LAS NOVEDADES EN EL RECONOCIMIENTO FACIAL

PRESENTADO POR

JUAN DAVID TRUJILLO SUÁREZ
20172161632

VISION ARTIFICIAL

INGENIERIA DE SOFTWARE

FACULTAD DE INGENIERIA

UNIVERSIDAD SURCOLOMBIANA

NEIVA-HUILA

2020-1

Contenido

1. RESUMEN:

El **reconocimiento facial** es una herramienta para brindar comodidad en diferentes espacios, como por ejemplo en teléfonos celulares donde ahora solo es necesario ubicarse frente a la cámara para desbloquearlo. En este artículo se propone encontrar y estudiar los algoritmos de **Vision Artificial** más apropiados y novedosos para el reconocimiento facial y aplicarlos en el sector de la seguridad. Se aplicaron técnicas de Vision Artificial como el Algoritmo en Cascada de Haar y el modelo **VGG16** para **redes neuronales convolucionales.**
Utilizando únicamente el **Algoritmo Clasificador de Haar** se obtuvo una precisión por debajo del 50%, pero, una vez implementado el modelo VGG16 se alcanzó una precisión por encima de la esperada con un valor de 97.67%.

PALABRAS CLAVES: Reconocimiento facial, Vision Artificial, VGG16, Redes Neuronales, Algoritmo Clasificador de Haar.

2. INTRODUCCION

En la actualidad, es muy común ver aplicativos con reconocimiento facial o reconocimiento de rostros, Facebook e Instagram aplican técnicas para reconocer rostros humanos. Desde hace algunos años los sistemas de seguridad han evolucionado hasta tal forma que hoy en día ya encontramos espacios a los que para acceder se deben recopilar datos biométricos o utilizar llaves electrónicas leídas por un chip para accionar un sistema de seguridad. Algunos ejemplos de seguridad por datos biométricos se pueden observar en los gimnasios, en los cuales se utiliza comúnmente la huella dactilar para acceder a estos, además, particularmente se ve que en los teléfonos celulares y algunas computadoras personales se puede apreciar desbloqueo por reconocimiento facial, el cual es realizado por una cámara infrarroja que capta las características biométricas del usuario para que una vez comparados en la base de datos se pueda utilizar el dispositivo electrónico.

En cuanto al tema a tratar en el siguiente escrito, es decir, el reconocimiento facial, se pueden observar sistemas que están siendo ya aplicados en ambientes diversos como en empresas, en las que requieren llevar control de la asistencia de sus trabajadores o en ambientes educativos, como en algunos países del continente asiático, donde ya están tomando la medida de reconocimiento facial para conocer si sus estudiantes asisten a clase y para mantener el control de acceso a esta, como en la Universidad Farmacéutica de China en la ciudad de Nankín, donde por medio del uso de este método se hace un control estricto de la asistencia a clases para que de esta forma sea posible lograr disminuir la inasistencia a estas.

Para realizar el reconocimiento facial se deben establecer puntos clave en la cara, los puntos que comúnmente se toman como rasgos característicos son: Los ojos, la nariz y la boca, de estos rasgos se extraen un grupo de pixeles que serán analizados por un algoritmo para determinar a qué persona corresponde cada rasgo. En el siguiente proyecto se estudiarán y aplicarán métodos de reconocimiento facial para realizar un sistema de entrada a un estamento educativo, el cual requiere mejorar el flujo de ingreso a este y optimizar su seguridad de ingreso.

3. DEFINICION DEL PROBLEMA:

La Universidad Surcolombiana cuenta con una gran afluencia de personas en su cede central ubicada en la ciudad de Neiva, esto representa una gran entrada y salida de estudiantes, profesores, administrativos y demás trabajadores de esta. Actualmente, esta utiliza como medio de acceso e identificación el uso del carné, este medio en algunas ocasiones no es bien utilizado por las personas que frecuentan el estamento educativo, ya que con frecuencia estas no portan dicho carné para ingresar y como consecuencia los guardas de seguridad encargados de registrar la entrada y salida de los usuarios dejan pasar libremente a cualquier individuo por el simple hecho de que estos requieren dirigirse a sus clases o labores cotidianas.

Otra problemática que presenta la universidad es que realmente no se conoce quien está presente en el momento dentro de esta.
Frente a esta situación es necesario plantearnos la incógnita que será resuelta a lo largo de este proyecto:

¿Qué medidas se pueden utilizar para garantizar un flujo ágil de personas en la entrada del establecimiento educativo?

4. JUSTIFICACION:

La presente investigación buscara resolver la problemática planteada en la pregunta anterior, ya que en los últimos años la Universidad Surcolombiana no ha mantenido un registro claro y conciso de quienes realmente se encuentran presentes en ella, y un buen orden para acceder y salir de esta. Por esta razón, en el siguiente escrito se propone entonces, estudiar y aplicar los diferentes algoritmos de vision artificial para lograr de algún modo identificar a cada individuo a la hora de pasar por la entrada del establecimiento, haciendo uso de técnicas ya previamente aplicadas como el algoritmo **Haar Cascade**, el cual utiliza **clasificadores de Haar** para cumplir su función, se ha decidido utilizar este método en conjunto con el modelo **VGG16** para redes neuronales ya que cuenta con un porcentaje de precisión cercano al 90%. Además de estas técnicas se hará uso de librerías como **OpenCV y Keras**, las cuales ayudaran en las tareas de "**Vision Artificial**" requeridas para resolver la incógnita planteada y por último debemos definir que el lenguaje que será utilizado para este proyecto será Python, el cual nos resulta útil a la hora de trabajar con OpenCV y algoritmos de **Deep learning**.

5. OBJETIVOS:

5.1 OBJETIVO GENERAL.

Aplicar tecnologías de Vision Artificial que garanticen la identificación facial para ingresar a la Universidad Surcolombiana.

5.2 OBJETIVOS ESPECIFICOS.

- Utilizar el reconocimiento facial para reemplazar la identificación por medio del uso del carné.

- Determinar los algoritmos de Deep Learning y Vision artificial más efectivos para crear un sistema que agilice la entrada a la universidad.

- Lograr identificar a todos aquellos que requieran entrar a la universidad con una precisión superior al 80%.

- Lograr la portabilidad del proyecto a otros espacios los cuales necesiten una solución similar a la planteada.

6. ESTADO DEL ARTE:

A. [1] Preeti Singh y Mukesh Tripathi (2013) abarcaron el problema de la detección facial en imágenes con fondos simples y complejos, las cuales con el uso del **clasificador haar cascade** lograron realizar tareas de reconocimiento facial. Se establece dicho algoritmo como uno de los más rápidos y seguros para tareas de **Vision Artificial**, ya que se basa en reconocer **características de Haar**, estas características se empiezan analizando en la imagen desde la esquina superior izquierda hasta la esquina inferior derecha. Los experimentos se hicieron con una base de datos de Rostros Indios (IFD por sus siglas en ingles) y con un **dataset** de Caltech. Todas las imágenes presentes fueron de la parte frontal del rostro. Además de esto, utilizaron OpenCV para implementar el clasificador. Como resultados obtuvieron un 100% de precisión con la base de datos de rostros Indios y 93.24% de precisión con el **dataset** de Caltech.

