BUAP

Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Detección de videos modificados por la técnica de intercambio de identidad DeepFake

Tesis para obtener el título de

Maestro en Ciencias de la Computación

PRESENTA

Odón David Carrasco Limón

DIRECTOR DE TESIS

Dra. Maya Carrillo Ruiz

Co ASESOR DE TESIS

Dra. María de Lourdes Sandoval Solís

H. Puebla de Z. octubre 2022

Agradecimientos

A mis padres y hermano por su apoyo incondicional en esta y muchas otras etapas de mi vida.

A la Doctora Blanca Bermúdez, que en paz descanse, por motivarnos siempre a dar el extra.

A mis amigos de cómputo matemático, que siempre me dieron su incondicional apoyo en conocimiento, haciendo de la maestría la época de risas y diversión más peculiar de todas.

A mis tías y tíos , las cual han dado tanto amor y cariño a mí y a mi familia siempre.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico brindado durante mis estudios de posgrado.

Dedicatoria

A mis amigos, los cual funge como pilar de apoyo en esta etapa de estudio, y posterior a esta. En especial a Fernando

Gracias por nunca verme como un estorbo, por saber escuchar y por inspirarme con tu alegría como siempre lo hiciste en la carrera.

https://acortar.link/GCGZKX

RESUMEN

Durante la aparición del algoritmo DeepFake, entre los años 2017 a 2019, el 96% de los videos generados por este, eran de naturaleza pornográfica. Con la aparición de aplicaciones móviles, al alcance de cualquier usuario, que generan estos videos, y la creciente preocupación de empresas como Facebook por dichos videos, surge la necesidad de herramientas que determinen si un video es real o generado por intercambio de identidad. Esta tesis muestra una revisión de los trabajos más relevantes en este tema, presenta la propuesta de un sistema y detalla los elementos que lo conforman. También define los parámetros necesarios para optimizar dichos elementos. Además de presentar una alternativa para entrenar modelos que excedan la potencia de cómputo. Los experimentos realizados reportan un AUC promedio de 0.9968, comparable al estado del arte.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

LISTA DE ACRÓNIMOS

AUC	*Area under the ROC Curve*	Área bajo la curva.
ACC	*Accuracy*	Exactitud.
AdaGrad	*Adaptative Gradient Algorithm*	Algoritmo de Gradiente Adaptativo.
Adam	*Adaptive moment estimation*	Estimación del momento adaptativo.
CNN	*Convolutional Neural Network*	Redes neuronales Convolucionales.
DDR4	*Double Data Rate type 4 Synchronous Dynamic Random-Access Memory*	Tasa de datos doble tipo 4
DSFD	*Doble shot face detector*	Detector de rostros de doble disparo.
GAN	*Generative Adversarial Networks*	Redes Generativas Antagónicas.
Gb		Gigabyte
GHz		Gigahercio
GPU	*Graphic processor Unity*	Unidad de procesamiento Grafico.
LR	*Learning Rate*	tasa de aprendizaje.
LSTM	*Long short-term memory*	Red de memoria a corto plazo
MHz		Megahercio
MSE	*Mean Square Error*	Error cuadrático medio.
MTCNN	*Multi-task cascaded convolutional networks*	Redes convolucionales en cascada multitarea.
RAM	*Random Access Memory.*	Memoria de acceso aleatorio.
RMSprop	*Root Mean Square Propagation*	Propagación de raíz cuadrática media.
RNN	*Recurrentional Neural Network*	Red neuronal recurrente.
ROC	*Receiver Operating Curve*	Curva de funcionamiento del receptor.
RTX	*Ray Tracing Texel eXtreme*	Trazado de rayos Texel extremo
VRAM	*Video Random Access Memory.*	Memoria de acceso aleatorio de video.
WSDAN	*Weakly Supervised Data Augmentation Network*	Red de aumento de datos débilmente supervisada.

CAPÍTULO 1

INTRODUCCIÓN

Desde el año 1865 existe la manipulación de material audiovisual [1] con diversos fines como intimidar y acosar a las personas o propagar desinformación [2]. En sus inicios, las técnicas de manipulación de fotografías y videos requerían de un dominio del tema y tiempo [3] pero gracias a los avances en redes neuronales convolucionales (CNN) hoy en día existen aplicaciones móviles[1] que generan manipulaciones al alcance de cualquier persona sin conocimiento del tema, por lo cual surge la necesidad de crear herramientas que permitan su detección al alcance de todos.

Tolosana et al. [4], describe 4 tipos de manipulación facial para la generación de videos falsos, los cuales son: **Síntesis facial completa**, la cual crea rostros a partir de poderosas redes generativas antagónica (GAN). **Manipulación de atributos**, que cambia características tales como color de ojos, pelo, edad o agrega accesorios como gafas o sombreros entre otras, mediante redes GAN. **Intercambio de expresiones**, la cual cambia las expresiones de una persona por la de otra, utilizando redes GAN con técnicas como Face2Face [5]. **Intercambio de identidad**, este como el nombre indica, intercambia el rostro de una persona por otro proveniente de algún video, por técnicas basadas en gráficos por computadora o por aprendizaje profundo. Cabe señalar que en el año 2017 el usuario en Reddit llamado DeepFake acuño el término homónimo a este tipo de manipulación facial, por haber desarrollado un algoritmo de aprendizaje automático, este algoritmo se usó principalmente para intercambiar el rostro de celebridades en videos Pornográficos.

En el año 2019 la cadena de noticias BBC en su reportaje [6] menciona que, del total de videos DeepFake encontrados en la red, hasta ese momento, el 96% eran de naturaleza pornográfica, lo que empezaba a generar preocupación por el uso de

[1] Fake App, DeepFake Studio, Jiggy, REFACE son algunas aplicaciones disponibles en las tiendas de aplicaciones de Android y Apple

esta tecnología, como lo fue Facebook [2]. En ese momento el costo monetario para generar cada video DeepFake era de 2.99 dólares, aunque hoy gracias a aplicaciones móviles centradas en DeepFake, el algoritmo está potencialmente al alcance de todos, Por lo que esta será el tipo de manipulación facial en la cual se centrará esta tesis

El objetivo general de esta tesis es diseñar una aplicación a partir de la selección de componentes documentados en el estado del arte, para establecer si un video donde aparezca un rostro humano fue modificado por la técnica de intercambio de identidad.

Para alcanzar el objetivo señalado, se estudiaron las líneas de investigación con métodos como: discrepancias temporales presentes en la transmisión del flujo óptico, con trabajos como el de Guerra et al. [1], y Sabir et al. [3]; Búsqueda de artefactos en las imágenes, con trabajos como el de Li et al. [7], y Tolosana et al.; también se estudiaron trabajos que crearon sus propias bases de datos, probando su comportamiento con diversas combinaciones de métodos, con trabajos como el de Dolhansky et al. [2], Rossler et al. [8], y Li et al. [9]. Finalmente se revisaron proyectos del concurso DeepFake Detection Challenge (DFDC) [10] sin artículos asociados, en los cuales los tres primeros lugares utilizaron combinaciones de los métodos antes mencionados.

Con revisión de las investigaciones anteriores se llegó a la elección del método y la identificación de las etapas que lo componen, las cuales son: la extracción del rostro, la extracción de características y la clasificación. En este trabajo se detallará cada una de estas etapas, las herramientas que nos permiten llegar a una exactitud comparable con el estado del arte y la calibración de estas. Todo ello para llegar a un resultado que nos permita determinar si un video fue modificado. Con base en las revisiones realizadas se establecieron como objetivos específicos de esta tesis los siguientes:

a) Seleccionar el componente encargado de extraer los rostros humanos en el video.

b) Seleccionar el componente encargado de la extracción de características en las imágenes con rostros humanos.
c) Seleccionar un clasificador conveniente para la tarea.
d) Seleccionar los conjuntos de datos para entrenar el modelo.
e) Calibrar los parámetros para cada componente de la aplicación.
f) Probar y comparar los resultados de la aplicación con los resultados del estado del arte.
g) Explorar la posibilidad de trabajar con bases de datos de segunda generación.

CAPÍTULO 2

TRABAJOS RELACIONADOS

Como se mencionó en la introducción, se estudiaron trabajos relevantes para la detección de videos modificados por la técnica de intercambio de identidad, dichos trabajos se presentarán en orden histórico para poder resaltar la evolución de las líneas de investigación, así como la aparición de herramientas y la evolución de estas.

En los siguientes trabajos se puede observar que existen etapas comunes para las soluciones propuestas, las cuales son: la primera la extracción de rostros en la cual se separa el mismo, pues es la zona donde hay más información, para determinar si un video es real o no; una segunda la extracción de características en la cual típicamente se utiliza alguna red CNN para que esta determine las características de las imágenes y una tercera de clasificación, donde a partir de los resultados de la extracción de características se determina si el video es real o no, dichas etapas se pueden observar en la Figura 1.

En el año de 2018, Güera et al. [1], proponen un modelo para la detección de videos modificados por la técnica de intercambio de identidad o DeepFake. Analizando la sucesión de fotogramas en los videos, buscando discrepancias temporales presentes en la transmisión del flujo óptico, mediante redes CNN y una red de memoria a corto plazo (LSTM). Dado que su red ocupa una secuencia de fotogramas, parte el video en lotes de 80 y los procesa en su modelo. Posterior a este proceso de extracción de características, utiliza unas reglas de inferencia en la última capa de estas redes y determina si el video fue modificado o no. Güera et al., prueban su modelo con una base de datos propietaria, midiendo el área bajo la curva (AUC) para determinar su exactitud, la cual es de 0.971. El modelo de Güera et al., Se puede observar en la Figura 2, en la cual podemos ver de manera gráfica y sintetizada lo antes mencionado.

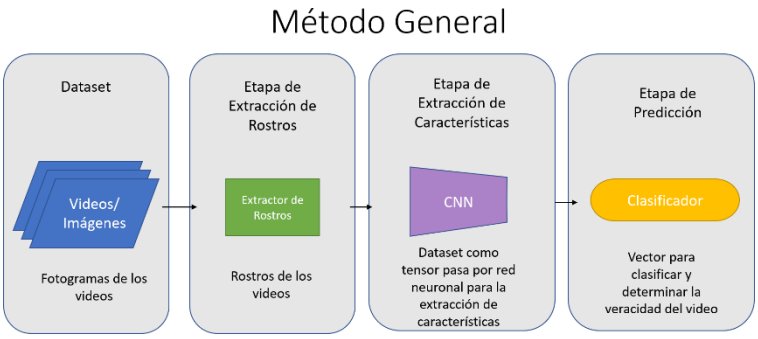


Figura 1 Etapas del modelo general para detección de DeepFake.

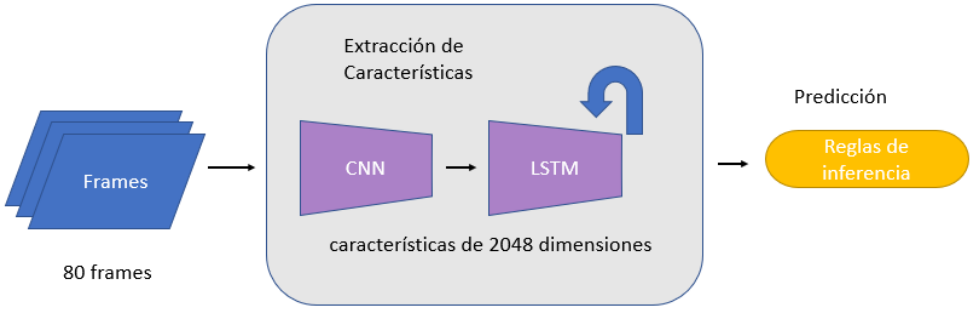


Figura 2 Modelo propuesto por Güera et al. [1], para detección de videos falsos.

En el año 2019, Rossler et al. [8], al percatarse de que no existía una base de datos dedicada a DeepFake que replicará las condiciones a las cuales se somete un video tras ser subido a redes sociales, mismas que tienen algoritmos de compresión que reducen la calidad de los videos, proponen crear su propia base de datos a la cual denominan FF++. Rossler et al., prueban su base de datos con los modelos disponibles en ese momento, usando combinaciones de métodos con extracción de rostro y sin extracción de rostro y probando diferentes CNN para los extractores de características. También lo hace buscando características en el rostro y en el área alrededor de este, probando diferentes combinaciones de extractores de características y midiendo el AUC para determinar la exactitud. Rossler llega a la exactitud más alta, con el extractor de rostros más robusto en el momento, Face2Face [5] del año 2016 y la red XceptionNet [11] con pesos pre entrenados con ImageNet y mediante reglas de inferencia para determinar si el video fue modificado o no. Este modelo al probarse con la base generada por ellos dio un AUC de 0.99 para los videos HQ y 0.97 para LQ.

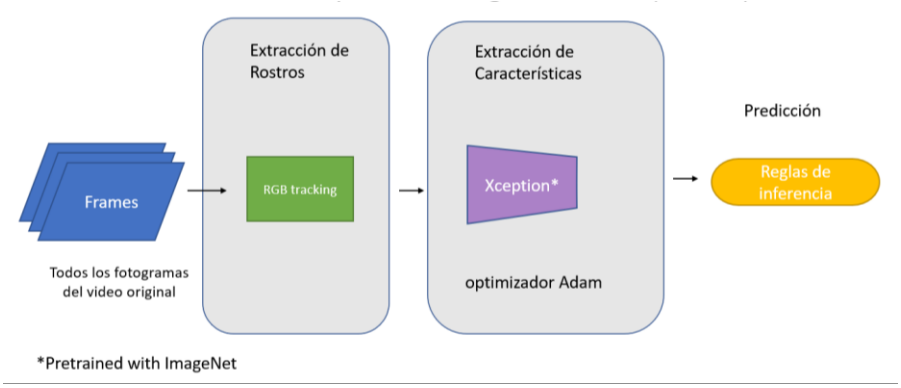


Figura 3 Modelo propuesto por Rossler et al. [8], para detección de videos falsos.

Dicho modelo se puede observar en la Figura 3, en la cual podemos resaltar la utilización de una etapa de extracción de rostros y la introducción de optimizadoras a las redes CNN, también comparte la etapa de predicción por medio de reglas de inferencia como Güera.

En el año 2019 Sabir et al. [3], siguiendo la línea de investigación en el flujo óptico, proponen otro modelo para detectar DeepFake mediante redes neuronales recurrentes (RNN) buscando características temporales en bloques de 5 fotogramas. Sabir et al., utilizan Face2Face, añadiendo a la extracción de rostros una etapa de alineamiento, para ejecutar más fácil la extracción de características.

Para esta etapa de alineamiento sugirieron dos métodos, uno implícito por medio de redes CNN que no logra darles resultados favorables, por lo cual usaron un método explícito utilizando puntos de referencia faciales. En la etapa de extracción de características usaron 2 CNN; una para extraer características de bajo nivel como pueden ser mandíbula discontinua, ojos borrosos, etc. y otra para las de alto nivel (macroscópicas). Ambas salidas pasan a una red RNN para extraer características espaciales y temporales. Los autores emplearon Feedforward para extraer el resultado de la clasificación por medio de reglas de inferencia. Sabir et al., probaron su modelo con la base de datos FF++ midiendo el AUC para determinar su exactitud, misma que obtuvo un valor de 0.963. El modelo de Sabir et al., se puede observar en la Figura 4, en el cual podemos remarcar la existencia de una etapa de extracción de rostros, así como el uso del mismo optimizador de las redes CNN que utilizo Rossler.

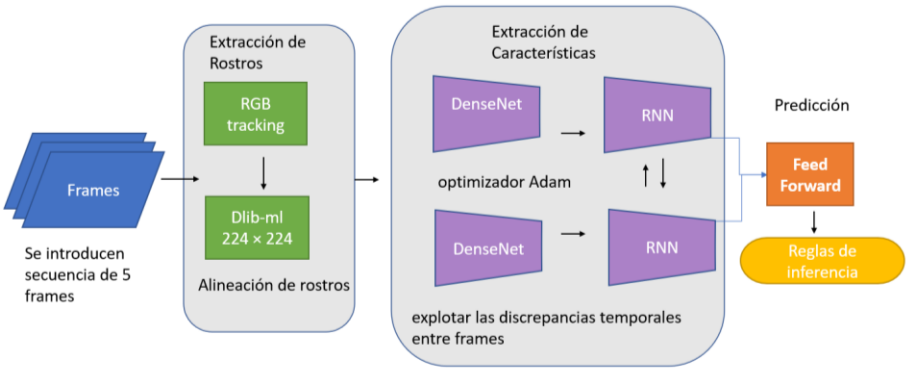


Figura 4 Modelo propuesto por Sabir et al., [3] para detección de videos falsos.

En el año 2019, Dolhansky et al. [2], crean una base de datos para el concurso DeepFake Detection Challenge [10] con el mismo nombre, posterior a la creación verificaron la precisión que tiene los modelos existentes con está. Entre los modelos probados, el que logró la mayor precisión usó Face2Face para extraer los rostros. Para la etapa de extracción de características empleó la red CNN XceptionNet y por medio de reglas de inferencia se determinó si el video era modificado o no. El modelo descrito obtuvo una precisión de 0.93 al ser probado con una porción de la base de datos, pues, Dolhansky et al., solo querían revisar el comportamiento de modelos con su base para ser aplicada al concurso del mismo nombre. Este modelo se puede observar en Figura 5, y se puede destacar el uso de Face2Face al igual que Sabir et al., y Rossler et al.; así como el uso de la red XceptionNet como Rossler et al. Finalmente la utilización de reglas de inferencia para la etapa de predicción.

