UNIVERSIDAD ESAN

FACULTAD DE INGENIERÍA

INGENIERÍA DE TECNOLOGÍAS DE INFORMACIÓN Y SISTEMAS

Impacto del preprocesamiento de imágenes en la efectividad de la verificación facial empleando visión computacional

Tesis para obtener el Título de Ingeniero en Tecnologías de Información y Sistemas que presenta:

Bruno Tafur Coronel Zegarra

Asesor: Joseph Ballón Álvarez

Lima, agosto de 2020

Esta tesis denominada:

IMPACTO DEL PREPROCESAMIENTO DE IMÁGENES EN LA EFECTIVIDAD DE LA VERIFICACIÓN FACIAL EMPLEANDO VISIÓN COMPUTACIONAL

ha sido aprobada.

…….

Javier Fernando Del Carpio Gallegos (Jurado Presidente)

…….

Marks Arturo Calderón Niquin (Jurado)

…….

Wilfredo Mamani Ticona (Jurado)

Universidad ESAN

2020

IMPACTO DEL PREPROCESAMIENTO DE IMÁGENES EN LA EFECTIVIDAD DE LA VERIFICACIÓN FACIAL EMPLEANDO VISIÓN COMPUTACIONAL

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El objetivo de la investigación es evaluar el impacto del preprocesamiento de imágenes en la efectividad de la verificación facial. Un sistema de verificación facial realiza un proceso basado en detección de rostro, preprocesamiento de imagen, extracción de características y verificación facial. Los sistemas de verificación enfrentan desafíos relacionados a la iluminación, expresión o pose. Se decidió evaluar el impacto del preprocesamiento buscando aliviar estas dificultades.

Se tomó en cuenta la evaluación del preprocesamiento en términos de alineamiento, suavizamiento, agudizamiento y ecualización. Se realizaron pruebas de efectividad en tres fuentes de información: *Labeled Faces in the Wild (LFW)*, *YouTube Faces DB (YTF)* y una base de datos obtenida dentro del contexto local. Asimismo, se evaluó en tres algoritmos de extracción de características basados en redes neuronales convolucionales: *OpenFace*, *VGGFace2* y *Light CNN*. Adicionalmente, se analizó con dos métodos de detección facial: basados en descriptores HOG y Haar. Se utilizó la metodología CRISP-DM para la analítica de datos y la metodología cascada para el desarrollo de software de dos prototipos.

Los resultados de las pruebas alcanzaron una efectividad de hasta 98.18% en *LFW*, 85.72% en *YTF* y 93.62% en la base de datos del contexto local. Se demostró la relación entre los métodos de preprocesamiento y la efectividad del sistema corroborando que métodos como el alineamiento son muy efectivos. Se demostró que este impacto puede ser positivo o negativo dependiendo de la combinación de factores como la fuente de información, el modelo de verificación facial y el método de detección facial.

Palabras clave: verificación facial, preprocesamiento de imágenes, visión computacional, redes neuronales convolucionales, detección facial

ABSTRACT

The objective of this research is to evaluate the impact of image preprocessing on the effectiveness of facial verification. A facial verification system performs a process based on face detection, image preprocessing, feature extraction and facial verification. Verification systems face challenges in effectiveness related to lighting, expression or pose. The aim is to evaluate the impact of preprocessing, seeking to alleviate these difficulties.

Preprocessing evaluation was considered in terms of alignment, smoothing, sharpening and equalization. Effectiveness tests were performed on three sources of information.: Labeled Faces in the Wild (LFW), YouTube Faces DB (YTF) and a database obtained within the local context. Likewise, tests were performed in three feature extraction algorithms based on convolutional neural networks: OpenFace, VGGFace2 and Light CNN. Additionally, results were analyzed with two methods of facial detection: based on HOG and Haar characteristics. The CRISP-DM methodology was used for data analysis and the software cascade methodology for the development of two prototypes.

The test results reached an effectiveness of 98.18% in LFW, 85.72% in YTF and 93.62% in the local context database. The relationship between preprocessing methods and system effectiveness was demonstrated by corroborating that methods such as alignment are very effective. It was shown that this impact can be positive or negative depending on the combination of factors such as the source of information, the facial verification model and the facial detection method.

Keywords: face verification, image preprocessing, computer vision, convolutional neural networks, face detection

INTRODUCCIÓN

La investigación busca analizar la relación entre el preprocesamiento de imágenes y la verificación facial. Esta evaluación se realiza mediante la construcción y desarrollo de dos prototipos que permiten evaluar de forma automática los métodos de preprocesamiento y su impacto en la efectividad de dicha verificación.

A nivel global el reconocimiento facial se ha convertido en un método muy importante utilizado para mejorar las capacidades automáticas de video vigilancia y sistemas de seguridad, software de video analítica y miles de aplicaciones del día a día relacionadas a entretenimiento, *retail* y redes sociales (LeCun, Bengio, & Hinton, 2015). Por lo tanto, este tipo de técnicas son de relevancia en la actualidad en diversos ámbitos.

La verificación facial es también un método de seguridad biométrica que se centra en verificar a una persona mediante la comparación de dos imágenes del rostro. La seguridad biométrica busca validar rasgos biométricos de la misma persona como es el caso de la verificación por iris, huella dactilar o verificación facial (Ortiz, Hernández, Jimenez, Mauledeoux, & Avilés, 2018). Estos métodos son cada vez más precisos con el surgimiento de nuevas técnicas, pero su adopción todavía está en desarrollo.

La visión computacional es un campo de estudio centrado en el reconocimiento y detección de objetos a través de algoritmos computacionales. En los últimos años, las redes neuronales convolucionales se han convertido en una de las técnicas más populares para resolver problemas relacionados a visión computacional teniendo casos de uso en clasificación de imágenes, detección de objetos, reconocimiento de rostros entre otros (Wu, He, Sun, & Tan, 2018). Ha sido tan grande el reconocimiento de dicho tipo de redes que tres de los precursores más importantes de estas técnicas - Yoshua Bengio, Geoffrey Hinton y Yann LeCun - han sido ganadores del Premio Turing 2018, al cual se le conoce como el "Premio Nobel de la Computación" (ACM, 2018). El premio se les brindó por los avances conceptuales y de ingeniería que han hecho que las redes neuronales profundas (*deep neural networks*), entre las que se encuentran las redes neuronales convolucionales, sean un componente crítico de la computación. Por lo tanto, debido a estos avances, han surgido distintos algoritmos basados en dichas redes para el reconocimiento y la verificación facial como es el caso de los que se utilizan en la presente investigación: *VGGFace2*, *OpenFace* y *Light CNN* (Wu, He, Sun, & Tan, 2018; Amos, Bartosz, & Satyanaray, 2016; Cao, Shen, Xie, Parkhi, & Zisserman, 2018).

Los principales problemas actualmente en el reconocimiento facial son los mismos de hace más de 50 años denominados PIE: posición, iluminación y expresión (Wu, He, Sun, & Tan,

2018). PIE hace referencia a la posición del rostro, la iluminación de la imagen y la expresión del rostro. Para contrarrestar este tipo de problemas, se busca aplicar diferentes técnicas, por ejemplo, entrenando modelos con cantidades masivas de data.

Por otro lado, para contrarrestar estos problemas se puede aplicar el preprocesamiento de imágenes (Dharavath, Talukdar, & Laskar, 2014; Heusch, Rodriguez, & Marcel, 2006). El preprocesamiento implica hacer cambios en la imagen antes de que sea procesada por el sistema de tal forma que este sea capaz de reconocer mejor lo que se encuentra en dicha imagen.

En la investigación se utilizaron tres fuentes de información para las evaluaciones en contextos diferentes. En primer lugar, se analizaron los resultados de la investigación en dos bases de datos globales y *open source*, en este caso se hicieron las pruebas en la base de *Labeled Faces In the Wild (LFW)* y en *YouTube Faces DB (YTF)*. Adicionalmente, para la tercera base de datos, se recolectaron fotos de documentos de identidad (DNIs) y del rostro de personas de Lima, Perú que utilizaron un prototipo web desarrollado en la investigación.

Asimismo, se hicieron las pruebas en tres modelos de redes neuronales convolucionales diferentes. Estos modelos son *OpenFace* (Amos, Bartosz, & Satyanaray, 2016) basado en *FaceNet* (Schroff, Kalenichenko, & Philbin, 2015), *VGGFace2* (Cao, Shen, Xie, Parkhi, & Zisserman, 2018) y *Light CNN* (Wu, He, Sun, & Tan, 2018). Además, se realizaron las pruebas mediante métodos de detección facial basados en descriptores *HOG* (*Histogram of Oriented Gradients*) y basados en descriptores *Haar*. A partir de estas pruebas se pudo evaluar el impacto del preprocesamiento de imágenes en la verificación facial en diferentes fuentes de información, modelos de verificación y métodos de detección facial.

En el primer capítulo se describe la problemática de investigación, se formulan los problemas, los objetivos principales, las hipótesis y la justificación del estudio.

En el segundo capítulo se observan los principales antecedentes a la investigación centrándose en el preprocesamiento de imágenes y los modelos de verificación facial. Asimismo, se describen las bases teóricas de la investigación. Estas incluyen las variables principales del estudio: el preprocesamiento de imágenes y le efectividad de la verificación facial. Asimismo, se describen los otros elementos del proceso de verificación facial: la detección de rostros y la extracción de características. También se explican las principales metodologías involucradas. Finalmente, se expone el contexto de la investigación tanto a nivel global como local.

La metodología de investigación se desarrolla en el tercer capítulo. En este se especifica el diseño de la investigación. Para esto se hace especial énfasis en el proceso seguido al momento

de desarrollar la investigación. Además, se define el tipo y el enfoque de estudio, la población y muestra y las técnicas más importantes a utilizar para el análisis de datos.

En el capítulo cuatro se describe el desarrollo de la solución mediante un enfoque práctico. Asimismo, se realiza el análisis de los resultados obtenidos de la investigación centrándose en las respuestas a las hipótesis planteadas.

Adicionalmente, en el capítulo cinco se discuten los resultados de la investigación, se resaltan las principales conclusiones a los problemas planteados y se realizan recomendaciones acerca de lo desarrollado en el estudio tomando en cuenta futuras investigaciones.

Finalmente, se incluyen anexos que complementarán la información con detalle representativo de lo descrito durante la investigación.

CAPÍTULO I: PLANTEAMIENTO DEL PROBLEMA

Este capítulo describe la problemática que existe actualmente y su relevancia, así como los objetivos y justificación del estudio.

1.1. Descripción de la Realidad Problemática

Esta investigación evalúa la relación entre el preprocesamiento de imágenes y la efectividad de la verificación facial. Asimismo, se busca analizar dichos resultados en distintos modelos, fuentes de información y métodos de detección facial, de tal forma que se generen resultados aplicables en otros contextos.

La verificación facial es considerada como un tipo de seguridad biométrica y se centra en identificar al usuario por características propias de este como las huellas dactilares, verificación facial o el reconocimiento de iris (Ortiz, Hernández, Jimenez, Mauledeoux, & Avilés, 2018). Es decir, se basan en datos que están ligados al usuario en sí, algo que no sucede con las contraseñas tradicionales. Por lo tanto, las características pertenecen únicamente a cada individuo.

De acuerdo con Dharavath, Talukdar y Laskar (2014), en los últimos años, el reconocimiento facial ha ganado mayor importancia como método de seguridad biométrica para la autenticación y muchas industrias alrededor del mundo están ahora tratando de implementar sistemas de autenticación que utilicen dicha tecnología. Además, en el mundo el reconocimiento facial se ha convertido en una herramienta significativa facilitando la creación de sistemas de seguridad, video analítica y miles de aplicaciones que se usan cotidianamente relacionadas a entretenimiento, compras y etiquetado automático en fotografías (Masi, Wu, Hassner, & Natarajan, 2018). Asimismo, varios gobiernos del mundo están ganando interés en

la video vigilancia con reconocimiento facial en lugares públicos como el caso de China donde el gobierno, bancos, aeropuertos, hoteles y hasta baños públicos están verificando la identidad de las personas analizando sus rostros (Denyer, 2018). En el 2018 se estimaba que China tenía 200 millones de cámaras de video vigilancia (Mozur, 2018). Es decir, las implementaciones de dichas tecnologías han ido creciendo considerablemente en los últimos años.

Figura 1. Video con software de reconocimiento facial utilizado en Beijing
Fuente: Mozur (2018). *Inside China's Dystopian Dreams: A.I., Shame and Lots of Cameras*

Las técnicas convencionales estaban limitadas en su habilidad de procesar data natural en su estado crudo. Por décadas, construir un sistema de reconocimiento de patrones o sistema de aprendizaje automático requería considerable conocimiento experto del dominio para diseñar un extractor de características que transformase la data cruda - como es el caso de los pixeles en una imagen - en una representación adecuada como un vector de características desde el cual se pueda detectar o clasificar patrones (LeCun, Bengio, & Hinton, Deep Learning, 2015). En el caso de la visión computacional, para la clasificación y reconocimiento de rostros, se realiza dicha extracción de características a partir de los pixeles en la imagen, esto para obtener una representación numérica que pueda luego ser usada para reconocer el rostro.

Estas dificultades en extraer características cambiaron con las nuevas técnicas que surgieron basadas en redes neuronales. Una de estas son las redes neuronales convolucionales, las cuales está diseñadas para procesar data que ingrese en la forma de múltiples matrices y extraer características diferentes (Schroff, Kalenichenko, & Philbin, 2015; Parkhi, Vedaldi, & Zisserman, 2015; Taigman, Yang, Ranzato, & Wolf, 2014). Esto es útil para modelos de visión

computacional dado que las imágenes suelen componerse por matrices 2D que representan los pixeles en cada canal de color.

En la actualidad, las redes neuronales convolucionales tienen gran relevancia a nivel global. Este tipo de técnicas han ganado popularidad en la visión computacional para aplicaciones como clasificación de imágenes, detección de objetos y reconocimiento facial (Wu, He, Sun, & Tan, 2018; Russakovsky, et al., 2015). Además, las redes neuronales convolucionales en sistemas de visión computacional han incentivado que la mayoría de las principales empresas de tecnología incluyendo Google, Facebook, Microsoft, IBM, Yahoo!, Twitter y Adobe así como varias *start-ups* inicien investigaciones y desarrollen proyectos para desplegar productos basados en este tipo de redes relacionados al entendimiento de imágenes (LeCun, Bengio, & Hinton, Deep Learning, 2015). Por lo tanto, han surgido distintos modelos basados en redes neuronales convolucionales para el reconocimiento facial como es el caso de *FaceNet* (Schroff, Kalenichenko, & Philbin, 2015), *VGGFace* (Parkhi, Vedaldi, & Zisserman, 2015), *DeepFace* (Taigman, Yang, Ranzato, & Wolf, 2014), entre otros.

Adicionalmente, Heusch, Rodriguez y Marcel (2006) destacan que, desde hace varias décadas, el reconocimiento facial ha sido un área de investigación existiendo ahora diferentes sistemas que pueden reconocer personas en ambientes controlados en los cuales las imágenes son adquiridas bajo las mismas condiciones globales.

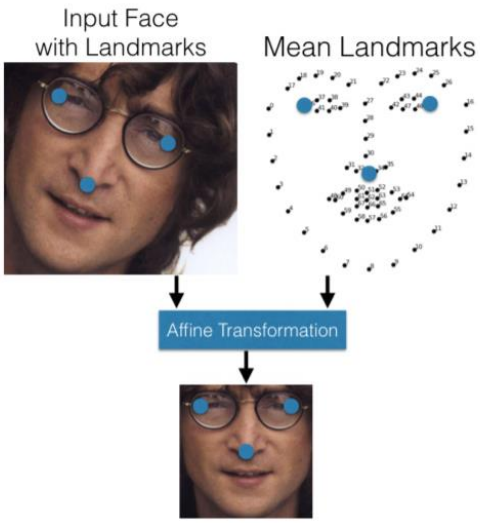


Figura 2. Ejemplo de Alineamiento 2D
Fuente: Amos, Bartosz & Satyanaray (2016). *OpenFace: A general-purpose face recognition library with mobile applications.* (p. 6)

Sin embargo, siguen existiendo dificultades al reconocer a personas en tiempo real y todavía existen retos para reconocer personas con diferente pose, expresión o iluminación (Han, Shan, Chen, & Gao, 2013; Heusch, Rodriguez, & Marcel, 2006). La calidad de la imagen es uno de los principales factores causando dichos problemas. Esto puede suceder principalmente en ambientes no controlados. Dicha calidad de imagen depende de factores como la eficiencia de la cámara, la distancia entre la cámara y la persona, la iluminación y la dirección en la que está mirando la persona (Dharavath, Talukdar, & Laskar, 2014). Los métodos de preprocesamiento de imágenes sirven para contrarrestar dichas irregularidades.

El preprocesamiento de imágenes es análogo a la normalización matemática de un conjunto de datos (Krig, 2014) estandarizando las características de las imágenes. Por lo tanto, los métodos de preprocesamiento buscan optimizar la imagen para que el sistema obtenga una mejor imagen de entrada que permita tener mayor efectividad en el reconocimiento a realizar. Algunos ejemplos de preprocesamiento se pueden ver en la Figura 2 y en la Figura 3.

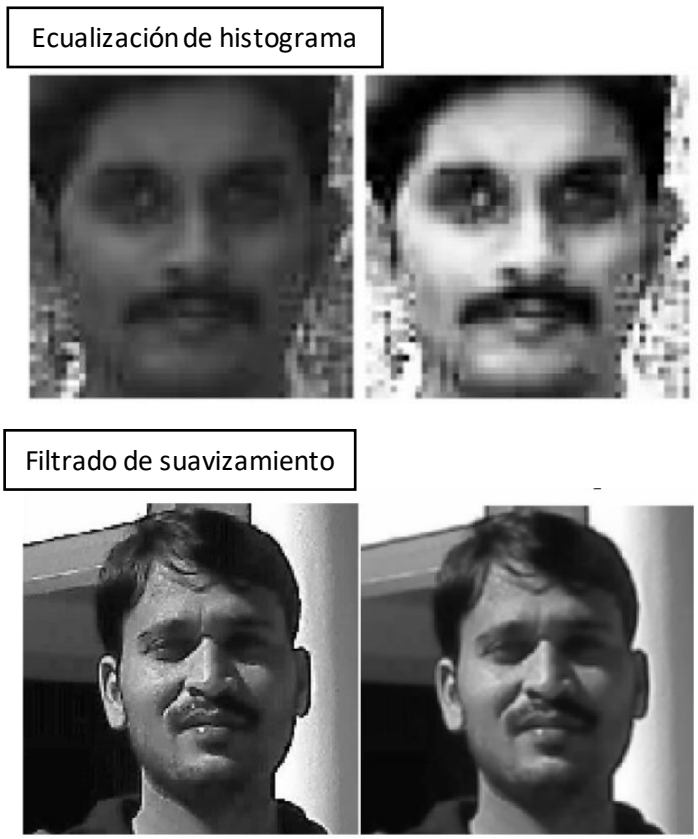


Figura 3. Ejemplo de Ecualización y de Filtrado de la Imagen
Fuente: Dharavath, Talukdar & Laskar (2014). *Improving Face Recognition Rate with Image Preprocessing.* (p. 1172)

Consecuentemente, el estudio plantea analizar los métodos de preprocesamiento tradicionales y su impacto en la verificación facial, evaluando diversos métodos de preprocesamiento en tres modelos de redes neuronales convolucionales, tres bases de datos diferentes y dos métodos de detección facial.

1.2. Formulación del problema

1.2.1. Problema general

PG: ¿Cómo el preprocesamiento de imágenes impacta en la efectividad de la verificación facial empleando visión computacional?

1.2.2. Problemas específicos

PE1: ¿El impacto del preprocesamiento en la efectividad de la verificación facial se verá alterado al analizarse en diferentes bases de datos de rostros disponibles?

PE2: ¿El impacto del preprocesamiento en la efectividad de la verificación facial variará al analizarse con distintas técnicas de preprocesamiento de imágenes?

PE3: ¿El impacto del preprocesamiento en la efectividad de la verificación facial se verá afectado al analizarse con diferentes métodos de detección facial?

PE4: ¿El impacto del preprocesamiento en la efectividad de la verificación facial se alterará al analizarse con diferentes modelos de verificación facial basados en redes neuronales convolucionales?

PE5: ¿Las métricas para medir el desempeño dan una visión comparable mediante una correcta evaluación del impacto del preprocesamiento en la efectividad de la verificación facial?

1.3. Determinación de Objetivos

1.3.1. Objetivo General

OG: Evaluar el impacto del preprocesamiento de imágenes en la efectividad de la verificación facial empleando visión computacional.

1.3.2. Objetivos Específicos

OE1: Obtener y generar datasets de rostros que permitan realizar el análisis de los modelos de verificación facial.

OE2: Aplicar las técnicas de preprocesamiento mediante suavizamiento, agudizamiento, ecualización y alineación para realizar el análisis de la efectividad de la verificación facial.

OE3: Utilizar los métodos HOG y Haar para la extracción de los vectores característicos de la detección facial.

OE4: Aplicar los modelos de redes neuronales convolucionales OpenFace, VGG-Face2 y Light CNN para realizar las evaluaciones de efectividad de la verificación facial.

OE5: Analizar el impacto del preprocesamiento de imágenes en la efectividad de la verificación facial mediante métricas de desempeño de las redes OpenFace, Vgg-Face2 y Light CNN.

1.4. Hipótesis

En base a los objetivos y las preguntas de investigación se plantearon las hipótesis que se describen a continuación.

1.4.1. Hipótesis General

HG: El preprocesamiento de imágenes impacta en la efectividad de los modelos de verificación facial empleando visión computacional.

1.4.2. Hipótesis Especificas

HE1: El impacto del preprocesamiento en la efectividad de la verificación facial se verá afectado por los *datasets* a evaluar.

HE2: El impacto del preprocesamiento en la efectividad de la verificación facial variará en base al tipo de preprocesamiento utilizado.

HE3: El impacto del preprocesamiento en la efectividad de la verificación facial dependerá del método de detección facial utilizado.

HE4: El impacto del preprocesamiento en la efectividad de la verificación facial dependerá de los modelos de redes neuronales convolucionales utilizados.

HE5: Las métricas de desempeño para medir la efectividad serán comparables y permitirán evaluar el impacto del preprocesamiento en la efectividad de la verificación facial.

1.5. Justificación de la investigación

1.5.1. Teórica

La visión por computador está teniendo avances significativos en el ámbito teórico. En los últimos años han surgido desarrollos importante en el reconocimiento de imágenes debido en parte a los nuevos modelos y a las mayores capacidades de procesamiento que ahora se tienen en las computadoras (LeCun, Bengio, & Hinton, 2015).

Los modelos de redes neuronales convolucionales que se utilizarán para la investigación planteada están entre los que brindan los mejores resultados en la actualidad en el campo de visión computacional. Un claro ejemplo de esto, son los resultados del concurso anual de *ImageNet*, en el cual se evalúa la capacidad de algoritmos de clasificar, reconocer o localizar una base de datos de casi 15 millones de imágenes donde estos algoritmos han sido principales ganadores los últimos años (Russakovsky, et al., 2015). Por lo tanto, el trabajo también contribuirá teóricamente al tratar este tipo de algoritmos modernos en una investigación en el habla hispana y en otros contextos.

Asimismo, en el ámbito del preprocesamiento de imágenes para la verificación facial existen estudios que analizan relaciones similares a las planteadas (Dharavath, Talukdar, & Laskar, 2014; Calvo, Baruque, & Corchado, 2013). Sin embargo, no suelen analizar algunas técnicas más modernas como el alineamiento de los rostros o realizar la evaluación en modelos de redes neuronales convolucionales. Además, existe escasa literatura referente a la combinación de varias de estas técnicas en conjunto, incluyendo filtros y alineamiento, así como su impacto en varias bases de datos y mediante modelos de verificación facial diferentes.

Finalmente, en el habla hispana este tipo de estudios son aún más escasos. Por lo tanto, existe la oportunidad de ampliar la teoría en este campo y este trabajo brindará una contribución teórica tanto en el ámbito global como en el habla hispana.

1.5.2. Práctica

En el ámbito práctico, las tecnologías basadas en reconocimiento facial están ganando importancia como método de seguridad a nivel global y en otras industrias (Dharavath, Talukdar, & Laskar, 2014). Un ejemplo de esto es el caso de China donde se tienen 200 millones de cámaras de video vigilancia, aproximadamente 4 veces más que en los Estados Unidos (Mozur, 2018). Ambos países son algunos de los cuales han apostado por el reconocimiento facial como medida de seguridad.

En el Perú, existe la utilización de reconocimiento facial como apoyo en la RENIEC (Registro Nacional de Identificación y Estado Civil) y en la PNP (Policía Nacional del Perú) (RENIEC, 2014). El reconocimiento facial se utiliza para las validaciones biométricas de la RENIEC y para la seguridad ciudadana en la PNP. Esto refuerza su implementación en el ámbito público. Sin embargo, todavía existe campo de crecimiento y de mejora en el contexto local.

Por otro lado, las redes neuronales convolucionales tienen campo de crecimiento en diversas áreas. La implementación de este tipo de modelos no solo sirve para detección y reconocimiento facial sino también para la clasificación de objetos (Szegedy et al. 2014), la descripción de imágenes (Karpathy & Fei-Fei, 2015), la detección de peatones u objetos en las calles (Benenson et al., 2012), la lectura de frases o números (LeCun, 1989), entre muchos otros. Variadas empresas como NVIDIA, Mobileye, Intel, Qualcomm y Samsung están desarrollando chips para redes neuronales convolucionales para facilitar aplicaciones de visión computacional en tiempo real en teléfonos móviles, cámaras, robots y carros autónomos (LeCun, Bengio, & Hinton, Deep Learning, 2015).

Además, estas han sido utilizadas e implementadas por grandes empresas de tecnología como Google, Facebook, Microsoft, IBM, Yahoo! y Twitter, llegando a desarrollar proyectos para desplegar productos relacionados al entendimiento de imágenes (LeCun, Bengio, & Hinton, Deep Learning, 2015). Este es el caso de la venta de productos basados en estos modelos que vuelven disponibles APIs para los desarrolladores permitiendo obtener capacidades similares a cualquier empresa. Un ejemplo de esto son los productos de IBM con sus APIs de *Visual Recognition* y Microsoft con su módulo de *Computer Vision* el cual incluye Face API para verificación facial. Por lo tanto, estos modelos brindan amplias posibilidades y pueden ser aplicados en diversas industrias.

Finalmente, en el ámbito práctico, la investigación se centra en un campo que está en constante expansión. El estudio servirá como apoyo al momento de desarrollar dichos sistemas dado que el análisis de los métodos de preprocesamiento y sus implicancias servirá como referencia para el desarrollo de estas implementaciones en búsqueda de una efectividad mayor tanto en el ámbito global como local.

1.5.3. Metodológica

Por otro lado, en el ámbito metodológico, se planea que el estudio aporte mediante la utilización de metodologías existentes tanto en el campo de la verificación facial como en los métodos relacionados a la evaluación de modelos de visión computacional. Además, se espera

contribuir mediante arquitecturas que apoyen el desarrollo modular de un sistema permitiendo las pruebas automatizadas a realizar durante la investigación. Finalmente, se contribuirá con aplicación de metodologías y métricas estadísticas para el análisis de los modelos y técnicas a evaluar.

1.6. Delimitación del Estudio

1.6.1. Espacial

Se pretende realizar un estudio en el que se evalúe el impacto de los métodos de preprocesamiento en la efectividad de la verificación facial. Se realizará el estudio en tres fuentes de información de imágenes recolectadas. Dos de las bases de datos a considerar son de amplio uso en investigaciones similares a nivel global (Amos, Bartosz, & Satyanaray, 2016; Parkhi, Vedaldi, & Zisserman, 2015; Schroff, Kalenichenko, & Philbin, 2015; Taigman, Yang, Ranzato, & Wolf, 2014): *YoutubeFaces* y *Labeled Faces in the Wild*. Adicionalmente se considera una base de datos del contexto peruano obtenida durante esta investigación.

La base de datos de *Youtube Faces* está compuesta por imágenes de rostros obtenidas de 3425 videos de YouTube de 1595 personas.

La base de datos de *Labeled Faces in the Wild* está compuesta por 13233 imágenes de rostros obtenidas en entornos no controlados de 5749 personas específicamente diseñada para este tipo de pruebas y publicada por investigadores de la Universidad de Massachusetts, Amherst.

Adicionalmente, la base de datos del contexto peruano (DNIs) está compuesta por 102 imágenes de 51 personas. Esta base incluye parejas de fotos de Documentos de Identidad y rostros de personas de Lima, Perú obtenidas en un ambiente controlado mediante un formulario web de registro utilizado desde dispositivos móviles y desplegado para esta investigación.

1.6.2. Temporal

Respecto a la delimitación temporal, la data del contexto peruano fue recolectada desde el año 2016 hasta el 2020. La data recolectada de *YouTube Faces Database* fue publicada en el año 2011 (Wolf, Hassner, & Maoz, 2011). La data recolectada de *Labeled Faces in the Wild* fue publicada en el año 2007 (Huang, Ramesh, Berg, & Learned-Miller, 2007).

1.6.3. Conceptual

El estudio se centrará en el análisis de los métodos de preprocesamiento en tres modelos de redes neuronales convolucionales de alta efectividad: *OpenFace* (Amos, Bartosz, &

Satyanaray, 2016), *VGGFace2* (Cao, Shen, Xie, Parkhi, & Zisserman, 2018) y *Light CNN* (Wu, He, Sun, & Tan, 2018). Asimismo, el estudio tendrá como alcance dos métodos de detección facial: mediante descriptores HOG (*Histogram of Oriented Gradients*) (Dalal & Triggs, 2005) y mediante descriptores Haar (Viola & Jones, 2001). Finalmente, se evaluarán distintos métodos de preprocesamiento tradicionales concernientes a filtrado, ecualización y alineamiento.

CAPÍTULO II: MARCO TEÓRICO

En primer lugar, se describen los antecedentes principales para la investigación. Además, se definirán los fundamentos teóricos relacionados con detección de rostros, preprocesamiento de imágenes, extracción de características y reconocimiento de rostros.

2.1. Antecedentes de la Investigación

Se empezará describiendo investigaciones relacionadas con los modelos de redes neuronales convolucionales para la verificación facial, así como investigaciones en las cuales se basan dichos modelos actuales. Luego, se describirán estudios relacionados con los métodos de preprocesamiento y su impacto en el reconocimiento y verificación facial.

2.1.1. Modelos para la Verificación Facial

A continuación, se describen los antecedentes relacionados con los modelos existentes para la verificación facial.

En primer lugar, dentro de los estudios relacionados con modelos algorítmicos para la verificación facial, se detalla el estudio titulado *OpenFace: A general-purpose fase recognition library with mobile applications* (Amos, Bartosz, & Satyanaray, 2016). Este busca brindar públicamente un modelo de verificación facial de alta efectividad.

Los últimos avances en reconocimiento facial están dominados por conjuntos de datos de la industria y del gobierno. Asimismo, los escenarios de aplicación en dispositivos móviles del reconocimiento facial deben ser capaces de realizar el reconocimiento en tiempo real ante contextos cambiantes. *OpenFace* busca crear un modelo *open source* basado en datos libres que funcione en escenarios móviles.

Por lo tanto, busca resolver el problema del desarrollo de un modelo de reconocimiento y verificación facial altamente efectivo y hacerlo disponible al público mediante una librería de código abierto. Para lograr esto, el estudio pretende desarrollar una librería de código abierto

disponible al público que aplique modelos de reconocimiento y verificación facial altamente efectivos.

OpenFace utiliza un algoritmo de redes neuronales convolucionales para la verificación facial. Este aplicó principalmente el proceso de verificación facial de *DeepFace* (*detect => align => represent => classify*) y el modelo de redes neuronales convolucionales y de *Triplet Loss* de *FaceNet*. Para esto desarrolló y codificó un modelo de red neuronal convolucional basado en dicha literatura y entrenado con 500 mil imágenes de acceso público de las bases de datos *CASIA-WebFace* (Yi, Lei, Liao, & Li, 2014) y *FaceScrub*. (Ng & Winkler, 2014). Las pruebas fueron realizadas en la base de datos *Labeled Faces in the Wild (LFW)*.

