ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

COMPARATIVA DE EXTRACTORES DE CARACTERÍSTICAS PARA CLASIFICACIÓN DE ROSTROS

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN SISTEMAS INFORMÁTICOS Y DE COMPUTACIÓN

GRANDA CÁRDENAS AIDA ALEXANDRA

DIRECTOR: PAZ ARIAS HENRY PATRICIO
CODIRECTOR: BENALCÁZAR PALACIOS MARCO ENRIQUE

Quito, Diciembre 2019

AVAL

Certificamos que el presente trabajo fue desarrollado por Alexandra Granda, bajo nuestra supervisión.

HENRY PAZ
DIRECTOR

MARCO BENALCÁZAR
CODIRECTOR

DECLARACIÓN DE AUTORÍA

Yo, Aída Alexandra Granda Cárdenas declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su Reglamento y por la normatividad institucional vigente.

ALEXANDRA GRANDA

DEDICATORIA

Dedico el presente proyecto de investigación a mis padres, quienes con su apoyo constante y su amor permitieron que pueda culminar esta etapa universitaria exitosamente. Además, a mis abuelos, quienes siempre esperaron ver este logro mío.

A mis hermanos, quienes merecen párrafo aparte porque supieron ser mi guía, mi compañía y mi referencia al momento de seguir mi carrera universitaria. Gracias por jamás ponérmelo fácil.

Pero sobre todo dedico este trabajo a mi madre, ella siempre estuvo a mi lado y con sus consejos, acertados o no, su apoyo y su palabra de aliento, fue indispensable para lograr este paso.

Alexandra.

AGRADECIMIENTO

Agradezco a Dios por ser mi guía en esta etapa universitaria y por permitirme culminarla exitosamente.

A mi madre Aída y a mi padre Randy, quienes me han enseñado que a pesar de todas las dificultades que se presenten en la vida, la constancia y la confianza en Dios son las únicas herramientas necesarias para lograr cualquier cosa que me proponga. Su fe en mi, su creencia en que soy mejor de lo que me pienso, su apoyo y su palabra de aliento infaltable fueron las herramientas principales para mi camino en esta universidad.

A mis hermanos, Alexander y Ariel, por su apoyo en los momentos más difíciles de mi carrera, por amanecerse conmigo cuando lo necesité, por siempre estar pendientes de mi a pesar de no estar siempre juntos. A ellos, les digo simplemente, gracias.

A Daniela, mi mejor amiga de la universidad, porque fue mi compañera de desgracias y ocurrencias, porque durante muchos años nos apoyamos mucho la una a la otra, y por haberme demostrado que a pesar de las circunstancias seguimos siendo buenas amigas, gracias.

A Victoria, Nathaly, José Miguel, Jose, Santiago, Jefferson, David, Sebastián y Cristian, con quienes he compartido los momentos más felices de mi vida universitaria. Les agradezco por estos años de amistad, porque probablemente sin ustedes me hubiera tardado más en terminar mis estudios, y por regalarme las mejores memorias estudiantiles. A quienes les digo que son también mi familia.

A mis tutores del proyecto de investigación Henry y Marco, quienes con su conocimiento y apoyo permitieron que este trabajo sea concluido satisfactoriamente.

Alexandra.

ÍNDICE DE CONTENIDO

RESUMEN

El presente trabajo propone realizar una comparativa entre dos métodos automatizados de extracción de características de imágenes para clasificación de rostros. Los métodos propuestos son: primero, método basado en extracción de vectores de características de las imágenes mediante una red neuronal convolucional; segundo, método basado en la extracción de características como en el primer caso, pero añadiendo una capa de clusterización usando para ellos el algoritmo KMeans, y luego comparar si el refinamiento de características mejora el rendimiento del proceso posterior de clasificación de las mismas. Los algoritmos a usar en la extracción de características son: una red neuronal profunda con la arquitectura GoogleNet en su versión InceptionV3, Kmeans para la clusterización; y dos algoritmos de clasificación supervisados para validar la calidad de las características, uno de ellos kNN y el otro SVM con un kernel lineal. Al final se comparará los resultados obtenidos con las cuatro pruebas, los dos algortimos de clasificación en los dos escenarios: con KMeans y sin KMeans, y se verificará si el agregar la fase de clusterización de características realmente aporta al rendimiento de la clasificación supervisada.

PALABRAS CLAVE: Extracción de características, clasificación de rostros, aprendizaje de máquina

ABSTRACT

This work proposes realizing a comparative between two image feature extraction automatized methods applied to the face classification problem. The methods proposed are: first, extraction of feature vectors from images through a convolutional neural network, and then compare and determine if refinement of these features through an unsupervised machine learning algorithm actually improves the posterior performance of the classifier algorithms. The algorithms to be used in the feature extraction are the following: a deep neural network with the GoogLeNet arqchitecture in it's InceptionV3 version, KMeans for unsupervised learning and two supervised learning algorithms to validate the quality of the features extracted. At the end a comparison will be made between the four results, and it will be verified if the feature refinement through the unsupervised machine learning algorithm really enhances the performance of supervised machine learning algorithms.

KEYWORDS: feature extraction, face detection, machine learning

1. INTRODUCCIÓN

Las redes neuronales convolucionales han mostrado efectividad a la hora de realizar trabajos relacionados con visión por computadora, reconocimiento de patrones en imágenes y clasificación de objetos. En otros trabajos se ha usado a las redes neuronales convolucionales como extractores de características de imágenes, para luego procesar estas características con otros algoritmos de clasificación supervisados. [1]

La red neuronal de arquitectura GoogleNet, en su versión InceptionV3 [2], ha demostrado ser uno de los modelos más efectivos para extracción de características [3,4,5]. Por esa razón, en el presente trabajo se propone dos arquitecturas de extracción de características, una que usa una red neuronal convolucional para extraer los vectores de características de las imágenes, y una segunda que propone la red neuronal integrada con un algoritmo de aprendizaje no supervisado, KMeans [6], para agrupar estas características. El interés de usar estos algoritmos se basa en que, como se menciona anteriormente, la red neuronal como extractor de características ha demostrado ser efectivo para la tarea [3, 4, 5] y se pretende usarla como un método de extracción de características más automatizado. El algoritmo KMeans, en cambio, se ha usado en varios trabajos como agrupador de características y ha demostrado efectividad [4, 5] en la tarea, por lo cual se lo usa también en el presente trabajo. También se pretende validar si la segunda arquitectura de hecho se comporta mejor que solo la red neuronal convolucional.

Luego de haber aplicado estos procesos a las imágenes, se procederá a evaluar las características mediante dos algoritmos de clasificación, que serán entrenados y testeados con estas características: kNN [7] y Support Vector Machine [8], también conocida como SVM o máquinas de vectores de soporte. Se seleccionan éstos algoritmos ya que se quiere evaluar el comportamiento de las características en dos escenarios: un algoritmo de clasificación lineal (SVM con kernel lineal) y un algoritmo de clasificación no lineal (KNN).

Para lograr este objetivo, se usará la metodología propia de la investigación con aprendizaje de máquina, la cual es frecuentemente usada en trabajos de investigación acerca del aprendizje de máquina, ya sea con algoritmos

supervisados o no supervisados [4, 9]. Ésta metodología consiste de dos fases principales: la fase de diseño, entrenamiento y validación preliminar, y la fase de testeo.

A continuación se explicará un poco más de la metodología tradicional de aprendizaje de máquina, es decir, las fases que comprenden el presente trabajo.

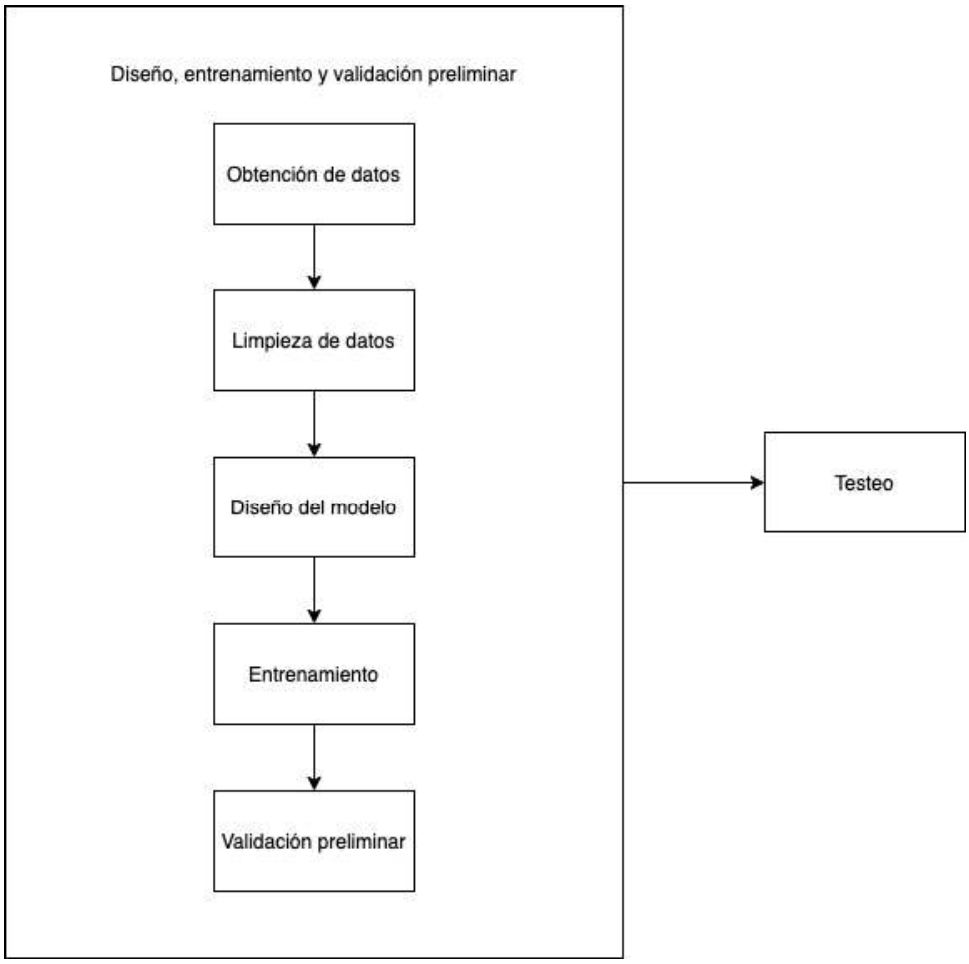

Figura 1.1. Fases del aprendizaje de máquina

Las etapas descritas en la Figura 1.1 comprenden dos fases principales con subtareas, las cuales serán descritas a continuación:

1. Diseño, entrenamiento y validación preliminar: en esta fase se procederá a elaborar la arquitectura para la extracción de características automatizada, y elaborar los modelos de clasificación para evaluar dichas características. Las subtareas de esta fase son las siguientes:
    a. Obtención de datos: Se refiere a la recolección de datos, la cual puede ser manual, o puede darse mediante la obtención de bases de datos previamente armadas, como es el caso del presente trabajo. Se usarán principalmente cuatro bases de datos de rostros:

i. SCFace [10].
ii. 10k US Adult Faces Database [11].
iii. The Extended Yale Face Database B [12].
iv. MIT-CBCL face recognition database [13].

Éstas bases de datos proveyeron de un considerable ´porcentaje de diversidad en el dataset de varios tipos: de iluminación y enfoque, ya que varias de éstas imágenes, como en el caso de la base de datos de Yale, fueron tomadas en ambientes controlados, mientras que las imágenes de bases de datos como la SCFace fueron tomadas en ambientes libres.

b. Limpieza de datos: En esta fase las imágenes recolectadas pasarán por un proceso de recorte y enfoque de las mismas, para eliminar el ruido que otros elementos en las imágenes puedan generar. También se pasarán por un proceso de cambio de tamaño, en el cual las imágenes de estas diferentes bases de datos serán puestas todas en un tamaño de 224x224 pixeles, ya que ésta es la configuración de los datos que la red neuronal usada en el presente trabajo necesita.
c. Diseño: En esta tarea se definirá la arquitectura de la extracción de características, y los algoritmos correspondientes que evaluarán el proceso de extracción
d. Entrenamiento: En esta tarea se implementarán los algoritmos propuestos, y se entrenarán los mismos con las bases de datos antes mencionadas.
e. Validación preliminar: en esta tarea podremos evaluar el algoritmo con algunos datos separados del mismo conjunto de datos usados para el entrenamiento, pero que no se usaron para el mismo.

2. Testeo: En esta fase se evaluará los algoritmos entrenados y su nivel de precisión. Una vez obtenidos estos resultados, se procederá a realizar la respectiva comparativa entre la precisión obtenida de los mismos, y con estos resultados podremos finalmente decidir cuál de las dos arquitecturas de extracción de características se comporta mejor.

Organización del presente documento

A continuación de la introducción se organiza el documento con la siguiente información: pregunta de investigación, el objetivo general, los objetivos específicos, el alcance y el marco teórico. Posteriormente, en el capítulo 2 se presenta la metodología, en donde se explica lo realizado en cada una de las fases de la metodología tradicional del aprendizaje de máquina. En el capítulo 3 se muestran los resultados y discusión. Por último, en el capítulo 4 se presentan las conclusiones del presente proyecto de investigación.

1.1 Pregunta de investigación

Al inicio del presente trabajo se realizó una investigación preliminar acerca de los métodos de aprendizaje no supervisados, de entre los cuales, en varios trabajos, se mencionaban diferentes aplicaciones del algoritmo KMeans [6]. En estos trabajos se proponía una comparativa entre varios de estos algoritmos de agrupamiento, sin embargo, la extracción de características de las imágenes se hacía de manera manual [14,15].

Otros trabajos usaban el algoritmo KMeans como parte de la fase de extracción de características, para obtener mejores resultados en su clasificación. Sin embargo, este proceso es intensivo computacionalmente y en memoria. Sería muy útil para futuros trabajos saber si este procesamiento de las características contribuye a mejorar el resultado de la clasificación posterior.

Por tanto, para el presente proyecto, se define la siguiente pregunta de investigación:

> ¿Permitirá la integración de K-Means con una red neuronal convolucional para la extracción de características de una imagen mejorar el resultado de la clasificación posterior?

1.2 Objetivo General

Comparar en términos de exactitud al momento de realizar la clasificación, usando como referencia porcentaje de precisión, un método de extracción de características que incluye un algoritmo de clusterización, en este caso KMeans, frente a otro que no para el problema de clasificación de rostros .

1.3 Objetivos Específicos

a) Obtener un valor de rendimiento con la primera arquitectura propuesta usando una red neuronal convolucional para obtener las características de las imágenes de rostros.
b) Obtener el valor del rendimiento añadiendo una etapa de clusterización a la red neuronal convolucional para la extracción de las características antes mecionadas.
c) Concluir cuál de los dos métodos de extracción de características se comporta mejor y en qué condiciones, tomando en cuenta la precisión resultante de los clasificadores.

