ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

GENERACIÓN DE UN MODELO DE RECONOCIMIENTO FACIAL EN VIDEOS USANDO REDES NEURONALES CONVOLUCIONALES BAJO UN ESQUEMA DE APRENDIZAJE POR TRANSFERENCIA.

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

JHIMMY ALEXANDER GONZÁLEZ CUEVA

DIRECTOR: Ph.D. FELIPE LEONEL GRIJALVA ARÉVALO

CODIRECTOR: M.Sc. JOSÉ ADRIÁN ZAMBRANO MIRANDA

Quito, enero 2021

AVAL

Certificamos que el presente trabajo fue desarrollado por Jhimmy Alexander González Cueva, bajo nuestra supervisión.

Ph.D. FELIPE LEONEL GRIJALVA ARÉVALO
DIRECTOR DEL TRABAJO DE TITULACIÓN

M.Sc. JOSÉ ADRIÁN ZAMBRANO MIRANDA
CODIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, JHIMMY ALEXANDER GONZÁLEZ CUEVA, 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 he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

JHIMMY ALEXANDER GONZÁLEZ CUEVA

DEDICATORIA

Este trabajo se lo dedico a mis padres, que junto a mis hermanos han estado siempre apoyándome sin importar la situación en la que me encuentre pasando. A mis abuelitos Balbina y Nicanor, que me brindaron su cariño en mis primeros años de vida. A Cristina, por haberme apoyado a seguir adelante en mi formación profesional. A Dennis, pues es la fuente de mi alegría y ganas de seguir avanzando en la vida.

Jhimmy González

AGRADECIMIENTO

Un agradecimiento especial a Dios, por haber puesto en mi vida a mis padres que siempre han estado apoyándome, por su gran paciencia en todo este tiempo que he estado junto a ellos.

Al Ph.D. Felipe Grijalva, por su guía brindada en la elaboración de este proyecto, que se ha desempeñado como un gran profesional y excelente persona.

A Cristina, por ser la madre de mi hijo y apoyarme en finalizar mis estudios.

A mis hermanos Noelia y Miguel, que han sido parte de mi vida.

A Cristian Paguay, Dustin Buitrón, Fernando Asimbaya, Jaime Ramírez y Manolo Revelo quienes aportaron de manera esencial al desarrollo del presente trabajo.

Finalmente, agradezco a mis compañeros Byron, Ronald, Francisco, Alex, Juan, Xavier, José, Alejandro y Ernesto, por haber sido participe de su amistad.

Gracias.

Jhimmy

ÍNDICE DE CONTENIDO

AVAL I
DECLARACIÓN DE AUTORIA II
DEDICATORIA III
AGRADECIMIENTO IV
ÍNDICE DE CONTENIDO V
RESUMEN XII
ABSTRACT XIII
1. INTRODUCCIÓN 1
1.1. JUSTIFICACIÓN 1
1.2. OBJETIVO GENERAL 2
1.3. OBJETIVOS ESPECÍFICOS 2
1.4. ALCANCE 2
1.5. MARCO TEÓRICO 3
1.5.1. APRENDIZAJE AUTOMÁTICO 3
1.5.1.1. Aprendizaje supervisado 4
1.5.1.2. Aprendizaje no supervisado 4
1.5.2. REDES NEURONALES Y APRENDIZAJE PROFUNDO 4
1.5.2.1. Entrenamiento de las redes con gradiente descendente 6
1.5.2.2. Entrenamiento de las redes con retropropagación 6
1.5.3. APRENDIZAJE PROFUNDO Y APRENDIZAJE POR TRANSFERENCIA . 8
1.5.4. ALGORITMO DE COHERENCIA TEMPORAL 9
1.5.5. REDES NEURONALES CONVOLUCIONALES RNC 9
1.5.5.1. Capa convolucional 9
1.5.5.2. Agrupación máxima (Max polling) 11
1.5.5.3. Aplanamiento (Flatten) 11
1.5.5.4. Abandono (Dropout) 11
1.5.5.5. Densa (Dense) 11
1.5.5.6. Funciones de activación 11

1.5.6. TECNOLOGÍAS DE SOFTWARE 12
1.5.6.1. Keras 12
1.5.6.2. OpenCV 13
1.5.6.3. Google Colab 13
1.5.6.4. Python 13
1.5.6.5. Anaconda 14
1.5.6.6. Spyder 14
1.5.6.7. PyCharm 14
2. METODOLOGÍA 16
2.1. BASE DE DATOS 16
2.2. RED NEURONAL CONVOLUCIONAL 21
2.3. RECONOCIMIENTO USANDO COHERENCIA TEMPORAL 33
2.4. EXPERIMENTOS 41
3. RESULTADOS Y DISCUSIÓN 43
3.1. RESULTADOS DE UTILIZAR LA VALIDACIÓN CRUZADA K-FOLD 43
3.2. RESULTADOS DE LAS PREDICCIONES DEL MODELO ELEGIDO JUNTO A UN ALGORITMO DE COHERENCIA TEMPORAL 44
4. CONCLUSIONES Y RECOMENDACIONES 50
4.1. CONCLUSIONES 50
4.2. RECOMENDACIONES 50
5. REFERENCIAS BIBLIOGRÁFICAS 52
ANEXOS 55
6. ANEXOS 56
ORDEN DE EMPASTADO 57

ÍNDICE DE TABLAS

2.1 Tabla de correspondencia etiqueta / posición / persona. 40
3.1 Tabla de correspondencia modelo / exactitud. 43
3.2 Tabla de eficiencia de cada ventana de los videos utilizados en los experimentos. 49

ÍNDICE DE FIGURAS

1.1 Funcionamiento de la programación con aprendizaje automático. 3
1.2 El Perceptrón, adaptado de [1]. 5
1.3 Red neuronal recurrente, diagrama adaptado de [2]. 6
1.4 Recursos utilizados en el Descenso de gradiente, figura adaptada de [3]. . . 7
1.5 Descenso de gradiente, figura tomada de [4]. 7
1.6 Retropropagación, figura tomada de [5]. 7
1.7 Deep Learning vs Transfer Learning, figura adaptada de [6]. 8
1.8 Modelos pre-entrenados, figura tomada de [7]. 8
1.9 Capas que conforman una red simple convolucional, figura adaptada de [8]. 9
1.10 Primera sección aplicado el operador convolución, figura adaptada de [9]. . . 10
1.11 Segunda sección aplicado el operador convolución, figura adaptada de [9]. . 10
1.12 Convolución total de la imagen, figura adaptada de [9]. 10
1.13 Proceso de MaxPolling, figura adaptada de [10]. 11
1.14 Proceso de aplanamiento, figura tomada de [11]. 11
1.15 Proceso de Dropout, figura adaptada de [12]. 12
1.16 Red neuronal con capas ocultas de tipo densa, figura adaptada de [13]. . . . 12
1.17 Funciones de activación. 13
1.18 Entorno de Google Colab. 14
1.19 Relación entre Python, Anaconda y Spyder, figura tomada de [14].. 14
2.1 Diagrama de bloques del proceso de extracción de rostros, entrenamiento del modelo y procesos de pruebas con un nuevo video. 16
2.2 Esquema del proceso de extracción de fotogramas. 17
2.3 Esquema del proceso de extracción de rostros detectados en fotogramas. . . 18
2.4 Imágenes de fotogramas para creación de datos de videos. 20
2.5 Imágenes de rostros recortados de los fotogramas. 20
2.6 Base de datos de rostros divididos por el número de video de cada persona. 20
2.7 Esquema que añade capas a el modelo. 22
2.8 Modelo VggFace 23
2.9 Modelo nuevo basado en el modelo VggFace 23
2.10 Esquema del entrenamiento del modelo. 24
2.11 Esquema de predicción de fotogramas. 29
2.12 Ventana deslizante 33
2.13 Esquema de predicción con coherencia temporal. 34
2.14 Predicciones que están presentes en la primera ventana. 37

2.15 Diccionario de ventanas constituida netamente por predicciones. 38
2.16 Diccionario de ventanas constituida con keys llenas. 38
2.17 Folds creados a partir de la base de datos. 41
2.18 Grupos de folds para entrenamiento de modelos. 42
3.1 Matriz de confusión del modelo con mayor exactitud en las predicciones. . . 44
3.2 Curva ROC de las predicciones del modelo elegido. 45
3.3 Precisión de cada ventana desde la ventana 1 a la 150. 49

SEGMENTOS DE CÓDIGOS

2.1 Código para extraer un fotograma. 17
2.2 Código para instanciar el detector MTCNN y detectar rostros en un fotograma. 18
2.3 Código para obtener el recuadro de coordenadas de píxeles que contiene la imagen y normalización de tamaño. 19
2.4 Importación de librerías necesarias. 21
2.5 Librerías necesarias para el proceso de entrenamiento. 22
2.6 Código que detalla la información de entrenamiento y validación. 23
2.7 Inicialización de pesos de cada clase. 25
2.8 Normalización de dimensiones y tamaño de lotes de imágenes de los directorios de entrenamiento y validación para su procesamiento. 25
2.9 Configuración del modelo para el entrenamiento. 26
2.10 Configuración de retro llamada *(Callback)* para guardar el nuevo modelo generado. 27
2.11 Variación de la tasa de aprendizaje. 27
2.12 Configuración de parámetros para entrenar el modelo. 28
2.13 Librerías utilizadas para realizar predicciones. 28
2.14 Código para cargar modelo. 29
2.15 Código que lista los fotogramas. 29
2.16 Lectura de la imagen y extracción de dimensiones y canales de la misma. . . 29
2.17 Instancia del detector de rostros. 30
2.18 Línea de código para detectar rostros. 30
2.19 Extracción de dimensiones del rectángulo que rodea al rostro detectado. . . 30
2.20 Normalización de la imagen del rostro detectado. 30
2.21 Creación de diccionario temporal por cada fotograma. 31
2.22 Código para guardar la imagen del rostro detectado e instanciado. 31
2.23 Código que carga la imagen para prepararlo para que el modelo haga la predicción. 31
2.24 Código que transforma la matriz de predicción en una lista y agrega un nuevo elemento para el diccionario llamado *nombre*. 32
2.25 Código que guarda los archivos pickle. 32
2.26 Librerías necesarias para utilizar la coherencia temporal. 33
2.27 Inicialización de variables. 33
2.28 Creación de elementos dentro de un diccionario de ventanas. 35
2.29 Verificación de archivos de predicción acorde al fotograma. 35

2.30 Predicciones del fotograma 0. 35
2.31 Predicciones del fotograma 1. 36
2.32 Inicialización de una lista de 0s de cada elemento del diccionario. 36
2.33 Promedio de predicciones tomando en cuenta la posición. 36
2.34 Elimina posiciones de la lista de predicciones que siguen en 0. 37
2.35 Elimina keys vacías. 38
2.36 Elimina predicciones que están bajo el umbral. 38
2.37 Número de veces que es reconocida una persona en un video. 39
2.38 Código para discriminar si una persona fue detectada al menos en una ventana. 39
2.39 Muestra los nombres de las personas reconocidas dentro del video. 40

RESUMEN

La utilización actual de redes neuronales convolucionales *CNN* en el aprendizaje automático ha dado paso a un sinnúmero de actividades y procesos que hasta hace pocos años eran impensables que se puedan realizar por programas computacionales. De manera notoria hay grandes avances en el reconocimiento facial que han partido del uso de las redes neuronales convolucionales.

Aprovechando estos importantes avances, este trabajo utiliza la técnica del aprendizaje por transferencia para poder utilizar redes neuronales convolucionales ya entrenadas en el reconocimiento facial. De esta manera, tomando en cuenta la gran exactitud que ha presentado la red neuronal convolucional VggFace, se tomó como base a dicha red para poder realizar la técnica de transferencia de aprendizaje. Específicamente, fue creada una base de datos de imágenes de rostros recolectada para este propósito. A partir de esta base, se entrenó un modelo de reconocimiento facial usando la red neuronal VggFace. Con el modelo entrenado se hizo predicciones de los fotogramas que conforman videos en los que aparecen múltiples personas. Las predicciones sobre los fotogramas de dichos videos fueron sometidas a un algoritmo de coherencia temporal. El algoritmo utiliza ventanas deslizantes sobre fotogramas consecutivos.

Los experimentos demuestran que la exactitud de predicción del modelo generado incrementa notablemente con el uso de coherencia temporal. Esto se logra mediante una ventana deslizante en las predicciones realizadas sobre videos de múltiples personas presentes en el mismo.

PALABRAS CLAVE: Redes neuronales convolucionales, aprendizaje por transferencia, validación cruzada k-fold, modelo, reconocimiento facial.

ABSTRACT

The current use of networks neuronal convolutional CNN in machine learning has given way to countless activities and processes that until a few years ago were unthinkable that could be carried out by computer programs. Notoriously, there are great advances in facial recognition that have started from the use of convolutional neural networks.

Taking advantage of these important advances, this work uses the transfer learning technique to be able to use convolutional neural networks already trained in facial recognition. In this way, taking into account the great accuracy presented by the convolutional neural network VggFace, this network was used as a basis to perform the learning transfer technique. Specifically, a database of images of faces collected for this purpose was created. From this basis, a facial recognition model was trained using the VggFace neural network. With the trained model, predictions were made of the frames that make up videos in which multiple people appear. The predictions on the frames of these videos were subjected to a temporal coherence algorithm. The algorithm uses sliding windows over consecutive frames.

Experiments show that the prediction accuracy of the generated model increases markedly with the use of temporal coherence. This is achieved through a sliding window in the predictions made on videos of multiple people present in the same.

KEY WORDS: Convolutional neural networks, transfer learning, k-fold cross-validation, model, facial recognition.

1. INTRODUCCIÓN

La mayoría de las investigaciones enfocadas a la detección y reconocimiento de rostros han sido realizadas en base a imágenes en escala de grises [15] [16], otras toman en cuenta el color de las imágenes [17] [18], así como la orientación del rostro [19] [20]. Existen dos grandes familias de métodos para realizar el reconocimiento facial: métodos holísticos y métodos no holísticos. Los métodos holísticos procesan la imagen del rostro de manera global. Entre estos métodos se tiene a Eigenfaces basado en análisis de componentes principales (PCA) [21, 22] y Fisherfaces [23]. Por otro lado, los métodos no holísticos están basados en la extracción manual o automática de características locales en las imágenes de rostros [24]. Por ejemplo, LPB (Local Binary Patterns) [25] el cual se basa en el procesamiento de texturas locales en la imagen del rostro. Otro ejemplo de métodos no holísticos son las redes neuronales profundas (i.e. deep learning) que son ahora el estado del arte en reconocimiento facial, las cuales, en lugar de usar extracción manual de características, están basadas en aprendizaje profundo mediante redes neuronales convolucionales para aprender las características de manera automática [26]. No obstante, el problema del aprendizaje profundo es que requiere grandes cantidades de información (i.e. extensas bases de datos de personas a ser reconocidas); en consecuencia, el procesamiento computacional aumenta considerablemente. Para solventar este problema algunos trabajos han propuesto el uso del aprendizaje por transferencia (Transfer Learning) [27]. El aprendizaje por transferencia consiste en aprender una nueva tarea a través de una tarea relacionada que ya ha sido aprendida; es decir, utiliza un modelo que ha sido pre-entrenado con grandes bases de datos para lograr un óptimo funcionamiento en una base de datos pequeña. De este modo, la idea general de aprendizaje por transferencia es adaptar un modelo pre-entrenado a una base de datos nueva. Por otra parte, el reconocimiento facial a partir de videos provee una oportunidad para aprovechar la coherencia temporal de los mismos, similar a lo propuesto en [28] donde se demuestra que la precisión del reconocimiento facial sobre videos aumenta considerando algoritmos que trabajan sobre varios fotogramas consecutivos del mismo. Es así como este estudio técnico pretende construir una base de datos de rostros en videos y generar un modelo de aprendizaje profundo, bajo un esquema de aprendizaje por transferencia aprovechando la coherencia temporal del video. A diferencia de [28], donde los videos contienen apenas un rostro a ser reconocido, en este proyecto se usarán videos con varios rostros en los mismos. Este proyecto se plantea como una alternativa en el campo de las aplicaciones que requieren analizar videos con el objetivo de reconocer múltiples rostros en los mismos.

1.1. JUSTIFICACIÓN

El presente trabajo está enfocado en predecir la identificación de más de una persona dentro de un video, que será validado por un algoritmo de coherencia temporal, con el objetivo de

obtener una mejor precisión de reconocimiento facial aprovechando múltiples fotogramas de un video.
Existen muchas aplicaciones prácticas como la biometría facial en sistemas de seguridad donde es crítico tener la menor cantidad de errores. Por otro lado, en aplicaciones como la videovigilancia (e.g. ojos de águila, red de cámaras ECU 911) donde la información a ser procesada es video, i.e. no imágenes estáticas, es imprescindible tener la menor cantidad de falsos positivos. De ahí la necesidad de contar con algoritmos que aprovechen información de varios fotogramas de un video como la coherencia temporal propuesta en este trabajo.
Este trabajo justifica su implementación pues, si bien es cierto modelos de reconocimiento facial con redes neuronales convolucionales ya han sido implementados [28], el presente proyecto usa información de los fotogramas de videos en los que se toma en cuenta más de una persona para predecir su identificación, mejorando la utilidad del modelo de reconocimiento a la hora de analizar videos con múltiples personas.

1.2. OBJETIVO GENERAL

Generar un modelo de reconocimiento facial en videos usando redes neuronales convolucionales bajo un esquema de aprendizaje por transferencia.

1.3. OBJETIVOS ESPECÍFICOS

- Analizar las redes neuronales convolucionales y el aprendizaje por transferencia en el contexto del reconocimiento de rostros.
- Entrenar una red neuronal convolucional para reconocimiento de rostros bajo un esquema de aprendizaje por transferencia.
- Adaptar el algoritmo de coherencia temporal para videos con varios rostros.
- Analizar la exactitud del modelo basado en coherencia temporal mediante validación cruzada [29].

1.4. ALCANCE

Se creó una base de datos con los rostros de 20 personas, los cuales fueron extraídos de videos que fueron grabados con una cámara de video en ambientes interiores y exteriores, que posteriormente se etiquetaron de acuerdo con la identidad de personas presentes en el mismo. La Fig.2.1 muestra un diagrama de bloques de los procesos que fueron implementados en este proyecto. En primera instancia se encuentra una base de datos conformada por los videos descritos anteriormente. Cada uno de estos videos se sometió a un proceso de extracción de fotogramas, dentro de los cuales se detectó y extrajo los rostros presentes en cada uno. Estos rostros detectados junto a su respectiva etiqueta formaron una base de

datos de imágenes de rostros, la cual sirvió para entrenar una red neuronal convolucional aprovechando el aprendizaje por transferencia, mediante un modelo pre-entrenado, y así se generó un modelo de reconocimiento facial acorde a nuestra información. Este modelo se sometió a evaluaciones con nuevos videos no utilizados en la fase de entrenamiento, que predijo las etiquetas correspondientes a los rostros detectados en cada fotograma. La decisión final sobre los rostros presentes en el video se dictaminó a través de un algoritmo de decisión basado en coherencia temporal similar a [28]. Este algoritmo de decisión usa varias predicciones en una ventana deslizante sobre N fotogramas consecutivos del video. Dado que en [28] se considera coherencia temporal en videos con un solo rostro presente, en este proyecto se adaptaron las ideas de [28] para videos con múltiples rostros. Finalmente, es importante mencionar que el enfoque propuesto fue validado en un esquema de validación cruzada [30]. Se obtuvo un producto final demostrable que es un archivo script escrito en Python y se entregó el modelo generado.