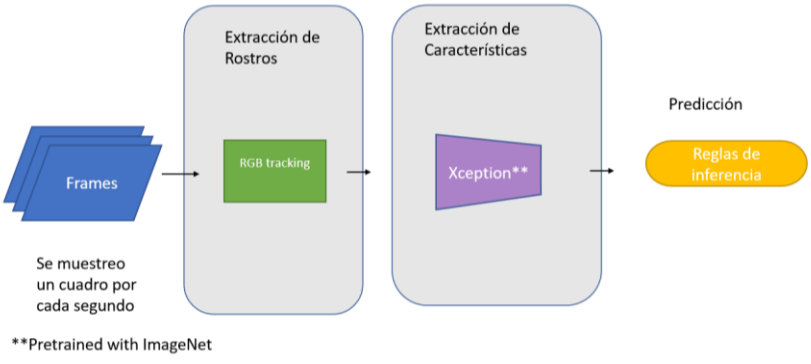


Figura 5 Modelo propuesto por Dolhansky et al. [2], para detección de videos falsos.

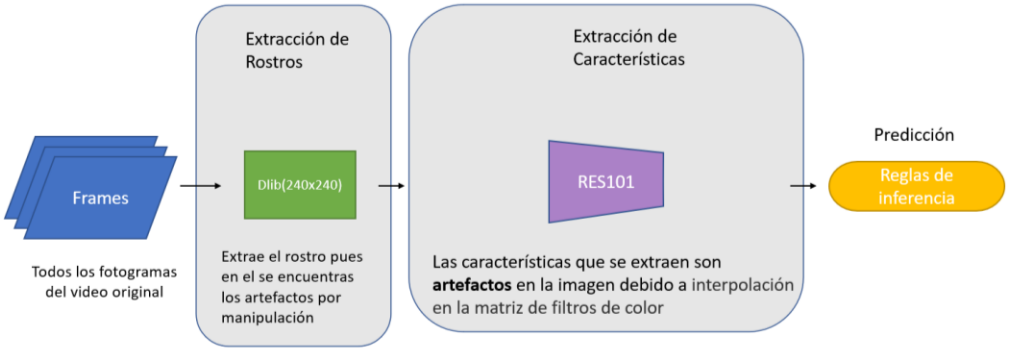


Figura 6 Modelo propuesto por Li et al. [7], para detección de videos falsos.

En el año 2019, Li et al. [7], parten de la hipótesis que los algoritmos DeepFake solo pueden generar imágenes de una resolución limitada, a los que, por medio de transformaciones, se hacen coincidir con los rostros del video original. Dichas transformaciones generan artefactos (*Artifacts*) en los videos, con lo cual propone buscar dichos artefactos para detectar si el video fue modificado.

Li et al., sugieren su propia base de datos denominada Celeb-DF [9], con lo cual entrenan su modelo para la detección de videos DeepFake. Esto mediante la toma de todos los fotogramas del video y extrayendo el rostro mediante el toolkit Dlib, luego probaron extraer las características con 4 redes CNN: VGG16 y tres ResNet de diferentes familias. Encontrando que la red con mejor exactitud fue ResNet101, con la cual por medio de reglas de inferencia determinaron si el video fue modificado o no. Su modelo se entrenó y probó en dos bases de datos: UADFV y DeepfakeTIMIT, tras la creación de su propia base de datos sé probó con UADFV, DFDC y Celeb-DF, con el AUC de cada una de estas bases obteniendo valores de 0.977 para UADFV, 0.755 para DFDC y 0.646 para Celeb-DF. El modelo de Li et al. se puede apreciar en la Figura 6, en el cual toma una CNN diferente a otros autores, pero de la misma manera por medio de reglas de inferencia extrae la predicción.

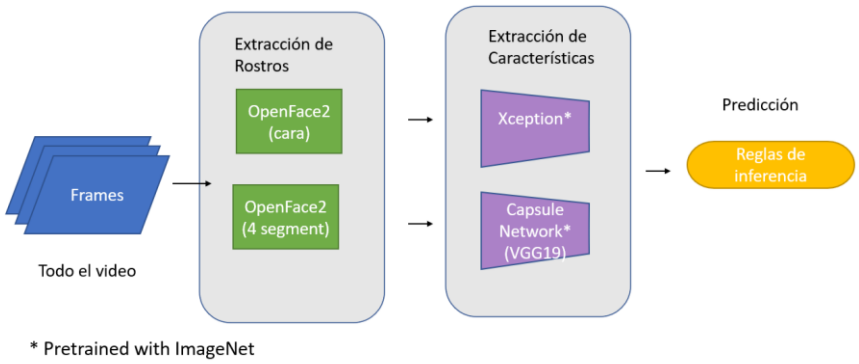


Figura 7 Modelo propuesto por Tolosana et al. [4], para detección de videos falsos.

En el año 2020, Tolosana et al. [4], crean su modelo para detectar videos DeepFake a partir de otros modelos que consideran características de alto y bajo nivel, [3] pero enfocando la búsqueda a la región del rostro, en ojos, la boca y la pose de la cabeza, como menciona Baltrusaitis et al. [12], toman cada fotograma del video y extraen el rostro, colocan en imágenes separadas los Landmarks referentes a los ojos, boca, nariz y el resto del rostro mediante el toolkit OpenFace2 [12]. Mediante una combinación de XceptionNet pre entrenada con ImageNet y una red cápsula VGG19, extraen las características de estas 4 fotografías y, mediante reglas de inferencia, logran diferenciar si un video fue modificado. Tolosana et al., prueban su modelo con las bases de datos UADFV, DFDC y Celeb-DF, midiendo el AUC, obteniendo valores de 1 para UADFV, 0.91 en DFDC y 0.836 para Celeb-DF. Este modelo se puede observar en la Figura 7, se puede apreciar la utilización de un extractor de rostros para tener más imágenes las cuales procesar, así como la utilización de la CNN XceptionNet antes vista, añadiendo una segunda red, lo cual solo se había visto en Sabir et al., Con combinaciones de redes RNN y al igual que todos los autores vistos hasta el momento, utiliza reglas de inferencia para extraer la predicción.

En el año 2020 se llevó a cabo el concurso DFDC, en el cual los ganadores aún no cuentan con un artículo asociado, pero cada uno de ellos describe en sus páginas de GitHub como fueron construidos sus modelos. En este concurso participaron diversos equipos y al analizar a los ganadores se pudo observar la utilización de nuevas herramientas.

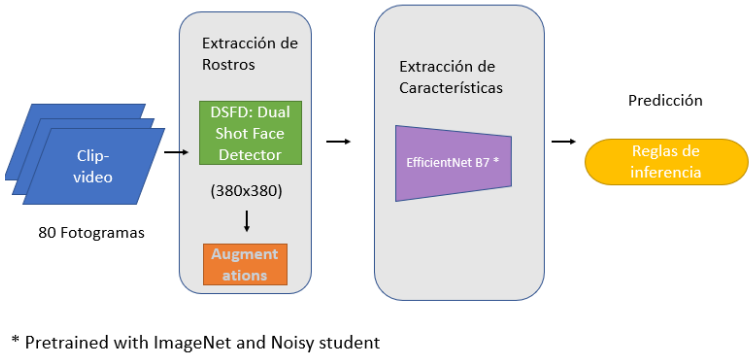


Figura 8 Modelo propuesto por A. Davletshin en [13] para detección de videos falsos

El equipo que obtuvo el tercer lugar lleva por nombre NTech [13] en el cual A. Davletshin menciona que ocupó una secuencia de fotogramas a los cuales se le extrajo el rostro mediante el detector de rostros de doble disparo (DSFD) la cual ocupa una red que recrea una malla 3D del mismo. En la etapa de extracción de características mencionan que ocuparon EfficientNet, el cual según Tan et al. [14], es una mejora a XceptionNet, Davletshin et al., utilizaron la familia B7 con pesos pre entrenados con Noisy Student, junto con aumentos a los rostros, como ruido gaussiano, transformaciones mediante rotación, escalamiento, escala de grises entre otros. Mediante reglas de inferencia logran diferenciar si un video fue modificado. El equipo logró un desempeño en exactitud medido en Log Loss [15] de 0.43452. Dicho modelo se puede apreciar en la Figura 8, en el cual podemos destacar a EfficientNet como una evolución de XceptionNet, la cual había sido la red CNN más ocupada hasta ese momento.

El equipo que alcanzo la medalla de plata lleva por nombre WM [16] en el cual H. Zhao describe algunas etapas de su modelo: en la etapa de extracción del rostro, utilizaron RetinaFace. En la etapa de extracción de características intentaron implementar una RNN LSTM; pero dado que los resultados no obtuvieron puntuaciones que los llevaran a ganar, decidieron cambiar por una red CNN. Entrenando 3 Redes en búsqueda de la que obtuviera valores de exactitud suficientes para ganar el concurso.

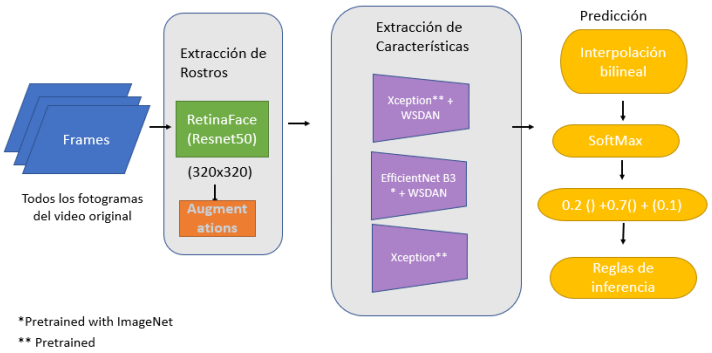


Figura 9 Modelo propuesto por H. Zhao en [16] para detección de videos falsos

Ocuparon una red XceptionNet pre entrenada con ImageNet, una combinación de XceptionNet con una red de aumento de datos débilmente supervisada (WSDAN) y una implementación de EfficientNet con WSDAN. Por motivos de tiempo optaron por usar una interpolación bilineal de estas tres redes, logrando un desempeño de 0.42842 en Log Loss. Dicho modelo se puede observar en la Figura 9, en la cual podemos destacar la combinación de 3 redes CNN, 2 XceptionNet y 1 EfficientNet y por primera vez una interpolación bilineal de estas, tomando este preprocesamiento para obtener las reglas de inferencia y obtener la predicción, aunque cabe señalar que en esta interpolación se le dio más peso a la red EfficientNet.

El equipo ganador fue el de Seferbekov [16] el cual describe que su modelo ocupa todos los fotogramas del video, extrayendo los rostros mediante Redes convolucionales en cascada multitarea (MTCNN) [18]. En la etapa de extracción de características empleaba una red XceptionNet, hasta el descubrimiento de las redes EfficientNet. Una vez comprobado su desempeño en exactitud y tiempo, implementa la familia B7 y aplicando aumentos a los rostros logró entrenar su modelo dando una exactitud de 0.42798 en Log Loss. Este modelo se puede apreciar en la Figura 10, donde nuevamente se remarca la utilización de la CNN EfficientNet y por primera vez mencionando explícitamente que se cambió a la CNN XceptionNet por la primera, dado su exactitud.

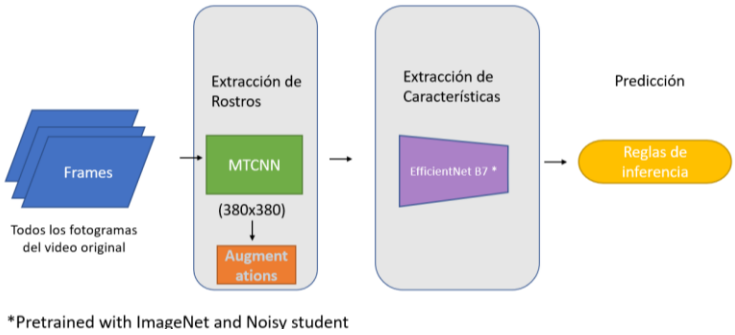


Figura 10 Modelo propuesto por Seferbekov en [17] para detección de videos falsos

Con la información recopilada en el estado del arte se crea la Tabla 1, donde se puede observar claramente los extractores de rostros empleados, así como los extractores de características, mostrando que todas las aproximaciones tienen de clasificador reglas de inferencia, de manera similar en la Tabla 2 se puede observar los diferentes paradigmas que se han utilizado hasta el momento.

Retomando el modelo general de la Figura 1, utilizando la información de la Tabla 1 se **seleccionaron las siguientes herramientas**: para la etapa de extracción de rostros Openface2; para la extracción de características EfficientNet; siguiendo la misma premisa de los demás autores se utilizará reglas de inferencia para la etapa de predicción. El modelo propuesto se puede observar en la Figura 11.

Cada herramienta fue seleccionada en su respectiva etapa por mostrar el mejor desempeño, y **cabe señalar que ningún autor hasta el momento ha utilizado la combinación propuesta**. Cada herramienta se abordará más a detalle en su respetivo capítulo, mostrando su evolución y demostrando porque es la mejor de cada etapa.

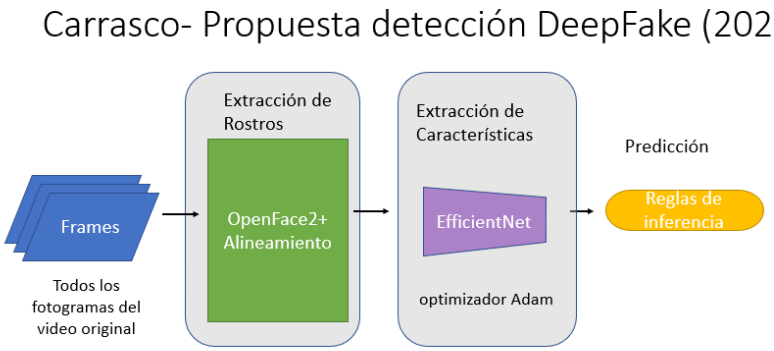


Figura 11 Modelo propuesto para detección de videos falsos

Tabla 1 Comparación del estado del arte.

Autor	Fotogramas	Extractor de Rostros	Extractor de Características	DB	Medida de Desempeño AUC
Güera- Temporal Features (2018)	80 fotogramas por paquete	-	CNN + LSTM	Propia	0.971
Rossler- Deep Learning Feature (2019)	Todos los Fotogramas	RGB tracking	Xception Preentrenada	FF ++	0.99 | 0.97
Sabir- Temporal Features (2019)	5 fotogramas por segundo	RGB tracking + Dlib-ml + Alineamiento	DenseNet + RNN	FF ++	0.969
Dolhansky – Deep Learning Features (2019)	1 fotograma por segundo	RGB tracking	Xception Preentrenada	DFDC	0.93 Precisión
Dolhansky – Deep Learning Features(2019)	1 fotograma por segundo	-	Xception Preentrenada	DFDC	0.784 Precisión
Li-Face Warping Features (2019)	Todos los Fotogramas	Dlib	Res101	UADFV | DFDC| Celef - DF	0.977 |0.755 | 0.646
Tolosana- Facial Regions Features (2020)	Todos los fotogramas	OpenFace2+ Alineamiento	Xception Preentrenada y Capsule Network (VGG19)	UADFV | DFDC | Celef - DF	**1** | 0.91 |**0.836**
A. Davletshin - NTech-(2020)	50 fotogramas por paquete	DSFD+ Aumentos	EfficientNet B7 Preentrenada con Noisy student	DFDC	0.43452 Log Loss
H. Zhao -WM- (2020)	Todos los Fotogramas	RetinaFace + Alineamiento + Aumentos	Xception Preentrenada y EfficientNet B3 + WSDAN y Xception Preentrenada + WSDAN	DFDC	0.42842 Log Loss
S. Seferbekov Selimsef-(2020)	Todos los Fotogramas	MTCNN + Aumentos	EfficientNet B7 Preentrenada con Noisy student	DFDC	0.42798 Log Loss

Tabla 2 Comparación de los paradigmas del estado del arte.

Autor	Paradigma
Güera-Temporal Features (2018)	Discrepancias temporales presentes en la transmisión del flujo óptico
Sabir- Temporal Features (2019)	Discrepancias temporales presentes en la transmisión del flujo óptico
Tolosana- Facial Regions Features (2020)	Búsqueda de artefactos en las imágenes
Li-Face Warping Features (2019)	Búsqueda de artefactos en las imágenes
Dolhansky – Deep Learning Feature (2019)	Creación de base de datos probando su desempeño
Rossler- Deep Learning Features (2019)	Creación de base de datos probando su desempeño
Li-Face Warping Features (2019)	Creación de base de datos probando su desempeño
Davletshin -NTech-(2020)	Combinación de Paradigmas
Zhao -WM- (2020)	Combinación de Paradigmas
Seferbekov Selimsef-(2020)	Combinación de Paradigmas

CAPÍTULO 3

EXTRACTOR DE ROSTROS

Como se mencionó en el capítulo anterior, en la Tabla 1 los autores utilizaron diversas propuestas en la etapa de extracción de rostros, habiendo métodos que no utilizaron ningún extractor y ocuparon toda la imagen como Güera et al. [1], hasta autores que extrajeron el rostro y posteriormente lo separaron en 4 imágenes extra como Tolosana et al. [4], por lo cual la elección de un extractor de rostros no es fácil.

Con base en la Tabla 1, se seleccionaron como posibles extractores de rostros a: Dlib [19], MTCNN [18] y OpenFace2 [12]; los cuales fueron los que en conjunto a su extractor de características dieron el mayor desempeño en AUC. Para seleccionar el mejor extractor de rostros entre estos, es necesario entender ¿qué es la extracción de rostros, qué algoritmos son los más comunes? y con esto poder elegir con fundamentos.