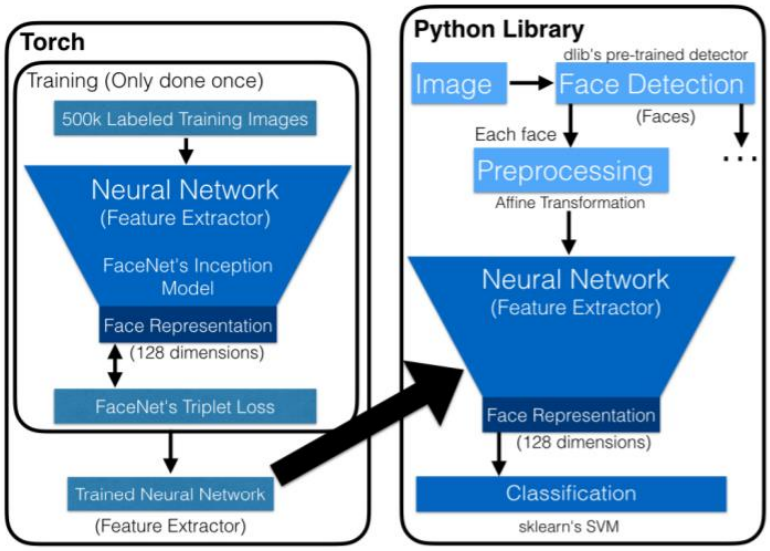


Figura 4. Estructura y proceso de proyecto OpenFace
Fuente: Amos, Bartosz & Satyanaray (2016). *OpenFace: A general-purpose face recognition library with mobile applications.* (p. 5)

Asimismo, dentro del aspecto técnico, se utilizó la librería *Torch* basado en el lenguaje de programación Lua para el entrenamiento del modelo y *Python* con las librerías *dlib* y *opencv* para consumir el modelo entrenado.

Dentro de los resultados, se alcanzó una efectividad de 92.92%, cercana a la planteada en los estudios predecesores utilizando una cantidad de datos mucho menor. Esta comparativa se observa en la Figura 5.

Technique	Accuracy
Human-level (cropped) [KBBN09]	0.9753
Eigenfaces (no outside data) [TP91][3]	0.6002 ± 0.0079
FaceNet [SKP15]	0.9964 ± 0.009
DeepFace-ensemble [TYRW14]	0.9735 ± 0.0025
OpenFace (ours)	0.9292 ± 0.0134

Figura 5. Efectividad de OpenFace en la base de datos de LFW versus otros modelos
Fuente: Amos, Bartosz & Satyanaray (2016). *OpenFace: A general-purpose face recognition library with mobile applications.* (p. 8)

Finalmente, como resultado de la investigación se desarrolló *OpenFace*, una librería de reconocimiento facial *open source*, haciendo disponible un modelo entrenado basado en datos abiertos. Asimismo, se obtuvo una efectividad competitiva teniendo una menor cantidad de datos de entrenamiento.

Otra investigación acerca de la creación de estos modelos en base a data pública es *Deep Face Recognition* (Parkhi, Vedaldi, & Zisserman, 2015). El modelo que se desarrolló es denominado *VGGFace* dado que los autores son miembros del *Visual Geometry Group* de Oxford. Posteriormente, en otra investigación de autores del mismo grupo se desarrollaron modelos denominados *VGGFace2* (Cao, Shen, Xie, Parkhi, & Zisserman, 2018), construidos en base a los lineamientos de esta primera investigación, con una mayor cantidad de datos y una arquitectura basada en ResNet-50 con y sin bloques *SeNet* (*Squeeze-and-Excitation*).

En el ámbito del reconocimiento facial, los conjuntos de datos públicos han sido escasos y en gran medida esto ha generado que los principales avances se vean restringidos a gigantes de Internet como Facebook y Google. Entrenan modelos con magnitudes de datos que están fuera de las capacidades de la mayoría de los grupos de investigación en academia.

Por consiguiente, esta investigación busca determinar cómo crear un conjunto de datos de rostros considerablemente grande requiriendo limitado esfuerzo humano para la anotación. Asimismo, busca determinar cómo generar una arquitectura de redes neuronales convolucionales para la identificación facial con alta efectividad que compita con los últimos modelos de la industria.

VGGFace propone un procedimiento para crear un conjunto de datos de rostros. Adicionalmente, propone la evaluación y entrenamiento de una arquitectura de redes neuronales convolucionales para la identificación facial que pueda competir con los últimos modelos de la industria.

Para la construcción del conjunto de datos de rostros, primero se obtuvo una lista de candidatos de 5000 identidades obtenidas de Freebase y de IMDB recolectando imágenes

representativas para varias celebridades, un aproximado de 200 imágenes por persona. El segundo paso fue filtrar la lista de candidatos, se eliminaron las personas con poca representación al buscarlas en Google y se eliminó el traslape con *benchmarks* públicos obteniendo finalmente un conjunto de datos de 2622 celebridades. El tercer paso fue ordenar los sets de imágenes, se descargaron 2000 imágenes por persona de Google y de Bing las cuales fueron ordenadas en base a un clasificador entrenado con las imágenes obtenidas en el paso 1, dejando solo las primeras 1000 imágenes pertenecientes a cada persona. El cuarto paso se centró en eliminar imágenes muy similares basándose en el descriptor *VLAD (Vector of Locally Aggregated Descriptors)*. Finalmente, en el quinto paso, se hizo un filtrado manual para terminar de estructurar las imágenes para cada identidad. Al final se obtuvo un conjunto de datos con un promedio de 375 imágenes por persona para las 2622 personas, dando un total de 982,803 imágenes. Se realizó el entrenamiento del modelo utilizando las imágenes obtenidas del tercer paso (antes de la depuración de imágenes similares y filtrado manual) y del quinto paso (con la data final filtrada). Asimismo, se hizo pruebas con alineación y sin alineación de rostros y con función de *triplet loss* para el entrenamiento.

Por el lado de las herramientas de desarrollo, se utilizó *MatConvNet* del *MATLAB Toolbox* junto con librerías de NVIDIA *CuDNN* para acelerar el entrenamiento.

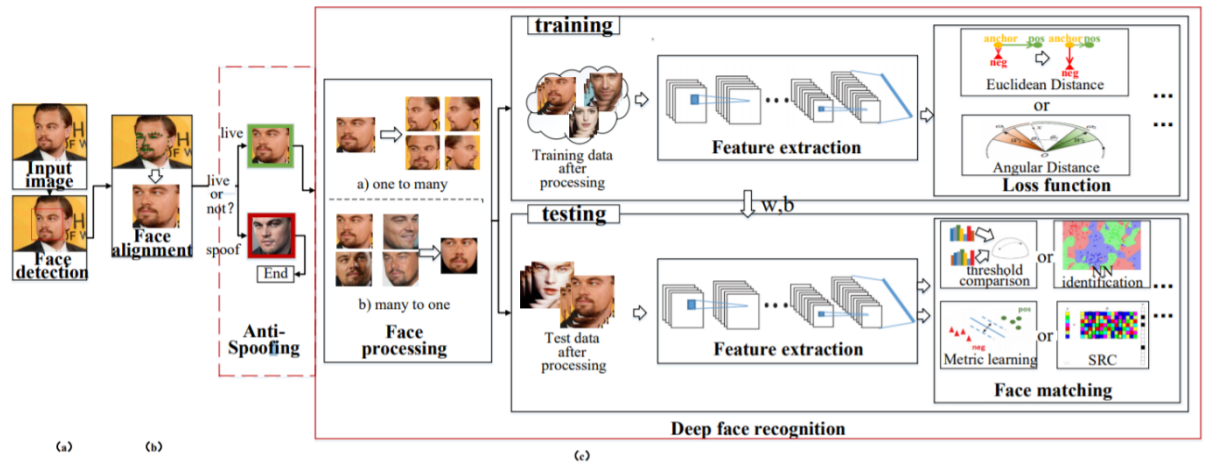


Figura 6. Sistema de reconocimiento facial de con detección facial y alineamiento
Fuente: Wang & Deng (2018). *Deep Face Recognition: A Survey.* (p. 4)

Dentro de los resultados, para la evaluación se utilizó tanto el *EER* (*equal error rate*) como la efectividad. El *EER* es la efectividad en el punto de corte en el cual los falsos positivos igualan a los falsos negativos. Se encontró que entrenar los modelos con data sin filtrado manual, brindó mejores resultados que la data final ordenada y depurada, teniendo un 95.8% versus un 92.83% de EER en *Labeled Faces in the Wild*. Asimismo, realizar el alineamiento

de los rostros durante la evaluación y utilizar el *Triplet Loss* ayudó a incrementar el *EER* llegando a 99.13% en *LFW*. Además, se obtuvo una efectividad de 98.95% en *LFW* y 97.3% en *Youtube Faces DB*.

Por lo tanto, en la investigación, se logró diseñar un procedimiento que es capaz de estructurar un conjunto de datos grande, con poco ruido y minimizando la cantidad de trabajo manual de anotación requerido. Asimismo, se demostró que una red neuronal convolucional sin muchas complejidades con el entrenamiento correcto puede ser comparable a los últimos modelos desarrollados en la industria. Esta red fue la que se denominó *VGGFace*.

Por otra parte, otro modelo relevante es el denominado *Light CNN* igualmente utilizado para el reconocimiento facial. A continuación, se describe dicha investigación titulada *A Light CNN for deep face representation with noisy labels* (Wu, He, Sun, & Tan, 2018).

En los últimos años, las redes neuronales convolucionales se han convertido en las técnicas más populares para resolver problemas de visión computacional y su performance se ha incrementado considerablemente. Para encontrar una efectividad óptima, la cantidad de data de entrenamiento ha ido incrementando. Varios datasets masivos libres se han ido creando como CASIA-WebFace, CelebFaces+ (Sun, Wang, & Tang, 2014), *VGGFace*, entre otros. Sin embargo, estos contienen en general mucho ruido especialmente cuando han sido recolectados de forma automática.

A causa de lo planteado, la investigación busca determinar cómo desarrollar un modelo de reconocimiento facial basado en data masiva con muchas anotaciones ruidosas. Por consiguiente, la investigación pretende presentar un marco y arquitectura para entrenar un modelo de reconocimiento facial que aborde dichos problemas.

En el artículo se propuso desarrollar redes neuronales convolucionales unificándolas con características que permitan reducir el ruido y hacer más eficiente el modelo. Para desarrollar estos modelos se planteó una operación de Max-Feature-Map (MFM) para obtener una representación compacta y realizar un filtrado que permita depurar neuronas en cada capa separando el ruido de lo útil. Se utilizó filtros pequeños, redes en redes (*Network in Network*) y bloques residuales (*residual blocks*) para reducir la cantidad de espacio utilizado en parámetros y mejorar la performance. Tres modelos fueron entrenados en el conjunto de datos de MS-Celeb-1M limpiado. Dicho conjunto de datos contiene 79,077 identidades en total con 5,049,824 imágenes. El primer modelo Light CNN-4 está construido con cuatro capas convolucionales con operaciones MFM y cuatro capas de Max-Pooling teniendo alrededor de 4,095 miles de parámetros. El segundo modelo Light CNN-9 contiene cinco capas convolucionales, cuatro capas de redes en redes (NIN), capas de MFM y cuatro capas de max-

pooling con alrededor de 5,556 miles de parámetros. El tercer modelo Light CNN-29 contiene veintinueve capas. Este modelo incluye bloques residuales y cada uno tiene dos capas 3x3 de convolución y dos operaciones MFM teniendo 12,637 miles de parámetros.

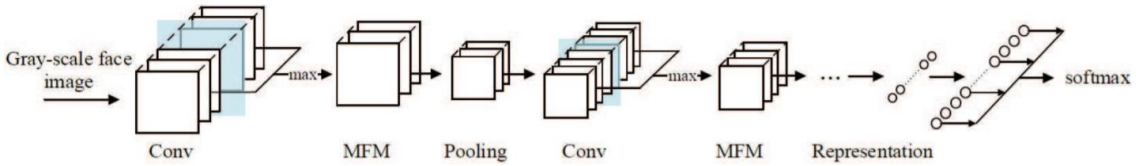


Figura 7. Arquitectura de Light CNN
Fuente: Wang & Deng (2018). *Deep Face Recognition: A Survey*

Los resultados de *Light CNN* en LFW fueron de 99.33% de efectividad con Light CNN-29. En el caso de la base de datos *Youtube Faces DB* se obtuvo 95.54% de efectividad. Asimismo, se evaluó la performance en un solo núcleo de Core i7-4790. El modelo Light CNN-4 se ejecutó extrayendo una representación de rostro en 75ms, Light CNN-9 en 67ms y Light CNN-29 en 121ms.

Por lo tanto, en la investigación se obtuvo un modelo que controla las anotaciones ruidosas lo que facilita que se pueda aplicar a sistemas de reconocimiento facial en tiempo real. Asimismo, el modelo *Light CNN* es más rápido y pequeño que otros de los más reconocidos en la actualidad siendo eficiente en su uso.

Por otro lado, dentro de los estudios relacionados con modelos algorítmicos para la verificación facial, se describe el estudio titulado *FaceNet: A Unified Embedding for Face Recognition and Clustering* (Schroff, Kalenichenko, & Philbin, 2015).

El estudio de FaceNet aborda la problemática de algunas redes neuronales convolucionales utilizadas para el reconocimiento facial relacionados con la generalización del modelo a nuevos rostros distintos a los utilizados para el entrenamiento. Para lograr esto, estos modelos anteriores debían tener una capa intermedia dentro de la red para generalizarlo a otros contextos. Al utilizar esta capa intermedia, las representaciones finales de rostros podían llegar a ser de miles de dimensiones. *FaceNet* es un modelo que genera representaciones de 128 dimensiones por cada rostro, lo cual es mucho más eficiente, esto gracias a su función de pérdida de *triplet loss*.

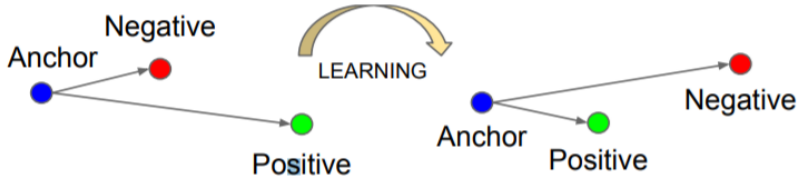


Figura 8. Triplet Loss minimiza la distancia entre imágenes de la misma persona
Fuente: Schroff, Kalenichenko & Philbin (2015). *FaceNet: A Unified Embedding for Face Recognition and Clustering.* (p. 3)

El estudio pretende resolver lo planteado mediante la generación de un modelo de verificación facial efectivo que obtenga un vector de valores representativo y comparable manteniendo una dimensionalidad eficiente.

La investigación se basa en obtener un vector de valores que se encuentre dentro de un espacio euclidiano por cada imagen. El hecho de generar un vector con estas características hace que sea posible realizar la comparación de la similitud de dos rostros, la verificación facial y *clustering* mediante el cálculo de distancias euclidianas entre los vectores de cada imagen. Asimismo, el estudio utilizó redes neuronales convolucionales que son una extensión de las redes neuronales tradicionales especialmente diseñadas para imágenes. Estas redes fueron entrenadas usando el *triplet loss* el cual busca minimizar las distancias euclidianas entre los vectores de características de imágenes de la misma persona y maximizar las distancias entre imágenes de personas diferentes. Asimismo, otra característica importante de este modelo es el hecho que utiliza capas *Inception* en sus redes.

Se utilizaron entre 100 millones y 200 millones de imágenes en miniatura de rostros para el entrenamiento de la red con alrededor de 8 millones de personas diferentes. Las imágenes eran desde 96x96 pixeles de tamaño hasta 224x224 pixeles. Para las pruebas del modelo se utilizaron 1 millón de imágenes de la misma base datos, 12 mil imágenes de fotos personales, 13 mil imágenes de la base de datos pública LFW (*Labeled faces in the Wild*) y 3425 videos de la base *YouTube Faces Database*.

Entre los resultados de *FaceNet* se tuvo una clasificación correcta de 89.4% con la base de 1 millón de imágenes, 99.63% en LFW y 95.12% en *Youtube Faces DB* al realizar la comparación entre rostros.

Finalmente, el método de aprendizaje mediante *triplet loss*, diferencia este modelo de otros métodos que requieren una capa intermedia que realice una generalización o de métodos que requieren post-procesamiento adicional. El entrenamiento propuesto por *FaceNet* simplifica esto y muestra que optimizar la función de pérdida mejora el rendimiento. Finalmente, otro

beneficio del modelo es que requiere un mínimo alineamiento de los rostros en su entrenamiento.

Por otro lado, otra investigación acerca de la creación de estos modelos de redes neuronales convolucionales es *Going deeper with convolutions* (Szegedy, y otros, 2014). Este modelo, fue ganador del concurso de ImageNet 2014 y se caracteriza por la creación de la capa *Inception*. Dicha capa fue una contribución importante, siendo utilizada en el estudio anterior para el modelo *FaceNet*.

Este estudio abarca la problemática del reconocimiento de imágenes. El estudio plantea que la calidad del reconocimiento de imágenes y detección de objetos ha ido progresando a un paso dramático y esto no se debe solo a hardware más poderoso, bases de datos más grandes o modelos más grandes sino a nuevas ideas, nuevos algoritmos y mejoras en la arquitectura de las redes. Asimismo, con las necesidades de los dispositivos móviles, la eficiencia de los algoritmos en uso de batería y memoria gana mayor importancia. Por lo tanto, el modelo desarrollado busca una arquitectura eficiente para la visión computacional denominada *Inception*.

La investigación busca determinar cómo se puede mejorar la utilización de recursos y a la vez aumentar la complejidad de una red neuronal convolucional en ancho y profundidad. En consecuencia, plantea la creación de una red neuronal convolucional que mejore la utilización de recursos a la vez que se incrementa la complejidad dentro del modelo.

Esto fue logrado al proponer un nuevo tipo de capa llamado *Inception*. El estudio utilizó 1.2 millones de imágenes para el entrenamiento de la red, 50 mil imágenes para validación y 100 mil imágenes para pruebas. El modelo debía clasificar 200 clases distintas. En la prueba de clasificación, el modelo asigna las 5 clases más probables al modelo. Se hicieron las pruebas en el conjunto de datos de la competencia de *ImageNet*.

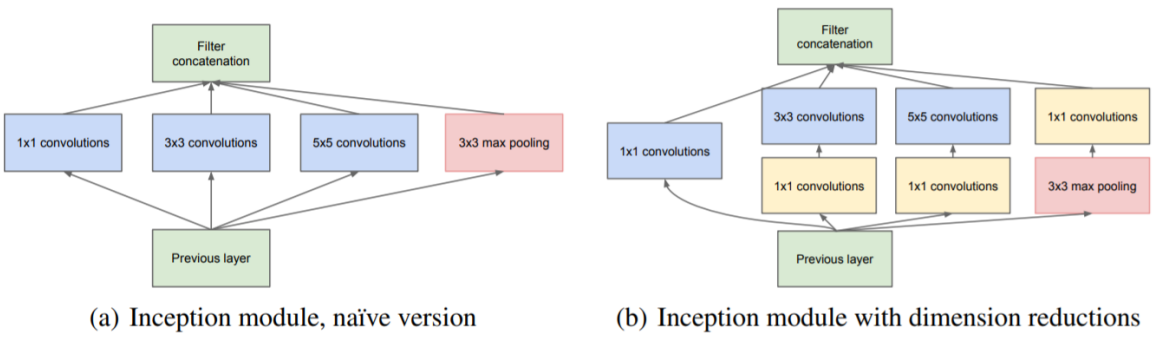


Figura 9. Módulo Inception en red neuronal convolucional
Fuente: Szegedy, et al. (2014). *Going deeper with convolutions.* (p. 5)

Entre los resultados, el 93.33% de clasificaciones tuvieron la clase correcta en el top 5 asignado. Asimismo, se probó la detección de los 200 objetos dentro de las imágenes. En este caso se obtuvo 60% de detecciones correctas.

En definitiva, la principal ventaja del modelo *FaceNet* es el significativo aumento de efectividad a cambio de un modesto incremento de requerimientos computacionales en comparación con redes menos profundas y menos anchas. Esto demuestra que es factible y útil realizar modelos más dispersos como el planteado.

Adicionalmente a los estudios descritos, es importante recalcar un estudio publicado en 1989 y que es uno de los precursores más importantes de las redes neuronales convolucionales titulado *Backpropagation Applied to Handwritten Zip Code Recognition* (LeCun, et al., 1989).

Estudios anteriores a este probaron que se podía obtener una buena generalización de tareas complejas diseñando una red que contenga un cierta cantidad de conocimiento previo sobre la tarea. Este estudio plantea que se debe reducir la cantidad de parámetros libres en la red sin reducir su poder computacional. Estudios anteriores recibían vectores de características como entrada. Este estudio recibe directamente las imágenes probando que las redes pueden recibir gran cantidad de data cruda. Asimismo, se aplica *backpropagation* (propagación hacia atrás) para reconocer dígitos escritos a mano de códigos postales.

La publicación buscó desarrollar un modelo de visión por computador que sea capaz de reconocer códigos postales en imágenes para el servicio postal de Estados Unidos (*US Postal Service*). Por lo tanto, tuvo el objetivo de desarrollar un modelo de redes neuronales en el servicio postal de Estados Unidos (*US Postal Service*) para el reconocimiento de códigos postales en imágenes.

Dentro de su metodología, este utiliza un algoritmo de redes neuronales basado en *backpropagation* para el aprendizaje y se utilizaron 167,263 patrones para entrenar el modelo.

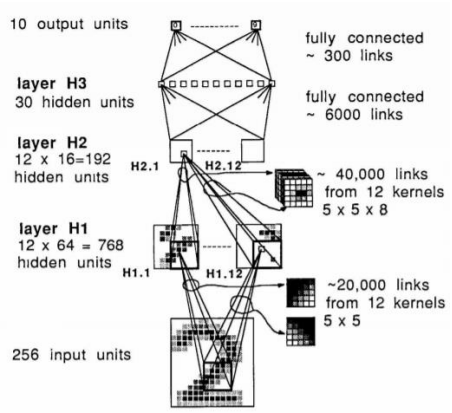


Figura 10. Arquitectura de red neuronal convolucional para detección de códigos postales
Fuente: LeCun, et al. (1989). *Backpropagation Applied to Handwritten Zip Code Recognition*. (p. 548)

Como resultado se obtuvo una precisión de 95% de patrones reconocidos correctamente.

En consecuencia, se aplicó el aprendizaje basado en *backpropagation* exitosamente a una tarea amplia del mundo real. Los resultados fueron de alta efectividad aplicados a data cruda con poco preprocesamiento. La red tenía conexiones con pocos parámetros libres. La red final de conexiones y sus pesos puede sin problema ser implementada en hardware comercial.

2.1.2. Métodos de Preprocesamiento para la Verificación Facial

A continuación, se describen los antecedentes relacionados a los métodos de preprocesamiento para la verificación facial.

Una investigación acerca del impacto del preprocesamiento de imágenes en la efectividad del reconocimiento facial es *Improving Face Recognition Rate with Image Preprocessing* (Dharavath, Talukdar, & Laskar, 2014).

La investigación describe cómo, en los últimos años, la verificación facial ha ganado mayor significancia como método biométrico de autenticación. Muchas industrias y gobiernos están buscando implementar el reconocimiento facial para la autenticación. Sin embargo, tienen problemas debido a la baja efectividad en reconocer a una persona en tiempo real debido a varios factores, siendo la calidad de la imagen uno de los más importantes. Varias técnicas de procesamiento pueden ser utilizadas para mejorar la imagen capturada y la efectividad del reconocimiento.

Por lo tanto, el estudio tiene el objetivo de mejorar el ratio de reconocimiento facial mediante técnicas de preprocesamiento de imágenes y de extracción de características.

Para la metodología se evaluó un sistema de reconocimiento facial el cual tiene un proceso general mostrado en la Figura 11. Este sigue los siguientes pasos: detección de rostro (*face detection*), preprocesamiento (*pre-processing*), extracción de características (*feature extracción*), verificación (*matching*) y finalmente retornando el resultado al dispositivo (*application device*).

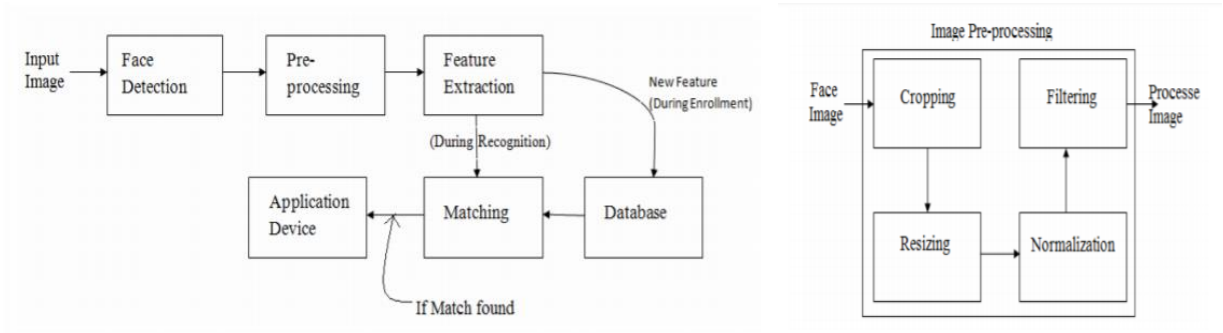


Figura 11. Proceso de un sistema de reconocimiento facial y proceso de preprocesamiento
Fuente: Dharavath, Talukdar, & Laskar (2014). *Improving Face Recognition Rate with Image Preprocessing*. (p. 117)

Se utilizaron las técnicas de preprocesamiento de recorte, escalamiento, ecualización y filtrado gausiano. Finalmente se evaluaron las técnicas de extracción de características (basadas en *eigenfaces*, en transformación discreta del coseno y en un método combinado). Se hicieron pruebas en 35 personas en la base de datos *Speech and Image Processing Research Lab (SIPRL)* con un promedio de 19 imágenes por persona.

Como resultado, se llegó a incrementar en promedio hasta 66.32% en el ratio de reconocimiento facial mediante la utilización de los métodos planteados. Asimismo, el método combinado de *eigenfaces* y transformación discreta del coseno generó una efectividad promedio de hasta 92.62%, esto siendo superior que la de cualquiera de los otros dos métodos de forma independiente.

Por lo tanto, se comprobó que el ratio de reconocimiento facial mejora con la utilización de métodos de preprocesamiento en especial cuando las imágenes son de diversos tamaños, tienen diferentes condiciones de iluminación o son ruidosas. Asimismo, la combinación de *eigenfaces* y transformación discreta del coseno para la extracción de características ha mostrado buenos resultados incluso antes del preprocesamiento.

Por otra parte, otro estudio acerca del impacto del preprocesamiento de imágenes en la efectividad del reconocimiento facial es *Study of the Pre-procesing Impact in a Face Recognition System* (Calvo, Baruque, & Corchado, 2013).

Este plantea que un sistema de clasificación es altamente influenciado por el preprocesamiento de datos y parte del éxito recae en seleccionar las mejores técnicas para la tarea a realizar. Debido a la naturaleza de las imágenes y las personas, la data inevitablemente incluirá mayor ruido que un problema clásico de clasificación. Hay factores humanos como actitud del modelo representado, gestos, ropa, distancia, tatuajes o cambios de apariencia. Asimismo, hay factores técnicos que incrementan la complejidad relacionados a los sistemas

de captura de imágenes: el sistema de iluminación (flash), o el tipo de data desde donde se obtiene la imagen (video, 3D, infrarrojo, fotografía, entre otros). Además, existen factores externos relacionados al ambiente como las condiciones de luz, el fondo, la temperatura y la presencia de otras personas.

´La investigación tiene el objetivo de evaluar el impacto del preprocesamiento en un sistema de reconocimiento facial.

La evaluación se realizó tomando en cuenta una estructura general del sistema basado en 5 fases: captura de imagen, detección facial, preprocesamiento, extracción de características y reconocimiento facial. La estructura general se observa en la Figura 12.

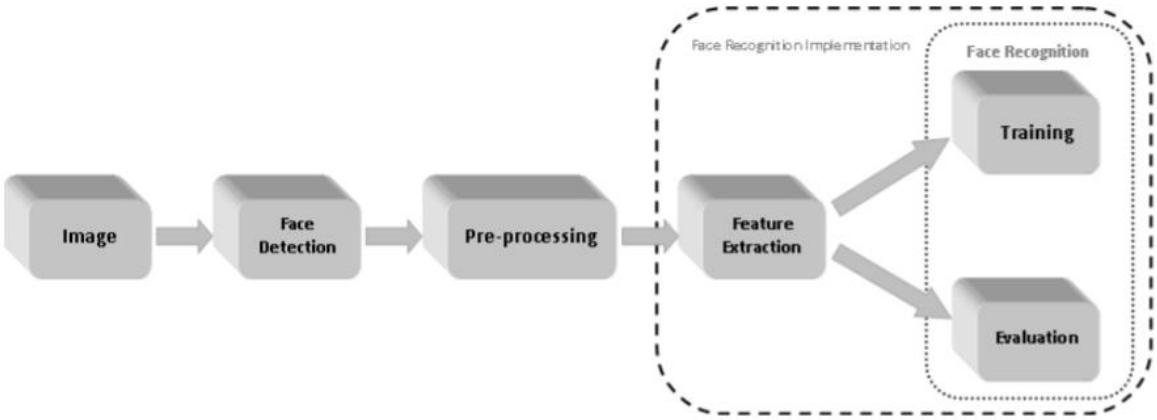


Figura 12. Estructura principal de un sistema de reconocimiento facial
Fuente: Calvo, Baruque & Corchado (2013). *Study of the Pre-processing Impact in a Facial Recognition System.* (p. 335)

Para la detección facial se utilizó la extracción de características SMQT (*Successive Mean Quantization Transform*) y clasificación SNoW (*Sparse Network of Winnows*). Para el preprocesamiento se utilizó la normalización mediante la transformación discreta del coseno y mediante la ecualización RGB. Asimismo, se utilizó el alineamiento de la imagen mediante el Coeficiente de Correlación Mejorado (ECC) que busca estimar la transformación 2D entre una imagen y una plantilla para alinearla; y el alineamiento mediante las coordenadas de los ojos las cuales se detectaban utilizando características Haar. Para el reconocimiento facial se utilizó *eigenfaces*, *fisherfaces* y el modelo oculto de Márkov (HMM). Se utilizó la validación cruzada (*cross-validation*) con 6 grupos de 60 imágenes (6 por persona) utilizando 50 para entrenamiento y 10 para pruebas. Se hicieron las pruebas en dos bases de datos: Caltech con imágenes de personas con diferentes fondos, condiciones de iluminación, expresión facial y distancia a la cámara y ORL donde las imágenes tienen el mismo fondo, posición y distancia.

Dentro de los resultados, se obtuvo hasta 93% de efectividad mediante la combinación de la transformación de coseno discreta con la alineación con el coeficiente de correlación mejorado utilizando *eigenfaces*. Con *fisherfaces* la efectividad más alta fue de 65% mediante el alineamiento de ojos y la alineación con el coeficiente de correlación. En el caso del modelo oculto de Márkov se obtuvo los mejores resultados de 85% sin ningún método de preprocesamiento o alineación.

Por consiguiente, los métodos diseñados para evitar las variaciones de luz, ruido o pose como *fisherfaces* y el modelo oculto de Márkov demostraron ser más robustos sin preprocesamiento previo mientras que modelos como *eigenfaces* demostraron mejoras considerables. Cuando las condiciones de la captura de la imagen no incluyen un ambiente controlado, los métodos de clasificación tienen problemas para obtener buenos niveles de efectividad que un buen preprocesamiento puede solucionar. La fase de preprocesamiento no es solo importante sino también delicada dado que se debe conocer el funcionamiento del clasificador para elegir el adecuado preprocesamiento.

A partir de los antecedentes se puede concluir que las implementaciones de este tipo de algoritmos en los últimos años se han desarrollado considerablemente y aumentado su efectividad. Asimismo, se han ido desarrollando alternativas a los modelos entrenados con conjuntos de datos privados de la industria mediante la creación de bases de datos públicas alternativas y la creación de modelos que exploten dichos conjuntos de datos. Por otro lado, algunos métodos de preprocesamiento han sido probados en técnicas de verificación facial pero escasos estudios se han centrado en la variedad de técnicas utilizadas en esta investigación, así como en modelos modernos como los basados en redes neuronales convolucionales. Por lo tanto, esta investigación busca complementar y expandir lo encontrado con los estudios antecesores aplicando una variedad de evaluaciones en técnicas modernas.