1.5. MARCO TEÓRICO

En este capítulo se describirán los temas que son base para el desarrollo del proyecto. Estos temas son el aprendizaje automático (*machine learning*), redes neuronales, aprendizaje profundo (*deep learning*), aprendizaje por transferencia (*transfer learning*), redes neuronales convolucionales (*CNN*), capa convolucional y las tecnologías de software que hicieron posible el desarrollo del proyecto.

1.5.1. APRENDIZAJE AUTOMÁTICO

El aprendizaje automático, que es una traducción del inglés *Machine Learning*, se lo considera como un subcampo de la inteligencia artificial (*AI*) que tiene como objetivo la creación de programas computacionales que van mejorando su rendimiento automáticamente basados en la experiencia adquirida. El aprendizaje automático no necesita que el programador indique los algoritmos o reglas que se necesitan para resolver un problema [31]. Como se puede observar en la figura 1.1.

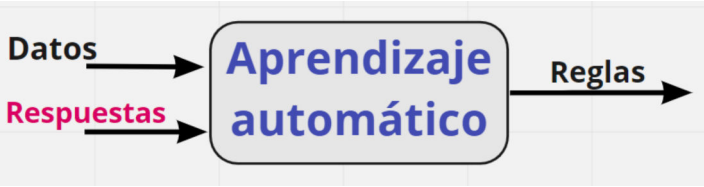

Figura 1.1: Funcionamiento de la programación con aprendizaje automático.

El aprendizaje automático en general necesita de una gran cantidad de datos y etiquetas. Estos datos pueden representar un sinnúmero de actividades, objetos, expresiones, etc. También se necesitan conocer las etiquetas que van a representar los datos. Las etiquetas que ingresan al aprendizaje automático son números, pues existen dos razones, la primera es que las computadoras trabajan mucho mejor con números y la segunda razón es que

al trabajar con texto podría haber un sesgo del idioma, pues no existe un solo idioma. Por tales motivos, al ser los números casi un idioma universal, las etiquetas son representadas a través de números.
El aprendizaje automático se divide en dos tipos, que son el aprendizaje supervisado y el aprendizaje no supervisado.

1.5.1.1. Aprendizaje supervisado

Es el tipo de aprendizaje que se utiliza con mayor frecuencia. La idea básica del aprendizaje supervisado es enseñarle a la computadora el cómo hacer algo. Este tipo de aprendizaje busca generalizar y predecir a partir de la información suministrada, además, se encarga de resolver problemas utilizando datos con sus etiquetas. Divide los problemas en problemas de regresión y de clasificación. Los problemas de regresión intentan predecir resultados asignando una variable de entrada a una función continua, mediante el uso de información cuantitativa. Por ejemplo, la predicción de la cosecha de cacao en un año promedio.
Los problemas de clasificación tratan de predecir resultados discretos, es decir localizar las variables de entrada en una categoría [32], por ejemplo, predecir la marca o el sabor de gaseosa (cola) que le gusta a una persona.

1.5.1.2. Aprendizaje no supervisado

Este tipo de aprendizaje deja que aprenda la computadora por sí misma. El aprendizaje no supervisado se encarga de los problemas que utilizan los datos sin etiquetas. Este tipo de aprendizaje tiene como objetivo el comprender y abstraer los patrones de los datos suministrados. Para encontrar estos patrones la forma más común es la agrupación. La agrupación crea grupos de datos con características diferentes [33].
Un ejemplo de aprendizaje no supervisado de la vida real es la clasificación de perros y gatos. Los algoritmos toman la decisión en función de las similitudes y diferencias entre las fotos.

1.5.2. REDES NEURONALES Y APRENDIZAJE PROFUNDO

El aprendizaje profundo, que es una traducción del inglés *Deep Learning*, es un subconjunto especializado de *Machine Learning* que crea capas de algoritmos para crear una red neuronal. Una red neuronal es un conjunto de unidades más pequeñas llamadas neuronas, que son unidades de computación modeladas sobre la manera de procesar la información del cerebro humano. La finalidad de imitar a la red neuronal biológica del cerebro es la de aproximar sus resultados de procesamiento mediante funciones matemáticas. [34]. La primera red neuronal fue el Perceptrón [35], esta fue la red neuronal más simple de una sola capa que consiste en nodos de entrada conectados directamente a un nodo de salida. A continuación, en la figura 1.2 se detalla la representación del gráfico del Perceptrón. Los nodos de entrada que están representados por 1, x1, x3, x4, son multiplicados por los correspondientes factores W que representan los pesos w1, w2, w3, w4 y cuyos resultados son sumados en la neurona de salida. En particular, la entrada 1, es multiplicada por el sesgo

que proviene del inglés *bias*[1] [36] que está representado por el peso w1, pero en la mayoría de literatura se representa este peso con la letra *b*. Finalmente, este sumatorio pasa por la función de activación descrita en la sección 1.5.5.6, cuyo propósito es añadir deformaciones no lineales a la salida de cada neurona para poder concatenar varias de estas.

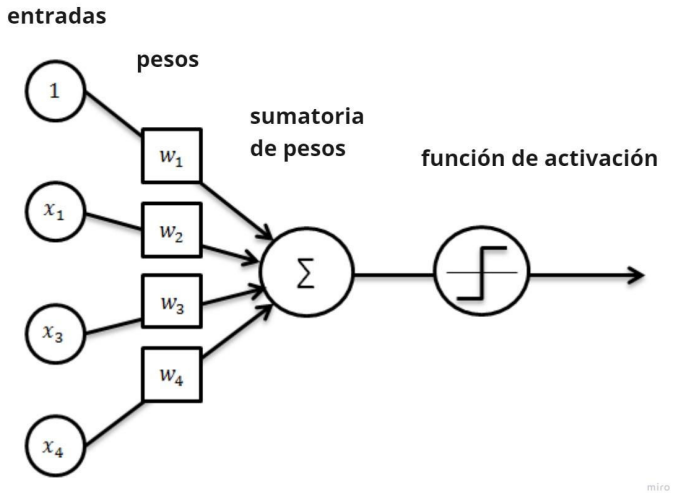

Figura 1.2: El Perceptrón, adaptado de [1].

Existen algunos tipos de redes neuronales, las principales son las siguientes:

- **Totalmente conectadas (Fully connect)**: Este tipo de red neuronal posee una conexión entre todas las neuronas de una capa con las neuronas de sus capas vecinas [37].
- **Convolucionales**: Estas son redes multicapa que se inspiran en el *cortex visual animal*. Estas redes están especializadas en aplicaciones en las que intervienen el procesamiento de imágenes, reconocimiento de video y el procesamiento del lenguaje natural [8]. Estas redes se abordarán en detalle en la sección 1.5.5.
- **Recursivas (RNNs)**: Este tipo de redes realizan las mismas operaciones para cada elemento de una secuencia. Estas poseen salidas que alimentan las entradas de la siguiente etapa. Las *RNNs* han transformado el reconocimiento de voz, el procesamiento del lenguaje natural y otras áreas. El diagrama básico de funcionamiento se muestra en la figura 1.3.

  Una de las redes que nacen de las RNNs es la red GRU. Una GRU es una modificación a las capas ocultas de una RNN que capturan conexiones de largo alcance de mejor manera. Esta característica ayuda a recordar, por ejemplo, palabras que expresan características de singularidad o pluralidad. Las GRUs se ejecutan computacionalmente un poco más rápido comparadas con la red de la que proviene llamada LSTM *Long short-term memory*.

[1] bias: Es la diferencia entre la predicción esperada de nuestro modelo y los valores verdaderos.

Las LSTMs pueden tener memorias de corto plazo como la de una RNN básica y una memoria de largo plazo. Estas redes son más generales pues poseen una compuerta de olvido, la cual facilita el remover información de la memoria. Las redes LSTMs son más antiguas que las GRUs.

Figura 1.3: Red neuronal recurrente, diagrama adaptado de [2].

1.5.2.1. Entrenamiento de las redes con gradiente descendente

La técnica del gradiente descendiente consiste en encontrar los valores de W y b (*bias*) para minimizar la función costo. Pues la función costo nos da el promedio de los errores de predicción [38]. Este proceso es realizado en cada época o iteración de entrenamiento del modelo. Existen tres tipos de entrenamiento con gradiente descendente los cuales son:

- **Descenso de gradientes por lotes:** Toma el promedio de los gradientes de todos los elementos del conjunto de datos de entrenamiento y luego usa ese gradiente promedio para actualizar todos los valores de los parámetros [38].
- **Descenso de gradiente estocástico:** Considera tomar un elemento a la vez del conjunto de entrenamiento, alimenta la red, luego calcula su gradiente. El gradiente calculado actualiza los pesos de todos los enlaces. Entonces, este procedimiento lo hace con cada uno de los elementos uno a uno [38].
- **Descenso de gradientes por mini-lotes:** Se utiliza lotes de datos para actualizar los pesos de los enlaces que hay entre neuronas. Elige un pequeño lote para alimentar la red. Con este pequeño lote, se calcula el gradiente medio y finalmente actualiza los pesos [38].

Estos tres tipos de Gradiente Descendente difieren en la utilización de recursos, tal como se ilustra en la figura 1.4.
Por otro lado, la figura 1.5 muestra como avanzan las iteraciones cuando se realiza el gradiente descendente.

1.5.2.2. Entrenamiento de las redes con retropropagación

Aplicar retropropagación es operar de forma recursiva capa tras capa moviendo el error hacia atrás. Gracias a esto cuando llegamos a la primera capa habremos obtenido cual es

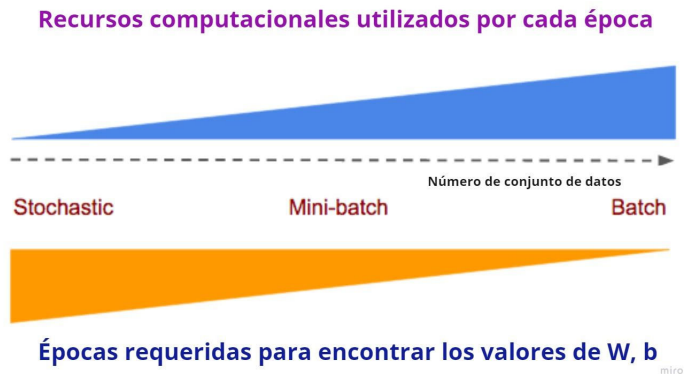

Figura 1.4: Recursos utilizados en el Descenso de gradiente, figura adaptada de [3].

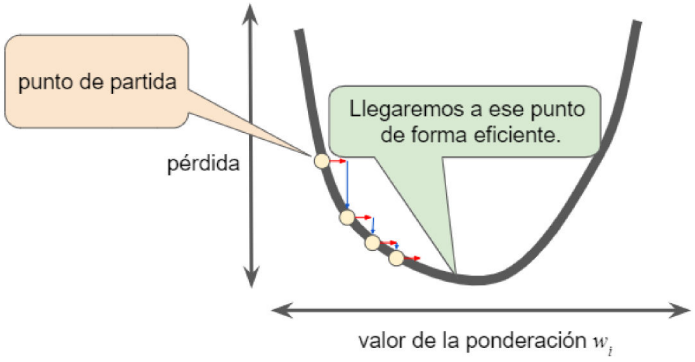

Figura 1.5: Descenso de gradiente, figura tomada de [4].

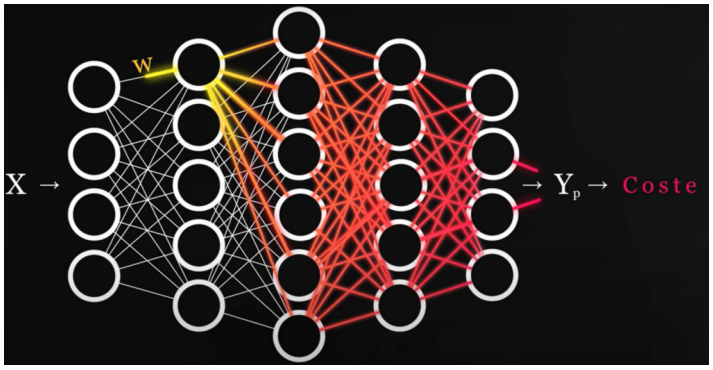

Figura 1.6: Retropropagación, figura tomada de [5].

el error por cada neurona y para cada uno de sus parámetros. Los errores obtenidos en cada neurona van a servir para calcular las derivadas parciales de cada uno de los parámetros de la red con respecto a la función costo [39]. Es así como con estas derivadas parciales llegamos a conformar el vector gradiente que es lo que necesita el algoritmo del descenso de gradiente visto en la sección 1.5.2.1 para minimizar el error para entrenar la red. En la figura 1.6 se puede esquematizar la retropropagación.

1.5.3. APRENDIZAJE PROFUNDO Y APRENDIZAJE POR TRANSFERENCIA

Para crear un modelo de redes neuronales desde cero, se necesita tener una gran cantidad de información como de recursos computacionales para obtener resultados aceptables. A menudo se realiza mayores progresos si se inicia la creación de nuevos modelos a partir de modelos pre-entrenados. Debido a que aprovecha las características que estos modelos pre-entrenados han aprendido en bajos niveles de la red, esta técnica es llamada aprendizaje por transferencia *(Transfer learning*) [27]. La comunidad de investigadores que trabajan con redes neuronales ha publicado bastante buena información sobre redes que se han entrenado con un enorme conjunto de datos. En la figura 1.7 se muestra la comparación de la cantidad de información que necesita un modelo para ser entrenado desde cero y utilizando la técnica del aprendizaje por transferencia.

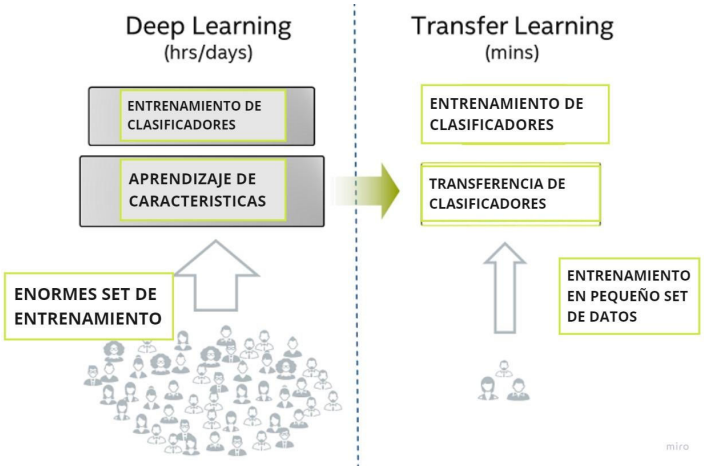

Figura 1.7: Deep Learning vs Transfer Learning, figura adaptada de [6].

Otra técnica de reutilización de modelos de redes neuronales pre-entrenadas es el ajuste fino (*Fine Tunnig*), la cual consiste en descongelar algunas capas cercanas a las últimas convoluciones para realizar un ajuste de estas y entrenar su nuevo conjunto de datos de entrenamiento.

Algunos de los modelos pre-entrenados se muestran en la figura 1.8.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.715	0.901	138,357,544	23
VGG19	549 MB	0.727	0.910	143,667,240	26
ResNet50	99 MB	0.759	0.929	25,636,712	168
InceptionV3	92 MB	0.788	0.944	23,851,784	159
InceptionResNetV2	215 MB	0.804	0.953	55,873,736	572
MobileNet	17 MB	0.665	0.871	4,253,864	88
DenseNet121	33 MB	0.745	0.918	8,062,504	121
DenseNet169	57 MB	0.759	0.928	14,307,880	169
DenseNet201	80 MB	0.770	0.933	20,242,984	201

Figura 1.8: Modelos pre-entrenados, figura tomada de [7].

1.5.4. ALGORITMO DE COHERENCIA TEMPORAL

La coherencia temporal analiza fotogramas consecutivos y comienza a procesar cada fotograma, esta información resultante es utilizada para ser comparada con los fotogramas consecutivos para emitir un resultado, como por ejemplo, el color de los píxeles, las predicciones, etc. El algoritmo de coherencia temporal es un código cuyo objetivo principal es iterar grupos o ventanas de información de cada fotograma tomando en cuenta la coherencia temporal de los mismos.

1.5.5. REDES NEURONALES CONVOLUCIONALES RNC

Las redes neuronales convoluciones son redes multicapa que se inspiran en el cortex animal visual. Este tipo de redes neuronales son utilizadas en aplicaciones tales como procesamiento de imágenes, reconocimiento de video y procesamiento de lenguaje natural. Son utilizadas en las aplicaciones anteriores porque son capaces de reducir los parámetros de la información a ser procesados. Las capas que las componen principalmente son las capas convolucionales, capas de agrupación y las densamente conectadas. De las capas convolucionales proviene su nombre de redes neuronales convolucionales, tal como se puede apreciar en la figura 1.9. Estas capas convolucionales son buenas para detectar estructuras simples en una imagen y unir estas características simples para construir características complejas. En una red convolucional este proceso ocurre sobre una serie de capas, cada una efectúa una convolución en la salida de la capa anterior [8].

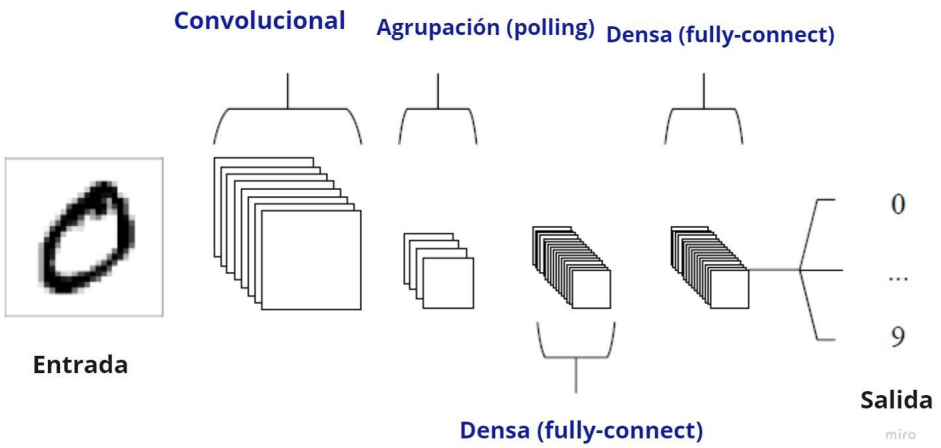

Figura 1.9: Capas que conforman una red simple convolucional, figura adaptada de [8].

