérdida de información. Lo hace creando nuevas variables no correlacionadas que maximizan sucesivamente la varianza. Encontrar estas nuevas variables, conocidas como componentes principales, implica resolver un problema de valores propios y vectores propios. Los componentes principales están determinados por el conjunto de datos en cuestión, lo que hace que PCA sea una técnica de análisis de datos adaptativa [1]. Preservar la mayor cantidad de variabilidad posible se traduce en encontrar nuevas variables que son funciones lineales de las variables originales del conjunto de datos, que maximizan sucesivamente la varianza y que no están correlacionadas entre sí.

En el caso de imágenes, y específicamente de reconocimiento de rostros, se puede utilizar PCA para reducir la dimensionalidad de las mismas que, por ejemplo, para el caso del primer modelo trabajado (PCA/SVC) reduce la dimensionalidad de 74 imágenes de 64x64px (74,4096) a una matriz de (74,40), como se mostrará más adelante. Es importante mencionar que para poder utilizar PCA en imágenes (y en cualquier conjunto de datos), es necesario primeramente normalizar las imágenes para después poder calcular la matriz de covarianzas y los eigenvectores y eigenvalores que nos permitirán proyectar nuestro conjunto de datos a menos dimensiones.

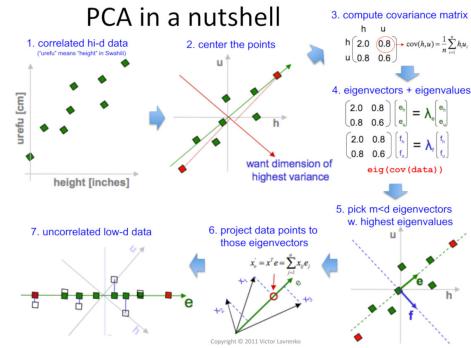


Figura 1: Los pasos de PCA, en resumen. [2]

I-B. Clasificación por Máquinas de Soporte Vectorial

Las máquinas de vectores de soporte (SVMs) son un conjunto de métodos de aprendizaje supervisado utilizados para clasificación, regresión y detección de valores atípicos. De [3], podemos ver las principales ventajas y desventajas de este tipo de clasificador:

Las ventajas de las máquinas de vectores de soporte son:

- Son efectivas en espacios de alta dimensionalidad (el espacio reducido igual contiene 40 dimensiones).
- Siguen siendo efectivas incluso cuando el número de dimensiones es mayor que el número de muestras.
- Utilizan solo un subconjunto de puntos de entrenamiento (llamados vectores de soporte) en la función de decisión, por lo que son eficientes en memoria.
- Son versátiles, ya que se pueden especificar diferentes funciones de kernel para la función de decisión. Se proporcionan funciones de kernel comunes, pero también es posible especificar funciones de kernel personalizadas.

Las desventajas de las máquinas de vectores de soporte incluyen:

- Si el número de características es mucho mayor que el número de muestras, es crucial evitar el sobreajuste al elegir funciones de kernel y términos de regularización.
- Las SVMs no proporcionan directamente estimaciones de probabilidad. Estas se calculan utilizando una validación cruzada de cinco pliegues, que puede ser costosa en términos computacionales.

I-C. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNN) son especialmente adecuadas para el procesamiento de señales de imagen, voz o audio. Estas redes se componen de capas convolucionales, capas de agrupación y capas totalmente conectadas (FC).

La capa convolucional es la primera capa de una CNN y es donde ocurren la mayoría de los cálculos. Utiliza filtros para realizar la convolución en campos receptivos de la imagen, extrayendo características espaciales como bordes, texturas y formas. Cada filtro aprende a detectar una característica específica en diferentes partes de la imagen.

Después de la capa convolucional, puede haber capas adicionales de convolución y capas de agrupación. La capa de agrupación reduce la dimensionalidad de la salida de las capas convolucionales, disminuyendo la cantidad de parámetros y mejorando la eficiencia del modelo. Puede realizar agrupación máxima o promedio para seleccionar el valor máximo o promedio dentro del campo receptivo respectivamente.

La capa totalmente conectada es la capa final de la CNN. En esta capa, cada nodo de salida está conectado directamente a los nodos de la capa anterior. Esta capa realiza la clasificación en base a las características extraídas por las capas anteriores y sus filtros. Utiliza una función de activación softmax para asignar probabilidades a las diferentes clases.

I-D. DeepFace

DeepFace es un paquete de reconocimiento de rostros híbrido, el cual cuenta con varios modelos empaquetados dentro de sí, entre ellos *VGG-Face*, *Google FaceNet*, *OpenFace*, *Facebook DeepFace*, *DeepID*, *ArcFace*, *Dlib* y *SFace*. Esta es una biblioteca ligera que permite realizar acciones como el reconocimiento de rostros y el análisis de atributos faciales de las personas como edad, género, emoción y raza.[4]

A partir del uso de esta biblioteca el usuario puede trabajar cinco etapas distintas del reconocimiento de rostros, entre las cuales tenemos la etapa de detección de la presencia de rostros en una imagen, la etapa de alineación y ubicación de los rostros dentro de la imagen, la normalización del rostro, la representación y finalmente la validación o verificación de dicho rostro. Gracias a esta biblioteca no es necesario que el usuario conozca o profundice en el funcionamiento de los modelos, puesto que únicamente deberá hacer uso de las funciones que vienen en ella gracias a que los modelos utilizados ya vienen preentrenados y listos para su aplicación.[4]

I-D1. VGG-Face: Uno de los modelos más utilizados para el reconocimiento de rostros es el modelo *VGG-Face*. Este modelo es una red neuronal convolucional que cuenta con veintidós capas y treinta y siete unidades de profundidad. La estructura de este modelo se puede apreciar en los siguientes gráficos.