2.2. Bases Teóricas

Esta sección describe la teoría relacionada a las variables de estudio, al proceso de verificación facial y a las metodologías para el desarrollo de la investigación.

El proceso de verificación facial abarca las siguientes fases de acuerdo con *DeepFace* (Taigman, Yang, Ranzato, & Wolf, 2014) y *OpenFace* (Amos, Bartosz, & Satyanaray, 2016):

Detectar → *Alinear* → *Representar* → *Clasificar*.

Otros estudios plantean el proceso a partir variaciones similares en las fases. Un ejemplo de esto es el siguiente (Calvo, Baruque, & Corchado, 2013; Dharavath, Talukdar, & Laskar, 2014):

Detección de rostros → Preprocesamiento de imágenes → Extracción de Características → Evaluación o Comparación

En general, pueden existir ligeras variantes en el proceso, pero en el fondo representan lo mismo. Siendo el alineamiento un tipo de preprocesamiento, se está describiendo esencialmente el mismo proceso. Por lo tanto, para fines de esta investigación, se estructurará el proceso de la siguiente manera:

1. Detección de rostros
2. Preprocesamiento de imágenes
3. Extracción de características
4. Verificación facial

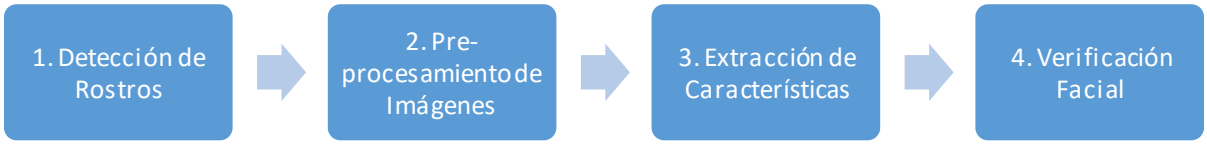


Figura 13. Proceso de Verificación Facial
Fuente: Elaboración Propia

En el punto 2.2.1 y 2.2.2 se abordará a detalle el Preprocesamiento de Imágenes y La Efectividad de la Verificación Facial que tienen relación directa con las variables principales de la investigación y con las etapas 2 y 4 del proceso. Posteriormente, en los puntos 2.2.3, 2.2.4 y 2.2.5 se detallarán los demás elementos del proceso y de visión computacional y en el punto 2.2.6, 2.2.7 y 2.2.8 se describirán algunas metodologías relacionadas al estudio.

2.2.1. Preprocesamiento de Imágenes

A continuación, se describirán las técnicas principales del preprocesamiento de imágenes. Los conceptos que se observarán a continuación están enfocados en técnicas genéricas de preprocesamiento que pueden aplicarse en distintos algoritmos de visión computacional incluyendo campos relacionados como la detección de rostros. Asimismo, se presentarán temas más centrados en la verificación facial, como es el caso del alineamiento de rostros. En la Figura 14 se resalta en color naranja la etapa del proceso que se está describiendo en esta sección.

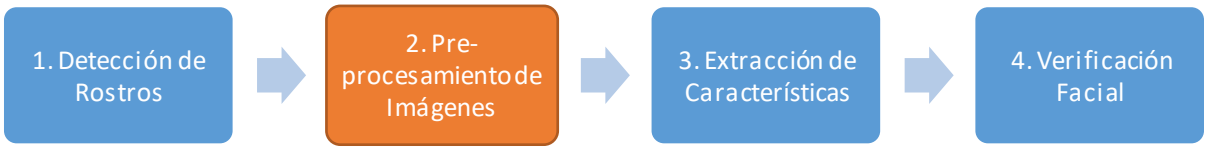


Figura 14. Proceso de Verificación Facial
Fuente: Elaboración Propia

Según Sucar y Gomez (2008), generalmente antes de realizar la extracción de características es necesario realizar el procesamiento temprano que consiste en resaltar los aspectos positivos y eliminar los no positivos dentro de una imagen. Un ejemplo de aspectos a eliminar es el ruido en una imagen.

El uso inteligente del preprocesamiento de imágenes puede proveer beneficios para resolver problemas que pueden llevar a una mejor detección de características locales y globales (Krig, 2014). Al afectar la imagen, estos métodos también pueden brindar resultados negativos por lo que depende de cómo se utilicen.

2.2.1.1. La Imagen

Previo a entender cómo aplicar las técnicas de preprocesamiento, es importante entender cómo se compone una imagen matemáticamente, de tal forma que las técnicas puedan aplicarse.

La imagen al ser capturada en un plano, representa los colores en dos dimensiones. Al realizar la transformación del mundo real al plano de la imagen se tiene que el "concepto de imagen está asociado a una función bidimensional f(x,y)" (De la Escalera, 2001). En el caso de las imágenes digitales, la imagen es en esencia una representación de valores numéricos. Es decir, la imagen se representa mediante valores establecidos en una matriz. Cada representación de color vendría a ser un píxel de la imagen. Para ilustrar mejor este concepto se muestra la Figura 15.

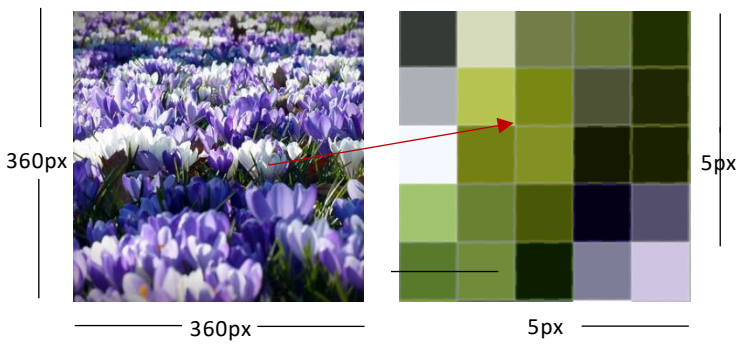


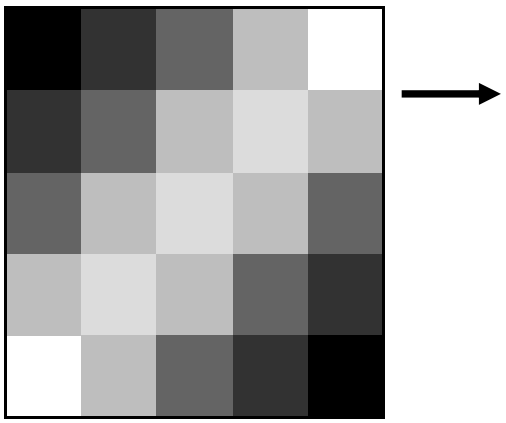
Figura 15. Acercamiento de la imagen
Fuente: Elaboración Propia

En la Figura se puede observar cómo una imagen está representada por cierta cantidad de unidades en un plano de dos dimensiones. A estas unidades se les denomina pixeles. Si se agranda una región pequeña de la imagen, se pueden observar algunos de estos pixeles a detalle, se puede ver que cada uno tiene un color único.

Cada uno de esos colores es representado por valores numéricos. Existen diversas representaciones que se pueden utilizar. En este caso se describirán algunas de las más comunes. Estas representaciones del color pueden ser escalares (De la Escalera, 2001) como se verá a continuación.

2.2.1.1.1. Imagen a Escala de Grises

Una imagen a escala de grises cuenta con la representación más simple. Se trata de una representación escalar por cada pixel. Esta representación suele ser dada por números en una escala entre 0 y 255 significando el nivel de iluminación de la imagen en cada pixel. "El 0 corresponde a un objeto no iluminado o que absorbe todos los rayos luminosos que inciden sobre él (negro), y el nivel 255 a un objeto muy iluminado o que refleja todos los rayos que inciden sobre él (blanco)" (De la Escalera, 2001). Para ejemplificar mejor esto se puede observar la Figura 16:



0	50	100	190	255
50	100	190	220	190
100	190	220	190	100
190	220	190	100	50
255	190	100	50	0

Figura 16. Representación de imagen en escala de grises
Fuente: Elaboración Propia

En la Figura 16 se muestra la representación de una imagen en escala de grises. Mientras más cercano es el número a 255, el pixel tiene una mayor iluminación. Por lo tanto, se tienen pixeles completamente negros en la esquina superior izquierda e inferior derecha (con valor 0) y pixeles blancos en la esquina superior derecha y la esquina inferior izquierda (con valor 255).

En caso de que se deseen más colores en la imagen, se tendrá que utilizar valores vectoriales en vez de escalares como es el caso de las imágenes RGB.

Finalmente, cabe mencionar que, al tener 256 valores posibles, estos se pueden almacenar en 8 bits. 1 bit es la unidad mínima de información pudiendo representar dos estados: 0 o 1 (Destruels, 2015). Mientras más bits se tengan en un pixel, mayores variantes de color se pueden obtener.

2.2.1.1.2. Imagen RGB

Según Sucar y Gómez (2008), el color que observa un humano parte de "la respuesta humana a diferentes longitudes de onda del espectro visible (400 - 700 nm) (…) existen tres tipos de sensores en el ojo que tienen una respuesta relativa diferente de acuerdo a la longitud de onda". La representación que imita los sensores humanos es también una forma común de representar las imágenes a color. Esta representación es mediante los canales RGB. Estos representan el rojo (*red*), verde (*green*) y azul (*blue*). Esta representación es similar a la imagen a escala de grises en el sentido que cada canal tiene valores entre 0 y 255. Sin embargo, ahora en vez de ser una representación escalar se tendrá una representación vectorial [R, G, B].

En la Figura 17 se puede visualizar el espectro visible del ojo humano.

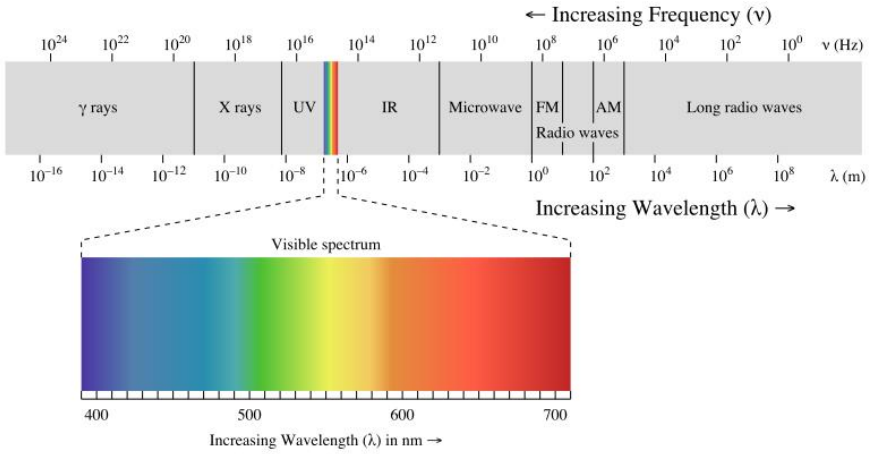


Figura 17. Representación del Espectro Visible
Fuente: Wikimedia Commons (2015). *EM Spectrum*

Dado que cada canal puede tener hasta 256 valores, se necesitan 8 bits de datos para cada canal. Por lo tanto, se tiene una profundidad de 24 bits para cada pixel. Esto quiere decir que en cada pixel es posible tener 16,777,216 colores diferentes (256 x 256 x 256) que van de la mano con lo que percibe el ojo humano (Destruels, 2015).

Para entender mejor los valores se presentan a continuación algunos ejemplos:

- Si todos los canales RGB toman los valores de 255 se tiene el color blanco: [255, 255, 255]
- Si todos toman el valor 0, se tiene el color negro: [0, 0, 0]
- Si solo se le da una magnitud de 255 al canal rojo [255, 0, 0] se tendrá el rojo puro, lo mismo sucederá en el caso del verde y el azul.
- El resto de los colores se genera mediante las combinaciones de los tres canales.

En la Figura 18, se puede observar lo mencionado mediante la representación de un cubo, mientras mayores sean los canales de rojo, verde y azul, más cercano se está al color blanco.

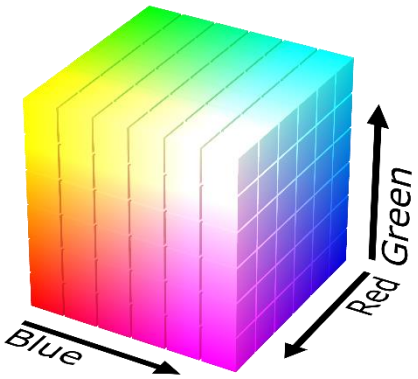
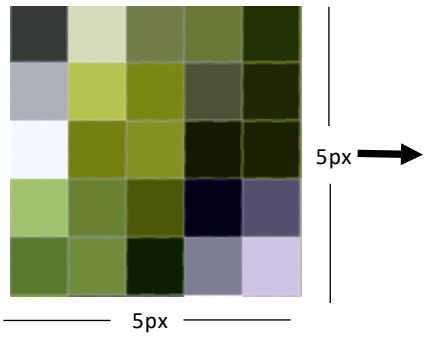


Figura 18. Representación de los colores RGB en un cubo
Fuente: Wikimedia Commons (2015). *RGB color solid cube*

Siguiendo el ejemplo de la imagen de las flores (Figura 15), se puede representar un cuadrado de 5 pixeles de ancho y 5 pixeles de alto mediante una matriz de la siguiente manera:



[49, 55, 51]	[214, 219, 187]	[114, 125, 69]	[104, 121,51]	[33, 46, 0]
[172, 177, 183]	[181, 196, 79]	[121, 136, 19]	[76, 81, 49]	[30, 37, 3]
[245, 248, 255]	[116, 128, 18]	[131, 145, 34]	[22, 25, 0]	[26, 33, 0]
[161, 195, 111]	[106, 129, 47]	[69, 88, 6]	[3, 0, 24]	[83, 76, 109]
[89, 122, 41]	[114, 139, 45]	[13, 29, 0]	[125, 125, 149]	[206, 197, 226]

Figura 19. Representación RGB
Fuente: Elaboración Propia

En la Figura 19 se observan combinaciones de una gran gama de colores. Esto se logra mediante variaciones en las combinaciones de valores de los 3 canales básicos R, G y B. Se

puede observar, que los pixeles más claros tienen valores más cercanos a 255 en cada canal mientras que los pixeles más oscuros tienen valores más cercanos a 0.

2.2.1.1.3. Imagen HSV

Por otro lado, se detallará otra forma de representar las imágenes a color. Esta es la representación HSV (*hue, saturation, value*). El espacio de color HSV es fundamentalmente diferente a la representación RGB dado que separa la intensidad (luminancia) de la información de color de la imagen (cromaticidad) (Shamik, Qian, & Pramanik, 2002). A continuación, se muestra la representación de HSV.

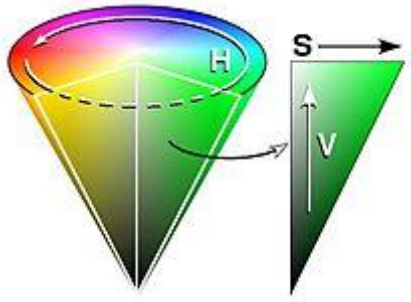


Figura 20. Representación HSV
Fuente: Van den Broek (2005). *Human-centered content-based image retrieval*

Como se observa en la Figura 20, el *hue* o matiz se obtiene como un ángulo de los 360 grados de la base del cono donde cada valor representa un color (rojo, verde, amarillo, entre otros). Asimismo, la saturación es la distancia hacia el eje central, mientras más pequeña sea la saturación, más cercano a las tonalidades de grises estarán los colores. Por lo tanto, con un valor de saturación de 0 se tendrá una imagen equivalente a que estuviese en escala de grises. Finalmente, el valor es la altura del cono que va desde más claro a más oscuro y mientras más cercano a 0 sea, más negro será el color.

2.2.1.1.4. Imagen YCbCr

YCbCr es una representación de imagen generalmente usada para sistemas de video digitales (Jack, 2005). La letra Y representa luma en la imagen, lo cual es una representación de la luminosidad en un sistema de video. Cr y Cb representan la información de color. En los casos de las representaciones RGB con 255 valores por canal, se pueden convertir a YCbCr mediante fórmulas directas por canal (Jack, 2005):

$$Y = 0.257R + 0.504G + 0.098B + 16$$

$$Cb = -0.148R - 0.291G + 0.439B + 128$$

$$Cr = 0.439R - 0.368G - 0.071B + 128$$

Estos son algunas de las representaciones de imágenes a color más conocidas. Como se pudo observar, todas buscan mediante mezclas de valores numéricos representar la imagen utilizando una combinación de canales.

2.2.1.2. Preprocesamiento por Binarización por Umbral

La binarización por umbral es una técnica de preprocesamiento sin mayor complejidad. En este caso se asigna el valor de 0 o 255 a cada pixel dependiendo si su intensidad es mayor o menor que cierto umbral (Sucar & Gómez, 2008). Un ejemplo de esto se observa en la Figura 21:

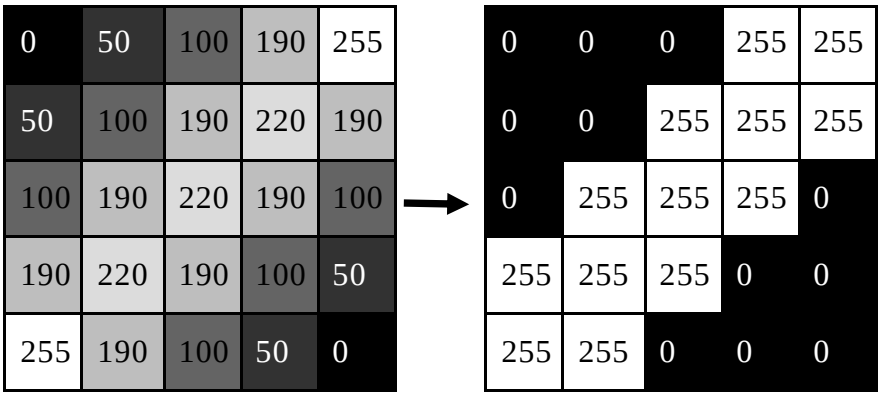


Figura 21. Binarización por umbral 120
Fuente: Elaboración Propia

Como se observa, si se tuviese un umbral de 120. Todos los valores menores a 120 se vuelven negros y todos los mayores se vuelven blancos. Esta técnica puede servir como una forma rudimentaria de separación de un objeto que se pretende encontrar en una imagen. Sin embargo, es común que los objetos sean más complejos de separar dentro de una imagen.

Asimismo, a continuación, se muestra el resultado que se obtiene al aplicar la binarización por umbral en una imagen.

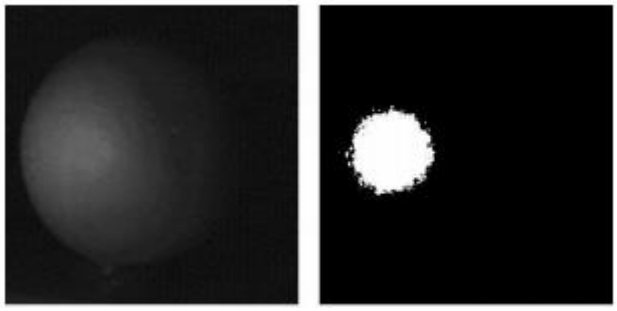


Figura 22. Resultado de binarización por umbral
Fuente: Sucar & Gómez (2008). *Visión Computacional*. (p. 19)

2.2.1.3. Preprocesamiento por Aumento lineal del contraste

En este caso lo que se desea es aumentar la distancia entre las intensidades de los pixeles (Sucar & Gómez, 2008). Para esto lo que se busca es llevar el valor mínimo a 0 y el máximo a 255. Se realiza mediante la siguiente fórmula:

$$C(x,y) = \frac{I(x,y) - min}{\max - min} * 255 \quad (1)$$

En Figura 23 se observa un ejemplo aplicado:

10	50	100
50	100	190
100	190	220

→

0	49	109
49	109	219
109	219	255

Figura 23. Aumento lineal del contraste
Fuente: Elaboración Propia

El valor máximo *max* es 220 y el mínimo *min* es 10. Por lo que se aplica la fórmula a cada pixel dando ese resultado. Por ejemplo en el caso del valor 190 se tendría lo siguiente:

$$C(x,y) = \frac{190 - 10}{220 - 10} * 255 = 219$$

Se realiza lo mismo con el resto de pixeles. Finalmente, se muestra una imagen antes y después de aplicarle el aumento lineal del contraste.

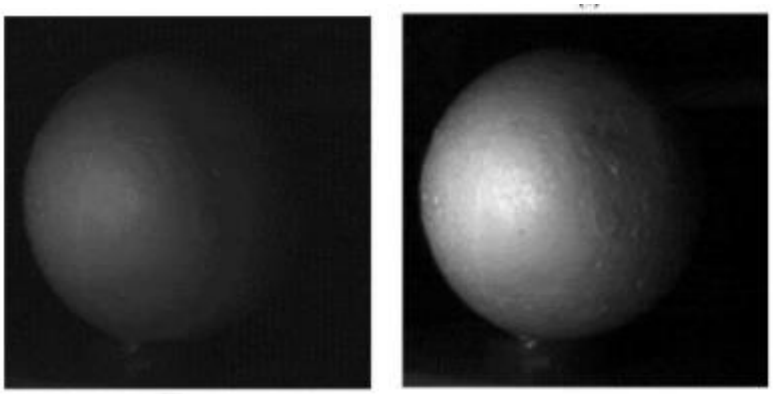


Figura 24. Resultado de aumento lineal de contraste
Fuente: Sucar & Gómez (2008). *Visión Computacional*. (p. 19)

2.2.1.4. Preprocesamiento por Ecualización de histograma de intensidades

En primer lugar, se explicará la definición de un histograma de una imagen. El histograma muestra la distribución de los valores de intensidad de los pixeles de una imagen (Sucar & Gómez, 2008; Caraig, 2017). Por lo tanto, se muestra la distribución de las frecuencias de los valores de la iluminación. Estos pueden ser entre 0 y 255 aunque a veces se agrupan los valores en rangos distintos.

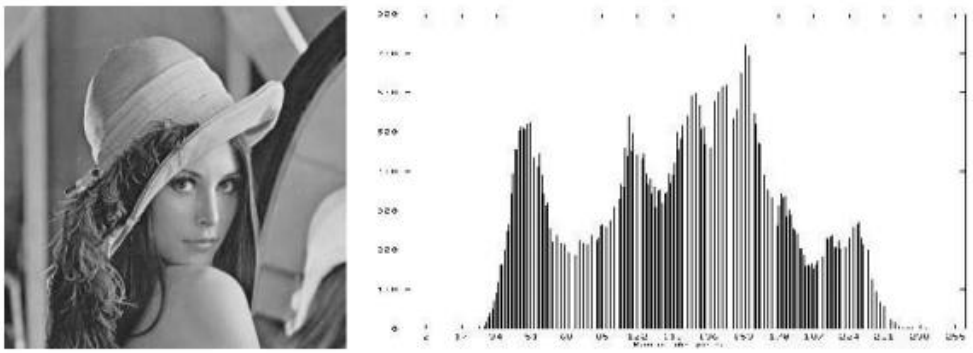


Figura 25. Ejemplo de histograma de una imagen
Fuente: Sucar & Gómez (2008). *Visión Computacional*. (p. 20)

El proceso de ecualización del histograma es un método de preprocesamiento para ajustar el contraste de una imagen dado que histogramas con picos altos agrupados por una sola sección suelen corresponder a imágenes con poco contraste (Krig, 2014). En este caso lo que se desea es uniformizar las frecuencias de los valores de las intensidades dentro de un histograma. En la Tabla 1 se muestra un ejemplo aplicado a los mismos valores de la Figura 23:

Tabla 1.

Cálculo de ecualización del histograma

Valor	10	50	100	190	220
Frecuencia	1	2	3	2	1
Probabilidad	0.111111	0.222222	0.333333	0.222222	0.111111
Probabilidad acumulada	0.111111	0.333333	0.666667	0.888889	1
Probabilidad Acum. * 255	28	85	170	226.6667	255

En la tabla, se observa una de las formas de ecualización. Para esto se calcula la distribución acumulada de probabilidades y se multiplica por el valor máximo para hallar el equivalente en dicha distribución. Por ejemplo, para el valor 10, su probabilidad acumulada es 0.11111. Si se multiplica por 255, se obtiene un valor aproximado de 28.

Finalmente, a continuación, se ilustra el resultado de esta técnica antes (izquierda) y después (derecha):

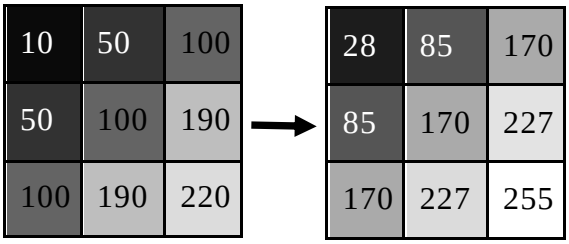


Figura 26. Ecualización del histograma
Fuente: Elaboración Propia

Asimismo, en la Figura 27 se muestra el resultado aplicado a una imagen:

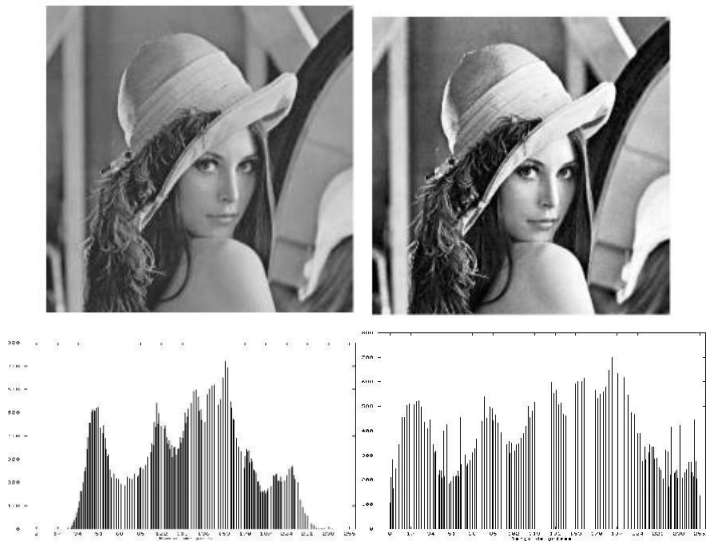


Figura 27. Resultado de aplicar ecualización de histograma
Fuente: Sucar & Gómez (2008). *Visión Computacional*. (p. 22)

2.2.1.4.1. Ecualización de histograma de intensidades a color

La ecualización del histograma también puede aplicarse en imágenes a color utilizando la misma técnica en uno o varios canales independientes. Por ejemplo, en una imagen RGB se puede aplicar a cada canal por separado como se observa en la Figura 28:

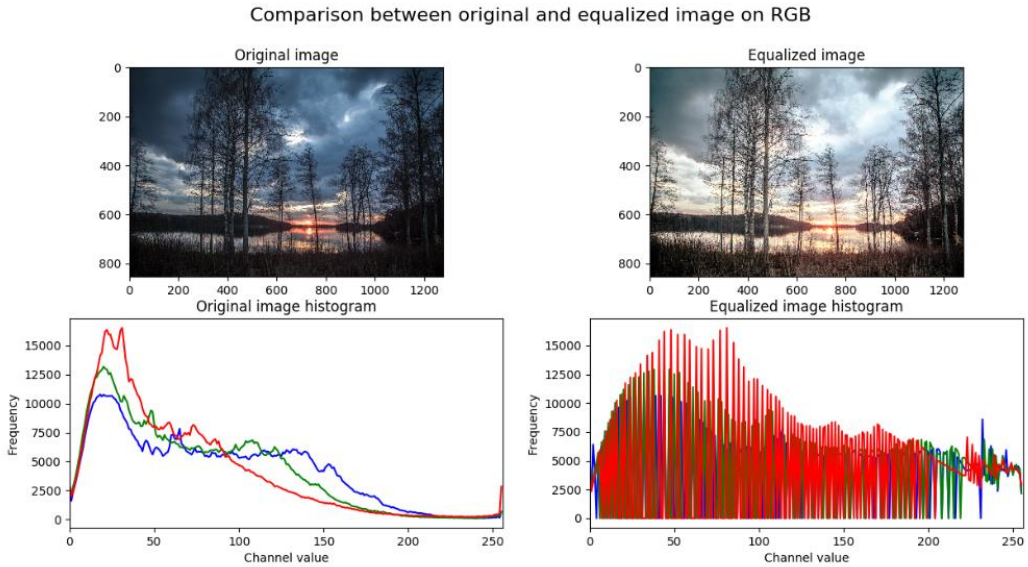


Figura 28. Resultado de aplicar ecualización del histograma RGB
Fuente: Caraig (2017). *Understanding Image Histograms with OpenCV*

Por otro lado, este mismo proceso se puede aplicar a una imagen YCbCr o HSV pero en estos casos se ecualiza sólo el canal de intensidad o luminosidad sin afectar los canales de color (Jeon & Lee, 2013). A continuación, un ejemplo de imagen ecualizada en YCbCr:

Figura 29. Resultado de ecualización de histograma YCbCr.
Sin ecualización (izquierda) vs. Con ecualización (derecha)
Fuente: Elaboración Propia

2.2.1.4.2. Ecualización de histograma local

Una variante de la ecualización de histograma es la ecualización de histograma local. Un problema con la ecualización de histograma es que no puede adaptarse a regiones locales de la imagen (Chang, Chen, Lee, Cheng-Chang, & Han, 2018). Por lo tanto, surgió la ecualización del histograma local o LHE (*Local Histogram Equalization*).

En LHE se subdivide la imagen en pequeños bloques y se realiza la ecualización del histograma de cada bloque por separado (Chang, Chen, Lee, Cheng-Chang, & Han, 2018). De esta forma se obtienen resultados locales de secciones de la imagen. El tamaño de los bloques es algo que se puede variar buscando los mejores resultados.

A continuación, se muestra un ejemplo de ecualización de histograma local:

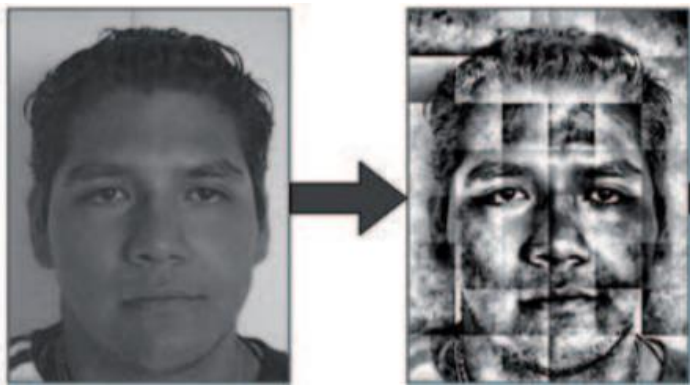


Figura 30. Resultado de aplicar LHE
Fuente: Benitez, Olivares, Aguilar & Sanchez (2012). *Face Identification Based on Contrast Limited Adaptive Histogram Equalization (CLAHE).* (p. 5)

Como se puede observar, al aplicarse por bloques hay ciertas variaciones marcadas entre cada bloque y puede prestarse a maximizar el ruido en bloques que contengan mucho ruido. Una variación de la técnica de ecualización de histograma local es el método CLAHE (*Contrast limiting adaptive histogram equalization*), ecualización de histograma local adaptativa con limitación de contraste. Dicha técnica busca aplicarle un límite al contraste en los bloques generados. Por ejemplo, si existiese una parte del histograma del bloque que tenga un contraste superior a un límite, esos pixeles se distribuyen uniformemente en el histograma evitando variaciones fuertes en dicho contraste. Asimismo, luego de la ecualización, suele aplicarse una interpolación bilineal de los valores de los bordes para remover la marcación de dichos bordes (Chang, Jung, Ke, Song, & Hwang, 2018). Por lo tanto, esta técnica busca minimizar los problemas de contraste y variaciones entre los bloques.