B. [2] J. G. RoshanTharanga, S. M. S. C. Samarakoon, T. A. P. Karunarathne 2, K. L. P. M. Liyanage, M. P. A. W. Gamage, D. Perera (2013) plantearon en este proyecto una solución a una empresa, la cual requería conocer y mantener un registro organizado de la asistencia al trabajo, todo esto fue posible mediante el uso de **Reconocimiento Facial** en tiempo real. Para esto fue implementado el algoritmo para detectar rasgos faciales conocido como **Haar Cascade** al ser uno de los más simples y rápidos para realizar tareas de reconocimiento facial manteniendo su alta precisión a pesar de su rapidez de cálculo. Para este sistema se basaron en computación en la web, ya que de esta forma se estarían descargando y subiendo datos óptimamente a la base de datos, el proceso que realizaba la aplicación resumidamente consistía en lo siguiente: Se captaba el fotograma a través de la cámara, este fotograma debía ser analizado para detectar las caras inicialmente, para luego esta ser comparada en tiempo real con los datos ya entrenados almacenados en la nube, los cuales se encontraban guardados en una base de datos, la cual para ser ingresada se ejecutaban sentencias SQL, para luego comparar cada imagen e identificar al empleado determinado, finalmente se devolvía un estado de asistencia en la pantalla y se mostraban los detalles del empleado en ella.

C. [3] Suma S L (2018) implemento un algoritmo de reconocimiento facial en tiempo real utilizando un Histograma de patrón binario lineal y el algoritmo de Viola Jones. Este método consistía en la fusión y reconocimiento de la comunicación. La extracción de características faciales mediante la **técnica LBPH** y clasificadores euclidianos utilizados para la detección facial. Este trabajo obtuvo un rango de precisión de entre 85% a 95%, este dependiendo de la luminosidad, casos en los que se presentan gemelos, personas con vello facial y/o personas utilizando anteojos.

D. [4] Souhail Guennouni implementó en el 2017 un sistema de detección de rostros agrupando métodos como "**Haar Cascade Classifier**" y emparejamiento de bordes. El algoritmo de emparejamiento de bordes y la selección de características similares a las de Haar, combinadas con clasificadores de Haar son las dos técnicas utilizadas en este sistema. Este algoritmo produce un mejor emparejamiento, pero la velocidad de detección es comparativamente menor.

E. [5]V. Mohanraj, S. Sibí, y V. Viadehi (2019) plantearon un reconocimiento facial basado en un conjunto de redes neuronales convolucionales (ECNN). El modelo propuesto aborda los desafíos de la expresión facial, el envejecimiento, la baja resolución y las variaciones de pose. El modelo ECNN propuesto supera a los modelos de vanguardia existentes, como los modelos **Inception-v3, VGG16, VGG19, Xception y ResNet50 CNN** con una precisión de rango 5 de 97.12% en el conjunto de datos de Web Face y 100% en el conjunto de datos de cara de YouTube.

F. [6] Puspita Majumdar, Akshay Agarwal, Richa Singh y Mayank Vatsa (2019) realizaron la investigación que propone un ataque parcial de manipulación de la cara, donde las regiones faciales se reemplazan o transforman para generar muestras manipuladas. Experimentos de verificación de rostros realizados con dos sistemas de estos de última generación, **VGG-Face y OpenFace** en el conjunto de datos **CMU-MultiPIE** indican la vulnerabilidad de estos sistemas ante el ataque. Además, se propone una red de detección de manipulación de cara parcial (PFTD) para la detección del ataque propuesto. La red captura las inconsistencias entre las imágenes originales y manipuladas combinando la información en bruto y de alta frecuencia de las imágenes de entrada para la detección de imágenes manipuladas. La red propuesta supera el rendimiento de las redes neuronales profundas de línea de base existentes para la detección de imágenes manipuladas.

G. [7] S. Mittal y S. Mittal (2019) exploran una solución basada en el aprendizaje profundo para la detección automática de género a partir de imágenes faciales de un conjunto de datos bien equilibrado. La solución implica un marco de aprendizaje de transferencia donde el conocimiento se reutiliza a partir de un modelo de aprendizaje profundo que funcionó óptimamente en la tarea de clasificación de otro dominio. Se investiga la reutilización de Visual Geometry Group-16 (VGG16), un modelo de red neuronal convolucional (CNN) que se entrena previamente en un gran conjunto de datos de imágenes naturales. El ajuste fino de un segmento de la **CNN** en el conjunto de datos de tamaño moderado produce un mejor rendimiento que el obtenido de los enfoques de vanguardia cuando se implementa en el conjunto de datos público LFW-Gender.

H. [8] Moghekar, Rajeshwar; Ahuja, Saching (2019) utilizan un subconjunto de la base de datos Indian Movies Face (IMFDB) que tiene una colección de imágenes de caras recuperadas de películas / videos de actores que varían en términos de desenfoque, pose, ruido e iluminación. Ellos se centran en el uso de modelos de aprendizaje profundo previamente entrenados y aplican el aprendizaje por transferencia a las características extraídas de las capas CNN. Los resultados muestran que la precisión es de 99.89 usando CNN como extractor de funciones y 96.3 cuando se ajusta el **VGG-Face**.

7. MARCO TEORICO:

En el proyecto fueron utilizados algoritmos de **vision artificial** para poder trabajar **detección de objetos**, específicamente detección y clasificación de rostros humanos. Para esto fue necesario implementar el **Algoritmo Clasificador de Haar** y el modelo **VGG16** para **Redes Neuronales Convolucionales**, estos serán explicados a continuación:

7.1 Vision Artificial: Según la Automated Imaging Association, la vision artificial son todas aquellas aplicaciones en las que el hardware y software en conjunto brinda operatividad a un sistema, el cual es capacitado para el procesamiento y análisis de imágenes.

7.2 Detección de Objetos: Es un método responsable de descubrir e identificar la existencia de un objeto de una clase determinada. Esta tarea puede ser realizada por diferentes algoritmos clasificadores, como arboles de decisión o clasificadores en cascada [9].

7.3 Algoritmo Clasificador de Haar: Es un algoritmo propuesto por Viola y Jones (2001), el cual se basa en encontrar rasgos específicos del rostro humano. Una vez encontrados estos, el algoritmo permite que el candidato a rostro avance a la siguiente etapa de detección. Un **candidato** es una sección rectangular de la imagen original llamada sub-ventana. El algoritmo explora toda la imagen con esta ventana y denota a cada sección un candidato a cara. El algoritmo utiliza una **imagen integral** para procesar los **rasgos de Haar** constantemente. Ese hace uso de una cascada de etapas que consiste en muchos rasgos diferentes de Haar, siendo estos clasificados por el algoritmo clasificador. Estos clasificadores generan una salida que será proporcionada al comparador de cada etapa. Este suma las salidas de los clasificadores y debe superar el umbral de la etapa para avanzar a la siguiente, una vez pasadas todas estas, se concluye que dicho candidato es finalmente un rostro humano.

7.3.1 Imagen Integral: La imagen integral se define como la suma de los pixeles de la imagen original**.** El valor en cualquier ubicación (x, y) de la imagen integral es la suma de los pixeles de la imagen original por encima y a la izquierda de la ubicación (x, y). La generación de la imagen integral es ilustrada en la **figura 1**. Las **características** rectangulares de una imagen se calculan utilizando la representación intermedia de una imagen, llamada imagen integral. Está siendo una matriz que contiene las sumas de los valores de intensidad de los pixeles en la izquierda y encima del píxel de la ubicación (x, y). Así que si A (x, y) en la imagen integral se define como AI [x, y] y se calcula como se muestra en la ecuación 1 y se ilustra en la **figura 2**.