En la sección anterior se había hablado acerca de lo que es una red convolucional, sin embargo, en esta sección profundizaremos un poco acerca de las capas que intervienen en este modelo, para así, entender lo que nuestro modelo realiza a partir de una imagen.

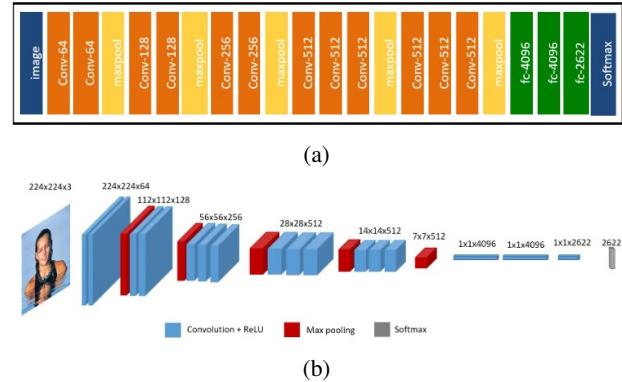


Figura 2: Capas que conforman la red convolucional *VGG-Face*[4]

Las capas que realizan convoluciones buscan aplicar filtros a la imagen para resaltar características relevantes de las fotografías, siendo que cada mapa de características obtenido representa la respuesta a un filtro aplicado por toda la imagen.

Las capas de *Zero padding*, buscan aumentar el tamaño de la imagen al añadir ceros alrededor de la misma, es decir, al borde, para que al momento de aplicar la convolución o un *Max pooling*, se preserve el tamaño espacial de la imagen y las operaciones no reduzcan progresivamente el tamaño de la imagen al verse afectado los bordes. Al momento de agregar ceros al borde, las operaciones se pueden realizar a lo largo de toda la imagen tomando en cuenta los bordes.

Por otro lado, las capas de *Max pooling* buscan submuestrear las imágenes reduciendo el tamaño de la imagen a partir de ventanas o mallas que buscan captar el valor máximo de cada región, y de esta manera, obtener una imagen reducida que contenga los valores máximos de las regiones submuestreadas.

Dentro de las últimas capas de este modelo, se cuentan con capas de *Dropout*, que buscan prevenir el sobreajuste al desactivar aleatoriamente un porcentaje de las neuronas de nuestras capas de entrenamiento, estableciendo la salida de estas neuronas desactivadas temporalmente a cero haciendo que estas neuronas no contribuyan a la propagación del gradiente. Dentro de este modelo se cuenta con dos capas de *Dropout* del 50 %.

Finalmente, la función de activación *Softmax*, es utilizada para obtener una distribución de probabilidad sobre múltiples clases mutuamente excluyentes. Esta función mapea los valores originales a un rango que vaya de 0 a 1, asegurando que la suma de todas las probabilidades sea 1. De esta manera, al trabajar problemas de clasificación multiclase busca asignar una probabilidad a cada una de las clases que participan en el problema de clasificación, donde la clase que cuente con un mayor valor será la clase a la cual se clasificará la imagen.

Tras haber comprendido cómo se encuentra construido este modelo, hablaremos acerca de la métrica de distancia utilizada para la evaluación de la clasificación o de la distancia entre imágenes.

I-D2. Métrica de distancia cosenoidal: La métrica de distancia cosenoidal utilizada funciona al multiplicar las imágenes a través de los vectores que las representan para obtener su producto escalar. Tras haber obtenido este escalar, se debe obtener la norma de los vectores normalizados de las imágenes cuya distancia queremos calcular, para finalmente obtener la similaridad cosenoidal al dividir el producto escalar de los vectores por el producto de sus normas. En este caso, este resultado que representa la similaridad cosenoidal es restado de 1, para que de esta manera entre más cercana sea la distancia cosenoidal a 1, más parecida sean las imágenes entre sí.

I-E. Conjunto de datos utilizado

Para la realización de esta práctica, se trabajó con el conjunto de datos de imágenes *Labeled Faces in the Wild* ó *LFW*, obtenido de la página <https://conradsanderson.id.au/lfwcrop/>, donde hicimos uso específicamente de la versión a color de las imágenes. En un inicio buscamos trabajar con todas aquellas personas que contaran con más de setenta imágenes recopiladas, sin embargo, a medida que fuimos trabajando con los distintos modelos decidimos trabajar con las personas que contaran con más de veinte imágenes de ellos mismos, pero redujimos el conjunto haciendo que cada una de estas personas contaran únicamente con diez imágenes.[5][6]



Figura 3: Carpetas que contienen el conjunto de imágenes utilizado.

De misma forma, cada uno de los integrantes del equipo decidió realizar sus propias capturas y añadir imágenes a este conjunto de datos en una carpeta que tuviera su nombre, siguiendo el formato del conjunto Ifw, y renombrando los archivos de las fotos al mismo formato que el del conjunto de datos Ifw.



Figura 4: Conjunto de imágenes de Rodolfo Keller.

Para las imágenes tomadas para la persona Rodolfo Keller, se decidió tomar una imagen frontal, una del lateral izquierdo de la persona y otra del lateral derecho para cada una de las variantes en el físico del estudiante. Entre estas variantes encontramos al estudiante con bigote, barba y lentes; bigote y barba; barba en forma de candado, bigote y lentes, barba en

forma de candado y bigote; bigote y lentes; bigote; únicamente lentes; y la cara limpia, sin ningún tipo de barba, bigote o accesorio.



Figura 5: Conjunto de imágenes de Shuai Shen.

Para el caso del estudiante Shuai Shen se decidió tomar imágenes en distintas locaciones para jugar con el fondo y el entorno, al igual que se decidió tomar como otra variable la distancia a la cual se tomaba la fotografía.