Las RNCs pueden estar constituidas por los siguientes tipos de capas:

1.5.5.1. Capa convolucional

Esta capa toma como entrada a regiones de píxeles cercanos de una imagen, que son recorridas de izquierda a derecha y hacia abajo a través de uno o varios filtros. Entre el filtro y las regiones de píxeles se realiza la operación de convolución. Esta operación es la característica principal de una capa convolucional. Al aplicar el operador convolución la imagen encoge y pierde información de bordes porque son muy pocas veces operados en

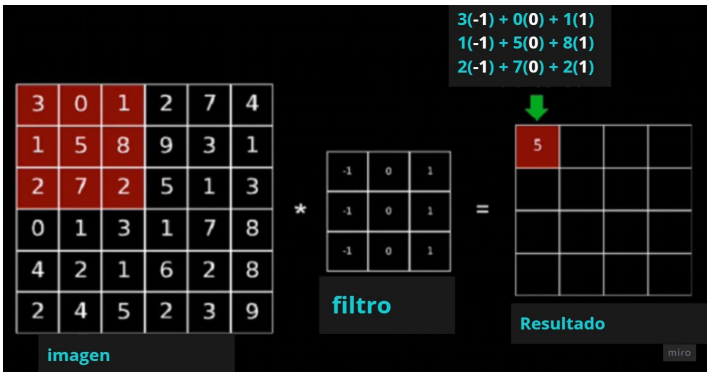

Figura 1.10: Primera sección aplicado el operador convolución, figura adaptada de [9].

comparación con los píxeles del centro de la imagen [40]. La pérdida de información de bordes puede ser solucionada utilizando un relleno de píxeles alrededor de la imagen. La operación convolución realiza una multiplicación de elemento a elemento entre el filtro y la región de píxeles de la imagen correspondiente, estos resultados son sumados. El resultado será el nuevo elemento de la nueva imagen o matriz. El proceso de esta operación se la puede observar en las figuras 1.10, 1.11 y 1.12.

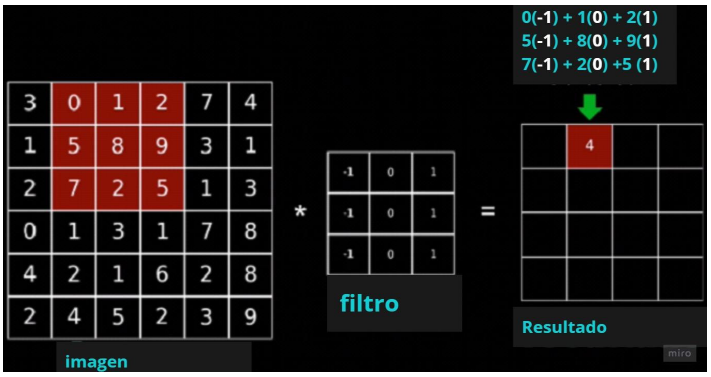

Figura 1.11: Segunda sección aplicado el operador convolución, figura adaptada de [9].

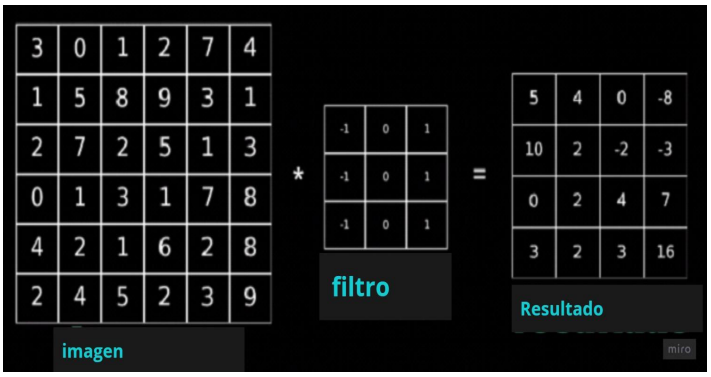

Figura 1.12: Convolución total de la imagen, figura adaptada de [9].

1.5.5.2. Agrupación máxima (Max polling)

Esta capa detecta múltiples características de un conjunto de píxeles cercanos, esto lo realiza a través de un filtro. Este filtro nos entrega como resultado la salida del máximo de cada grupo dentro de la imagen. Además de mantener la relación espacial de la imagen. En la figura 1.13 puede observar el proceso.

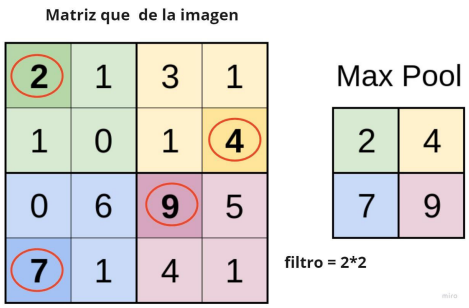

Figura 1.13: Proceso de MaxPolling, figura adaptada de [10].

1.5.5.3. Aplanamiento (Flatten)

Esta capa lo que hace es transformar a una matriz de n-dimensiones en un vector plano, es decir, de dimensión 1 *(uno)*. En la figura 1.14 se observa este proceso.

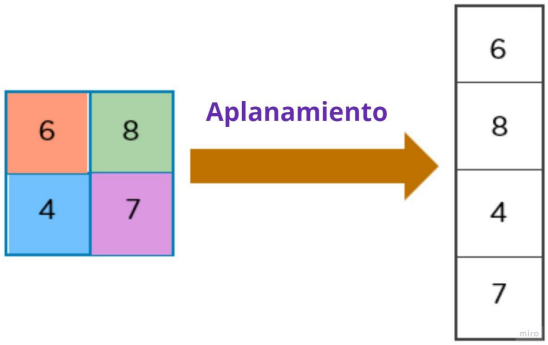

Figura 1.14: Proceso de aplanamiento, figura tomada de [11].

1.5.5.4. Abandono (Dropout)

El proceso que realiza es desconectar algunas neuronas al azar para evitar el sobreentrenamiento de la red. Entonces realiza el proceso mostrado en la figura 1.15.

1.5.5.5. Densa (Dense)

Estas capas conectan cada neurona de la capa anterior con cada neurona de la capa actual *Dense*. Como se puede observar en la figura 1.16.

1.5.5.6. Funciones de activación

Las funciones de activación tienen un propósito fundamental el cual es propagar las salidas de los nodos de una capa hacia la siguiente capa [41]. Entre las más utilizadas están las

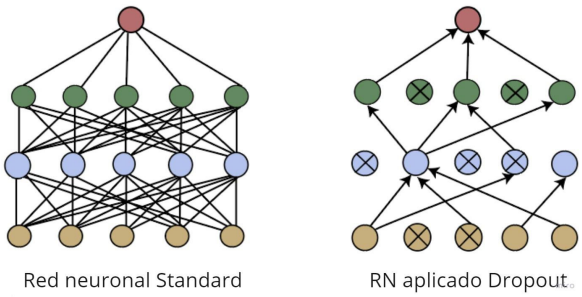

Figura 1.15: Proceso de Dropout, figura adaptada de [12].

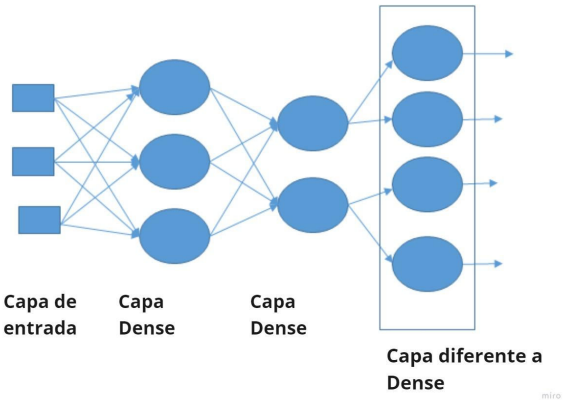

Figura 1.16: Red neuronal con capas ocultas de tipo densa, figura adaptada de [13].

siguientes:

Sigmoide: Utilizada con mayor frecuencia como función de activación en la neurona de salida cuando se requiere realizar una clasificación binaria.

Relu: Es la función de activación con mayor utilización de la sigmoide pues tiene una curva de aprendizaje mucho mejor. Solo activa ciertas neuronas pues activara a las neuronas que tengan la variable z mayor a 0.

Softmax: Es utilizada en problemas multiclase, y así obtener predicciones de más de 2 posibles resultados como el caso binario.

La representación matemática de las funciones de activación mencionadas se muestra en la figura 1.17.

1.5.6. TECNOLOGÍAS DE SOFTWARE

1.5.6.1. Keras

Es un API de alto nivel de aprendizaje escrito en Python. Fue desarrollada por Google y, lanzado en el año 2015. El objetivo de este software es acelerar la creación de redes neuronales. Para lograr este objetivo puede soportar redes convolucionales, redes recurrentes y cualquier combinación de ambas. Keras puede ser ejecutado sobre frameworks como Tensorflow. Tensorflow realiza la implementación de Keras así (*tf.keras*) [42].

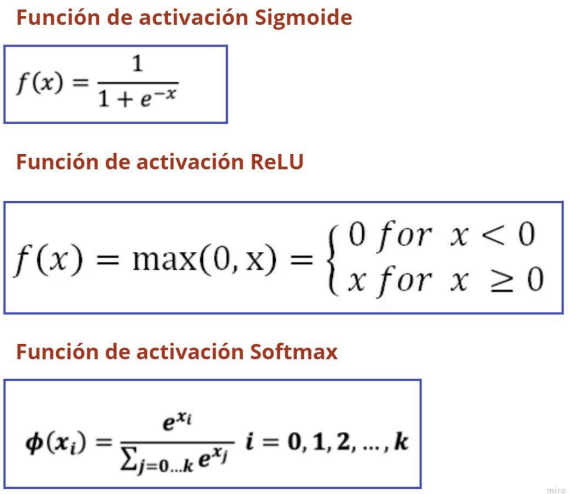

Figura 1.17: Funciones de activación.

1.5.6.2. OpenCV

OpenCv fue creado por la división rusa de Intel. Es una biblioteca muy extensa de funciones de visión por computadoras. Tiene bibliotecas para múltiples lenguajes de programación como Python, C++, java. OpenCv posee más de 500 funciones para realizar procesamiento de imágenes, como reconocimiento de objetos, calibración de cámaras, visión robótica, etc. Es una solución de alto nivel gratuita y flexible multiplataforma [43].

1.5.6.3. Google Colab

Es un servicio de Cloud que permite el uso gratuito de las GPUs y TPUs de Google, es una herramienta con la cual se puede desarrollar aplicaciones basadas en el aprendizaje automático y el aprendizaje profundo. Se puede utilizar como fuentes de datos los archivos que estén en las cuentas de Google Drive, Github e incluso en otros sistemas de almacenamiento de Cloud como Amazon. La única limitante en este servicio es que permite el trabajo continuo solo por 12 horas. Pues si se excede este tiempo, el servicio se detiene y se borran las variables utilizadas con los resultados obtenidos. El entorno de trabajo de Colab permite escribir segmentos de código y texto [44]. Esto se puede observar en la figura 1.18.

1.5.6.4. Python

Es un lenguaje de programación muy fácil de manejar. La primera versión de Python se publicó hace más de 30 años en 1989 desarrollado por Guido Van Rossum. Es muy versátil pues es un lenguaje multiparadigma y multiplataforma. Python contiene una gran cantidad de librerías y funciones que permiten realizar muchas tareas comunes sin necesidad de ser programadas desde cero. Una de las razones para su gran aceptación es que es de código abierto. Python puede trabajar con grandes volúmenes de información pues facilita las herramientas para extraer y procesar estos, por esta razón, es elegido por empresas de Big Data [45].

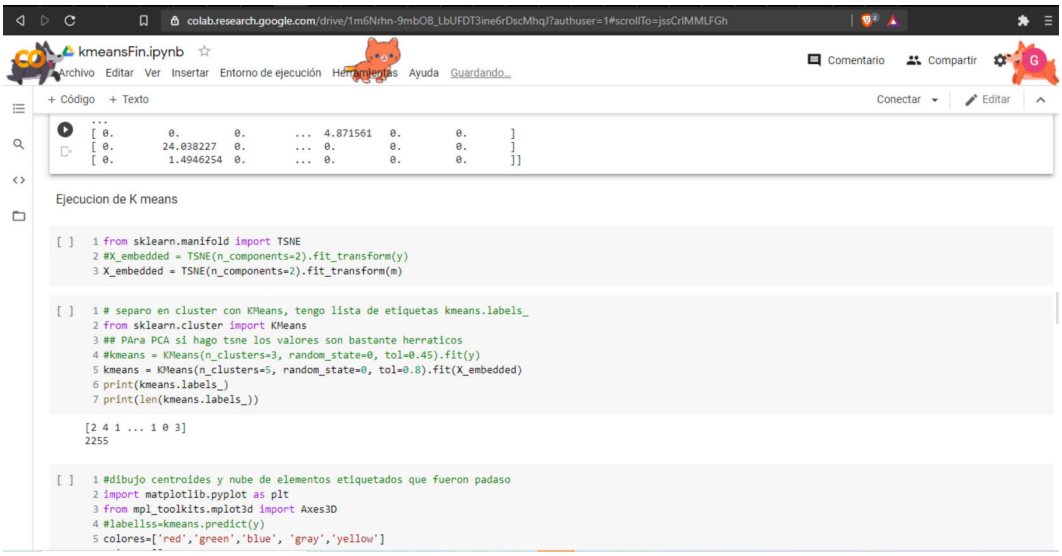

Figura 1.18: Entorno de Google Colab.

1.5.6.5. Anaconda

Es una distribución de código abierto que abarca una serie de aplicaciones, librerías y conceptos para ser utilizados en el desarrollo de la ciencia de datos con Python. Funciona como un gestor de entorno y posee una gran colección de paquetes de código abierto. Al instalar Anaconda trae integrado el IDE Spyder, la figura 1.19 representa esta relación que hay entre Python, Anaconda y Spyder.

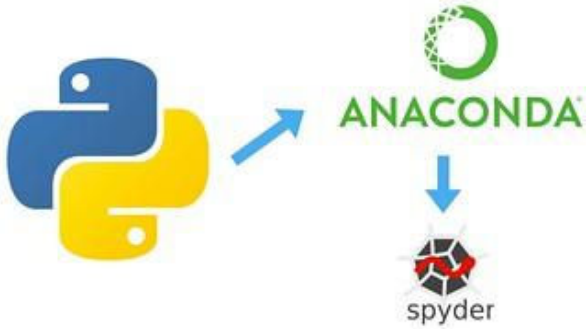

Figura 1.19: Relación entre Python, Anaconda y Spyder, figura tomada de [14]..

1.5.6.6. Spyder

Es un entorno de desarrollo científico de Python. Es un entorno de desarrollo integrado (IDE) que viene integrado en la distribución de Anaconda. Está conformado por funciones de edición, pruebas interactivas, depuración e introspección.

1.5.6.7. PyCharm

Es un entorno de desarrollo de los más completos para programar en Python. Es parte del suite de herramientas de desarrollo ofrecidas por JetBrains. Existen dos opciones, la profesional y community. La primera está enfocada al desarrollo de páginas web, mientras

que la segunda está orientada al desarrollo científico. La interfaz de PyCharm es intuitiva y similar a entornos de desarrollo. Tiene un editor inteligente para completar código, otra característica notable es que tiene la posibilidad de *refactorizar*, esto es modificar el código sin comprometer la ejecución del mismo [46].

2. METODOLOGÍA

En este capítulo será descrito el proceso seguido para la generación del modelo de reconocimiento facial, el cual es obtenido por la técnica llamada aprendizaje por transferencia y cuyo procedimiento está representado en el diagrama de bloques de la figura 2.1. Para esto se grabaron varios videos por cada uno de los individuos, posteriormente, se aplicó la técnica mencionada al modelo de reconocimiento facial VGGFace, el cual a su vez toma la base de datos formada por los rostros extraídos de los videos grabados. Para la elección del mejor modelo fue utilizada la técnica de validación cruzada k-fold (*k-fold cross validation*). Una vez obtenido el modelo se procede a realizar las predicciones, para lo cual se emplean videos que en este caso contienen a múltiples personas. Dichas predicciones se realizan sobre cada fotograma y posteriormente se someten a un algoritmo de coherencia temporal, el cual utiliza ventanas deslizantes conformadas por las predicciones obtenidas de n fotogramas y cuya finalidad es la de aumentar la precisión de las predicciones conforme la ventana incrementa su tamaño. Finalmente, se obtiene las predicciones resultantes indicando la identidad de las personas que se encuentran dentro del video.

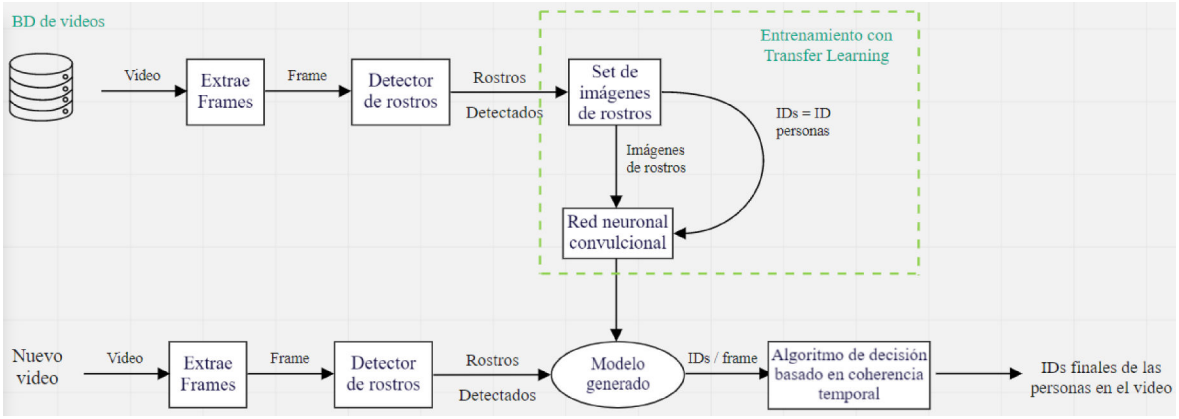

Figura 2.1: Diagrama de bloques del proceso de extracción de rostros, entrenamiento del modelo y procesos de pruebas con un nuevo video.

2.1. BASE DE DATOS

Para realizar la fase de entrenamiento del modelo se inició recopilando una base de datos de 200 videos previamente etiquetados en los cuales aparecen 20 personas, conformadas por 13 hombres y 7 mujeres. Para cada uno de los individuos se han grabado 10 videos con una duración de entre 10 a 15 segundos en formato mp4, con resolución 1080p y a 30 fotogramas por segundo (FPS).

Enseguida se procedió a realizar el siguiente proceso para extraer los rostros de los fotogramas de cada video:

1. Se extrajo los fotogramas que conforman los vídeos y fueron almacenados en una carpeta, de los cuales algunos se muestran en la figura 2.4. En el segmento de código

2.1 se muestra el proceso realizado para cada video y es representado en el esquema de la figura 2.2.