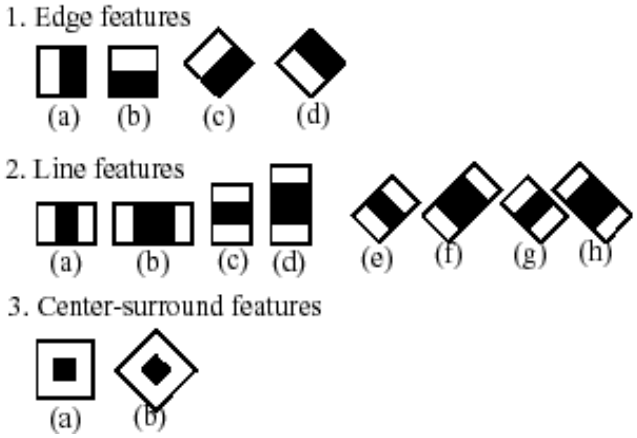

Figura 1. Rasgos a través de imágenes integrales.

Fuente: (OpenCV Dev Team, 2014).

$$AI(x,y) = \sum_{x' \le x, y' \le y} A\ (x', y')$$

Ecuación 1

Los **rasgos** girados a 45° como el **rasgo 1(b)**, requieren otra representación llamada imagen rotativa o auxiliar de suma rotativa. La integral rotada de la imagen original se calcula encontrando la suma de la intensidad de los pixeles que se encuentran en el ángulo mencionado a la izquierda y por encima para el valor x, y por debajo para el valor y. Por lo tanto, A (x, y) es la imagen original y AR [x, y] es la imagen integral rotativa, y se calcula como se encuentra representado en la **ecuación 2** e ilustrada en la **figura 2(b)**.

$$AR(x,y) = \sum A\ (x', y')$$
$$x' \leq x\ , x' \leq x - |y - y'|$$

Ecuación 2.

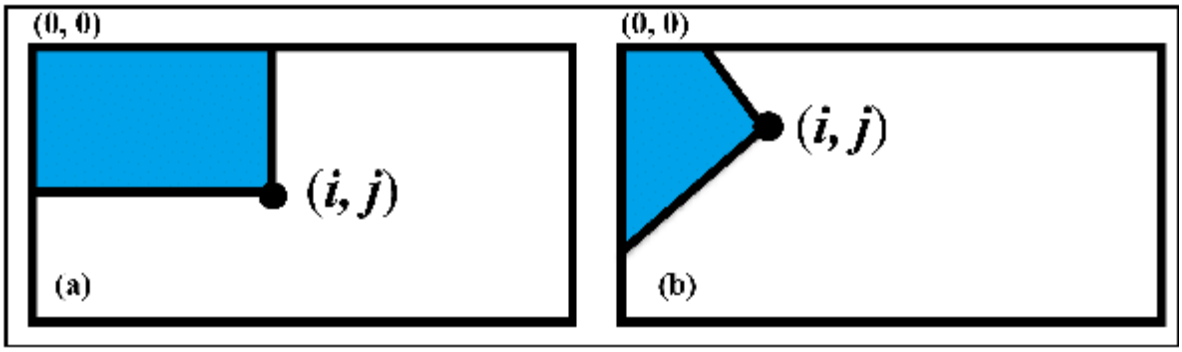

Figura 2. Ejemplo de imagen integral rotativa

Fuente: (Sarker, Md. Mostafa Kamal & Song, Moon. 2014).

Sólo se necesitan dos pasadas para calcular ambos conjuntos de imágenes integrales, una para cada conjunto. Usando la integral apropiada y tomando la diferencia entre seis u ocho elementos de la matriz que forman dos o tres rectángulos conectados, se puede calcular una característica de cualquier escala. Así, el cálculo de una característica es extremadamente rápido y eficiente.

7.3.2 Rasgos Haar: Los **rasgos de haar** se componen de dos o tres rectángulos. Cada candidato es escaneado y analizado en cada etapa. El peso y el tamaño de cada rasgo y estos mismos se generan utilizando un algoritmo de Machine Learning de AdaBoost [10]. Los pesos son constantes generados por el algoritmo de aprendizaje. Hay una variedad de formas características como se ve en la **figura 3.** Cada **característica de Haar** tiene un valor que se calcula tomando el área de cada rectángulo y multiplicando cada uno por su propio peso, luego se deben sumar los resultados. El área de cada rectángulo se encuentra fácilmente utilizando la imagen integral. La coordenada de cualquiera de las esquinas de un rectángulo puede ser usada para obtener la suma de todos los pixeles de arriba y de la izquierda, el área puede ser calculada rápidamente como se aprecia en la **figura 4.** Dado que L se resta dos veces, hay que volver a sumarla para obtener el área correcta del rectángulo. El área del rectángulo R, denotada como la integral del rectángulo, puede ser calculada de la siguiente manera usando las ubicaciones de la imagen integral: D – C – B + A.

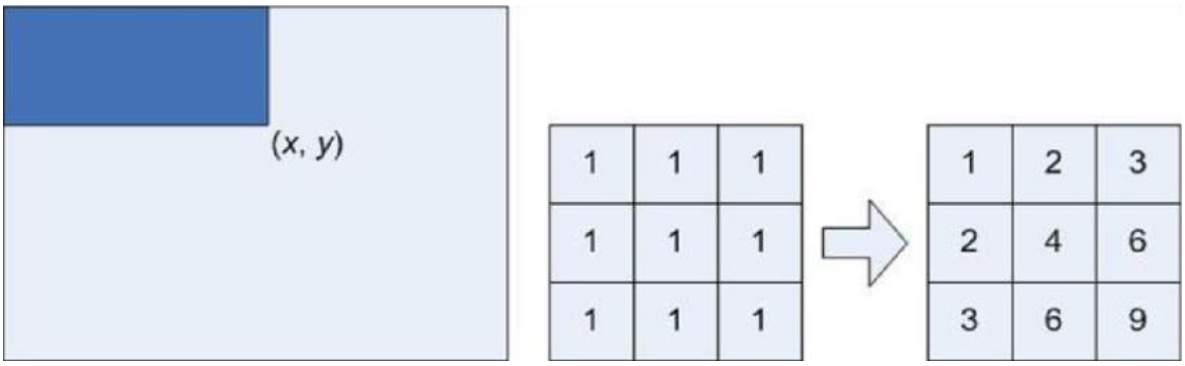

Figura 3. Valores rasgos haar.

Fuente: Kadbe, Premanand. (2015).