1.4 Alcance

El presente proyecto tiene la finalidad de responder la pregunta formulada en la hipótesis, si la integración de un algoritmo de clusterización con una extracción de características automatizada puede mejorar los resultados de la clasificación posterior, de manera que pueda servir como referencia para futuros trabajos que se hagan en el mismo campo. De la misma manera se presenta una forma de extracción de características automatizada, como es la red neuronal convolucional habiendo quitado la última capa, la de clasificación. El desarrollo del trabajo comprende la elaboración de un proceso de extracción de características y clasificación automatizado en su mayoría, ya que la fase de limpieza se hará de una manera manual, a gran escala, las arquitecturas definidas son las que se observan en las Figuras 1.2 y 1.3

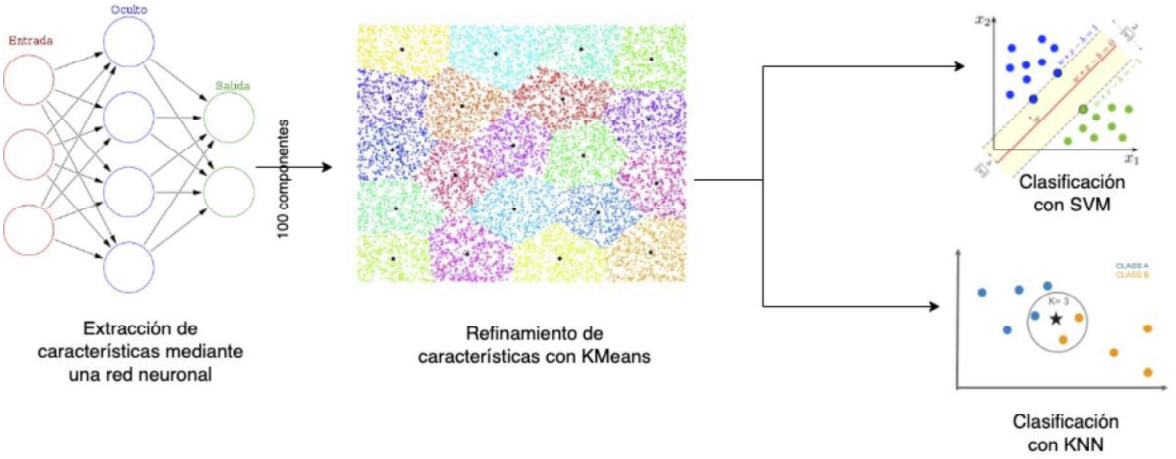

Figura 1.2. Primera arquitectura propuesta de extracción de características

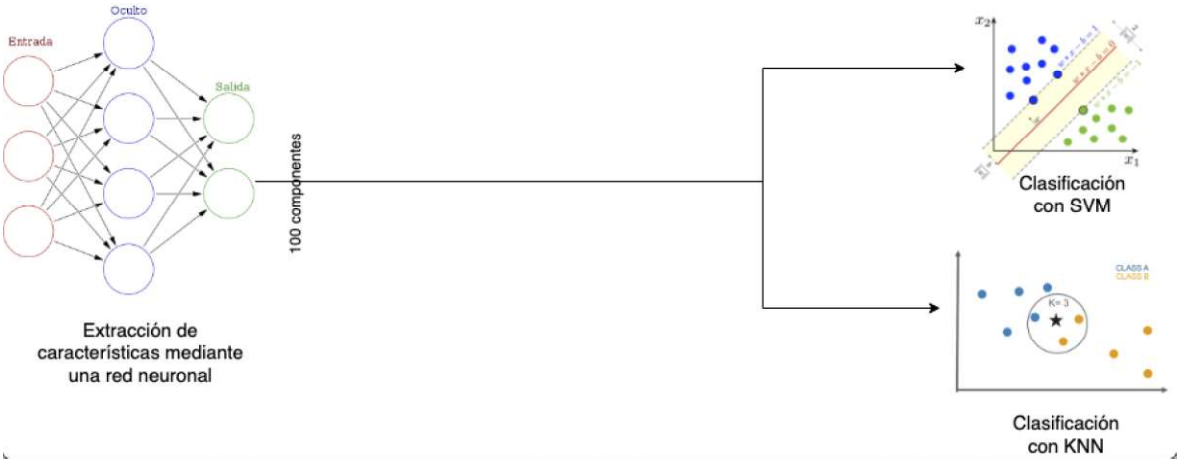

Figura 1.3. Segunda arquitectura propuesta de extracción de características.

En la Figura 1.2 se detalla lo siguiente:

- El proceso de extracción de características será mediante una red neuronal convolucional sin la capa de clasificación.
- Se evaluará dichas carcaterísticas con el algoritmo de clusterización KMeans, para refinar las mismas.
- Por último, se evaluará dichas características con dos algoritmos de clasificación, KNN y SVM con un kernel lineal.

En la Figura 1.3, podremos observar que la segunda arquitectura propuesta es similar a la primera, sin embrago, carece del paso de refinamiento de las características con el algoritmo KMeans. Con esta comparativa se pretende hallar respuesta a la pregunta de investigación.

1.5 Marco Teórico

El aprendizaje de máquina permite a los programas “aprender” a través de la experiencia. El aprendizaje de máquina implica la construcción de algoritmos que se adapten al problema plantado para mejorar su habilidad para hacer predicciones. [1]

El uso de computadoras para reconocimiento de patrones y objetos dentro de una imagen es mucho menos práctico sin el uso de técnicas de aprendizaje de máquina. Los algoritmos de reconocimiento de imágenes, también llamados clasificadores de imágenes, pueden ser entrenados para clasificar imágenes basadas en su contenido. Éstos algoritmos son entrenados mediante el procesamiento de varias imágenes de ejemplo que han sido clasificadas previamente. Usando las semejanzas y diferencias de dichas imágenes, estos programas mejoran actualizándose cada vez que una nueva imagen es procesada. Para esta forma de aprendizaje de máquina que involucra el procesamiento de imágenes, generalmente se usa una red neuronal artificial de tipo convolucional. [1]

Redes neuronales convolucionales

Las redes neuronales convolucionales analizan los elementos visuales de una imagen e intentan mapear las ocurrencias de dichas características [16]. Éstas redes procesan las imágenes como volúmenes, recibiendo una imagen de colores como una matriz rectangular donde la altura y el ancho son el número de pixeles conrrespondientes a las medidas de la imagen original. Dichas matrices tienen una profundidad de 3 elementos, cada una correspondiente a los canales de color, RGB. Estas capas se conocen también como canales. En cada pixel de la imagen, la intensidad de color R, G o B se representa con un número, y este número es parte de 3 matrices de dos dimensiones que se alimenta como datos iniciales a la red neuronal convolucional. La red empieza a filtrar la imagen mediante la selección de un subconjunto de cuadros de pixeles juntos y buscando patrones, realizando lo que se conoce como una convolución [16].

Una convolución, o filtrado digital, es una de las operaciones más comunes usadas en el procesamiento de imágenes en la actualidad. Una convolución puede expresarse como un producto de polinomios, usado frecuentemente en técnicas de filtrado de bloque [17]. Para un mejor entendimiento del funcionamiento de la convolución, se puede observar la Figura 1.4.

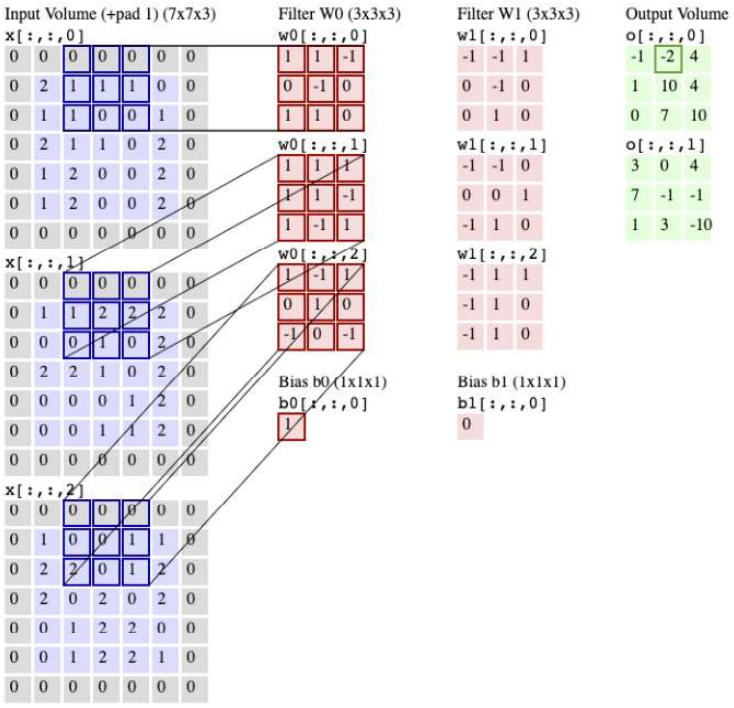

Figura 1.4. Proceso de convolución en una CNN [16]

Las redes neuronales convolucionales están compuestas de capas de neuronas que aplican diferentes funciones a los datos de entrada, las más comunes son las siguientes, aparte de la convolución:

- Pooling: El pooling, hablando de redes neuronales, consiste en "acumular" características de mapas generados por la convolución de un filtro sobre una imagen. Formalmente, esta función reduce progresivamente el tamaño espacial de la matriz resultante de la convolución para reducir el número de parámetros y y el costo computacional de la operación en la red neuronal. La forma más común de pooling es el "max pooling", el cual se hace en parte para reducir el overfitting proveyendo una forma abstraída de la representación de la imagen. De esta forma, también reduce el costo computacional reduciendo el número de parámetros que serán alimentados a las siguientes neuronas. El pooling se realiza aplicando un filtro como funciones máximo, promedio o general, a submatrices que no se sobreponen de la matriz inicial [18]. Para visualizar cómo actúa esta función sobre una matriz, se puede observar la Figura 1.5.

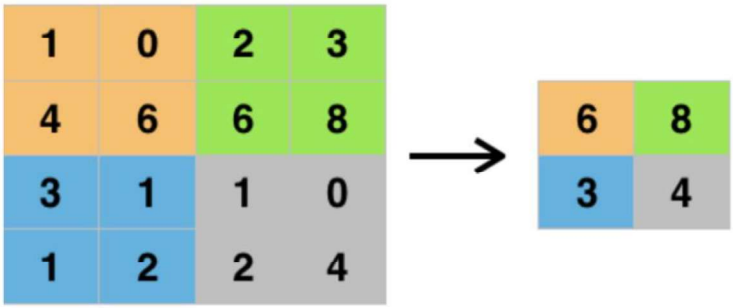

Figura 1.5. Funcionamiento del max pooling. [18]

- Dropout: Es una técnica que previene el overfitting y provee una manera de de combinar varias arquitecturas de redes neuronales. El término “dropout” se refiere a desprenderse temporalmente de unidades, en una red, junto con todas sus conexiones de entrada y de salida. La decisón de cuál unidad desprender se toma randómicamente [19].
- Softmax: En muchos casos, es la última capa de la red neuronal, la encargada de la clasificación. Softmax asigna probabilidades decimales a cada una de las clases en un problema de varias clases. Todas estas probabilidades decimales sumadas deben dar como resultado 1.0. Esta limitación adicional ayuda a que, en el entrenamiento, las rutas óptimas converjan más rápido de lo que lo harían de otra manera. Sin embargo, el algoritmo se vuelve impreciso cuando un elemento puede pertenecer a múltiples clases, en cuyo caso es recomendable usar regresiones lineares [20]. La operación usada para calcular la probabilidad de pertenencia a una clase, y luego seleccionando la clase de mayor probabilidad se asigna una etiqueta al elemento. La operación usada en una neurona softmax es la que se especifica en la Ecuación 1-1.

$$\frac{e^{(w_y^T x + b_j)}}{\sum_{\kappa \in \mathrm{K}} e^{(w_k^T x + b_k)}}$$

Ecuación 1-1

Transferencia de aprendizaje

La transferencia de aprendizaje es una técnica de aprendizaje de máquina supervisado, que reusa partes de un modelo previamente entrenado en una nueva red que va a tener una tarea diferente, pero con un planteamiento de problema similar. Las técnicas que habilitan transferencia de conocimiento entre modelos representan un progreso hacia hacer el aprendizaje de máquina más efectivo [21].

En ciertos trabajos [22, 23], se ha observado que el remover la capa softmax del modelo y usar la red neuronal para extraer características de las imágenes da como resultados características que son una buena representación de la imagen original.

Arquitectura Inception-V3

Varias arquitecturas de redes neuronales convolucionales para extracción de características de imágenes han sido propuestas [24, 25], pero una que destaca frente a trabajos anteriores es la GoogLeNet [25], desarrollada por Google, en el 2014. Esta arquitectura de red neuronal convolucional profunda tiene como característica principal que mejora la utilización de los recursos computacionales dentro de la red neuronal [25]. Para optimizar la calidad, las decisiones de arquitectura fueron tomadas en base al principio Hebbiano, el cual es un método para determinar la forma de modificar los pesos entre neuronas, y la intuición del procesamiento multiescala [25].

Google no paró con esta red neuronal, y la siguió mejorando y desarrollando hasta su tercera versión, la cual dio a conocer como "Inception-V3". Esta última versión de la GoogLeNet se diferencia de otras arquitecturas como VGG16 [24] y AlexNet [26] en que, si bien estas últimas son mucho más simples en su arquitectura, vienen con la desventaja que evaluar la red tienen un alto costo computacional [2], y la calidad de las características extraídas por la arquitectura Inception-V3 suelen ser de buena calidad dependiendo del problema aplicado[3, 5]. En las Figuras 1.6, 1.7, 1.8 y 1.9 podremos observar la arquitectura de la GoogLeNet en su versión Inception-V3, la cual si bien es compleja, como se mencionó con antelación, provee de un procesamiento adecuado para imágenes bajo ambientes controlados. La Figura 1.6 muestra en una tabla la arquitectura general de la red neuronal convolucional Inception-V3, y las Figuras 1.7, 1.8 y 1.9 muestran diferentes configuraciones de las capas de "incepción", como se llama a ciertas capas de extracción de características en la arquitectura Inception-V3, propuestas en el mismo trabajo, que consisten de capas de convolución de diferentes tipos, conectadas en el orden que se muestra en las Figuras 1.7, 1.8 y 1.9 y una capa de pooling al final.

type	patch size/stride or remarks	input size
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 1.7	35×35×288
5×Inception	As in figure 1.8	17×17×768
2×Inception	As in figure 1.9	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figura 1.6. Arquitectura de la red Inception-V3 [2]

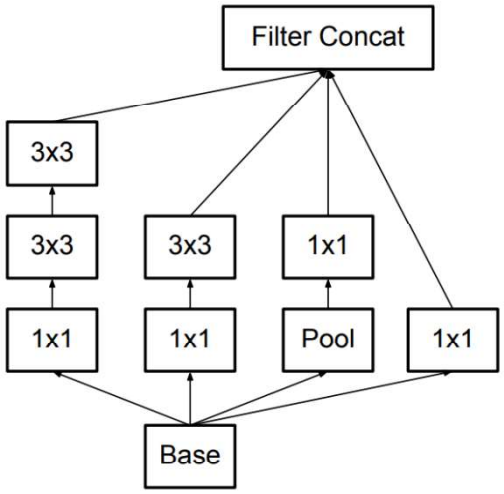

Figura 1.7. Primera arquitectura Inception mencionada en la Figura 1.6 [2]

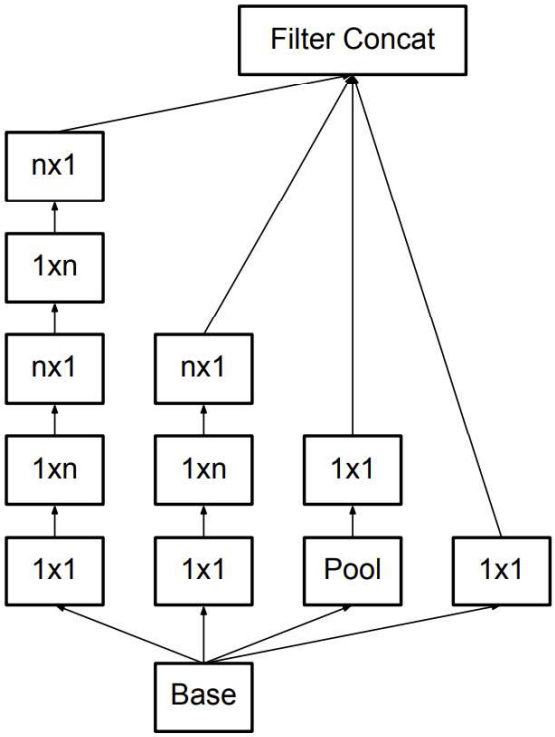

Figura 1.8. Segunda arquitectura Inception mencionada en la Figura 1.6 [2]

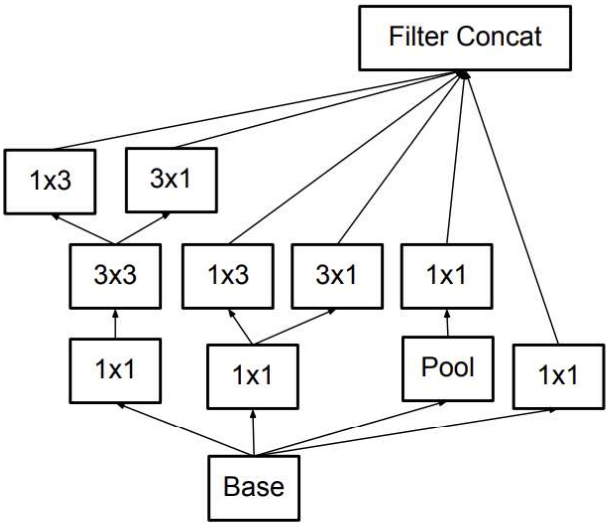

Figura 1.9. Tercera arquitectura Inception mencionada en la Figura 1.7 [2]

En el presente trabajo, como se mostró en las Figuras 1.2 y 1.3, no sólo se usará redes neuronales convolucionales como parte del modelo diseñado, sino que también se usará otros algoritmos de aprendizaje supervisado para evaluar la calidad de las características obtenidas de todo el modelo: kNN y SVM.