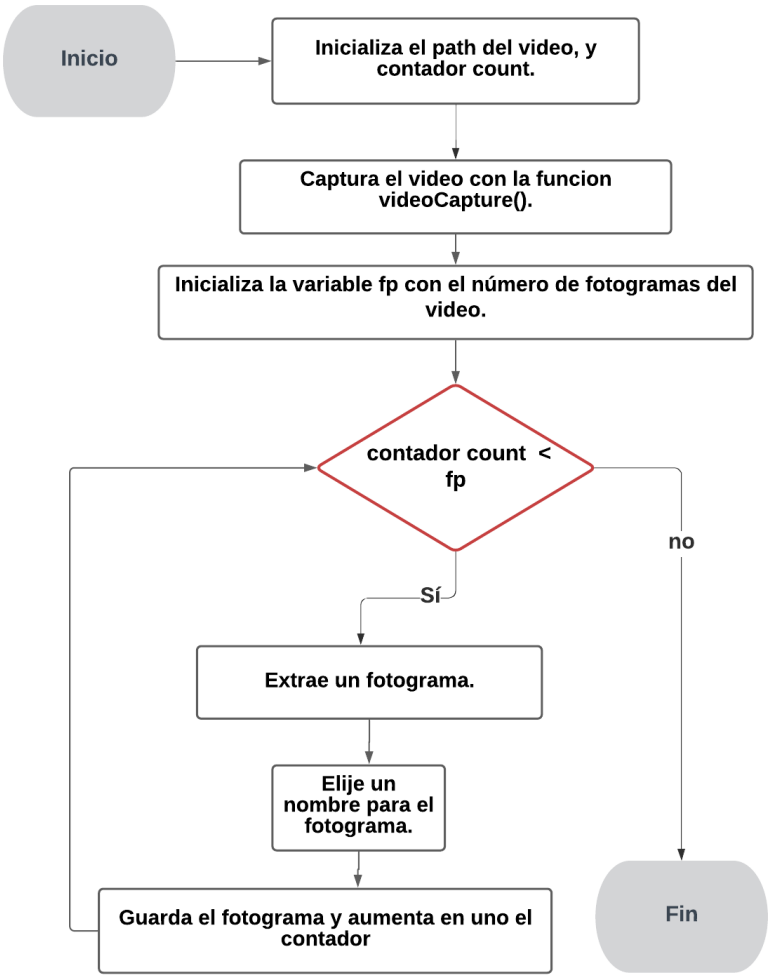

Figura 2.2: Esquema del proceso de extracción de fotogramas.

Segmento de código 2.1: Código para extraer un fotograma.

```
#importamos las librerías necesarias
import os
import cv2
#variable que se inicializa con el directorio del video
pathVideo= C:\\P01_V01.mp4
#captura el video
vidcap = cv2.VideoCapture(pathVideo)
success,frame = vidcap.read()  #captura un fotograma del video
nombre= "prueba"  # nombre de la imagen
cv2.imwrite("G:\\frames\\%s.jpg" %nombre, frame) # finalmente guarda el ...
    fotograma
```

2. Se usó la red neuronal MTCNN para extraer los rostros encontrados en cada fotograma, de los cuales algunos se pueden observar en la figura 2.5. El código que realiza dicho proceso para cada fotograma se muestra en los segmentos 2.2 y 2.3 que están representados por el esquema de la figura 2.3.

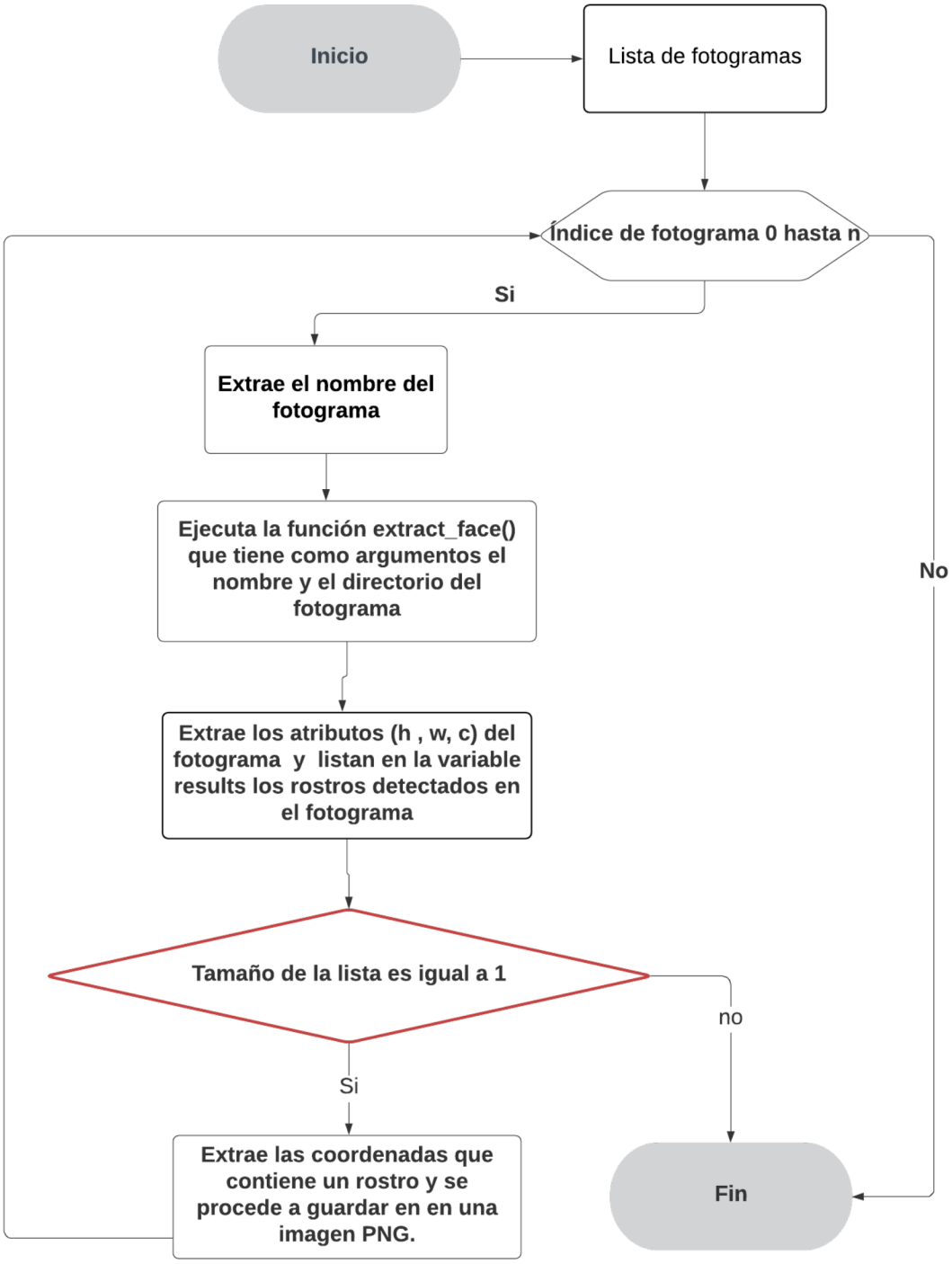

Figura 2.3: Esquema del proceso de extracción de rostros detectados en fotogramas.

Segmento de código 2.2: Código para instanciar el detector MTCNN y detectar rostros en un fotograma.

```
#importamos las siguientes librerias
from matplotlib import pyplot
from PIL import Image
from numpy import asarray
from mtcnn.mtcnn import MTCNN
#import _builtin_
import cv2
pixels = pyplot.imread(filename)#carga el fotograma
detector = MTCNN() # carga el detector
results = detector.detect_faces(pixels)# detecta los rostros dentro del ...
    fotograma y lo guarda en una lista llamada results
```

3. Se normalizaron las imágenes de los rostros extraídos. Luego estas imágenes fueron almacenadas en una carpeta individual que identifica a la persona y video. En la figura 2.6 se observa las carpetas individuales. El código que realiza este proceso está en el segmento de código 2.3.

Segmento de código 2.3: Código para obtener el recuadro de coordenadas de píxeles que contiene la imagen y normalización de tamaño.

```
x1, y1, width, height = results[0]['box'] #Coordenadas de un rostro ...
    detectado
x2, y2 = x1 + width, y1 + height #Coordenadas de la esquina superior ...
    derecha del rostro
face = pixels[y1:y2, x1:x2]# Matriz de píxeles que delimita un rostro
image = Image.fromarray(face)
image = image.resize((224,224))# normalización el tamaño del rostro a ...
    224,224
face_array = asarray(image) #convierte en array la imagen
nombreVideo = os.path.splitext(os.path.basename(myFile))[0][:7]# ...
    extrae nombre del video
nombreFrame=os.path.splitext(os.path.basename(filename))[0] # extrae ...
    nombre del frame
cv2.imwrite("G:\\carasRecortadas\\%s\\%s.png" ...
    %(nombreVideo,nombreFrame), face_array)# guarda el rostro
```

Otra forma de guardar la imagen es utilizando la función `save()` de la librería PIL, así:

```
im = image.save("G:\\carasRecortadas\\%s\\%s.png" %(nombreVideo, ...
    nombreFrame))
```

De esta manera evitamos la línea de código número 6.

Una vez normalizadas las imágenes, estas son guardadas en formato PNG porque es un formato con compresión sin pérdida.

(a) P18-V01-21 (b) P19-V03-56 (c) P20-V01-58

Figura 2.4: Imágenes de fotogramas para creación de datos de videos.

(a) P18-V01-22 (b) P19-V03-56 (c) P20-V01-56

Figura 2.5: Imágenes de rostros recortados de los fotogramas.

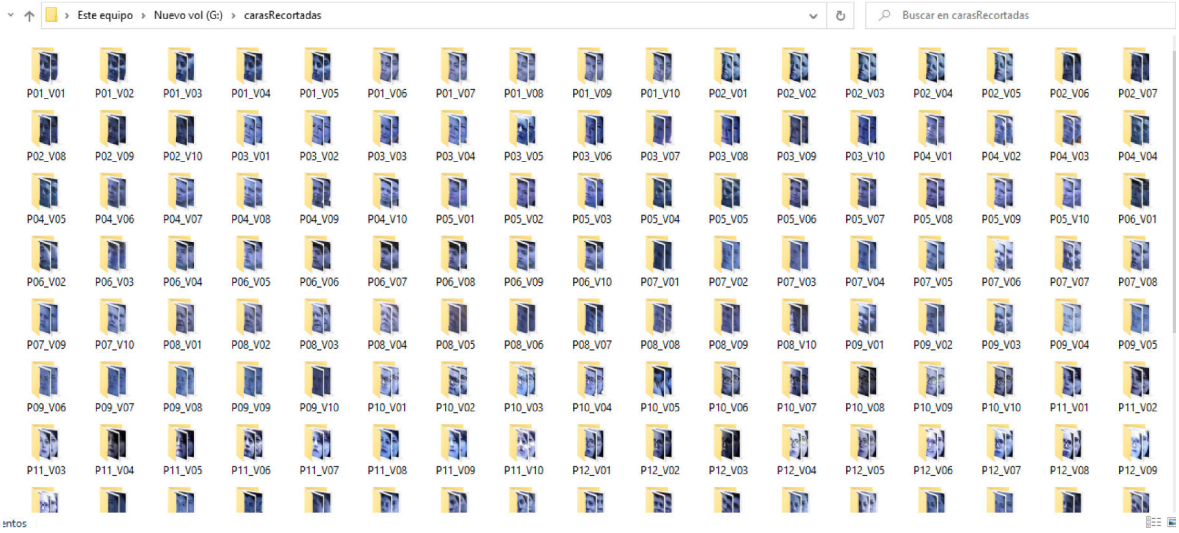

Figura 2.6: Base de datos de rostros divididos por el número de video de cada persona.

El script que se utilizó para obtener la base de datos está en el anexo B, tiene el nombre de *ExtraccionDeRostrosDeLosVideos.ipynb*.

2.2. RED NEURONAL CONVOLUCIONAL

Utilizamos la red neuronal VggFace de la figura 2.8 como base para la creación del nuevo modelo. Las principales capas que forman esta red neuronal son las capas *convolucionales* y de agrupación máxima (*MaxPolling*). La capa *convolucional* es un pilar fundamental para construir una CNN, la parte fundamental de una capa convolucional es implementar la operación de convolución de una imagen de entrada con un filtro para extraer características de estas imágenes de entrada. Por otra parte, una capa de agrupación máxima *MaxPolling* detecta múltiples características de las imágenes trasladándose sobre las imágenes como un filtro para elegir el máximo valor, además de mantener la relación espacial de la imagen. Entonces, tomando como punto de partida esta red neuronal aplicamos la técnica llamada *aprendizaje por transferencia* con la intención de obtener una nueva red. Para esto, se congelaron todas las capas de la red original. En este trabajo se agregó una capa de *Aplanamiento* que se encarga de convertir los elementos de la matriz que representa una imagen a un vector plano, es decir, de una sola dimensión. También se agregó una capa *Dropout* que ignora aleatoriamente algunas neuronas en el proceso de entrenamiento, lo cual ayuda a disminuir el sobre entrenamiento del modelo a ser generado. Finalmente, se agregó la capa *Dense*. Esta última se inicializa con 20 neuronas, puesto que queremos clasificar 20 rostros de personas y utiliza la función de activación multivariable Softmax. De este modo logramos una nueva red neuronal como se puede observar en la figura 2.9.
El segmento de código 2.4 fue utilizado para el proceso anterior descrito y es representado por el esquema de la figura 2.7.

Segmento de código 2.4: Importación de librerías necesarias.

```
# importamos las siguientes librerías
import keras
from keras import applications
import keras_vggface
from keras import optimizers
import numpy as np
from keras_vggface.vggface import VGGFace
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
from keras.models import Sequential, Model
 # normalizamos el tamaño de imágenes de entrada
img_width=224
img_height=224
#cargamos el modelo inicial
model = VGGFace(weights = "vggface", include_top=False, input_shape = ...
    (img_width, img_height,3))
#congelamos las capas del modelo de red neuronal convolucional inicial
```

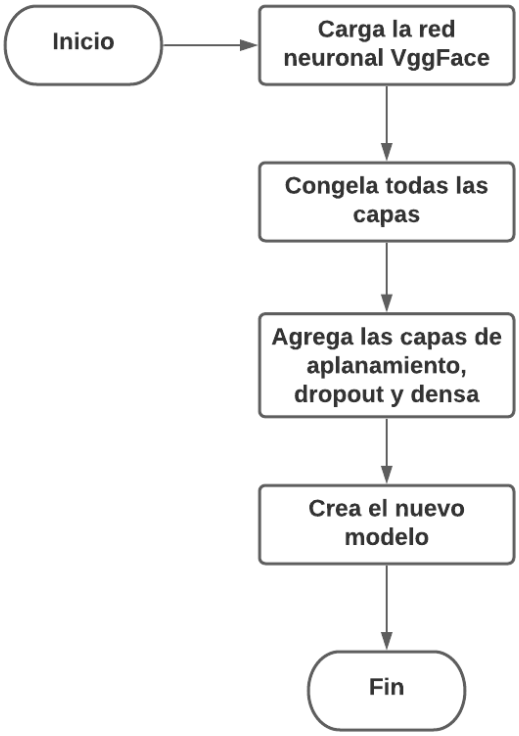

Figura 2.7: Esquema que añade capas a el modelo.

```
for layer in model.layers[:]:
  layer.trainable = False
# añadimos capas
x = model.output
x = Flatten()(x)
x = Dropout(0.25)(x)
predictions = Dense(20, activation="softmax")(x)
#creamos el modelo final
model_final = Model(inputs = model.input, outputs = predictions)
```

A continuación, se mencionan las configuraciones realizadas para el entrenamiento de la red, las cuales son representadas con el esquema de la figura 2.10:

1. Se importa las librerías necesarias, como se observa en el segmento de código 2.5:

Segmento de código 2.5: Librerías necesarias para el proceso de entrenamiento.

```
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras import optimizers
from keras.models import Sequential, Model
from keras import backend as k
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, ...
    TensorBoard, EarlyStopping, ReduceLROnPlateau
```

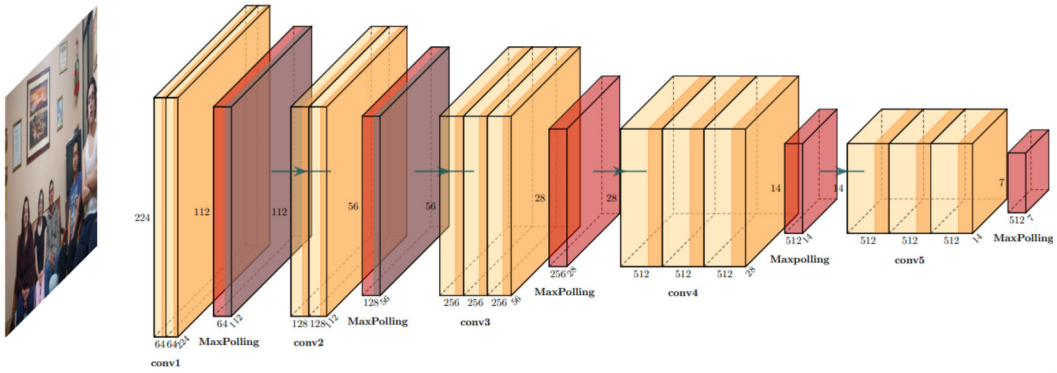

Figura 2.8: Modelo VggFace

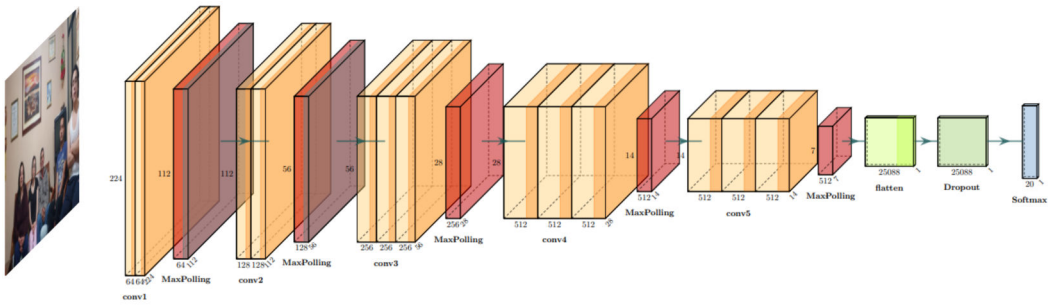

Figura 2.9: Modelo nuevo basado en el modelo VggFace

```
import numpy as np
from keras.callbacks import EarlyStopping
import itertools
```

2. Se colocan los directorios de las carpetas de entrenamiento y validación, además de especificar número de elementos que tiene cada directorio y la cantidad de imágenes que son utilizadas en cada iteración. Con esta información se obtiene la cantidad de iteraciones que son representadas por los parámetros `val_steps` y `train_steps`, que luego sirven para procesar las carpetas `TRAIN` y `VALIDATION`. Este proceso se hizo con el código 2.6:

Segmento de código 2.6: Código que detalla la información de entrenamiento y validación.

```
# directorios de entrenamiento y validación
train_data_dir = '/home/TRAIN'
validation_data_dir = '/home/VALIDATION'
#elementos de cada directorio
nb_train_samples = 27320
nb_validation_samples = 9694
# cantidad de imágenes que fueron procesadas en cada iteración
batch_size = 300
#número de iteraciones necesarias para procesar el directorio de ...
    entrenamiento y validación respectivamente en cada época.
train_steps = np.ceil(nb_train_samples / batch_size)
```