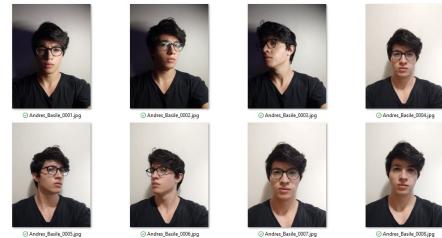


Figura 6: Conjunto de imágenes de Andrés Basile.

Por último para las imágenes del estudiante Andrés Basile se decidió tomar como variable la iluminación de la imagen, tomando fotografías con iluminación dentro del cuarto y sin iluminación dentro del cuarto.

Finalmente, se trabajó con 65 clases o personas para el reconocimiento de rostros, donde para algunos de los modelos trabajados decidimos reducir el conjunto de datos, pero este fue el conjunto de datos de partida para el proyecto.

II. OBJETIVO

El objetivo principal de esta práctica busca que el alumno logre comprender e implementar un algoritmo de reconocimiento de rostros mediante el método PCA o cualquier otro que deseé investigar.

III. DESARROLLO: RECONOCIMIENTO DE ROSTROS

III-A. Reconocimiento de rostros con PCA/SVC y PCA/MLP

Como primer acercamiento a la resolución del problema de reconocimiento de rostros en imágenes, optamos por la utilización conjunta de un algoritmo de reducción de dimensionalidad (PCA) y dos algoritmos de clasificación distintos (SVC y MLP). Para este método de resolución, trabajamos con un conjunto de imágenes que contiene aproximadamente 20 imágenes por cada persona, con un total de cinco personas (se trabajó con 74 imágenes en total). Dicho conjunto de imágenes fue leído y separado en un arreglo de imágenes de 64x64 pixeles aplanado (74, 4096) y un arreglo de etiquetas (74,) que describe a qué persona corresponde cada imagen.

Posteriormente, se normalizaron las imágenes utilizando el *Standard Scaler* de la biblioteca *sklearn*, el cual simplemente realiza:

$$z = \frac{x - \text{mean}}{\text{standard deviation}} \quad (1)$$

para todas las imágenes de nuestro conjunto de datos.



Figura 7: Ejemplo de imagen del conjunto de datos antes de la normalización utilizando *Standard Scaler*.



Figura 8: Ejemplo de imagen del conjunto de datos después de la normalización utilizando *Standard Scaler*.

Luego, se aplica el algoritmo de PCA de *sklearn*, escogiendo el número de componentes que retengan el 95 % de la varianza de nuestros datos, lo cual también puede ser interpretado como que queremos mantener el 95 % de la información

del conjunto de datos original. Aplicando la función *fit* y luego *transform*, logramos transformar las imágenes aplanadas con dimensiones (74,4096) para convertirlas en un conjunto de datos reducido prácticamente cien veces (74, 40), utilizando 40 componentes principales que, realizando una combinación lineal de ellas, permiten reconstruir la información manteniendo un 95 % de la varianza. No obstante, si quisieramos mantener un porcentaje de varianza menor, podríamos reducir incluso más el tamaño del conjunto de datos, utilizando un menor número de componentes principales.

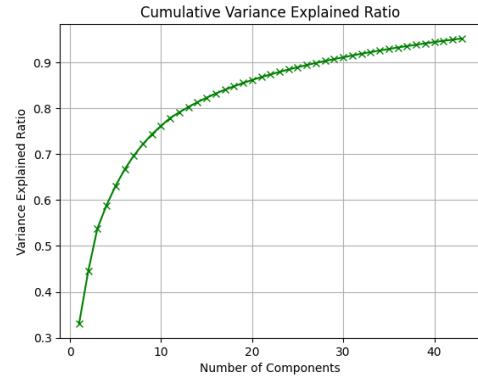


Figura 9: Si escogiéramos mantener un menor porcentaje de la varianza total (perdiendo más información sobre el conjunto de datos original), podríamos reducir la dimensionalidad del conjunto de datos aún más. La imagen muestra que, por ejemplo, para mantener el 80 % de la varianza de la información, necesitaríamos ocupar únicamente 13 componentes principales, resultando en un conjunto de datos reducido de (74,13).

Al trabajar con reconocimiento de rostros, a los vectores calculados por PCA se les conoce como *eigenfaces*. Dichas eigenfaces nos permiten reconstruir una imagen a partir de una combinación lineal de ellas. Existe una eigenface por cada componente principal que deseamos mantener en nuestro conjunto de datos reducido. Para nuestro conjunto de datos, podemos observar como cada una de ellas guarda distinta información sobre las imágenes. Las primeras eigenfaces (primeras componentes principales) resguardan la mayor cantidad de información sobre nuestro conjunto de datos; por ello, con únicamente 5 eigenfaces, podemos conservar el 60 % de la información total de las imágenes.



Figura 10: Las primeras 10 componentes principales (eigenfaces) para nuestras imágenes. Cada una de ellas conserva información sobre nuestro conjunto de datos original, siendo que la primera eigenface resguarda la mayor cantidad de información, seguida de la segunda, tercera, etc.