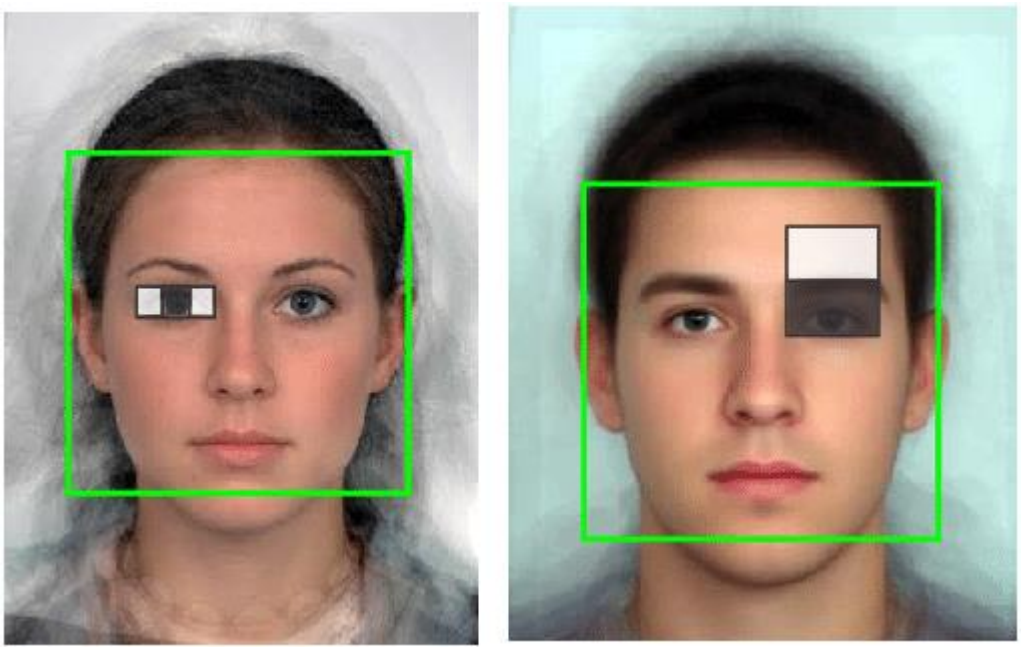

Figura 4. Ejemplo de rasgos haar.

Fuente: (Rezaei, Mahdi, 2016).

7.3.3 Clasificador de Características de Haar: El clasificador de rasgos de Haar utiliza la integral del rectángulo para calcular el valor de un rasgo. Este multiplica el peso de cada rectángulo por su área y sus resultados se suman. Varios clasificadores componen un escenario. Un comparador de etapas suma todos los resultados dados por el clasificador en una etapa y compara este resumen con un umbral dado en cada etapa. Este es constante y se obtiene del algoritmo AdaBoost. Cada etapa no tiene un número determinado de rasgos de Haar. Dependiendo de los parámetros de los datos de entrenamiento, las etapas individuales pueden tener un numero variable de características de Haar, como se muestra en la **figura 5**.

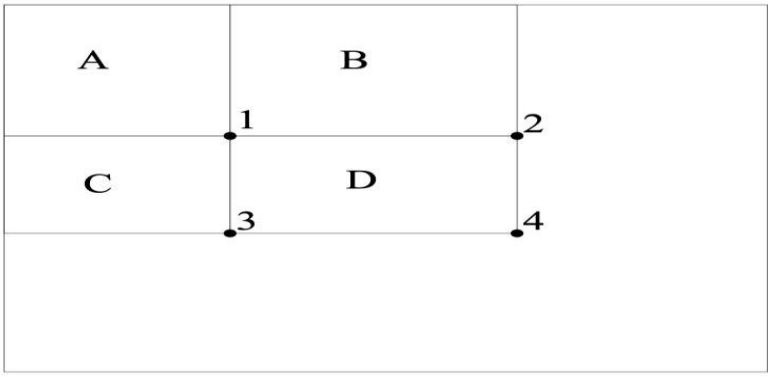

Figura 5. Sectorización de imagen.

Fuente: (Zhang, Zhengyou. 2010).

7.3.4 Clasificador en Cascada: El algoritmo de detección de rostros de Viola y Jones elimina rápidamente a los candidatos a rostro usando una cascada de etapas. La cascada elimina a los candidatos estableciendo requisitos más estrictos por cada fase, siendo las posteriores mucho más difíciles de superar. Estos candidatos salen de la cascada si pasan todas las etapas o si fallan en alguna. Se detecta un rostro si un candidato pasa todas las etapas satisfactoriamente. Este proceso se muestra en la **figura 6**.

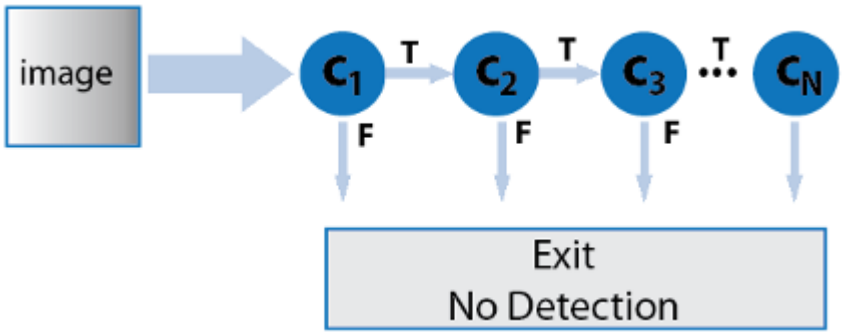

Figura 6. Representación clasificación en cascada.

Fuente: (MathWorks, 2020).

7.4 **Artificial Neural Networks:** Son sistemas computacionales de procesamiento, los cuales son inspirados en los sistemas nerviosos biológicos. Se encuentran compuestos por un alto número de nodos interconectados, entre los cuales el trabajo se entrelaza de manera distribuida para aprender colectivamente de la entrada para optimizar su salida final [11]. Estos consisten en tres capas (Capa de entrada, capa escondida y capa de salida), cada una de estas están hechas de unidades de procesamiento adaptativas, las cuales están interconectadas.

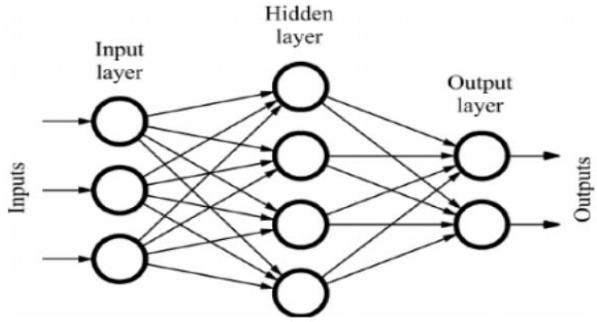

Figura 7: Representa un ejemplo de un modelo de red neuronal

Fuente: (Databricks 2020).

7.5 **Convolutional Neural Network (CNN):** Es un algoritmo de aprendizaje profundo que puede reconocer y clasificar características en imágenes para la vision artificial. Cuenta con capas para rectificar el estado de la imagen y normalizar los datos [12]. En este caso fue utilizada la función VGG16 al ser una red neuronal óptima para trabajar el reconocimiento facial.

7.6 **VGG16:** Es una arquitectura de red neuronal convolucional de 16 capas creada por K. Simonyan y A. Zisserman (2014). Sus capas consisten en capas convolucionales, capas de agrupación máxima, capas de activación y capas completamente conectadas. En la **figura 8** se puede apreciar el modelo de esta arquitectura [13]. Este algoritmo debe contar con una función de activación, función de perdida y una función de optimización, en este proyecto fue utilizada la función **Softmax** para la activación, la función de **Crossentropía Categórica** para la perdida y la función de **Adam** para la optimización respectiva.