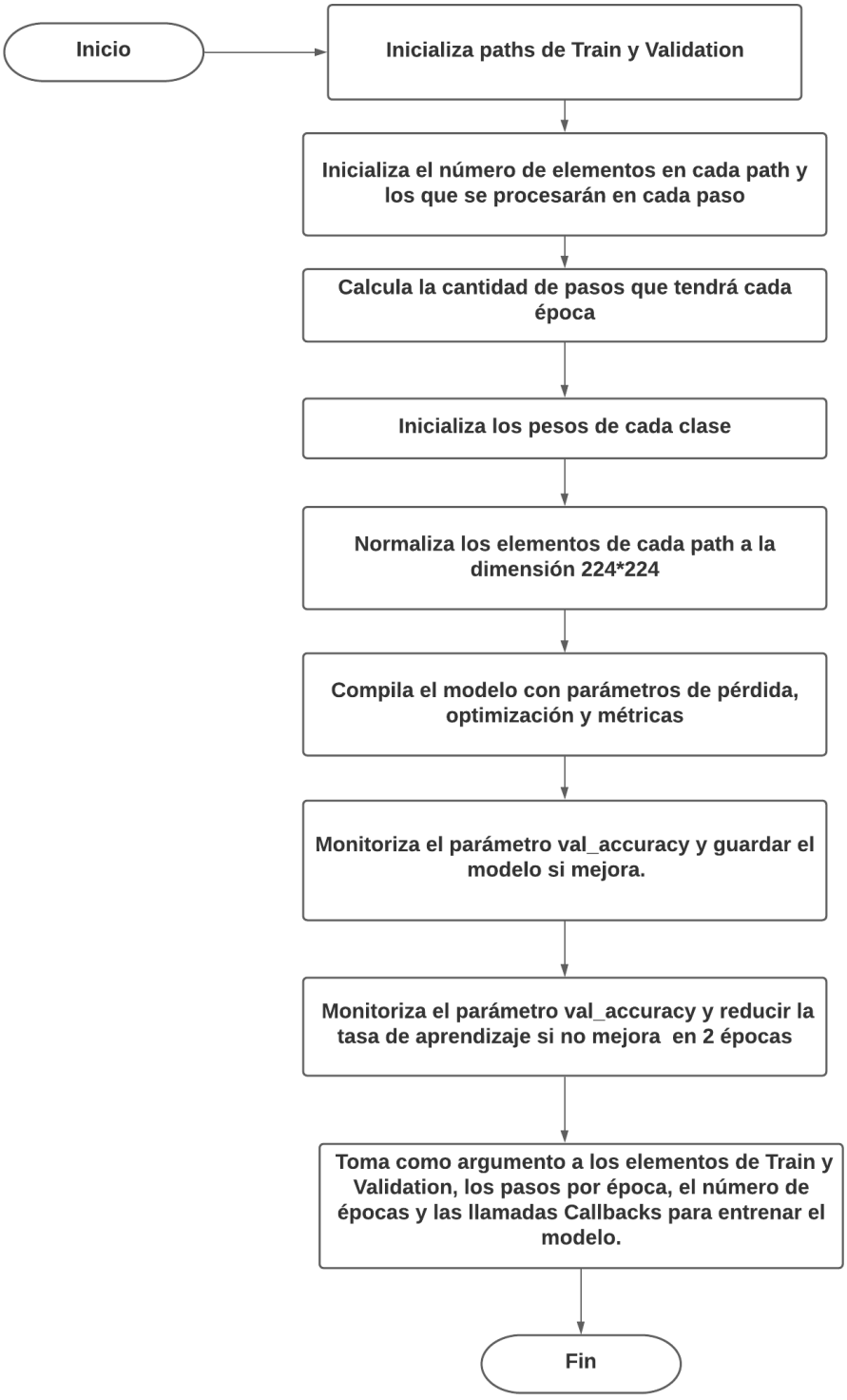

Figura 2.10: Esquema del entrenamiento del modelo.

```
11  val_steps = np.ceil(nb_validation_samples / batch_size)
```

3. Se procede a inicializar los pesos de clase en 1, ya que la base de datos es balanceada, tal como se observa en el segmento de código 2.7.

Segmento de código 2.7: Inicialización de pesos de cada clase.

```
class_weights={
  0: 1.0,  # P01
  1: 1.0,  # P02
  2: 1.0,  # P03
  3: 1.0,  # P04
  4: 1.0,  # P05
  5: 1.0,  # P06
  6: 1.0,  # P07
  7: 1.0,  # P08
  8: 1.0,  # P09
  9: 1.0,  # P10
  10: 1.0,  # P11
  11: 1.0,  # P12
  12: 1.0,  # P13
  13: 1.0,  # P14
  14: 1.0,  # P15
  15: 1.0,  # P16
  16: 1.0,  # P17
  17: 1.0,  # P18
  18: 1.0,  # P19
  19: 1.0,  # P20
  }
```

4. A continuación, se toma las imágenes de entrenamiento y validación para así normalizar el tamaño a través del parámetro `target_size`, además de inicializar el tamaño de los lotes representado por el parámetro `batch_size` e indicar el orden de las imágenes con el parámetro `suffle=True`. Estas configuraciones son realizadas con el segmento de código 2.8:

Segmento de código 2.8: Normalización de dimensiones y tamaño de lotes de imágenes de los directorios de entrenamiento y validación para su procesamiento.

```
  # función que se aplicara a cada entrada después de ser normalizada
  train_batches = ImageDataGenerator(
  preprocessing_function= \
  keras.applications.vgg16.preprocess_input).flow_from_directory(
  #directorio Train
  train_data_dir,
  #tamaño normalizado
  target_size=(img_width, img_height),
  #tamanio de lote de imagen es por iteración
  batch_size=batch_size,
  #orden de las imagenes
  shuffle=True)
  valid_batches = ImageDataGenerator(
  preprocessing_function= \
  keras.applications.vgg16.preprocess_input).flow_from_directory(
```

```
16      # directorio Validation
17      validation_data_dir,
18      target_size=(img_width, img_height),
19      batch_size=batch_size,
20      shuffle=True)
```

5. Se compila la nueva red neuronal mediante los siguientes parámetros:

   - La función de pérdida `"categorical_crossentropy"`, la cual es utilizada para tareas de clasificación de múltiples categorías.
   - El optimizador de gradiente descendiente `SGD` con una tasa de aprendizaje `lr`, que representa cuánto deben cambiar los parámetros en cada iteración, además de un `momentum` destinado para acelerar el descenso del gradiente. Este tipo de optimizadores son empleados para mejorar el rendimiento y la velocidad de entrenamiento de un modelo.
   - Finalmente, la métrica `"accuracy"` para monitorizar el proceso de aprendizaje de la red.

   Esto fue realizado con el segmento de código 2.9.

Segmento de código 2.9: Configuración del modelo para el entrenamiento.

```
1 #configuración del modelo, pérdidas y métricas
2 model_final.compile(loss = "categorical_crossentropy", optimizer = ...
      optimizers.SGD(lr=0.0001, momentum=0.9), metrics=["accuracy"])
```

6. Se procede a elegir un path para guardar el modelo generado y, a través de la función `ModelChekpoint`, se guarda dicho modelo tomando en cuenta el `'val_accuracy'` como el valor de monitor.

   Se emplea el argumento `save_best_only` para comparar el `val_accuracy` obtenido en la época actual con el de la anterior. Si el actual supera al anterior, entonces se sobrescribirá un nuevo modelo, caso contrario, seguirá con el resultado de épocas pasadas.

   El argumento `save_weights_only` es inicializado con `False` debido a que no se requiere guardar solo los pesos sino el modelo completo.

   El argumento `mode` se inicializa con `'max'` ya que se necesita que la métrica monitorizada, es decir el `val_accuracy`, sea el máximo.

   Finalmente, el argumento `period` es inicializado en `1` para verificar los resultados de acuerdo al valor monitorizado y, de acuerdo a eso, guarde o no el nuevo modelo una vez acabada cada época procesada.

   Lo anteriormente mencionado se configuró con el siguiente segmento de código 2.10:

Segmento de código 2.10: Configuración de retro llamada *(Callback)* para guardar el nuevo modelo generado.

```
#directorio donde se guarda el modelo
filepath = "/content/drive/My Drive/colab/modelf1.h5"
#retro llamada utilizada para guardar el modelo o los pesos al ...
    finalizar una época de entrenamiento tomando en cuenta la variable ...
    monitor
checkpoint = ModelCheckpoint(filepath, monitor='val_accuracy', ...
    verbose=1, save_best_only=True, save_weights_only=False, ...
    mode='max', period=1)
```

7. Para manejar una tasa de aprendizaje variable y reducirla fue utilizada la función `ReduceLROnPlateau()`. Esta función inicializa el parámetro `patience` que va a describir el número de épocas esperadas para reducir o no la tasa de aprendizaje. En el presente caso se inicializó en 2 épocas. El parámetro `factor`, que representa el factor de reducción de la tasa de aprendizaje, se inicializó con 0.5. Esto significa que será reducido a la mitad de tasa de aprendizaje actual. Para el parámetro `monitor`, que representa la característica que será analizada para reducir o no la tasa de aprendizaje, se asignó el `'val_accuracy'`. Dicho valor representa un conjunto de validación apropiado. El argumento `mode` se inicializó con `'max'` ya que se necesita que la métrica monitorizada, es decir, el `val_accuracy` sea el máximo. Y finalmente, el parámetro `min_lr`, representa el límite inferior de la tasa de aprendizaje.

   Esto fue configurado con el segmento de código 2.11:

Segmento de código 2.11: Variación de la tasa de aprendizaje.

```
reduce_lr = ReduceLROnPlateau(monitor='val_accuracy', factor=0.5, ...
    patience=2, verbose=1, mode='max', min_lr=0.000001)
```

8. Finalmente, para comenzar el entrenamiento se utiliza la función `fit_generator` inicializando los siguientes parámetros: `train_batches y validation_data` que, respectivamente, toman las imágenes de entrenamiento `train_batches` y las imágenes de validación `valid_batches` que están en el segmento de código 2.8. Los pasos a ser utilizados para procesar las imágenes de entrenamiento y validación toman el valor de `train_steps` y `val_steps` del segmento de código 2.6. Los pesos de cada clase son tomados del segmento de código 2.7. En este caso se utilizaron 10 épocas de entrenamiento controlado por el parámetro `epochs`, pues de este modo se asegura una exactitud alta del modelo generado. Los pasos de validación se toman de `val_steps`. Por último, para las retro llamadas `callbacks`, que son funciones que comparan métricas de la época actual con las anteriores, en este caso se utilizó el `checkpoint y reduce_lr` con la finalidad de guardar el modelo y reducir la tasa de entrenamiento, respectivamente.

Estos parámetros se configuran mediante el segmento de código 2.12.

Segmento de código 2.12: Configuración de parámetros para entrenar el modelo.

```
history = model_final.fit_generator(
train_batches,
steps_per_epoch = train_steps,
class_weight=class_weights,
epochs = 10,
validation_data = valid_batches,
validation_steps = val_steps,
callbacks = [checkpoint, reduce_lr])
```

Al finalizar el entrenamiento se obtuvo un modelo resultante. El script completo de entrenamiento se encuentra en el anexo B, con el nombre *entrenarModeloNuevo.ipynb*.

El entrenamiento se hizo utilizando 5 combinaciones de los conjuntos de entrenamiento y validación. Estas combinaciones se crearon a partir de la técnica llamada *validación cruzada k-folds*, que se explica en la sección 2.4. Una vez realizado el mencionado proceso, se obtuvieron 5 modelos resultantes, de los cuales se eligió el de mayor exactitud.
Con el modelo elegido se logró realizar las predicciones de videos de múltiples personas y cuyo proceso esta representado por el esquema de la figura 2.11, para lo cual se sigue el siguiente proceso:

1. Se realiza la importación de algunas librerías que servirán para poder realizar las predicciones de los fotogramas de un video, que se hizo con el segmento de código 2.13:

Segmento de código 2.13: Librerías utilizadas para realizar predicciones.

```
import keras
from matplotlib import pyplot
from PIL import Image
from numpy import asarray
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array
from mtcnn.mtcnn import MTCNN
import cv2
import glob
from keras_vggface import utils
import os
import shutil
import numpy as np
import pickle
```

2. Luego se cargó el modelo resultante, haciendo uso de la siguiente línea de código 2.14.

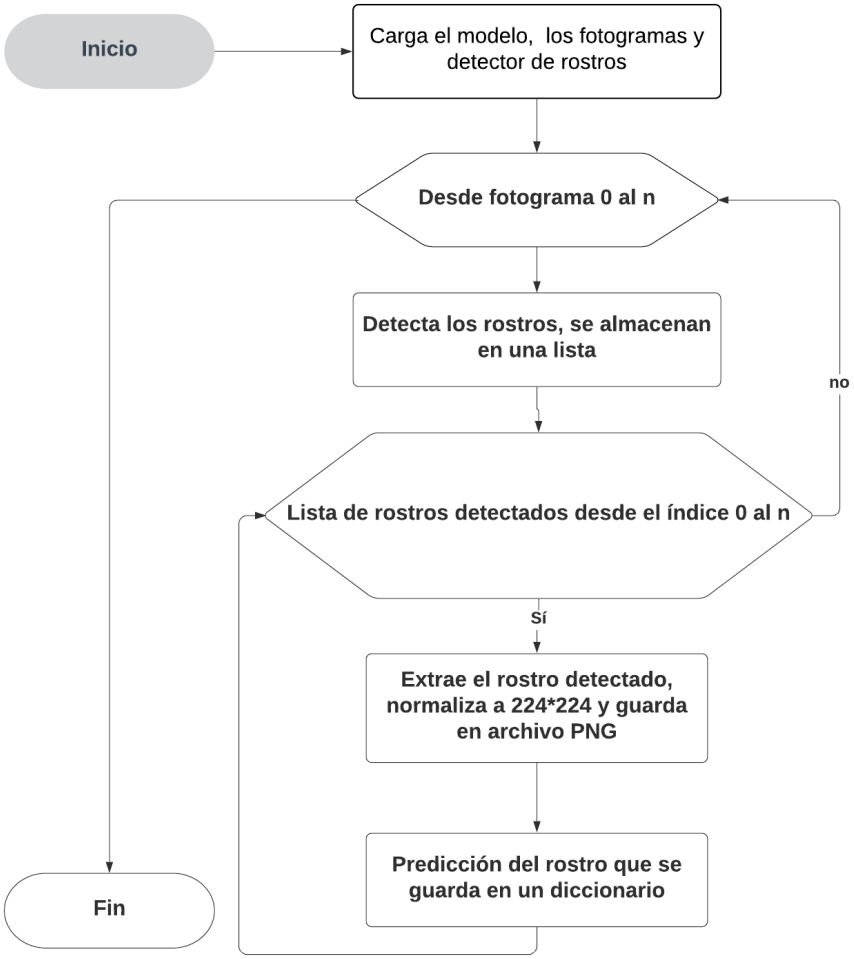

Figura 2.11: Esquema de predicción de fotogramas.

Segmento de código 2.14: Código para cargar modelo.

```
1 new_model = keras.models.load_model('/content/drive/MyDrive/modelf2.h5')
```

3. Se procede a listar las imágenes con el código 2.15:

Segmento de código 2.15: Código que lista los fotogramas.

```
1 files = glob.glob("/content/drive/My Drive/todos/famil02/*.jpg")
```

4. Se toma un elemento de la lista de paths, representada por la variable `files` que hace referencia a los fotogramas del video. A continuación, se extrae el nombre con la variable `nombre`, para así leer la imagen y extraer tanto las dimensiones como el número de canales. Esto se realizó con el código 2.16:

Segmento de código 2.16: Lectura de la imagen y extracción de dimensiones y canales de la misma.

```
nombre=os.path.splitext(os.path.basename(myFile))[0]# extracción del ...
    nombre de la imagen
pixels = pyplot.imread(myFile) #lectura de la imagen a un array
h, w, c = pixels.shape   # extracción de dimensiones y canales
```

5. Luego, es instanciado el detector de rostros. Esto se realiza con la siguiente línea de código 2.17:

Segmento de código 2.17: Instancia del detector de rostros.

```
detector = MTCNN() # Instanciamos el detector de rostros
```

6. Se procede a detectar los rostros dentro de la imagen. Los rostros encontrados son guardados en una lista llamada `results`. Dicho proceso se realiza con el código 2.18:

Segmento de código 2.18: Línea de código para detectar rostros.

```
results = detector.detect_faces(pixels)
```

7. Seguidamente, fue seleccionado un elemento de la lista `results` que representa un rostro detectado del cual fueron extraídas las coordenadas (`x1`, `y1`) y dimensiones (`width`, `height`). Las coordenadas hacen referencia a la posición del rostro detectado dentro del fotograma. Al sumar (`x1 + width`) se obtiene `x2`, mientras que al sumar (`y1 + height`) se obtiene `y2`. Con las coordenadas `x1, x2, y1 y y2` se recorta la sección que contiene el rostro detectado en el fotograma. Esto fue realizado con el código 2.19:

Segmento de código 2.19: Extracción de dimensiones del rectángulo que rodea al rostro detectado.

```
# seleccionamos el elemento 0 de la lista y tomamos la clave 'box'
x1, y1, width, height = results[0]['box']
#con las dimensiones extraídas creamos las otras coordenadas del ...
    recuadro del rostro
x2, y2 = x1 + width, y1 + height
# el rostro esta limitado por
face = pixels[y1:y2, x1:x2]
```

8. Se cargan los píxeles que contienen el rostro en una imagen para así ser redimensionado y posteriormente convertirlo en una matriz. Esto se realizó con el código 2.20:

Segmento de código 2.20: Normalización de la imagen del rostro detectado.

```
#transforma un array a imagen
img = Image.fromarray(face)
#normalización de la imagen
img = img.resize((224,224))
# convertir la imagen PIL en una matriz (array)
face_array = asarray(img)
```

9. Se crea un diccionario para guardar las predicciones, procedimiento realizado con el código 2.21:

Segmento de código 2.21: Creación de diccionario temporal por cada fotograma.

```
  if (type (nombre)is dict):
    print("ya esta creado el diccionario")
  else:
    nombre={}
```

10. Se procede a guardar la imagen en formato PNG, para posteriormente instanciar el path con la variable `test_data_dir`. Esto se realizó con el código 2.22:

Segmento de código 2.22: Código para guardar la imagen del rostro detectado e instanciado.

```
cv2.imwrite("/content/drive/My Drive/todos/recorteFamil02/%s_%d.png" %...
    (os.path.splitext(os.path.basename(filename))[0],countk),face_array)
    # carga el directorio de la imagen
test_data_dir = "/content/drive/My ...
    Drive/todos/recorteFamil02/%s_%d.png" %...
    (os.path.splitext(os.path.basename(filename))[0],countk)
```

11. Se carga la imagen que contiene el rostro, preparándola para ingresar al proceso de predicción. Esto implica agregar una dimensión a la matriz que representa la imagen, cumpliendo así con la matriz de entrada que tendrá 4 dimensiones. Concretamente, muestras, filas, columnas y canales, que sirven para obtener una matriz con dimensiones similares a `(1, 224, 224, 3)`. Finalmente, se procesa esta matriz a través de dicho modelo, obteniendo así las predicciones. Esto se realizó con el código 2.23:

Segmento de código 2.23: Código que carga la imagen para prepararlo para que el modelo haga la predicción.

```
    # cargar imagen desde el archivo
    image = load_img(test_data_dir, target_size=(224, 224))
    # remodelar los datos para el modelo
    x = img_to_array(image)
```

```
    # preparar la imagen para el modelo VGG
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
      # predecir la probabilidad en todas las clases de salida
    preds = new_model.predict(x)
```

12. Se transforman las predicciones obtenidas en una lista, para obtener así el índice de la mayor probabilidad. De acuerdo a dichos índices, se agrega la predicción al diccionario que en el presente caso ha sido llamado *nombre*, utilizando como clave el propio índice para guardar la matriz de predicciones.

    Esto se hizo con el código 2.24:

Segmento de código 2.24: Código que transforma la matriz de predicción en una lista y agrega un nuevo elemento para el diccionario llamado *nombre*.

```
    # transforma un array a lista
    lisk= preds.tolist()
    #toma el elemento maximo de la lista
    indice=lisk[0].index(max(lisk[0]))
    #guardo la prediccion en el diccionario nombre
    nombre[indice]= preds[0]
```

13. Finalmente, se guarda el diccionario de cada fotograma en archivos pickle, que después serán utilizados para aplicar la coherencia temporal en todos los fotogramas del video.