Una vez reducido el conjunto de datos original, podemos aplicar algoritmos de clasificación para lograr el reconoci-

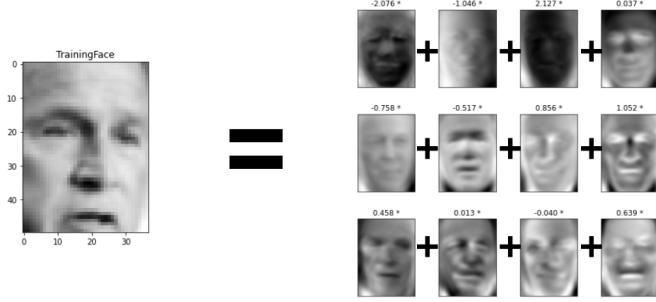


Figura 11: A partir de las eigenfaces, podemos reconstruir las imágenes de nuestro conjunto de datos original como una combinación lineal de éstas, manteniendo hasta un 95 % de la información.

miento de rostros. En nuestro caso, primeramente aplicamos un clasificador por máquina de soporte vectorial (SVC), tal como es descrita en [7]. Para ello, dividimos el conjunto de datos con dimensionalidad reducida utilizando la función `train_test_split`, manteniendo el 80 % de las imágenes para entrenamiento y el 20 % como conjunto de prueba. El clasificador por máquina de soporte vectorial utiliza un kernel polinomial, que permite clasificar datos aún cuando se cuenta con dimensionalidades altas. La SVC encuentra el mejor estimador probando con varios valores de regularización que son pasados al modelo como parámetro. En nuestro caso, observamos que el mejor parámetro de regularización fue de 100 para un kernel polinomial.

| | precision | recall | f1-score | support |
|-----------------|-----------|--------|----------|---------|
| Shuai Shen | 1.00 | 0.50 | 0.67 | 2 |
| Rodolfo Keller | 0.75 | 0.75 | 0.75 | 4 |
| Andres Basile | 0.80 | 1.00 | 0.89 | 4 |
| Vicente Fox | 1.00 | 1.00 | 1.00 | 1 |
| Nestor Kirchner | 0.50 | 0.50 | 0.50 | 4 |
| accuracy | | | 0.73 | 15 |
| macro avg | 0.81 | 0.75 | 0.76 | 15 |
| weighted avg | 0.75 | 0.73 | 0.73 | 15 |

Figura 12: La SVC para clasificación de rostros obtuvo estos valores de precisión para las cinco personas con las que fue entrenada y probada. Observamos un porcentaje de precisión media elevado (81 %), pero ligero sobreajuste en las clases *Shuai Shen* y *Vicente Fox*. Como ejercicio posterior, se podrá ajustar los parámetros de regularización y cambiar el kernel de la máquina de soporte vectorial en búsqueda de un menor sobreajuste; o incluso se podrían agregar más imágenes para cada clase o hacer algún otro ajuste a los datos o al modelo.

De manera similar, implementamos la clasificación de rostros utilizando un perceptrón multicapa (MLP). El perceptrón multicapa puede verse como una red neuronal simple, completamente conectada con únicamente una capa oculta. En nuestro MLP, la entrada son imágenes reducidas por PCA (1,40), luego una capa oculta de 512 nodos y finalmente 5 nodos de salida, donde aquel con mayor valor al finalizar el entrenamiento representará la clase a la que pertenece una imagen. En la

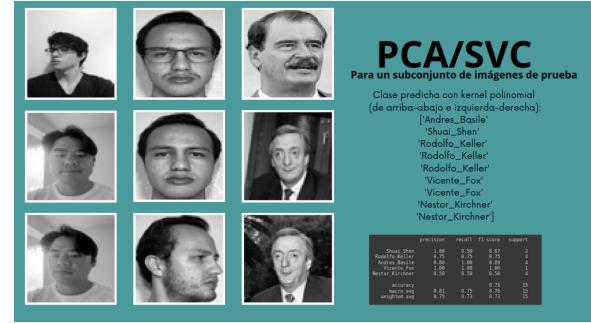


Figura 13: Para la clasificación con SVC, vemos en esta imagen las categorías reales y las predichas por el modelo en un subconjunto del conjunto de imágenes de prueba.

implementación del MLP, se utilizó la biblioteca descrita en [8], pasando como parámetro de entrenamiento el que se dicho entrenamiento se detenga cuando entre una época y otra no haya un cambio en la pérdida del MLP. Para este segundo clasificador implementado, obtuvimos una precisión promedio de 81 %, nuevamente con un posible sobreajuste en las clases *Vicente_Fox* y *Shuai_Shen*, que podría eliminarse cambiando la configuración del perceptrón multicapa.

| | precision | recall | f1-score | support |
|-----------------|-----------|--------|----------|---------|
| Shuai Shen | 1.00 | 0.50 | 0.67 | 2 |
| Rodolfo Keller | 0.75 | 0.75 | 0.75 | 4 |
| Andres Basile | 0.80 | 1.00 | 0.89 | 4 |
| Vicente Fox | 1.00 | 1.00 | 1.00 | 1 |
| Nestor Kirchner | 0.50 | 0.50 | 0.50 | 4 |
| accuracy | | | 0.73 | 15 |
| macro avg | 0.81 | 0.75 | 0.76 | 15 |
| weighted avg | 0.75 | 0.73 | 0.73 | 15 |

Figura 14: El MLP para clasificación de rostros obtuvo estos valores de precisión para las cinco personas con las que fue entrenada y probada. Observamos un porcentaje de precisión media elevado (81 %), pero ligero sobreajuste en las clases *Shuai Shen* y *Vicente Fox*. Como ejercicio posterior, se podrá ajustar los parámetros del perceptrón en búsqueda de un menor sobreajuste.

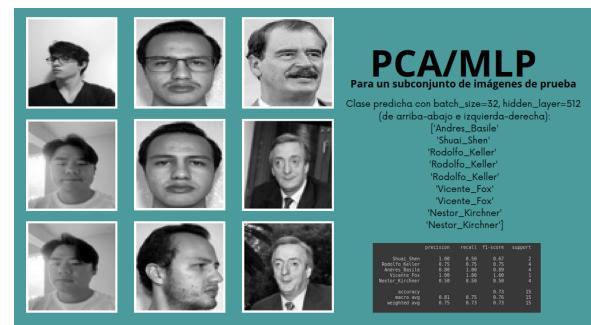


Figura 15: Para la clasificación con MLP, vemos en esta imagen las categorías reales y las predichas por el modelo en un subconjunto del conjunto de imágenes de prueba.