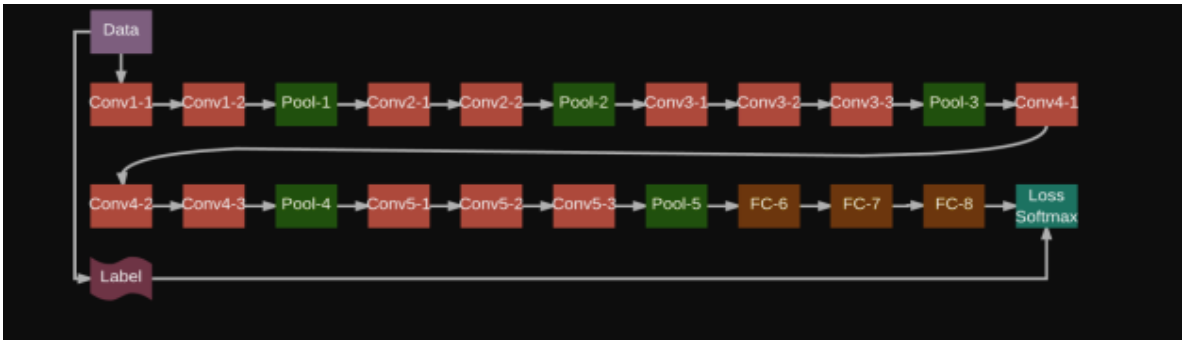

Figura 8. Capas de la red VGG-16

Fuente: (H. Qassim, A. Verma and D. Feinzimer 2018)

7.6.1 Softmax: La función Softmax es otro tipo de función de activación usada en la computación neural, la cual fue utilizada en este proyecto. Se utiliza para calcular la distribución de probabilidad de un vector de números reales. La función Softmax produce una salida que es un rango de valores entre 0 y 1, con la suma de las probabilidades siendo igual a 1. La función Softmax se calcula usando la siguiente relación [14].

$$f(x_i) = \frac{\exp(x_i)}{\sum \exp(xj)} - (1.12)$$

7.6.2 Categorical Crossentropy Loss: La función de perdida utilizada en el sistema fue la crossentropía categórica, esta es una función que se utiliza para la categorización de una sola etiqueta. Esta sólo se aplica a una categoría para cada punto de datos. En otras palabras, un ejemplo puede pertenecer a una sola clase. Matemáticamente es representada de la siguiente manera:

$$L(y,\hat{y}) = -\sum_{j=0}^{M}\sum_{i=0}^{N}(y_{ij} * log(\hat{y}_{ij}))$$

Donde ŷ es el valor predictivo, este comparará la distribución de las predicciones (las activaciones en la capa de salida, una para cada clase) con la distribución real, donde la probabilidad de la clase real se establece en 1 y 0 para las otras clases. Dicho de otra manera, la clase verdadera se representa como un vector codificado de un punto, y cuanto más cerca estén las salidas del modelo de ese vector, menor será la pérdida.

7.6.3 Adam Algorithm: El algoritmo de optimización usado en el proyecto fue el algoritmo de Adam. Este puede utilizarse en lugar del clásico procedimiento de descenso de gradiente estocástico, su función es actualizar los pesajes de la red de neuronal de forma iterativa en base a los datos de entrenamiento. Este algoritmo fue presentado por Diederik Kingma de OpenAI y Jimmy Ba de la Universidad de Toronto en su documento de la ICLR de 2015 titulado “Adam: un método para la optimización estocástica” [15].

8 MATERIALES Y METODOS:

El presente proyecto fue realizado en la ciudad de Neiva, en un computador personal con las siguientes características:

- Motherboard: Asus Tuf x470-Plus Gaming
- Procesador: Ryzen 5 3600, 6 nucleos – 12 hilos.
- Ram: 16 Gb ram a 3000Mhz
- Tarjeta Gráfica: Nvidia rtx2060 super – 8Gb vram.
- Almacenamiento: 1Tb HDD.

El proceso de realización se dividió en 4 fases principales: Preparación del ambiente de desarrollo, Obtención y organización del dataset, Entrenamiento y análisis de los modelos y por último el despliegue. Cada una de estas fases será explicada a continuación detalladamente.

8.1 Preparación del ambiente de desarrollo:

Para preparar el ambiente de desarrollo se instaló Python 3.8.2 en el sistema operativo Windows 10, en conjunto con Anaconda 4.8.3. En cuanto a los paquetes y librerías se utilizaron las siguientes: OpenCV, Numpy, Keras, Matplotlib, y Pillow. Las versiones se adjuntan en la **figura 9.**

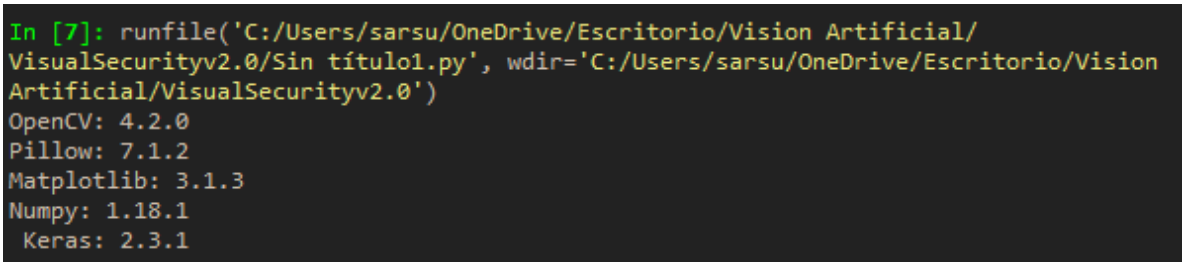

Figura 9. Librerías utilizadas.

8.2 Obtención y organización del dataset:

Para realizar el dataset se utilizó un modelo por defecto de OpenCV para detectar la parte frontal del rostro, se definió la función "fase_ext" la cual recibe la "img" (Imagen), donde se utiliza la función de Haar para detectar el multi escalado de la imagen con un factor de escala de 1.3 y un mínimo de vecindarios de 5. En caso de que con este método se encontrara un rostro se iniciaría un ciclo en el cual se extraen las coordenadas en x e y, junto con la altura y anchura definidos como h y w **(figura 10)**.

```python
import cv2
import numpy as np


#Importing HAAR CASCADE CLASSIFIER!

haar = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_default.xml')

#Creating functions to extract images of each frame of the video

def face_ext(img):

    #img, scale factor = 1.3, min neighbours = 5
    faces = haar.detectMultiScale(img, 1.3, 5)


    if faces is ():
        return None

    #Cropping al faces
    # X,Y coordinates Width and Height
    for (x,y,w,h) in faces:
        x=x-10
        y=y-10
        #Cropped face frame is between y,y+height+50 and x,x+width+50
        cropped_face = img[y:y+h+50, x:x+w+50]
```

Figura 10. Creando las funciones de extracción facial.

Luego se definió un ciclo en el cual en caso de que encontrara un rostro se empezaría a extraer imágenes de este para luego hacerles un escalado a 400x400, finalmente estas fueron guardadas en la dirección “Images”. Una vez se obtuvieran las 400 imágenes o se presionara la tecla “q” se detendría la recolección de estas **(figura 11).**

```python
#start recording

cap = cv2.VideoCapture(0)
count = 0

#Collecting 400 samples of each person

while True:

    ret, frame = cap.read()
    #If a face is detected
    if face_ext(frame) is not None:
        count += 1
        #Each image is resized to 400x400
        face = cv2.resize(face_ext(frame), (400, 400))

        #Saving file in the path
        file_name_path = 'Images/' + str(count) + '.jpg'
        cv2.imwrite(file_name_path, face)

        #Putting count on images and displaying live count

        cv2.putText(face, str(count), (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2)
        cv2.imshow('Face Cropper', face)

    else:
        print("Face not found")
        pass

    if cv2.waitKey(1) & 0xFF == ord('q') or count == 400:
        break

cap.release()
cv2.destroyAllWindows()
print("Collecting Samples Complete!")
```

Figura 11. Recopilando las imágenes para el dataset.