    Esto se realizó con el código 2.25:

Segmento de código 2.25: Código que guarda los archivos pickle.

```
#crea un archivo pickle
pickle_file = open('/content/drive/My ...
    Drive/todos/%s.pickle'%(os.path.splitext(os.path.basename(myFile))[0]) ...
    , 'wb')
#guarda el diccionario nombre en el archivo pickle creado
pickle.dump(nombre, pickle_file)
pickle_file.close()
```

Una vez que se procesaron todos los fotogramas del vídeo, se obtuvo un archivo pickle por fotograma, el mismo que contiene las predicciones de cada rostro detectado. Es decir, al tener N fotogramas, se pudo obtener como máximo N archivos pickle que en su interior contienen un diccionario con una o más predicciones. Estos archivos pickle se utilizaron en la siguiente sección, donde se aplicó la coherencia temporal en todos los fotogramas del video.

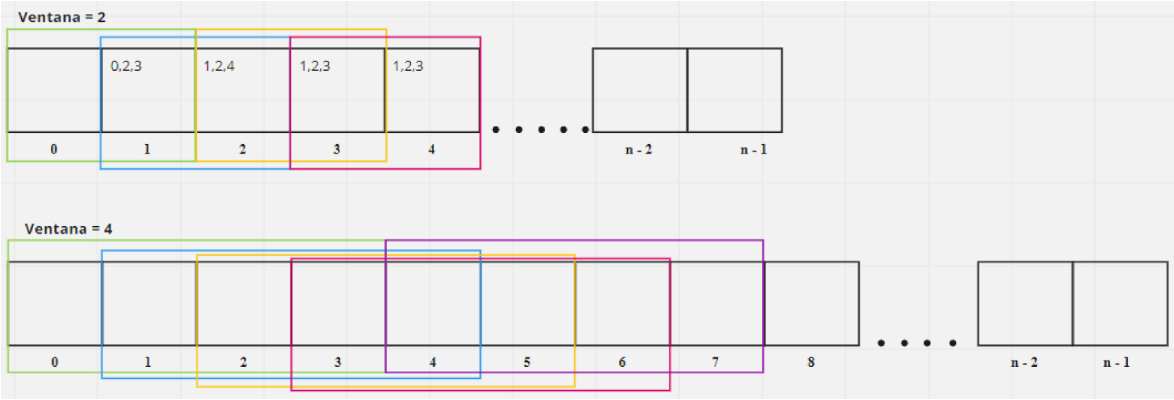

Figura 2.12: Ventana deslizante

El script que se utilizó para realizar las predicciones en videos de múltiples personas está en el Anexo C y se llama *PrediccionesVideoCompletoMultiplesPersonas.ipynb*.

2.3. RECONOCIMIENTO USANDO COHERENCIA TEMPORAL

Una vez realizadas las predicciones de todos los fotogramas del video múltiple, se procede a utilizarlas en el algoritmo de coherencia temporal que usa ventanas deslizantes a lo largo del video para predecir si una persona está o no está en el video, dicho algoritmo toma grupos de predicciones de fotogramas consecutivos de acuerdo con la ventana elegida como se muestra en la figura 2.12.
Este algoritmo es representado por el esquema de la figura 2.13 y será explicado con un ejemplo de predicciones en un video que contiene a 3 personas. A este video se le aplicó una `ventana`=2. A continuación se explica paso a paso el código utilizado:

1. Se importan las librerías necesarias con el segmento de código 2.26:

Segmento de código 2.26: Librerías necesarias para utilizar la coherencia temporal.

```
1    import pickle
2    import glob
3    import numpy as np
4    import os
```

2. A continuación, se inicializan las variables globales que servirán para desarrollar el algoritmo con el segmento de código 2.27:

Segmento de código 2.27: Inicialización de variables.

```
1    # Número de fotogramas que se tomaran para analizar sus predicciones
2    ventana = 2
3    # Número de personas con las que fue entrenado el modelo
```

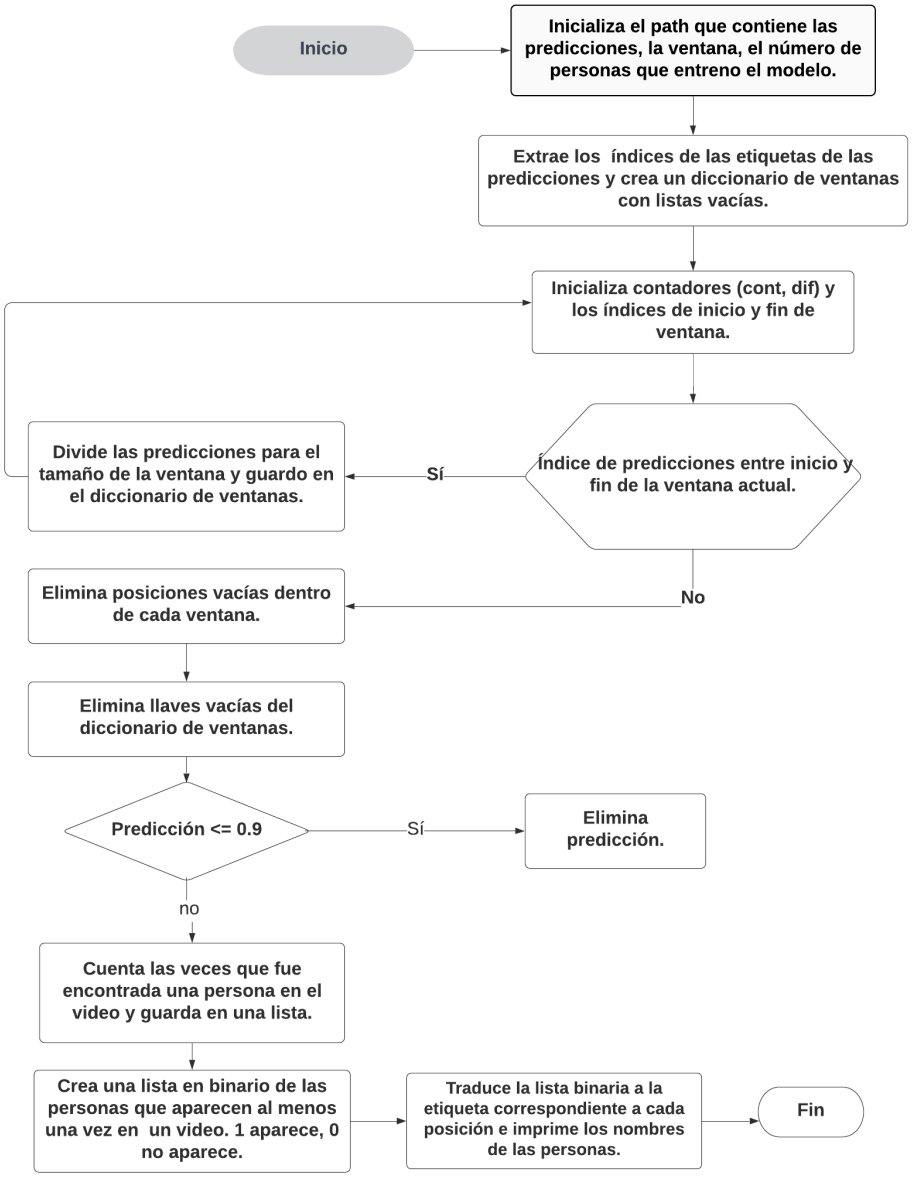

Figura 2.13: Esquema de predicción con coherencia temporal.

```
4       npersonass=20
5       # contador que se utiliza para verificar que cada ventana tenga la ...
            misma cantidad de elementos
6       cont=0
7       #contador que se utiliza para verificar que cada ventana sea del ...
            mismo tamaño
8       dif=0
9       #variable que representa el índice inicial del fotograma de la ...
            primera ventana
10      inicio=0
11      #variable que representa el índice final del fotograma de la ...
            primera ventana
12      fin=ventana
```

```
13      nvideo = 'Buitron01' #nombre del video
14      #directorio donde se encuentran las predicciones
15      path='/content/drive/My Drive/todos/predicBuitron'
16      #diccionario donde se guardan las predicciones resultantes de las ...
            ventanas
17      dicVentana={}
```

3. Después se crean tantos keys como archivos pickle existan y se inicializa el diccionario de ventanas con listas vacías utilizando el segmento de código 2.28:

Segmento de código 2.28: Creación de elementos dentro de un diccionario de ventanas.

```
1       #lista de archivos pickle
2       ndicc= glob.glob("%s/*.pickle" %path)
3       for i in range (0,(len(ndicc))):
4           dicVentana[i]=[]
```

Al imprimir dicho diccionario se observa esta salida:

```
{0: [], 1: [], 2: [], ........, 307: [], 308: []}
```

4. Se buscan los archivos de predicciones de los fotogramas que conforman la actual ventana para cargar las predicciones. Esto se hizo con el segmento de código 2.29:

Segmento de código 2.29: Verificación de archivos de predicción acorde al fotograma.

```
1       if os.path.isfile('%s/%s_%s.pickle' %(path,nvideo,b)):  ...
            #verificamos si existen los archivos pickle
2           q = open('%s/%s_%s.pickle' %(path,nvideo,b),'rb')  #Abrimos ...
                este archivo
3           dic = pickle.load(q)  # cargamos el diccionario que esta dentro ...
                del archivo
```

Debido a que se utiliza una ventana igual a 2, la primera de estas está conformada por las predicciones del fotograma `0` y `1`. Al imprimir las predicciones de cada fotograma se muestra 2.30 y 2.31:

Segmento de código 2.30: Predicciones del fotograma 0.

```
1           {3: array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
2            0., 0., 0., 0., 0., 0., 0.], dtype=float32), 5: array([0., 0.,
3            0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
4           0., 0., 0.], dtype=float32), 4: array([0., 0., 0., 0., 1., 0.,
5           0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
6           0., 0., 0.], dtype=float32)}
```

Segmento de código 2.31: Predicciones del fotograma 1.

```
{3: array([0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., ...
    0., 0., 0., 0.,
0., 0., 0.], dtype=float32), 5: array([0., 0., 0., 0., 0., 1., ...
    0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0.], dtype=float32), 4: array([0., 0., 0., 0., 1., 0., ...
    0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
0., 0., 0.], dtype=float32)}
```

Se puede observar que en las predicciones de los fotogramas `0 y 1`, se encuentran contenidas las keys `3, 4 y 5`.

5. Se procede a inicializar la primera ventana con una cantidad de ceros igual a la cantidad de personas con las que fue entrenado el modelo, en este caso, 20. Esto se realiza con el código 2.32:

Segmento de código 2.32: Inicialización de una lista de 0s de cada elemento del diccionario.

```
        for c in range(0,npersonass ):
            dicVentana[a].append(0)
```

Al imprimir el diccionario principal de ventanas se observa lo siguiente:

```
{0: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
....., 306: [], 307: [], 308: []}
```

6. Se realiza un promedio entre las probabilidades que tienen los fotogramas 0 y 1 de acuerdo con las keys encontradas dentro de cada predicción. Dicho resultado es guardado en la posición correspondiente de la ventana dentro del diccionario de ventanas *dicVentana* de acuerdo con las keys de los fotogramas. Esto se realiza con el código 2.33:

Segmento de código 2.33: Promedio de predicciones tomando en cuenta la posición.

```
        dicVentana[0][c] += (dic[c])/ventana
```

En el código anterior, `c` representa la posición dentro de la ventana. Al imprimir estos resultados se muestra lo siguiente:

```
{0: [0, 0, 0, array([0. , 0. , 0. , 1 , 0. , 0. , 0. , 0. , 0. , 0.
, 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. ], dtype=float32),
array([0. , 0. , 0. , 0. , 1 , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.
```

```
, 0. , 0. , 0. , 0. , 0. , 0. , 0. ], dtype=float32), array([0. , 0.
, 0. , 0. , 0. , 1 , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0. , 0.
, 0. , 0. , 0. , 0. ], dtype=float32), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0], 1: [], 2: [],....., 305: [], 306: [], 307: [], 308: []}
```

Las predicciones son los arrays presentes en la ventana y la posición de estas predicciones representan a la persona predicha. En este caso particular, la ventana `0` contiene 3 predicciones que favorecen a la posición `3, 4 y 5`. Para más detalle observe la figura 2.14. En la tabla 2.1 se puede observar la correspondencia posición/persona.

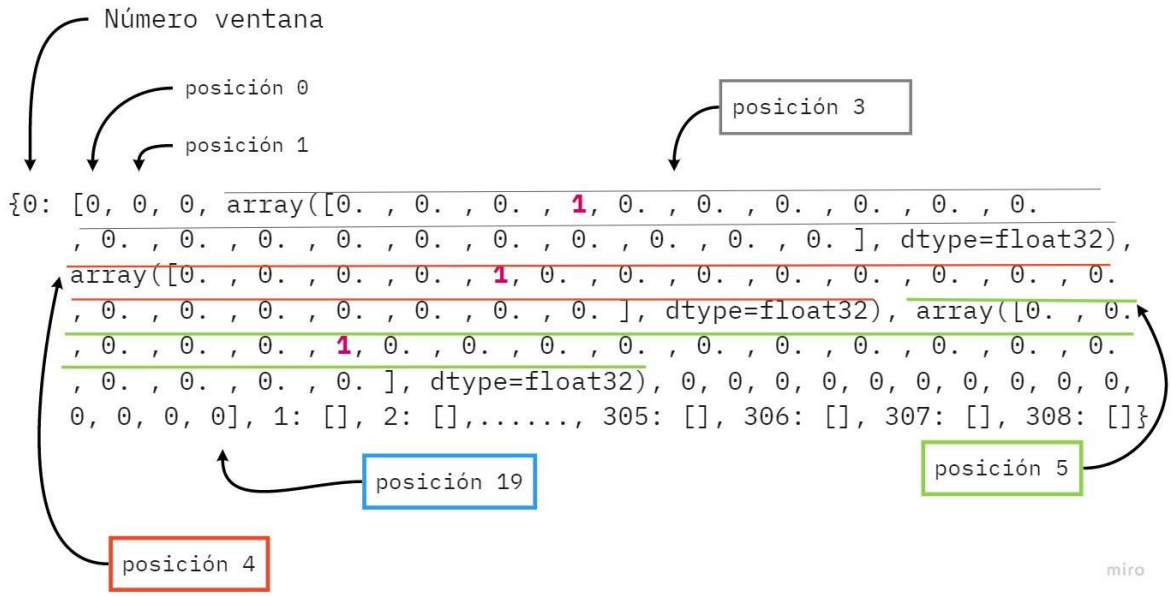

Figura 2.14: Predicciones que están presentes en la primera ventana.

7. Una vez recorrida la ventana deslizante sobre todos los fotogramas del video, se eliminan todas las posiciones de cada ventana que no contengan predicciones. Para esto se utiliza el código 2.34:

Segmento de código 2.34: Elimina posiciones de la lista de predicciones que siguen en 0.

```
1            if type(dicVentana[k][y])==int:
2             dicVentana[k].pop(type(dicVentana[k][y]))
```

El diccionario de ventanas estará solo con predicciones como se observa en la siguiente figura 2.15.

8. Tomando en cuenta que inicialmente se crearon tantos keys como fotogramas predichos hubieron, si la ventana deslizante es mayor a 1, habrán keys vacías. Estas se eliminarán mediante el código 2.35:

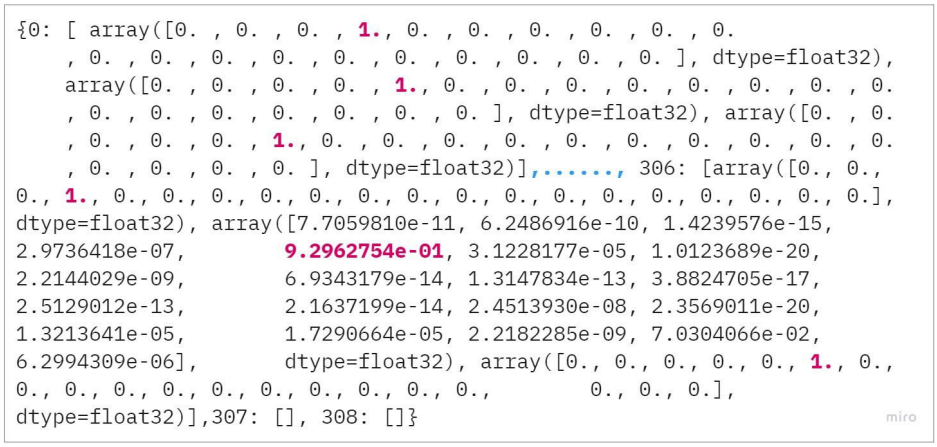

Figura 2.15: Diccionario de ventanas constituida netamente por predicciones.

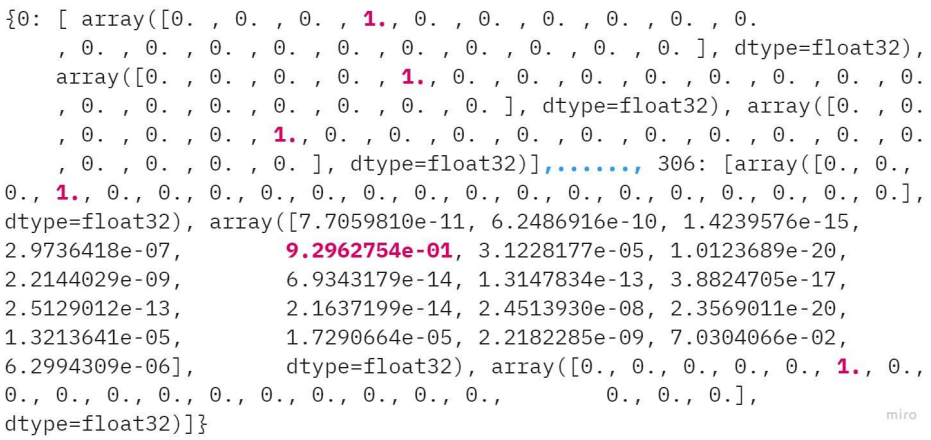

Figura 2.16: Diccionario de ventanas constituida con keys llenas.

Segmento de código 2.35: Elimina keys vacías.

```
1       if len(dicVentana[it])==0:
2           dicVentana.pop(it)
```

En el segmento de código anterior, `it` representa la key de diccionario de ventanas. Una vez realizado este proceso obtendremos un diccionario representado por la siguiente figura 2.16.

9. Se procede a comparar los valores máximos de cada predicción con un umbral de `0.9`, el cual decide si esta predicción es válida o es eliminada. Esto se realiza con el código 2.36.

Segmento de código 2.36: Elimina predicciones que están bajo el umbral.

```
1         if np.amax(dicVentana[a][b])<= 0.9:
2           del dicVentana[a][b]
```

La variable `a` representa los keys del diccionario de ventanas y `b` representa los índices

de los elementos que existe en cada Key.

10. Se realiza un conteo de los elementos que cada posición obtuvo. Esto fue realizado con el siguiente segmento de código 2.37.