La extracción de rostros es el proceso por el cual se localizan los patrones que contiene un rostro en una imagen digital, por lo que su descomposición en patrones fáciles de localizar es fundamental. Las técnicas para localizar rostros en imágenes buscan modelos óptimos para la localización de dichos patrones a lo largo de toda la imagen de manera eficiente.

En la actualidad el algoritmo base para los diferentes modelos de extracción de rostros es Haar Cascade, desarrollado por Viola et al. [20], el cual es un algoritmo de detección de objetos usado para: identificar caras en imágenes o videos en tiempo real, identificar los rostros mediante el reconocimiento de cuatro patrones bien definidos representados por matrices específicas. Este algoritmo ha sido utilizado en diversos conjuntos de herramientas o toolkit.

Haar Cascade presenta problemas cuando se tienen escenarios diversos, con condiciones de iluminación variables y cuando la persona se mueve mucho, estos escenarios son denominados “en lo salvaje “. Por lo que una mejora fue ocupar una CNN para detectar los patrones en la imagen y ocupando detección en cascada se

creó una familia de algoritmos, uno de los que destaca es MTCNN el cual utiliza Zhang et al. [18], obteniendo mejores resultados que Haar Cascade.

En la detección de rostros existen diversas líneas de investigación, pues aparte de detectar si una fotografía o video contiene un rostro, también es importante delimitar esta región, estimar donde se encuentran regiones de ojos, nariz y boca: estimar hacia donde apunta la mirada, la pose del rostro, si la persona se encuentra sonriendo. Estas líneas se pueden agrupar en 4 principales que son: detección de los puntos de referencia faciales o Landmarks, estimar la postura de la cabeza, reconocer las expresiones faciales y hacer una estimación de la mirada. Cada una de estas líneas de investigación son importantes y cuentan con algoritmos particulares, mismos que se detallarán a continuación y se mostrarán una comparación de cuáles obtienen el mejor desempeño.

3.1 LANDMARKS

Los puntos de características faciales también llamados Landmarks se encuentran principalmente alrededor de componentes faciales como ojos, boca, nariz y mentón. Como se puede ver en la Figura 12, suelen tener 68 puntos si se considera el contorno del rostro y 49 si no se contempla el contorno.

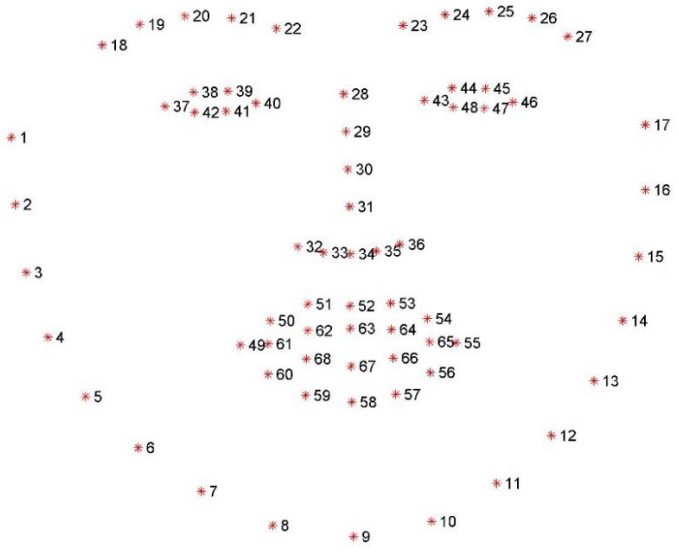


Figura 12 68 Landmarks utilizados

Tabla 3 Resultados de Landmarks medidos en la mediana del error normalizado. Extraída de [21].

	Con contorno (68 puntos)		Sin contorno (49 puntos)	
Aproximación	Frontal	Perfil	Frontal	Perfil
CLNF-2014	4.39	7.73	3.82	6.22
CFAN-2014	4.89	20.26	4.37	22.92
DRMF-2013	-	-	4.55	25.52
CFSS-2015	4.16	7.66	3.57	6.79
PO-CR2015	-	-	3.73	21.2
TCDCN-2014	4.91	9.26	4.53	9.09
3DDFA-2016	5.9	8.14	4.85	6.48
CE-CLM-2018	**4.09**	**6.31**	**3.47**	**5.19**

La estimación de Landmarks es un proceso supervisado o parcialmente supervisado a partir de un conjunto etiquetado en el cual, al delimitar la región del rostro, ya sea por el modelo Haar Cascade o por algún otro modelo. Para estas estimaciones se tienen algoritmos que estiman los Landmarks, incluso cuando una zona del rostro no esté visible.

De acuerdo con Zadeh et al. [21], algunos modelos que realizan la estimación de Landmarks son los que se pueden observar en la Tabla 3, en el trabajo mencionado se utiliza el conjunto de datos IJB-FL [22]. En dicho trabajo se presenta las medianas del error normalizado de los Landmarks para cada modelo, tanto en imágenes de perfil como frontales.

Como se puede observar en la Tabla 3, el modelo con mejor desempeño es CE-CLM del año 2018, el cual es implementado en Openface2 con algunas mejoras para su funcionamiento en tiempo real.

3.2 POSE DE LA CABEZA

La pose de la cabeza es importante, al estar en referencia a la cámara en orientación y posición relativa, por ello se cuenta con métodos para estimarla. Dicha estimación requiere de la región del rostro, la cual es delimitada por métodos, como el de construir un cubo con coordenadas en tres ejes X, Y y Z. Esto nos permite calcular diversas poses del rostro.

Tabla 4 Resultados de la estimación de la pose de la cabeza. Medido en la media del error absoluto. Extraída de [12].

Método	Yaw	Pitch	Roll	Media
Reg. Forests-2011	7.2	9.4	7.5	8
CLM-Z-2012	5.1	3.9	4.6	4.6
CLM-2011	4.8	4.2	4.5	4.5
Chehra-2014	13.9	14.7	10.3	13
OpenFace-2016	3.6	3.6	3.6	3.6
OpenFace 2-2018	**3.1**	**3.5**	**3.1**	**3.2**

De acuerdo con Baltrusaitis et al. [12], algunos métodos que realizan la estimación de la pose de la cabeza son los que se pueden observar en la Tabla 4, en el trabajo mencionado se utiliza el conjunto de datos ICT-3DHP [23]. Para todos los métodos considerados, el grado de error se mide con la media del error absoluto. Según los resultados de la Tabla 4 los métodos con mejor desempeño es OpenFace2 del año 2018.

3.3 ESTIMACIÓN DE LA MIRADA

La estimación de la mirada de un rostro consta de trazar un rayo desde los ojos hacia la cámara, teniendo en cuenta la dirección, a partir de la cual se estima hacia donde apunta la mirada.

Según Baltrusaitis et al., algunos de los métodos que realizan esta estimación de la dirección de la mirada son los que se pueden observar en la Tabla 5, en el trabajo mencionado se utiliza el conjunto de datos MPIIGaze [24]. Para todos los métodos considerados se presenta la media del error absoluto. Como se puede observar en la Tabla 5, el método con mejor desempeño es OpenFace2 del año 2018.

Tabla 5 Resultados de la estimación de la mirada, medida en error de grado absoluto. Extraída de [12].

Método	Error
EyeTab-2014	47.1
CNN on UT-2015	13.91
CNN on SynthesEyes-2015	13.55
CNN on SynthesEyes + UT-2015	11.12
OpenFace-2016	9.96
UnityEyes-2016	9.95
OpenFace2-2018	**9.1**

3.4 EXPRESIONES FACIALES

La estimación de las expresiones faciales se realiza mediante la estimación de los Landmarks y se aproxima con unidades de acción facial, las cuales describen objetivamente las activaciones de los músculos faciales, con esto se extrae información necesaria para otros campos.

Las unidades de acción de la expresión facial usando se estiman usando: histograma de gradientes orientados, características geométricas, como la localización de Landmarks y parámetros de forma, mediante un clasificador supervisado.

Según Baltrusaitis et al., algunos de los métodos que realizan la estimación de las unidades de acción facial son los que se pueden observar en la Tabla 6, el trabajo mencionado utiliza el conjunto de datos DISFA [25]. Para cada método se calcula el coeficiente de correlación de Pearson presentando la media para la detección de diferentes acciones.

Con lo observado en la Tabla 3, Tabla 4, Tabla 5 y Tabla 6 se concluye que, de las herramientas seleccionadas Openface2 es la que presenta el mejor desempeño y dado que utiliza implícitamente MTCNN, al ocupar este toolkit se tendrá el mejor extractor de rostros. Así se justifica la utilización de dicho extractor en nuestro modelo como se pudo observar en la Figura 11.

AU	Full name	Illustration
AU1	INNER BROW RAISER	
AU2	OUTER BROW RAISER	
AU4	BROW LOWERER	
AU5	UPPER LID RAISER	
AU6	CHEEK RAISER	
AU7	LID TIGHTENER	
AU9	NOSE WRINKLER	
AU10	UPPER LIP RAISER	
AU12	LIP CORNER PULLER	
AU14	DIMPLER	
AU15	LIP CORNER DEPRESSOR	
AU17	CHIN RAISER	
AU20	LIP STRETCHED	
AU23	LIP TIGHTENER	
AU25	LIPS PART	
AU26	JAW DROP	
AU28	LIP SUCK	
AU45	BLINK	

Figura 13 Lista de Unidades de Acción Facial en OpenFace2. Extraída de [12].

Tabla 6 Comparación de coeficiente de correlación de Pearson para la estimación de Unidades de acción facial. Extraída de [12].

Método	AU1	AU2	AU4	AU5	AU6	AU9	AU12	AU15	AU17	AU20	AU25	AU26	Mean
IRKR-2016	**0.7**	**0.68**	0.68	0.49	**0.65**	0.43	0.83	0.34	0.35	0.21	0.86	0.62	0.57
LT-2015	0.41	0.44	0.5	0.29	0.55	0.32	0.76	0.11	0.31	0.16	0.82	0.49	0.43
CNN-2015	0.6	0.53	0.64	0.38	0.55	**0.59**	**0.85**	0.22	0.37	0.15	0.88	0.6	0.53
D-CNN-2016	0.49	0.39	0.62	0.44	0.53	0.55	**0.85**	0.25	0.41	0.19	0.87	0.59	0.51
CCNF-2014	0.48	0.5	0.52	0.48	0.45	0.36	0.7	**0.41**	0.39	0.11	**0.89**	0.57	0.49
OpenFace2(SVR-HOG)-2018	0.64	0.5	**0.7**	**0.67**	0.59	0.54	**0.85**	0.39	**0.49**	**0.22**	0.85	**0.67**	**0.59**

CAPÍTULO 4

EXTRACTOR DE CARACTERÍSTICAS

Retomando la Tabla 1 del estado del arte, donde la mayoría de los autores utilizaron CNN en sus modelos, como puede ser el caso de Li et al. [7], con una ResNet101, Dolhansky et al. [2], y Rossler et al. [8], con una XceptionNet y Davletshin [13] ,Seferbekov [17] y Zhao [16] quienes utilizaron EfficientNet, es claro que la correcta elección de un extractor de características es relevante para tener un buen desempeño. En este capítulo se presenta la selección de un extractor conveniente para la investigación en curso.

Una Red Neuronal Convolucional es una generalización de una Red Neuronal clásica, con la particularidad de que al aumentar las dimensiones de la red los pesos sinápticos no serán números si no matrices n-dimensionales que aplicarán convoluciones a la entrada para obtener su salida.

Las CNN puede descubrir características en las imágenes, como detección de bordes y esquinas hasta llegar a detectar características más complejas, a través del entrenamiento con grandes conjuntos de datos, de manera particular la CNN encuentra patrones en los rostros humanos y nos permiten determinar si un rostro es real o no.

Una CNN cuenta con parámetros propios, dichos parámetros son: **Tamaño de canal**, es el tamaño del vector de entrada en cada capa, a mayor tamaño de canal, mayor deberá ser el número de convoluciones para llegar a la capa final, pues las convoluciones irán reduciendo las dimensiones de cada capa. **Profundidad**: la profundidad de una CNN será el número de capas ocultas de la red, utilizadas para conectar la capa de entrada con la capa de salida, en cada capa se harán convoluciones hasta alcanzar el tamaño deseado del vector de salida, mientras más profunda sea la red CNN aprende más características, pero tarda más en entrenarse. **Ancho**: el ancho de una CNN es el tamaño del vector de salida. Dichos parámetros se pueden observar en la Figura 14.

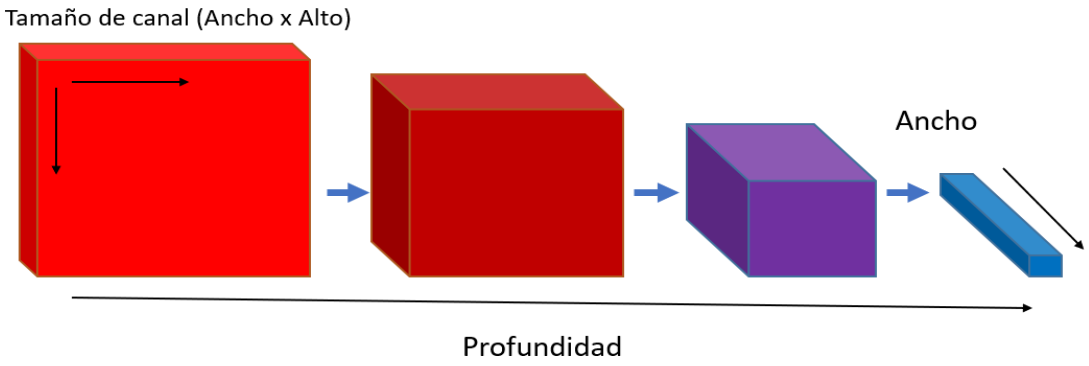


Figura 14 Diagrama de CNN con parámetros Tamaño de canal, ancho y profundidad.

En el campo de visión por computadora se han creado diversas arquitecturas de CNN y se han puesto a prueba en el conjunto de datos ImageNet con el fin de detectar objetos en imágenes. Tales CNN han sido ResNet, InceptionNet y XceptionNet, por dar algunos ejemplos. Todas estas redes buscan tener mayor precisión en la detección de objetos, trabajando con redes robustas incrementando continuamente su tamaño. Al escalar la red en profundidad, ancho (tamaño de vector de salida) o tamaño de canal (tamaño de vector de entrada); se obtienen mayor precisión. Sin embargo, el tiempo de entrenamiento se incrementa, así como el número de parámetros y el coste computacional. Ejemplo de lo mencionado es la red ResNet, misma que al aumentar su profundidad genera una familia de arquitecturas como ResNet20, Resnet50 etc. La mayoría de las CNN buscan escalar un solo valor de los mencionados, pero nunca una combinación de estos.

Por otro lado, EfficientNet surge como una red que busca encontrar un factor de escalamiento para los parámetros antes mencionados, buscando maximizar la precisión, minimizando el número de parámetros y el coste computacional. Esta red se modeló en forma de función y de manera empírica se encontraron valores para escalar la red, logrando una precisión superior a las CNN mencionadas previamente. Con estos factores de escala, los autores propusieron un caso base y aplicando estos factores en forma recursiva crearon una familia de CNN llamada EfficientNet. B0 es el caso base, B1 el siguiente factor y así sucesivamente hasta el B7. Teniendo similar o mejor precisión que otras CNN con un número menor de parámetros y número de operaciones de punto flotante como se puede ver en la Figura 15.

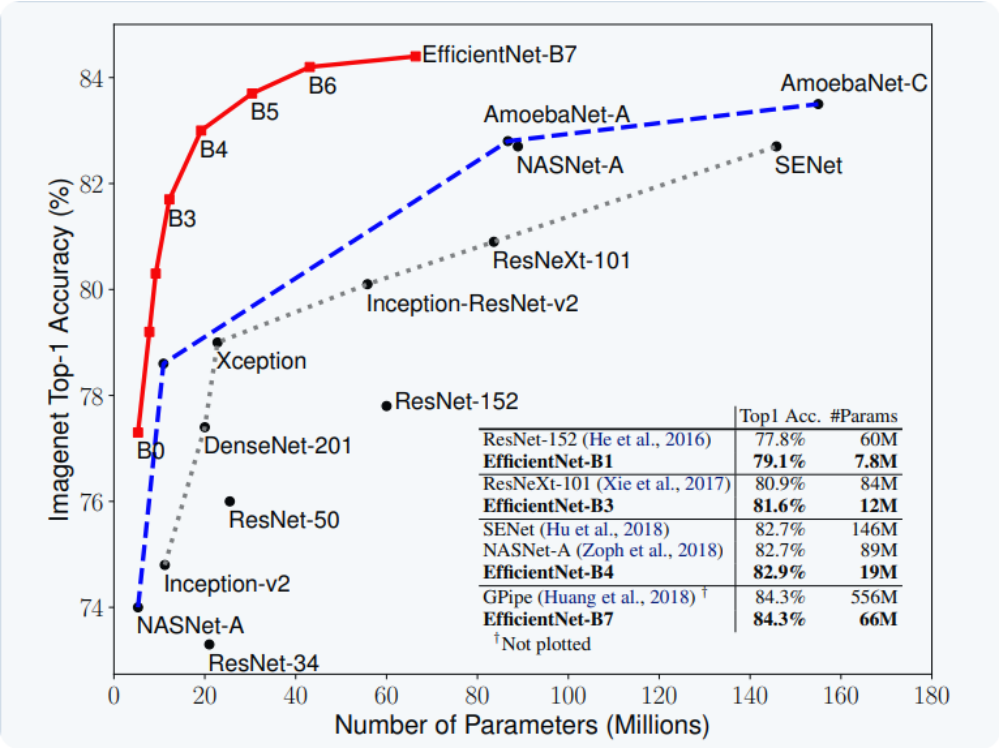


Figura 15 Comparación de las Familias EfficientNet con otras CNN, en número de parámetros. Extraída de [14]

Esta familia de CNN muestra una disminución en el número de parámetros utilizados y la cantidad de operaciones de punto flotante por segundo como se puede observar en la Figura 16, alcanzando resultados en precisión similar o superior a otras redes CNN, mostrando que la elección de parámetros óptimos de escalamiento permite aumentar la exactitud y disminuir el número de operaciones de punto flotante.