Algoritmo kNN

El algoritmo kNN es simple que guarda registro de todos los casos, y clasifica los nuevos casos basado en una función de distancia. El algoritmo kNN se usaba en estimación estadística y reconocimiento de patrones ya en la década de 1970 como una técnica no paramétrica [6]. Este algoritmo se usa para clasificar y para regresión linear. El principio del algoritmo es que, si la mayoría de los vecinos más similares a un punto en un espacio vectorial pertenecen a cierta categoría, entonces el veredicto es que este nuevo punto pertenece a esta categoría. La similaridad puede ser medida por la distancia de los puntos en el espacio vectorial. Para aplicar este algoritmo, se necesita primero un conjunto de datos de entrenamiento correctamente etiquetados. Después, para el nuevo punto cuya etiqueta se desconoce y que se representa como un vector en el espacio vectorial, se calculan las distancias entre este nuevo punto y todos los puntos usados en el conjunto de entrenamiento. Después de ordenar los resultados de este cálculo de distancias, la decisión de la etiqueta de la clase se toma de acuerdo al número k de vecinos, o puntos más cercanos en el conjunto de datos de entrenamiento. La distancia entre dos puntos en un espacio vectorial multidimensional se puede calcular de varias maneras. El método más usado y el que se usará en el presente trabajo, es la distancia euclidiana, la cual se calcula con la Ecuación 1-2 [27].

$$dist(p,q) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$$

Ecuación 1-2

La calidad del conjunto de datos de entrenamiento afecta directamente a los resultados de la clasificación. Al mismo tiempo, la elección del parámetro k es muy importante, ya que la elección de diferentes k puede resultar en la elección de diferentes etiquetas. Sin embargo, este es un algoritmo de un cálculo simple y se puede aplicar a conjuntos de datos de alta dimensionalidad, aunque si los conjuntos de datos de entrenamiento, validación y testeo son de gran tamaño, la complejidad computacional será alta y el tiempo de operación será largo [27].

Algoritmo SVM

El algoritmo SVM (por sus siglas en inglés Support Vector Machine) es un algoritmo que usa un hiperplano para analizar datos nuevos, no etiquetados, una vez que el algoritmo ha sido entrenado [28]. Para los algoritmos de Support Vector Machine, cada dato es un vector de p dimensiones, y el algoritmo intenta crear un clasificador lineal ajustando el dato dentro de un hiperplano de p-1 dimensiones. Suponiendo que estamos ajustando un modelo con dos clases, de acuerdo a lo propuesto por el algoritmo SVM debemos encontrar los puntos que se acercan más a dichas clases. Estos puntos se conocen como vectores de soporte. La distancia entre los vectores de soporte y la línea que separa las clases se conoce como margen. La meta del algoritmo SVM es la de maximizar este margen, ya que cuando éste margen alcance un valor máximo, el hiperplano clasificador se vuelve el óptimo. Support Vector Machine es un clasificador linear en principio, pero se extiende fácilmente a problemas no lineales mapeando los vectores de los datos de entrada en un espacio vectorial de dimensiones potencialmente infinitas. El escoger una adecuada función de mapeo permite convertir a los datos en linealmente separables por categorías por un hiperplano. Estas funciones de mapeo se conocen también como funciones kernel [28]. La función del kernel se usa para tomar los datos de entrada y transformarlos a la forma requerida. Diferentes algoritmos SVM usan distintos tipos de kernel. Las funciones del kernel pueden ser de varios tipos, siendo los más conocidos:

- Kernel lineal: Se usa cuando se manejan pocos ejemplos de entrenamiento comparado con la dimensión de cada vector. Los datos de entrada se mapean con la Ecuación 1-3

$$k(x_i, x_j) = x_i^T x_j + c$$

Ecuación 1-3

- Kernel polinomial: Sus datos de entrada se mapean con la Ecuación1-4:

$$k(x_i, x_j) = (x_i \cdot x_j + 1)^d$$

Ecuación 1-4

- Kernel RBF (Función de base radial): Sus datos de entrada se mapean con la Ecuación 1-5 [29]:

$$k(x_i, x_j) = exp(-\frac{||x_i - x_j||^2}{2\sigma^2})$$

Ecuación 1-5

Estos dos algoritmos explicados evaluarán las características extraídas por el modelo propuesto, y con los respectivos resultados de precisión se determinará cuál modelo tiene mejor precisión para el escenario propuesto.

Aprendizaje no supervisado

El aprendizaje no supervisado es una técnica de aprendizaje de máquina que encuentra y analiza patrones escondidos en datos crudos o no etiquetados. Ignorando las etiquetas, un modelo que usa una técnica de aprendizaje no supervisado puede inferir relaciones complejas y sutiles entre datos no clasificados que otras técnicas pueden pasar por alto. Y lo hacen sin el costo computacional y de tiempo que consumen los algoritmos de aprendizaje supervisado [30].

Algoritmo KMeans

En el presente trabajo se usará el algoritmo de aprendizaje no supervisado KMeans, con el propósito de refinar las características. Este algoritmo agrupa los datos de acuerdo a sus características en contraposición a los algoritmos supervisados, que lo hacen de acuerdo a categorías predefinidas. El número de grupos en los cuales los datos serán divididos es determinado por la persona que implementa el algoritmo, y el resultado de estas iteraciones serán los centroides de los grupos [31]. El algoritmo consiste en empezar con k grupos cada unos de los cuales consiste de un único punto randómico, después ir continuamente añadiendo cada nuevo punto al grupo cuyo centro o promedio sea el más cercano. Después de que un punto es añadido a un grupo, el centro o promedio del grupo es reajustado de manera que se tome en cuenta el nuevo punto añadido. De esta manera, en cada etapa el algoritmo k-Means tiene k centros o promedios de k grupos a los cuales representan. Entonces, dado un conjunto de datos en el cual cada punto es un vector de d dimensiones, el algoritmo KMeans lo que intenta es particionar las n observaciones en k grupos, dado que k < n, de manera que podamos minimizar la suma de distancias cuadráticas dentro de cada clúster, también conocida como varianza. El objetivo se puede expresar mediante la Ecuación 1-6, donde x es el punto y μ es el centroide del clúster [32]:

$$min \sum_{i=1}^{k} \sum_{x \epsilon S_i} ||x - \mu_i||^2 = min \sum_{i=1}^{k} |S_i| \, Var(S_i)$$

Ecuación 1-6

Algoritmo Elbow

El algoritmo de elbow es un método que ayuda a los científicos de datos a seleccionar número óptimo de clústers para el algoritmo KMeans, ajustando el modelo con un rango de valores para K, donde K es el número de clústers. Se realiza un gráfico comparando la suma de distancias entre cada punto y cada centro de clúster contra el número de clusters. Si la línea en el gráfico resultante luce como un brazo, entonces el codo ("elbow" en inglés, de ahí su nombre) o el punto de inflexión de la curva es un buen indicativo de que el algoritmo y sus datos se ajustarán mejor con ese número de clústers, es decir, se debe seleccionar el número de clústers en el cual el descenso del error se estabiliza. El error para el método de elbow está calculado mediante la suma del error cuadrático, que se puede expresar matemáticamente mediante la Ecuación 1-7 [33]:

$$Costo = \sum_{i=1}^{k} \sum_{x_i \in S_k} ||X_i - C_k||_2^2$$

Ecuación 1-7

Metodología tradicional de aprendizaje de máquina.

El procedimiento utilizado para la obtención, procesamiento y validación de datos descrito en la sección 2, ya que es el proceso usado en varios trabajos de aprendizaje de máquina y ha dado buenos resultados [9, 14, 15]. En la primera fase se diseña el modelo de aprendizaje, se entrena con los datos provistos, y se procede a una validación con un pequeño set de datos que no pertenecieron a los datos de entrenamiento. En la etapa de validación preliminar podremos darnos cuenta de si nuestro modelo está clasificando de manera correcta, o si por el contrario padece del problema conocido como "overfitting".

En la segunda fase se procede a testear el modelo con datos diferentes a los de entrenamiento y a los de validación, para comprobar cómo se comporta el modelo con datos a los que no ha estado expuesto anteriormente. Es decir, como se va a comportar en un ambiente más cercano a la realidad.

Algoritmo PCA

Matemáticamente, el algoritmo PCA se define como una transformación lineal ortogonal, que transforma los datos que recibe a un nuevo sistema de coordenadas de manera que la mayor varianza de una proyección escalar recaiga sobre el primer elemento principal, la segunda mayor varianza recaiga sobre el segundo

componente principal, y así sucesivamente. Este algoritmo se conoce como de análisis de componentes principales PCA (por sus siglas en inglés principal component analysis), el cual reduce la dimensionalidad de los datos mientras retiene la mayor parte de la variación del conjunto de datos. Logra esto mediante la identifiación de direcciones en los vectores, llamados componentes principales, al mismo tiempo que busca que la variación de los datos sea máxima. Usando unos pocos componentes, se puede lograr representar cada dato por unos pocos números en lugar de largos vectores cargados de datos y variables. Otro beneficio de aplicar este algoritmo a los datos es que, reduciendo sus dimensiones, se puede graficarlos [34]. Debido a todos estos beneficios antes mencionados, y debido a que el procesamiento de vectores de características extensos se vuelve intensivo en procesamiento, este algoritmo se usará en el presente trabajo. Su utilidad es la de reducir las dimensiones de los vectores de características obtenidos de la red neuronal convolucional.

Bases de datos:

Como se ha mencionado anteriormente, las bases de datos a usar en el presente trabajo son tres:

a. SCFace [10]: Es una base de datos de imágenes estáticas de rostros humanos. Las imágenes fueron tomadas en un ambiente no controlado usando cinco cámaras de video de vigilancia de varias calidades. La base de datos contiene 4160 imágenes estáticas (en espectro visible e infrarrojo) de 130 personas. Las imágenes de cámaras de diferente calidad imitan las condiciones de la vida real y permiten un reconocimiento facial robusto. Algunos puntos que hacen interesantes a esta base de datos son los siguientes:
    - **a.** Para tomar las imágenes se usaron cámaras de diferente calidad y resolución.
    - **b.** Las imágenes fueron tomadas bajo condiciones de iluminación no controladas
    - **c.** Las imágenes fueron tomadas desde varias distancias.
    - **d.** En otras bases de datos es común hacer posar a los sujetos y tomar imágenes arriba de su cuello, lo que hace del entrenamiento del algoritmo de clasificación excesivamente controlado. Durante la toma

de varias de las imágenes para la base de datos, los sujetos no estaban mirando a un punto en específico.

   e. La base de datos contiene imágenes de 130 sujetos, lo cual reduce la probabilidad de reconocimiento por casualidad a menos de 0.7%
   f. Es una base datos apta para los problemas de identificación y verificación.
   g. Es una base de datos que se obtiene libre de costo.
   h. Siendo una base de datos de Croacia, se agrega un poco de diversidad al conjunto de datos de entrenamiento, los cuales suelen provenir de sujetos de los Estados Unidos. [35]

b. 10k US Adult Faces Database [11]: Esta base de datos contiene 10168 fotografías naturales de rostros de personas residentes de Estados Unidos. Las imágenes contenidas en esta base de datos son de formato JPEG de una resolución de 72x256 pixeles. Esta base de datos también es de distribución gratuita previa solicitud, por ello, conveniente para el presente trabajo. Puede considerarse que, debido a que las fotos son todas de personas de los Estados Unidos, carecen de diversidad, sin embargo se tomó en consideración que las fotos de esta base de datos son de adultos que residen en los Estados Unidos, no necesariamente de ascendencia ni originarios del país. También considerando que el país recibe gran cantidad de inmigrantes todos los años, revisando las fotos se encontraron personas de ascendencia asiática y latina, de manera que la diversidad en la base de datos no es una preocupación mayor. [36]
c. The MIT-CBCL Face Recognition Database [13]: Proveen imágenes de 10 sujetos en diferentes posiciones, e imágenes generadas automáticamente a partir de éstas, desde modelos 3D de los sujetos. Se proveen dos datasets de entrenamiento: uno con imágenes de alta resolución, incluyendo vistas frontales, de perfil y medio perfil, e imágenes sintéticas (324 por sujeto) renderizadas desde un modelo 3D de cada uno de los sujetos. Los modelos 3D aquí descritos no se incluyen en el dataset disponible al público. La base de datos está disponible gratis para investigadores. [13]. El dataset usado en el presente trabajo contiene en 2429 imágenes de los 10 sujetos con distintas variaciones.

d. The Extended Yale Face Database B [12]: Esta base de datos contiene 16128 imágenes de 28 personas en 9 poses diferentes y 64 condiciones de iluminación diferentes. A diferencia de la base de datos de Yale B original, con solo 10 personas, las imágenes fueron alineadas manualmente, recortadas y cambiados su tamaño a un formato de 168x192 pixeles. Estos 28 sujetos son capturados con diferentes conceptos de iluminación y variación en tanto en la pose como el ambiente. El atractivo de esta base de datos para el presente trabajo es que, de manera similar a las anteriors bases de datos mencionadas, es de acceso gratuito para investigadores. Además, aparte de las imágenes obtenidas, fueron generadas automáticamente más imágenes a partir de las originales con variaciones de iluminación del fondo y rotadas. Otro de los usos comunes de esta base de datos es para reconocer emociones en los rostros, debido a que los rostros fueron tomados con diferentes expresiones como: triste, adormilado, sorprendido y guiñando un ojo [37].

2. METODOLOGÍA

El procedimiento utilizado para la obtención, análisis y limpieza de los datos, entrenamiento y validación de los mismos se enmarca en las fases de la metodología tradicional del aprendizaje de máquina, tal como se explica en el apartado 1.5 del presente documento.

Las fases de la metodología incluyen: Diseño, entrenamiento y validación preliminar y Testeo. La fase de diseño, entrenamiento y validación preliminar contiene varias subtareas que son las mencionadas anteriormente: obtención de datos, limpieza de datos, diseño del modelo, entrenamiento y validación preliminar, y por último testeo. Estas tareas serán explicadas a mayor detalle a continuación.