Segmento de código 2.37: Número de veces que es reconocida una persona en un video.

```
    #convertimos a lista cada predicción que hay en una key
    lisk= dicVentana[a][b].tolist()
    #tomamos el índice del elemento con mayor probabilidad
    indice=lisk.index(max(lisk))
    #Sumamos 1 unidad al índice que corresponde al máximo valor de la ...
        predicción.
    listaconteo[indice] += 1
```

Al imprimir esta lista se mostrará lo siguiente: `[0, 1, 0, 304, 285, 306, 0, ... 0, 0, 0, 0, 0, 1, 0, 0, 19, 0, 0, 5, 0]`. En ella se observan notoriamente las posiciones que obtuvieron la mayor cantidad de predicciones, que son la `3, 4 ... y 5`.

11. Teniendo ya establecida la lista de conteo de predicciones de cada posición, se la traslada al último filtro. Este consiste en validar si el número de predicciones por posición es mayor al 50 % del número de ventanas, retornando una lista binaria donde 1 significa que una posición está presente y 0, que no. Esto fue realizado con el código 2.38:

Segmento de código 2.38: Código para discriminar si una persona fue detectada al menos en una ventana.

```
#Verifico que al menos haya una detección en una ventana
 #lista de elementos binarios que representan con 1 la presencia de la ...
     persona en con posición correspondiente
listaFinal=[]
if listaconteo[i]> 0:
    print("si")
    listaFinal[i]=1
```

Esta operación retorna la lista final de posiciones favorecidas con un valor de `1`. Mientras que las demás estarán con valor de `0`. Al imprimir esta lista se mostrará lo siguiente: `[0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`

12. Finalmente, se realiza el proceso de traducción de la lista obtenida en el paso anterior a la etiqueta correspondiente a la persona que aparece en el video analizado, Para esto utilizamos la información de la tabla 2.1 que muestra la relación que existe entre la posición de una predicción con la persona a la que representa.

    Este proceso fue realizado con el código 2.39.

Segmento de código 2.39: Muestra los nombres de las personas reconocidas dentro del video.

```
# lista donde se guardan las etiquetas de las personas que están en el video
listaEstanEnvideo=[]
#diccionario con las identidades de cada posicion
dicTraductor={0:"Asimbaya01",1:"Asimbaya02",2:"Asimbaya03", 3:"Buitron01",4:"Buitron02",5:"Buitron03",6:"Paguay01",
7:"Paguay02",8:"Paguay03",9:"Ramirez01",10:"Ramirez02",
11:"Ramirez03",12:"Revelo01",13:"Revelo02",14:"Revelo03", 15:"Noelia",16:"Miguel",17:"Jhimmy",18:"Angel",19:"Antonia"}
for ind in range(0,len(listaFinal)):
    if listaFinal[ind]!=0:
        listaEstanEnvideo.append(dicTraductor[ind])
print("En el video están: ",listaEstanEnvideo)
```

Como resultado nos imprimira algo similar a:

```
En el video están: [Buitron01, Buitron02, Buitron03]
```

Etiqueta	Posición	Persona
p01	0	Asimbaya01
p02	1	Asimbaya02
p03	2	Asimbaya03
p04	3	Buitron01
p05	4	Buitron02
p06	5	Buitron03
p07	6	Paguay01
p08	7	Paguay02
p09	8	Paguay03
p10	9	Ramirez01
p11	10	Ramirez02
p12	11	Ramirez03
p13	12	Revelo01
p14	13	Revelo02
p15	14	Revelo03
p16	15	Noelia
p17	16	Miguel
p18	17	Jhimmy
p19	18	Angel
p20	19	Antonia

Tabla 2.1: Tabla de correspondencia etiqueta / posición / persona.

El script completo utilizado para el reconocimiento usando coherencia temporal está en el anexo D, el archivo se llama *ReconociCoherenciaTemporal.ipynb*.

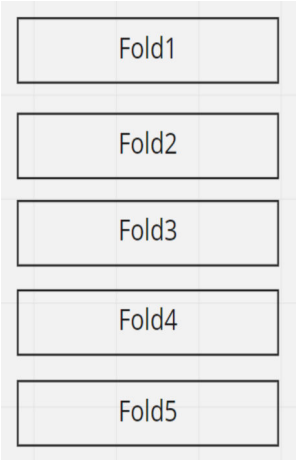

Figura 2.17: Folds creados a partir de la base de datos.

2.4. EXPERIMENTOS

Se realizaron dos experimentos principales, los cuales son: aplicar la validación cruzada K-fold al set de videos (1) y la realización de predicciones con vídeos que contenían múltiples personas (2).

1. Para el primer experimento fué primordial trabajar sobre la base de datos obtenida en la sección 2.1, la cual fue sometida al proceso de validación cruzada K-fold, donde K=5. Este proceso toma todos los videos correspondientes a cada una de las personas. Además, de forma aleatoria toma una cantidad igual de videos de cada persona para distribuirlo en 5 Folds. En el presente caso, cada fold contiene 2 videos por persona que, considerando que se analizaron un total de 20 personas, se tendrá un total de 40 videos. La representación de los folds se muestra en la figura 2.17, los cuales son asignados a los conjuntos de videos: entrenamiento, validación y prueba. En este caso la distribución fue asignada de la siguiente manera: al conjunto de entrenamiento el 60 % de folds, es decir, tres folds de videos, el 20 % para el conjunto de validación y el restante 20 % para el conjunto de prueba. Se formaron 5 combinaciones teniendo en cuenta que el fold de prueba no se repetirá en ninguna combinación como se ilustra en la figura 2.18. Cada combinación de datos será entrenada con la nueva red neuronal 2.9, lo cual se hizo para evaluar el mejor modelo de predicción.

   El script que se utilizó para calcular el primer experimento está en el anexo E, con el nombre de *seleccionCarpetasKfoldCrossValidation.ipynb*.

2. Para el segundo experimento se utilizó el modelo obtenido por el segundo grupo de folds de la figura 2.18. Fueron filmados 6 videos que contenían los rostros de múltiples personas, dentro de los cuales respectivamente existía la siguiente cantidad de individuos 3, 3, 3, 3, 4 y 5. Los vídeos tuvieron una duración de entre 10 a 12 segundos y fueron sometidos a las predicciones del modelo resultante elegido.

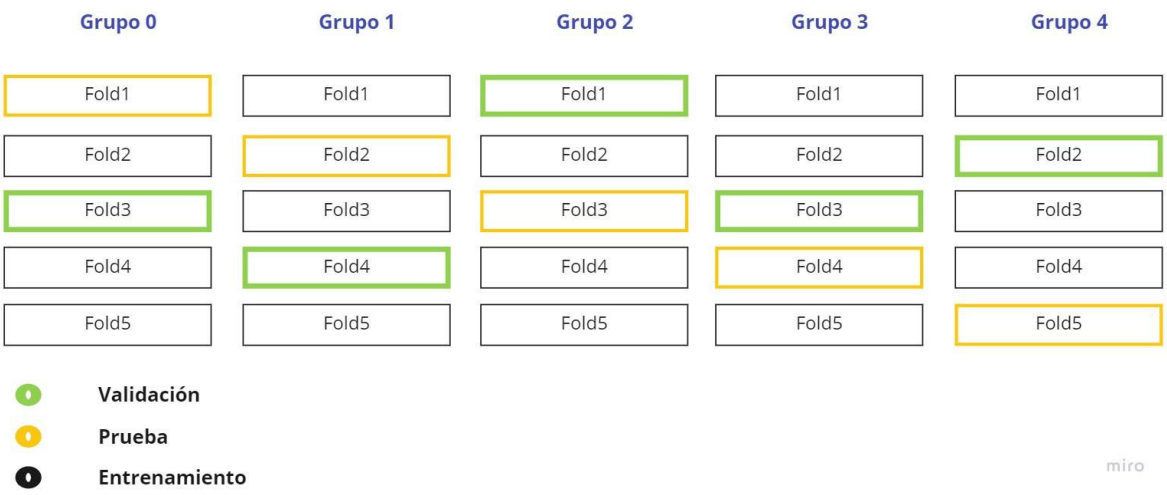

Figura 2.18: Grupos de folds para entrenamiento de modelos.

3. RESULTADOS Y DISCUSIÓN

En esta sección se presentan los resultados de utilizar la *validación cruzada K-fold* para la elección del modelo que obtuvo las mejores predicciones. Además, también se muestran los resultados de las predicciones que fueron realizadas por el modelo elegido junto a un algoritmo de coherencia temporal.

3.1. RESULTADOS DE UTILIZAR LA VALIDACIÓN CRUZADA K-FOLD

El resultado de entrenar cada grupo de folds de la figura 2.18 es un modelo. Al hacer el proceso de predicción en la carpeta *Prueba* de cada grupo de la figura 2.18 se pudo obtener una matriz de confusión, que sirvió para el cálculo de la exactitud de predicción de cada modelo. La exactitud de predicción de cada modelo se muestra en la tabla 3.1. Entonces, basados en la matriz de confusión, fue elegido el modelo con mayor exactitud de predicciones que corresponde al modelo generado por el grupo 2. La matriz de confusión del modelo elegido está en la figura 3.1.

Grupo	Modelo	Exactitud
grupo 0	modelo 0	93.88 %
grupo 1	modelo 1	97.35 %
grupo 2	modelo 2	98.96 %
grupo 3	modelo 3	97.25 %
grupo 4	modelo 4	94.99 %

Tabla 3.1: Tabla de correspondencia modelo / exactitud.

En la matriz de confusión observamos que las clases que obtuvieron menor exactitud son las etiquetadas con p07 y p18. La exactitud obtenida fue del 93 % y 94 % respectivamente. Por otro lado, las etiquetas p17 y p03 obtuvieron una exactitud del 97 % y 98 % en el mismo orden correspondiente. También se observa que las etiquetas p01, p13 y p20 tienen una exactitud del 99 %. Finalmente, observamos que las demás etiquetas, es decir, p02, p04, p05, p06, p08, p09, p10, p11, p12, p14, p15, p16 y p19 obtuvieron el 100 %. Por lo tanto, el modelo tiene un promedio de exactitud del 98.96 %. En la tabla 2.1 se puede observar las etiquetas de la persona a la que representada.
También se puede observar su curva ROC en la figura 3.2. La curva Roc que representa los verdaderos positivos en el eje Y y los falsos positivos en el eje X, idealmente se tendría una línea horizontal que inicie en 1 en la parte superior izquierda de la gráfica, con un área bajo la curva *(AUC)* de 1. Como el modelo elegido no es 100 % efectivo, esta área es menor. Teniendo en cuenta este parámetro se puede ver que el área bajo la curva Roc es muy cercana a el área de la curva Roc ideal.
Las predicciones se realizaron a cada conjunto de prueba con el código del anexo F, con nombre *PrediccionDelConjuntoPrueba_Evaluar_modelo.ipynb*.

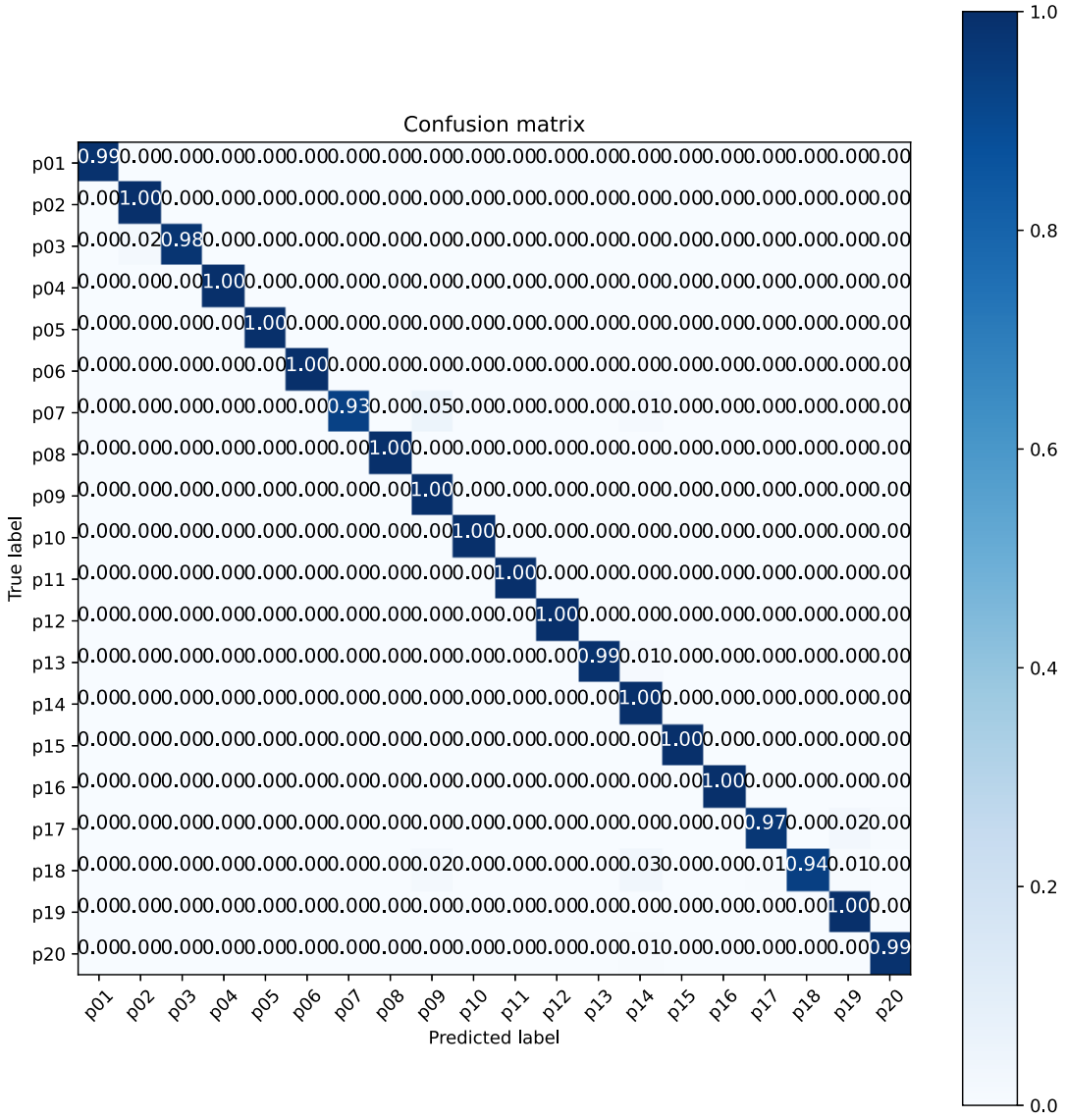

Figura 3.1: Matriz de confusión del modelo con mayor exactitud en las predicciones.

3.2. RESULTADOS DE LAS PREDICCIONES DEL MODELO ELEGIDO JUNTO A UN ALGORITMO DE COHERENCIA TEMPORAL

Los resultados de las predicciones del modelo elegido fueron sometidos a varias ventanas deslizantes que inician desde la ventana 1 a la 150, estos se pueden observar en la tabla 3.2 y están representados en la figura 3.3. En esta figura se observa la media de la exactitud, que está representada por la línea de color rojo tiende a incrementar al incrementar la ventana. Por otro lado, se observa alrededor de la media un sombreado azul, el cual representa el intervalo de confianza. Este intervalo de confianza entre más pequeño sea indicará un menor margen de error de las predicciones. Por tal razón, se observa que la precisión del modelo aumenta conforme la ventana se incrementa. Dependiendo de la duración del video, las predicciones pueden durar aproximadamente 10 minutos si el video dura 5 segundos. Este

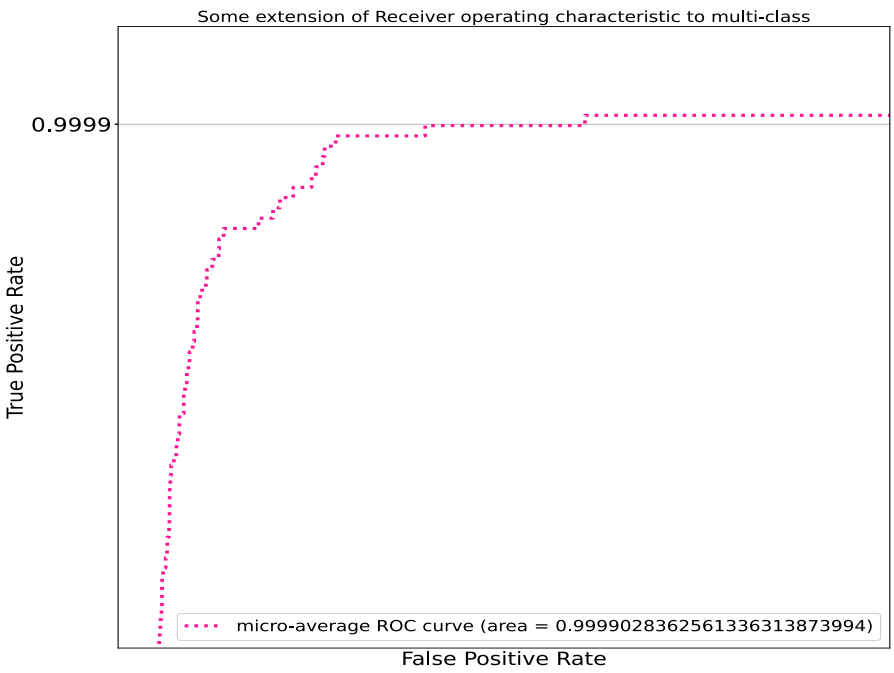

Figura 3.2: Curva ROC de las predicciones del modelo elegido.

tiempo en la práctica es relativamente adecuado, pudiendo representar una predicción en tiempo real en ciertos escenarios. Particularmente, el video que contenía 4 personas obtuvo una baja eficiencia en la primera ventana y alcanzó su mayor eficiencia en la ventana 140, pues el modelo fue entrenado solo con una persona de las contenidas. Por otro lado, los videos para los cuales se consideraron 3 personas llegaron a su eficiencia máxima antes de la ventana 20. Finalmente, el video de 5 personas alcanzó su mayor eficiencia en la ventana 89. Este segundo experimento se hizo para evaluar la exactitud de predicciones al utilizar un algoritmo basado en coherencia temporal.