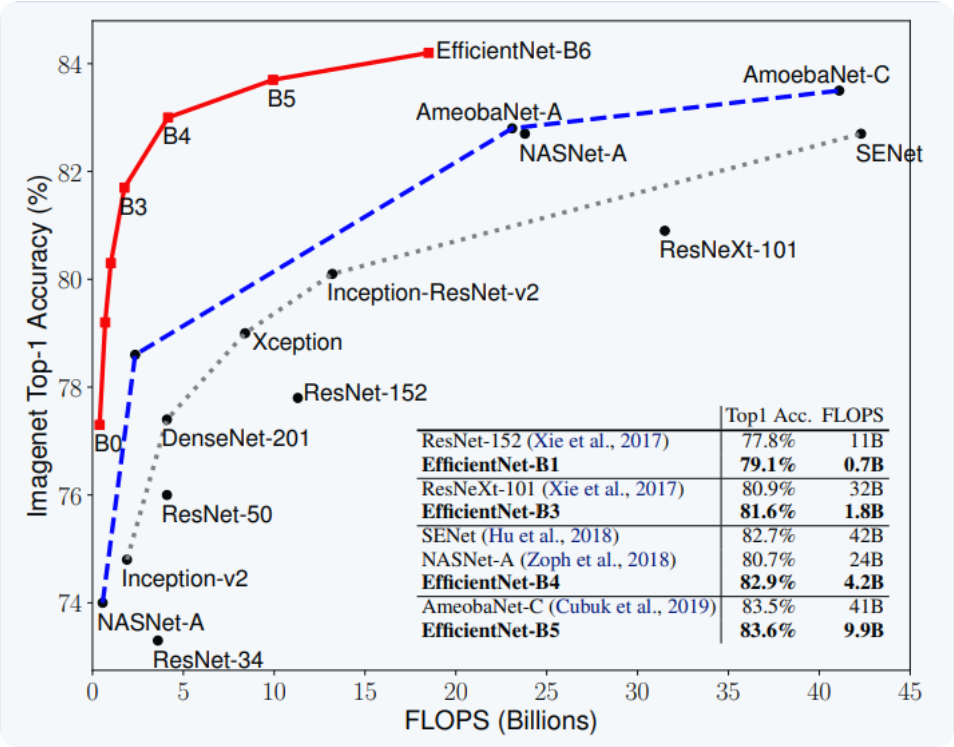


Figura 16 Comparación de las Familias EfficientNet con otras CNN, en número de operaciones de punto flotante por segundo. Extraída de [14].

En el concurso DFDC los ganadores utilizan EfficientNet como extractor de características demostrando que esta familia de arquitecturas no solo logra discernir objetos en imágenes, si no también logra extraer características en rostros humanos.

Esta disminución de parámetros y la reducción significativa de operación de punto flotante hacen que EfficientNet funcione como extractor de características y pueda utilizarse en cualquier equipo. Aunque la disminución de poder de cómputo requerido no es suficiente para utilizar los modelos más grandes en equipos personales. Por lo tanto, se justifica la elección de EfficientNet como extractor de características en nuestro modelo como se pudo observar en la Figura 11.

CAPÍTULO 5

CONJUNTO DE DATOS PRUEBA

Como se vio en el capítulo 2 algunos autores crearon sus propios conjuntos de datos como es el caso de Li et al. [9], Rossler et al. [8], y Dolhansky et al. [2]. Dado que en su momento no existían conjuntos de datos que estuviesen formados en su totalidad por videos para la técnica de intercambio de identidad.

A la fecha del presente trabajo existen dos generaciones de conjuntos prueba para dicha técnica: la primera generación con pocos videos, de baja resolución y con personas hablando a la cámara; la segunda se generó con más videos, de resoluciones más altas y con diversos contextos de iluminación, profundidad y poses de personas. En la Tabla 7 se presentan las bases de datos de prueba para el intercambio de identidad.

UADFV es una colección de 98 videos etiquetados, 49 reales y 49 modificados por la técnica de intercambio de identidad DeepFake, creada por Y. Li, X. Yang, P. Sun, H. Qi, y S. Lyu, cada video tiene una duración de entre 6-16 segundos, con tamaños variados entre 300x200 pixeles a 600x400 pixeles, con poca variedad de movimiento de luminosidad, con rostros lo más alineados a la cámara, considerada como base de datos de primera generación.

Tabla 7 Base de datos de prueba para Intercambio de Identidad, extraída de [4].

1st Generation		
Database	**Real Videos**	**Fake Videos**
UADFV (2018) [82]	49 (Youtube)	49 (FakeApp)
DeepfakeTIMIT (2018) [1]	-	620 (faceswap-GAN)
FaceForensics++ (2019) [12]	1,000 (Youtube)	1,000 (FaceSwap) 1,000 (DeepFake)
2nd Generation		
Database	**Real Videos**	**Fake Videos**
DeepFakeDetection (2019) [86]	363 (Actors)	3,068 (DeepFake)
Celeb-DF (2019) [26]	890 (Youtube)	5,639 (DeepFake)
DFDC Preview (2019) [83]	1,131 (Actors)	4,119 (Unknown)

Celef-db es una colección más extensa con 6,529 videos etiquetados, 890 reales y 5,639 modificados por la técnica de intercambio de identidad DeepFake, creada por Y. Li et al. Cada uno con una duración de entre 6-16 segundos, con tamaños variados y con diferentes condiciones lumínicas y ángulos de cámara, considerada como base de datos de segunda generación.

Según los datos de Tolosana et al. [4], algunos modelos analizados en su momento solían tener dificultades en alcanzar buena certidumbre al medir el AUC con bases de datos de segunda generación, por lo que se utilizara UADFV como una base de datos de primera generación. Para trabajos futuros se utilizará Celef-db

Con esto termina la selección de componentes, mismos que se pueden observar en la Figura 11, los siguientes capítulos se centraran en explicar los parámetros a calibrar y los experimentos que se llevaron a cabo.

CAPÍTULO 6

CALIBRACIÓN DE LOS PARÁMETROS PARA LOS COMPONENTES DE LA APLICACIÓN

En este capítulo se retomarán los elementos del modelo de la Figura 11, dichos elementos son: Openface2 [12] para el extractor de rostros y EfficientNet [14] para el extractor de características. Como se vio en los capítulos 3 y 4 la elección de estos componentes influye en el modelo; pero no solo basta con implementar estos elementos para alcanzar una exactitud comparable con el estado del arte. Para cada extractor se le deberán de colocar distintos parámetros para alcanzar dicha exactitud, en este capítulo se dará una breve explicación de estos y se divide en secciones para facilitar la lectura.

El extractor de rostros al ser un toolkit se deberá explicar cuál es la estructura que arroja al procesar un video, cuáles son los archivos que nos interesan y cuál es la técnica utilizada para filtrar información y refinar este proceso. Para el extractor de características se explicarán los parámetros que utiliza, dando una explicación de cada uno y tomando alternativas a cuáles son los parámetros sugeridos por experimentos de diversos autores.

6.1 CALIBRACIÓN DE EXTRACTOR DE ROSTROS

Como se mencionó en el capítulo 3, Openface2 es un toolkit el cual nos permite estimar Landmarks, la pose de la cabeza, la estimación de la mirada y las expresiones faciales, por lo que es de esperar que este genere una serie de archivos al ser utilizado, cabe destacar que esta herramienta nos permite introducir tanto imágenes como videos; los cuales son interpretados como una colección de imágenes consecutivas llamados fotogramas, con lo que los archivos que son generados al introducir una imagen y un video son muy similares; solo varia la cantidad de los archivos .

Openface2 tiene diversas herramientas [26], las cuales son: FeatureExtraction la que nos permite analizar una secuencia de fotogramas que contienen solo un rostro; FaceLandmarkVid el cual realiza la misma acción que FeatureExtraction pero con múltiples rostros; FaceLandmarkVidMulti esté es similar a FaceLandmarkVid pero en múltiples videos; FaceLandmarkImg que nos permite analizar solo una imagen. De estas 4 herramientas se utilizará FaceLandmarkVidMulti para extraer los rostros de videos, pues al permitir múltiples videos facilita la extracción de toda la base de datos.

Los archivos que genera Openface2 son los siguientes: un archivo de video con terminación ".avi" el cual es una copia del video procesado, añadiendo la pose de la cabeza con un recuadro azul, marcando la estimación de Landmarks con círculos rojos con centro azul y proyectando la estimación de la mirada con líneas verdes. Un archivo de historial de gradientes orientados con terminación ".hog". Una carpeta con el nombre del video seguido de un guion medio y la leyenda "aligned" en el que se encuentran los rostros extraídos del video, alineados y con la nomenclatura "frame_det_" seguido de dos valores numéricos, los cuales indican a que rostro se asocian y finalizando con el número del fotograma en el que aparece dicho rostro. Los valores "00" aluden al rostro que se repite mayormente en el video, los valores "01" se asignan al segundo rostro con más aparición en el video y así sucesivamente. Un archivo separado por comas con terminación ".csv" en el que se arroja toda la información detallada de cada fotograma, mostrando si se encontró un rostro, si la extracción de este fue exitosa, la posición de la estimación de la mirada en coordenadas x,y,z para cada ojo, además de mostrar cada uno de los Landmarks en dos y tres dimensiones y las expresiones faciales. Finalmente se crea un archivo de texto plano con la terminación ".txt" y la leyenda "of_details" detallando los archivos antes mencionados. Estos archivos se pueden ver en la Tabla 8.

Tabla 8 Archivos generados por Openface2

Archivo	Descripción
Video con terminación “.avi”	Copia del video procesado con Landmarks, estimación de la mirada y pose de la cabeza.
Historial de gradientes orientados con terminación “.hog”	Contiene los gradientes orientados del video.
Carpeta con rostros alineados con la leyenda “aligned”	Contiene todos los rostros extraídos y alineados con la nomenclatura “frame_det_” seguido de dos valores numéricos y finalizando con el número de fotograma.
Archivo separado por comas con terminación “.csv”	Contiene la información detallada de cada fotograma, Landmarks, pose de la cabeza, estimación de la mirada y expresiones faciales.
Archivo de texto plano con la terminación “.txt”	Muestra el nombre del archivo original y los nombres de los archivos antes mencionados.

Para nuestro modelo algunos de estos archivos no nos serán de utilidad y solo utilizaremos la carpeta de rostros alineados y el archivo separado por comas. A los cuales les realizamos el siguiente tratamiento para limitar los rostros a sólo uno y eliminar posibles detecciones de falsos positivos. Estos falsos positivos se dan por que el toolkit hace una estimación de Landmarks para los casos en los que partes de este no se encuentran visibles y existen casos en los cuales se estiman dichos Landmarks en fotogramas sin rostros. En caso de que existan fotogramas, que no contengan un rostro en la etapa de entrenamiento del extractor de características, existe la posibilidad de que nuestro modelo encuentre características erróneas, por lo que es importante una etapa de limpieza.

Para limitar los rostros encontrados a uno, bastará con eliminar los que no se repiten en el video, esto se realizará borrando las imágenes que estén en la carpeta de rostros alineados y que tengan un valor distinto de “00”.

Para la eliminación de posibles falsos positivos se realizarán dos alternativas, la primera será buscar en el archivo separado por comas en la columna de “face_id” que mostrará un 0 cuando se detecte un rostro y un 1 en otro caso. Además la

columna de “success” muestra si el proceso de extracción fue exitoso con un 1, y 0 en cualquier otro caso. Así en esta etapa de limpieza se iterará sobre este archivo eliminando cualquier fotograma que tenga valores distintos a 0 en la columna “face_id” y 1 en “success”.

En caso de existir falsos positivos, que pasen estas etapas y no sean eliminados, se sugiere pasar la carpeta nuevamente al toolkit como entrada, repetir el proceso de limpieza y así asegurar que las imágenes solo contengan rostros. Dicho proceso se resume en la Tabla 9 como algoritmo.

6.2 CALIBRACIÓN DE EXTRACTOR DE CARACTERÍSTICAS

Como se mencionó en el capítulo 4, una CNN es una generalización de una red neuronal y gracias a que utilizaremos EfficientNet no tendremos que preocuparnos por el diseño de esta; en cuanto a número de capas ocultas y esto nos ayuda reduciendo la cantidad de hiperparámetros que forman dicha red.

Tong Yu et al. [27] mencionan que los hiperparámetros son valores que no pueden actualizarse durante el entrenamiento del aprendizaje automático. Estos pueden ser propios de la construcción de la arquitectura del modelo: número de capas ocultas y función de activación; en la determinación de la eficiencia y exactitud durante el entrenamiento: tasa de aprendizaje (LR) del gradiente estocástico, el tamaño del lote y el optimizador. También mencionan que la correcta elección de estos hiperparámetros, se realiza mediante la experiencia de expertos y la utilización de otros hiperparámetros de experimentos similares o mediante la prueba de cada uno de estos. A continuación, se explicarán brevemente dichos hiperparámetros y cuáles son sus valores sugeridos.

Tabla 9 Algoritmo para garantizar rostros en la base de datos.

1	*Iterar la carpeta de rostros alineados.*
2	*Comparar si el título del rostro no contiene un valor *_00_*.*
3	*Eliminar dicha imagen.*
4	*Iterar el archivo separado por comas.*
5	*Comprobar la columna face_id diferente a 0 o la columna success diferente a 1*
6	*Buscar dicha imagen en la carpeta de rostros alineados y eliminarla.*
7	*Utilizar la carpeta alineada como entrada y ejecutar nuevamente Openface2.*
8	*Repetir los pasos 1,2,3,4,5 y 6.*

6.2.1 TAMAÑO DE LOTE

También llamado mini lote, es un hiperparámetro que indica el número de muestras por actualización de gradiente. Está relacionado con el optimizador y permite "acelerar el proceso de entrenamiento, especialmente en un gran conjunto de datos" [27]. Reduce el ruido y aumenta la probabilidad de convergencia según menciona Andrew Ng en [28], además está altamente relacionado con la memoria de la unidad de cálculo, por lo que Tang sugiere que sea un múltiplo de 2.

6.2.2 TASA DE APRENDIZAJE LR

Este es un escalar positivo, determina el tamaño de paso durante el descenso del gradiente estocástico (SDG) y es crucial para determinar la velocidad de convergencia del modelo. Su ajuste mejora la exactitud de este según Bengio Y. [29] El LR puede ser fijada en un valor o puede utilizar diversas técnicas para variar en el transcurso de las iteraciones. Para nuestro modelo se utilizará el LR por defecto en el Optimizador.

6.2.3 OPTIMIZADOR

El optimizador o algoritmo de optimización permite generar pesos utilizando el gradiente de la función de coste, para cada peso de la red. Dado que la función busca minimizar el error, modifica dicho peso en dirección negativa del gradiente y multiplicando dicho vector por un LR, agilizando la convergencia de dicha función a su mínimo.

Los métodos iterativos que reducen la búsqueda de mínimos locales son conocidos como métodos de optimización basados en descenso del gradiente [30]. La selección de un optimizador adecuado es una tarea complicada [27], por lo que nos basaremos en la elección de optimizadores en tareas similares [30] con el uso de los optimizadores RMSprop y Adam.

RMSprop (Root Mean Square Propagation) es uno de los optimizadores más utilizados en el entrenamiento de redes neuronales profundas [31]. Similar a Adadelta y Adagrad [32] mantiene un LR diferente para cada dimensión, pero en este caso dicho LR se realiza dividiéndolo por la media del declive exponencial del cuadrado de los gradientes.

El algoritmo Adam (Adaptive moment estimation) combina las bondades de AdaGrad y RMSProp. Se mantiene un LR por parámetro y además de calcular RMSProp, cada LR también se ve afectado por la media del momentum del gradiente [31]. Adam añade corrección de sesgos y momentum a RMSprop, lo que le permite superar ligeramente a RMSprop en la última fase de la optimización. Calcula la media ponderada exponencial, así como los cuadrados de los gradientes pasados, además conserva una media exponencial decreciente de los gradientes pasados.

Para la etapa de experimentos **utilizaremos a RMSprop y Adam como posibles optimizadores**, dado que existen varios experimentos [27] [30] [31] mostrando su eficiencia en tareas similares de clasificación de imágenes. Aunque cabe remarcar que **este campo es el más prometedor en cuanto a avances y es un nicho de oportunidad dado que ningún equipo ha hablado de establecer un optimizador heurístico para esta etapa**.