III-B. Reconocimiento de rostros con DeepFace y VGG-Face

Como segundo acercamiento a la solución del problema buscamos hacer uso de una biblioteca existente, por lo que al momento de estar realizando nuestra investigación nos dimos cuenta de la existencia de la biblioteca *DeepFace*, el cual es un *framework* de reconocimiento de rostros y análisis de atributos faciales para Python. Este *framework* de reconocimiento de rostros incluye modelos como lo son: *VGG-Face*, *Google FaceNet*, *OpenFace*, *Facebook DeepFace*, *DeepID*, *ArcFace*, *Dlib* y *SFace*.[4]

La principal razón por la cual decidimos hacer uso de esta biblioteca fue debido a que incorpora una enorme cantidad de modelos con una alta precisión debido a los grandes conjuntos de entrenamiento y pruebas con los que son construidos.

| Model | LFW Score | YTF Score |
|---------------------|-----------|-----------|
| Facenet512 | 99.65% | - |
| SFace | 99.60% | - |
| ArcFace | 99.41% | - |
| Dlib | 99.38 % | - |
| Facenet | 99.20% | - |
| VGG-Face | 98.78% | 97.40% |
| <i>Human-beings</i> | 97.53% | - |
| OpenFace | 93.80% | - |
| DeepID | - | 97.05% |

Figura 16: Precisión de los modelos que trabaja la biblioteca *DeepFace*

Dentro de esta biblioteca contamos con distintas funciones que nos permiten realizar, entre otras cuestiones, una verificación de individuos a través de imágenes para comprobar si las personas que se encuentran dentro de dos imágenes distintas son la misma o no; el reconocimiento de rostro a través de una base de datos para obtener las imágenes de la persona que se relaciona a una imagen de entrada; transformar imágenes directamente a vectores representativos a través del uso de *embeddings*; y el análisis de atributos faciales como la edad, el género, la raza y la emoción de una persona, entre otros aspectos.

A partir de estas funcionalidades decidimos trabajar y poner a prueba esta biblioteca, con lo cual inicialmente decidimos hacer uso de la función *verify()*, para verificar si dos imágenes se trataban, o no, de la misma persona. En un primer caso trabajamos con dos imágenes de Andrés Basile, donde esta función nos indicó que efectivamente las imágenes se trataban de la misma persona, siendo que se evaluó esta similitud a partir del modelo *VGG-Face*, con una distancia cosenoideal de similitud de 0.3368.

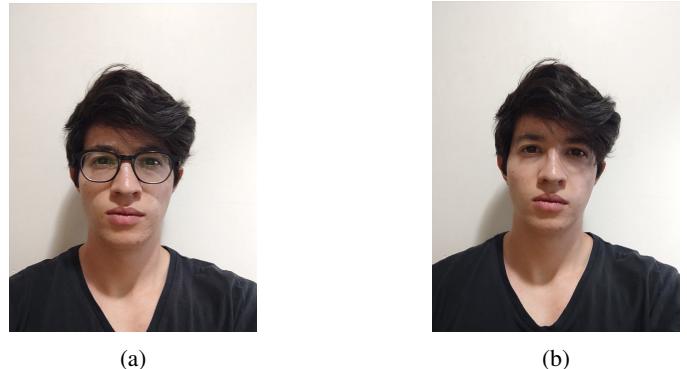


Figura 17: Imágenes comparadas de una misma persona para verificar la identidad del individuo.



Figura 18: Imágenes comparadas de distintas persona para verificar la identidad del individuo.

Por otro lado, al trabajar con una comparación de imágenes entre Rodolfo Keller y Shuai Shen, obtuvimos que las imágenes no pertenecían a la misma persona, realizando la medición de distancia a partir de la métrica cosenoideal que nos dió de 0.53.

Por otra parte, como siguiente función a evaluar trabajamos con *analyze()*, que es una función que permite evaluar los atributos faciales de las personas a partir de una imagen. En este caso, evaluamos los atributos faciales de Rodolfo Keller, a partir de las imágenes que considerábamos eran más distintas entre sí.

Esto resultó ser sumamente interesante, puesto que a pesar de tratarse de la comparación de los atributos faciales de un mismo individuo, para la imagen 19 (a), se obtuvo una edad de 24 años, con un género dominante masculino, una raza blanca dominante y una emoción dominante de tristeza, mientras que para la imagen 19 (b), se observa una edad de 34 años, con un género dominante masculino, una raza dominante de latino hispano y una emoción dominante neutral. Se tiene una gran diferencia en la edad, así como en la raza de la persona a través del análisis de atributos faciales, todo gracias a que el individuo se quitó el bigote, la barba y los lentes.

Por último, se procedió a trabajar con la función que más nos interesaba, la cual era *find()*. Esta función nos permitía



(a)



(b)

Figura 19: Comparación de salida de atributos faciales de una misma persona

realizar la búsqueda de imágenes similares a la imagen de una persona a partir de una base de datos. De esta forma, al ingresar una imagen podíamos obtener como respuesta un *DataFrame* formado por una serie de imágenes que pudieran, o no, pertenecer a la misma persona junto con el valor de la métrica de distancia cosenoide que nos indicaba que tan parecido o no era la persona que se presentaba en una imagen con respecto a otra.

Tras haber trabajado un poco con la salida de esta función, pudimos obtener un *DataFrame* que nos presentara el nombre de la persona a la que pertenecía la foto semejante a la entrada, el número de la foto de dicha persona y el valor de la métrica coseno que nos indicaba qué tanto se parecían las personas en ambas imágenes.