2.1. Diseño, entrenamiento y validación preliminar: obtención de datos

La primera tarea de la primera fase del método tradicional de aprendizaje de máquina comprende de la obtención de los datos con los cuales se va a trabajar. Se realizó una investigación preliminar, en la cual se evaluaron varias bases de datos de rostros de diferentes universidades alrededor del mundo, y dado que el

presente experimento requería gran cantidad de imágenes [37], se eligieron las bases de datos antes mencionadas.

SCFace

El acceso a la base de datos SCFace es, de hecho, el más complicado de los tres. Se debe cumplir un cierto número de requerimientos:

- Una carta de presentación solicitando formalmente la base de datos. La carta debe ser escrita en una hoja que contenga las insignias de la institución para la cual se está realizando el trabajo de investigación, y debe incluir una dirección y número telefónico legibles. Se recomienda incluir el correo del solicitante.
- Un acuerdo de transacción correctamente completado, en el cual se puede leer los términos de uso de la base de datos, que son similares a otras bases de datos de este tipo:
    - No redistribuir, publicar o copiar de ninguna manera ninguno de los elementos de la base de datos, ya sea para beneficio propio o no.
    - Por razones legales, si se quiere poner unos ejemplos de las imágenes que contiene la base de datos, únicamente se pueden usar imágenes autorizadas por el autor del artículo, en este caso las imágenes de los sujetos con ID 001, 002, 045 o 102.
    - Las imágenes serán usadas únicamente con propósitos de investigación o académicos. Su uso para cualquier fin comercial está estrictamente prohibido.
    - Todos los documentos o artículos que reporten resultados usando esta base de datos deben reconocer el crédito al artículo original.
    - En caso de que algún trabajo sea publicado, debe enviársele inmediatamente una copia al investigador principal del artículo [10].
- El acuerdo descrito anteriormente no puede ser firmado por un estudiante, debe, por obligación, ser firmado por un empleado regular de la institución [10], en este caso, el tutor.
- Estos documentos deberán ser escaneados y enviados al investigador principal del artículo, el Dr. Mislav Grgic [10].
- En el ANEXO I, podrán encontrar el acuerdo de transacción firmado por el tutor, que posteriormente fue enviado al investigador principal del artículo [10].

- Una vez completado exitosamente este proceso, uno de los asistentes de investigación del laboratorio que generó el artículo proveyó un link de descarga válido por 24 horas, que por razones de confidencialidad no se puede compartir.

Figura 2.1. Algunas de las imágenes cuya divulgación de ejemplo está permitida en la base de datos SCFace

10k US Adult Faces Database

El acceso a la base de datos 10k US Adult Faces Database, si bien similar al anterior, no requiere de un proceso tan complejo ni largo de realizar, ni tampoco documentación específica como en el anterior caso. Lo que se debe hacer es enviar un correo a la Dra. Wilma Bainbridge, investigadora principal del artículo que originó la base de datos, con ciertos datos, como nombre completo del investigador, correo, institución para la que se realiza el trabajo, dirección de la institución, otras personas involucradas en la investigación, en este caso el tutor, y la razón por la que se desea acceder a la base de datos. En la Figura 2.2 podremos ver el correo enviado.

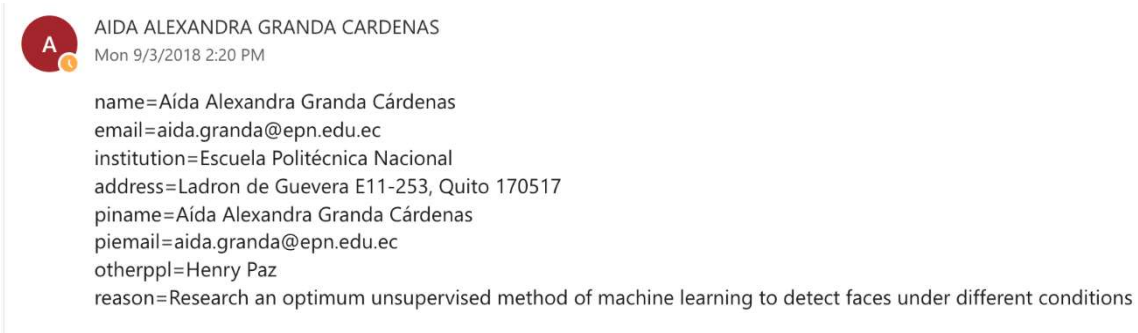

Figura 2.2. Correo enviado a la Dra. Wilma Bainbridge para la obtención de la base de datos 10k US Adult Faces Database

Como se mencionó anteriormente, la obtención de esta base de datos era sumamente importante para el desarrollo del presente trabajo, ya que representa un volumen de datos considerable, además que las imágenes están enfocadas ya en el rostro de las personas, y es un proyecto al cual se le da mantenimiento periódicamente [11]. La aprobación del uso de la base de datos se dio casi automáticamente, y se tuvo que acceder a ella a través de un link exclusivo, junto con una contraseá personalizada. Estos datos no se pueden mencionar en el presente trabajo por motivos de confidencialidad.

The Extended Yale Face Database B

Esta base de datos es aún más fácil de conseguir que las anteriores. Si bien no es una base de datos que contenga fotos de varios sujetos distintos, las imágenes si han sufrido variaciones en posición, emoción del sujeto, iluminación directa y del fondo. La base de datos se encuentra disponible en el sitio web de visión por computadora de la universidad de California San Diego[1], la cual ofrece dos opciones: la base de datos con las imágenes originales, o la base de datos con las imágenes recortadas. Para el presente trabajo, se tomaron las imágenes recortadas, ya que después pasarían por un proceso de recorte de todos modos como parte del proceso de limpieza.

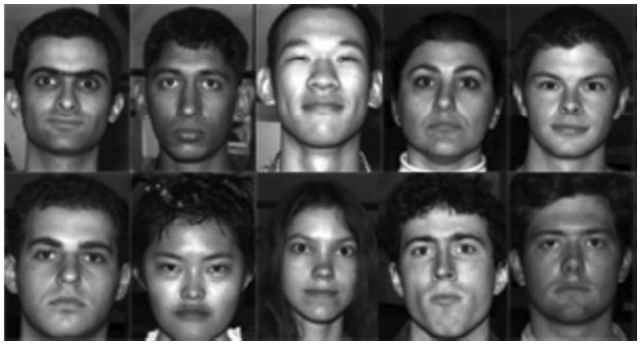

Figura 2.3. Algunas imágenes de la base de datos Extended Yale Face Database B

The MIT-CBCL Face Recognition Database

El acceso a esta base de datos es, sin duda, el más fácil de todas las bases de datos antes mencionadas, debido a que es pública y sólo se necesita aprobar un acuerdo de transacción, como en la base de datos SCFace, pero a diferencia de ésta se acepta el acuerdo únicamente con un click en un link, que nos lleva

[1] http://vision.ucsd.edu/~leekc/ExtYaleDatabase/download.html

directamente a otra página donde podemos descargar la base de datos[2]. Esta es una base de datos elaborada por el MIT (por sus siglas en inglés Masachussets Institute of Technology), y sus acuerdos para usar las imágenes de la base de datos son muy parecidos a los acuerdos propuestos para obtener la base de datos SCFace [38].

2.2. Diseño, entrenamiento y validación preliminar: limpieza de datos

Esta tarea es parte de la primera fase de la metodología tradicional de aprendizaje de máquina. En esta tarea, limpiamos los datos de todo el ruido posible. Se realizó un enfoque y recorte manual de las imágenes mediante una red neuronal pre-entrenada para detección de rostros, esto para eliminar todo el fondo de las imágenes que pudiera causar algún tipo de ruido. Al mismo tiempo que las imágenes son recortadas, son redimensionadas para que todas tengan unas características lo más estándar posible antes de entrar a la siguiente tarea de la fase. La comparación entre las imágenes de entrada y las imágenes después del proceso de limpieza de datos se puede observar en la Figura 2.4:

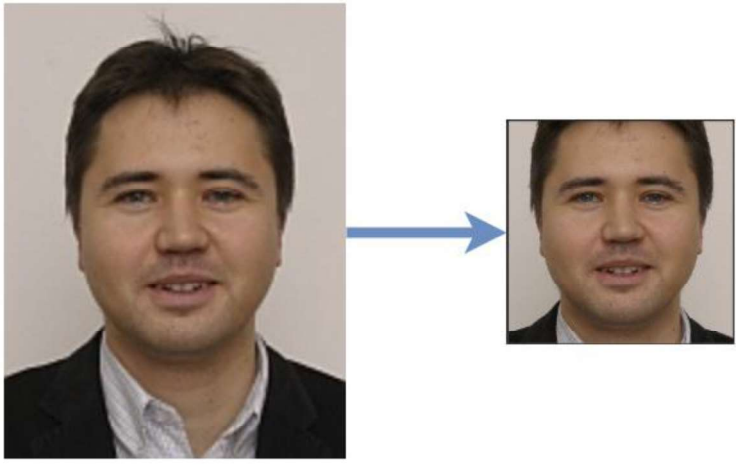

Figura 2.4. Comparación de imágenes antes y después de la limpieza

2.3. Diseño, entrenamiento y validación preliminar: diseño del modelo

Esta tarea sigue siendo parte de la primera fase de la metodología tradicional de aprendizaje de máquina. Para esta fase también se realizó una investigación

[2] http://cbcl.mit.edu/software-datasets/heisele/download/download.html

preliminar, para entender cómo estaban siendo usados los algoritmos de aprendizaje no supervisado, más específicamente KMeans, como parte de la extracción y refinamiento de características. En ciertos trabajos se pudo apreciar que, para el problema de detección facial, se usaba el algoritmo KMeans como refinador de características después de la extracción de las mismas [39, 40, 41]. Todos estos trabajos tienen un planteamiento de problema similar, y varios pasos intermedios entre la extracción de características y la clasificación, y debido a ello surge la pregunta: ¿Es realmente valioso o necesario añadir el algoritmo KMeans a estos modelos planteados?

A manera de réplica de alto nivel de estos trabajos, se diseñó dos arquitecturas que incluyan una extracción de características en gran parte automatizada. Estas son las arquietcturas graficadas y explicadas en el apartado 1.4, graficadas en las Figuras 1.2 y 1.3. Debido al estudio de otros trabajos [42, 43, 44], se infiere que un buen método de extracción de características automatizado son las redes neuronales convolucionales. Debido a lo mencionado en el artículo de la red neuronal GoogLeNet en su versión InceptionV3 y su rendimiento superior a comparación de otras redes conocidas por su buen rendimiento a la hora de extraer características [2], en el presente trabajo se decidió usar esta arquitectura, la cual ya se explicó en el apartado 1.5. Debido a que el enfoque del presente trabajo no son las redes neuronales en sí, se usará una técnica de transferencia de aprendizaje, explicada en el apartado 1.5, al igual que los beneficios de su uso. Los pesos usados para la técnica de transfer learning del presente trabajo serán los obtenidos por el entrenamiento de la red Inception-V3 con la base de datos ImageNet [45]. Esta base datos se caracteriza por poseer imágenes de varios tipos, incluyendo animales clasificados a distintos niveles de ontología, y objetos inanimados, como vehículos y cohetes espaciales. La base de datos ImageNet está construida en la estructura jerárquica provista por WordNet, con árboles y subárboles [45].

Dentro de la primera arquitectura graficada en la Figura 1.2, también se especifica un etapa en la cual las características de esta arquitectura serán refinadas mediante el algoritmo KMeans, elegido por ser usado en varios trabajos [39, 40, 41].

Por último, debido a que éste es un escenario altamente controlado a pesar de la variabilidad de las imágenes, queremos comprobar si éstas características no solo se comportan mejor con o sin el refinamiento de las mismas a través del algoritmo

de KMeans, sino si se comportan mejor con un algoritmo de clasificación lineal, como es Support Vector Machine con un kernel lineal, o con un algoritmo no lineal, como es kNN.

2.4. Diseño, entrenamiento y validación preliminar: entrenamiento

En esta tarea se realizará la implementación de los algoritmos descritos anteriormente. Primero, tenemos el código que extrae las características de las imágenes, que es descrito de manera más visual por la siguiente Figura 2.4.

El código en el ANEXO II implementa la extracción de características aquí descrita. Cabe recalcar que la implementación de este proceso se dio con la ayuda del framework Keras para Python [46], la cual es una API de alto nivel capaz de correr sobre otros frameworks como Tensorflow, Theano y CNTK. Este framework fue pensado para habilitar una experimentación rápida, ya que ellos plantean que "ir de la idea al resultado con el menor retraso posible es la clave para realizar una buena investigación" [47].

Para el caso de la extracción de características del presente trabajo, el framework de Keras nos habilitó realizar de manera sencilla la técnica de transferencia de aprendizaje, redimensionamiento de la imagen, y extracción de características de la imagen con la red neuronal pre-entrenada, como se podrá observar en el ANEXO II.

Luego de que estas características son recolectadas, se pasan por un algoritmo de KMeans, el cual fue explicado en el apartado 1.5. Este algoritmo lo que hará es refinar las características. Esto lo hace mediante la agrupación de los datos en el número de clústers que se programe al algoritmo. Este número "ideal" de clústers se obtuvo mediante la aplicación del algoritmo de elbow a los vectores de características.

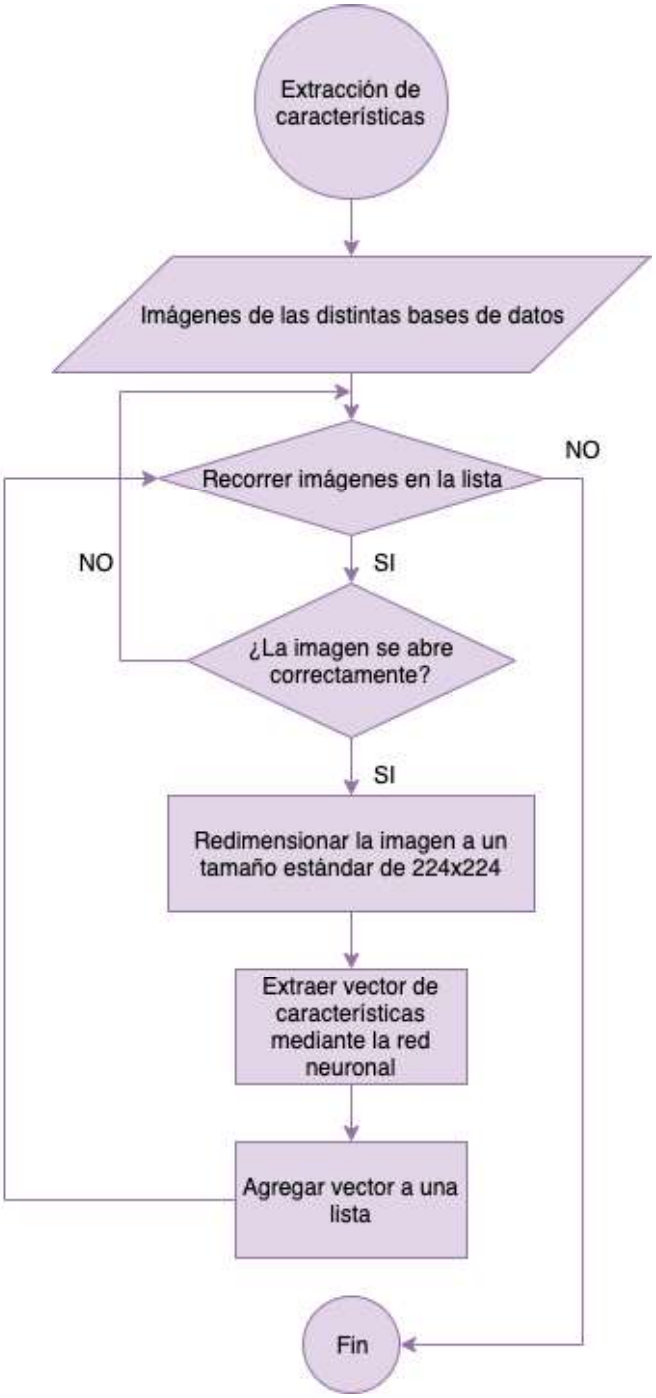

Figura 2.5. Proceso de extracción de características

El algoritmo de elbow fue implementado manualmente, tal como se indica en el ANEXO III, el resultado obtenido se puede visualizar en la Figura 2.6. En el gráfico se puede observar la sumatoria de las distancias de los puntos con su centroide en contraste con el número de clústers.