6.2.4 FUNCIÓN DE PÉRDIDA

Como se mencionó anteriormente, los algoritmos de aprendizaje profundo utilizan un enfoque de descenso de gradiente estocástico para optimizar y aprender objetivos. Otra forma de asegurar que dicho aprendizaje sea rápido y preciso es asegurar la representación matemática de los objetivos, también conocida como pérdida, que sea capaz de cubrir incluso los casos límites [33]. Algunas funciones de pérdida ampliamente usadas en la clasificación se agrupan en 4 categorías: Basadas en la distribución, Basadas en la región, Basadas en los límites y Compuestas, como se puede ver en la Tabla 11 y se puede observar más a detalle que hace cada función de pérdida en la Tabla 10.

Tabla 10 Funciones de pérdida explicadas extraída de [33].

Función de pérdida	Casos de uso
Entropía cruzada binaria	Funciona mejor en escenarios de distribución equitativa de datos entre clases. Función de pérdida basada en la distribución Bernoulli. Entropía cruzada categórica de la distribución Multinoulli.
Entropía cruzada ponderada	Ampliamente utilizado con conjuntos de datos sesgados Pondera los ejemplos positivos por el coeficiente β.
Entropía cruzada equilibrada	Similar a la entropía cruzada ponderada, utilizada ampliamente con conjuntos de datos sesgados pondera tanto los ejemplos positivos como los negativos por β y 1 - β respectivamente.
Pérdida focal	Funciona mejor con un conjunto de datos muy desequilibrados ponderar a la baja la contribución de los ejemplos fáciles, permitiendo que el modelo aprenda los ejemplos difíciles.
Término de penalización por pérdida derivada del mapa de distancia	Variante de la entropía cruzada. Se utiliza para los límites difíciles de segmentar.
Pérdida de Dice	Inspirado en el coeficiente de Dice, una métrica para evaluar los resultados de la segmentación. Como el coeficiente de Dice no es convexo por naturaleza, se ha modificado para hacerlo más manejable.
Pérdida de sensibilidad-especificidad	Inspirado en las métricas de sensibilidad y especificidad Se utiliza para los casos en los que hay más atención a los Verdaderos Positivos.
Pérdida de Tversky	Variante del Coeficiente de Dice Añade peso a los falsos positivos y a los falsos negativos.
Pérdida focal de Tversky	Variante de la pérdida de Tversky centrada en ejemplos difíciles.
Pérdida de Dice de Log-Cosh	Variante de Pérdida de Dice y enfoque log-cosh de regresión inspirado para el suavizado. Se pueden utilizar variaciones para un conjunto de datos sesgados.
Pérdida de distancia de Hausdorff	Inspirado en la métrica de la distancia de Hausdorff utilizada para evaluar la segmentación La pérdida aborda la naturaleza no convexa de la métrica de la distancia añadiendo algunas variaciones.
Pérdida de conciencia de la forma	Variación de la pérdida de entropía cruzada añadiendo un coeficiente basado en la forma utilizado en casos de límites difíciles de segmentar.
Combo Pérdida	Combinación de pérdida de Dice y entropía cruzada binaria Utilizado para la clase ligeramente desequilibrada aprovechando los beneficios del BCE y la pérdida de Dice.
Pérdida logarítmica exponencial	Función combinada de pérdida de Dice y entropía cruzada binaria Se centra en los casos de menor exactitud de predicción.
Pérdida de similitud estructural maximizada por correlación	Se centra en la estructura de segmentación. Se utiliza en casos de importancia estructural, como las imágenes médicas.

Tabla 11 Funciones de pérdida por categoría extraída de [33].

Tipo	Función de pérdida
Pérdida basada en la distribución	Entropía cruzada binaria Entropía cruzada ponderada Entropía cruzada equilibrada Pérdida focal Término de penalización por pérdida derivada del mapa de distancia
Pérdida basada en la región	Pérdida de Dice Pérdida de sensibilidad especificidad Pérdida de Tversky Pérdida focal de Tversky Pérdida de Dice de Log-Cosh
Pérdida basada en los límites	Pérdida de distancia de Hausdorff Pérdida de conciencia de la forma
Pérdida acumulada	Combo Pérdida logarítmica exponencial

Dado que nuestro modelo clasificará imágenes en un conjunto balanceado y solo lo hará entre dos clases. Utilizaremos la entropía cruzada categórica, la cual es una variación de entropía cruzada binaria especializada en esta tarea [33] [34].

6.2.5 MÉTRICA DE EVALUACIÓN

Otro parámetro para tener en consideración es la métrica, que se utilizará para evaluar el modelo. En general, la métrica de evaluación se describe como la herramienta que mide el rendimiento del clasificador [35]. Las métricas de evaluación pueden utilizarse en tres tipos distintos de aplicación [36]: Para evaluar la capacidad de generalización del clasificador entrenado, donde la exactitud o la tasa de error es una de las métricas más comunes. Como evaluador para la selección de modelos, cuyo objetivo es determinar el mejor clasificador entre los diferentes tipos de clasificadores entrenados. Finalmente, como un discriminador para seleccionar la solución óptima, entre todas las soluciones generadas durante el entrenamiento de la clasificación.

Tabla 12 Matriz de confusión para la clasificación binaria.

	Clase positiva prevista	**Clase negativa prevista**
Clase positiva real	Verdadero positivo (tp)	Falso negativo (fn)
Clase real negativa	Falso positivo (fp)	Verdadero negativo (tn)

Para la primera y segunda aplicación, cualquier métrica de evaluación pueden aplicarse como mencionan Hossin et al. [35], pero para la tercera aplicación es necesario prestar atención a la métrica a utilizar. Para los problemas de clasificación binaria, la evaluación de la discriminación, de la solución óptima durante el entrenamiento de la clasificación, puede definirse basándose en la matriz de confusión como se puede observar en la Tabla 12. A continuación se presentarán las métricas de evaluación más usadas para clasificar imágenes.

6.2.5.1 EXACTITUD (ACC)

La exactitud, es la métrica de evaluación más utilizada en la práctica de diversos problemas de clasificación. La calidad de la solución se evalúa en base al porcentaje de predicciones correctas sobre el total de instancias, dicho calculo se realiza con la Ecuación 1. Una de sus ventajas es la facilidad de calcularla, se puede aplicar a varias clases y etiquetas; su puntuación es fácil de usar y entender para cualquier persona. Se sugiere su aplicación cuando los conjuntos de datos son balanceados [35]. Aunque tiene limitaciones en procesos de evaluación y discriminación, como lo es producir valores menos distintivos y discriminantes [37] [38], y en consecuencia puede llegar a determinar un clasificador subóptimo.

$$Accuracy = \frac{tp + tn}{tp + fp + tn + fn} \tag{1}$$

Ecuación 1 Formula para calcular la exactitud.

6.2.5.2 ERROR CUADRÁTICO MEDIO (MSE)

MSE mide la diferencia entre las soluciones predichas y las soluciones deseadas. Cuanto menor sea el valor de MSE, mejor será el entrenamiento del modelo. El MSE se calcula con la Ecuación 2.

$$MSE = \frac{1}{n}\sum_{j=1}^{n}\left(P_j - A_j\right)^2 \tag{2}$$

Ecuación 2 Formula para calcular MSE.

Donde P_j es el valor predicho de la instancia j, A_j es el valor objetivo real de la instancia j y n es el número total de instancias. Al igual que la exactitud, las limitaciones de MSE es que no proporciona información de compensación entre los datos de las clases y esto lleva a soluciones subóptimas. Además de depender de la inicialización de los pesos para las clases minoritarias.

6.2.5.3 ÁREA BAJO LA CURVA ROC (AUC)

El AUC es una de las métricas de clasificación más populares. El AUC refleja el rendimiento general de un clasificador, para un problema de dos clases, el AUC se calcula con la Ecuación 3 [39].

$$AUC = \frac{S_p - \frac{n_p(n_n + 1)}{2}}{n_p n_n} \tag{3}$$

Ecuación 3 Formula para calcular AUC.

Donde, S_p es la suma de todos los ejemplos positivos clasificados, $n_p \, y \, n_n$ son el número de ejemplos positivos y negativos respectivamente. Según Huang et al. [40] en teoría AUC es mejor que utilizar exactitud al evaluar el rendimiento del clasificador y elegir una solución óptima durante el entrenamiento. Aunque tiene el inconveniente de consumir más poder computacional, pues en problemas multiclase la complejidad del tiempo es $O(|C|^2 n \log n)$ para modelos AUC de Hand y Till [39].

Para finalizar, recapitularemos los hiperparámetros que optimizaremos en el siguiente capítulo de experimentos. El Optimizador que probaremos será RMSprop y Adam; el número de épocas, en la cual variaremos dicho valor. Utilizaremos la función de pérdida entropía cruzada categórica y las métricas ACC, AUC y MSE. Además de que en este punto tenemos las bases de datos como imágenes de rostros alineadas.

Tabla 13 Crecimiento de potencia de cómputo por familias EfficientNet.

Familia	Resolución	UADFV [7]	Celef-db [9]	Número de parámetros
B0	224	4.88GB +modelo	52.59 GB + modelo	4052133
B1	240	5.60 GB +modelo	60.37 GB + modelo	6577801
B2	260	6.57 GB +modelo	70.85 GB + modelo	7771387
B3	300	8.75 GB +modelo	94.33 GB + modelo	10786609
B4	380	14.03 GB +modelo	151.35 GB +modelo	19000000
B5	456	20.20 GB +modelo	217.94 GB +modelo	30000000
B6	528	27.09 GB +modelo	292.20 GB +modelo	43000000
B7	600	34.98 GB +modelo	377.32 GB +modelo	66000000

6.3 REQUERIMIENTOS DE LA POTENCIA DE COMPUTO

Una vez revisado los optimizadores y teniendo las bases de datos como imágenes dando un total de: 34,778 imágenes para UADFV y 375,136 para Celef-db. Observamos como la potencia de cómputo requerida para entrenar los modelos de EfficientNet crece de acuerdo con la Tabla 13. En la cual al utilizar la Ecuación 4.

$$Capacidad = res^2 * chanel * n \tag{4}$$

Ecuación 4 Formula para calcular la potencia de cómputo requerida.

Donde res es la resolución de la familia a utilizar, chanel es el número de canales de la imagen 3 para RGB y 1 para escala de gris, y n el número de imágenes que se utilizará para el entrenamiento.

Como se puede observar en la Tabla 13, la potencia de cómputo crece al aumentar la familia de EfficientNet, superando rápidamente lo que entendemos como un ordenador personal y dando problemas con la potencia requerida, por lo que se sugiere una alternativa con la que pueda entrenar CNN con menor potencia de cómputo.

6.4 APRENDIZAJE PARTICIONADO

Como alternativa para operar con tensores (conjuntos de datos) que superen la potencia de cómputo disponible. Se propone ocupar lo que llamamos Aprendizaje particionado (AP). El AP se realiza dividiendo un tensor para la resolución del problema, entrenando la red con segmentos del tensor y ocupando los modelos resultantes con otros segmentos distintos del mismo, para reducir la potencia de cómputo necesaria y así disminuir el tiempo de entrenamiento. En la Tabla 14 se mencionan el algoritmo de AP.

En el siguiente capítulo se realizarán experimentos para probar los hiperparámetros mencionados en este capítulo, los cuales son: el optimizador, el número de épocas, la familia de EfficientNet y el AP. Los hiperparámetros que se fijarán para cada experimento serán: el tamaño del lote, la función de perdida y las métricas.

Tabla 14 Algoritmo para AP.

1	*Partir el tensor T en dos subconjuntos A y B, asignándoles el D% e ID% de datos respectivamente.*
2	*Crear un modelo M entrenado en 15 épocas, con el D% de datos del subconjunto A.*
3	*Utilizar el ID% de datos del subconjunto B para entrenar el modelo M, en 5 épocas. Y producir los Modelos D% + ID%*

Capítulo 7

EXPERIMENTOS

Este capítulo contendrá la definición de los experimentos, la metodología usada, las características del equipo de cómputo y el lenguaje utilizado para programarlos; cada experimento se presentará como secciones con las tablas de los resultados separadas por exactitud, AUC, MSE y tiempo, finalmente se colocará una sección de discusión de dichos resultados. Cada experimento se realizó con la base de datos UADFV.

7.1 DEFINICIÓN DE EXPERIMENTOS

Como se mencionó en el capítulo 6, con el objetivo de alcanzar una exactitud comparable con el estado del arte, se realizarán experimentos probando distintos valores de número de épocas, distintos optimizadores y probando el comportamiento de las diferentes familias de EfficientNet. finalmente se probará la alternativa de AP.

A partir de la sección 7.4 empezaran dichos experimentos nombrados : Optimizador, en el cual se probará Adam y RMSProp; Variación de épocas, en el cual se utilizarán 7,10 y 15 épocas; variación de familias EfficientNet, donde use probaran B0,B1,B2 y B3; finalmente el experimento AP, donde se implementarán distintos modelos para comparar con esta técnica.

7.2 METODOLOGÍA

Para realizar los experimentos antes mencionados se contó con las imágenes extraídas y alineadas por medio de Openface2 como se mencionó en el capítulo anterior. Posteriormente se creó un tensor formado por imágenes colocadas en arreglos con valores de entre 0 y 255, con dimensiones como se muestra en la Tabla 13. Se colocaron las etiquetas de dichas imágenes en un arreglo de vectores donde [1,0] identificará rostros falsos, y [0,1] rostros reales o verdaderos, dicho proceso se puede observar en la Figura 17.

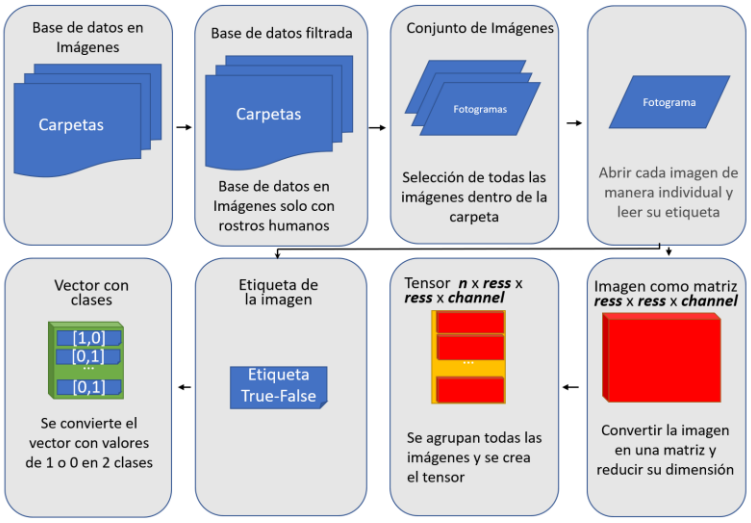


Figura 17 Proceso para la creación de tensor.

Para el entrenamiento de cada modelo, se seleccionó una muestra de imágenes hasta alcanzar la cantidad mostrada en la Tabla 15, donde el porcentaje corresponde a la totalidad de imágenes de la base de datos UADFV. A esta muestra se le llamará subtensor, donde la mitad corresponde a imágenes falsas y la mitad a verdaderas, esto para tener un subconjunto balanceado.

Se utilizó clasificación a 5 pliegues con el 80% de los datos para entrenamiento (training) y 20% para pruebas (test), partiendo cada subtensor en 5 partes de igual tamaño y asegurándonos de que cada parte esté al menos una vez en el conjunto de pruebas y en el conjunto de entrenamiento, como se puede observar en la Figura 18.

Tabla 15 Nombre de los modelos y cantidad de imágenes

Nombre Modelo	Imágenes
EF0 10%	3477
EF0 20%	6956
EF0 50%	17389
EF0 100%	34778

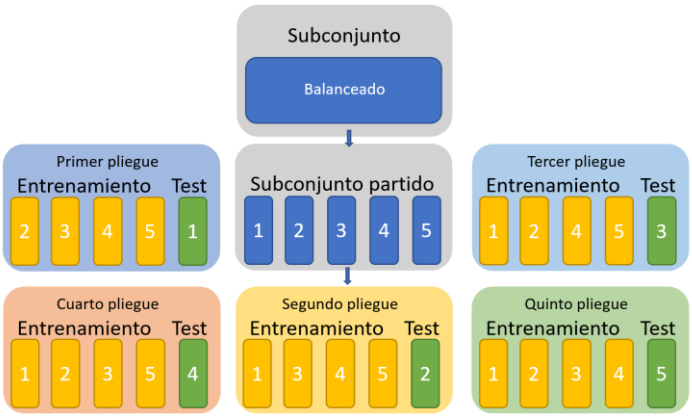


Figura 18 Explicación de ejercicios a 5 pliegues.

7.3 EQUIPO UTILIZADO

Para la realización de los experimentos se utilizó el siguiente equipo:

Hardware: como se mencionó en el capítulo anterior y con la Tabla 13 donde se muestra la potencia de cómputo requerida, se utilizó el siguiente equipo de cómputo el cual cuenta con un procesador de 6 núcleos y 12 hilos Ryzen 2600 a una frecuencia base de 3.4 GHz, con un disco duro solido de 240 GB, 32 GB de memoria RAM DDR4 a 3.2 MHz, finalmente una tarjeta Nvidia RTX 3070 con 8 GB de VRAM.

Software: Para la realización de los experimentos se utilizó una instalación de Ubuntu 20.04 LTS, Docker 20.10.14, Openface2, y EfficientNet mediante TensorFlow 2.6.1 como contenedor Docker utilizando Python 3.8.10 para la programación con cuadernos Jupyter 6.4.5, se creó una guía de instalación que se agrega como Anexo A.

7.4 EXPERIMENTO OPTIMIZADOR

Se utilizaron 15 épocas y se probaron los optimizadores Adam y RMSProp, los resultados se muestran en las Tabla 16, Tabla 17, Tabla 18 y Tabla 19 correspondientes a la exactitud, AUC, MSE y tiempo respectivamente.