Este proceso se aplicó repetidas veces hasta obtener 2550 imágenes para nuestro dataset. Este fue dividido en 2 carpetas, la primera para entrenamiento teniendo el 80% de las imágenes y la segunda para testeo con las imágenes restantes, finalmente en cada carpeta se encuentran 3 subcarpetas con los respectivos nombres de cada persona escaneada. Se utilizo esta distribución de tal forma que durante el entrenamiento y predicción se pudiera imprimir el nombre de cada uno **(figura 12).** En este caso no interesa el nombre de cada imagen, solo interesa que en las carpetas de entrenamiento se encuentren las imágenes correspondientes de cada persona **(figura 13).**

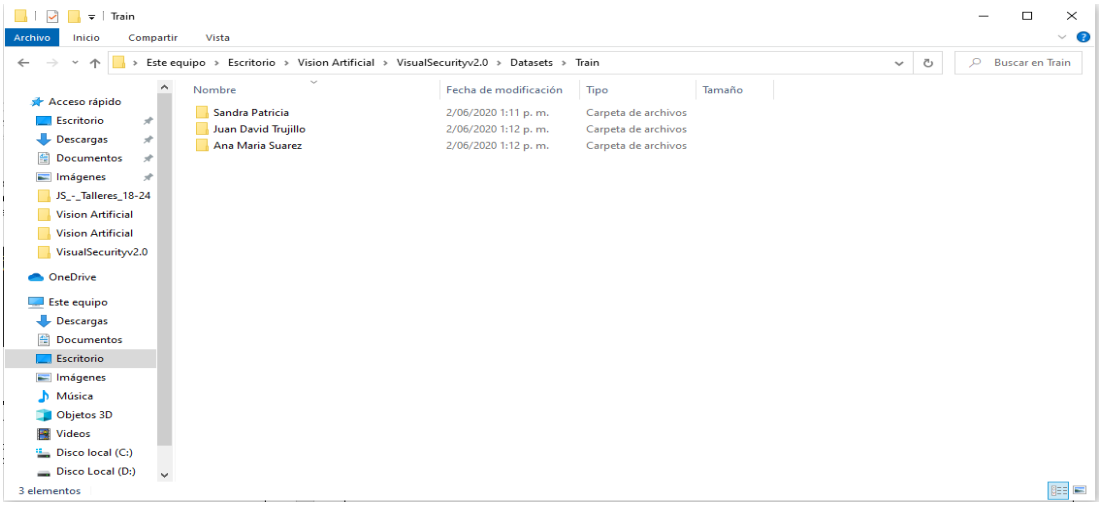

Figura 12. Organización de las carpetas de entrenamiento.

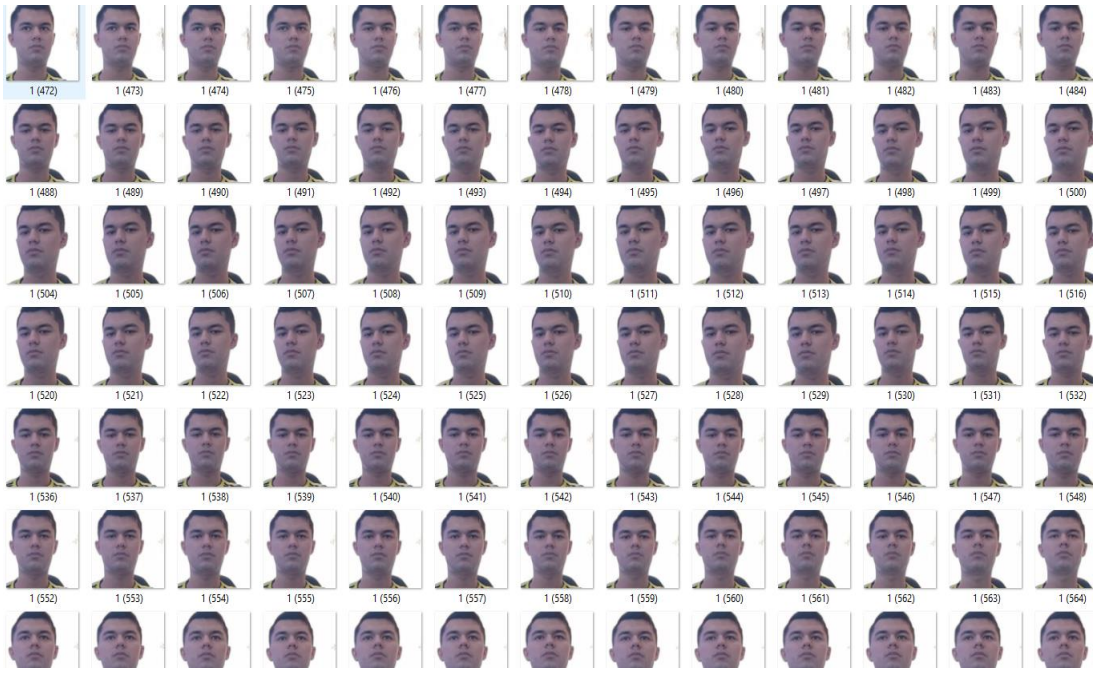

Figura 13. Organización del dataset

8.3 Entrenamiento y análisis de los modelos:

Para la realización del entrenamiento inicialmente se utilizó el algoritmo **Haar Cascade** como clasificador para los rostros, pero frente a la baja precisión que se consiguió con este, se decidió aplicar redes neuronales de 16 capas para apoyar a la clasificación.

Inicialmente se importaron todas las librerías requeridas, luego se realizó un escalado a las imágenes a 244x244, se extraen las direcciones de las carpetas de entrenamiento

y test del dataset y se añaden las capas de preprocesamiento a la red neuronal convolucional, en este caso se utilizaron 3 canales y las medidas de "imagenet" por defecto. **(Figura 14)**

```
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt

# Re-sizing all the images to 224x244
IMAGE_SIZE = [224, 224]

#Taking the path of the images
train_path = 'Datasets/Train'
valid_path = 'Datasets/Test'

# Adding preprocessing layer to the front of VGG using 3 chanels and imagenet wieghts.
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

Figura 14. Importando las librerías y extrayendo las imágenes.

Luego se extraen las clases (subcarpetas) del dataset, se establece la función de activación, en este caso se usó la función Softmax y se imprime la estructura del modelo. Finalmente se definen los algoritmos de perdida y optimización, estos fueron el algoritmo de Crossentropía Categórica y el algoritmo de Adam respectivamente (**figura 15**).

```
# Trainning only unexisting weights
for layer in vgg.layers:
  layer.trainable = False



  # Getting the # of classes
folders = glob('Datasets/Train/*')


# Defining activation function
x = Flatten()(vgg.output)
prediction = Dense(len(folders), activation='softmax')(x)

# Creating the model output
model = Model(inputs=vgg.input, outputs=prediction)

# Displaying the model structure
model.summary()

# Telling the model what loss and optimization algorithms to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

Figura 15. Definiendo la capa de salida y sus algoritmos.

Ahora se define el generador de imágenes de entrenamiento, se define el tamaño de cada imagen a entrenar de 244x244 y el tamaño de muestras por ciclo (batch size) de 32 (**figura 16**).

```
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('Datasets/Train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory('Datasets/Test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')
```

Figura 16. Extrayendo las imágenes.