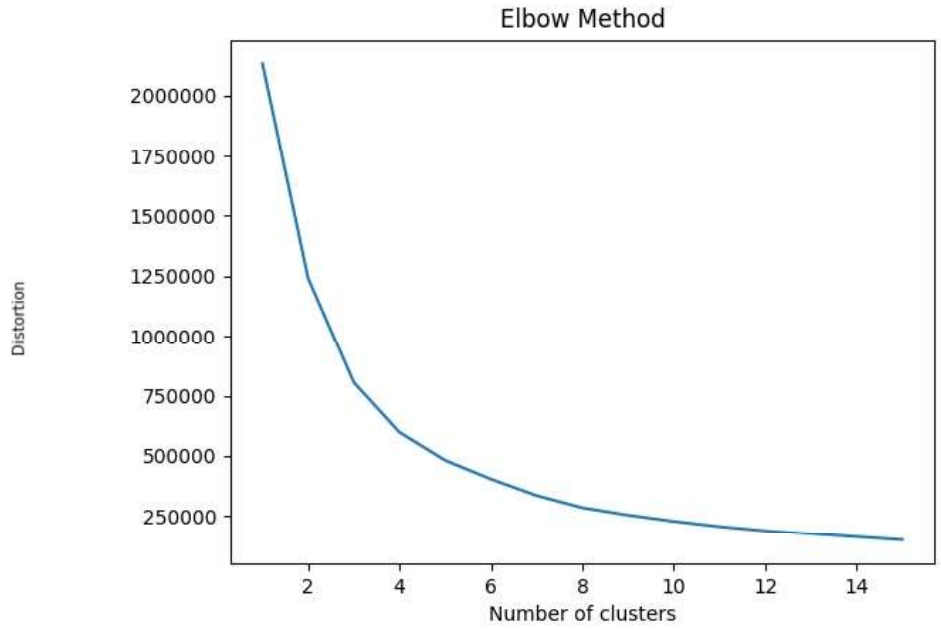

Figura 2.6. Algoritmo elbow para el presente trabajo, en el cual se observa que la gráfica se estabiliza con 4 clústers

Esta fase también comprende el entrenamiento de los algoritmos que evaluarán la calidad de las características obtenidas por la red neuronal convolucional. De esta manera, los vectores de características obtenidos en la etapa anterior alimentarán tanto al algoritmo de kNN como al algoritmo de Support Vector Machine, cuyo funcionamiento fue explicado en el apartado 1.5.

Para implementar estos algoritmos, se uso otro framework muy conocido entre los científicos de datos con expriencia en Python, la API de Scikit-learn [48].

El proyecto scikit learn empezó en el 2007 como parte del evento “Google Summer Code” organizado por David Copernau. Un poco más tarde ese mismo año, Matthieu Brucher empezó a trabajar en el proyecto como parte de su tesis. En el 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel tomaron liderazgo del proyecto e hicieron su primer release al público, el 1 de Febrero de 2010. Desde entonces varios releases han aparecido siguiendo un ciclo de más o menos 3 meses, y una creciente comunidad internacional ha liderado el proyecto [49].

Luego de extraer los vectores de características con la red neuronal, se procederá a reducir la dimensión de los mismos a 100 componentes por vector con el algoritmo PCA explicado en el apartado 1.5, por motivos de rendimiento de los algoritmos

clasificadores, y de la disponibilidad de recursos físicos que soporten el procesamiento de vectores de mayor tamaño. La implementación del algoritmo PCA se puede observar en el Anexo II.

Para entrenar los algoritmos kNN y SVM se usaron 35000 datos, que son los vectores de carcaterísticas ya mencionados. Sin embargo, para cumplir con las arquitecturas propuestas, se tiene un set de vectores que es el obtenido después de aplicar PCA a los datos. En este punto divergen los dos modelos explicados en el apartado 1.4, ya que en uno de los modelos se aplicará el algoritmo KMeans a los vectores de características después de PCA. Estos últimos vectores de características tendrán una dimensión de 4 componentes.

La implementación de estos algoritmos se encontrará en el ANEXO IV para kNN y en el ANEXO V para SVM con kernel linear.

2.5. Diseño, entrenamiento y validación preliminar: validación

Esta es la última etapa de la primera fase. Una vez que todos los algoritmos finales de clasificación han sido entrenados, kNN y SVM, viene la etapa de validación preliminar. Para ellos, antes de entrenar los algoritmos, debimos haber separado una parte de los datos de entrenamiento y reservarlos como datos de validación preliminar. En este caso, para todos los algoritmos se decidió separar el 25% de los datos de entrenamiento para el proceso de validación preliminar. Esto se podrá observar en la implementación de estos algoritmos en los ANEXOS IV y V. Esta técnica de evaluación del modelo se conoce como validación preliminar, en la cual se toman algunos datos de entrenamiento al azar y, en lugar de usarlos para entrenar al modelo se los usa para evaluar el mismo luego de haber sido entrenado. Esto se logra haciendo que el modelo entrenado entregue una clasificación de los datos que se separaron al principio, y se compara este resultado con la verdadera clasificación de estos datos, ya que esto es conocido. De esta comparación se obtiene un porcentaje de resultados acertados, y ésta es la precisión de el algoritmo [50].

2.6. Testeo

La fase de testeo es la última de la metodología tradicional del aprendizaje de máquina. Esta fase se realiza con datos diferentes a los datos de entrenamiento.

Esta fase es importante porque nos deja saber si el modelo está en riesgo de sufrir un overfitting. Estadísticamente hablando, se habla de overfitting cuando la producción de un análisis se ajusta demasiado o exactamente a cierto conjunto de datos, por lo que probablemente fallará en ajustar nuevos datos si se los presentan, o en su predicción [51]. Los resultados del testeo del presente trabajo se mostrarán en el siguiente apartado. Para la fase de testeo, en el presente trabajo se usaron 3530 datos no pertenecientes al dataset de entrenamiento ni de validación preliminar. Para un mejor entendimiento, se puede referir a las arquirecturas expuestas en el apartado 1.4.

3. RESULTADOS Y DISCUSIÓN

3.1. Resultados

Los resultados presentados a continuación se consiguieron después de realizar la extracción de características con la red neuronal, y luego haber extraído 100 componentes principales mediante el algoritmo PCA. Estos 100 componentes fueron determinados con dos criterios principales: recursos de hardware y precisión. Mediante experimentación se determinó que 200 componentes eran demasiado intensivos en procesamiento para una máquina normal dados los recursos disponibles, y con 150 datos la precisión tenía una variación de entre 0.01% y 0.05%, al mismo tiempo que seguía representando una carga significativamente alta en procesamiento. Considerando lo anterior, se determinó que 100 componentes eran adecuados. En la Figura 3.1 se puede observar la distribución de los datos después de la extracción de características con la red neuronal, las mismas que fueron reducidas a dos componentes principales mediante PCA. En el eje x tenemos el primer componente principal, que representa el 74.11% de la varianza, y en el eje y tenemos el segundo componente principal, que representa el 12.46% de la varianza. En conjunto los dos elementos contienen el 86.57% de la información.

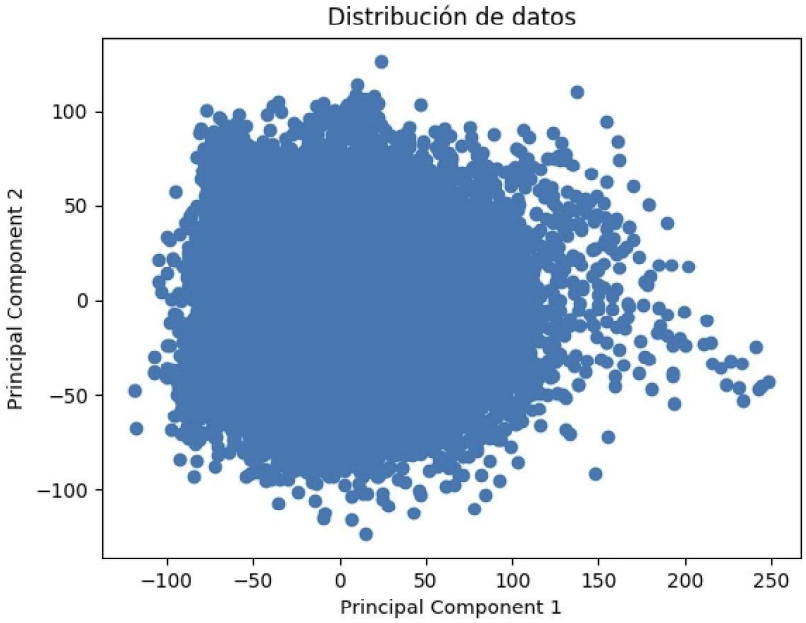

Figura 3.1. Distribución original de los datos representados con sus componentes principales 1 y 2

A partir de aquí, se dividirán los resultados de acuerdo a si los vectores de características fueron refinados con el algoritmo KMeans o no.

Caso 1: Resultados obtenidos de la primera arquitectura, sin el uso de KMeans

En este primer caso se presentarán los algoritmos usados para evaluar las características obtenidas por la red neuronal, sin aplicar el algoritmo KMeans, y sus resultados de clasificación y testeo.

a. kNN

Se llegó a la conclusión de que el número óptimo de vecinos para este problema era 7 mediante experimentación y haciendo una comparativa de la precisión usando diferentes k, como se observa en la Figura 3.2.

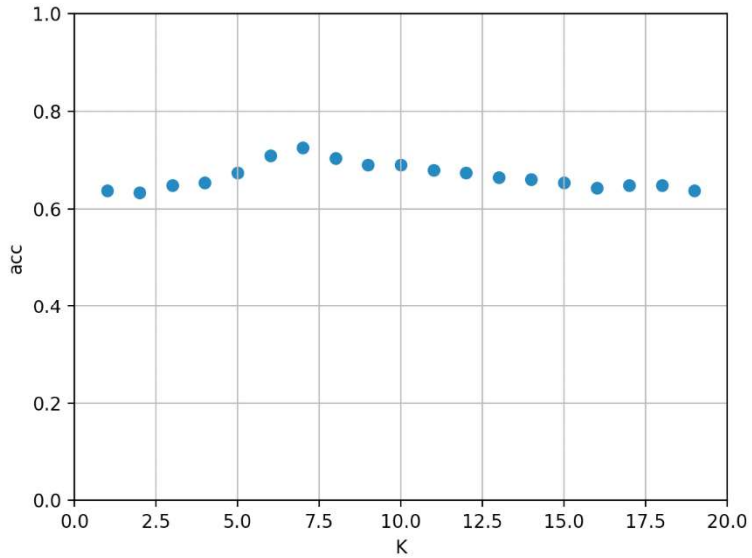

Figura 3.2. Comparativa de número de K vs precisión

En la Figura 3.3, podemos observar la matriz de confusión generada por el entrenamiento y validación del algoritmo kNN con las características sin aplicar el algoritmo KMeans, y en la Tabla 3.1 podemos observar los errores absolutos calculados con los datos de entrenamiento y los datos de testeo, la cual parece mostrar un sobreajuste para este caso.

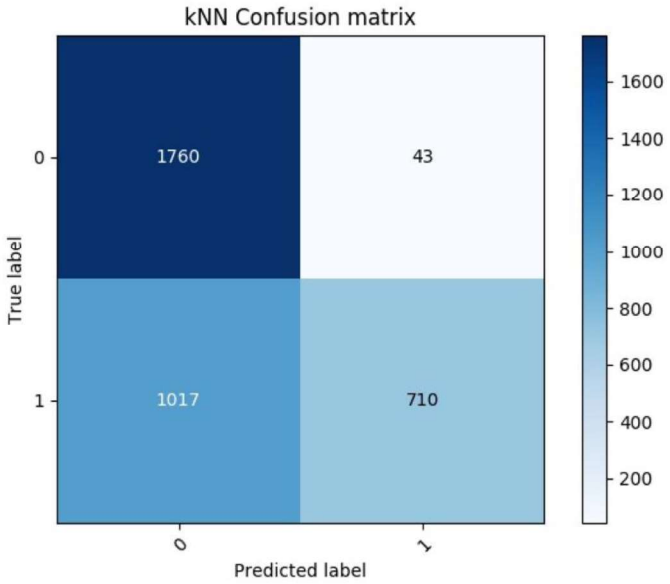

Figura 3.3. Matriz de confusión del algoritmo kNN sobre los datos de testeo.

Tabla 3.1. Error absoluto de entrenamiento y de testeo para el algoritmo kNN

Error absoluto de entrenamiento	Error absoluto de testeo
23.367%	30.028%

b. SVM con kernel lineal

En la Figura 3.4, podemos observar la matriz de confusión generada por el entrenamiento y validación del algoritmo SVM usando un kernel lineal, explicado en el apartado 1.5, para la transformación de características, con los vectores de características sin aplicar el algoritmo KMeans, y en la Tabla 3.2 podemos observar los errores absolutos de este algoritmo calculados con los datos de entrenamiento y los datos de testeo, lo cual muestra que aparentemente el modelo presentaría sobreajuste.

Tabla 3.2. Error absoluto de entrenamiento y de testeo para el algoritmo SVM con kernel linear

Error absoluto de entrenamiento	Error absoluto de testeo
44.609%	55.694%

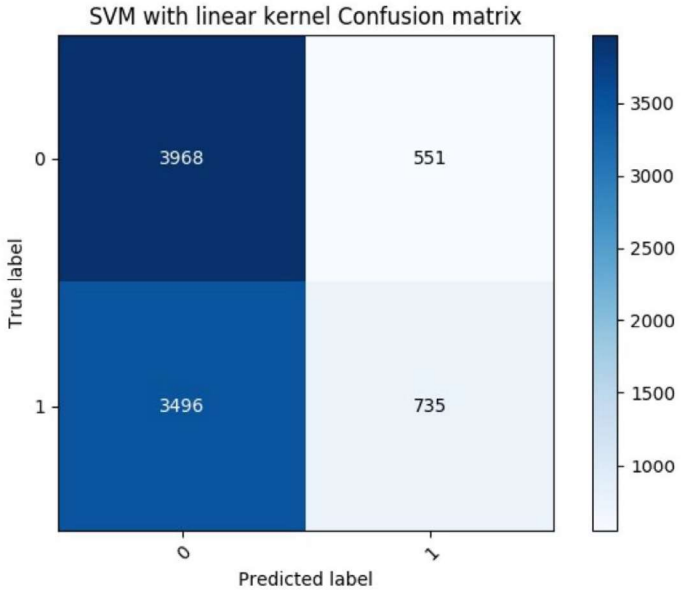

Figura 3.4. Matriz de confusión del algoritmo SVM con kernel linear.

c. SVM con kernel polinomial

En la Figura 3.5, podemos observar la matriz de confusión generada por el entrenamiento y validación del algoritmo SVM usando un kernel polinomial, explicado en el apartado 1.5, para la transformación de características, con los vectores de características sin aplicar el algoritmo KMeans, y en la Tabla 3.3 podemos observar los errores absolutos de este algoritmo calculados con los datos de entrenamiento y los datos de testeo, los cuales muestran que existe sobreajuste.