Figura 20: Ejemplo del reconocimiento de imágenes similares de personas a partir de una entrada

Posteriormente, esta función la modificamos de tal forma que nos entregara únicamente el nombre de la persona a la cual más se parecía el individuo que se encontraba dentro de una imagen, siendo que esta función nos permitió crear posteriormente una aplicación que reconociera imágenes en tiempo real a través de video, tal y como se explicará en la siguiente sección.

III-C. Reconocimiento de rostros con TensorFlow y Keras

El último enfoque que decidimos utilizar para resolver el problema planteado en esta práctica es la biblioteca Keras de TensorFlow. Esta opción resulta beneficiosa debido a su capacidad para trabajar con imágenes y modelos de aprendizaje profundo. Keras ofrece una amplia variedad de capas y funciones de procesamiento de imágenes, como convoluciones, max pooling y flatten, las cuales son fundamentales para la detección y extracción de características faciales. Además, la facilidad de uso de TensorFlow Keras permite construir y entrenar modelos de clasificación facial de manera eficiente, aprovechando su capacidad de procesamiento paralelo y optimización de hardware.

El procedimiento para utilizar este modelo es la siguiente:

1. Preprocesamiento de datos:

- Se cargan las imágenes de las caras y se aplican transformaciones de preprocesamiento, como la normalización de los valores de píxeles para que estén en el rango de 0 a 1.

2. Generadores de datos:

- Se crean generadores de datos para el conjunto de entrenamiento y el conjunto de validación.
- Estos generadores cargan las imágenes desde un directorio y generan lotes de imágenes junto con sus etiquetas correspondientes.

3. Construcción del modelo:

- Se define una arquitectura de red neuronal convolucional (CNN) utilizando capas convolucionales y de agrupación máxima (max pooling).
- Las capas convolucionales aprenden a extraer características de las imágenes, mientras que las capas de agrupación máxima reducen la dimensionalidad y destacan características importantes.
- Se añaden capas completamente conectadas (densas) para clasificar las características extraídas.

4. Compilación y entrenamiento del modelo:

- Se compila el modelo especificando la función de pérdida y el optimizador que se utilizarán durante el entrenamiento.
- El modelo se entrena utilizando el generador de datos de entrenamiento, ajustando los pesos de la red para minimizar la pérdida.
- El rendimiento del modelo se evalúa utilizando el generador de datos de validación.

5. Evaluación del modelo:

- Se crea un generador de datos para el conjunto de prueba y se utiliza para evaluar el rendimiento final del modelo.
- Se calcula la pérdida y la precisión del modelo en el conjunto de prueba.

6. Guardar el modelo entrenado:

- El modelo entrenado se guarda en un archivo para su uso posterior.

Utilizando un directorio con 5 clases y alrededor de 15 imágenes en cada una, obtuvimos los siguientes resultados:

| | precision | recall | f1-score | support |
|----------------|-----------|--------|----------|---------|
| Andres_Basile | 1.00 | 1.00 | 1.00 | 1 |
| Angelina_Jolie | 1.00 | 0.50 | 0.67 | 2 |
| Rodolfo_Keller | 1.00 | 1.00 | 1.00 | 4 |
| Shuai_Shen | 1.00 | 1.00 | 1.00 | 3 |
| Vicente_Fox | 0.67 | 1.00 | 0.80 | 2 |
| accuracy | | | 0.92 | 12 |
| macro avg | 0.93 | 0.90 | 0.89 | 12 |
| weighted avg | 0.94 | 0.92 | 0.91 | 12 |

Figura 21: Precisión final de 91 %

Posteriormente realizamos algunas pruebas con imágenes independientes y esto es el resultado



Figura 22: Pruebas realizadas

III-D. Detección de rostros

Finalmente, decidimos implementar nuestros modelos de clasificación en una aplicación real, utilizando la biblioteca de `face_recognition` para crear un programa capaz de detectar nuestro rostro, clasificarlo según cualquiera de los modelos hechos anteriormente y encerrarlo en un cuadro. En términos generales, el programa accede a la cámara y configura la

resolución y velocidad de los fotogramas deseados. Posteriormente, en un bucle infinito, captura un fotograma, lo convierte al formato RGB y verifica si ha transcurrido un segundo. Si ha pasado un segundo, se detectan los rostros en el fotograma y, en caso de encontrar al menos uno, se realiza la clasificación correspondiente. Por último, se dibuja un cuadro alrededor del rostro detectado y se muestra el resultado de la clasificación debajo del cuadro durante un segundo. Por último, el fotograma con la clasificación se muestra en una ventana.

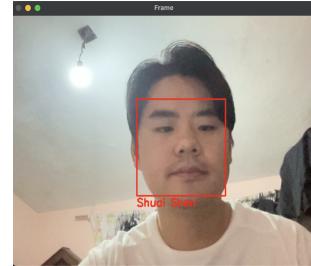


Figura 23: Detección y clasificación de rostro

IV. CÓDIGO

Durante el desarrollo de esta práctica, se utilizó Google Colab como herramienta de gestión de versiones del código, así como método para trabajar de forma remota con el equipo. Se presentan algunos elementos importantes del código a continuación:

IV-A. PCA

IV-A1. Leyendo las imágenes y realizando el primer preprocessamiento:

```
import zipfile
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
from PIL import Image
from sklearn.decomposition import PCA

def read_images_from_folder(folder_path):
    images = []
    labels = []
    persons = os.listdir(folder_path)
    for person in persons:
        person_folder = os.path.join(folder_path, person)
        if not os.path.isdir(person_folder):
            continue
        for file_name in os.listdir(person_folder):
            image_path = os.path.join(person_folder, file_name)
            try:
                image = Image.open(image_path)
                image = image.convert('L') # Convert to grayscale
                image = image.resize((64, 64)) # Resize if needed
                image = np.array(image).flatten() # Flatten the image
                images.append(image)
                labels.append(person)
            except Exception as e:
                print(f"Error reading image {image_path}: {str(e)}")
    return np.array(images), np.array(labels)

folder_path = '/content/drive/MyDrive/Proyecto_Final_RP/My_Dataset/PCA_DATASET/lfw_20_exactas_5/'

images, labels = read_images_from_folder(folder_path)
```