VENTANA	EFICIENCIA					
	Buitrón	González	Ramírez	Revelo	Paguay	Asimbaya
1	0,9024	0,8886	0,7445	0,9518	0,9901	0,3042
2	0,9717	0,9013	0,8915	0,9926	0,9921	0,3875
3	0,9889	0,9087	0,9431	0,9968	0,9927	0,4308
4	0,9944	0,9112	0,9647	0,9989	0,9937	0,4659
5	0,9966	0,9168	0,9747	1	0,9947	0,5042
6	0,9988	0,9211	0,9848	1	0,9952	0,5323
7	1	0,9236	0,9904	1	0,9958	0,5587
8	1	0,9253	0,9961	1	0,9963	0,5833
9	1	0,927	0,9961	1	0,9968	0,6078

10	1	0,9283	0,999	1	0,9973	0,6239
11	1	0,9241	0,999	1	0,9978	0,61
12	1	0,925	1	1	0,9983	0,6331
13	1	0,9261	1	1	0,9973	0,6097
14	1	0,9266	1	1	0,9983	0,6153
15	1	0,9278	1	1	0,9978	0,6293
16	1	0,9288	1	1	0,9989	0,623
17	1	0,9298	1	1	0,9994	0,6404
18	1	0,9308	1	1	1	0,6484
19	1	0,9319	1	1	1	0,6674
20	1	0,933	1	1	1	0,6828
21	1	0,9337	1	1	1	0,6611
22	1	0,9343	1	1	1	0,6746
23	1	0,9355	1	1	1	0,6788
24	1	0,9357	1	1	1	0,683
25	1	0,937	1	1	1	0,6642
26	1	0,9383	1	1	1	0,6666
27	1	0,9406	1	1	1	0,6806
28	1	0,9414	1	1	1	0,6936
29	1	0,9421	1	1	1	0,6928
30	1	0,9429	1	1	1	0,7028
31	1	0,9426	1	1	1	0,6913
32	1	0,9429	1	1	1	0,7031
33	1	0,9446	1	1	1	0,695
34	1	0,9425	1	1	1	0,7034
35	1	0,9429	1	1	1	0,7101
36	1	0,9438	1	1	1	0,7169
37	1	0,9447	1	1	1	0,728
38	1	0,9456	1	1	1	0,7415
39	1	0,9465	1	1	1	0,745
40	1	0,9475	1	1	1	0,7616
41	1	0,9473	1	1	1	0,7744
42	1	0,9476	1	1	1	0,7807
43	1	0,9485	1	1	1	0,7824
44	1	0,9495	1	1	1	0,7655
45	1	0,9499	1	1	1	0,7717
46	1	0,9509	1	1	1	0,771
47	1	0,9518	1	1	1	0,7822
48	1	0,9529	1	1	1	0,7987
49	1	0,9539	1	1	1	0,8135

50	1	0,9549	1	1	1	0,8155
51	1	0,9547	1	1	1	0,8256
52	1	0,9551	1	1	1	0,8389
53	1	0,9562	1	1	1	0,8327
54	1	0,9573	1	1	1	0,8406
55	1	0,9571	1	1	1	0,8546
56	1	0,9582	1	1	1	0,8661
57	1	0,9593	1	1	1	0,8657
58	1	0,9598	1	1	1	0,8714
59	1	0,9609	1	1	1	0,8772
60	1	0,9620	1	1	1	0,8832
61	1	0,9625	1	1	1	0,8892
62	1	0,963	1	1	1	0,8955
63	1	0,9642	1	1	1	0,8984
64	1	0,9633	1	1	1	0,87822
65	1	0,9638	1	1	1	0,8464
66	1	0,965	1	1	1	0,8309
67	1	0,9663	1	1	1	0,8362
68	1	0,9675	1	1	1	0,8417
69	1	0,9688	1	1	1	0,8411
70	1	0,9701	1	1	1	0,8467
71	1	0,9707	1	1	1	0,8523
72	1	0,972	1	1	1	0,8582
73	1	0,9733	1	1	1	0,8641
74	1	0,9747	1	1	1	0,8702
75	1	0,9753	1	1	1	0,8697
76	1	0,9767	1	1	1	0,8759
77	1	0,9789	1	1	1	0,8823
78	1	0,9819	1	1	1	0,8888
79	1	0,9842	1	1	1	0,892
80	1	0,9849	1	1	1	0,8987
81	1	0,9856	1	1	1	0,9057
82	1	0,9864	1	1	1	0,9128
83	1	0,9887	1	1	1	0,9201
84	1	0,9902	1	1	1	0,9276
85	1	0,9919	1	1	1	0,8821
86	1	0,9934	1	1	1	0,8888
87	1	0,9942	1	1	1	0,8958
88	1	0,995	1	1	1	0,8991
89	1	1	1	1	1	0,9063

90	1	1	1	1	1	0,9098
91	1	1	1	1	1	0,9098
92	1	1	1	1	1	0,9251
93	1	1	1	1	1	0,933
94	1	1	1	1	1	0,9369
95	1	1	1	1	1	0,9409
96	1	1	1	1	1	0,9449
97	1	1	1	1	1	0,949
98	1	1	1	1	1	0,8986
99	1	1	1	1	1	0,9022
100	1	1	1	1	1	0,9058
101	1	1	1	1	1	0,9095
102	1	1	1	1	1	0,9132
103	1	1	1	1	1	0,91705
104	1	1	1	1	1	0,9209
105	1	1	1	1	1	0,9248
106	1	1	1	1	1	0,9289
107	1	1	1	1	1	0,933
108	1	1	1	1	1	0,9282
109	1	1	1	1	1	0,9323
110	1	1	1	1	1	0,9365
111	1	1	1	1	1	0,9408
112	1	1	1	1	1	0,9452
113	1	1	1	1	1	0,913
114	1	1	1	1	1	0,917
115	1	1	1	1	1	0,9211
116	1	1	1	1	1	0,9253
117	1	1	1	1	1	0,9296
118	1	1	1	1	1	0,934
119	1	1	1	1	1	0,9384
120	1	1	1	1	1	0,943
121	1	1	1	1	1	0,9476
122	1	1	1	1	1	0,9523
123	1	1	1	1	1	0,9371
124	1	1	1	1	1	0,9417
125	1	1	1	1	1	0,9465
126	1	1	1	1	1	0,9513
127	1	1	1	1	1	0,9562
128	1	1	1	1	1	0,9613
129	1	1	1	1	1	0,9664

130	1	1	1	1	1	0,9717
131	1	1	1	1	1	0,9771
132	1	1	1	1	1	0,9826
133	1	1	1	1	1	0,9768
134	1	1	1	1	1	0,9824
135	1	1	1	1	1	0,9881
136	1	1	1	1	1	0,9822
137	1	1	1	1	1	0,9821
138	1	1	1	1	1	0,9879
139	1	1	1	1	1	0,9939
140	1	1	1	1	1	1
141	1	1	1	1	1	1
142	1	1	1	1	1	1
143	1	1	1	1	1	1
144	1	1	1	1	1	1
145	1	1	1	1	1	1
146	1	1	1	1	1	1
147	1	1	1	1	1	1
148	1	1	1	1	1	1
149	1	1	1	1	1	1
150	1	1	1	1	1	1

Tabla 3.2: Tabla de eficiencia de cada ventana de los videos utilizados en los experimentos.

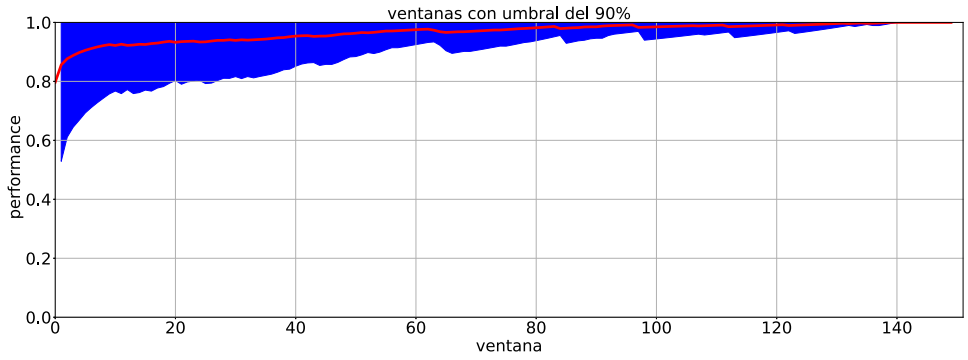

Figura 3.3: Precisión de cada ventana desde la ventana 1 a la 150.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

- En este trabajo se creó una base de datos de fotogramas de videos. En esta se utilizó la validación cruzada k-fold, asignando el valor de 5 a la variable K, de esta forma se obtuvo 5 grupos donde cada uno fue dividido en el conjunto de entrenamiento, validación y prueba. Entonces, se efectuó el entrenamiento de cada grupo y se eligió el mejor modelo tomando en cuenta la exactitud de predicción en los conjuntos de prueba. Posteriormente, con el modelo que obtuvo la mayor exactitud al predecir el conjunto de prueba se realizaron predicciones en vídeos de múltiples personas. Estas predicciones se sometieron a un algoritmo de coherencia temporal donde se utilizaron ventanas deslizantes sobre las predicciones de fotogramas consecutivos.

- Se concluye que el aprendizaje por transferencia fue de gran utilidad para poder obtener modelos de predicción sin tener que utilizar grandes bases de datos. Además, se pudo obtener un alto porcentaje de exactitud de predicción que alcanza el 80 % en videos con múltiples personas. Esto se lo puede observar en la figura 3.3.

- El modelo generado a través de la técnica del aprendizaje por transferencia incrementa su exactitud hasta un 98 %. Esto es posible debido a la coherencia temporal que es aplicada sobre videos.

- Al utilizar la coherencia temporal se descartan falsas detecciones de rostros dado que en ocasiones el detector de rostros captura zonas donde no hay rostro alguno.

- Google Colab ofrece varias alternativas para acelerar el tiempo de desarrollo de un algoritmo de aprendizaje automático. Esto lo hace gracias a sus recursos computacionales como procesadores comunes CPU, los procesadores gráficos GPU y por tiempos menores los procesadores tensoriales llamados TPU, que son procesadores diseñados para trabajar en ambientes aprendizaje automático.

4.2. RECOMENDACIONES

- Se recomienda utilizar ventanas deslizantes superiores a 40, ya que sobre esta ventana el intervalo de confianza se reduce y la exactitud de las predicciones están sobre el 90 %.

- Dentro del algoritmo de coherencia temporal se recomienda verificar que la ventana sea menor al número de fotogramas que contiene el video.

- Para la creación de la base de datos se recomienda grabar videos a una distancia máxima de 3 metros, considerando que en este caso la calidad de grabación fue de 1080p. Esta distancia será mayor si se utiliza videos grabados en mayor calidad.

- Para la creación de la base de datos se recomienda que los rostros contenidos dentro del video tengan un ángulo de giro máximo de 45 grados con respecto a la cámara, puesto que con ángulos mayores la detección de rostros disminuye.

- Como trabajo futuro se recomienda utilizar otras arquitecturas diferentes de VGG, como Inception-Resnet para comparar sus resultados con los obtenidos en el presente trabajo.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] ataspinar, “The Perceptron,” Dec. 2016, (accedido ene. 02, 2021). [Online]. Disponible en: https://ataspinar.com/2016/12/22/the-perceptron/

[2] A. Kirillov, “ANNT : Recurrent neural networks,” Dec. 2018, (accedido ene. 02, 2021). [Online]. Disponible en: https://www.codeproject.com/Articles/1272354/ANNT-Recurrent-neural-networks

[3] S. H. Khor, “The Gentlest Introduction to Tensorflow – Part 2,” (accedido dic. 23, 2020). [Online]. Disponible en: https://www.kdnuggets.com/the-gentlest-introduction-to-tensorflow-part-2.html/2/

[4] C. Xia, “Reducción de la pérdida: Tasa de aprendizaje,” (accedido ene. 02, 2021). [Online]. Disponible en: https://developers.google.com/machine-learning/crash-course/reducing-loss/learning-rate?hl=es-419

[5] Dot CSV, “¿Qué es una Red Neuronal? Parte 3 : Backpropagation | DotCSV,” Oct. 2018, (accedido ene. 02, 2021). [Online]. Disponible en: https://www.youtube.com/watch?v=eNIqz_noix8

[6] V. Kommineni, “How to use transfer learning for sign language recognition,” Mar. 2019, (accedido ene. 02, 2021). [Online]. Disponible en: https://www.freecodecamp.org/news/asl-recognition-using-transfer-learning-918ba054c004/

[7] v. amudha, “Xception Neural Network Transfer learning and Data Processing using AI,” Jan. 2019, (accedido ene. 02, 2021). [Online]. Disponible en: https://vigneshgig.medium.com/xception-neural-network-transfer-learning-and-data-processing-using-ai-c3e7a4ea7bf2

[8] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” 2015.

[9] B. AI, “Redes neuronales convolucionales,” Nov. 2019, (accedido ene. 02, 2021). [Online]. Disponible en: https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8

[10] A. G. Walkers, “max-pooling,” (accedido ene. 02, 2021). [Online]. Disponible en: https://austingwalters.com/wp-content/uploads/2019/01/max-pooling.png

[11] A. Biswal, “Convolutional Neural Network Tutorial,” Oct. 2020, (accedido ene. 02, 2021). [Online]. Disponible en: https://www.simplilearn.com/tutorials/deep-learning-tutorial/convolutional-neural-network

[12] S. Jaiswal, “Validation of Convolutional Neural Network Model - javatpoint,” (accedido ene. 02, 2021). [Online]. Disponible en: https://www.javatpoint.com/pytorch-validation-of-convolutional-neural-network-model

[13] M. Kazemi, “python - How to implement a neural network with a not-fully-connected layer as the final layer?” (accedido ene. 02, 2021). [Online]. Disponible en: https://stackoverflow.com/questions/63675602/how-to-implement-a-neural-network-with-a-not-fully-connected-layer-as-the-final

[14] L. Gonzalez, “Instalando Python y Spyder,” Apr. 2019, (accedido ene. 04, 2021). [Online]. Disponible en: https://aprendeia.com/como-instalar-python-y-spyder-curso-de-python/

[15] D. E. Villalón De La Vega, “Diseño e implementación de una plataforma de software para reconocimiento facial en video,” 2012.

[16] R. Cazorla Martínez, “Software para la detección y el reconocimiento de rostros.”

[17] L. Torres, J.-Y. Reutter, and L. Lorente, “The importance of the color information in face recognition,” in *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, vol. 3. IEEE, 1999, pp. 627–631.

[18] A. Yip and P. Sinha, “Role of color in face recognition,” 2001.

[19] P. J. Phillips, H. Wechsler, J. Huang, and P. J. Rauss, “The feret database and evaluation procedure for face-recognition algorithms,” *Image and vision computing*, vol. 16, no. 5, pp. 295–306, 1998.

[20] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. fisherfaces: Recognition using class specific linear projection,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 19, no. 7, pp. 711–720, 1997.

[21] B. O. P. Pico, P. Rondón, and H. Arguello, “Sistema de reconocimiento facial basado en imágenes con color,” *Revista UIS Ingenierías*, vol. 10, no. 2, pp. 113–122, 2011.

[22] M. A. Turk and A. P. Pentland, “Face recognition using eigenfaces,” in *Proceedings. 1991 IEEE computer society conference on computer vision and pattern recognition*. IEEE Computer Society, 1991, pp. 586–587.

[23] P. N. Belhumeur, J. P. Hespanha, and D. J. Kriegman, “Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection,” in *Computer Vision — ECCV ’96*, B. Buxton and R. Cipolla, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 43–58.

[24] G. Dong and H. Liu, *Feature engineering for machine learning and data analytics*. CRC Press, 2018.

[25] M. A. Rahim, M. S. Azam, N. Hossain, and M. R. Islam, “Face recognition using local binary patterns (lbp),” *Global Journal of Computer Science and Technology*, 2013.

[26] P. A. Pastore *et al.*, “Reconocimiento facial en imágenes,” B.S. thesis.

[27] L. Torrey and J. Shavlik, “Transfer learning,” in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[28] L. B. Neto, F. Grijalva, V. R. M. L. Maike, L. C. Martini, D. Florencio, M. C. C. Baranauskas, A. Rocha, and S. Goldenstein, “A kinect-based wearable face recognition system to aid visually impaired users,” *IEEE Transactions on Human-Machine Systems*, vol. 47, no. 1, pp. 52–64, 2016.

[29] L. Laura-Ochoa, “Evaluación de Algoritmos de Clasificación utilizando Validación Cruzada,” in *Proceedings of the 17th LACCEI International Multi-Conference for Engineering, Education, and Technology: “Industry, Innovation, and Infrastructure for Sustainable Cities and Communities”*. Latin American and Caribbean Consortium of Engineering Institutions, 2019, (accedido ene. 06, 2021). [Online]. Disponible en: http://laccei.org/LACCEI2019-MontegoBay/meta/FP471.html

[30] T. Fushiki, “Estimation of prediction error by using k-fold cross-validation,” *Statistics and Computing*, vol. 21, no. 2, pp. 137–146, 2011.

[31] X.-D. Zhang, “Machine learning,” in *A Matrix Algebra Approach to Artificial Intelligence*. Springer, 2020, pp. 223–440.

[32] J. Zambrano, “¿Aprendizaje supervisado o no supervisado?” Mar. 2018, (accedido ene. 07, 2021). [Online]. Disponible en: https://medium.com/@juanzambrano/aprendizaje-supervisado-o-no-supervisado-39ccf1fd6e7b

[33] T. Hastie, R. Tibshirani, and J. Friedman, “Unsupervised learning,” in *The elements of statistical learning*. Springer, 2009, pp. 485–585.

[34] A. Banafa, “¿Qué es el aprendizaje profundo?” Aug. 2016, (accedido ene. 05, 2021). [Online]. Disponible en: https://www.bbvaopenmind.com/tecnologia/mundo-digital/que-es-el-aprendijaze-profundo/

[35] L. Bärmann, “Multimodal goal-oriented dialog using encoder-decoder-networks,” Ph.D. dissertation, Informatics Institute, 2018.

[36] L. Gonzalez, “Introducción a Bias y Varianza,” Nov. 2018, (accedido feb. 17, 2021). [Online]. Disponible en: https://aprendeia.com/bias-y-varianza-en-machine-learning/

[37] A. G. Schwing and R. Urtasun, “Fully connected deep structured networks,” *arXiv preprint arXiv:1503.02351*, 2015.

[38] S. Patrikar, “Batch, Mini Batch & Stochastic Gradient Descent,” Oct. 2019, (accedido ene. 08, 2021). [Online]. Disponible en: https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

[39] P. A. Blanco, “Algoritmo de Retropropagaci´on,” p. 8.

[40] J. Koushik, “Understanding convolutional neural networks,” *arXiv preprint arXiv:1605.09081*, 2016.

[41] B. AI, “Redes neuronales,” Nov. 2019, (accedido ene. 08, 2021). [Online]. Disponible en: https://bootcampai.medium.com/redes-neuronales-13349dd1a5bb

[42] L. Llamas, “Machine learning con TensorFlow y Keras en Python,” (accedido ene. 08, 2021). [Online]. Disponible en: https://www.luisllamas.es/machine-learning-con-tensorflow-y-keras-en-python/

[43] F. Castillo, “Introducción y Procesamiento de imágenes en Python con OpenCV,” Jul. 2017, (accedido ene. 04, 2021). [Online]. Disponible en: https://unipython.com/introduccion-procesamiento-imagenes-python-opencv/

[44] □. M. d. l. F. Sanz, “Google Colab: Python y Machine Learning en la nube,” Jun. 2019, (accedido ene. 04, 2021). [Online]. Disponible en: https://www.adictosaltrabajo.com/2019/06/04/google-colab-python-y-machine-learning-en-la-nube/

[45] □. Robledano, “Qué es Python: Características, evolución y futuro,” Sep. 2019, (accedido ene. 04, 2021). [Online]. Disponible en: https://openwebinars.net/blog/que-es-python/

[46] J. Garcia, “PyCharm, un potente IDE para crear programas con Python | Linux Adictos,” (accedido ene. 04, 2021). [Online]. Disponible en: https://www.linuxadictos.com/pycharm-un-potente-ide-para-crear-programas-con-python.html

6. ANEXOS

Los siguientes anexos se encuentran en formato digital:

ANEXO A. Extracción de rostros de los videos.ipynb

ANEXO B. entrenarModeloNuevo.ipynb

ANEXO C. PrediccionesVideoCompletoMultiplesPersonas.ipynb

ANEXO D. ReconociCoherenciaTemporal.ipynb

ANEXO E. seleccionCarpetasKfoldCrossValidation.ipynb

ANEXO F. PrediccionDelConjuntoPrueba_Evaluar_modelo.ipynb

7. ORDEN DE EMPASTADO