A continuación, se muestra un ejemplo de CLAHE:

Figura 31. Resultado de aplicar CLAHE
Fuente: Benitez, Olivares, Aguilar & Sanchez (2012). *Face Identification Based on Contrast Limited Adaptive Histogram Equalization (CLAHE).* (p. 5)

2.2.1.5. Preprocesamiento por Filtrado de la Imagen

Para la explicación del filtrado se definirá primero el concepto de convolución. Una convolución es una operación discreta espacial que puede ser realizada en procesamiento en una, dos y tres dimensiones en la cual se tiene la imagen y un filtro (Krig, 2014). Una convolución es la aplicación de un filtro en cierta posición de la imagen mediante una multiplicación de sus valores.

$$g(x,y) = \sum_i \sum_j f(i,j)w(i,j) \tag{2}$$

Cabe destacar que el filtro se aplica al pixel central. Por lo tanto, los filtros usualmente tendrán una cantidad impar de filas y columnas. En este caso:

- *f(i, j)* representa los valores actuales del pixel y los pixeles a su alrededor

- *w(i, j)* son los valores del filtro a aplicar
- *g(x,y)* será el nuevo valor del pixel central.

Por definición la convolución implica invertir primero el filtro antes de realizar la multiplicación mientras que la correlación no invierte el filtro. Sin embargo, por lo general los filtros son simétricos, por lo que invertir el filtro no influirá en el resultado. Por este motivo, se obviará la inversión del filtro en el ejemplo que se mostrará.

Al momento de aplicar un filtro, los pixeles que están a los bordes no tienen pixeles a todo su alrededor. En estos casos, se puede ignorar la aplicación del filtro o agregar un borde de un pixel con un valor por defecto antes de realizar la convolución.

2.2.1.5.1. Filtro de Suavizamiento

Un tipo de filtro es el filtro de suavizamiento. Este sirve para reducir el ruido hallando el promedio de los pixeles alrededor. El suavizamiento se puede realizar mediante la media, la mediana o el gaussiano. En el caso de la media con un filtro 3x3, este se realiza mediante la multiplicación por los siguientes pesos:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} * \frac{1}{9}$$

A continuación, se describe un ejemplo de la aplicación de este filtro y se muestra gráficamente en la Figura siguiente:

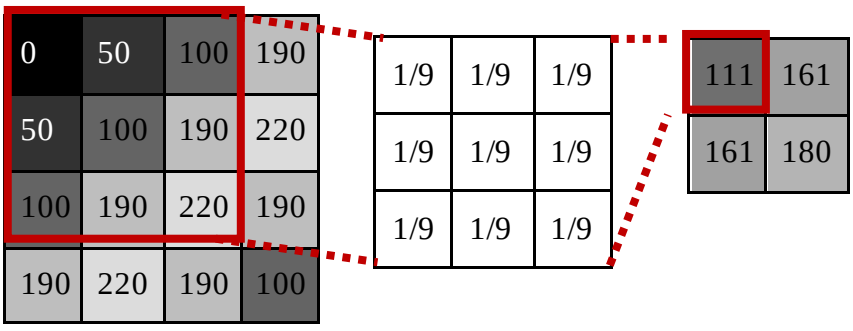


Figura 32. Ejemplo de aplicación de filtro de media
Fuente: Elaboración Propia

Como se puede observar el tamaño de la imagen se reduce de 4x4 a 2x2. Si se desea que no suceda esto, se puede agregar un borde de un pixel alrededor de la imagen como se describió

anteriormente. Para el cálculo del primer valor 111, se aplica el filtro multiplicando cada elemento por su equivalente en la matriz del filtro:

$$0*1/9+50*1/9+100*1/9+50*1/9+100*1/9+190*1/9+100*1/9+190*1/9+220*1/9 = 111$$

Esto equivale a obtener el promedio de esa zona. Para obtener el resto de los valores se debe desplazar el filtro. En este caso el filtro solo puede estar en 4 posiciones distintas por lo que se generan 4 valores de salida.

Lo mismo se puede hacer con el filtro de suavizamiento gaussiano:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 8 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

La Figura siguiente muestra el resultado de aplicación de ambos tipos de filtro:

(a) (b) (c)

Figura 33. Ejemplo del resultado de la aplicación de filtros de suavizamiento:
(a) muestra imagen con ruido. (b) muestra aplicación del filtro de media en (a). (c) ilustra aplicación del filtro gaussiano en (a).
Fuente: Sucar & Gómez (2008). *Visión Computacional*. (p. 25)

2.2.1.5.2. Filtro de Agudizamiento

Otro tipo de filtros es el filtro de agudizamiento. Este filtro tiene la intención de amplificar los cambios en el pixel central en relación con los pixeles adyacentes (Krig, 2014). Este sirve para marcar más los cambios de intensidad en la imagen y acentuarlos. De esta forma se busca hacer más notoria la diferencia entre valores.

Un filtro común que realiza el agudizamiento es el siguiente:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Un ejemplo de su aplicación se visualiza a continuación:

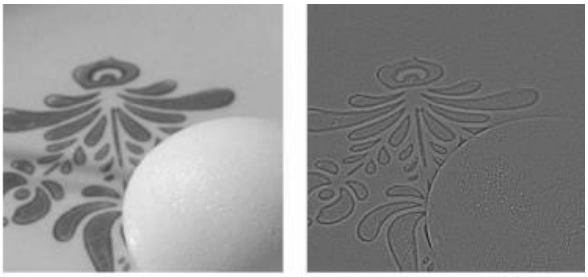


Figura 34. Aplicación del filtro de agudizamiento
Fuente: Sucar & Gómez (2008). *Visión Computacional*. (p. 27)

2.2.1.6. Alineamiento de Rostros

El alineamiento del rostro implica procesar la imagen para poder tener los rostros mirando en dirección frontal. El tener ambas imágenes alineadas mirando en dirección frontal, también puede facilitar la verificación facial dado que ya no se debe entrenar el modelo para aceptar imágenes rotadas, reduciendo tiempos de ejecución y necesidades de procesamiento (Amos, Bartosz, & Satyanaray, 2016). A continuación, se muestra gráficamente el funcionamiento.

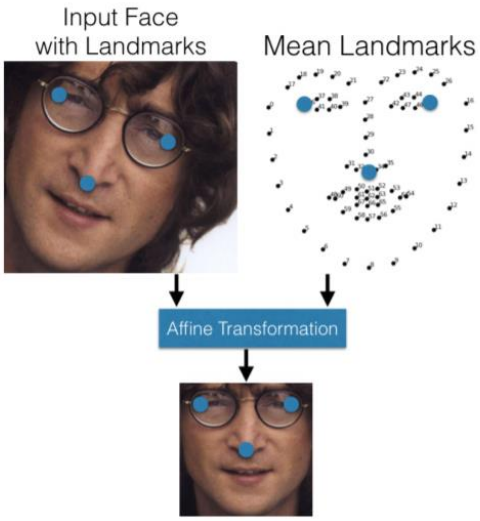


Figura 35. Ejemplo de Alineación de rostros 2D
Fuente: Amos, Bartosz & Satyanaray (2016). *OpenFace: A general-purpose face recognition library with mobile applications*. (p. 6)

Por ejemplo, en el caso de OpenFace se utiliza la alineación afín 2D (Amos, Ludwiczuk, & Satyanarayanan, 2016). Esto se hace buscando que los ojos, nariz y boca aparezcan siempre en posiciones similares. En este caso primero se obtienen 68 puntos de referencia dentro del rostro, se elijen 2 para los ojos y 1 para la nariz y se acerca la imagen a la posición

ideal mediante transformaciones en la escala, rotación o posición de los pixeles en la imagen utilizando una transformación afín.

Para lograr esto se realiza lo siguiente. Primero, se calcula la matriz por la que se debe multiplicar los puntos para que lleguen desde la posición actual a la posición de destino. En este caso, se tienen dos conjuntos de coordenadas de tres puntos (OpenCV, 2012):

- Actual: Se tiene las coordenadas actuales de los tres puntos de ojos y nariz.
- Destino: Se tiene las coordenadas a la que se desea que los puntos de ojos y nariz lleguen.

Por lo tanto, se obtiene *m* para cada pareja de puntos, donde *m* cumple con lo siguiente:

$$\begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = m \cdot \begin{bmatrix} x^i \\ y^i \\ 1 \end{bmatrix} \quad (3)$$

Siendo i = 0,1,2 para los 3 puntos a calcular.

Además, el vector $\begin{bmatrix} x_i' \\ y_i' \end{bmatrix}$ representa las coordenadas de destino para dicho punto y el vector $\begin{bmatrix} x^i \\ y^i \\ 1 \end{bmatrix}$ representa las coordenadas actuales de dicho punto.

Mediante dicha transformación se obtiene una matriz *m* con dimensiones 2 x 3. Esta matriz luego se multiplica por las coordenadas de toda la imagen actual para generar una nueva imagen con el rostro alineado como se muestra a continuación.

Figura 36. Aplicación de Alineamiento Ojos y Nariz
Fuente: Elaboración Propia

Otra alternativa de alineamiento es mediante solamente la rotación de los ojos. Esta ha sido aplicada con éxito anteriormente (Karaaba, Surinta, Schomaker, & Wiering, 2015).

Para el alineamiento de ojos primero se calcula el ángulo en el que se encuentran mediante la siguiente fórmula.

$$ángulo = \arctan\left(\left|\frac{y}{x}\right|\right) \quad (4)$$

Donde:

$$y = ojoIzquierdo_y - ojoDerecho_y$$

$$x = ojoIzquierdo_x - ojoDerecho_x$$

Posteriormente, se rota la imagen del rostro utilizando el ángulo hallado para que los ojos estén en una línea recta. A continuación, se ilustra un ejemplo de este tipo de alineamiento.

Figura 37. Aplicación de Alineamiento solo ojos
Fuente: Karaaba, Surinta, Schomaker & Wiering (2015). *In-plane Rotational Alignment of Faces by Eye and Eye-pair Detection.* (p. 396)

2.2.2. Efectividad de la Verificación Facial

En esta sección se describe la efectividad de la verificación facial. La efectividad de la verificación facial es la que se mide para poder evaluar si las técnicas aplicadas al final del proceso están brindando buenos resultados.

Para poder medir la efectividad, se aplican métricas de clasificación reconocidas. Estas métricas se obtienen luego de procesar el resultado de la verificación facial.

Como se mencionó al inicio de las bases teóricas, la verificación facial ocurre al final, tal como se muestra a continuación resaltado en color naranja:

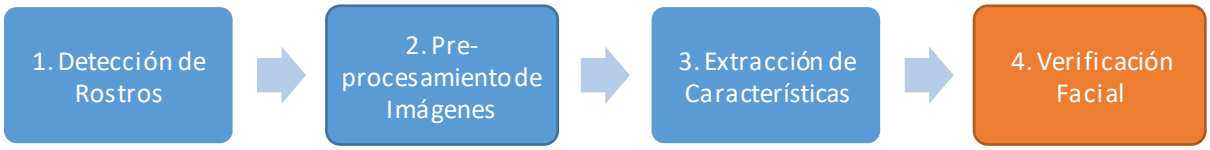


Figura 38. Proceso de Verificación Facial
Fuente: Elaboración Propia

Una vez la imagen es procesada, y se extraen las características de los rostros, estas tienen una representación vectorial por cada rostro. En base a dicha representación se pueden realizar distintas acciones. Este caso se centra en la verificación facial y cómo medirla. Para la verificación facial, se debe comparar las características entre ambos rostros y determinar si son de la misma persona.

La Figura 39 ilustra ejemplo de cómo se genera los vectores de características y luego se realiza dicha comparación.

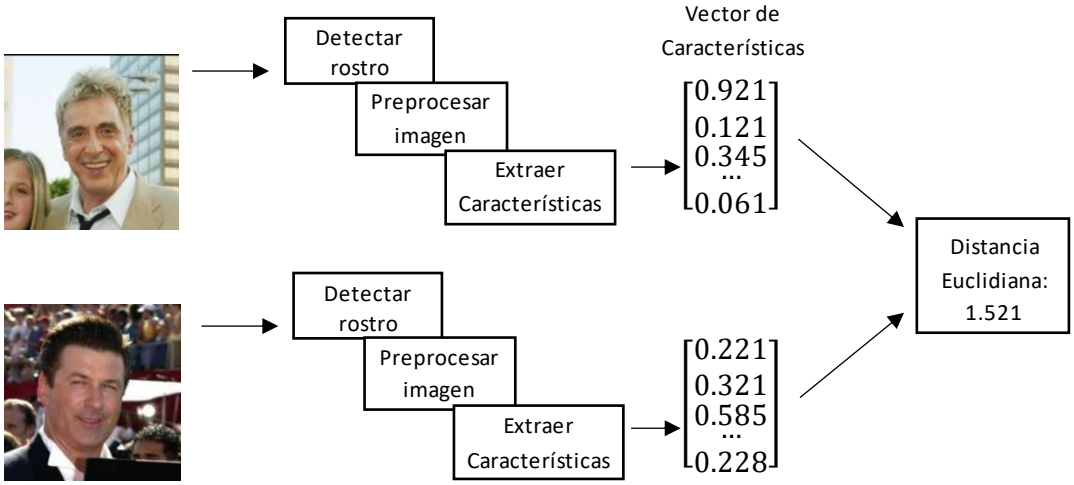


Figura 39. Ejemplo de cálculo de distancia entre dos rostros
Fuente: Elaboración Propia

Para la comparación de dos rostros, una forma de realizar el cálculo de la similitud es mediante la distancia euclidiana entre ambas representaciones de características (Schroff, Kalenichenko, & Philbin, 2015). En el caso *FaceNet*, esta distancia le genera valores entre 0 y 4.

Para determinar si son o no la misma persona, se asigna un punto de corte. Por ejemplo, en el caso de *FaceNet*, en base a varias pruebas, determinaron un punto de corte en 1.242. Este punto de corte establece que, si ambas imágenes tienen una distancia menor a 1.242, se concluye que ambas imágenes son de la misma persona y viceversa. Este punto de corte es variable, en el caso de *OpenFace* (Amos, Bartosz, & Satyanaray, 2016), el punto de corte fue establecido en 0.99. Por lo tanto, a partir de la comparación con el punto de corte definido, el sistema toma una decisión comprobando si es o no la misma persona.

En el ejemplo planteado en la Figura 39, se compara la distancia 1.521 versus el punto de corte definido. Por ejemplo, siguiendo el de *OpenFace*, sería 0.99. Por lo tanto, en este caso, se determina que dado que la distancia es mayor, las imágenes de los rostros son de diferentes personas.

Por lo tanto, una vez que el sistema determina cuales son las imágenes de la misma persona y las de diferentes personas, se puede realizar medidas de la efectividad de los modelos en base a un conjunto de resultados. Las métricas que se describen a continuación son estándar en los antecedentes de la investigación (Amos, Bartosz, & Satyanaray, 2016; Szegedy, et al., 2014; Cao, Shen, Xie, Parkhi, & Zisserman, 2018). Para entender mejor estas métricas de efectividad cabe describir la matriz de confusión.

Tabla 2.

Matriz de Confusión

		PREDICHO	
		Positivo	Negativo
REAL	Positivo	Cantidad de Verdaderos positivos	Cantidad de Falsos Negativos
	Negativo	Cantidad de Falsos Positivos	Cantidad de Verdaderos Negativos

La matriz de confusión de la Tabla 2 muestra la cantidad de resultados clasificados divididos en sus categorías respectivas. Este caso se centra en la clasificación binaria, se hace una comparación de valores reales esperados versus los valores predichos del modelo. Por ejemplo, si se predice 1 pero en verdad es 0, se tiene un falso positivo. Si se predice 0 pero es 1, se tiene un falso negativo. Si se predice 1 y es 1, se tiene un verdadero positivo y si se predice 0 y es 0 se tiene un verdadero negativo.

En el presente caso, las clasificaciones positivas (representadas como 1) hacen referencia a parejas de rostros que son de la misma persona mientras que clasificaciones negativas (representadas como 0) se refieren a parejas de rostros que son de diferentes personas.

Precisión

La precisión evalúa, en caso de tener una predicción positiva, qué porcentaje son verdaderamente positivos.

$$Precisión = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Positivos} \quad (5)$$

Ratio de Verdaderos Positivos

El Ratio de Verdaderos Positivos evalúa, en caso de ser una clase positiva, qué porcentaje se logra clasificar correctamente.

$$Precisión = \frac{Verdaderos\ Positivos}{Verdaderos\ Positivos + Falsos\ Positivos} \quad (6)$$

Ratio de Falsos Positivos

El Ratio de Falsos Positivos evalúa, en caso de ser una clase negativa, qué porcentaje se clasifica incorrectamente.

$$Ratio\ de\ Falsos\ Positivos = \frac{Falsos\ Positivos}{Verdaderos\ Negativos + Falsos\ Positivos} \quad (7)$$

Exactitud *(Accuracy)*

Este representa qué porcentaje del total de la data se clasifica correctamente.

$$Exactitud = \frac{Verdaderos\ Positivos + Verdaderos\ Negativos}{Total} \quad (8)$$

Equal Error Rate (EER)

El Equal Error Rate representa la exactitud en el punto de corte en el que se tiene un ratio de falsos positivos igual al ratio de falsos negativos. La ventaja principal es que el resultado es independiente del punto de corte definido haciendo que modelos diferentes sean comparables.

Curva ROC

La curva ROC muestra el Ratio de Verdaderos Positivos versus el Ratio de Falsos Positivos ante los diferentes puntos de corte pudiendo comparar dichos puntos.

Área Bajo la Curva (AUC)

El área bajo la curva (AUC) se calcula bajo la curva ROC y representa el grado de separabilidad entre las clases, mientras más cercano a 1 mejor. En otras palabras, representa la probabilidad de que una clasificación positiva elegida al azar tenga un mejor puntaje que una muestra negativa elegida al azar, en este caso el mejor puntaje se representa mediante una menor distancia.

$$AUC = P(\, score(x^{+}) > \text{score}(x^{-}) \,) \tag{9}$$

2.2.3. Detección de Rostros

En esta sección se explicará la detección de rostros. Este proceso forma parte del proceso de verificación facial tal como se muestra a continuación:

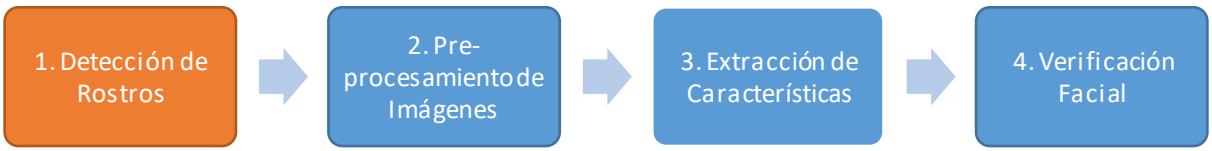


Figura 40. Proceso de Verificación Facial
Fuente: Elaboración Propia

Para explicar la detección de rostros, se empezará describiendo la detección de objetos. Posteriormente se describirán las tres fases principales del proceso de detección de rostros: el preprocesamiento de imágenes, la extracción de características y la clasificación. Cabe destacar que las fases de preprocesamiento de imágenes y extracción de características de la detección de rostros son diferentes a las del proceso de verificación facial.

La detección de objetos implica estimar cuáles son y dónde están ubicados los objetos en una imagen (Zhao, Zheng, Xu, & Wu, 2018). Por ejemplo, en la Figura 41 se observa la detección de objetos mediante un algoritmo que muestra dónde se encuentran los objetos en la imagen y los clasifica por el tipo de objeto detectado.

Figura 41. Ejemplo de detección de Objetos
Fuente: Remanan (2019). *Beginner's Guide to Object Detection Algorithms*

Las metodologías de detección y las de reconocimiento de objetos siguen un patrón similar. En la forma más resumida, el patrón tradicional consta de dos módulos principales: el de extracción de características y el del clasificador (LeCun, Bottou, Bengio, & Haffner, 1998).

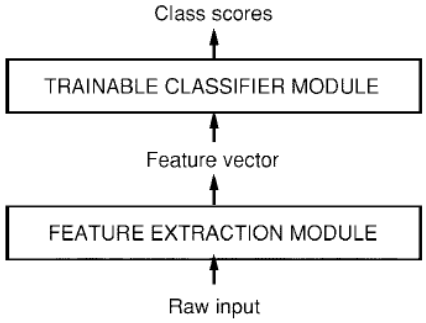


Figura 42. Patrón tradicional de reconocimiento
Fuente: LeCun, Bottou, Bengio & Haffner (1998). *Gradient Based Learning Applied to Document Recognition.* (p. 2)

Tal como se observa en la Figura 42, en primer lugar, se tienen ciertos valores de entrada (*raw input*) que suele ser la matriz de valores de los pixeles de la imagen descrita anteriormente. Estos valores de entrada son procesados por un módulo de extracción de características generando un vector de características (*feature vector*). Este vector de características son datos numéricos que resumen las características de la imagen, es fácilmente comparable con vectores de otras imágenes y es relativamente invariante a transformaciones y distorsiones de los patrones de entrada (LeCun, Bottou, Bengio, & Haffner, 1998). Finalmente, estos vectores numéricos se suelen entrenar mediante un clasificador de forma similar a lo que se hace en los algoritmos de predicción estadística o de aprendizaje automatizado. El clasificador puede ser un modelo de regresión logística, SVM (*Support Vector Machine*), árboles de decisión, AdaBoost, entre otros.

Las combinaciones mediante esta metodología son innumerables en la literatura. Siguiendo la idea básica de transformar las imágenes en vectores comparables que luego puedan ser clasificados mediante un algoritmo de clasificación existente. Por lo tanto, surgieron métodos de extracción de características que luego eran combinados con distintos métodos de clasificación.

A continuación, se muestra un ejemplo de la metodología tradicional en mayor detalle:

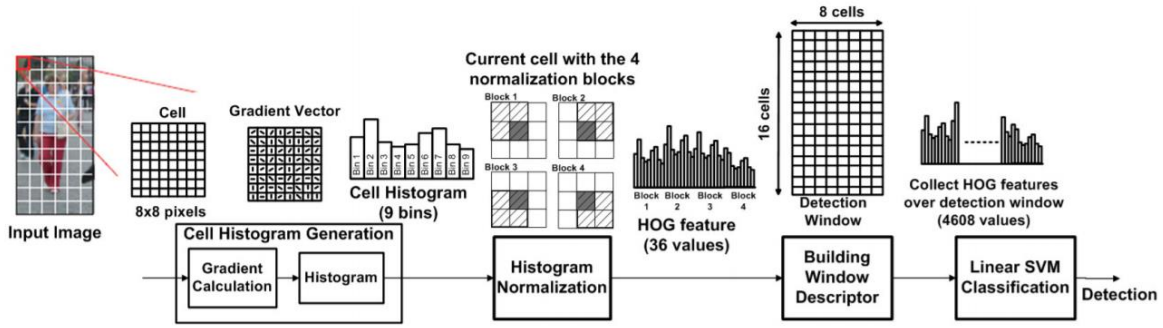


Figura 43. Proceso de detección de objetos mediante descriptor HOG y clasificación SVM
Fuente: Suleiman & Sze (2014). *Energy-Efficient HOG-based Object Detection at 1080HD 60 fps with Multi-Scale Support.* (p. 2)

Este proceso muestra la aplicación en la detección de objetos con un método específico de extracción de características y un modelo específico de clasificación. Se tiene una estructura muy similar a la estructura de la Figura 42 pero expresado aplicando técnicas más al detalle. Se puede observar que se tienen tres pasos de *Gradient Calculation, Histogram Normalization* y *Building Window Desriptor*, los cuales son parte de la extracción de características, la cual es realizada con el descriptor HOG (*Histogram of Oriented Gradients*). Finalmente, en el último paso denominado *Linear SVM Classification* se aplica la clasificación, la elección de clasificación fue mediante un clasificador SVM (*Support Vector Machine*).

En el caso de la detección de rostros, se sigue un proceso similar. A continuación, se muestra otro ejemplo del proceso pero aplicado a detección de rostros:

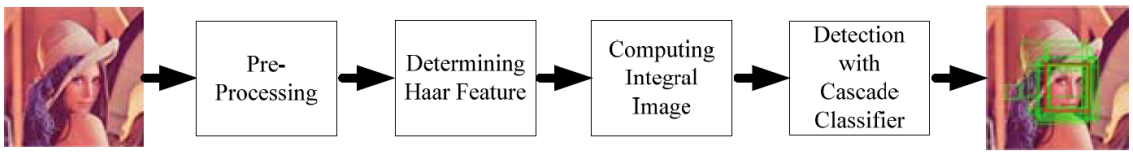


Figura 44. Proceso de detección de rostros basado en características Haar
Fuente: Santoso & Pulungan (2013). *A Parallel Architecture for Multiple-Face Detection Technique Using AdaBoost Algorithm and Haar Cascade.* (p. 593)

Luego de adquirir la imagen, se hace un preprocesamiento de la imagen (*Pre-processing*), la extracción de características (*Haar Feature* y *Computing Integral Image*) y la clasificación (*Detection with Cascade Classifier*) para determinar si hay un rostro o no. Se puede observar que en este caso aplican un preprocesamiento al inicio. Hay casos como este, en los que se considera preprocesar la imagen algo necesario.

A continuación, se explicará con mayor detalle el proceso tradicional de detección de rostros, usando como base los tres pasos principales que se han podido observar: preprocesamiento de imágenes, extracción de características y clasificación.

2.2.3.1. Preprocesamiento de imágenes

El preprocesamiento aplicado en esta etapa de detección de rostros es similar en técnicas a las descritas en la sección 2.2.1. Preprocesamiento de Imágenes.

El preprocesamiento de imágenes es análogo a la normalización matemática de un conjunto de datos (Krig, 2014). Este busca estandarizar las características de las imágenes antes de que se ejecuten los siguientes pasos del sistema. El preprocesamiento de imágenes para la detección facial puede utilizar métodos muy similares a los utilizados para el reconocimiento o verificación facial.

2.2.3.2. Extracción de características

Como se mencionó anteriormente, las técnicas de extracción de características que existen para la detección facial son variadas. Algunas de las más conocidas son mediante descriptores HOG (*Histogram of Oriented Gradients*) y Haar.

"La necesidad de los extractores de características se debía al hecho que las técnicas de aprendizaje usadas por los clasificadores estaban limitadas a espacios de pocas dimensiones con clases fácilmente separables" (LeCun, Bottou, Bengio, & Haffner, 1998). La idea básica de la extracción de características es resumir los valores de los pixeles de la imagen en valores clasificables y poco propensos a fallar por pequeñas variaciones en la imagen. Por ejemplo, si se quisiese clasificar peatones, las imágenes de peatones pueden contener un peatón en distintas ubicaciones de la imagen, pueden tener diversos tipos de iluminación y diferentes tamaños. Por lo tanto, ingresar los valores de los pixeles de cada imagen al clasificador en bruto, no generaría buenos resultados.

Lo que se busca es convertir esos valores de pixeles en vectores de valores representativos de cada imagen y que estos valores sean comparables entre sí (LeCun, Bottou, Bengio, & Haffner, 1998). Es así como los extractores de características deben centrarse no solo en los detalles sino en la imagen como un todo y en la interacción de los distintos pixeles entre sí.

2.2.3.2.1. Descriptor HOG

La Figura 45 muestra el uso de los descriptores HOG en una imagen. Este descriptor se centra en la cuantificación de los cambios en iluminación para poder resumir los valores de los

pixeles. Por lo tanto, funciona cuantificando la magnitud y la orientación de la iluminación por pequeños bloques de la imagen. Estos son luego resumidos mediante valores en histogramas. Finalmente, estos histogramas se convierten en vectores de características para cada imagen. Este es uno de los métodos más conocidos de extracción de características.

Un resumen del proceso de cálculo se describe a continuación (Intel, 2015):

1. Se divide la imagen en pequeñas regiones (celdas) y se calcula el histograma de orientaciones de las gradientes para los pixeles en cada celda.
2. Se agrupan los pixeles dependiendo del ángulo de la gradiente. Por ejemplo, todos los pixeles con orientación entre 0° y 40°, 21° y 60°, 61° y 100° hasta llegar a 360°.
3. Cada pixel debe contribuir con cierta cantidad de gradiente en su grupo dependiendo de qué tan lejos está su ángulo del centro del grupo.
4. Se agrupan las celdas en bloques.
5. Se normalizan los histogramas de cada bloque.
6. Se agrupan los histogramas de todos los bloques, esto genera el vector de características extraído.

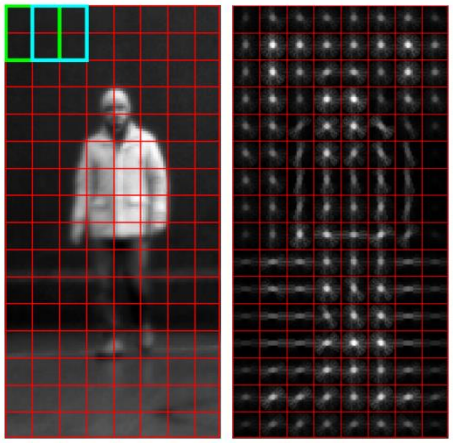


Figura 45. Descriptor HOG
Fuente: Fotiadis, Garzón & Barrientos (2013). *Human Detection from a Mobile Robot Using Fusion of Laser and Vision Information.* (p. 11614)

Los descriptores HOG se hicieron especialmente reconocidos por su utilidad en el reconocimiento de peatones. Incluso, estos fueron extendidos y personalizados en posteriores investigaciones para hacerlos más flexibles a cambios en la proporción de los objetos y a los movimientos de sus partes (Felzenszwalb, Girshick, McAllester, & Ramanan, 2009).

2.2.3.2.2. Descriptor Haar

Por otro lado, otro método de extracción reconocido de detección es mediante el uso de las características Haar. En este caso se tienen filtros definidos que luego son aplicados a la imagen mediante multiplicaciones matriciales o convoluciones. Este método se hizo especialmente reconocido para la detección facial.

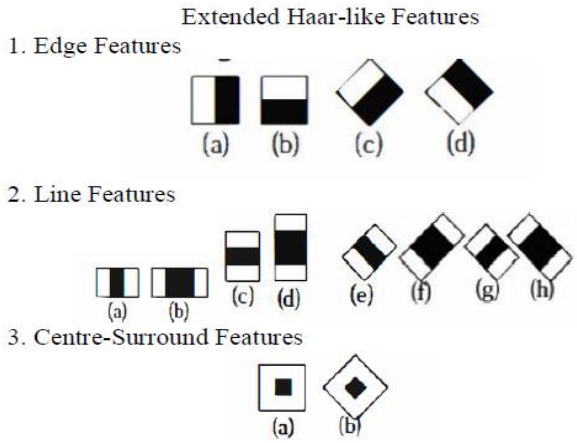


Figura 46. Características de HAAR extendidas
Fuente: Lienhart & Maydt (2002). *An extended set of Haar-like features for rapid object detection.* (p. 901)

Por ejemplo, en la Figura 46 se muestran las características de Haar planteadas por Lienhart y Maydt (2002). Estas características son aplicadas en toda la imagen. Cada filtro de Haar tiene zonas rectangulares negras y blancas. Al aplicar estos filtros en la imagen, los pixeles que se ubiquen en la zona blanca tendrán una contribución negativa mientras que los que se ubiquen en la zona negra tendrán una contribución positiva. Por lo tanto, se suman las intensidades de los pixeles que se encuentran en la parte negra de las características y se restan las intensidades de los pixeles de la parte blanca. Se aplica cada filtro en toda la imagen en varias escalas y en varias posiciones posibles. Al unir los resultados de aplicar los filtros se genera un vector con las características extraídas.

Cabe destacar que las imágenes son transformadas a escalas de grises. Por la naturaleza de estos extractores de características, se trabaja con los valores escalares de las intensidades de cada pixel en vez de trabajar con vectores por pixel.

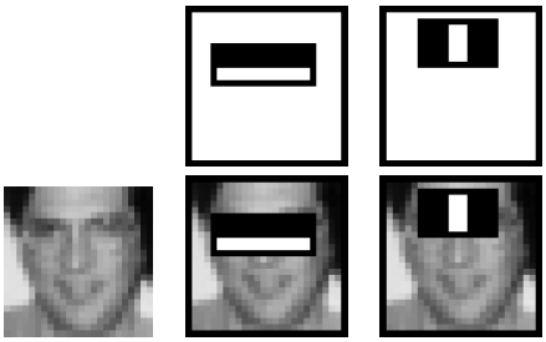


Figura 47. Aplicación de características
Fuente: Viola & Jones (2001). *Rapid Object Detection using a Boosted Cascade of Simple Features.* (p. 4)

La Figura 47 muestra la aplicación de dos filtros en una posición específica. Estos filtros se van desplazando por toda la imagen en iteraciones con varias escalas. A continuación, se mostrará un ejemplo de la aplicación de un filtro de Haar para una imagen de 8 pixeles de ancho y 2 pixeles de alto.