Tabla 16 Resultados de Optimizador medidos en exactitud.

	RMSProp				Adam			
	10%		20%		10%		20%	
Modelo	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.99024	0.97356	**0.99123**	0.986133	0.98154	**0.95804**	0.9829	0.98592
EF1	0.99076	0.97413	0.99066	0.9866	0.98576	0.977	0.9879	0.98536
EF2	0.9876	0.96695	0.9886	0.98135	0.97645	0.9820	0.9791	0.9856

Tabla 17 Resultados del Optimizador medidos en AUC

	RMSProp				Adam			
	10%		20%		10%		20%	
Modelo	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.99888	0.9884	0.99863	0.99753	0.99564	**0.97024**	0.99858	0.99854
EF1	0.9984	0.9944	0.99893	0.99913	0.99823	0.99743	0.99873	0.99776
EF2	0.99655	0.99445	0.998	0.99725	0.99335	0.99445	0.9969	**0.99915**

Tabla 18 Resultados del Optimizador medidos en MSE

	RMSProp				Adam			
	10%		20%		10%		20%	
Modelo	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.00714	0.022	**0.00683**	0.00983	0.0145	0.03862	**0.01134**	0.01016
EF1	0.00756	0.02083	0.007	0.0088	0.0104	0.015633	0.00906	0.0107
EF2	0.0098	0.02635	0.0087	0.0132	0.0191	0.01635	0.01575	0.0104

Tabla 19 Resultados del Optimizador medidos en segundos.

	RMSProp		Adam	
Modelo	10%	20%	10%	20%
EF0	236.7886	493.6004	237.3528	**474.7055**
EF1	412.6683	815.4348	**407.8955**	**801.8612**
EF2	569.6143	1148.95	**547.8574**	**1083.773**

7.5 EXPERIMENTO VARIACIÓN DE NÚMERO DE ÉPOCAS

Se utilizaron 7, 10 y 15 épocas con el optimizador Adam, los resultados se muestran en las Tabla 20, Tabla 21, Tabla 22 y Tabla 23 correspondientes a la exactitud, AUC, MSE y tiempo respectivamente.

Tabla 20 Resultados de Épocas medido en exactitud.

	7 épocas				10 épocas				15 épocas			
	10%		20%		10%		20%		10%		20%	
	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.97052	**0.93276**	0.987767	0.988	0.98684	0.97702	0.9826	0.9854	0.98154	0.95804	0.9829	0.98592
EF1	0.982033	0.9818	0.983833	0.978467	0.987767	0.983267	0.9864	**0.988267**	0.985767	0.977	0.9879	0.985367
EF2	0.97665	0.95835	0.97565	0.98635	0.9797	0.98635	0.9837	0.98815	0.97645	0.98205	0.97915	0.9856

Tabla 21Resultados de Épocas medido en AUC

	7 épocas				10 épocas				15 épocas			
	10%		20%		10%		20%		10%		20%	
	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.9915	0.9767	0.998733	0.9977	0.9981	0.979818	0.9966	0.999067	0.99564	**0.97024**	0.99858	0.99854
EF1	0.996367	0.9981	0.9976	0.995433	0.998333	0.997367	0.999	**0.9997**	0.998233	0.997433	0.998733	0.997767
EF2	0.9934	0.9913	0.9956	0.99935	0.9966	0.99665	0.9978	0.9992	0.99335	0.99445	0.9969	0.99915

Tabla 22 Resultados de Épocas medido en MSE.

	7 épocas				10 épocas				15 épocas			
	10%		20%		10%		20%		10%		20%	
	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.02254	0.05336	0.009533	0.0087	0.01038	**0.2138**	0.0136	0.0094	0.01458	0.03862	0.01134	0.01016
EF1	0.014333	0.01216	0.0121	0.0172	0.010033	0.013167	0.0097	**0.007667**	0.0104	0.015633	0.009067	0.0107
EF2	0.02	0.0286	0.0181	0.00985	0.01635	0.01035	0.0126	0.00945	0.0191	0.01635	0.01575	0.0104

Tabla 23 Resultados de Épocas medido en segundos.

	7 épocas		10 épocas		15 épocas	
	10%	20%	10%	20%	10%	20%
EF0	**114.8176**	**223.7785**	163.1994	315.9875	237.3528	474.7055
EF1	**204.6015**	**385.4686**	284.9488	546.2879	407.8955	801.8612
EF2	**258.3977**	**505.8778**	366.7396	731.605	547.8574	1083.773

7.6 EXPERIMENTO VARIACIÓN DE FAMILIA EFFICIENTNET

Se utilizaron 15 épocas con el optimizador Adam variando las familias de EfficientNet y el total de la base de datos UADFV, los resultados se muestran en las Tabla 24, Tabla 25, Tabla 26 y Tabla 27 correspondientes a la exactitud, AUC, MSE y tiempo respectivamente.

Tabla 24 Resultados de Familias medido en exactitud.

	10 %		20 %		50 %		100 %	
	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.98154	0.95804	0.9829	0.98592	0.98728	0.98482	0.9958	**0.9977**
EF1	0.985767	0.977	0.9879	0.985367	0.9857	0.9878	0.9957	**0.9945**
EF2	0.97645	0.98205	0.97915	0.9856	0.98685	0.9885	0.9957	**0.9945**
EF3	0.9788	0.98635	0.98445	0.98385	0.9861	0.9871	x	x

Tabla 25 Resultados de Familias medido en AUC.

	10 %		20 %		50 %		100 %	
	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.99564	0.97024	0.99858	0.99854	0.99956	0.99922	0.9998	**0.9998**
EF1	0.998233	0.997433	0.998733	0.997767	0.9993	0.9997	0.9998	**1**
EF2	0.99335	0.99445	0.9969	0.99915	0.9994	**0.99955**	0.9996	0.99461
EF3	0.9961	**0.9995**	0.99875	0.9991	0.9993	0.9988	x	x

Tabla 26 Resultados de Familias medido en MSE.

	10 %		20 %		50 %		100 %	
	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.01458	0.03862	0.01134	0.01016	0.0079	0.00912	0.0027	**0.0018**
EF1	0.0104	0.015633	0.009067	0.0107	0.00865	0.00685	0.0027	**0.0033**
EF2	0.0191	0.01635	0.01575	0.0104	0.00835	**0.0077**	0.0034	0.0628
EF3	0.0162	**0.0085**	0.01085	0.01215	0.0085	0.0119	x	x

Tabla 27 Resultados de Familias medido en segundos.

	10 %	20 %	50 %	100 %
EF0	**237.3528**	474.7055	2381.919	4270.279
EF1	**407.8955**	801.8612	2238.078	9934.17
EF2	**547.8574**	1083.773	3551.373	19571.17
EF3	**979.3758**	1948.381	7251.882	x

7.7 EXPERIMENTO AP

Se utilizaron los modelos del experimento anterior como modelos base según el paso 2 de la Tabla 14, con el optimizador Adam, los resultados se muestran en la Tabla 28, Tabla 29, Tabla 30 y Tabla 31 correspondientes a la exactitud, AUC, MSE y tiempo respectivamente.

Tabla 28 Resultados de AP medido en exactitud.

	20 %		10%+ 10%		50 %		20%+30 %		100 %		50%+50%	
	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.9829	**0.98592**	0.984433	0.970767	0.98728	0.98482	0.98685	**0.9878**	0.9958	**0.9977**	0.99615	0.99515
EF1	0.9879	0.985367	0.98615	**0.98635**	0.9857	0.9878	0.99005	**0.9892**	0.9957	**0.9945**	0.99506	0.99278
EF2	0.97915	**0.9856**	0.9858	0.982	0.98685	**0.9885**	0.98255	0.98755	0.9957	0.9945	0.99535	**0.99505**
EF3	0.98445	0.98385	0.9795	**0.98565**	0.9861	**0.9871**	0.98405	0.9866	x	x	x	x

Tabla 29 Resultados de AP medido en AUC.

	20 %		10%+ 10%		50 %		20%+30 %		100 %		50%+50%	
	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.99858	**0.99854**	0.997667	0.994567	0.99956	0.99922	0.9996	**0.99975**	0.9998	**0.9998**	0.99985	0.99955
EF1	0.998733	0.997767	0.9987	**0.99965**	0.9993	**0.9997**	0.99875	0.9995	0.9998	**1**	0.99964	0.99984
EF2	0.9969	0.99915	0.9982	**0.9993**	0.9994	**0.99955**	0.99845	0.9995	0.9996	0.99461	0.9997	**0.9998**
EF3	0.99875	**0.9991**	0.9968	0.9988	0.9993	0.9988	0.99905	**0.99965**	x	x	x	x

Tabla 30 Resultados de AP medido en MSE.

	20 %		10%+ 10%		50 %		20%+30 %		100 %		50%+50%	
	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test	Training	Test
EF0	0.01134	**0.01016**	0.011667	0.021933	0.0079	0.00912	0.0075	**0.0063**	0.0027	**0.0018**	0.0024	0.00345
EF1	0.009067	0.0107	0.0105	**0.00775**	0.00865	**0.00685**	0.0076	0.0072	0.0027	**0.0033**	0.0035	0.0049
EF2	0.01575	**0.0104**	0.01115	0.0108	0.00835	**0.0077**	0.012	0.0096	0.0034	0.0628	0.0033	**0.0035**
EF3	0.01085	0.01215	0.0153	**0.0091**	0.0085	0.0119	0.01085	**0.0082**	x	x	x	x

Tabla 31 Resultados de AP medido en segundos.

	20 %	10%+10%	50 %	20%+30 %	100 %	50%+50%
EF0	474.7055	**322.5955**	2381.919	**713.6824**	4270.279	**3312.106**
EF1	801.8612	**551.1468**	2238.078	**1215.976**	9934.17	**4511.613**
EF2	1083.773	**738.694**	3551.373	**1631.983**	19571.17	**5895.888**
EF3	1948.381	**1330.615**	7251.882	**2930.39**	x	x

7.8 DISCUSIÓN DE RESULTADOS.

Todos los datos mencionados a continuación se obtuvieron utilizando la base de datos UADF.

Partiendo de lo visto en el capítulo 6 se evaluó el desempeño al utilizar el optimizador Adam y RMSProp como primer hiperparámetro. En la Tabla 16 medida en exactitud el valor más alto fue de 0.991233 del modelo EF0 20% RMSProp en entrenamiento, el valor más bajo fue de 0.95804 del modelo EF0 10% Adam en el

conjunto de pruebas, la desviación estándar es de 0.008121164 y la varianza es de 6.28127E-05. Para la Tabla 17 medida en AUC el valor más alto fue de 0.99915 del modelo EF3 20% Adam en el conjunto de pruebas y el valor más bajo fue de 0.97024 del modelo EF0 10% Adam igualmente en pruebas, la desviación estándar es de 0.006343245 y la varianza es de 3.83207E-05. En la Tabla 18 medida en MSE donde el valor más pequeño es el mejor resultado el cual fue de 0.006833 del modelo EF0 20% RMSProp en entrenamiento, y el valor con peor resultado fue de 0.03862 del modelo EF0 10% Adam en pruebas, donde la desviación estándar es de 0.007605782 y la varianza de 5.50933E-05. Finalmente, en la Tabla 19 podemos observar que los modelos RMSProp tardan más en ser entrenados, en promedio 2.65053126% más tiempo. Se puede observar que existe una ganancia en tiempo con el optimizador Adam contra la perdida de exactitud frente a RMSProp, por lo que en equipos con poco poder de cómputo es conveniente la utilización de Adam teniendo presente la perdida en exactitud.

Para el hiperparámetro de número de épocas se utilizó los valores de 7,10,15 y se tienen los siguientes resultados. En la Tabla 20 medida en exactitud, el valor más alto fue de 0.988267 del modelo EF1 20% 10 épocas en pruebas y el valor más bajo fue de 0.93276 del modelo EF0 10% 7 épocas en pruebas, la desviación estándar es de 0.010838392 y la varianza de 0.000114208. Para la Tabla 21 medida en AUC el valor más alto fue de 0.9997 del modelo EF1 20% 10 épocas en pruebas y el valor más bajo fue de 0.97024 del modelo EF0 10% 15 épocas en pruebas, con una desviación estándar de 0.006470727 con una varianza de 4.07072E-05. En la Tabla 22 medida en MSE el mejor resultado fue de 0.007667 del modelo EF1 20% 10 épocas en pruebas y el peor valor fue de 0.2138 del modelo EF0 10% 10 épocas en pruebas, con una desviación estándar de 0.034290396 y una varianza de 0.001143169. Finalmente, la Tabla 23 medida en segundos podemos observar que al aumentar el número de épocas aumenta el tiempo de entrenamiento, de manera similar que al aumentar la cantidad de imágenes con las que entrenar. Teniendo en cuanta la perdida de exactitud frente a la ganancia en tiempo, nuevamente es conveniente la utilización de 7 épocas en equipos con potencia de cómputo reducida.

Para las diferentes familias de EfficientNet desde B0 hasta B3 se tienen los siguientes resultados. En la Tabla 24 medida en exactitud el valor más alto fue de 0.9977 del modelo EF0 100% en pruebas y el valor más bajo fue de 0.95804 de EF0 10% en pruebas, la desviación estándar fue de 0.007623632 y la varianza de 5.61824E-05. En la Tabla 25 medida en AUC el valor más alto fue de 1 del modelo EF1 100% en pruebas y el valor más bajo de 0.97024 del modelo EF0 10% en pruebas, la desviación estándar fue de 0.005423405 y una varianza de 2.84329E-05. Para la Tabla 26 medida en MSE el mejor valor fue de 0.0018 para el modelo EF0 100% en pruebas y el peor valor fue de 0.0628 del modelo EF2 100% en pruebas, la desviación estándar fue de 0.011698213 y con una varianza de 0.000132287. Finalmente, en la Tabla 27 medida en tiempo muestra el crecimiento al aumentar la familia de EfficientNet y el número de imágenes para entrenar a los modelos. Con esto se sugiere elegir la familia de EfficientNet que, con relación al tiempo empleado en el entrenamiento, logre compensar la ganancia en exactitud.

En la utilización del AP se muestra lo siguiente. Para la Tabla 28 medida en exactitud el valor más alto fue de 0.9977 para el modelo EF0 100% en pruebas y el valor más bajo fue de 0.970767 del modelo EF0 10% + 10 con AP en pruebas, con una desviación estándar de 0.0055273 y una varianza de 2.986E-05. En la Tabla 29 medida en AUC el valor más alto fue de 1 para el modelo EF1 100% en pruebas y el valor más bajo fue de 0.994567 en el modelo EF0 10% + 10% con AP en pruebas, con una desviación estándar de 0.0012132 y una varianza de 1.438E-06. Para la Tabla 30 medida en MSE el mejor valor fue de 0.0018 para el modelo EF0 100% en pruebas y el peor valor fue de 0.0628 para el modelo EF2 100% en pruebas. Finalmente, en la Tabla 31 medida en segundos, los modelos con AP tardaron menos tiempo en entrenarse respecto a los modelos base, siendo en promedio 45.75561% más rápidos. Nuevamente se sugiere la utilización de AP en equipos con poca potencia de cómputo dado la mejora en tiempo durante el entrenamiento.

Utilizando el AUC promedio de los experimentos tenemos que nuestro modelo alcanza un valor de 0.996837011, con el cual se postula la viabilidad de estos hiperparámetros y los elementos de nuestro modelo como una alternativa para la problemática.

En la Tabla 32 colocamos todos los trabajos revisados en el capítulo 2 e incluimos nuestro modelo, pudiendo así compararnos con trabajos que utilizaron el mismo conjunto de datos UADFV. En dicha tabla podemos observar que el valor más alto de AUC para UADFV es de 1, el cual lo obtuvo Tolosana con Openface2 como extractor de rostros y utilizando una combinación de redes Xception y Capsule Network. Seguido de nuestro modelo con un AUC de 0. 996837011 , el cual como mencionamos fue el promedio, pues en un modelo se alcanzo el AUC de 1. Superando al modelo de Li el cual obtuvo un AUC de 0.977 utilizando Dlib como extractor de rostros y una red Res101.

Por otra parte dado que nuestro modelo mantiene el mismo rango de valores de AUC que los modelos que no utilizan el mismo conjunto de datos, podemos ocupar dicho valor como referencia sin compararnos con estos modelos.

Para trabajos futuros será necesario realizar pruebas con bases de datos de segunda generación para poder compararnos con mas modelos, con la limitante de la gran potencia de computo que se requiere para poder procesar dichas bases.

Tabla 32 Comparación del estado del arte con nuestro modelo.