IV-A2. Normalización de las imágenes:

```
def normalize_data(data):
    scaler = StandardScaler()
    normalized_data = scaler.fit_transform(data)
    return normalized_data
```

IV-A3. Aplicación de PCA:

```
pca = PCA(n_components=0.95) # Retain 95% of the variance
pca.fit(normalized_data)

# Transform the flattened images using the PCA model
transformed_images = pca.transform(normalized_data)
```

IV-A4. Clasificación con SVC:

```
print("Ajustando el clasificador al conjunto de entrenamiento.")
t0 = time()
param_grid = {'C': [1e2, 1e3, 5e3, 1e4, 5e4, 1e5], 'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }

# Hacemos uso de una Máquina de Soporte Vectorial como nuestro algoritmo de clasificación
clf = GridSearchCV(SVC(kernel='poly', class_weight='balanced'), param_grid)
clf = clf.fit(X_train, y_train)
print("Realizado en %0.3fs \n" % (time() - t0))
print("El mejor estimador encontrado para la búsqueda:")
print(clf.best_estimator_)

print("\nPrediciendo el nombre de las personas en el conjunto de prueba:")
t0 = time()
y_pred = clf.predict(X_test)
print("Realizado en %0.3fs\n" % (time() - t0))
```

IV-A5. Clasificación y predicción con MLP:

```
clf = MLPClassifier(hidden_layer_sizes=(512,), batch_size=32, verbose=False,
                     early_stopping=True).fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred, target_names=target_names))
```

IV-B. DeepFace

IV-B1. Importando las bibliotecas que vamos a utilizar:

```
!pip install deepface --user
from deepface import DeepFace
from deepface.basemodels import VGGFace
import pandas as pd
import os
```

IV-B2. Uso de función para verificación de personas:

```
result = DeepFace.verify(img1_path = "lfw_10_train\Andres_Basile\Andres_Basile_0007.jpg", img2_path =
    "lfw_10_train\Andres_Basile\Andres_Basile_0008.jpg")
result
```

IV-B3. Uso de función para análisis de atributos faciales:

```
objs = DeepFace.analyze(img_path = "lfw_10_train\Rodolfo_Keller\Rodolfo_Keller_0020.jpg", actions = ['age',
    'gender', 'race', 'emotion'])
objs
```

IV-B4. Definición de funciones para el reconocimiento de personas a partir de una base de datos:

```
def extract_first_name(identity):
    # Divide la cadena de la ruta en partes
    parts = os.path.normpath(identity).split(os.path.sep)
    # Obtiene el número de imagen (último fragmento de la ruta)
    picture_number = parts[2].split("_")[-1].split(".")[-1]
    # Obtiene el primer nombre después de las dos carpetas de inicio
    first_name = parts[2].split("_")[0] + " " + parts[2].split("_")[1]
    return first_name, picture_number

# model = VGGFace.loadModel()

def FaceRecognition(img_path_usr, db_path_usr):
    dfs2 = DeepFace.find(img_path = img_path_usr, db_path = db_path_usr, model_name = "VGG-Face",
        distance_metric = "cosine")
    dfs2_convertido = pd.DataFrame(dfs2[0], columns=['identity', 'source_x', 'source_y', 'source_w',
        'source_h', 'VGG-Face_cosine'])
    dfs2_convertido_nombre = dfs2_convertido

    # Aplica la función a la columna 'identity' y crea dos nuevas columnas 'first_name' y 'num_imagen'
    dfs2_convertido[['first_name', 'num_imagen']] =
        dfs2_convertido['identity'].apply(extract_first_name).apply(pd.Series)
    # Columnas deseadas
    dfs2_salida = dfs2_convertido_nombre[['first_name', 'num_imagen', 'VGG-Face_cosine']]
    # Damos formato a la salida de la métrica de distancia
    dfs2_salida['VGG-Face_cosine'] = dfs2_salida['VGG-Face_cosine'].apply(lambda x: '{:.6f}'.format(x))
    return dfs2_salida

def BestRecognition(img_path_usr, db_path_usr):
    dfs2_salida = FaceRecognition(img_path_usr, db_path_usr)
    if dfs2_salida.loc[0, 'VGG-Face_cosine'] == 0:                      #En caso de reconocer la misma imagen
        first_name = dfs2_salida.loc[1, 'first_name']
    else:
        first_name = dfs2_salida.loc[0, 'first_name']
    return str(first_name)
```

IV-B5. Uso de funciones para el reconocimiento de personas a partir de una base de datos:

```
df = BestRecognition("lfw_10_train\Rodolfo_Keller\Rodolfo_Keller_0020.jpg", "lfw_10_train")
df
```

IV-B6. Uso de funciones para obtener imágenes similares a una persona dentro de una imagen deseada a partir de una base de datos:

```
df = FaceRecognition("lfw_10_train\Rodolfo_Keller\Rodolfo_Keller_0020.jpg", "lfw_10_train")
df
```

IV-C. CNN TensorFlow

IV-C1. Preprocesamiento:

```
# Directorio de datos en Google Drive
data_dir = '/Users/shuaishen/Desktop/lfw_20_exactas_5'

# Preprocesamiento de datos
datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    validation_split=0.2 # Porcentaje para validación (ajusta según tus necesidades)
)

batch_size = 5
img_height = 224
img_width = 224

# Generadores de datos
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

val_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

num_classes = train_generator.num_classes
class_indices = train_generator.class_indices
class_names = list(class_indices.keys())
print(class_names)
```