Se usará el filtro:

En la siguiente imagen:

200	100	150	80
100	180	220	220

Se obtiene lo siguiente:

- Escala 1 en 6 posiciones posibles:

200	100	150	80
100	180	220	220

= 200 – 100 = 100

200	**100**	150	80
100	180	220	220

= 100 – 150 = -50

200	100	**150**	80
100	180	220	220

= 150 – 80 = 70

200	100	150	80
100	180	220	220

200	100	150	80
100	**180**	220	220

200	100	150	80
100	180	**220**	220

= 100 – 180 = -80 = 180 – 220 = -40 = 220 – 220 = 0

Se unen estos valores en un vector de la siguiente manera: [100, -50, 70, -80, -40, 0]

- Escala 2 en 2 posiciones posibles:

200	100	150	80
100	180	220	220

= 200 + 100 – 150 - 80 = 30

200	100	150	80
100	180	220	220

= 100 + 180 – 220 – 220 = -160

Se unen estos valores en un vector de la siguiente manera: [30, -160]

- Escala 3 en 3 posiciones posibles:

200	100	150	80
100	180	220	220

= 200 + 100 –100 –180 = 20

200	100	150	80
100	180	220	220

= 100 + 180 –150 –220 = -90

200	100	150	80
100	180	220	220

= 150 + 220 – 80 –220 = 100

Se unen estos valores en un vector de la siguiente manera: [20, -90, 100]

- Escala 4 en 1 posición posible:

200	100	150	80
100	180	220	220

= 200 + 100 + 100 + 180 – 150 – 80 – 220 – 220 = -90

Por lo tanto, en esta escala solo se tiene -90.

Finalmente se unen los resultados de las todas las escalas y posiciones de este filtro dando como vector resultante:

[100, -50, 70, -80, -40, 0, 30, -160, 20, -90, 100, -90]

Este filtro representará los cambios de intensidad de los pixeles de izquierda a derecha. Por lo tanto, tendrá altos valores cuando el contraste en esa dirección sea mayor. Cabe destacar que

este vector de características es sólo el resultado de aplicar un filtro en una imagen pequeña. Los vectores pueden llegar a tener miles de valores al tener imágenes más grandes y aplicar varios filtros en distintas escalas. Hay formas de optimizar el proceso mediante precálculos de las sumatorias en cada zona de la imagen con una técnica llamada imagen integral (Viola & Jones, 2001). Además, existen algunas técnicas más avanzadas para aplicar filtros rotados en ángulos como 45° (Lienhart & Maydt, 2002). Sin embargo, la idea general es la misma que la que se ha planteado en este ejemplo.

2.2.3.3. Clasificación

Tal como se dijo anteriormente, tradicionalmente la clasificación de imágenes se hacía con métodos como la regresión logística, SVM (*Support Vector Machine*) y AdaBoost. En el caso de la detección de rostros, dichos modelos se usan para determinar si es que se encontró un rostro o no en dicha imagen en base a las características obtenidas en el paso anterior. A continuación, se discutirá la técnica de SVM como método de clasificación.

2.2.3.3.1. SVM

SVM (*Support Vector Machine*) o Máquinas de Soporte Vectorial, es una técnica de clasificación. Esta ha sido usada para diversas aplicaciones como reconocimiento de dígitos escritos a mano, reconocimiento de objetos, categorización de textos y detección de rostros (Burges, 1998). En el caso de la presente investigación se usa para detectar si existe un rostro o no en una imagen. Para obtener un resultado, el modelo recibe un conjunto de datos, en este caso las características extraídas de la imagen y determina si existe o no el rostro.

SVMs forman parte de los métodos basados en *kernels* los cuales se basan en las multiplicaciones escalares generando ventajas como su aplicabilidad en métodos no-lineales de clasificación (Ben-Hur & Weston, 2010). Para lograr esto, SVM construye hiperplanos de separación lineal; a continuación, se muestra un ejemplo de esto:

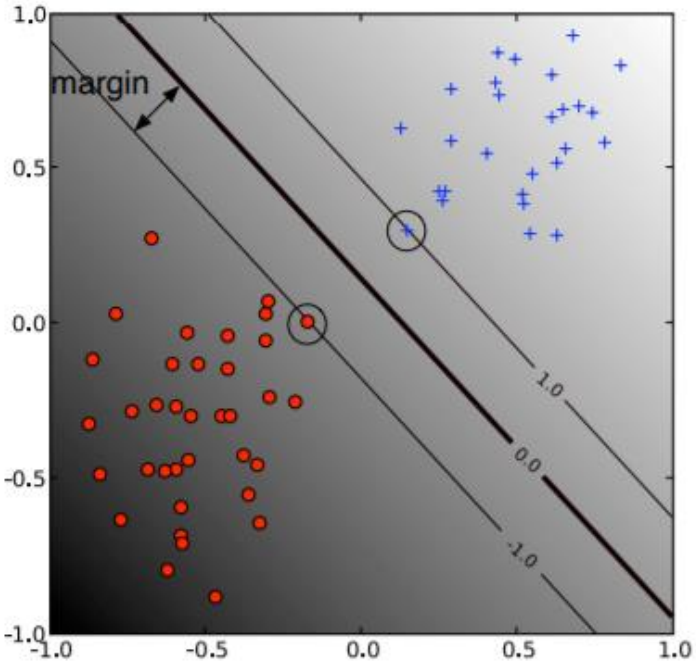


Figura 48. Ejemplo de separación lineal con SVM
Fuente: Ben-hur & Weston (2010). *A User's Guide to Support Vector Machines.* (p. 2)

En el ejemplo de la Figura 48, se tiene data en dos dimensiones, esta data es totalmente separable linealmente generando una clara distinción lineal entre la data azul y la roja. En el caso actual, una clase podría representar que existe un rostro y la otra clase representaría que no existe rostro en la imagen.

Dicha función se entrena utilizando los puntos establecidos en la data. En el caso de las imágenes de rostro, se utiliza características extraídas de imágenes con rostro y sin rostros. Las imágenes con rostro se anotan como 1 y las sin rostro se anotan como -1.

Para representar dicho clasificador, se utiliza la siguiente función para cada conjunto de características x de un rostro (Hearst, 1998):

$$f(x) = sign(x \cdot w + b) \tag{10}$$

Por lo tanto, dicha función, se describe mediante las siguientes funciones:

$$w \cdot x + b > 0, si\ y = 1 \tag{11}$$

$$w \cdot x + b < 0, si\ y = -1 \tag{12}$$

Es decir, las características del rostro x que al ser ingresadas en la función retornen valores mayores a 0 serán consideradas como características de una imagen donde existe un rostro ($y = 1$). En caso el resultado sea menor a 0, serán consideradas como que no existe rostro ($y = -1$).

El objetivo de SVM es construir dichos hiperplanos de separación y maximizar la distancia entre las dos clases, estos son considerados los hiperplanos óptimos (Hearst, 1998). El margen es la distancia más corta entre el hiperplano y el punto más cercano de cada clase (Burges, 1998). Es decir, el margen es la distancia para maximizar representando la distancia perpendicular a la línea entre los puntos más próximos rojo y azul. Como se observa en la Figura 48, se maximiza ese margen entre ambas clases.

Sin embargo, existen casos donde la data no es linealmente separable, es decir se traslapan. En este caso es necesario llevar los puntos hacia un espacio de una dimensión superior donde exista una clara separación entre los grupos (Kandan, 2017). De esta forma se suavizan los márgenes. A continuación, se muestra un ejemplo de cómo la data puede no ser linealmente separable en una dimensión y sí en otra.

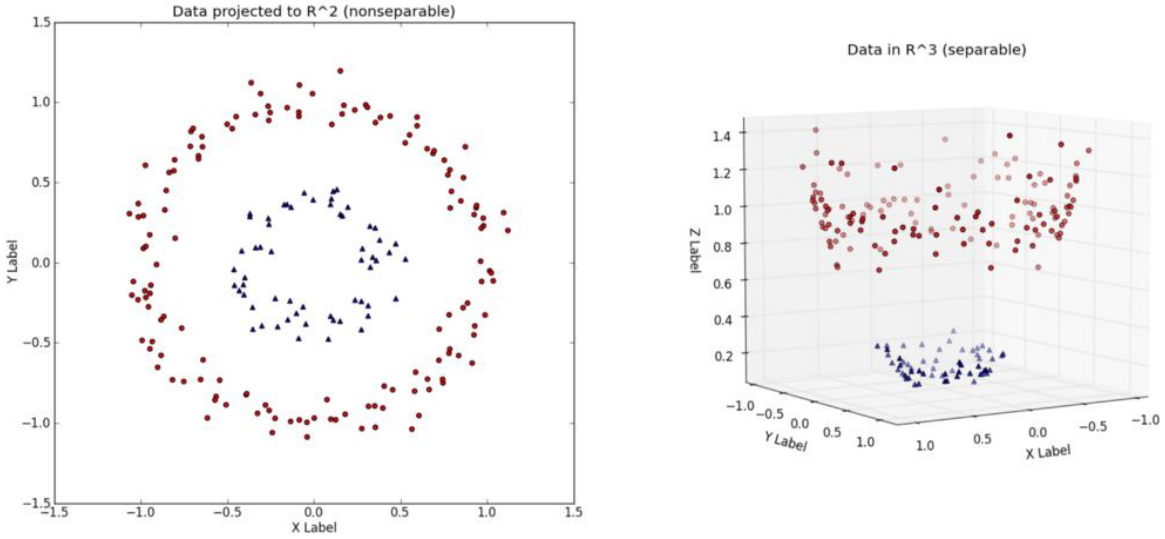


Figura 49. SVM con data en dos y en tres dimensiones
Fuente: Kandan (2017). *Understanding the kernel trick*

En el gráfico de la izquierda de la Figura 49, la data no es linealmente separable en esa dimensión. Sin embargo, si se lleva la data a una dimensión superior, esta puede convertirse en separable.

Se busca aplicar un método lineal, pero en un espacio de características de dimensión superior relacionado de forma no-lineal a los valores de entrada (Hearst, 1998). De tal forma que se pueda lograr esto, se aplican funciones denominadas *kernels*. Estas funciones se pueden usar para reemplazar productos escalares dentro de la función.

Por lo tanto, en forma general un *kernel* sigue la siguiente función (Burges, 1998):

$$K(x_i, x_k) = (\emptyset(x_i) \cdot \emptyset(x_k)) \tag{13}$$

A partir de dicha función, se puede obtener variaciones de *kernels* en varias dimensiones. Un ejemplo es la función de *kernel* en dos dimensiones (Burges, 1998):

$$K(x_i, x_k) = (x_i \cdot x_k)^2 \tag{14}$$

Buscando la relación $K(x, y) = \big(\emptyset(x) \cdot \emptyset(y)\big)$ este *kernel* daría como resultado $\emptyset(x) = (\ x_1^2, \sqrt{2}x_1\ x_2, x_2^2\ )$

Los *kernels* representan funciones de transformación y pueden reemplazar las características. Por lo tanto, es posible mantener un clasificador lineal incorporando dichos *kernels*. Esto daría como resultado la siguiente función para una de las observaciones x (Hearst, 1998):

$$f(x) = sign(b + \sum_{i=1}^{n} v_i\, K(x, x_I)) \tag{15}$$

Esto aplicado por las n observaciones de entrenamiento. Sin embargo, esto solo se debe calcular para las observaciones que representan los vectores de soporte que podría ser un conjunto de observaciones denominado q para ganar eficiencia computacional:

$$f(\vec{x}) = b_0 + \sum_{i \in q} a_i\, K(\vec{x}, \overrightarrow{x_n}) \tag{16}$$

Por lo tanto, modificando los *kernels* se pueden lograr distintos tipos de funciones. Por ejemplo, a continuación, se muestra el efecto de *kernels* de distintas dimensiones:

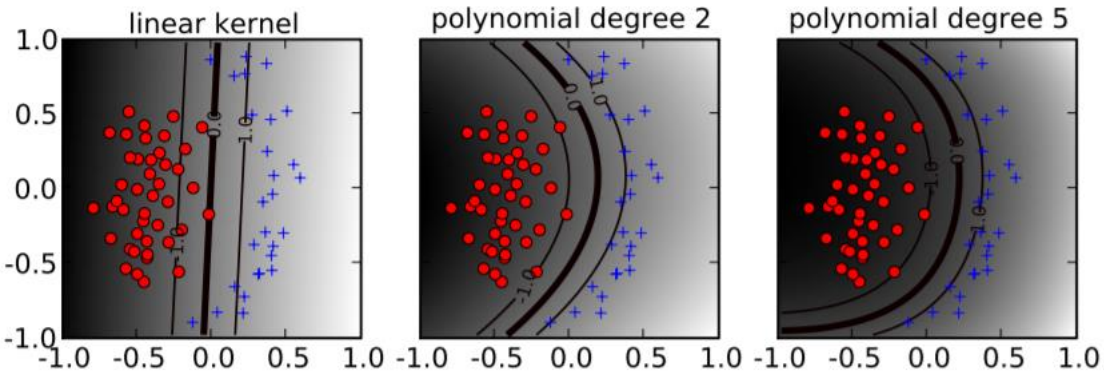


Figura 50. Ejemplos de kernel lineal, de polinomio grado 2 y de polinomio grado 5
Fuente: Ben-hur & Weston (2010). *A User's Guide to Support Vector Machines.* (p. 9)

Asimismo, a continuación, se muestra un ejemplo de *kernel* radial:

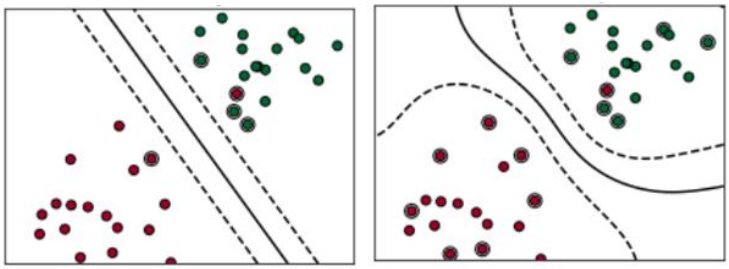


Figura 51. Ejemplo de kernel radial (derecha) vs. kernel lineal (izquierda)
Fuente: Chen *(2019). Support Vector Machine - Simply Explained*

El *kernel* radial tiene la siguiente función (Hearst, 1998):

$$K(x, y) = \exp\left(-\|x - y\|^2/(2\sigma^2)\right) \tag{17}$$

Finalmente, como se pudo observar, las máquinas de soporte vectorial forman parte de las técnicas de clasificación. Esta técnica ha sido útil para distintas aplicaciones entre las cuales se encuentra la clasificación de rostros usándose para determinar si existe o no un rostro a partir de un conjunto de características. Con esta técnica finaliza la sección de detección de rostros. Una vez detectado el rostro se continúa con los siguientes pasos del proceso dentro de la verificación facial.

2.2.4. Extracción de Características con Redes Neuronales Convolucionales

Después de la adquisición de la detección del rostro y el preprocesamiento se procede a realizar la extracción de características para la verificación facial.

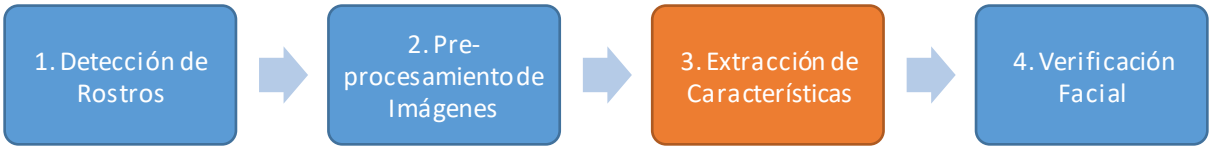


Figura 52. Proceso de Verificación Facial
Fuente: Elaboración Propia

Este es un proceso de extracción de características en el que se procesa solo la zona del rostro ya detectado y recortado. En este caso, se pueden usar también algunos métodos tradicionales. Sin embargo, el método que más importancia ha ganado en los últimos años es el de las redes neuronales convolucionales (Russakovsky, et al., 2015).

Las redes neuronales convolucionales son capaces de extraer características, clasificar a las imágenes o hacer ambas cosas (LeCun, et al., 1989). En consecuencia, el proceso de detección facial también pudo haber sido realizado con estas redes. Sin embargo, tienen la desventaja de necesitar ser entrenadas con gran cantidad de imágenes y por grandes periodos. En este caso, las redes se utilizarán para la etapa de extracción de características de la verificación facial, la cual es de suma importancia dado que a partir de estas características se realizará la comparación.

Por lo tanto, lo que se plantea es que la red neuronal se encargue de extraer las características óptimas. Como se observará esto se hace en forma de filtros y convoluciones. La idea es que la misma red defina qué filtros son los más adecuados para cierto conjunto de datos mediante el aprendizaje de estos mismos.

Se empezará describiendo las redes neuronales clásicas para entender las bases de las redes neuronales convolucionales.

2.2.4.1. Redes Neuronales

Las redes neuronales son un conjunto de unidades de procesamiento interconectadas. Una unidad de procesamiento, algunas veces referida como neurona, recibe ciertos valores de entrada y devuelve algún valor de salida (Stutz, 2014). Estas redes se centran en recibir valores de entrada que al pasar por toda la red y ser procesados por las distintas capas de neuronas, devuelven un vector de valores optimizados hacia un objetivo. Por ejemplo, un algoritmo de riesgo crediticio podría recibir de entrada valores de los ingresos de un cliente, estado civil, gastos, edad, estado laboral, entre otros y devolver las probabilidades que el cliente pertenezca a cada escala de riesgo.

Por lo tanto, las redes neuronales tienen la característica de generar conexiones entre un conjunto de variables de entrada y un conjunto de variables de salida. Estas conexiones suelen ser mediante funciones no-lineales. Una red neuronal está compuesta por un conjunto de perceptrones o unidades de procesamiento

2.2.4.1.1. El Perceptrón

Una unidad de procesamiento o perceptrón se muestra a continuación (Haykin, 2009):

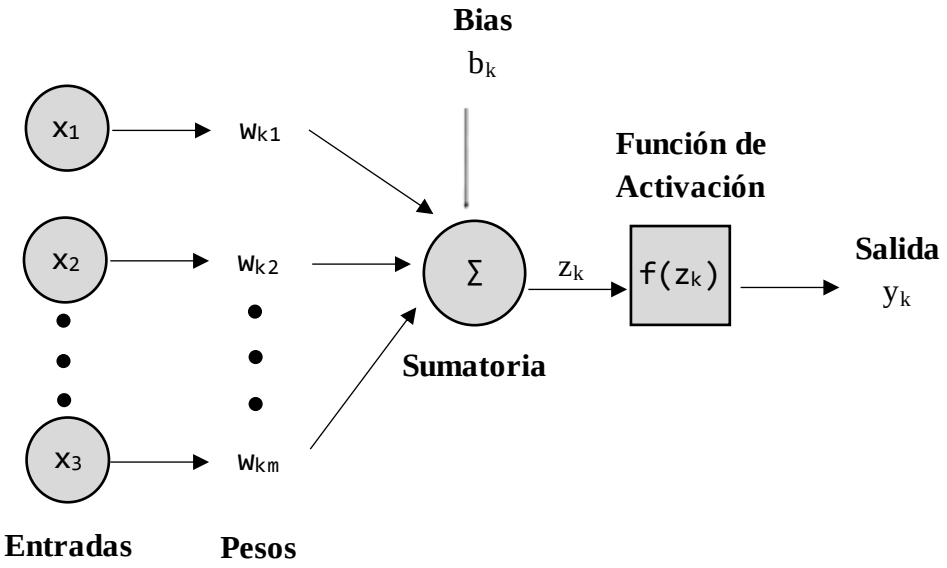


Figura 53. Unidad de procesamiento de red neuronal
Fuente: Haykin (2009). *Neural Networks and Learning Machines*

Como se puede observar en la Figura 53, una unidad de procesamiento tiene ciertos valores de entrada (*x*) y ciertos pesos (*w*) en las conexiones que luego son procesados mediante la multiplicación y sumatoria entre ambos. El resultado de la sumatoria luego se procesa mediante una función de activación generando el resultado final del perceptrón.

Esto se traduce en la siguiente fórmula:

$$z_k = \sum_{j=1}^{m} (w_{kj}\, x_j) + b_k \tag{18}$$

Donde:

- m representa el total de pesos
- w_{kj} representa el peso de la neurona k proveniente de la entrada j
- x_j representa el valor de la entrada j
- b representa el bias. Este tiene el efecto de incrementar o reducir los valores de la sumatoria antes de pasar por la función de activación.

Adicionalmente se puede asumir al bias como parte de los pesos. Para esto se puede tomar en consideración lo siguiente (Haykin, 2009):

$$b_k = w_{k0}$$
$$x_0 = 1 \tag{19}$$

Esto permite simplificar la ecuación de la siguiente manera:

$$z_k = \sum_{j=0}^{m} (w_{kj}\, x_j) \tag{20}$$

A la salida del perceptrón se le aplica una función de activación entre cada capa generando y_k. Por lo tanto, se calcula y_k de la siguiente manera:

$$y_k = \mathrm{f}(z_k) \tag{21}$$

Como se mencionó, lo planteado anteriormente también se puede representar incluyendo el *bias* dentro del vector de los pesos y agregando una neurona de entrada con el valor de 1. Esto se observa en la siguiente representación:

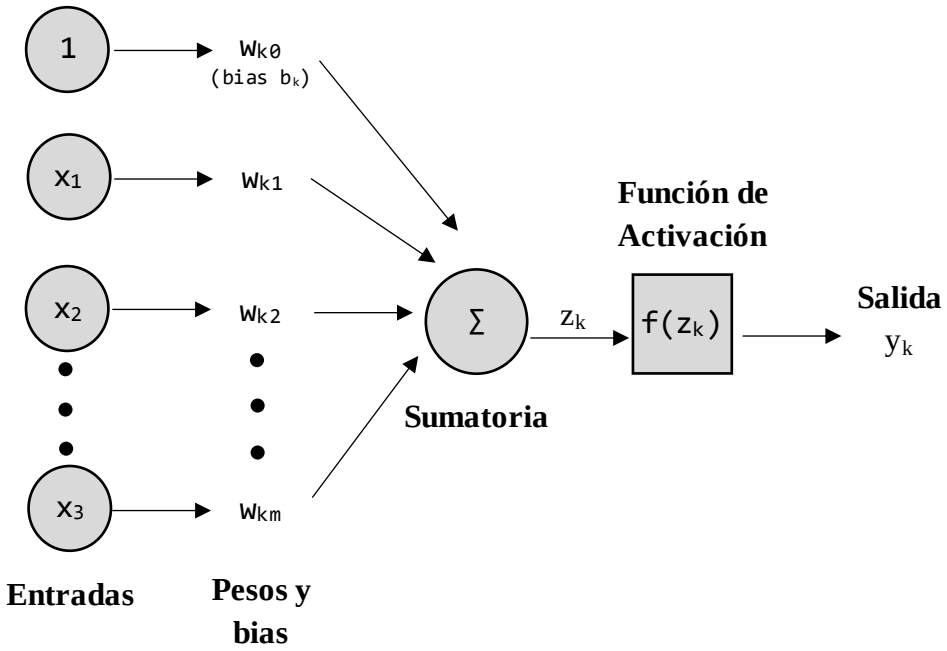


Figura 54. Unidad de procesamiento de red neuronal con bias incluido en los pesos
Fuente: Haykin (2009). *Neural Networks and Learning Machines*

Esto también se puede representar de forma vectorial siendo *x* un vector con los valores de entrada y *w* una matriz con los pesos para cada neurona:

$$z = w \quad x \tag{22}$$

$$\text{Siendo } \mathrm{y} = \mathrm{f}(\mathrm{z})$$

2.2.4.1.2. Función de Activación

Existen diversas funciones de activación que se pueden usar. Entre las más conocidas se encuentran las siguientes (Karpathy, Johnson, & Fei-Fei, 2016):

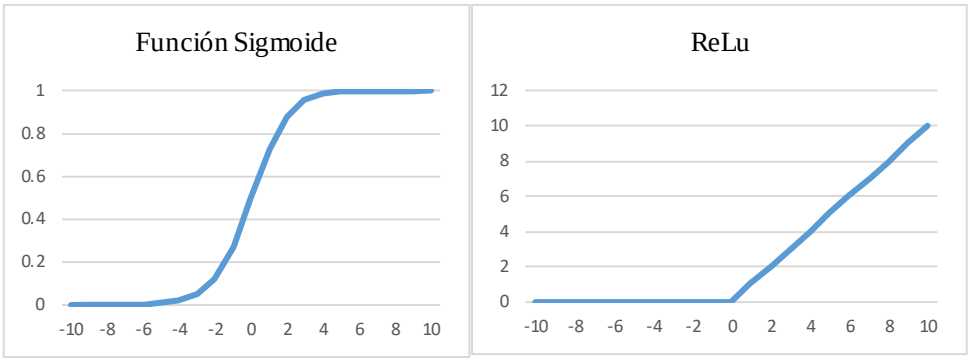


Figura 55. Funciones de activación
Fuente: Elaboración Propia

- Función sigmoide: Comúnmente usada en la regresión logística. Tiene la ventaja de devolver números entre 0 y 1. La desventaja principal es que, al limitar los valores, estos se pueden estancar muy cerca a los bordes de la función (cerca a 1 y cerca a 0).

$$f(z) = \frac{1}{1 + e^{-z}} \tag{23}$$

- Función ReLU (*Rectified Linear Units*): Tiene la ventaja que los valores tienen libertad para crecer. La principal desventaja es que, si surgen valores menores a 0, estos pesos mueren al pasar a la siguiente neurona.

$$f(z) = \max(0, z) \tag{24}$$

2.2.4.1.3. Perceptrón Multi-capa (MLP)

Si se juntan varias unidades de procesamiento y se estructuran en capas se obtiene una estructura multicapa de perceptrones o red neuronal como la Figura 56. Cabe destacar que por lo general las redes suelen tener más capas y neuronas.

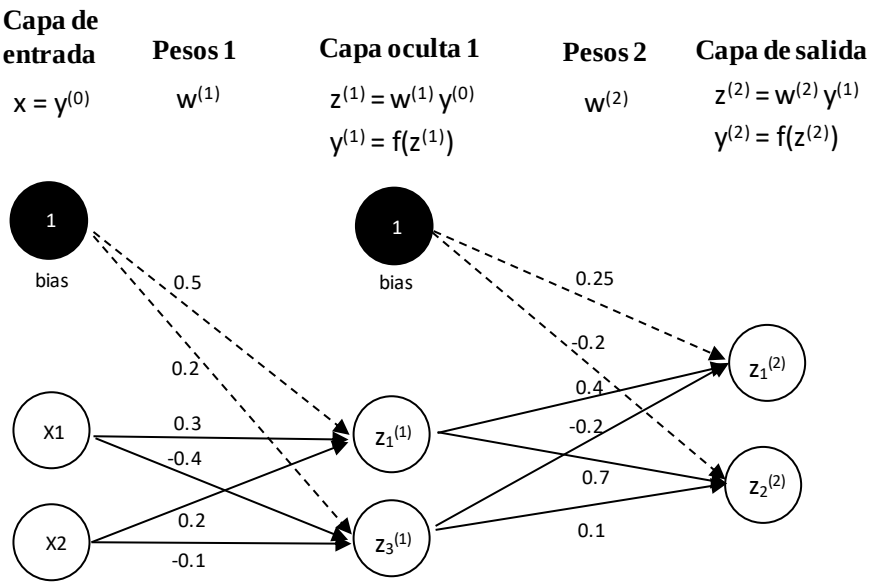


Figura 56. Red neuronal
Fuente: Elaboración Propia

Una red neuronal consiste en L capas ocultas, una capa de entrada y una capa de salida (Stutz, 2014). Cada conexión entre capas tiene asignado un peso. Los valores óptimos de estos pesos (*w*) son los que la red neuronal intentará hallar durante su entrenamiento. En un inicio estos pesos se inicializan al azar o con algún método de inicialización con características aleatorias. La salida de una capa oculta es la entrada de la siguiente capa.

En el caso de un modelo de clasificación, la capa de salida suele tener una neurona por cada clase y la salida de cada neurona en dicha capa representa la probabilidad de que cierto vector de valores de entrada pertenezca a cada una de las clases.

El entrenamiento de una red neuronal para clasificación implica ingresar a la red una gran cantidad de datos clasificados. Es decir, un conjunto de vectores de *x* con sus respectivas clases. Luego, la red inicializa los pesos (*w*) mediante un método aleatorio e intenta optimizarlos para hallar los valores que clasifican correctamente la mayor cantidad de estos vectores de entrenamiento.

Este proceso se divide en dos fases importantes, la de iteración hacia adelante (*forward propagation*) y la de iteración hacia atrás (*backpropagation*):

- La iteración hacia adelante consiste en encontrar el resultado de la clasificación de los vectores de entrenamiento con los pesos actuales.
- La iteración hacia atrás consiste en calcular las gradientes en los pesos y *bias* para poder ajustar y actualizar los pesos luego de hallar el error de acuerdo con el resultado obtenido.

Por lo tanto, este es un proceso iterativo que repite ambas fases y se actualizan los pesos y *bias* hasta que estos encuentran un valor óptimo o llegar a un límite de iteraciones.

A continuación, se empezará explicando la iteración hacia adelante. Para esto, se realiza el cálculo de los valores de salida. Siguiendo lo planteado anteriormente, el cálculo de salida de cada unidad de procesamiento se puede dar mediante la fórmula siguiente (Stutz, 2014):

$$\mathrm{y} = \mathrm{f(z)} \text{ donde } z^{(l)} = w^{(l)}\, y^{(l-1)} \tag{25}$$

Siendo *l* el número de capa.

En este caso las variables son vectores o matrices. Como se observa, en cada neurona primero se calcula la suma de los pesos multiplicados por su respectivo valor de entrada. Luego se aplica la función de activación *f(z)* al resultado para obtener el valor de salida. Se aplican las operaciones en orden en cada capa a lo largo de la red hasta llegar a la última capa obteniendo los valores de salida donde finaliza la iteración hacia adelante. En este caso el *bias* se incluye dentro del vector de pesos como se observó anteriormente.

Por otro lado, en el caso de la iteración hacia atrás, lo que se intenta es calcular la gradiente del error respecto a los pesos para luego actualizar estos pesos minimizando el error de clasificación del algoritmo. El error de clasificación se halla a partir de los valores de salida obtenidos en la iteración hacia adelante. En este caso también existe más de una representación del error que se puede utilizar. Una de las más comunes para clasificación multi-clase es la medida de distancia euclidiana. Esta se representa con la siguiente fórmula:

$$E_k = \| d_k - y_k \| \tag{26}$$

Por lo tanto (Valenzuela, 2015):

$$E_k = \frac{1}{2} \sum_j (d_{k,j} - y_{k,j})^2 \tag{27}$$

Donde:

- *k* representa el número del vector de entrenamiento
- *j* representa el número de la clase
- $y_{k,j}$ el valor de salida o probabilidad que devuelve la red neuronal de que el vector x_k pertenezca a la clase j.
- $d_{k,j}$ el valor real que define si el vector x_k pertenece o no a la clase *j*.

Además, el error de toda la red se representa mediante la fórmula:

$$\varepsilon = \frac{1}{L}\sum_{k} E_k \tag{28}$$

Donde:

- L representa el número de vectores de entrenamiento

Recalcando lo dicho anteriormente, lo que busca la red neuronal es hallar los pesos que minimizan el error. La derivada del error con respecto a los pesos debe ser lo más cercano a 0. La forma más común de resolver esto es mediante la propagación hacia atrás. Esta parte de la idea que la gradiente o derivada se puede propagar en cadena de adelante hacia atrás. Para hacer esta propagación uno de los algoritmos a utilizar es el descenso de gradiente. El descenso de gradiente es un algoritmo iterativo que, en cada iteración, calcula la gradiente en cada dirección y luego se actualizan los pesos de cada capa de acuerdo a qué tan fuerte es la gradiente en esa dirección. Para esto se parte de la teoría de la derivación en cadena (Valenzuela, 2015).