Tabla 3.3. Error absoluto de entrenamiento y de testeo para el algoritmo SVM con kernel polinomial

Error absoluto de entrenamiento	Error absoluto de testeo
12.403%	52.181%

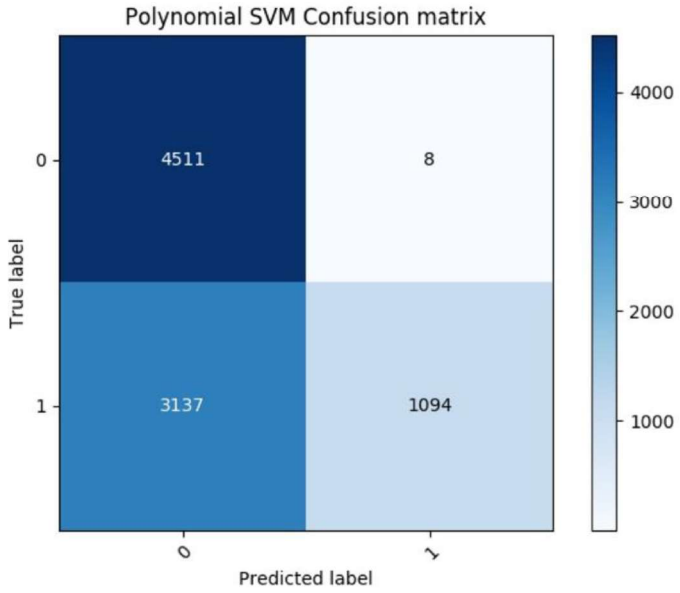

Figura 3.5. Matriz de confusión del algoritmo SVM con kernel polinomial

d. Regresión logística

Este algoritmo, al igual que SVM con kernel polinomial, no fue plantado al inicio del trabajo, sin embargo, se planteó de interés para comparar el comportamiento del algoritmo SVM con kernel lineal. En la Figura 3.6, podemos observar la matriz de confusión generada por el entrenamiento y validación del algoritmo de regresión lineal, que se lo ha implementado para comparar su comportamiento con el algoritmo de SVM con kernel linear, con los vectores de características sin aplicar el algoritmo KMeans, y en la Tabla 3.4 podemos observar los errores absolutos de este algoritmo calculados con los datos de entrenamiento y los datos de testeo, los cuales muestran que el modelo parece mostrar sobreajuste.

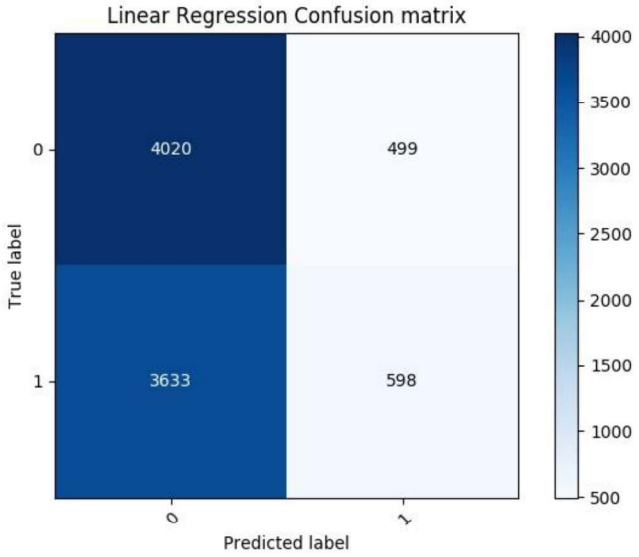

Figura 3.6. Matriz de confusión del algoritmo de regresión logística

Tabla 3.4. Error absoluto de entrenamiento y de testeo para el algoritmo de regresión logística

Error absoluto de entrenamiento	Error absoluto de testeo
45.333%	55.212%

Caso 2: Resultados obtenidos de la segunda arquitectura, usando el algoritmo KMeans para refinamiento de características

Como ya se mencionó anteriormente, los mismos algoritmos se usaron para evaluar los vectores de características ahora refinados y ajustados con el algoritmo KMeans usando 4 clústeres. La distribución de datos ajustados con el algoritmo KMeans es como se muestra en la Figura 3.7. En el apartado 2.4, en la Figura 2.6 podemos observar que, mediante el método de Elbow, el número óptimo de clústeres para el presente problema es 4, por ello se usó dicho número.

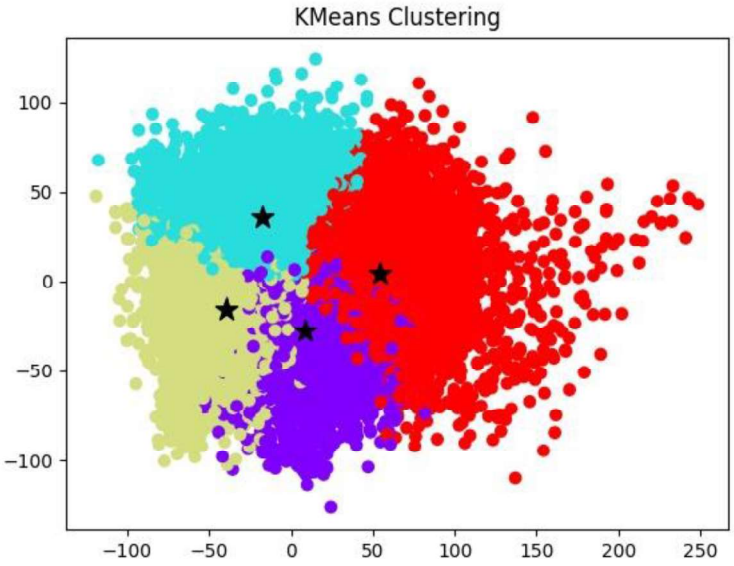

Figura 3.7. Distribución de datos ajustados con el algoritmo KMeans

a. kNN

En la Figura 3.8, podemos observar la matriz de confusión generada por el entrenamiento y validación del algoritmo kNN con las características aplicando el algoritmo KMeans, y en la Tabla 3.5 podemos observar los errores absolutos calculados con los datos de entrenamiento y los datos de testeo, por los cuales se observa que el modelo puede sufrir sobreajuste.

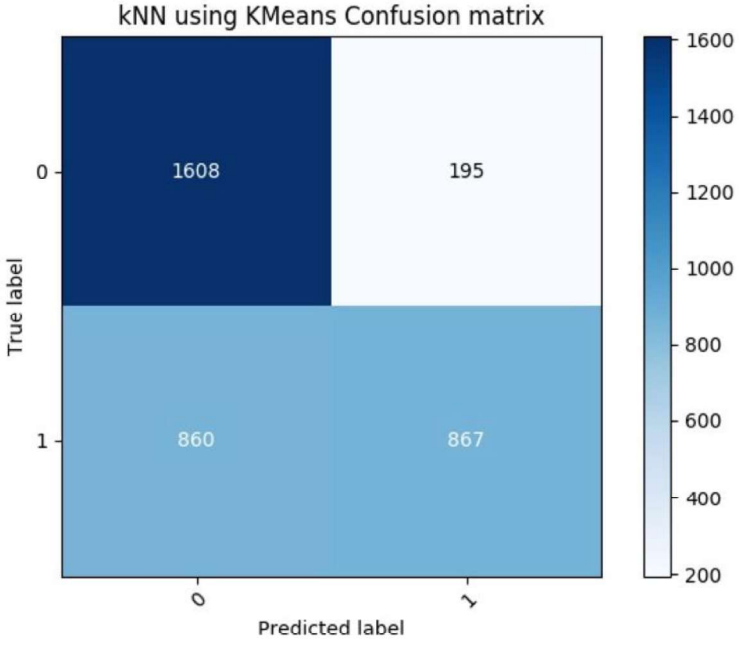

Figura 3.8. Matriz de confusión del algoritmo kNN usando el algoritmo KMeans para refinamiento de características

Tabla 3.5. Error absoluto de entrenamiento y de testeo para el algoritmo SVM con kernel polinomial

Error absoluto de entrenamiento	Error absoluto de testeo
19.649%	29.886%

b. SVM con kernel lineal

En la Figura 3.9, podemos observar la matriz de confusión generada por el entrenamiento y validación del algoritmo SVM usando un kernel lineal, explicado en el apartado 1.5, para la transformación de características, con los vectores de características sin aplicar el algoritmo KMeans, y en la Tabla 3.6 podemos observar los errores absolutos de este algoritmo calculados con los datos de entrenamiento y los datos de testeo.

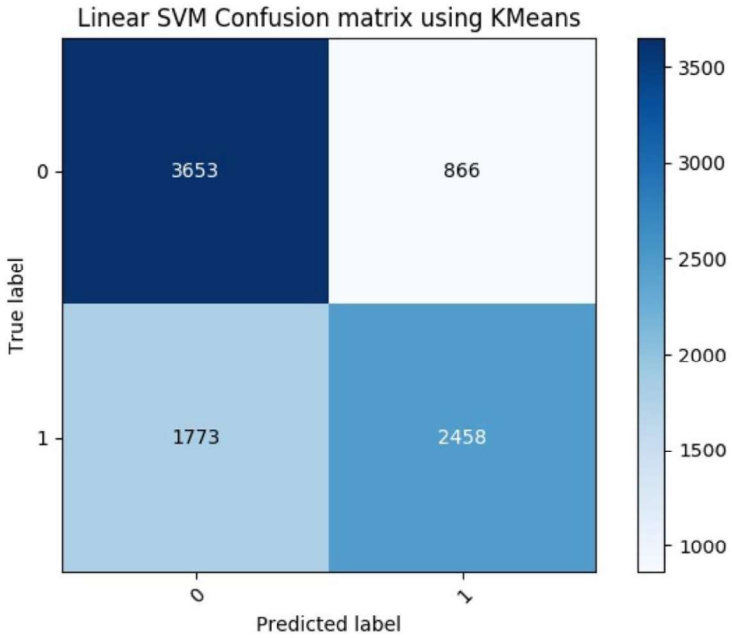

Figura 3.9. Matriz de confusión para el algoritmo SVM con kernel linear usando KMeans para refinamiento de características

Tabla 3.6. Error absoluto de entrenamiento y de testeo para el algoritmo SVM con kernel polinomial

Error absoluto de entrenamiento	Error absoluto de testeo
30.548%	31.189%

c. SVM con kernel polinomial

Tabla 3.7. Error absoluto de entrenamiento y de testeo para el algoritmo SVM con kernel polinomial usando el algoritmo KMeans para refinamiento de características

Error absoluto de entrenamiento	Error absoluto de testeo
28.773%	34.419%

En la Figura 3.10, podemos observar la matriz de confusión generada por el entrenamiento y validación del algoritmo SVM usando un kernel polinomial, explicado en el apartado 1.5, para la transformación de características, con los vectores de características aplicando el algoritmo KMeans, y en la Tabla 3.7 podemos observar los errores absolutos de este algoritmo calculados con los datos de entrenamiento y los datos de testeo.

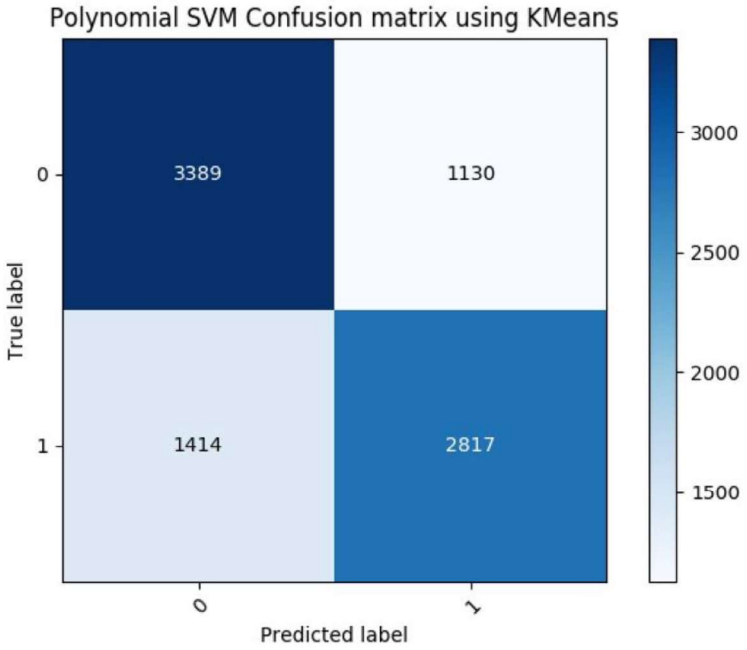

Figura 3.10. Matriz de confusión del algoritmo SVM con kernel polinomial usando el algoritmo KMeans para refinamiento de características

d. Regresión logística

En la Figura 3.11, podemos observar la matriz de confusión generada por el entrenamiento y validación del algoritmo de regresión logística, que se lo ha implementado para comparar su comportamiento con el algoritmo de SVM con kernel linear, con los vectores de características aplicando el algoritmo KMeans, y en la Tabla 3.8 podemos observar los errores absolutos de este algoritmo calculados con los datos de entrenamiento y los datos de testeo.

Tabla 3.8. Error absoluto de entrenamiento y de testeo para el algoritmo de regresión logística usando el algoritmo KMeans para refinamiento de características

Error absoluto de entrenamiento	Error absoluto de testeo
30.274%	31.246%

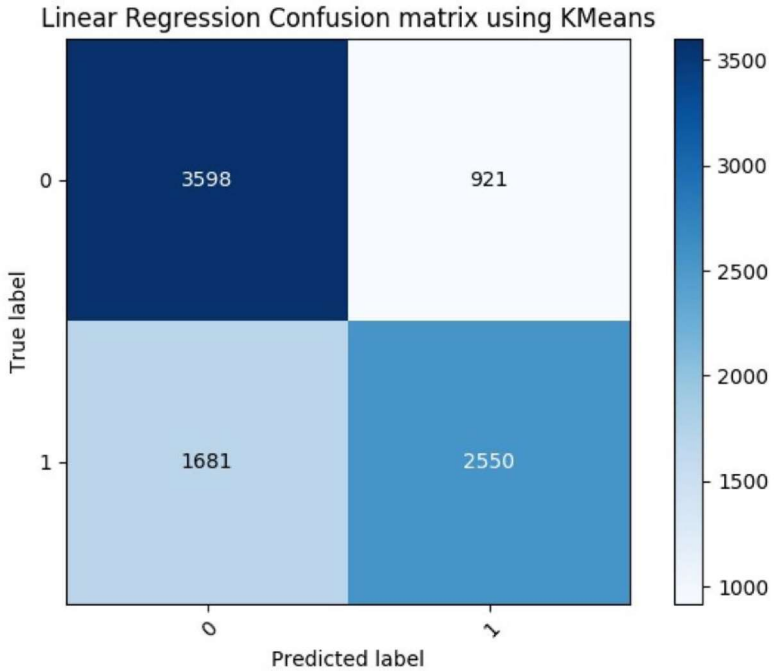

Figura 3.11. Matriz de confusión del algoritmo de regresión logística usando el algoritmo KMeans para refinamiento de características

A continuación, en la Tabla 3.9, podemos observar una comparativa de todos los algoritmos anteriormente mencionados en sus dos casos, con sus porcentajes obtenidos del proceso de validación preliminar y del proceso de testeo. También se

logra una comparativa más directa entre el rendimiento de los algoritmos con las características antes de aplicar el refinamiento con el algoritmo KMeans, y después del mismo.