Autor	Fotogramas	Extractor de Rostros	Extractor de Características	DB	Medida de Desempeño AUC
Güera- Temporal Features (2018)	80 fotogramas por paquete	-	CNN + LSTM	Propia	0.971
Rossler- Deep Learning Feature (2019)	Todos los Fotogramas	RGB tracking	Xception Preentrenada	FF ++	0.99 | 0.97
Sabir- Temporal Features (2019)	5 fotogramas por segundo	RGB tracking + Dlib-ml + Alineamiento	DenseNet + RNN	FF ++	0.969
Dolhansky – Deep Learning Features (2019)	1 fotograma por segundo	RGB tracking	Xception Preentrenada	DFDC	0.93 Precisión
Dolhansky – Deep Learning Features(2019)	1 fotograma por segundo	-	Xception Preentrenada	DFDC	0.784 Precisión
Li-Face Warping Features (2019)	Todos los Fotogramas	Dlib	Res101	UADFV | DFDC| Celef - DF	0.977 |0.755 | 0.646
Tolosana- Facial Regions Features (2020)	Todos los fotogramas	OpenFace2+ Alineamiento	Xception Preentrenada y Capsule Network (VGG19)	UADFV | DFDC | Celef - DF	**1** | 0.91 |0.836
A. Davletshin - NTech-(2020)	50 fotogramas por paquete	DSFD+ Aumentos	EfficientNet B7 Preentrenada con Noisy student	DFDC	0.43452 Log Loss
H. Zhao -WM- (2020)	Todos los Fotogramas	RetinaFace + Alineamiento + Aumentos	Xception Preentrenada y EfficientNet B3 + WSDAN y Xception Preentrenada + WSDAN	DFDC	0.42842 Log Loss
S. Seferbekov Selimsef-(2020)	Todos los fotogramas	MTCNN + Aumentos	EfficientNet B7 Preentrenada con Noisy student	DFDC	0.42798 Log Loss
Carrasco (2022)	Todos los fotogramas	Openface2	EfficientNet	UADFV	0.996

Capítulo 8

CONCLUSIONES Y TRABAJOS FUTUROS

Dando una conclusión a cada uno de los objetivos específicos tenemos lo siguiente:

Con lo visto en el capítulo 3 relacionado a los extractores de rostros y la Tabla 1 se seleccionó Openface2 con base a los resultados de Baltrusaitis et al. [12]. Además, con el capítulo 6 de optimizadores y los resultados del capítulo 7 podemos concluir que utilizar dicho extractor de rostros resulta conveniente para la tarea.

Con la revisión del capítulo 4 relacionado a los extractores de características y la Tabla 1 se seleccionó EfficientNet en base a los resultados de Tang et al.[14], además con los resultados del capítulo 7 podemos concluir que la utilización de EfficientNet logra un desempeño comparable con el estado del arte. Además, podemos utilizar la última capa de dicha red para realizar la clasificación como todos los autores de la Tabla 1.

Con la revisión de los hiperparámetros vistos en el capítulo 6 y tras los experimentos realizados en el capítulo 7 podemos concluir que la utilización de Adam y RMSProp, en conjunto con una cantidad de épocas de entre 7 y 15 y la familia de EfficientNet entre B0 a B3 da resultados comparables con el estado del arte. **Tomando en cuenta que utilizar uno u otro afectará en tiempo y que dado que es un proceso heurístico no se puede asegurar los mismos resultados**. Tomando en cuenta lo anterior, la elección de Adam con 7 épocas y una elección de familia de EfficientNet con modelos AP lograran darnos una mejora en tiempo de entrenamiento manteniendo un desempeño comparable a los resultados obtenidos en el capítulo 7.

Con lo visto anteriormente se utilizaron los modelos de EF1 100% para una aplicación mostrada en el Anexo B. Para trabajos futuros se probará esta aplicación y se explorará la opción de convertirla en una aplicación móvil y web para su utilización en cualquier equipo sin importar los requerimientos de este.

Además, se explorarán alternativas a AP para poder trabajar con tensores de mayor tamaño . Con esto se buscará poder trabajar con bases de datos de segunda generación, las cuales no se encuentran balanceadas y tienen peores valores de AUC según datos de Tolosana et al. [4].

REFERENCIAS

[1] D. Guera y E. Delp, «Deepfake Video Detection Using Recurrent Neural Networks,» *International Conference on Advanced Video and Signal Based Surveillance,* 2018.

[2] B. Dolhansky, R. Howes, B. Pflaum, N. Baram y C. Ferrer, «The Deepfake Detection Challenge (DFDC) Preview Dataset,» *arXiv preprint arXiv:1910.08854,* 2019.

[3] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, I. Masi y P. Natarajan, «Recurrent Convolutional Strategies for Face Manipulation Detection in Videos,» *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops,* 2019.

[4] R. Tolosana, R. Vera-Rodriguez, J. Fierrez, A. Morales y J. Ortega-Garcia, «DeepFakes and Beyond: A Survey of Face Manipulation and Fake Detection,» *Information Fusion,* vol. 64, pp. 131-148, 2020.

[5] J. Thies, M. Zollhofer, M. Stamminger, C. Theobalt y M. Nießner, «Face2Face: Real-time Face Capture and Reenactment of RGB Videos,» *Proc. Conferencia IEEE / CVF sobre visión por computadora y reconocimiento de patrones,* 2016.

[6] R. Cellan-Jones, «BBC,» 07 Octubre 2019. [En línea]. Available: https://www.bbc.com/news/technology-49961089. [Último acceso: 10 Septiembre 2022].

[7] Y. Li y S. Lyu, «Exposing DeepFake Videos By Detecting Face Warping Artifacts,» *Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops,* 2019.

[8] A. Rossler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies y M. Nießner, «FaceForensics++: Learning to Detect Manipulated Facial Images,» *in Proc. IEEE/CVF International Conference on Computer Vision,* 2019.

[9] Y. Li, X. Yang, P. Sun, H. Qi y S. Lyu, «Celeb-DF: A Large-Scale Challenging Dataset for DeepFake Forensics,» *IEEE/CVF Conference on Computer Vision and Pattern Recognition,* pp. 3207-3216, 2020.

[10] B. Dolhansky, J. Bitton, B. Pflaum, J. Lu, R. Howes, M. Wang y C. Canton-Ferrer, «The DeepFake Detection Challenge (DFDC) Dataset,» *arXiv:2006.07397v4,* 2020.

[11] F. Chollet, «Xception: Deep Learning with Depthwise Separable Convolutions,» *2017 IEEE Conference on Computer Vision and Pattern Recognition,* pp. 1800-1807, 2017.

[12] T. Baltrusaitis, A. Zadeh, Y. C. Lim y L. -P. Morency, «OpenFace 2.0: Facial Behavior Analysis Toolkit,» *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2018),* pp. 59-66,2018, 2018.

[13] A. Davletshin, «Github,» 8 Junio 2020. [En línea]. Available: https://github.com/NTech-Lab/deepfake-detection-challenge. [Último acceso: 10 Agosto 2022].

[14] M. Tan y Q. V. Le, «EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks,» *36th International Conference on Machine Learning,* pp. 1-11, 2019.

[15] A. Painsky y G. W. Wornell, «Bregman Divergence Bounds and Universality Properties of the Logarithmic Loss,» *IEEE Transactions on Information Theory,* vol. 66, nº 3, pp. 1658-1673, 2020.

[16] H. Zhao, H. Cui y W. Zhou, «Github,» 13 Enero 2020. [En línea]. [Último acceso: 10 Agosto 2022].

[17] S. Seferbekov, «GitHub,» 08 Noviembre 2021. [En línea]. Available: https://github.com/selimsef/dfdc_deepfake_challenge. [Último acceso: 10 Agosto 2022].

[18] K. Zhang, Z. Zhang, Z. Li y Y. Qiao, «Joint face detection and alignment using multi-task cascaded convolutional networks,» *IEEE Signal Processing Letters vol. 23, no. 10,* pp. 1499-1503, 2016.

[19] D. E. King, «Dlib-ml: A Machine Learning Toolkit,» *Journal of Machine Learning,* pp. 1755-1758, 2009.

[20] P. Viola y M. Jhones, «Rapid Object Detection using a Boosted Cascade of Simple Features,» *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001,* pp. 1-1, 2001.

[21] A. Zadeh y T. Baltrusaitis, «Convolutional Experts Network for Facial Landmark Detection,» *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW),* pp. 2051-2059, 2017.

[22] k. Kim, T. Baltrusaitis, A. Zadeh, L. Morency y G. Medionni, «Holistically Constrained Local Model: Going Beyond Frontal Poses,» *BMVC,* 2016.

[23] T. Baltrusaitis, P. Robinson y L.-P. Morency, «3D Constrained Local Model for rigid and non-rigid facial tracking,» *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* 2012.

[24] X. Zhang, Y. Sugano, M. Fritz y A. Bulling, «Appearance-Based Gaze Estimation in the Wild,» *Conference: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2015.

[25] S. M. Mavadati, M. H. Mahoor, K. Bartlett, P. Trinh y J. F. Cohn, «DISFA: A Spontaneous Facial Action Intensity,» *IEEE Transactions on Affective Computing,* vol. 4, nº 2, pp. 151-160, 2013.

[26] T. Baltrusaitis , «Github,» 4 Julio 2019. [En línea]. Available: https://github.com/TadasBaltrusaitis/OpenFace/wiki/. [Último acceso: 22 Septiembre 2022].

[27] T. Yu y H. Zhu, «Hyper-Parameter Optimization: A Review of Algorithms,» *arXiv,* 2020.

[28] N. Andrew, «Improving deep neural networks: Hyperparameter tuning, regularization and,» *Deeplearning. ai on Coursera,* 2017.

[29] Y. Bengio, «Practical recommendations for gradient-based training of deep architectures,» *Montavon, G., Orr, G.B., Müller, KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science,* vol. 7700, p. 437–478, 2012.

[30] L. Velasco, «velascoluis,» velascoluis, 27 Abril 2020. [En línea]. Available: https://velascoluis.medium.com/optimizadores-en-redes-neuronales-profundas-un-enfoque-pr%C3%A1ctico-819b39a3eb5. [Último acceso: 1 Noviembre 2021].

[31] A. Karpathy y a. et, «Cs231n convolutional neural networks for visual recognition,» *Neural networks,* p. 1:1, 2016.

[32] Z. Matthew, «Adadelta: an adaptive learning rate method,» *arXiv,* 2012..

[33] S. Jadon, «A survey of loss functions for semantic segmentation,» *IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB),* pp. 1-7, 2020.

[34] «Peltarion,» Knowledge Center, [En línea]. Available: https://peltarion.com/knowledge-center/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy. [Último acceso: 20 Septiembre 2022].

[35] M. Hossin y M. Sulaiman, «A Review on Evaluation Metrics for Data Classification Evaluations,» *International Journal of Data Mining & Knowledge Management Process,* vol. 5, pp. 01-11, 2015.

[36] N. Lavesson y P. Davidsson, «Generic Methods for Multi-Criteria Evaluation,» *Proc. of the Siam Int. Conference on Data Mining, Atlanta, Georgia, USA: SIAM Press,* nº 541-546, 2008.

[37] J. Huang y C. X. Ling, «Constructing new and better evaluation measures for machine learning,» *R. Sangal, H. Mehta and R. K. Bagga (Eds.) Proc. of the 20th International Joint Conference on Artificial Intelligence (IJCAI 2007), San Francisco, CA, USA A: Morgan Kaufmann Publishers Inc,* pp. 859-864, 2007.

[38] A. Rakotomamonyj, «Optimizing area under ROC with SVMs,» *J. Hernandez-Orallo, C. Ferri, N. Lachiche and P. A. Flach (Eds.) 1st Int. Workshop on ROC Analysis in Artificial Intelligence (ROCAI 2004), Valencia, Spain,* pp. 71-80, 2004.

[39] D. Hand y R. Till, «A simple generalization of the area under the ROC curve to multiple class classification problems,» *Machine Learning,* nº 45, pp. 171-186, 2001.

[40] J. Huang y C. Ling, «Using AUC and accuracy in evaluating learning algorithms,» *IEEE Transactions on Knowledge Data Engineering,* nº 17, pp. 299-310, 2005.

[41] D. Montana y L. Davis, «Training Feedforward Neural Networks Using Genetic Algorithms,» *IJCAI'89: Proceedings of the 11th international joint conference on Artificial intelligence ,* vol. 1, p. 762–767, 1989.

[42] L. Dabbish, C. Stuart, J. Tsay y J. Herbsleb, «Social coding in GitHub: transparency and collaboration in an open software repository,» *CSCW '12: Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work,* p. 1277–1286, 2012.

[43] Q. Xie, M. Luong, E. Hovy y Q. Le, «Self-training with Noisy Student improves ImageNet classification,» *arXiv:1911.04252,* 2020.

[44] T. Ridnik, E. Ben-Baruch, A. Noy y L. Zelnik-Manor, «ImageNet-21K Pretraining for the Masses,» *arXiv:2104.10972,* 2021.

[45] S. Targ, D. Almeida y K. Lyman, «Resnet in Resnet: Generalizing Residual Architectures,» *arXiv:1603.08029,* 2016.

Anexo A INSTALACIÓN DE SOFTWARE

Para la instalación del software se creó la siguiente guía, se mostrará el método de instalación sugerido para cada elemento ocupado durante los experimentos. Se denominó ambiente de trabajo al conjunto de este software.

A 1 SISTEMA OPERATIVO Y DEPENDENCIAS.

El sistema operativo utilizado en el ambiente de trabajo fue Ubuntu 20.04[2], el cual se instaló en el disco duro de estado sólido de 240 Gb, creando las particiones por defecto con el punto de montura en "/", la partición swap de 8gb y el resto de los valores por defecto. Una vez terminada la instalación del sistema operativo es necesario actualizar algunos repositorios por lo que se sugiere ejecutar los siguientes comandos.

sudo apt-get update
sudo apt-get upgrade

A-1.1 GIT

Es indispensable instalar Git para poder clonar los repositorios de Openface2, por lo cual se deberá ejecutar el siguiente comando:

sudo apt-get install git

A-1.2 PYTHON

Ubuntu 20.04 cuenta con la instalación de Python, pero para verificar la versión será necesario ejecutar la siguiente instrucción.

python3 -V

[2] Ubuntu 20.04 LTS Focal fosa puede descargarse desde la web https://ubuntu.com/download/desktop

En caso de que la consola reporte que tenemos instalado Python 3.8 o superior no será necesario continuar con esta seccion. Para instalar Python 3.8 es necesario ejecutar el siguiente comando

sudo apt install -y python3-pip
sudo apt install python3-pip

A 3 DRIVER DE VIDEO NVIDEA

Ubuntu tiene la opción de instalar los drivers de video Nvidia mediante la siguiente línea de comandos, la cual desplegará una lista de drivers estables.

ubuntu-drivers devices

Del listado es necesario buscar los que contengan el número “470” los cuales harán referencia a los modelos RTX 3070 en el mes de octubre de 2022, por lo cual será necesario ejecutar el siguiente comando:

sudo apt install nvidia-driver-470

Es necesario realizar un reinicio del sistema operativo para que los cambios se vean reflejados, tras lo cual podremos comprobar la instalación de los drivers mediante el siguiente comando:

nvidia-smi

Si tras aplicar dicho comando se despliega una lista de información de la tarjeta de video querrá decir que se habrá instalado correctamente, estos drivers cuentan con CUDA, que será indispensable para el entrenamiento de EfficientNet y la ejecución de TensorFlow más adelante.

A-4 OPENFACE2

Para instalar Openface2[3] en el equipo se necesitará ejecutar las siguientes líneas de comando para preparar el entorno de desarrollo.

[3] Openface2 se puede instalar mediante la siguiente guía
https://github.com/TadasBaltrusaitis/OpenFace/wiki/Unix-Installation

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install g++-8
sudo apt-get install cmake
sudo apt-get install libopenblas-dev
wget https://github.com/opencv/opencv/archive/4.1.0.zip
sudo unzip 4.1.0.zip
cd opencv-4.1.0
sudo mkdir build
cd build
sudo cmake -D CMAKE_BUILD_TYPE=RELEASE -D
CMAKE_INSTALL_PREFIX=/usr/local -D BUILD_TIFF=ON -D WITH_TBB=ON ..
sudo make -j2
sudo make install
cd ../..
wget http://dlib.net/files/dlib-19.13.tar.bz2;
tar xf dlib-19.13.tar.bz2;
cd dlib-19.13;
mkdir build;
cd build;
cmake ..;
cmake --build. --config Release;
sudo make install;
sudo ldconfig;
cd ../..;
sudo apt-get install libboost-all-dev
```

Una vez se ejecuten estas líneas de comando se procederá a instalar OpenFace2 mediante las siguientes instrucciones:

```
git clone https://github.com/TadasBaltrusaitis/OpenFace.git
cd OpenFace
mkdir build
cd build
```

Con lo cual tendremos instalado Openface2 en el ambiente de desarrollo, pero faltará descargar los modelos con los comandos:

```
chmod +x download_models.sh
./download_models.sh
sudo apt install python3-opencv
echo -e 'export PATH="$PATH:/home/dickaprio/.local/bin"' >> $HOME/.bashrc
```

A-5 DOCKER

Dado que utilizaremos algunos contenedores de TensorFlow y Jupyter será necesario instalar Docker[4]. Para corroborar que tengamos el listado más reciente de paquetes ejecutaremos un “update” seguido de los siguientes comandos.

```
sudo apt-get update
sudo apt-get install \
 apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo \
  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu \
  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io
sudo docker run hello-world
```

En caso de mostrar un mensaje como el de la Figura 19 se habrá completado esta sección exitosamente

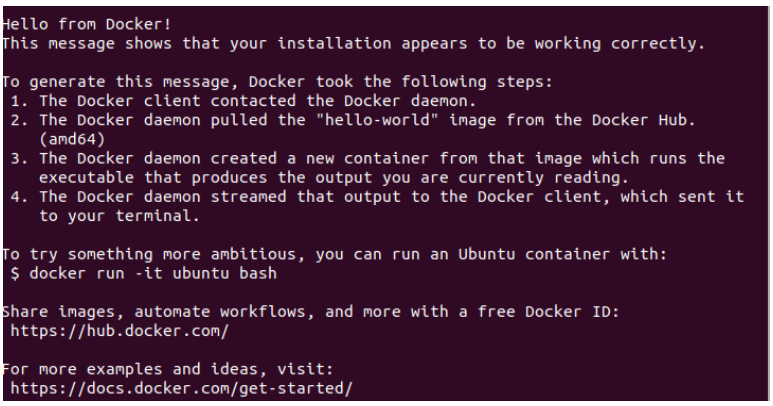


Figura 19 Mensaje Exitoso de Instalación Docker.

[4] Docker se puede instalar mediante la siguiente guía https://docs.docker.com/engine/install/ubuntu/

A-5.1 NVIDEA DOCKER

Además de instalar los drivers de Nvidia será necesario habilitarlos en los contenedores de Docker, por lo cual se requiere de la instalación del contenedor Nvidia Docker [5] mediante los siguientes comandos.