Derivada en la última capa

En el caso de la última capa o capa de salida, esto se traduce en el cálculo de la gradiente del error mediante la siguiente derivada en cadena:

$$\frac{\partial E}{\partial z_j^l} = \frac{\partial E}{\partial y_j^l}\frac{\partial y_j^l}{\partial z_j^l} = \delta_j^l \tag{29}$$

Donde:

- E representa el error
- y_j^l representa la salida de la activación de la neurona j en la capa l
- z_j^l representa la salida en la neurona j de la capa l previa activación
- δ_j^l representa el grado de cambio del error en relación a la activación de la neurona j en la capa l.

Estas denominaciones se mantendrán para las siguientes fórmulas.

Por lo tanto, el error de la última capa es el siguiente (Nielsen, 2015):

$$\delta_j^L = \frac{\partial E}{\partial y_j^L} f'(z_j^L) \tag{30}$$

Donde:

- L representa la última capa.

Asimismo, si la función del error es el euclidiano, la derivada del error respecto a y es la siguiente:

$$E = \frac{1}{2}\sum_j (\, d_j - y_j^L)^2$$

$$\frac{\partial E}{\partial y_j^L} = (y_j^L - \, d_j \,) \tag{31}$$

Entonces, se obtiene que el error en la última capa se da con la siguiente fórmula:

$$\boxed{\delta_j^L = (y_j^L - \, d_j \,)\, f'(z_j^L)} \tag{32}$$

Además, si se usa la función de activación sigmoide, se tiene que:

$$f'(z_j^L) = \, f(z_j^L)\,(1 - f \quad (z_j^L)) \tag{33}$$

$$f'(z_j^L) = \, y_j^L (1 - y_j^L)$$

Reemplazando, el error en la última capa se da por la siguiente fórmula:

$$\delta_j^L = (y_j^L - \, d_j \,)\, y_j^L (1 - y_j^L) \tag{34}$$

<u>Derivada en las capas ocultas:</u>

En el caso de las capas ocultas, se debe tomar en cuenta la variación de la capa siguiente hacia la capa anterior. Por lo tanto, se tiene lo siguiente (Nielsen, 2015):

$$\delta_j^l = \frac{\partial E}{\partial z_j^l} \tag{35}$$

$$\delta_j^l = \sum_k \frac{\partial E}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l}$$

$$\delta_j^l = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \, \delta_k^{l+1}$$

Donde los valores z_k^{l+1}, z_j^l y δ_k^{l+1} siguen la misma denominación que en las fórmulas anteriores.

Dado que:

$$\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} f'(z_j^l) \tag{36}$$

Donde:

- w_{kj}^{l+1} representa el peso entre la neurona j y la neurona k de la capa $l+1$

Se obtiene lo siguiente:

$$\boxed{\delta_j^l = \sum_k \left(w_{kj}^{l+1} \delta_k^{l+1}\right) f'(z_j^l)} \tag{37}$$

Derivada respecto al bias:

En el caso del cálculo de la gradiente para el *bias*, se sigue la regla de cadena en base a lo calculado en las capas ocultas (Nielsen, 2015).

$$\boxed{\frac{\partial E}{\partial b_j^l} = \delta_j^l} \tag{38}$$

Derivada respecto al peso:

De igual forma para los pesos, se sigue la regla de cadena en base a lo calculado en las capas ocultas (Nielsen, 2015).

$$\frac{\partial E}{\partial w_j^l} = \delta_j^l \frac{\partial z_j^l}{\partial w} \tag{39}$$

$$\boxed{\frac{\partial E}{\partial w_{kj}^l} = y_k^{l-1} \delta_j^l}$$

Esto servirá para el cálculo adecuado de la gradiente. Asimismo, se actualizarán los pesos y *bias* en cada capa dependiendo de qué tan empinada se encuentre la gradiente en cada dirección mediante la regla de aprendizaje (Valenzuela, 2015):

$$\Delta w_{kj}^l = \eta\, y_k^{l-1} \delta_j^l \tag{40}$$

$$\Delta b_j^l = \eta\, \delta_j^l \tag{41}$$

En este caso, el parámetro η representa el ratio de aprendizaje que tendrá el algoritmo. Generalmente es entre 0 y 1. Si se tiene un ratio muy alto, los cambios en *w* serán más bruscos pero se corre el riesgo de que el algoritmo pase por alto los valores óptimos de *w*. Si se tiene un ratio muy bajo, habrá más iteraciones para actualizar *w* pero puede que el algoritmo se demore mucho más en encontrar los valores óptimos de *w*.

En resumen, a continuación se describe el proceso:

- Se inicializan los pesos w de forma aleatoria o mediante un método de inicialización
- Se hace una iteración hacia adelante tal como se mostró en el ejemplo anterior a través de la red.
- Se calcula la gradiente del error respecto a z en la capa de salida. En caso sea función del error euclidiano, esta se da mediante la Ecuación 32:
  - $\delta_j^L = (y_j^L - d_j)\, f'(z_j^L)$ (32)
- Se calculan las gradientes de los errores de las capas intermedias a partir del error anterior. Por lo tanto, se calculan las derivadas de derecha a izquierda. El error de cada capa depende del error de la capa siguiente. Para esto se utiliza Ecuación 37:

$$\delta_j^l = \sum_k \left(w_{kj}^{l+1}\delta_k^{l+1}\right) f'(z_j^l) \quad (37)$$

- Se calculan el nivel de aprendizaje para los pesos y el *bias* mediante las Ecuaciones 41 y 42:

$$\Delta w_{kj}^l = \eta\, y_k^{l-1}\delta_j^l \quad (40)$$

$$\Delta b_j^l = \eta\, \delta_j^l \quad (41)$$

- Se actualizan los pesos restándole las gradientes mediante las siguientes fórmulas:

$$w_{jk}^l = w_{jk}^l - \Delta w_{kj}^l \quad (42)$$

$$b_j^l = b_j^l - \Delta b_j^l \quad (43)$$

  Tomando en cuenta un factor de aprendizaje η.
- Finalmente, este proceso se realiza durante muchas iteraciones hasta que deje de haber mayor variación en los pesos, se considere que se ha llegado al óptimo resultado o se agoten la cantidad de iteraciones.

Se ha descrito el esencial funcionamiento de las redes neuronales. Existen otros conceptos que se pueden agregar como otras funciones de activación, otras medidas de error, o agregar

parámetros para prevenir el sobreajuste. Sin embargo, siempre se sigue planteamientos similares a lo mostrado.

2.2.4.2. Redes Neuronales Convolucionales

Las redes neuronales convolucionales son una extensión de las redes neuronales clásicas, pero con más dimensiones al recibir valores matriciales de imágenes en más de un canal. Asimismo, estas redes tienen varias características que las diferencian entre las que resaltan el compartir pesos entre neuronas y el uso de pesos matriciales por cada neurona (Karpathy, Johnson, & Fei-Fei, 2016).

Al igual que en una red neuronal tradicional, el propósito es encontrar los pesos óptimos que generen la clasificación adecuada. Su optimización también se puede hacer usando la técnica de propagación hacia atrás vista anteriormente y el descenso de gradiente.

Según LeCun (1989), las metodologías tradicionales demostraron que es muy importante extraer características locales en vez de centrarse en los pixeles de manera específica en una imagen. Muchos objetos podían aparecer distorsionados o en distintas posiciones haciendo que sea necesario extraer características generales que describan la imagen en su conjunto o por áreas. Por esta razón, los modelos se debían centrar en evaluar regiones de la imagen para así detectar características en diversos tamaños y posiciones. Este comportamiento puede ser replicado en una red neuronal forzando a las capas ocultas a combinar fuentes de información local de la imagen. Por lo tanto, distintas características especiales pueden aparecer en distintos lugares de la imagen y ser detectados de igual manera. Esta es la idea que pretenden explotar las redes neuronales convolucionales.

A continuación, se empezará describiendo la arquitectura básica de este tipo de redes neuronales.

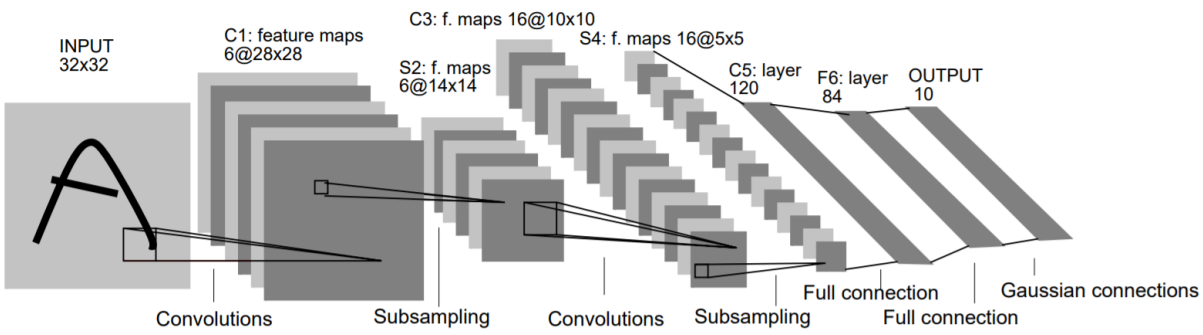


Figura 57. Red neuronal convolucional para reconocimiento de dígitos LeNet-5
Fuente: LeCun, Bottou, Bengio & Haffner (1998). *Gradient Based Learning Applied to Document Recognition* (p. 7)

La Figura 57 muestra un ejemplo de arquitectura de una red neuronal convolucional. Como se puede observar existen algunas capas importantes: capa convolucional (*convolutional layer*), capa de submuestreo (s*ubsampling*) y capa totalmente conectada (*fully connected layer*). Cada una tiene un propósito que se explicará en las siguientes subsecciones.

Debajo de la arquitectura general, los conceptos básicos son bien similares a los de una red neuronal clásica. Sigue existiendo una capa de entrada y una capa de salida con las clases. Asimismo, siguen existiendo neuronas conectadas con pesos y se utilizarán funciones similares de activación. Además, de igual manera existe un algoritmo de optimización para minimizar los valores de los pesos. A continuación, se explicarán las capas principales.

2.2.4.2.1. Capa de convolución

En la sección de preprocesamiento se describió la aplicación de filtros mediante convoluciones con la Ecuación 2. Esta se vuelve a mostrar a continuación:

$$g(x,y) = \sum_i \sum_j f(i,j)w(i,j) \tag{2}$$

Algo similar es lo que se realiza en las redes neuronales convolucionales con el propósito de hallar los pesos *w*. Por lo tanto, el propósito de la red neuronal convolucional será el de aprender los filtros *w* que den el resultado óptimo en la clasificación de las imágenes.

Cada capa de convolución tendrá una cantidad determinada de filtros que deberán ser aprendidos. Cada filtro se aplicará a toda la imagen, esto representa la característica de pesos compartidos (Karpathy, Johnson, & Fei-Fei, 2016). Esto permite que las características que encuentre un filtro puedan ser encontradas alrededor de toda la imagen y evita la redundancia entre filtros diferentes.

Durante la iteración del pase hacia adelante, de forma similar a una red neuronal, se multiplican los pesos por los valores de entrada. En este caso los valores de entrada serán multiplicados por cada filtro alrededor de toda la imagen, generando un volumen de salida que luego deberá ser activado por la función de activación que se decida.

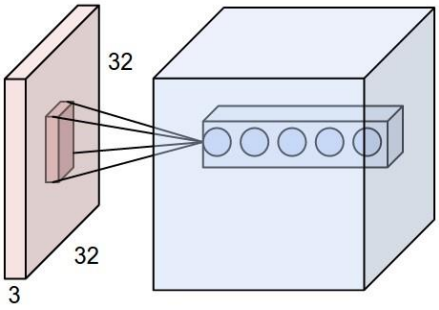


Figura 58. Capa de convolución
Fuente: Karpathy, Johnson & Fei-Fei (2016). *CS231n Convolutional Neural Networks for Visual Recognition*

La Figura 58 muestra de mejor manera el proceso. En este caso se tiene una imagen de 32 pixeles de ancho por 32 pixeles de alto y con una profundidad de 3. La profundidad está compuesta por las tres matrices, una de cada canal de la imagen RGB. Luego se tiene la capa de convolución. En este caso la capa tiene 5 neuronas de profundidad. Cada neurona representa un filtro distinto. Se desplaza cada uno de esos 5 filtros alrededor de toda la imagen y se obtendrá de salida con una profundidad de 5. A continuación, se detalla una convolución sencilla basado en forma general en el ejemplo de Karpathy et al. (2016):

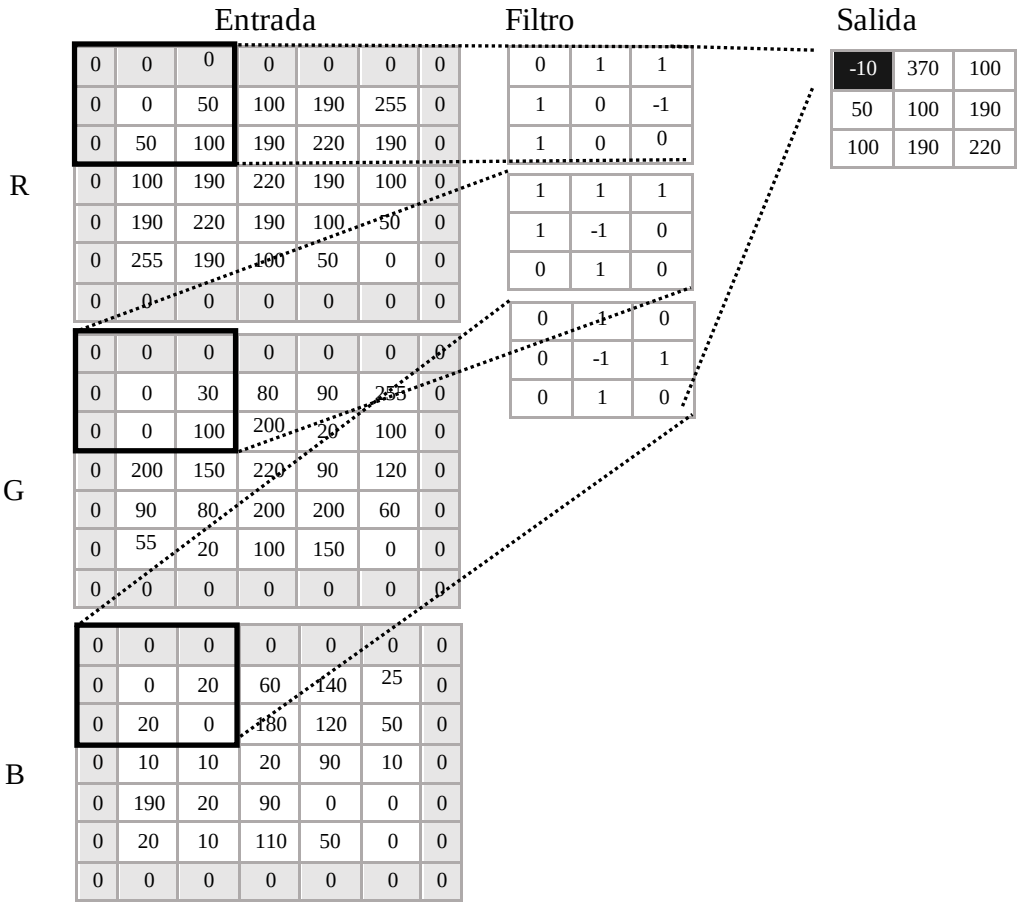


Figura 59. Ejemplo de convolución
Fuente: Elaboración Propia

En este caso la convolución se aplicó sin invertir el filtro al igual que en el ejemplo de preprocesamiento de imágenes. Invertir la matriz del filtro no afectará el resultado ya que las matrices de pesos son calculadas por la misma red. La Figura 59 muestra el ejemplo con un solo filtro. Es decir, la salida tendrá una profundidad de 1. Cabe destacar que en este caso la imagen tiene un tamaño de 5 pixeles de ancho y 5 pixeles de altura. Se le ha agregado un borde de pixeles con el valor 0 para que se pueda realizar las convoluciones en los bordes. Como se puede observar, se tienen 3 canales de entrada por lo que el filtro tendrá una profundidad de 3 lo que se ve representado por las 3 matrices en el filtro. Cada matriz realiza la convolución en una de las capas RGB. Luego, el resultado se suma y se asigna a una celda de la matriz de salida. Por lo tanto, en el ejemplo se resaltan las matrices que deben realizar la convolución para hallar solo el primer resultado de la matriz de salida. Para hallar los otros valores, se debe seguir el mismo proceso desplazando el filtro en la imagen.

En este caso la matriz de salida tiene menor dimensión de ancho y alto (3x3) que la imagen de entrada (5x5). Esto se debe a que, en este caso el filtro se desplazará dos casillas alrededor de la imagen al aplicar la convolución. La cantidad de casillas que se desplaza el filtro es un hiperparámetro que puede ser elegido al momento de crear el modelo (Karpathy, Johnson, & Fei-Fei, 2016). En caso se hubiese desplazado sólo una casilla, la matriz de salida habría tenido el mismo ancho y alto (5x5).

A continuación, se listan los hiperparámetros más importantes de la capa de convolución (Karpathy et al., 2016):

- K es el número de filtros a aplicar o profundidad (*depth*). En el caso del ejemplo fue 1.
- F es el tamaño del filtro (*field size*). Es decir, si se tiene filtros 3x3, F será 3. Los filtros siempre tienen dimensiones iguales en lo ancho y en lo alto. En el caso del ejemplo fue 3.
- S es el desplazamiento del filtro (*stride*). En el caso del ejemplo fue 2.
- P es la cantidad de borde que se ha agregado (*padding*). En el caso del ejemplo fue 1 pixel.

Con estos parámetros se puede calcular el tamaño del volumen de salida mediante las siguientes fórmulas:

- Ancho del volumen de salida = $\frac{W_1-F+2P}{S}+1$. Siendo W_1 el ancho del volumen de entrada. En el caso del ejemplo sería $\frac{5-3+2(1)}{2}+1=3$

- Altura del volumen de salida = $\frac{H_1 - F + 2P}{S} + 1$. Siendo H_1 el alto del volumen de entrada. En el caso del ejemplo sería $\frac{5-3+2(1)}{2} + 1 = 3$
- Profundidad de volumen de salida = K. En el caso el ejemplo sería 1.

Al igual que en la red neuronal tradicional, a la multiplicación de los pesos se le suele agregar un factor de sesgo (*bias*). El *bias* es también una variable aprendida por cada filtro y es solo un factor escalar. Por lo tanto, si en el ejemplo de arriba el filtro tuviese un *bias* de 20, se le sumaría 20 a todos los valores de salida.

2.2.4.2.2. Capa de submuestreo (*pooling*)

En esta capa lo que se hace es reducir el tamaño de las matrices. Es común hacer esto entre capas de convolución para reducir progresivamente el número de parámetros de cada capa. Por ejemplo, se podría utilizar filtros de 2x2 con un desplazamiento de 2 en los cuales se elige el pixel que tiene el valor máximo en cada zona (*max-pooling*). En este caso se elige el máximo de 4 números y se desplaza a los siguientes 4 números para hacer lo mismo. Esto da como resultado descartar 75% de los valores (Karpathy, Johnson, & Fei-Fei, 2016).

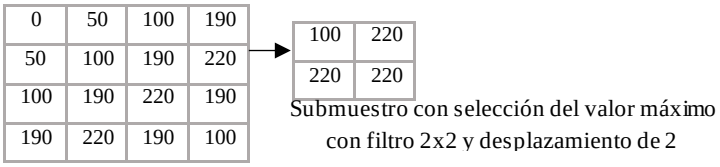


Figura 60. Ejemplo de submuestro max-pooling
Fuente: Elaboración Propia

En la Figura 60 se muestra un ejemplo de submuestreo. Se puede observar que para obtener el primer valor se determina el máximo de $\begin{bmatrix} 0 & 50 \\ 50 & 100 \end{bmatrix}$ el cual es 100. El filtro se desplaza 2 pixeles para obtener el resultado en las otras 3 posiciones.

2.2.4.2.3. Capa totalmente conectada

En esta capa todas las neuronas se conectan a todas las activaciones de la capa anterior como en una red neuronal tradicional (Karpathy, Johnson, & Fei-Fei, 2016). Generalmente es la última capa de la red en la que se busca realizar la clasificación o llegar a una representación de características a partir de las matrices obtenidas hasta el momento. Sin embargo, igual es posible tener más de una de estas capas en la red.

2.2.4.2.4. Capa de activación

Algunos consideran la capa de activación como una capa aparte (Karpathy, Johnson, & Fei-Fei, 2016). Sin embargo, esta puede estar contenida dentro de otras capas de manera implícita (LeCun, Bottou, Bengio, & Haffner, 1998). En sí lo que hace la capa de activación es aplicar la función no-lineal de activación a los pesos como en la red neuronal tradicional. Como se vio anteriormente esta puede ser la función sigmoide, la función ReLU, entre otras.

2.2.4.2.5. *Backpropagation* en una Red Neuronal Convolucional

A continuación, se explicará el aprendizaje de la red neuronal convolucional. De igual manera como con una red neuronal tradicional se utiliza la iteración hacia adelante y la iteración hacia atrás. Se calcula la iteración hacia adelante dependiendo del tipo de capa. Posteriormente, se calcula el error, este se puede calcular de igual manera que en la red neuronal tradicional.

Para este caso se tomará como referencia la siguiente red:

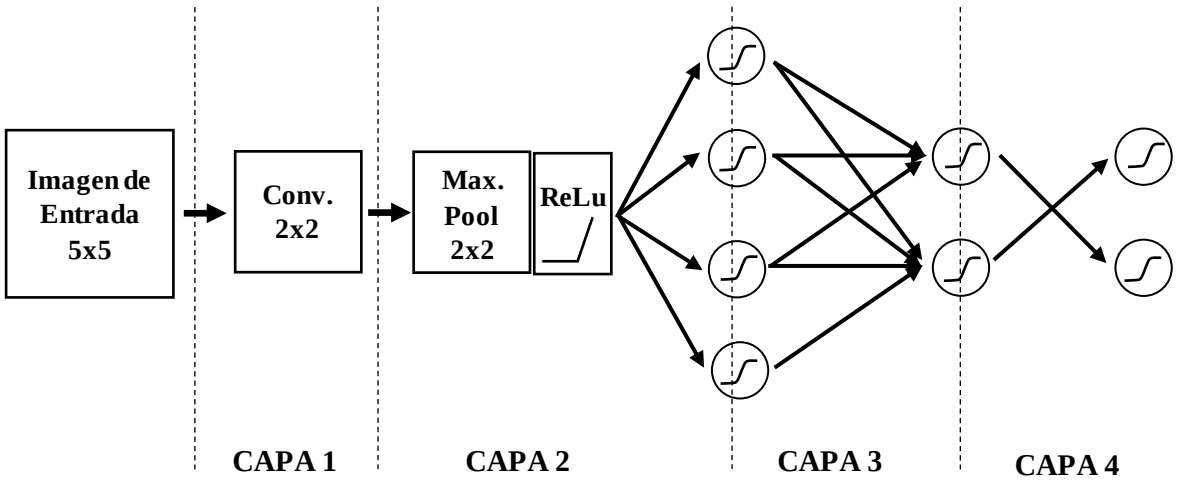


Figura 61. Ejemplo práctico de arquitectura de red neuronal convolucional
Fuente: Elaboración Propia

Se puede observar que se tienen 4 capas:

- Capa 1: Capa de convolución con un kernel 2x2.
- Capa 2: Capa de Max-pooling con activación ReLU posterior.
- Capa 3: Capa totalmente conectada con activación sigmoide en cada neurona.
- Capa 4: Capa totalmente conectada con activación sigmoide en cada neurona.

Adicionalmente, se tiene en consideración que la entrada es una imagen en escala de grises 5x5. Esta entrada para motivos del ejemplo estará normalizada con valores entre 0 y 1. Los valores serán los siguientes:

0.1	0.3	0.5	0.8	1
0.3	0.1	0.3	0.8	0.8
0.5	0.3	0.5	0.3	0.5
0.8	0.8	0.3	0.1	0.3
1	0.8	0.5	0.3	0.1

Figura 62. Ejemplo de imagen de entrada para red neuronal convolucional
Fuente: Elaboración Propia

Asimismo, se inicializarán los pesos al azar. A continuación, se describirá la iteración hacia adelante.

Capa 1 –Filtro de convolución

En la capa 1, los pesos serán los del kernel 2x2. En la Figura 63, se muestra los pesos que se tomarán como ejemplo.

0.8	0.9
0.7	0.5

Figura 63. Ejemplo de kernel de convolución Capa 1
Fuente: Elaboración Propia

Donde cada peso será representado por $w_{j\,k}^{l}$

- l es la capa de la red
- j es la posición vertical del peso
- k es la posición horizontal del peso

Asimismo, se tomará como referencia un b_1^1 de -0.1.

Por lo tanto, de acuerdo con lo mostrado en la Figura 63, se tienen los siguientes pesos más el *bias*:

- $w_{11}^1 = 0.8$
- $w_{12}^1 = 0.9$
- $w_{21}^1 = 0.7$
- $w_{22}^1 = 0.5$
- $b_1^1 = -0.1$

Mediante la aplicación del filtro se obtiene lo siguiente:

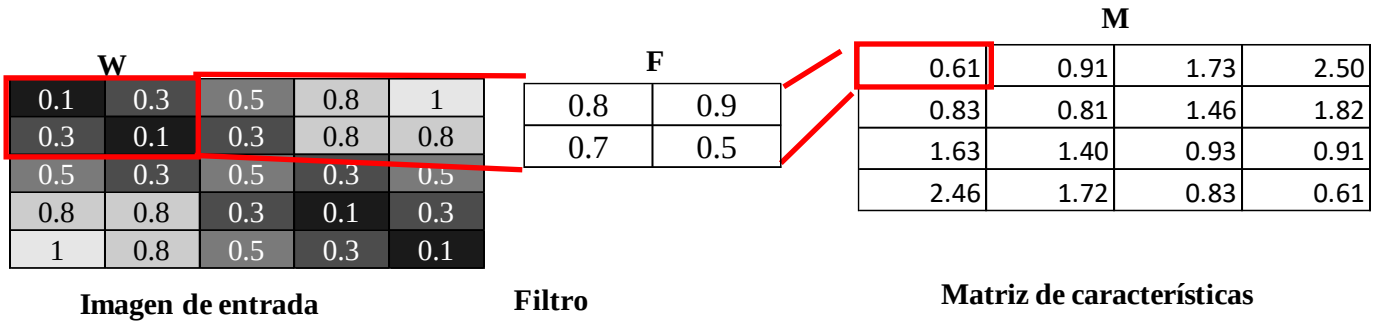


Figura 64. Ejemplo de aplicación del kernel de convolución en capa 1
Fuente: Elaboración Propia

Esto se obtuvo luego de realizar las siguientes operaciones al multiplicar el filtro y la imagen de entrada. Por ejemplo, para los tres primeros valores se obtendría de la siguiente manera usando la Ecuación mostrada anteriormente:

$$g(x,y) = \sum_i \sum_j f(i,j)w(i,j) \tag{2}$$

$$M_{1\,1} = 0.8(0.1) + 0.9(0.3) + 0.7(0.3) + 0.5(0.1) = 0.61$$

$$M_{1\,2} = 0.8(0.3) + 0.9(0.5) + 0.7(0.1) + 0.5(0.3) = 0.91$$

$$M_{1\,3} = 0.8(0.5) + 0.9(0.8) + 0.7(0.3) + 0.5(0.8) = 1.73$$

Y así sucesivamente desplazando el filtro por la imagen para hallar el resto de valores. Donde M es la matriz resultante de la multiplicación del filtro por la imagen de entrada.

Luego de sumar el *bias* b_1^1 a la matriz de características V se tiene la siguiente matriz resultante:

0.51	0.81	1.63	2.40
0.73	0.71	1.36	1.72
1.53	1.30	0.83	0.81
2.36	1.62	0.73	0.51

Figura 65. Resultado de matriz de características tras aplicar bia
Fuente: Elaboración Propia

Donde la salida de la capa será representada por $y_{j\,k}^{l}$.

- l es la capa de la red
- j es la posición vertical del resultado de la matriz de características

- k es la posición horizontal del resultado de la matriz de características

Por ejemplo:

- y_{41}^{1} = 2.36

Capa 2 – Capa de Submuestreo

En la capa de submuestreo se aplica submuestreo *max-pooling* a la matriz resultante de la capa de convolución. Por lo tanto, se tendría lo siguiente:

0.51	0.81	1.63	2.40
0.73	0.71	1.36	1.72
1.53	1.30	0.83	0.81
2.36	1.62	0.73	0.51

Matriz de características

→

0.81	2.40
2.36	0.83

Resultado de max-pooling

Figura 66. Ejemplo de aplicación de max-pooling en capa 2
Fuente: Elaboración Propia

Cada resultado de esta operación está representado por z_{jk}^{l}.

- l es la capa de la red
- k es la posición horizontal del resultado de la aplicación del *max-pooling*
- j es la posición vertical del resultado de la aplicación del *max-pooling*

Para calcular el resultado final dentro de esta capa, se aplica la función de activación ReLU a cada valor *y = f(z)*.

Por lo tanto, se calcula lo siguiente aplicando ReLU:

$$y_{11}^{2} = \max(0, z_{11}^{2}) = \max(0, 0.81) = 0.81$$
$$y_{12}^{2} = \max(0, z_{12}^{2}) = \max(0, 2.4) = 2.4$$
$$y_{21}^{2} = \max(0, z_{21}^{2}) = \max(0, 2.36) = 2.36$$
$$y_{22}^{2} = \max(0, z_{22}^{2}) = \max(0, 0.83) = 0.83$$

Finalmente, para poder ser procesados por la capa totalmente conectada, se aplana la matriz resultante en un solo vector de 4 valores.

Esto se ilustra a continuación:

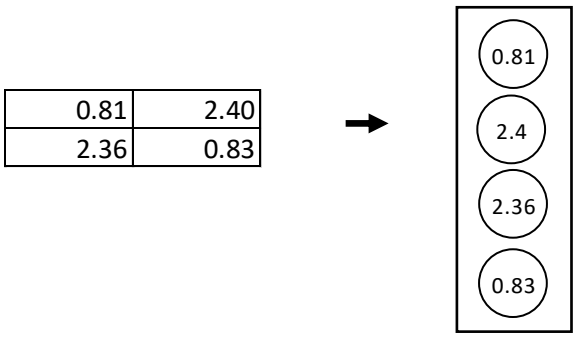


Figura 67. Ejemplo de aplanar matriz para capa totalmente conectada
Fuente: Elaboración Propia

Por lo tanto, el resultado final de la capa sería el siguiente:

$$y_1^2 = 0.81$$
$$y_2^2 = 2.4$$
$$y_3^2 = 2.36$$
$$y_4^2 = 0.83$$

Capa 3 – Capa totalmente conectada

Para la capa totalmente conectada se recibe de entrada los cuatro valores de salida de la capa anterior y se aplica la fórmula de la red neuronal tradicional. A continuación, se muestra la estructura de esa capa a detalle.

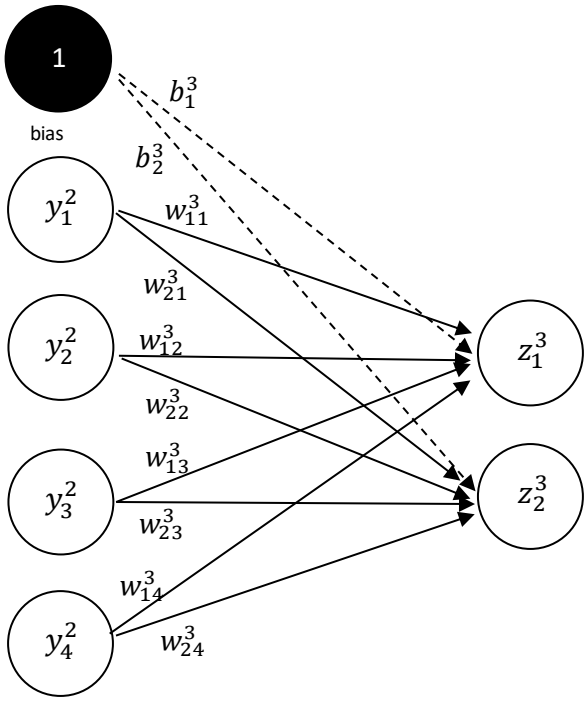


Figura 68. Ejemplo mostrando detalle de la capa 3
Fuente: Elaboración Propia

Donde w_{kj}^{l} representa el peso entre la neurona j y la neurona k de la capa l.