Tabla 3.9. Comparativa de Resultados

	Algoritmos	Precisión de validación preliminar	Precisión de testeo
CASO 1	Algoritmo kNN con 7 vecinos **sin KMeans**.	72.514%	69.971%
	Algoritmo SVM con kernel linear **sin KMeans**.	53.748%	44.305%
	Algoritmo de regresión logística **sin KMeans**.	52.777%	44.787%
	Algoritmo SVM con kernel polinomial **sin KMeans.**	64.057%	47.818%
CASO 2	Algoritmo kNN con 7 vecinos **con KMeans**.	74.8%	70.113%
	Algoritmo SVM con kernel linear **con KMeans**.	69.84%	68.801%
	Algoritmo de regresión logística **usando KMeans**.	70.262%	68.753%
	Algoritmo SVM con kernel polinomial **usando KMeans.**	70.925%	65.580%
	Algoritmo Naive Bayes, usando Domain Adaptation como preprocesamiento [52]	N/A	24%

3.2. Discusión

Al observar los datos en la Tabla 3.9, es fácilmente observable que los modelos no se ajustan bien cuando se usa un algoritmo de clasificación lineal, como es Support Vector Machine usando un kernel lineal. Sin embargo, se puede observar que el rendimiento mejora cuando se usa el refinamiento de características con KMeans. Y si observamos el comportamiento de ambos modelos de extracción de

características al ser evaluados por el algoritmo kNN, podemos observar que, de igual manera, hay un mejor comportamiento cuando se usa el refinamiento de características. Se observa también una mayor varianza entre la presición del algoritmo entre su entrenamiento y su testeo, probablemente a que este tipo de características complejas en tamaño se ajustan mejor con modelos no lineales.

De esta manera se responde a la pregunta de investigación de manera afirmativa, en todos los casos estudiados la aplicación del algoritmo KMeans como refinamiento de características permite mejorar el rendimiento de los algoritmos de clasificación aplicados posteriormente.

Tambien, para el caso estudiado si se necesita seleccionar el modelo más eficiente en términos de exactitud de clasificación, de acuerdo a la Tabla 3.9, es el modelo con kNN usando KMeans.

Se logró completar los objetivos planteados en el plan de tesis, los cuales eran obtener resultados de exactitud de clasificación de los dos modelos de arquitecturas diferentes, compararlos y obtener una respuesta a la pregunta de investigación. Un trabajo futuro podría ser mejorar los modelos que se comportaron mejor con las características extraídas de las imágenes, los modelos no lineales.

4. CONCLUSIONES

- De la implementación de la primera arquitectura propuesta, se obtuvieron dos valores de precisión muy diferentes en los dos algoritmos de evaluación, con lo que se concluye que, de acuerdo a los datos presentados en la Tabla 3.9, para los datos usados en el presente problema, los algoritmos de clasificación lineal no se ajustan tan bien para el presente modelo sin el refinamiento de características con el algoritmo KMeans.
- Después del refinamiento de características con el algoritmo KMeans, cada vector tenía un menor número de componentes, y ésto reducía la complejidad de los datos en tamaño o dimensiones al momento de ajustar el modelo. Esto se puede observar en el rendimiento de los modelos.
- Observando el desempeño que tuvieron las características en el Caso 1 y Caso 2, después de haberlos evaluado con el algoritmo SVM con kernel lineal y no lineal, y observando que el mejor comportamiento lo tuvo el algoritmo kNN usando el algoritmo KMeans para refinamiento de

características, se llega a la conclusión de que este problema no se comporta adecuadamente con clasificadores lineales, lo que probablemente se deba a la naturaleza del problema y a la distribución de los datos.

- La utilización del framework Keras sobre TensorFlow facilita el proceso de integración de la fase de extracción de características con la fase de clasificación donde el foco principal de la investigación no se halle sobre las redes neuronales convolucionales. Su uso e implemetación es bastante amigable, a diferencia de TensorFlow puro.

Trabajos futuros

En el presente trabajo se usaron distintas bases de datos de rostros bajo diferentes condiciones de posición, iluminación, e incluso color, ya que varias de las imágenes usadas eran a color, y otra buena parte eran a blanco y negro. A pesar de esto, se logró una precisión en los algoritmos algo interesante, llegando cerca del 75% de precisión en uno de los casos. Una oportunidad de trabajo futuro es continuar con la investigación de modelos de clasificación de rostros con imágenes bajo diversos ambientes para una extracción de características automatizadan como la del presente trabajo.

Otra oportunidad de mejora del presente trabajo es la mejora del vector de características obtenido de la red neuronal, con el propósito de mejorar la precisión de los algoritmos de clasificación. Podría considerarse el entrenamiento de las últimas capas de la red neuronal para ajustar la extracción de características al problema de clasificación de rostros, o ajustar el número de componentes reducidos en la etapa PCA, en el caso de disponer de los recursos de hardware necesarios.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] Deep AI, Inc., «Machine Learning,» 2019. [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/machine-learning. [Último acceso: 7 November 2019].

[2] C. Szegedy, V. Vanhouke, S. Ioffe, J. Shlens y Z. Wojna, «Rethinking the Inception Architecture for Computer Vision,» *Computer Vision and Pattern Recognition (CVPR),* 2016.

[3] E. Culurciello, A. Canziani y A. Paszke, «An Analysis of Deep Neural Network Models for Practical Applications,» 2016.

[4] S. Changpinyo, M. Sandler y A. Zhmoginov, «The Power of Sparsity in Convolutional Neural Networks,» 2017.

[5] T. Akilan, J. Wu, A. Safaei y W. Jiang, «A late fusion approach for harnessing multi-cnn model high-level features,» de *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Banff, AB, Canada, 2017.

[6] N. S. Altman, «An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression,» *The American Statistician,* vol. 46(3), pp. 175-185, 1992.

[7] E. Fix y J. L. Hodges, « Discriminatory analysis—Nonparametric discrimination: Consistency properties.,» *Internat. Statist,* nº 57, pp. 238-247, 1989.

[8] C. Cortes y V. Vladimir, «Support-vector networks,» *Machine Learning,* vol. 20, nº 3, pp. 273-297, 1995.

[9] L. Kong, J. Lv, M. Li y H. Zhang, «Extracting Generic Features of Artistic Style via Deep Convolutional Neural Network,» *Proceedings of the International Conference on Video and Image Processing - ICVIP,* 2017.

[10] M. Grgic, K. Delac y S. Grgic, «surveillance cameras face database,» *Multimedia Tools and Applications Journal,* vol. 51, nº 3, pp. 863-879, 2011.

[11] W. Bainbridge, «10k US Adult Faces Database,» 2012-2019. [En línea]. Available: https://wilmabainbridge.com/facememorability2.html. [Último acceso: 30 October 2019].

[12] Yale, «The Extended Yale Face Database B,» 2001.

[13] J. H. B. H. a. V. B. B. Weyrauch, «Component-based Face Recognition with 3D Morphable Models,» *First IEEE Workshop on Face Processing in Video,* 2004.

[14] J. J. R. G. G. R. M. R. Ashwin T S, «An E-learning System With Multifacial Emotion Recognition Using Supervised Machine Learning,» *2015 IEEE Seventh International Conference on Technology for Education (T4E).,* 2015.

[15] D. Ciresan, U. Meier, J. Masci, L. Gambardella y J. Schmidhuber, «Flexible, High Performance Convolutional Neural Networks for Image Classification.,» *International Joint Conference on Artificial Intelligence, North America,* 2011.

[16] Deep AI Inc., «Convolutional Neural Networks,» 2019. [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/convolutional-neural-network. [Último acceso: 29 October 2019].

[17] M. T. Heideman, «Convolution and Polynomial Multiplication,» de *Multiplicative Complexity, Convolution, and the DFT*, New York, NY, Springer New York, 1988, pp. 27-60.

[18] Deep AI, Inc., «What is Max Pooling?,» 2019. [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/max-pooling. [Último acceso: 29 October 2019].

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever y R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting,» *Journal of Machine Learning Research,* vol. 15, pp. 1929-1958, 2014.

[20] Google Developers, «Multi-Class Neural Networks: Softmax,» Google LLC, 2019. [En línea]. Available: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax. [Último acceso: 12 November 2019].

[21] L. Torrey y J. Shavlik, «COMPUTER SCIENCES,» 16 January 2009. [En línea]. Available: http://ftp.cs.wisc.edu/machine-learning/shavlik-group/torrey.handbook09.pdf. [Último acceso: 20 November 2019].

[22] J. Masci, U. Meier, D. Cireşan, J. Schmidhuber y T. Honkela, «Stacked Convolutional Auto-Encoders for Hierarchical Feature Extraction,» de *Artificial Neural Networks and Machine Learning – ICANN 2011*, Berlin, Heidelberg, 2011.

[23] T. Li, A. Chan y A. Chun, «Automatic Musical Pattern Feature Extraction Using Convolutional Neural Network,» de *International MultiConference of Engineers and Computer Scientists 2010 Vol I.*, Hong Kong, 2010.

[24] K. Simonvan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition,» 2014.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke y A. Rabinovich, «Going Deeper with Convolutions,» *Computer Vision and Pattern Recognition (CVPR),* 2015.

[26] A. Krizhevsky, I. Sutskever y G. Hinton, «ImageNet classification with deep convolutional neural networks,» *Communications of the ACM,* vol. 60, nº 6, pp. 84-90, 2017.

[27] Q. Kuang y L. Zhao, «A Practical GPU Based KNN Algorithm,» de *Proceedings of the Second Symposium International Computer Science and Computational Technology*, Huangshan, 2009.

[28] S. Amari y S. Wu, «Improving support vector machine classifiers by modifying kernel functions,» *Neural Networks,* vol. 12, pp. 783-789, 1999.

[29] Deep AI, Inc., «What are Support Vector Machines?,» 2019. [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/support-vector-machine. [Último acceso: 30 October 2019].

[30] Deep AI, Inc., «What is Unsupervised Learning?,» 2019. [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/unsupervised-learning. [Último acceso: 30 October 2019].

[31] Deep AI, Inc, «How does K-Means Clustering Work?,» 2019. [En línea]. Available: https://deepai.org/machine-learning-glossary-and-terms/k-means. [Último acceso: 29 October 2019].

[32] J. MacQueen, «Some methods for classification and analysis of multivariate observations.,» de *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, California, 1967.

[33] M. A. Syakur, «Integration K-Means Clustering Method and Elbow Method For Identification of The Best Customer Profile Cluster,» de *IOP Conference Series: Materials Science and Engineering*, 2018.

[34] M. Ringnér, «What is principal component analysis?,» *Nature Biotechnology,* vol. 26, nº 3, pp. 303-304, 2008.

[35] University of Zagreb, Faculty Electrical Engineering and Computing, «SCface - Surveillance Cameras Face Database,» 2011. [En línea]. Available: http://www.scface.org/. [Último acceso: 12 November 2019].

[36] W. Bainbridge, P. Isola y A. & Oliva, «The intrinsic memorability of face images,» *Journal of Experimental Psychology: General. Journal of Experimental Psychology: General,* vol. 142, nº 4, pp. 1323-1334, 2013.

[37] VCL, «Face Recognition Homepage, Databases,» 28 August 2019. [En línea]. Available: http://www.face-rec.org/databases/. [Último acceso: 28 October 2019].

[38] MIT, «Face Recognition Database,» 2005. [En línea]. Available: http://cbcl.mit.edu/software-datasets/heisele/facerecognition-database.html. [Último acceso: 30 October 2019].

[39] Y. Sato, I. Yoda y K. Sakaue, «Automatic face classifications by self-organization for face recognition,» *2003 IEEE International SOI Conference.,* nº No.03CH37443, pp. 165-172, 2003.

[40] T. Lange, M. H. C. Law, A. K. Jain y J. M. Buhman, «Learning with Constrained and Unlabelled Data.,» *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05).,* 2005.

[41] X. Wang y K. Chandra, «Age Estimation via Unsupervised Neural Networks,» *IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG). ,* vol. 11, 2015.

[42] Y. Hayakawa, T. Oonuma, H. Kobayashi, A. Takahashi, S. Chiba y N. Fujiki, «Feature extraction of video using deep neural network,» *2016 IEEE 15th International Conference on Cognitive Informatics & Cognitive Computing (ICCI*CC),* vol. 15, 2016.

[43] H. Çevikalp, G. Dordinejad y M. Elmas, «Feature extraction with convolutional neural networks for aerial image retrieval,» *2017 25th Signal Processing and Communications Applications Conference (SIU),* 2017.

[44] F. Del Frate, G. Licciardi, F. Pacifici, C. Pratola y D. Solimini, «Pulse Coupled Neural Network for automatic features extraction from COSMO-Skymed and TerraSAR-X imagery,» de *2009 IEEE International Geoscience and Remote Sensing Symposium*, Cape Town, South Africa, 2009.

[45] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li y L. Fei-Fei, «ImageNet: A large-scale hierarchical image database,» de *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, 2009.

[46] F. Chollet y others, «Keras: The Python Deep Learning library,» *Astrophysics Source Code Library, record ascl:1806.022,* June 2018.

[47] F. Chollet, «Keras: The Python Deep Learning library, Keras Documentation,» 2018. [En línea]. Available: https://keras.io/. [Último acceso: 12 November 2019].

[48] P. e. al., «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research,* vol. 12, pp. 2825-2830, 2011.

[49] Scikit-learn developers, «About us,» 2019. [En línea]. Available: https://scikit-learn.org/stable/about.html#people. [Último acceso: 12 November 2019].

[50] G. P. Nason, «Wavelet Shrinkage Using Cross-Validation,» *Journal of the Royal Statistical Society: Series B (Methodological),* vol. 58, nº 2, pp. 463-479, 1996.

[51] Lexico.com, «overfitting,» Oxford, 2019. [En línea]. Available: https://www.lexico.com/en/definition/overfitting. [Último acceso: 12 November 2019].

[52] S. Banerjee, S. Samantha y S. Das, «Face Recognition in Surveillance Conditions with Bag-of-words, using Unsupervised Domain Adaptation,» de *Proceedings of the 2014 Indian Conference on Computer Vision Graphics and Image Processing - ICVGIP*, 2014.

6. ANEXOS

ANEXO I

SCface Database Release Agreement

SCface is a database of static images of human faces. Images were taken in uncontrolled indoor environment using five video surveillance cameras of various qualities. Database contains 4160 static images (in visible and infrared spectrum) of 130 subjects. Images from different quality cameras should mimic real-world conditions and enable robust face recognition algorithms testing, emphasizing different law enforcement and surveillance use case scenarios. SCface database of facial imagery was collected at the Video Communications Laboratory, Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. The database is used to develop, test, and evaluate face recognition algorithms. To advance the state-of-the-art in face recognition the SCface database will be made available to researchers in face recognition on a case by case basis only. All requests for the SCface database must be submitted in writing to Prof. Mislav Grgic. To receive a copy of the database, the researcher must sign this document and thereby agree to honour the restrictions listed in this document. Breach of the restrictions in this document will result in access being denied for the balance of the SCface database and being subject to civil damages in the case of publication of images that have not been approved for release, a violation of restriction 2 and 3 below. The database will be available to researchers online. There will be no charge for SCface database images. The researcher(s) agrees to the following restrictions on the database:

1. The SCface database will not be further distributed, published, copied, or further disseminated in any way or form whatsoever, whether for profit or not. This includes further distributing, copying or disseminating to a facility or organization unit in the requesting university, organization, or company. SCface images may be used to create new images but the downloaded images will not be modified.
2. For legal reasons and for the privacy of the database participants, images that can appear in reports, papers, and other documents published or released are those with subjectID: 001, 002, 045 or 102 in the SCface database. For these images no approval is required. All other images will never appear in any document of any form. Any image will never be released in commercial materials, newspapers, or other public medias.
3. All the images will be used for the purpose of academic or scientific research only. The SCface database, in whole or in part, will not be used for any commercial purpose in any form. Commercial distribution or any act related to commercial use of this database is strictly prohibited.
4. All documents and papers that report research results obtained using the SCface database will acknowledge the use of the SCface database. Use of the SCface database will be acknowledged as follows: "Portions of the research in this paper use the SCface database of facial images. Credit is hereby given to the University of Zagreb, Faculty of Electrical Engineering and Computing for providing the database of facial images." and citations to:

   Mislav Grgic, Kresimir Delac, Sonja Grgic, *SCface - surveillance cameras face database*, Multimedia Tools and Applications Journal, Vol. 51, No. 3, February 2011, pp. 863-879
5. A copy of all published papers, reports and other documents that use the SCface database must be forwarded to Prof. Mislav Grgic immediately upon publication, to the address available at the bottom of this document.
6. While every effort has been made to ensure accuracy, SCface database owners cannot accept responsibility for errors or omissions.
7. Use of SCface database is free of charge.
8. SCface database owners reserve the right to revise, amend, alter or delete the information provided herein at any time, but shall not be responsible for or liable in respect of any such revisions, amendments, alterations or deletions.
9. The final explanation of this agreement refers to Prof. Mislav Grgic.