Para finalizar se definen los parámetros de entrenamiento, en este caso se definió que los epochs (ciclos por entrenamiento) el cual se iniciaba con un valor de 10 epochs y se finalizó con un valor de 30 epochs (aumentando en 10 cada entrenamiento) es decir el primer entrenamiento se realizó con 10 epochs, el segundo 20 y el tercero 30 respectivamente. Posteriormente se imprimen los valores de perdida y precisión por cada entrenamiento y por último se guarda el modelo como "facefeatures_new_model.h5" (**figura 17**).

```
# fitting the model
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs = 30,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
# Getting the loss Values
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')

# Getting the accuracy values
plt.plot(r.history['accuracy'], label='train accuracy')
plt.plot(r.history['val_accuracy'], label='val accuracy')
plt.legend()
plt.show()
plt.savefig('AccVal_accuracy')

import tensorflow as tf

from keras.models import load_model

model.save('facefeatures_new_model.h5')
```

Figura 17. Entrenamiento del modelo.

8.4 Despliegue:

Una vez realizado el entrenamiento y guardado el modelo se importa en nuestro sistema de reconocimiento facial. Adicionalmente se extraen los nombres de los directorios de cada persona y se carga el modelo de clasificador en cascada para identificar rostros y el modelo de red neuronal para clasificar y predecir quien es cada persona (**figura 18**).

```
import os
from PIL import Image
from keras.applications.vgg16 import preprocess_input
import base64
from io import BytesIO
import json
import random
import cv2
from keras.models import load_model
import numpy as np

from keras.preprocessing import image


people_dir = 'Datasets/Test'
name_list = []
for person_name in os.listdir(people_dir):
    name_list.append(person_name)

print(name_list)

model = load_model('facefeatures_new_model.h5')

# Loading the cascades
face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_default.xml')
```

Figura 18. Importando el modelo y las librerías.

A continuación, se define una función que recibirá la imagen (fotogramas del video) y se utiliza el método para detectar los rostros humanos utilizando los mismos parámetros que se usaron para recopilar el dataset, en caso de que ningún rostro sea detectado no retornaría dato alguno y en caso de que se detecte rostro se obtendría la

posición y medidas del rostro detectado, esto para almacenarlas en una variable (**figura 19**).

```
def face_extractor(img):
    # Function detects faces and returns the cropped face
    # If no face is detected, it returns the input image

    #gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(img, 1.3, 5)

    if faces is ():
        return None

    # Crop all faces found
    for (x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,255),2)
        cropped_face = img[y:y+h, x:x+w]

    return cropped_face

# Doing some Face Recognition with the webcam
video_capture = cv2.VideoCapture(0)
num= 0
```

Figura 19. Extrayendo las coordenadas de la cara.

En el siguiente proceso se empieza a utilizar la cámara dentro del bucle, se empieza a capturar la cámara y se extrae el rostro del fotograma, en caso de que se detecte un rostro se extraen sus rasgos (posición de cada rasgo y ancho, alto de este) se hace un escalado a 244x244 del rostro encontrado y se pasa a formato de arreglo en RGB. Ya que nuestro modelo de redes neuronales fue en 4D, es decir, con 4 parámetros (La imagen, altura, ancho y los canales) se debe cambiar la dimensión de 128x128x3 a 1x128x128x3 donde el valor 1 equivale a la cantidad de imágenes que va a procesar el sistema. Finalmente se almacena la predicción en una variable que será utilizada a continuación (**figura 20**)

```
while True:

    _, frame = video_capture.read()
    #canvas = detect(gray, frame)
    #image, face =face_detector(frame)

    face=face_extractor(frame)

    if type(face) is np.ndarray:
        face = cv2.resize(face, (224, 224))
        im = Image.fromarray(face, 'RGB')
           #Resizing into 224x244 because we trained the model with this exact image size.
        img_array = np.array(im)
                    #Our keras model used a 4D tensor with 4 parameters, (images x height x width x channels)
                    #So we must change the dimension from 128x128x3 into 1x128x128x3
        img_array = np.expand_dims(img_array, axis=0)
        pred = model.predict(img_array)
```

Figura 20. Codificación de la predicción.

Para finalizar, si la predicción para el usuario x (Donde estarían organizados de 0 a n según el orden de las carpetas) es mayor al 90% se imprime en el video, el nombre y porcentaje de predicción con la cual se detectó la persona. En caso de que no se

detecte a la persona se imprimirá un mensaje declarando que no se encuentra la persona en el dataset. Y si por el contrario no se encuentra un rostro se enviará un mensaje que declare esto. Si se oprime la tecla q el bucle while se terminará y en este caso se cerrará el programa (**figura 21**).

```
        #IF the accuracy of the prediction is higher than 90% we print the person matched name
        if(pred[0][num]>0.9):
                value = (pred[0][num] * 100)
                value = int(value)
                cv2.putText(frame,name_list[num] + " " + str(value) + "%", (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0,255,0), 2)
                print(name_list[num] + str(num))

            #if the person isn't in the trainned dataset it will print "None Matching"
        else:
            cv2.putText(frame,"None Matching", (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (0,0,255), 2)
            #If no face is found it will print "No face found"
    else:
        cv2.putText(frame,"No face found", (50, 50), cv2.FONT_HERSHEY_COMPLEX, 1, (255,0,0), 2)
    cv2.imshow('Video', frame)
    #If we press q it will exit the program.
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
video_capture.release()
cv2.destroyAllWindows()
```

Figura 21. Imprimiendo el porcentaje de predicción.

9 RESULTADOS Y DISCUSIONES:

Inicialmente aplicando el Algoritmo Clasificador de Haar se obtuvo una precisión cercana al 50% frente a la precisión alcanzada por P. Singh y M. Tripathi (2013) de 93.24%. Sin embargo, una vez implementadas las redes neuronales se incrementó la precisión al 81% con una intensidad de entrenamiento de 10 epochs (**tabla1**) esta se encuentra por debajo de la precisión obtenida por A. Elmahmudi y H. Ugail (2019). La precisión del modelo aumento significativamente aumentando de 10 a 20 epochs con un valor de 94.42% y una pérdida de 0.0022 (**tabla 2**). Finalmente, con 30 epochs se alcanzó el tope de precisión con un 97.67% y una pérdida de 0.0009 (**tabla 3**) siendo este modelo planteado superior al modelo planteado por H. Chen y C. Haoyu (2019) quienes obtuvieron una precisión de 91% siendo su precisión más alta. En todos los casos se utilizó la misma distribución del dataset y el mismo tamaño de imágenes.

Tabla 1. Utilizando VGG16, aumentando la precisión del sistema.

TEST 1	
Data Size	224x224
Color Dimension Dataset	RGB
Epochs	10
Train Size	1950
Test Size	600
Accuracy	81.48%
Loss	0.0096

Tabla 2. Cambios realizados en el modelo (20 epochs).

TEST 2	
Data Size	224x224
Color Dimension Dataset	RGB
Epochs	20
Train Size	1950
Test Size	600
Accuracy	94.42%
Loss	0.0022

Tabla 3. Cambios realizados en el modelo (30 epochs).

TEST 3	
Data Size	224x224
Color Dimension Dataset	RGB
Epochs	30
Train Size	1950
Test Size	600
Accuracy	97.67%
Loss	0.0009

Con el sistema una vez desplegado se obtuvo una precisión de 92% al detectar rostros humanos y 90% en clasificación. En la mayoría de los casos el sistema podía reconocer e imprimir el nombre de la persona detectada como se observa en la **figura 22**.