IV-C2. Construcción del modelo:

```
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(img_height, img_width, 3)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
```

IV-C3. Entrenamiento y validación:

```
# Compilación y entrenamiento del modelo
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

num_epochs = 6 # Ajusta el número de épocas según tus necesidades

model.fit(train_generator, epochs=num_epochs, validation_data=val_generator)

# Guardar el modelo entrenado en Google Drive
model.save('/Users/shuaishen/Desktop/Reconocimiento de patrones/trained_model_local.h5')

# Evaluación del modelo
test_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(img_height, img_width),
    batch_size=batch_size,
    class_mode='categorical',
```

```
subset='validation',
shuffle=False
)

loss, accuracy = model.evaluate(test_generator)
print("Accuracy on test data:", accuracy)
```

V. CONCLUSIONES

A lo largo de esta proyecto final, se tuvo como principal objetivo que el alumno lograra comprender e implementar un algoritmo de reconocimiento de rostros mediante el método PCA o cualquier otro que deseara investigar. Partiendo de esta idea, consideramos que el objetivo del proyecto final se cumplió en su totalidad debido a que los alumnos fueron capaces de comprender y construir distintos modelos que fueran capaces de realizar el reconocimiento de rostros, donde dos de estos modelos fueron construidos en su totalidad como lo fue el caso de PCA/SVC y PCA/MLP; logramos implementar un modelo preexistente para el reconocimiento de rostros como lo fue en el caso del uso de la biblioteca *DeepFace* y del modelo *VGG-Face*; y finalmente desarrollamos una red neuronal convolucional CNN a partir del uso de *Tensorflow* y *Keras*.

Para los modelos desarrollados por nosotros haciendo uso del Análisis de Componentes Principales obtuvimos un 81 % de precisión para ambos casos, trabajando con cinco clases y un total de setenta y cuatro imágenes en total, donde se logró reducir las características esenciales a 40 componentes, manteniendo el 95 % de la varianza, mientras que al trabajar con únicamente 13 componentes principales seguimos manteniendo el 80 % de la varianza, lo cual reduce significativamente la dimensionalidad de la información con la que trabajamos.

Como segunda aproximación a la resolución de este problema decidimos hacer uso de la biblioteca *DeepFace*, a través de la cual se reconoce que se cuenta con más de un 98 % de precisión para el reconocimiento de personas al trabajar con el conjunto de datos *Labeled Faces in the Wild*, mismo conjunto de datos con el cual nosotros estuvimos trabajando, por lo que al momento de trabajar con este modelo logramos hacer uso de las funciones que vienen por defecto en la biblioteca para reconocer características faciales de las imágenes y así realizar un análisis facial de las personas, así como logramos reconocer la similitud que existía entre personas en distintas imágenes. Por otro lado, se logró modificar las funciones de la biblioteca para crear una función que permitiera el reconocimiento de personas a partir de una imagen, buscando similitudes en imágenes dentro de un conjunto de datos.

Por último, como aproximación final a esta problemática decidimos implementar una red neuronal convolucional haciendo uso de la biblioteca de *tensorflow* y *keras*, que lograra realizar el reconocimiento de rostros a partir del uso de distintas capas que fueran modificando la imagen.

Tras haber realizado estos modelos, se logró desarrollar una aplicación que fuera capaz de realizar detección de rostros, localizar el rostro dentro de un área definida y finalmente lograr la clasificación de dicho rostro en tiempo real a través de los distintos modelos desarrollados, siendo que el modelo *VGG-Face* trabajado a partir de la biblioteca *DeepFace*, fue el que nos proporcionó una mayor precisión al momento de realizar el reconocimiento de rostros en tiempo real.

Finalmente, podemos decir que el trabajo realizado a lo largo de la práctica nos permitió reconocer el funcionamiento

de distintos modelos al aprender más acerca del proceso de normalización de imágenes y de la reconstrucción de imágenes a partir de la suma de componentes principales conocidos como *eigenfaces*, así como logramos comprender el funcionamiento e implementación de modelos como SVC, MLP, CNN en el ámbito del uso de imágenes y reconocimiento de rostros.

En conclusión, consideramos que este proyecto final fue excelente debido a que nos permitió comprender con mayor detalle la importancia del manejo y la normalización de imágenes; la construcción, funcionamiento y uso de distintos modelos para el reconocimiento de rostros; así como poner a prueba nuestras habilidades para la creación de aplicaciones que puedan ser utilizadas en la vida cotidiana, por ejemplo, en sistemas de reconocimiento de caras para la validación de personal, sistemas de reconocimiento de rostros para el control de acceso, entre muchos otros usos. Este fue un proyecto final que permitió que los alumnos pusieran a prueba todos los conceptos aprendidos a lo largo del curso.

REFERENCIAS

- [1] I. T. Jolliffe, “Principal component analysis: a review and recent developments,” 2016.
- [2] A. Vidhya, “An end to end comprehensive guide for pca.” <https://www.analyticsvidhya.com/blog/2020/12/an-end-to-end-comprehensive-guide-for-pca/>,” 2020.
- [3] S.-K. Learn, “I.4. support vector machines,” 2020.
- [4] serengil, “Deepface - github - <https://github.com/serengil/deepface>,” 2023.
- [5] C. Sanderson, “Lfwcrop face dataset - <https://conradsanderson.id.au/lfwcrop/>,” 2009.
- [6] U. of Massachusetts, “Labeled faces in the wild - <http://vis-www.cs.umass.edu/lfw/>,” 2018.
- [7] S.-K. Learn, “sklearn.svm.svc,” 2020.
- [8] ———, “sklearn.neural_network.mlpclassifier,” 2020.