PRINTED NAME: Diana Patricia Paz Ariza

SIGNATURE:

DATE: 22 Jul 2018

ORGANIZATION: Escuela Politécnica Nacional

ADDRESS: Ladrón de Guevara y Avda. Isidro Ayora 2018

E-MAIL: diana.paz@epn.edu.ec

ESCUELA POLITECNICA NACIONAL FACULTAD DE INGENIERIA DE SISTEMAS DEL ANATO FB QUITO

Prof. Mislav Grgic
University of Zagreb
Faculty of Electrical Engineering and Computing
Department of Wireless Communications
Unska 3 / XII
HR-10000 Zagreb
CROATIA

ANEXO II

```
# Código para obtener las características de las imágenes mediante una red neuronal convolucional

from keras import backend as K
from keras.preprocessing import image
from keras.applications.inception_v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input as preprocess_inputV3
import sklearn
from keras.utils import plot_model
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA, IncrementalPCA
from sklearn.neighbors import NearestNeighbors, KNeighborsClassifier
from sklearn.model_selection import train_test_split
import skimage.feature as feature
import matplotlib.pyplot as plt
import time

import numpy as np
import os

rootdir ='/home/ec2-user/Documents/ProcessedFaces'
landscapesRootDir = '/home/ec2-user/Documents/LandscapeImages'

modelInceptionV3 = InceptionV3(weights='imagenet', include_top=False)
modelInceptionV3.summary()
inceptionV3_feature_list = []

listOfFiles = list()

listOfLandscapes = list()
for (dirpath, dirnames, filenames) in os.walk(rootdir):
    listOfFiles += [os.path.join(dirpath, file) for file in filenames]

for (dirpath, dirnames, filenames) in os.walk(landscapesRootDir):
    listOfLandscapes += [os.path.join(dirpath, file) for file in filenames]

print("Number of faces: "+str(len(listOfFiles)))
print("Number of landscapes: "+str(len(listOfLandscapes)))
print('The scikit-learn version is {}.'.format(sklearn.__version__))
time.sleep(10)

# Print the files
for elem in listOfFiles:
    print(elem)

for file in listOfLandscapes:
    if file.endswith(".jpg"):
        try:
```

```
            img = image.load_img(file, target_size=(224, 224))
            img_data = image.img_to_array(img)
            img_data = np.expand_dims(img_data, axis=0)
            #With InceptionV3
            img_data = preprocess_inputV3(img_data)
            inceptionv3_feature = modelInceptionV3.predict(img_data)

            inceptionV3_feature_np = np.array(inceptionv3_feature)
            inceptionV3_feature_list.append(inceptionV3_feature_np.flatten())
            print("Added "+file+" to array")
        except Exception:
            print("Couldn't open file: "+file)
            continue

print("Processing faces...")

for file in listOfFiles:
    if file.endswith(".jpg") or file.endswith(".png"):
        img = image.load_img(file, target_size=(224, 224))
        img_data = image.img_to_array(img)
        img_data = np.expand_dims(img_data, axis=0)
        #With InceptionV3
        img_data = preprocess_inputV3(img_data)
        inceptionv3_feature = modelInceptionV3.predict(img_data)

        inceptionV3_feature_np = np.array(inceptionv3_feature)
        inceptionV3_feature_list.append(inceptionV3_feature_np.flatten())
        print("Added "+file+" to array")

inceptionV3_feature_list_np = np.array(inceptionV3_feature_list)

n = inceptionV3_feature_list_np.shape[0] # how many rows we have in the dataset
print("Number of rows in the dataset: "+str(n))
chunk_size = 1000 # how many rows we feed to IPCA at a time, the divisor of n
ipca = IncrementalPCA(n_components=100, batch_size=100)
reduced_data = None

for i in range(0, n//chunk_size):
print("Fitting pca until "+ str((i+1))*chunk_size)
if i == 0:
reduced_data = ipca.fit_transform(inceptionV3_feature_list_np[i*chunk_size : (i+1)*chunk_size])
elif i != 42:
reduced_data = np.vstack((reduced_data, ipca.fit_transform(inceptionV3_feature_list_np[i*chunk_size :
(i+1)*chunk_size])))
else:
continue
print("Reduced data until now: ")
print(reduced_data)
print(str(len(reduced_data)))
```

ANEXO III

```
# Código que implementa el método de Elbow, el cual permite determinar el número ideal de clústeres a usar en el algoritmo K_Means

import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import KMeans

wcss = []
for i in range (1,16): #15 cluster
    kmeans = KMeans(n_clusters = i, init='k-means++', random_state=0)
    kmeans.fit(reduced_data)
    wcss.append(kmeans.inertia_)
np.savetxt('reduced_features.txt', reduced_data, fmt='%d')
kmeans = KMeans(n_clusters=4, random_state=0, n_jobs=1, init='k-means++').fit(reduced_data)
plt.plot(range(1,16),wcss)
plt.title('Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('wcss')
plt.savefig("Elbow.jpg")
```

ANEXO IV

```
# Código para entrenar el algoritmo kNN

from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
import numpy as np
import time
import os

print("Loading array...")
reduced_data = np.loadtxt('reduced_features.txt', dtype = np.float64)
print("Array loaded")
print(reduced_data)
print(str(len(reduced_data)))

test_faces = np.random.choice([0,0], size=1803)
test_landscapes = np.random.choice([1,1], size=1727)
test_results = np.concatenate((test_landscapes, test_faces), axis=None)

reduced_test_data = np.loadtxt('test_features.txt', dtype=np.float64)

faces = np.random.choice([0,0], size=18123)
landscapes = np.random.choice([1,1], size=16877)
result = np.concatenate((landscapes, faces), axis=None)

print("Result...")
print(result)
print("Size of result: "+str(len(result)))

print("Training kNN...")
(trainFeat, testFeat, trainLabels, testLabels) = train_test_split(reduced_data, np.array(result), test_size=0.25,
random_state=42)
model = KNeighborsClassifier(n_neighbors=7, n_jobs=3)
model.fit(trainFeat, trainLabels)

acc = model.score(testFeat, testLabels)
print("Accuracy: "+str(acc))

print("Shape of the training data")
print(trainFeat.shape)
print("Shape of the testing data")
print(reduced_test_data.shape)

print("Test")
Test_predict = model.predict(reduced_test_data)
```

```
acc = accuracy_score(test_results, Test_predict)
print("Test Accuracy: "+str(acc))
cm = confusion_matrix(test_results, Test_predict)

cmap = plt.cm.Blues
classes = np.array([0,1])
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]), yticks=np.arange(cm.shape[0]),
        xticklabels=classes, yticklabels=classes,
        title="kNN Confusion matrix",
        ylabel='True label', xlabel='Predicted label')
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
fmt = 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
   for j in range(cm.shape[1]):
      ax.text(j, i, format(cm[i, j], fmt), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
plt.savefig("kNN_CM.jpg")

training_predictions = model.predict(trainFeat)
train_error = mean_absolute_error(training_predictions, trainLabels)
print("Training error: "+str(train_error))

test_error = mean_absolute_error(Test_predict, test_results)
print("Testing error: "+str(test_error))
```

ANEXO V

```
# Código para entrenar el algoritmo SVM con kerner linear.


from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC, SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
import numpy as np
import time
import os


print("Loading array...")
reduced_data = np.loadtxt('reduced_features.txt', dtype=np.float64)
print("Array loaded")
print(reduced_data)
print(str(len(reduced_data)))

reduced_test_data = np.loadtxt('test_features.txt', dtype=np.float64)

faces = np.random.choice([0,0], size=18123)
landscapes = np.random.choice([1,1], size=16877)
result = np.concatenate((landscapes, faces), axis=None)

test_faces = np.random.choice([0,0], size=1803)
test_landscapes = np.random.choice([1,1], size=1727)
test_results = np.concatenate((test_landscapes, test_faces), axis=None)

print("Result...")
print(result)
print("Size of result: "+str(len(result)))

print("Rescaling...")
ss = StandardScaler()
ss.fit(reduced_data)
reduced_data = ss.transform(reduced_data)
ss.fit(reduced_test_data)
reduced_test_data = ss.transform(reduced_test_data)

print("Training...")
X_train, X_test, y_train, y_test = train_test_split(reduced_data, np.array(result), test_size=0.25, random_state=42)
svm = LinearSVC(random_state=20)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
cm = confusion_matrix(y_test, y_pred)

cmap = plt.cm.Blues
classes = np.array([0,1])
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]), yticks=np.arange(cm.shape[0]),
        xticklabels=classes, yticklabels=classes,
        title="Polynomial SVM Confusion matrix",
        ylabel='True label', xlabel='Predicted label')
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
fmt = 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
   for j in range(cm.shape[1]):
      ax.text(j, i, format(cm[i, j], fmt), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
plt.savefig("SVM_CM_Kernel_poly.jpg")

print('Model accuracy is: ', accuracy)
print("Confusion matrix, without normalization")
print(cm)

print("Testing...")
test_predictions = svm.predict(reduced_test_data)
test_accuracy = accuracy_score(test_results, test_predictions)
print('Testing accuracy is: ', test_accuracy)

training_predictions = svm.predict(X_train)
train_error = mean_absolute_error(training_predictions, y_train)
print("Training error: "+str(train_error))

test_error = mean_absolute_error(test_predictions, test_results)
print("Testing error: "+str(test_error))
```

ANEXO VI

```
# Código para implementar el algoritmo SVM de kernel polinomial

from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC, SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error
import matplotlib.pyplot as plt
import numpy as np
import time
import os


print("Loading array...")
reduced_data = np.loadtxt('reduced_features.txt', dtype=np.float64)
print("Array loaded")
print(reduced_data)
print(str(len(reduced_data)))

reduced_test_data = np.loadtxt('test_features.txt', dtype=np.float64)

faces = np.random.choice([0,0], size=18123)
landscapes = np.random.choice([1,1], size=16877)
result = np.concatenate((landscapes, faces), axis=None)

test_faces = np.random.choice([0,0], size=1803)
test_landscapes = np.random.choice([1,1], size=1727)
test_results = np.concatenate((test_landscapes, test_faces), axis=None)

print("Result...")
print(result)
print("Size of result: "+str(len(result)))

print("Rescaling...")
ss = StandardScaler()
ss.fit(reduced_data)
reduced_data = ss.transform(reduced_data)
ss.fit(reduced_test_data)
reduced_test_data = ss.transform(reduced_test_data)

print("Training...")
X_train, X_test, y_train, y_test = train_test_split(reduced_data, np.array(result), test_size=0.25, random_state=42)
svm = SVC(kernel='poly', probability=True, random_state=42)
svm.fit(X_train, y_train)
y_pred = svm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
cm = confusion_matrix(y_test, y_pred)

cmap = plt.cm.Blues
classes = np.array([0,1])
```

```
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]), yticks=np.arange(cm.shape[0]),
        xticklabels=classes, yticklabels=classes,
        title="Polynomial SVM Confusion matrix",
        ylabel='True label', xlabel='Predicted label')
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
fmt = 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
plt.savefig("SVM_CM_Kernel_poly.jpg")

print('Model accuracy is: ', accuracy)
print("Confusion matrix, without normalization")
print(cm)

print("Testing...")
test_predictions = svm.predict(reduced_test_data)
test_accuracy = accuracy_score(test_results, test_predictions)
print('Testing accuracy is: ', test_accuracy)

training_predictions = svm.predict(X_train)
train_error = mean_absolute_error(training_predictions, y_train)
print("Training error: "+str(train_error))

test_error = mean_absolute_error(test_predictions, test_results)
print("Testing error: "+str(test_error))
```

ANEXO VII

```
# Código para implementar el algoritmo de regresión linear

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.metrics import mean_absolute_error
import numpy as np
import time
import os


print("Loading array...")
reduced_data = np.loadtxt('reduced_features.txt', dtype=np.float64)
print("Array loaded")
print(reduced_data)
print(str(len(reduced_data)))

reduced_test_data = np.loadtxt('test_features.txt', dtype=np.float64)

faces = np.random.choice([0,0], size=18123)
landscapes = np.random.choice([1,1], size=16877)
result = np.concatenate((landscapes, faces), axis=None)

test_faces = np.random.choice([0,0], size=1803)
test_landscapes = np.random.choice([1,1], size=1727)
test_results = np.concatenate((test_faces, test_landscapes), axis=None)

print("Result...")
print(result)
print("Size of result: "+str(len(result)))

print("Rescaling...")
ss = StandardScaler()
ss.fit(reduced_data)
reduced_data = ss.transform(reduced_data)
ss.fit(reduced_test_data)
reduced_test_data = ss.transform(reduced_test_data)

print("Training...")
X_train, X_test, y_train, y_test = train_test_split(reduced_data, np.array(result), test_size=0.25, random_state=42)
LR = LogisticRegression()
LR.fit(X_train, y_train)
y_pred = LR.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

```
cm = confusion_matrix(y_test, y_pred)
cmap = plt.cm.Blues
classes = np.array([0,1])
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
ax.set(xticks=np.arange(cm.shape[1]), yticks=np.arange(cm.shape[0]),
        xticklabels=classes, yticklabels=classes,
        title="Linear Regression Confusion matrix",
        ylabel='True label', xlabel='Predicted label')
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mode="anchor")
fmt = 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
   for j in range(cm.shape[1]):
      ax.text(j, i, format(cm[i, j], fmt), ha="center", va="center", color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
plt.savefig("Linear_regression_CM.jpg")

print('Model accuracy is: ', accuracy)
print("Confusion matrix, without normalization")
print(cm)

print("Testing...")
test_predictions = LR.predict(reduced_test_data)
test_accuracy = accuracy_score(test_results, test_predictions)
print('Testing accuracy is: ', test_accuracy)

print("Training error")
training_predictions = LR.predict(X_train)
train_error = mean_absolute_error(training_predictions, y_train)
print("Training error: "+str(train_error))

print("Testing error")
test_error = mean_absolute_error(test_predictions, test_results)
print("Testing error: "+str(test_error))
```

ANEXO VIII

```
# Código para reducer las dimensiones de los vectores de características
con el algoritmo PCA.

n = inceptionV3_feature_list_np.shape[0] # how many rows we have in the dataset
print("Number of rows in the dataset: "+str(n))
chunk_size = 1000 # how many rows we feed to IPCA at a time, the divisor of n
ipca = IncrementalPCA(n_components=100, batch_size=100)
reduced_data = None
print("N//chunck_size: "+str(n//chunk_size))

for i in range(0, 42):
    print("Fitting pca until "+ str((i+1))*chunk_size)
    if i == 0:
        reduced_data = ipca.fit_transform(inceptionV3_feature_list_np[i*chunk_size : (i+1)*chunk_size])
    elif i != 42:
        reduced_data = np.vstack((reduced_data, ipca.fit_transform(inceptionV3_feature_list_np[i*chunk_size :
(i+1)*chunk_size])))
    else:
        continue
    print("Reduced data until now: ")
    print(reduced_data)
    print(str(len(reduced_data)))
```

ORDEN DE EMPASTADO