```
curl https://get.docker.com | sh \
  && sudo systemctl --now enable Docker
distribution=$(. /etc/os-release;echo $ID$VERSION_ID) \
   && curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add - \
   && curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-docker.list |
sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update
sudo apt-get install -y nvidia-docker2
sudo systemctl restart docker
sudo docker run --rm --gpus all nvidia/cuda:11.0-base nvidia-smi
```

A-6 TENSORFLOW

Para instalar EfficientNet es necesario descargar TensorFlow[6], cabe señalar que no utilizaremos TensorFlow directamente en nuestro equipo, por el contrario, utilizaremos Docker para gestionar la instalación y control de versiones de TensorFlow, y facilitar el despliegue del aplicativo, añadiendo Jupyter Notebook para facilitar más la portabilidad del ambiente. Esta instalación se realizará mediante la siguiente línea de comando.

```
docker pull tensorflow/tensorflow:latest-gpu-jupyter
```

A-7 COMANDOS BÁSICOS

Para poder trabajar con el ambiente de desarrollo se darán una serie de comandos para interactuar con los elementos instalados.

[5] Para instalar Nvidea Docker se puede seguir la siguiente guía https://docs.nvidia.com/datacenter/cloud-native/container-toolkit/install-guide.html#install-guide

[6] TensorFlow se instalará mediante la siguiente guía https://www.tensorflow.org/install/docker?hl=es-419

Para ejecutar pruebas con Openface2 se necesitará ejecutar el siguiente comando, pasando la dirección de la carpeta donde se encuentra nuestra base de datos de videos o imágenes.

./bin/ FaceLandmarkVidMulti -f "../directorio_del_dataset" -f

Para realizar pruebas en EfficientNet será necesario crear un contenedor Docker, pasando las carpetas donde se encuentre la base de datos como parámetro, teniendo de esta manera la carpeta en cuestión en la maquina Docker para trabajar con ella. Para crear un contenedor de TensorFlow con Jupyter incluido se deberá ejecutar:

docker run --gpus all -it -p 8888:8888 -v /directorio_del_dataset/:/nombre_como_lo_veremos_en_docker tensorflow/tensorflow:latest-gpu-jupyter

Se puede agregar más carpetas siguiendo con la nomenclatura -v seguido de la carpeta en nuestro ambiente de desarrollo y el nombre que queramos que tenga en Docker.

Para poder borrar datos de Docker se sugiere el siguiente comando:

*rm -rf ~/.local/share/Trash/**

Otros comandos importantes serán el de listar contenedores creados para poder iniciarlos o saber su tamaño

docker ps -a --format "{{.Names}} {{.Size}}"\n\n

Para iniciar un contenedor creado será necesario saber su nombre y ejecutar el comando

docker start -i name_machine

Los cuadernos Jupyter permiten interactuar de manera más fácil con la maquina Docker por lo que se sugiere hacer cualquier instalación de software desde dichos cuadernos. Dado que es necesario instalar algunas dependencias se sugiere crear un cuaderno y ejecutar las siguientes líneas.

```
!pip install pandas
!pip install sklearn
!pip install opencv-python
!apt-get update
!apt-get install ffmpeg libsm6 libxext6  -y
```

Para comprobar que se está utilizando la tarjeta gráfica en lugar del procesador se pueden ocupar las siguientes líneas en el cuaderno Jupyter, cuya salida deberá ser algo similar a la Figura 20.

```
import tensorflow as tf
tf.config.experimental.list_physical_devices()
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

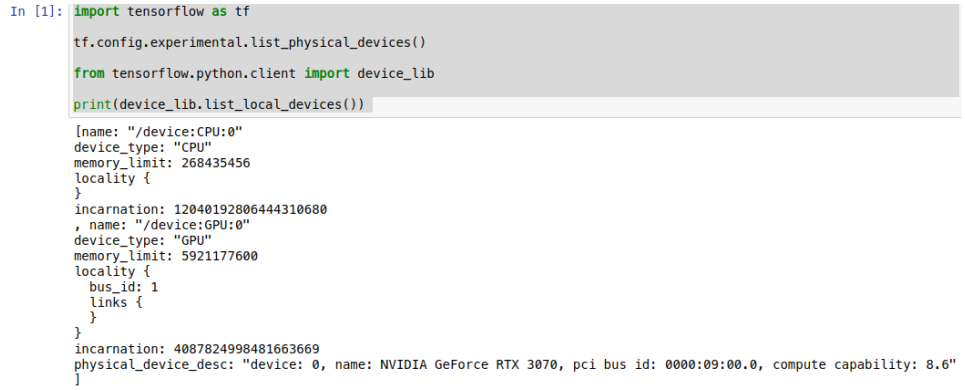


Figura 20 Mensaje Exitoso de uso de GPU en Jupyter Docker.

Alternativamente se pueden utilizar las siguientes líneas con el mismo propósito y cuyo mensaje de salida será el de la Figura 21.

```
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
gpus = tf.config.list_physical_devices('GPU')
if gpus:
  # Restrict TensorFlow to only use the first GPU
  try:
    tf.config.set_visible_devices(gpus[0], 'GPU')
    logical_gpus = tf.config.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
  except RuntimeError as e:
    # Visible devices must be set before GPUs have been initialized
    print(e)
```

In [3]:
```
import tensorflow as tf
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  1
```

In [6]:
```
gpus = tf.config.list_physical_devices('GPU')
if gpus:
  # Restrict TensorFlow to only use the first GPU
  try:
    tf.config.set_visible_devices(gpus[0], 'GPU')
    logical_gpus = tf.config.list_logical_devices('GPU')
    print(len(gpus), "Physical GPUs,", len(logical_gpus), "Logical GPU")
  except RuntimeError as e:
    # Visible devices must be set before GPUs have been initialized
    print(e)
```

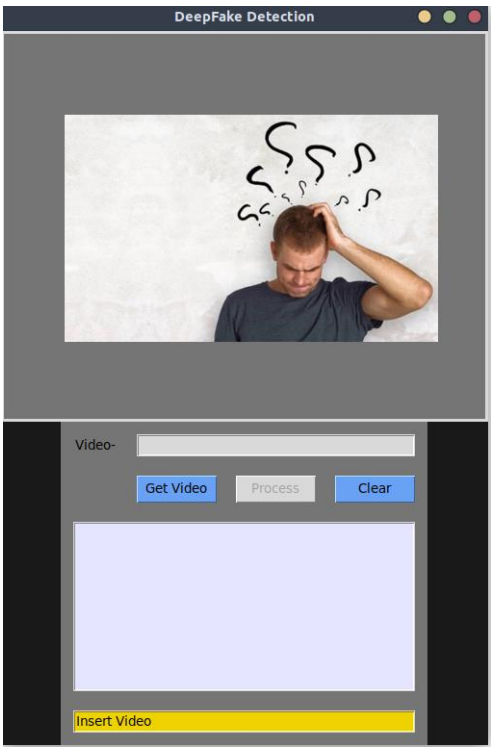
```
1 Physical GPUs, 1 Logical GPU
```

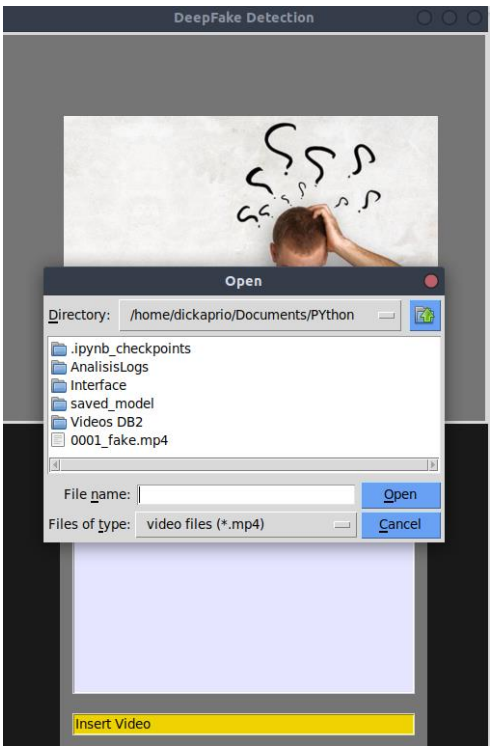
Figura 21 Mensaje exitoso alternativo de uso de GPU en Jupyter Docker

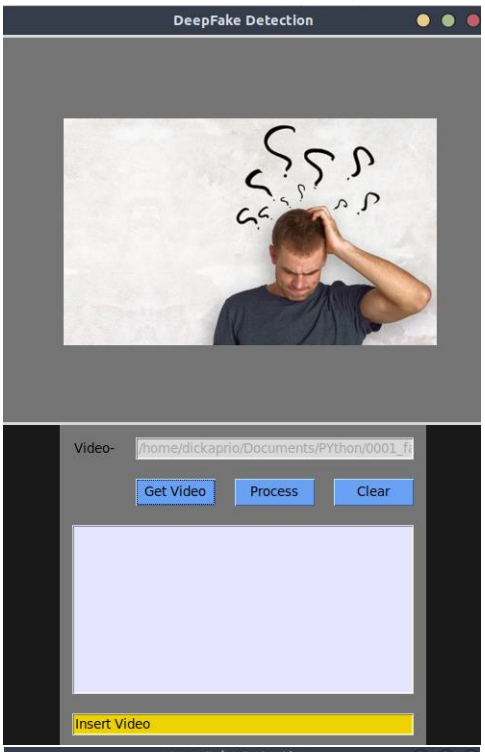
Anexo B APLICACIÓN DE USUARIO

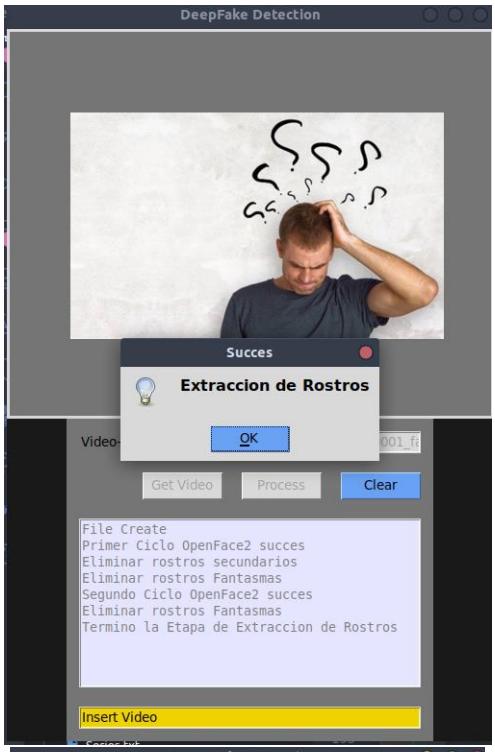
A continuación, se mostrarán algunas capturas de pantalla de una versión preliminar de aplicación para el usuario, donde podrá colocar videos con una persona hablando a la cámara, de frente sin movimientos bruscos y con una misma condición lumínica para todo el video, además de ser videos de corta duración, es decir, videos similares a la base de datos UADFV.

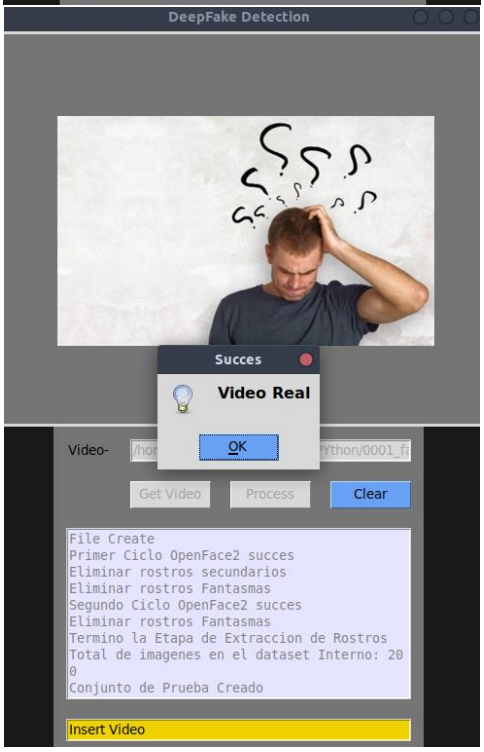
La aplicación creará una copia del video introducido y ejecutará Openface2 para extraer los rostros, utilizando el proceso de limpieza sugerido en el capítulo 6. Posterior a esto cargará el modelo entrenado con EfficientNet EF1 100% y evaluará las imágenes. De encontrar alguna imagen que sea clasificada por nuestro modelo como falsa se informará que dicho video es falso, caso contrario se reportará que el video es real.

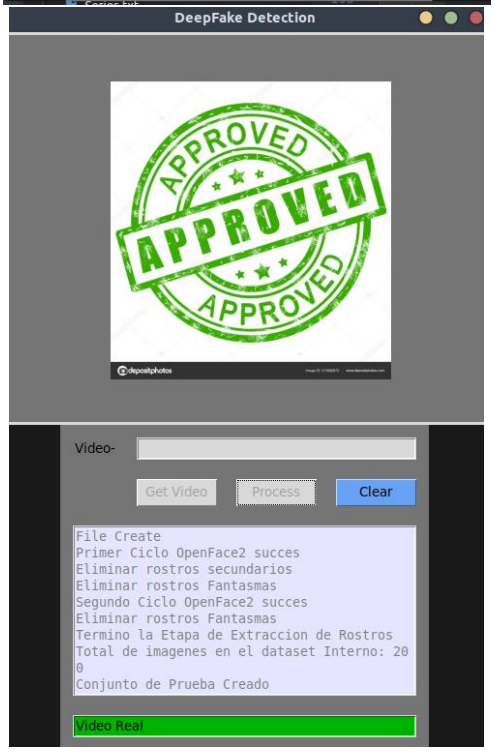





DeepFake Detection
Video-
/home/dickaprio/Documents/PYthon/0001_fa
Get Video
Process
Clear
Insert Video


DeepFake Detection
Succes
Extraccion de Rostros
OK
Video-
Get Video
Process
Clear
File Create
Primer Ciclo OpenFace2 succes
Eliminar rostros secundarios
Eliminar rostros Fantasmas
Segundo Ciclo OpenFace2 succes
Eliminar rostros Fantasmas
Termino la Etapa de Extraccion de Rostros
Insert Video


DeepFake Detection
Succes
Video Real
OK
Video-
Get Video
Process
Clear
File Create
Primer Ciclo OpenFace2 succes
Eliminar rostros secundarios
Eliminar rostros Fantasmas
Segundo Ciclo OpenFace2 succes
Eliminar rostros Fantasmas
Termino la Etapa de Extraccion de Rostros
Total de imagenes en el dataset Interno: 20
0
Conjunto de Prueba Creado
Insert Video


DeepFake Detection
APPROVED
APPROVED
APPROVED
Video-
Get Video
Process
Clear
File Create
Primer Ciclo OpenFace2 succes
Eliminar rostros secundarios
Eliminar rostros Fantasmas
Segundo Ciclo OpenFace2 succes
Eliminar rostros Fantasmas
Termino la Etapa de Extraccion de Rostros
Total de imagenes en el dataset Interno: 20
0
Conjunto de Prueba Creado
Video Real

GLOSARIOS

DeepFake –Técnica de intercambio de identidad para modificar videos.

Feedforward - Las redes neuronales feedforward son las redes neuronales de propósito más general. El punto de entrada es la capa de entrada y consta de varias capas ocultas y una capa de salida. Cada capa tiene una conexión con la capa anterior. [41]

GitHub- GitHub es un portal creado para alojar el código de las aplicaciones de cualquier desarrollador. [42]

Noisy Student - Noisy Student Training es una técnica de aprendizaje semisupervisado, amplía la idea del auto entrenamiento y la destilación con el uso de modelos de estudiantes iguales o mayores. [43]

ImageNet – ImageNet es una base de datos de imágenes organizada según la jerarquía de WordNet. El proyecto ha sido fundamental en el avance de la visión por computadora y la investigación de aprendizaje profundo. [44]

ResNet- La red residual se inspira en el hecho biológico de que algunas neuronas se conectan con neuronas en capas no necesariamente contiguas, saltando capas intermedias. [45]

Yaw-es la rotación con respecto al eje vertical de la cabeza. [12]

Pitch-es la rotación con respecto al eje horizontal de la cabeza que une a los oídos. [12]

Roll-es la rotación con respecto al eje horizontal de la cabeza de la nariz a la nuca. [12]

Momentum- Es una magnitud física derivada de tipo vectorial que describe el movimiento de un cuerpo en cualquier teoría mecánica. [35]

Log Loss- Métrica específica de desempeño, la fórmula de Log Loss es -1 * el logaritmo de la función de probabilidad, para cualquier problema dado, un valor de pérdida logarítmica más bajo significa mejores predicciones. [35]