Asimismo, en este caso, los pesos y bias también han sido inicializados y se tomarán como referencia los siguientes valores:

$$w_{11}^{3} = 0.3$$
$$w_{12}^{3} = 0.25$$
$$w_{13}^{3} = 0.8$$
$$w_{14}^{3} = 0.4$$

$$w_{21}^{3} = 0.45$$
$$w_{22}^{3} = 0.6$$
$$w_{23}^{3} = 0.2$$
$$w_{24}^{3} = 0.7$$

$$b_{1}^{3} = 0.15$$
$$b_{2}^{3} = 0.2$$

Por lo tanto, se sabe que:

$$z_{1}^{3} = y_{1}^{2}(w_{11}^{3}) + y_{2}^{2}(w_{12}^{3}) + y_{3}^{2}(w_{13}^{3}) + y_{4}^{2}(y_{14}^{3}) + b_{1}^{3}$$
$$z_{1}^{3} = 0.81(0.3) + 2.4(0.25) + 2.36(0.8) + 0.83(0.4) + 0.15$$
$$z_{1}^{3} = 3.21$$

$$z_{2}^{3} = y_{1}^{2}(w_{21}^{3}) + y_{2}^{2}(w_{22}^{3}) + y_{3}^{2}(w_{23}^{3}) + y_{4}^{2}(y_{24}^{3}) + b_{2}^{3}$$
$$z_{2}^{3} = 0.81(0.45) + 2.4(0.6) + 2.36(0.2) + 0.83(0.7) + 0.2$$
$$z_{2}^{3} = 3.06$$

Asimismo, aplicando la función de activación sigmoide, se obtiene el siguiente resultado:

$$y_{1}^{3} = \frac{1}{1 + e^{-z_{1}^{3}}} = \frac{1}{1 + e^{-3.21}} = 0.96$$
$$y_{2}^{3} = \frac{1}{1 + e^{-z_{2}^{3}}} = \frac{1}{1 + e^{-3.06}} = 0.95$$

Capa 4 – Capa totalmente conectada

Para la capa final se aplica la misma forma de resolución que en la capa 3. Se recibe de entrada los dos valores de salida de la capa anterior y se aplica la fórmula de la red neuronal tradicional.

En esta capa, los pesos y *bias* también han sido inicializados y se tomarán como referencia los siguientes valores:

$$w_{11}^4 = 0.4$$
$$w_{12}^4 = 0.8$$

$$w_{21}^4 = 0.25$$
$$w_{22}^4 = 0.65$$

$$b_1^4 = 0.05$$
$$b_2^4 = 0.1$$

A continuación, se muestra la resolución a detalle:

$$z_1^4 = y_1^3(w_{11}^4) + y_2^3(w_{12}^4) + b_1^4$$
$$z_1^4 = 0.96(0.4) + 0.95(0.8) + 0.05$$
$$z_1^4 = 1.2$$

$$z_2^4 = y_1^3(w_{21}^4) + y_2^3(w_{22}^4) + b_2^4$$
$$z_2^4 = 0.96(0.25) + 0.95(0.65) + 0.1$$
$$z_2^4 = 0.96$$

Finalmente, se aplica la función de activación sigmoide, obteniendo el siguiente resultado:

$$y_1^4 = \frac{1}{1 + e^{-z_1^3}} = \frac{1}{1 + e^{-1.2}} = 0.77$$
$$y_2^4 = \frac{1}{1 + e^{-z_2^3}} = \frac{1}{1 + e^{-0.96}} = 0.72$$

Si se asume que el resultado deseado para esta imagen de entrada (Figura 62) es 1 para la primera neurona de salida y 0 para la segunda. Se tiene el siguiente cálculo del error euclidiano:

$$E_k = \frac{1}{2}\sum_j \left(d_{k,j} - y_{k,j} \right)^2 = \frac{1}{2}((1-0.77)^2 + (0-0.72)^2)$$

$$E_k = 0.29$$

A continuación, se empieza con el cálculo de las gradientes mediante *backpropagation* en la red. Esto utiliza los mismos principios teóricos que en una red neuronal tradicional

Gradiente en la Capa 4 – Capa totalmente conectada:

En primer lugar, se debe calcular la gradiente en la última capa. Para esto se parte de que se está usando el error de distancia euclidiana.

De acuerdo con la Ecuación 30 definida anteriormente, la derivada de la última capa de una red totalmente conectada se obtiene con la siguiente fórmula:

$$\delta_j^L = \frac{\partial E}{\partial y_j^L} f'(z_j^L) \tag{30}$$

Por lo tanto, en este caso, se tendría lo siguiente:

$$\delta_j^L = (y_j^L - d_j)\, y_j^L (1 - y_j^L)$$

$$\delta_1^4 = (0.77 - 1) \times 0.77 \times (1 - 0.77) = -0.04$$

$$\delta_2^4 = (0.72 - 0) \times 0.72 \times (1 - 0.72) = 0.14$$

En base a lo hallado, es posible actualizar los pesos de dicha capa con la Ecuación 39:

$$\frac{\partial E}{\partial w_{jk}^l} = y_k^{l-1} \delta_j^l \tag{39}$$

$$\frac{\partial E}{\partial w_{11}^4} = 0.96(-0.04) = -0.0397$$

$$\frac{\partial E}{\partial w_{21}^4} = 0.96(0.14) = 0.1392$$

$$\frac{\partial E}{\partial w_{12}^4} = 0.95(-0.04) = -0.0394$$

$$\frac{\partial E}{\partial w_{22}^4} = 0.95(0.14) = 0.1383$$

Asimismo, es posible calcular la gradiente respecto al *bias* mediante la Ecuación 38:

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l \tag{38}$$

$$\frac{\partial E}{\partial b_1^4} = \delta_1^4 = -0.04$$

$$\frac{\partial E}{\partial b_2^4} = \delta_2^4 = 0.14$$

Gradiente en la Capa 3 – Capa totalmente conectada:

En la siguiente capa se puede hacer el mismo procedimiento con la fórmula de capa oculta definida en la Ecuación 37:

$$\delta_j^l = \sum_k \left(w_{kj}^{l+1}\delta_k^{l+1}\right) f'\left(z_j^l\right) \tag{37}$$

$$\delta_1^3 = \left(w_{11}^4\delta_1^4 + w_{21}^4\delta_2^4\right)\left(y_1^3\right)\left(1 - y_1^3\right)$$

$$\delta_1^3 = \left(0.4(-0.04) + 0.25(0.14)\right)(0.96)(1 - 0.96)$$

$$\delta_1^3 = 0.00073$$

$$\delta_2^3 = \left(w_{12}^4\delta_1^4 + w_{22}^4\delta_2^4\right)\left(y_1^3\right)\left(1 - y_1^3\right)$$

$$\delta_2^3 = \left(0.8(-0.04) + 0.65(0.14)\right)(0.95)(1 - 0.95)$$

$$\delta_2^3 = 0.00262$$

Se calculan las derivadas respecto a los pesos y *bias* de la capa 3:

$$\frac{\partial E}{\partial w_{jk}^l} = y_k^{l-1}\delta_j^l$$

$$\frac{\partial E}{\partial w_{11}^3} = 0.81(0.00073) = 0.0006$$

$$\frac{\partial E}{\partial w_{21}^3} = 0.81(0.00262) = 0.0021$$

$$\frac{\partial E}{\partial w_{12}^3} = 2.4(0.00073) = 0.0018$$

$$\frac{\partial E}{\partial w_{22}^3} = 2.4(0.00262) = 0.0063$$

$$\frac{\partial E}{\partial w_{13}^3} = 2.36(0.00073) = 0.0017$$

$$\frac{\partial E}{\partial w_{23}^3} = 2.36(0.00262) = 0.0062$$

$$\frac{\partial E}{\partial w_{14}^3} = 0.83(0.00073) = 0.0006$$

$$\frac{\partial E}{\partial w_{24}^3} = 0.83(0.00262) = 0.0022$$

$$\frac{\partial E}{\partial b_1^3} = \delta_1^3 = 0.00073$$

$$\frac{\partial E}{\partial b_2^3} = \delta_2^3 = 0.00262$$

Gradiente en la Capa 2 – Capa de Submuestreo *Max -pool*

En la capa 2 de *max-pool* primero se calculará la derivada respecto al error de la misma forma que en las capas anteriores. Sin embargo, en este caso al ser la función RELU se obtendrá su derivada mediante la siguiente función:

$$f'(x) = \begin{cases} 1, & si\ x > 0 \\ 0, & si\ x <= 0 \end{cases}$$

Por lo tanto, aplicando la Ecuación 37 se obtiene lo siguiente:

$$\delta_j^l = \sum_k \left(w_{kj}^{l+1}\delta_k^{l+1}\right) f'\left(z_j^l\right) \tag{37}$$

$$\delta_1^2 = (w_{11}^3\delta_1^3 + w_{21}^3\delta_2^3)\,(1)$$

$$\delta_1^2 = 0.3((0.00073) + 0.45(0.00262))(1)$$

$$\delta_1^3 = 0.0014$$

$$\delta_2^2 = (w_{12}^3\delta_1^3 + w_{22}^3\delta_2^3)\,(1)$$

$$\delta_2^2 = (0.25(0.00073) + 0.6(0.00262))\,(1)$$

$$\delta_2^3 = 0.00175$$

$$\delta_3^2 = (w_{13}^3\delta_1^3 + w_{23}^3\delta_2^3)\,(1)$$

$$\delta_3^2 = (0.8(0.00073) + 0.2(0.00262))\,(1)$$

$$\delta_2^3 = 0.00111$$

$$\delta_4^2 = (w_{14}^3 \delta_1^3 + w_{24}^3 \delta_2^3)\,(y_4^2)(1)$$
$$\delta_4^2 = (0.4(0.00073) + 0.7(0.00262))\,(1)$$
$$\delta_2^3 = 0.00213$$

Las variaciones calculadas en el error son de la capa del Max-pool. Estas gradientes se colocan en los lugares donde se tuvo el máximo valor del mapa de características de la convolución. Por lo tanto, se obtiene la siguiente matriz con las variaciones de los pesos del submuestreo:

0.00000	0.00140	0.00000	0.00175
0.00000	0.00000	0.00000	0.00000
0.00000	0.00000	0.00213	0.00000
0.00111	0.00000	0.00000	0.00000

Figura 69. Resultado de propagación hacia atrás en capa de submuestreo
Fuente: Elaboración Propia

Gradiente en la Capa 1 – Capa de Convolución

Para la capa de convolución se parte del resultado de la matriz anterior y se aplica la fórmula de propagación hacia atrás pero de forma matricial:

En este caso no hay función de activación por lo que se tiene lo siguiente (Gibiansky, 2014):

$$\frac{\partial E}{\partial w_{kj}^l} = \sum_{a=0}^{M-m} \sum_{b=0}^{N-n} y_{(k+a),(j+b)}^{l-1} \delta_{ab}^l \tag{44}$$

Donde δ_{kj}^1 es representado por la matriz de la Figura 69.

- k es la posición horizontal del peso en la matriz.
- j es la posición vertical del peso en la matriz.
- m es el alto del kernel. En este caso es 2.
- n es el ancho del kernel. En este caso es 2.
- M es el alto de la matriz y. En este caso es la matriz de entrada con alto 5.
- N es el ancho de la matriz y. En este caso es la matriz de entrada con ancho de 5.

Este cálculo se ilustra de la siguiente manera:

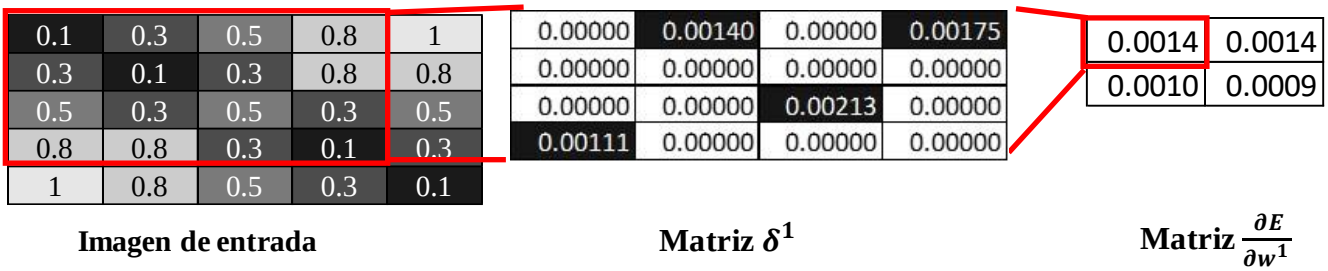


Figura 70. Resultado de propagación hacia atrás en capa de convolución
Fuente: Elaboración Propia

Por lo tanto, se calcula lo siguiente:

$$\frac{\partial E}{\partial w_{11}^1} = y_{11}^0\delta_{11}^1 + y_{21}^0\delta_{21}^1 + y_{31}^0\delta_{31}^1 + y_{41}^0\delta_{41}^1 + y_{12}^0\delta_{12}^1 + y_{22}^0\delta_{22}^l + y_{32}^0\delta_{32}^l + y_{42}^0\delta_{42}^l$$
$$+ y_{13}^0\delta_{13}^l + y_{23}^0\delta_{23}^l + y_{33}^0\delta_{33}^1 + y_{43}^0\delta_{43}^l + y_{14}^0\delta_{14}^l + y_{24}^0\delta_{24}^l + y_{34}^0\delta_{34}^l + y_{44}^0\delta_{44}^l$$
$$\frac{\partial E}{\partial w_{11}^1} = 0 + 0 + 0 + 0.8(0.00111) + 0.3(0.0014) + 0 + 0 + 0 + 0 + 0 + 0.5(0.00213)$$
$$+ 0 + 0.8(0.00176) + 0 + 0 + 0 = 0.0038$$

$$\frac{\partial E}{\partial w_{12}^1} = y_{21}^0\delta_{11}^1 + y_{31}^0\delta_{21}^1 + y_{41}^0\delta_{31}^1 + y_{51}^0\delta_{41}^1 + y_{22}^0\delta_{12}^1 + y_{32}^0\delta_{22}^l + y_{42}^0\delta_{32}^l + y_{52}^0\delta_{42}^l$$
$$+ y_{23}^0\delta_{13}^l + y_{33}^0\delta_{23}^l + y_{43}^0\delta_{33}^1 + y_{53}^0\delta_{43}^l + y_{24}^0\delta_{14}^l + y_{34}^0\delta_{24}^l + y_{44}^0\delta_{34}^l + y_{54}^0\delta_{44}^l$$
$$\frac{\partial E}{\partial w_{12}^1} = 0 + 0 + 0 + 1(0.00111) + 0.1(0.0014) + 0 + 0 + 0 + 0 + 0 + 0.3(0.00213)$$
$$+ 0 + 0.8(0.00176) + 0 + 0 + 0 = 0.0033$$

$$\frac{\partial E}{\partial w_{21}^1} = y_{12}^0\delta_{11}^1 + y_{22}^0\delta_{21}^1 + y_{32}^0\delta_{31}^1 + y_{42}^0\delta_{41}^1 + y_{13}^0\delta_{12}^1 + y_{23}^0\delta_{22}^l + y_{33}^0\delta_{32}^l + y_{43}^0\delta_{42}^l$$
$$+ y_{14}^0\delta_{13}^l + y_{24}^0\delta_{23}^l + y_{34}^0\delta_{33}^1 + y_{44}^0\delta_{43}^l + y_{15}^0\delta_{14}^l + y_{25}^0\delta_{24}^l + y_{35}^0\delta_{34}^l + y_{45}^0\delta_{44}^l$$
$$\frac{\partial E}{\partial w_{21}^1} = 0 + 0 + 0 + 0.8(0.00111) + 0.5(0.0014) + 0 + 0 + 0 + 0 + 0 + 0.3(0.00213)$$
$$+ 0 + 1(0.00176) + 0 + 0 + 0 = 0.004$$

$$\frac{\partial E}{\partial w_{22}^1} = y_{22}^0 \delta_{11}^1 + y_{32}^0 \delta_{21}^1 + y_{34}^0 \delta_{31}^1 + y_{52}^0 \delta_{41}^1 + y_{23}^0 \delta_{12}^1 + y_{33}^0 \delta_{22}^l + y_{43}^0 \delta_{32}^l + y_{53}^0 \delta_{42}^l + y_{24}^0 \delta_{13}^l + y_{34}^0 \delta_{23}^l + y_{44}^0 \delta_{33}^1 + y_{54}^0 \delta_{43}^l + y_{25}^0 \delta_{14}^l + y_{35}^0 \delta_{24}^l + y_{45}^0 \delta_{34}^l + y_{55}^0 \delta_{44}^l$$

$$\frac{\partial E}{\partial w_{22}^1} = 0 + 0 + 0 + 0.8(0.00111) + 0.3(0.0014) + 0 + 0 + 0 + 0 + 0 + 0.1(0.00213) + 0 + 0.8(0.00176) + 0 + 0 + 0 = 0.0029$$

En el caso del *bias* de la capa de convolución, de igual manera se actualiza con la siguiente fórmula:

$$\frac{\partial E}{\partial b_j^l} = \sum_{a=0}^{m} \sum_{b=0}^{n} \delta_{ab}^l \tag{45}$$

$$\frac{\partial E}{\partial b_1^1} = 0 + 0.0014 + 0 + 0.00176 + 0 + 0 + 0 + 0 + 0 + 0 + 0.00213 + 0 + 0.00111 + 0 + 0 + 0 = 0.0064$$

Finalmente se actualizan los pesos y *bias* en cada capa basado en las Ecuaciones 40 y 41, como se describió en la red neuronal tradicional.

$$\Delta w_{j\,k}^l = \eta \frac{\partial E}{\partial w_{j\,k}^l} \tag{40}$$

$$\Delta b_j^l = \eta \frac{\partial E}{\partial b_j^l} \tag{41}$$

Donde η es el ratio de aprendizaje. En este caso se usará 0.4 como ratio de aprendizaje a modo de ejemplo.

<u>Actualizar pesos capa 1 – Capa de convolución</u>

En este caso es similar como se describió en la red neuronal tradicional pero utilizando las filas y columnas de la matriz de pesos.

$$w_{j\,k}^l = w_{j\,k}^l - \Delta w_{j\,k}^l$$

$$w_{11}^1 = 0.8 - 0.4(0.0014) = 0.7985$$

$$w_{21}^1 = 0.7 - 0.4(0.0033) = 06987$$

$$w_{12}^1 = 0.9 - 0.4(0.0040) = 0.8984$$

$$w_{22}^1 = 0.5 - 0.4(0.0029) = 0.4988$$

$$b_j^l = b_j^l - \Delta b_j^l$$

$$b_1^1 = -0.1 - 0.4(0.0064\ ) = -0.103$$

Por lo tanto, el nuevo filtro es el siguiente:

0.7985	0.8984
0.6987	0.4988

Figura 71. Nuevo filtro w^1 para la capa de convolución
Fuente: Elaboración Propia

Actualizar pesos capa 3 – Capa totalmente conectada

En este caso es como la capa totalmente conectada de la red neuronal tradicional.

$$w_{kj}^l = w_{kj}^l - \Delta w_{kj}^l$$

$$w_{11}^3 = 0.3 - 0.4(0.0006) = 0.2998$$
$$w_{12}^3 = 0.25 - 0.4(0.0018) = 0.2493$$
$$w_{13}^3 = 0.8 - 0.4(0.0017) = 0.7993$$
$$w_{14}^3 = 0.4 - 0.4(0.0006) = 0.3998$$

$$w_{21}^3 = 0.45 - 0.4(0.0021) = 0.4492$$
$$w_{22}^3 = 0.6 - 0.4(0.0063) = 0.5975$$
$$w_{23}^3 = 0.2 - 0.4(0.0062) = 0.1975$$
$$w_{24}^3 = 0.7 - 0.4(0.0022) = 0.6991$$

$$b_j^l = b_j^l - \Delta b_j^l$$

$$b_1^3 = 0.15 - 0.4(0.00073\ ) = 0.1497$$
$$b_2^3 = 0.2 - 0.4(0.00262\ ) = 0.1990$$

Actualizar pesos capa 4 – Capa totalmente conectada

$$w_{kj}^l = w_{kj}^l - \Delta w_{kj}^l$$

$$w_{11}^4 = 0.4 - 0.4(-0.03966) = 0.4163$$
$$w_{12}^4 = 0.8 - 0.4(0.139) = 0.8167$$

$$w_{21}^4 = 0.25 - 0.4(-0.0394) = 0.2012$$
$$w_{22}^4 = 0.65 - 0.4(0.138) = 0.6001$$

$$b_j^l = b_j^l - \Delta b_j^l$$
$$b_1^4 = 0.05 - 0.4(-0.04) = 0.067$$
$$b_2^4 = 0.1 - 0.4(0.14) = 0.042$$

Con lo planteado en esta sección, se ejemplifica la aplicación de *backpropagation* en una red neuronal convolucional. Como se puede observar, muchos de los principios y fórmulas de una red neuronal tradicional, aplican de forma similar en este tipo de redes.

2.2.4.2.6. Ejemplos de Arquitectura tradicional

Se han podido observar los tipos principales de capas de las redes neuronales convolucionales. Estas se unen de maneras innumerables para generar la red. A continuación, se muestran algunos ejemplos gráficos de dichas arquitecturas.

AlexNet

En la Figura 72 se observa la arquitectura de AlexNet:

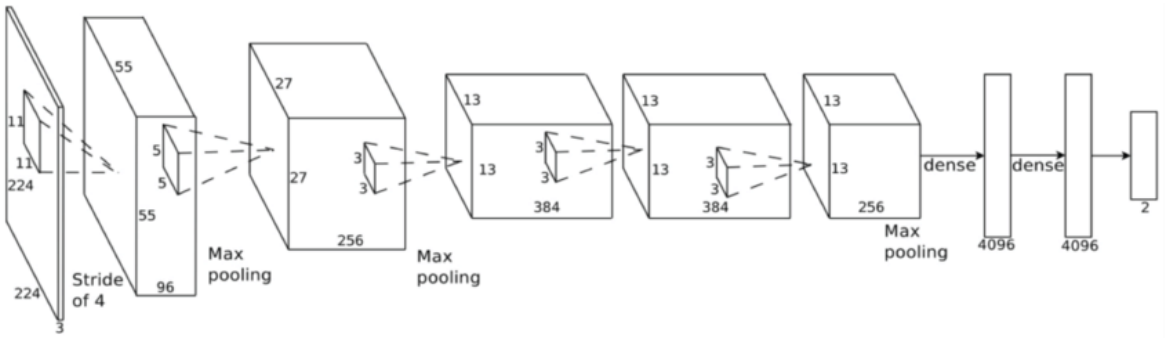


Figura 72. Arquitectura de AlexNet
Fuente: Krizhevsky, Sutskever & Hinton (2012). *ImageNet Classification with Deep Convolutional Neural Networks.* (p. 5)

Esta arquitectura fue la ganadora en el reto ImageNet 2012 denominado ImageNet Large Scale Visual Recognition Challenge (ILSVRC). Obtuvo mayor efectividad en comparación a los otros modelos tradicionales de aprendizaje automático y visión computacional. Esto fue uno de los puntos de quiebre en la historia del *deep learning* (Alom, et al., 2018).

A continuación, se muestra el detalle de esta arquitectura. Además, se puede observar el cambio en tamaño del volumen dado una imagen de entrada de 227 x 227 pixeles.

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55 x 55 x 96	11x11	4	relu
	Max Pooling	96	27 x 27 x 96	3x3	2	relu
2	Convolution	256	27 x 27 x 256	5x5	1	relu
	Max Pooling	256	13 x 13 x 256	3x3	2	relu
3	Convolution	384	13 x 13 x 384	3x3	1	relu
4	Convolution	384	13 x 13 x 384	3x3	1	relu
5	Convolution	256	13 x 13 x 256	3x3	1	relu
	Max Pooling	256	6 x 6 x 256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	Softmax

Figura 73. Detalle de capas y cambio de volumen en modelo AlexNet
Fuente: Gao (2017). *A Walk-through of AlexNet*

Si bien en la Figura 72 se observa un tamaño de entrada de 224 x 224 x 3, esto se debe a un error en la publicación original, dado que para que los números tengan relación, se debe tener una entrada de 227 x 227 x 3 (Gao, 2017).

Por lo tanto, se inicia con un volumen de tamaño 227 x 227 x 3 dado una imagen de entrada con 227 pixeles de ancho, 227 pixeles de alto y 3 canales de color RGB. A través de capas de convolución, capas de submuestreo *max-pooling* y capas totalmente conectadas, este volumen se va transformando. Esto se puede observar en la columna de tamaño (*size*). En las capas intermedias se utiliza ReLU como función de activación. En la capa final se propone una función de *softmax* dado que permite generar resultados de probabilidad para clasificación multiclase.

VGG-16

Por otro lado, a continuación, se muestra un ejemplo de la arquitectura VGG-16 (Simonyan & Zisserman, 2014):

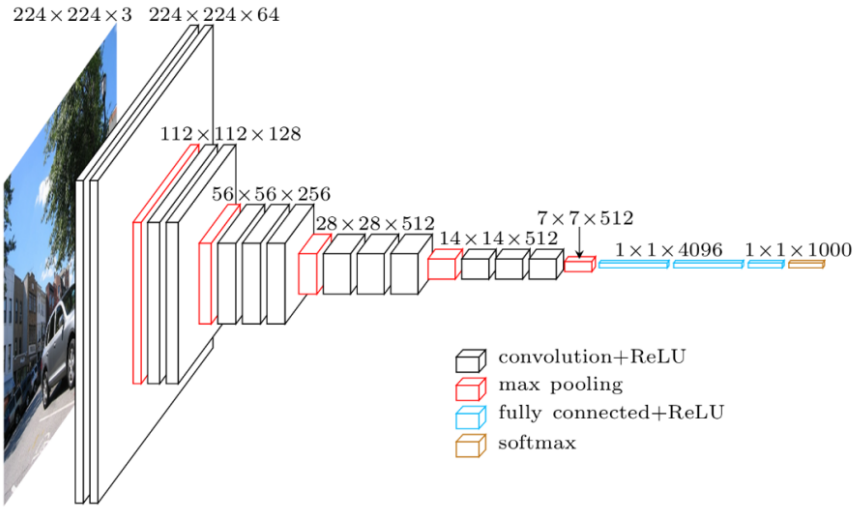


Figura 74. Arquitectura VGG-16
Fuente: Sugata & Yang (2017). *Leaf App: Leaf recognition with deep convolutional neural networks* (p. 4)

VGG-16 es una arquitectura que participó del concurso de ImageNet 2014. Esta arquitectura consiste en capas de convolución, las cuales utilizan ReLU como función de activación y capas de submuestreo *max-pooling*. Se crearon varias arquitecturas VGG, entre las que se encuentran VGG-16 y VGG-19. VGG-16 utiliza 13 capas de convolución mientras que VGG-19 utiliza 16 capas de convolución. Asimismo, en la capa final utiliza *sofmax* para el cálculo de las probabilidades.

Como se puede observar, los conceptos básicos se mantienen vigentes en cualquier arquitectura de redes neuronales convolucionales. Las nuevas redes neuronales convolucionales se construyen sobre dichas capas tradicionales agregando nuevos elementos o variaciones. A partir de estos conceptos básicos se han construido redes neuronales más complejas y se han creado capas más avanzadas. Un ejemplo importante de esto es el módulo *Inception* creado en Google LeNet (Szegedy et al., 2014) y que fue parte de uno de los modelos ganadores de ImageNet en el 2014. A continuación, se describirá este módulo.

2.2.4.2.7. Capa *Inception*

El módulo *Inception* plantea el uso de convoluciones 1x1, 3x3, 5x5 en paralelo y adiciona también un submuestreo 3x3 con selección del valor máximo (*max-pooling*).

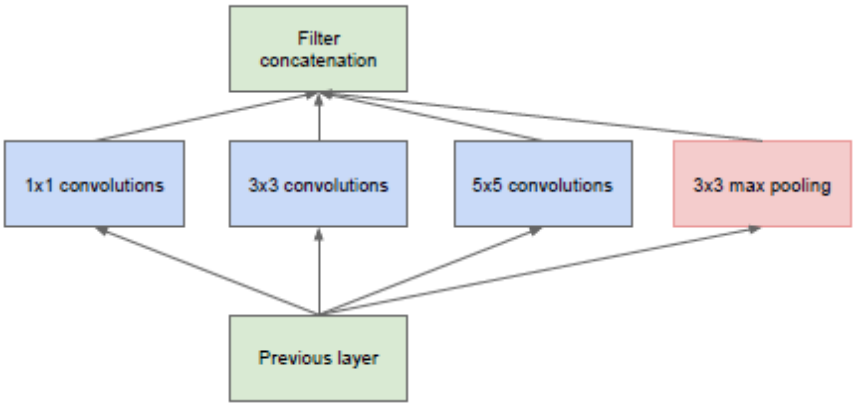


Figura 75. Ejemplo de módulo Inception (a)
Fuente: Szegedy et al. (2014). *Going deeper with convolutions.* (p. 5)

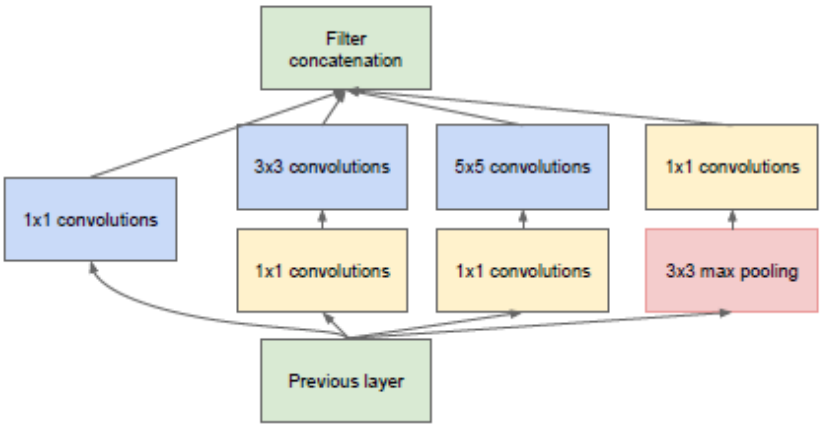


Figura 76. Ejemplo de Módulo Inception (b)
Fuente: Szegedy et al. (2014). *Going deeper with convolutions.* (p. 5)

Debido a la gran cantidad de parámetros y complejidad que esto agregaba, se modificó la idea planteando la reducción de la dimensionalidad usando convoluciones 1x1. Esto se ejemplifica en la Figura 75 y 76 (Szegedy, et al., 2014). Además, estas convoluciones incluyen la función de activación. En este caso utilizan la función ReLU.

Un ejemplo de arquitectura y tamaño de los volúmenes se muestra en le Figura 77:

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figura 77. Detalle de arquitectura y cambio de volúmenes en modelo Inception
Fuente: Szegedy et al. (2014). *Going deeper with convolutions.* (p. 6)

En este caso, la red contiene capas de convolución y de submuestro. Además, contiene una capa *dropout* que, al azar desactiva ciertas neuronas durante cada iteración para evitar que se sobreajuste el modelo a ciertas neuronas. Asimismo, se utiliza una función softmax en la capa final. La principal diferencia con las arquitecturas anteriores es que incluye capas *Inception*. Estas capas *Inception*, como se observó en la Figura 75 y 76, tienen variantes en su arquitectura y están compuestas por varias capas internamente. En este caso, se empieza con una imagen de tamaño 224 x 224 x 3 pixeles que, luego de la primera convolución, reduce su tamaño a la mitad (112 x 112 x 3 pixeles). Posteriormente el volumen sigue cambiando, dependiendo de las operaciones de cada capa hasta llegar a generar las probabilidades de la capa final.