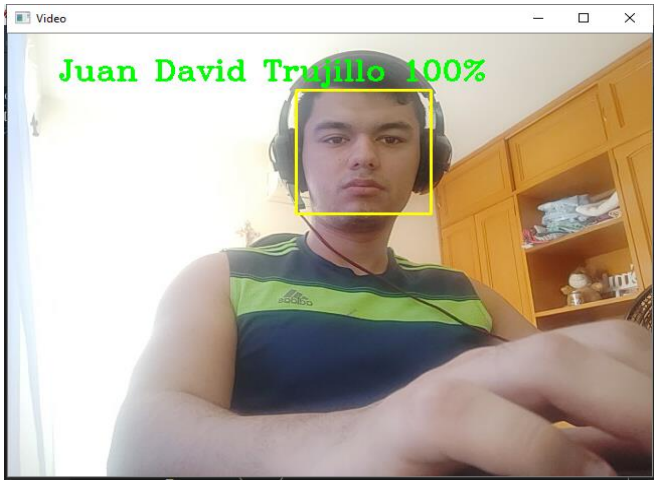

Figura 21. Imprimiendo el porcentaje de predicción.

10 CONCLUSIONES Y FUTUROS TRABAJOS:

En este trabajo se logró resolver la incógnita planteada, utilizando métodos de Vision Artificial se pudo garantizar una agilidad y precisión a la hora de detectar y clasificar rostros humanos. Pudimos observar que utilizando algoritmos simples como el algoritmo clasificador de Haar Cascade no se logra la suficiente estabilidad y precisión por la posible falta de capas de entrenamiento. Sin embargo, aplicando Redes Neuronales Convolucionales con el modelo VGG16 para estas, se puede apreciar que la precisión del sistema es casi perfecta y su velocidad de detección es casi instantánea (milésimas de segundos).

Se garantizo la precisión mayor al 80% planteada en los objetivos, se encontraron los algoritmos más fiables para el reconocimiento facial y muy posiblemente si es implementado este mismo modelo planteado se podrá aplicar a entornos similares como a la detección de emociones o de razas de animales.

No obstante, ningún sistema es perfecto y se han encontrado algunos problemas en el modelo planteado. Algunos de ellos son los siguientes:

- No es posible detectar los rostros si la persona tiene anteojos.
- Fallos al detectar personas de perfil.
- El sistema lograba identificar rostros humanos en imágenes digitales.

Frente a esto se podrían plantear posibles mejoras en trabajos futuros donde se pueda aplicar algoritmos que puedan reconocer y clasificar personas de perfil, con anteojos y que resulte imposible que una persona falsifique una entrada utilizando imágenes digitales.

11 REFERENCIAS BIBLIOGRAFICAS:

[1] Preeti Singh, Mukesh Tripathi. Haar Cascade Classifier provides high accuracy even the images are highly afected by the Illumination, International Journal of Science, Technology & Management Volumen No.02, Septiembre 2013.

[2] J. G. RoshanTharanga, S. M. S. C. Samarakoon , T. A. P. Karunarathne 2 , K. L. P. M. Liyanage, M. P. A. W. Gamage, D. Perera. Smart Attendance using real time Face Recognition (SMART - FR), SAITM Research Symposium on Engineering Advancements 2013.

[3] S L Suma, Sarika Raga. "Real Time Face Recognition of Human Faces by using LBPH and Viola Jones Algorithm." International Journal of Scientific Research in Computer Science and Engineering ,Vol.6, 2018

[4] Souhail Guennouni, Anass Mansouri."Face Detection: Comparing Haar-like combined with Cascade Classifiers and Edge Orientation Matching", International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS), pp. 02-04, 2017.

[5] Mohanraj V., Sibi Chakkaravarthy S., Vaidehi V. (2019) Conjunto de redes neuronales convolucionales para el reconocimiento facial. En: Kalita J., Balas V., Borah S., Pradhan R. (eds) Desarrollos recientes en aprendizaje automático y análisis de datos. Avances en sistemas inteligentes y computación, vol 740. Springer, Singapur

[6] Puspita Majumdar, Akshay Agarwal, Richa Singh, Mayank Vatsa ; Los talleres de la Conferencia IEEE sobre Visión por Computadora y Reconocimiento de Patrones (CVPR), 2019, pp. 0-0

[7] S. Mittal y S. Mittal, "Reconocimiento de género a partir de imágenes faciales usando la red neuronal convolucional", Quinta Conferencia Internacional sobre Procesamiento de Información de Imágenes (ICIIP) de 2019 , Shimla, India, 2019, pp. 347-352, doi: 10.1109 / ICIIP47207. 2019.8985914.

[8] Moghekar, Rajeshwar; Ahuja , Saching. Modelo de aprendizaje profundo para el reconocimiento facial en un entorno sin restricciones. Journal of Computational and Theoretical Nanoscience , Volumen 16, Número 10, octubre de 2019, pp. 4309-4312 (4)

[9] Sander Soo, Object Detection using Haar-Cascade Classifier, Institute of Computer Science, University of Tartu. p. 1

[10] P. Viola and M. Jones, "Rapid object detection using a boosted classifier of simple features," in Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 2001, vol. 1, pp. 511–518.

[11] O'Shea, Keiron & Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks. ArXiv e-prints.

[12] Recuperado de UNIVERSITY OF WISCONSIN–MADISON , Computer Sciences http://pages.cs.wisc.edu/~twalker/CS766HW4/ Octubre 2019.

[13] CNN Architecture Series — VGG-16 with implementation https://medium.com/datadriveninvestor/cnn-architecture-series-vgg-16-with-implementation-part-i-bca79e7db415 .

[14] C. Enyinna, W. Ijomah, A. Gachagan and S. Marshall, Activation Functions: Comparison of Trends in Practice and Research for Deep Learning. Online: https://arxiv.org/pdf/1811.03378.pdf .

[15] J. Brownlee, online: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[16] Hongling Chen and Chen Haoyu 2019 J. Phys.: Conf. Ser. 1229 012015

[17] Ali Elmahmudi and HassanUgail, Deep Face Recognition using imperfect facial data, Centre for Visual Computing, Faculty of Engineering and Informatics, University of Bradford, Bradford BD7 1DP, UK. 2019

REFERENCIA DE FIGURAS:

Figura 1. OpenCV Dev Team, 2014 online: https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

Figura 2. Sarker, Md. Mostafa Kamal & Song, Moon. (2014). Real-Time Vehicle License Plate Detection Based on Background Subtraction and Cascade of Boosted Classifiers. The Journal of The Korean Institute of Communication Sciences. 39. 11. 10.7840/kics.2014.39C.10.909.

Figura 3. Kadbe, Premanand. (2015). Vision Based Hand Gesture Recognition with Haar Classifier and AdaBoost Algorithm.

Figura 4. Rezaei, Mahdi. (2016). Computer Vision for Road Safety: A System for Simultaneous Monitoring of Driver Behavior and Road Hazards.

Figura 5. Zhang, Zhengyou. (2010). A Survey of Recent Advances in Face Detection.

Figura 6. MathWorks. (2020). Online: https://www.mathworks.com/help/vision/ref/cascade_object_detector_cascade.png

Figura 7. DataBricks. (2020). Online: https://databricks.com/glossary/neural-network

Figura 8. H. Qassim, A. Verma and D. Feinzimer, "Compressed residual-VGG16 CNN model for big data places image recognition," 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, 2018, pp. 169-175, doi: 10.1109/CCWC.2018.8301729.