2.2.4.2.8. Triplet Loss

En el caso del reconocimiento facial o comparación de similitud entre rostros usando las redes neuronales convolucionales, uno de los avances más importantes fue el de *FaceNet* (Schroff, Kalenichenko, & Philbin, 2015). Esta red extrae las características mediante una red neuronal convolucional generando una representación de 128-bytes por cara.

Figura 78. Triplet Loss, medida de distancia euclidiana usada por FaceNet
Fuente: Schroff, Kalenichenko & Philbin (2015). *FaceNet: A Unified Embedding for Face Recognition and Clustering.* (p. 3)

Tal como se observa en la Figura 78, esta representación utiliza una función de error que busca minimizar la distancia euclidiana entre imágenes de la misma persona y maximizarla entre imágenes de distintas personas. Asimismo, esto se observa con rostros de personas en la Figura a continuación:

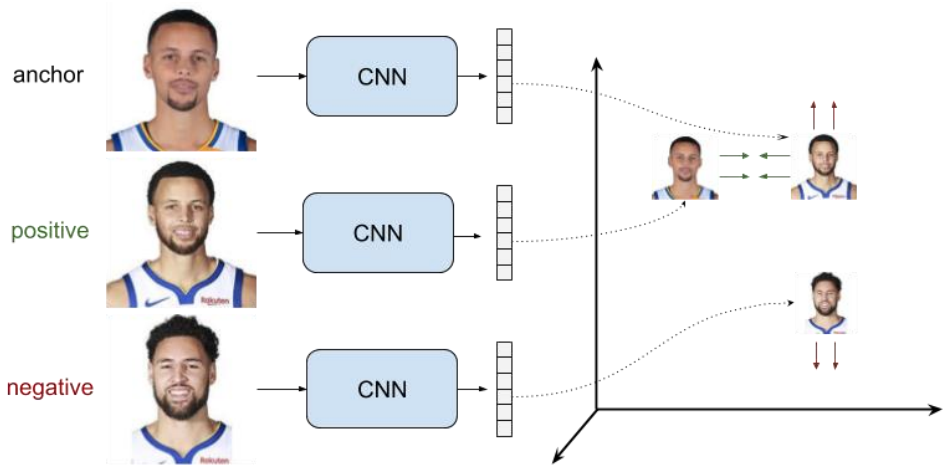


Figura 79. Ejemplo de Triplet Loss versus una imagen positiva y una negativa
Fuente: Gómez (2019). *Understanding Ranking Loss, Contrastive Loss, Margin Loss, Triplet Loss, Hinge Loss and all those confusing names*

Consecuentemente, se entrena la red para que cumpla con estas características. A esta distancia se le denomina *Triplet Loss* y se representa mediante la minimización del error de la siguiente función:

$$E = \sum_{i}^{N} \left[\left\| f(x_i^a) - f(x_i^p) \right\|_2^2 - \left\| f(x_i^a) - f(x_i^n) \right\|_2^2 + \alpha \right] \quad (46)$$

Donde x_i^a representa los valores vectoriales de características de la imagen actual, x_i^p representa los valores vectoriales de las imágenes positivas con los que la distancia se debe minimizar, x_i^n representa los valores negativos con los que la distancia se debe maximizar y α representa un margen de diferencia mínimo que debe existir entre valores positivos y negativos.

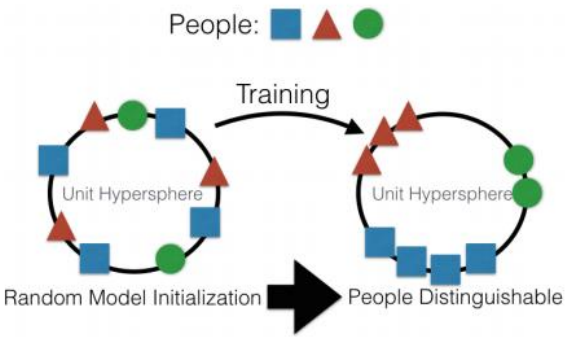


Figura 80. Triplet Loss, antes del entrenamiento y luego del entrenamiento
Fuente: Amos, Bartosz & Satyanaray (2016). *OpenFace: A general-purpose face recognition library with mobile applications.* (p. 4)

Por lo tanto, tal como se observa en la Figura 80, se inicia teniendo imágenes mezcladas al azar y se obtiene finalmente un vector de características por cada imagen que las separa. Este vector de características se encuentra en una hiperesfera unitaria, es decir, todos los vectores tendrán una distancia de 1.0 del origen (Amos, Bartosz, & Satyanaray, 2016).

2.2.4.2.9. Ejemplos de Modelos de Verificación Facial

En base a lo mencionado anteriormente, se describen tres modelos de verificación facial basados en redes neuronales convolucionales y que cuentan con implementaciones disponibles públicamente.

OpenFace

OpenFace tiene como objetivo hacer disponible al público un modelo de verificación facial con alta efectividad (Amos, Bartosz, & Satyanaray, 2016). Este modelo tuvo como referencias principales a *DeepFace* de Facebook (Taigman, Yang, Ranzato, & Wolf, 2014) y a *FaceNet* de Google (Schroff, Kalenichenko, & Philbin, 2015) al momento de construir su arquitectura. El entrenamiento estuvo basado en *Triplet Loss*, descrito en la sección anterior (2.2.4.2.8). Esta red neuronal convolucional se utilizó para extraer características generando vectores de 128 valores representativos de un rostro que luego podían ser comparados para realizar una verificación.

La arquitectura se muestra en la imagen a continuación:

type	output size	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj
conv1 ($7 \times 7 \times 3, 2$)	$48 \times 48 \times 64$						
max pool + norm	$24 \times 24 \times 64$						m $3 \times 3, 2$
inception (2)	$24 \times 24 \times 192$		64	192			
norm + max pool	$12 \times 12 \times 192$						m $3 \times 3, 2$
inception (3a)	$12 \times 12 \times 256$	64	96	128	16	32	m, 32p
inception (3b)	$12 \times 12 \times 320$	64	96	128	32	64	ℓ_2, 64p
inception (3c)	$6 \times 6 \times 640$		128	256,2	32	64,2	m $3 \times 3, 2$
inception (4a)	$6 \times 6 \times 640$	256	96	192	32	64	ℓ_2, 128p
inception (4e)	$3 \times 3 \times 1024$		160	256,2	64	128,2	m $3 \times 3, 2$
inception (5a)	$3 \times 3 \times 736$	256	96	384			ℓ_2, 96p
inception (5b)	$3 \times 3 \times 736$	256	96	384			m, 96p
avg pool	736						
linear	128						
ℓ_2 normalization	128						

Figura 81. Arquitectura de OpenFace
Fuente: Amos, Bartosz & Satyanaray (2016). *OpenFace: A general-purpose face recognition library with mobile applications.* (p. 18)

Esta arquitectura está en su mayoría basada en *FaceNet* con modificaciones en la cantidad de capas y tamaños de entrada facilitando que tenga menos parámetros (Amos, Bartosz, & Satyanaray, 2016). Como se puede observar, se utiliza una arquitectura basada en capas de convolución (*conv1*), capas de submuestreo (*max pool* y *avg pool*), capas *Inception* y capas totalmente conectadas (*linear*).

VGGFace2

VGGFace2 es el resultado de una investigación que busca crear una base de datos de rostros accesibles al público que sirvan como apoyo en investigaciones relacionadas a la verificación facial. Asimismo, dicha investigación tiene el propósito de generar un modelo de verificación facial de alta efectividad (Cao, Shen, Xie, Parkhi, & Zisserman, 2018). Esta investigación fue desarrollada por autores del *Visual Geometry Group* de Oxford y generó modelos sucesores a los creados en una investigación anterior, creadora de *VGGFace* (Parkhi, Vedaldi, & Zisserman, 2015). La nueva investigación generó los modelos *VGGFace2* creando nuevas arquitecturas y basándose en los lineamientos de la anterior investigación.

En esta investigación se crearon modelos con una mayor cantidad de datos y arquitecturas basadas en ResNet-50 con y sin bloques *SeNet* (*Squeeze-and-Excitation*). Principalmente se divide en 2 tipos de modelos:

- Modelo VGGFace2 basado en ResNet-50
- Modelo VGGFace2 basado en Se-ResNet-50

Ambas arquitecturas están basadas en bloques residuales. Las arquitecturas de bloques residuales buscan solucionar el problema de la pérdida de la gradiente a lo largo de redes con muchas capas. La pérdida de gradiente se debe a que, durante el *backpropagation*, multiplicar muchas veces los números por valores entre 0 y 1, tiende a volver cada vez más pequeña la gradiente cuando se propaga entre capa y capa. Esto genera que la gradiente tienda a 0 cuando la propagación llega a las primeras capas. Esta pérdida de gradiente genera como consecuencia pérdidas en efectividad. Para solucionarlo, se crearon los bloques residuales. A continuación, se muestra un ejemplo de bloque residual (He, Zhang, Ren, & Sun, 2015):

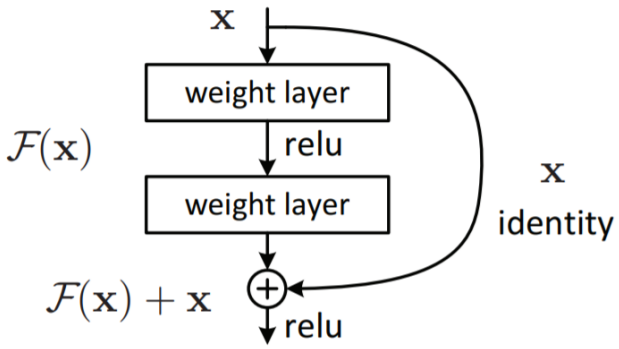


Figura 82. Ejemplo de bloque residual
Fuente: He, Zhang, Ren & Sun (2015). *Deep Residual Learning for Image Recognition*. (p. 2)

ResNet busca aliviar el problema incorporando conexiones directas que saltan una o más capas a lo largo de la red, estas conexiones se denominan de acceso directo y se suman al resultado residual F(x). Por lo tanto, en lugar de tener F(x) como salida de un bloque tradicional, se tiene como salida F(x) + x.

Adicionalmente, en el caso de Se-ResNet-50, se utilizan bloques *Squeeze-and-Excitation* (SeNet). Por defecto, la red neuronal convolucional le da la misma importancia a cada canal al momento de calcular las salidas de las convoluciones. SeNet busca darle una importancia o peso adaptable por canal. Esto se logra mediante una pequeña red neuronal que recibe como entrada los bloques convolucionales, los reduce mediante submuestro y los procesa por dos capas totalmente conectadas (Pröve, 2017). Este bloque tiene como resultado pesos que luego se aplican a cada uno de los canales de los bloques convolucionales.

A continuación, se muestra cómo estos bloques de *Squeeze-and-Excitation* se utilizan en una red de bloque residuales, en la red denominada SE-ResNet.

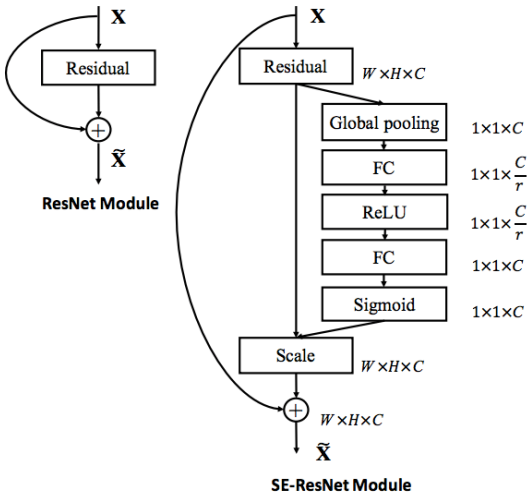


Figura 83. Ejemplo de ResNet y Se-ResNet
Fuente: Hu, Shen, lbanie, Sun & Wu (2018). *Squeeze-and-Excitation networks.* (p. 4)

Como se puede observar, el bloque SeNet se agrega como capa final luego del bloque residual y antes de calcular la salida para ir al siguiente bloque.

VGGFace2 está basado en los modelos ResNet-50 y Se-ResNet-50. A continuación, se muestra la arquitectura de referencia de ambos: ResNet-50 (izquierda) y Se-ResNet-50 (medio):

	Output size	ResNet-50	SE-ResNet-50	SE-ResNeXt-50 (32 × 4d)
	112 × 112	conv, 7 × 7, 64, stride 2		
	56 × 56	max pool, 3 × 3, stride 2		
Stage 1		$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 64 \\ \text{conv}, 3\times3, 64 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \quad C=32 \\ \text{conv}, 1\times1, 256 \\ fc, [16, 256] \end{bmatrix} \times 3$
Stage 2	28 × 28	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 128 \\ \text{conv}, 3\times3, 128 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \quad C=32 \\ \text{conv}, 1\times1, 512 \\ fc, [32, 512] \end{bmatrix} \times 4$
Stage 3	14 × 14	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 256 \\ \text{conv}, 3\times3, 256 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \quad C=32 \\ \text{conv}, 1\times1, 1024 \\ fc, [64, 1024] \end{bmatrix} \times 6$
Stage 4	7×7	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 512 \\ \text{conv}, 3\times3, 512 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$	$\begin{bmatrix} \text{conv}, 1\times1, 1024 \\ \text{conv}, 3\times3, 1024 \quad C=32 \\ \text{conv}, 1\times1, 2048 \\ fc, [128, 2048] \end{bmatrix} \times 3$
	1 × 1	global average pool, 1000-d fc, softmax		

Figura 84. Arquitectura de referencia para *VGGFace2*
Fuente: Hu, Shen, lbanie, Sun & Wu (2018). *Squeeze-and-Excitation networks.* (p. 5)

Las operaciones dentro de cada bloque residual se muestran entre las llaves y la cantidad de bloques se muestra con la multiplicación. Asimismo, estos se agrupan por etapas (*stages*) en los que se comparte la misma cantidad de pesos a aprender. Además, como se observa en la arquitectura, en el caso de los bloques SeNet, estos se referencian como capas totalmente conectadas finalizando cada bloque residual.

Light CNN

Light CNN es un modelo de redes neuronales convoluciones que buscó aplicar características que permitan la reducción del ruido. Por lo tanto, este modelo plantea la operación denominada *Max-Feature-Map* (MFM) que busca depurar neuronas en cada capa de tal forma que se separe el ruido de lo útil (Wu, He, Sun, & Tan, 2018). La idea central del *Max-Feature-Map* es filtrar las matrices de características en base a las que tienen valores máximos. Esto es otra forma de prevenir el problema de la gradiente que tiende a 0.

A continuación, se muestra una arquitectura de referencia de *Light CNN*.

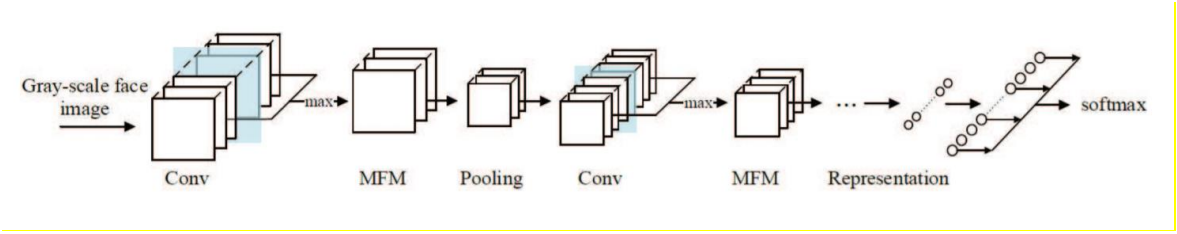


Figura 85. Arquitectura de Light CNN
Fuente: Wang & Deng (2018). *Deep Face Recognition: A Survey*

Como se puede observar, la arquitectura se basa principalmente en capas de convolución, *max-pooling*, capa totalmente conectada y MFM. Un dato importante es que *Light CNN* recibe como entrada imágenes a escala de grises, a diferencia de otros modelos que se basan en imágenes con tres canales de color.

Asimismo, la red también utilizó bloques residuales para mejorar la performance. En la investigación de *Light CNN* se desarrollaron 3 modelos: Light CNN-4, Light CNN-9 y Light CNN-29. El más avanzado es el Light CNN-29 que contiene 29 capas en la red con 12,637 miles de parámetros.

En conclusión, las redes neuronales convolucionales son una extensión de las redes neuronales tradicionales. Estas agregan ciertas características especiales que permiten el procesamiento adecuado de las imágenes. Asimismo, pueden ser utilizadas tanto para la extracción de características como la clasificación o ambas cosas.

2.2.5. Visión Computacional

En la visión computacional se busca describir el mundo que vemos en una o más imágenes y reconstruir sus características, como la forma, la iluminación y las distribuciones de color.

Esto es algo que el ser humano puede hacer de manera natural pero que dentro de un algoritmo de visión por computador se vuelve complejo y propenso a errores (Szeliski, 2010). Asimismo, como menciona Klette (2010), esta disciplina tiene como objetivo utilizar cámaras para analizar o comprender escenas en el mundo real. Por otro lado, según Sucar y Gómez (2008), la visión computacional es el estudio de los procesos de reconocer y localizar objetos usando el procesamiento de imágenes de tal forma que se logre un mayor entendimiento de estos. Por lo tanto, la visión computacional busca el entendimiento, análisis y descripción de lo visto en imágenes del mundo real para poder, posteriormente, aplicar lo comprendido hacia algún objetivo.

Por otro lado, de acuerdo con Szeliski (2010), el preprocesamiento de imágenes es el primer paso en la mayoría de las aplicaciones de visión computacional. Esto se realiza para procesar la imagen y convertirla en un resultado que sea adecuado para realizar mayor análisis. Algunas de estas operaciones fueron evaluadas en la sección 2.2.1, como el afinamiento, la reducción del ruido o suavizamiento y las correcciones de color como la ecualización.

Adicionalmente, la visión computacional estudia problemas metodológicos y algorítmicos, así como temas relacionados con la implementación de soluciones relacionadas. Por ejemplo, esta hace posible analizar la distancia de un edificio hacia una cámara, detectar si un vehículo está dentro de su carril, reconocer personas o contar la cantidad de personas en una escena (Klette, 2014). Por lo tanto, la visión computacional es muy amplia y abarca distintos campos de aplicación y estudio, entre estos campos se encuentran los siguientes (Szeliski, 2010):

- Reconocimiento de Caracteres (OCR): por ejemplo, aplicado en el reconocimiento de códigos escritos a mano o placas de automóviles.
- Imágenes Médicas: aplicado en el análisis de imágenes médicas preoperatorias e intraoperatorias.
- Construcción de modelos 3D a partir de imágenes
- Reconocimiento de objetos en *retail*: el reconocimiento de productos en las colas de pago.
- Seguridad vehicular: aplicado en la detección de obstáculos en las calles como los peatones.
- Videovigilancia: se puede aplicar para el monitoreo de intrusos, análisis de tráfico en la carretera y grupos de monitoreo para víctimas de ahogamiento;
- Detección y Reconocimiento Facial, así como la autenticación facial en base a la verificación del rostro.

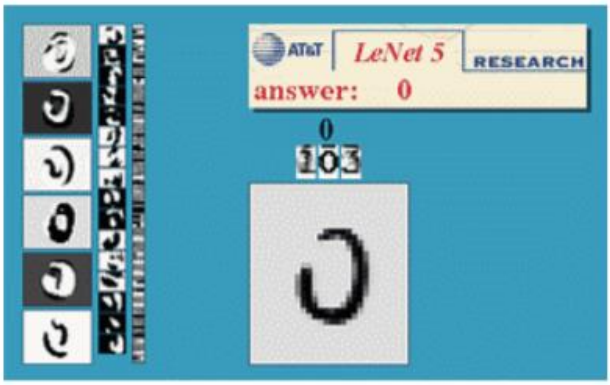


Figura 86. Ejemplo de reconocimiento de caracteres (OCR)
Fuente: Szeliski (2010). *Computer Vision: Algorithms and Applications.* (p. 6)

Estos son solo algunos de los campos dentro de los cuales se puede aplicar la visión computacional. En la investigación, se aplica la visión computacional mediante la detección y verificación de rostros.

En los últimos años, la visión por computadora ha convertido en una tecnología clave en muchos campos para productos de consumo modernos. Esto incluye aplicaciones para teléfonos móviles, asistencia para conducir automóviles o interacción del usuario con juegos de computadora. Otro ejemplo es la automatización industrial, donde la visión por computadora se usa habitualmente para el control de calidad o de procesos (Klette, 2014).

Adicionalmente, en la Figura 87, se observa un resumen del desarrollo de la visión computacional a través del tiempo.

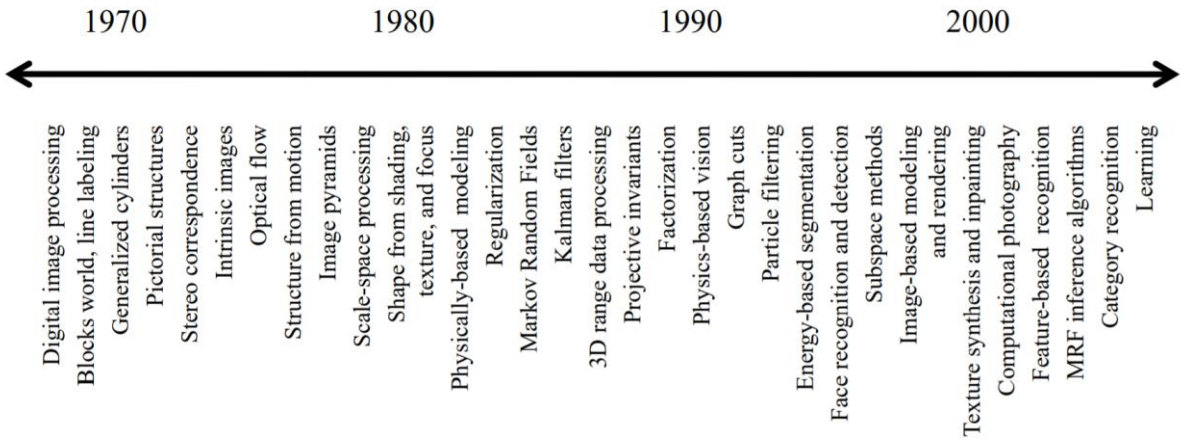


Figura 87. Historia de la Visión Computacional
Fuente: Szeliski (2010). *Computer Vision: Algorithms and Applications.* (p. 11)

Como se puede observar, la visión computacional se ha desarrollado a través de los años tomando diferentes aplicaciones. Inicialmente se transformó del procesamiento de imágenes digitales, a técnicas que buscaban obtener estructuras del mundo real en 3D desde las imágenes.

Esto incluye, por ejemplo, las técnicas de etiquetado de líneas (*line labeling*) para detectar bordes en las imágenes e inferir una visión 3D del mundo (Szeliski, 2010). Posteriormente, en los noventas surgen aplicaciones como el reconocimiento y la detección facial. Estas técnicas seguirían evolucionando hasta la actualidad con nuevos modelos (Amos, Bartosz, & Satyanaray, 2016; Taigman, Yang, Ranzato, & Wolf, 2014).

Luego, en el siglo XXI, nacieron nuevas áreas de investigación como el campo de fotografía computacional, esto ayudó a realizar mejoras en imágenes digitales mediante algoritmos como, por ejemplo, las imágenes de alto rango dinámico (HDR). Adicionalmente, surgieron las técnicas basadas en descriptores de características para detección de objetos (*Feature-based recognition*). Finalmente, empezaron a investigarse nuevos campos basados en el aprendizaje automático (*machine learning*) (Szeliski, 2010). El aprendizaje automático posteriormente dio paso al aprendizaje profundo o *deep learning*.

En la actualidad, las técnicas de *deep learning* han tomado mayor importancia dentro de la visión computacional. En los últimos años, se ha demostrado que los métodos de *deep learning* superan a las técnicas de aprendizaje automático de última generación en varios campos, siendo la visión por computadora uno de los casos más destacados. Los modelos de *deep learning* aplicados a visión computacional incluyen las redes neuronales convolucionales, las máquinas de Boltzmann (DBM) y redes de creencia profunda (DBN) (Voulodimos, Doulamis, Doulamis, & Protopapadakis, 2018).

El *deep learning* ha impulsado considerablemente el desarrollo de la visión computacional en aplicaciones como la detección de objetos, el seguimiento de movimiento, la detección de poses, entre otros. En el caso de las redes neuronales convolucionales, estas han sido muy útiles en aplicaciones de visión por computadora, tales como reconocimiento facial, detección de objetos, robótica y vehículos autónomos (Voulodimos, Doulamis, Doulamis, & Protopapadakis, 2018).

Finalmente, como se puede observar, el campo de la visión computacional es bien amplio e incluye aplicaciones en diversos sectores: industriales, de consumo masivo, *retail*, automóviles, entre otros. Es un campo con áreas de investigación y de aplicación muy diversas. Asimismo, sus algoritmos han ido evolucionando considerablemente en el tiempo, pero manteniendo la premisa básica de buscar reconstruir y comprender las características del mundo real a partir de imágenes. La investigación se centra en una parte de este universo, específicamente en su aplicación para la detección y verificación facial.

2.2.6. Metodología CRISP-DM

La metodología CRISP-DM es una metodología de Minería de Datos considerada la más utilizada dentro del campo de analítica, ciencias de datos y minería de datos (Piatetsky, 2014). Esta fue propuesta en 1999 por un consorcio de varias empresas europeas entre las que se encontraban SPSS, NCR, Teradata y Daimer Chrysler (Gallardo, 2009). Ellos buscaban una metodología que sea abierta para que pueda ser evolucionada constantemente por la comunidad. Como se observa en la Figura 88, la metodología cuenta con 6 fases que pueden iterar entre sí en ciertos puntos del proceso.

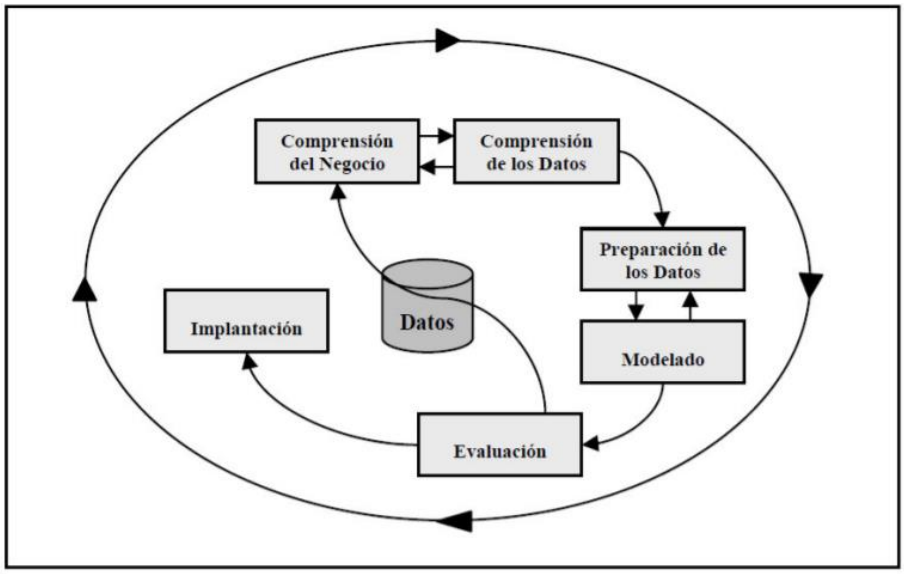


Figura 88. Etapas de Metodología CRISP-DM
Fuente: Gallardo (2009). *Metodología para la Definición de Requisitos en Proyectos de Data Mining (ER-DM).* (p. 17)

Las fases se describen a continuación (Gallardo, 2009):

- Comprensión del Negocio: Busca comprender los objetivos o requisitos del proyecto desde la perspectiva de la empresa o institución. Se realiza la comprensión del contexto en el que se desarrolla el problema y se evalúa la situación actual.
- Comprensión de Datos: Se realiza la recolección, descripción y análisis de los datos. Se verifica la calidad de los datos.
- Preparación de Datos: Se seleccionan, limpian y estructuran los datos de la fase anterior. Esto puede llevar al formateo de los datos, integración o creación de nuevas variables.
- Modelado: Se escogen las técnicas y herramientas a utilizar, así como la creación de los modelos.

- Evaluación: Se realizan las pruebas en el modelo tomando en consideración los criterios de éxito que se van a evaluar. Asimismo, se evalúa el proceso seguido planteando mejoras que pudieran realizarse. Finalmente, se decide si se continúa a la siguiente fase o se regresa a una fase anterior.
- Implementación: Se realiza la entrega del sistema con los modelos validados. Esto incluye la implementación, mantenimiento y revisión del proyecto.

2.2.7. Metodología de Desarrollo de Software Cascada

El modelo de desarrollo cascada fue propuesto en 1970 por Winston Royce (Bassil, 2012). Es la metodología más tradicional de desarrollo de software y define lo que se denomina el ciclo de vida del software. El ciclo de vida plantea las fases de análisis, diseño, construcción, pruebas y mantenimiento. Las fases se describen a continuación (Bassil, 2012):

- Análisis: Se realiza el análisis de requerimientos funcionales y no funcionales del cliente. Se intenta conocer lo que se espera del sistema. Esto se documenta mediante entregables que pueden depender de cada organización. Un ejemplo de estos son los casos de uso.
- Diseño: Se realiza el diseño de la arquitectura. Esto puede incluir el modelamiento de la base de datos y la interrelación entre los componentes del sistema. Asimismo, puede incluir los diseños de interfaces de usuario.
- Construcción: En esta etapa se desarrolla el sistema que se desea implementar.
- Pruebas: En esta etapa se realizan las pruebas para verificar que el sistema no tenga errores. Existen diversos tipos de pruebas. Algunas de las más comunes son las pruebas unitarias, las de aceptación y las de sistema.
- Mantenimiento: En esta etapa realizan mantenimientos o modificaciones del sistema una que ha sido vez puesto en producción.

2.2.8. Metodología de Desarrollo de Software Iterativo

La metodología de desarrollo iterativo se basa en el modelo tradicional de cascada (Mohammed, Munassar, & Govardhan, 2010). La diferencia principal consiste en que la metodología de desarrollo iterativo realiza el ciclo de vida en varias iteraciones. Cada iteración del proceso tiene como resultado un producto funcional. Además, esto permite obtener opiniones de los usuarios prontamente acerca de la situación del proyecto. Esto brinda más

flexibilidad en el desarrollo de los requerimientos mediante desarrollos incrementales (Mohammed, Munassar, & Govardhan, 2010).

En la Figura 89 se ilustra la metodología de desarrollo iterativo.

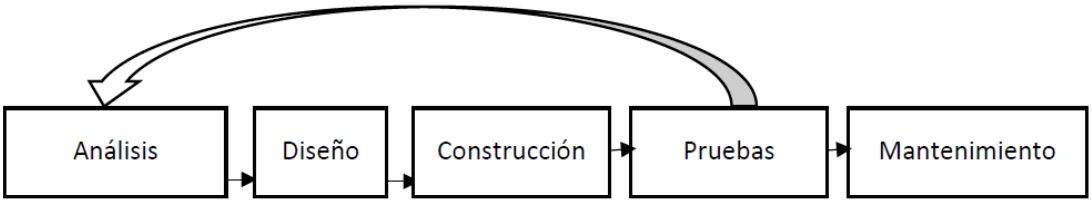


Figura 89. Metodología de Desarrollo de Software Iterativo
Fuente: Bassil (2012). *A Simulation Model for the Waterfall Software Development Life Cycle.* (p. 2)

Las fases son las mismas que en la metodología cascada pero se regresa de forma iterativa a la etapa de análisis y para realizar entregables más pequeños.

2.3. Contexto de la Investigación

El gran incremento en la cantidad de imágenes accesibles libremente en Internet y el incremento en las capacidades computacionales, ha facilitado la creación de mejores modelos estadísticos, los cuales han perfeccionado los sistemas de visión computacional. En el 2014, se mencionaba cómo los ratios de error de estos modelos habían decrecido en los últimos 20 años en tres órdenes magnitud (Taigman, Yang, Ranzato, & Wolf, 2014). Han existido grandes avances en la mejora de este tipo de modelos que apoyan a la verificación y reconocimiento facial. Parte de estos avances se debe a que las redes neuronales convolucionales han demostrado mayores capacidades que los algoritmos tradicionales al momento de procesar grandes cantidades de imágenes. Por lo tanto, el contexto actual para este tipo de modelos se encuentra en constante desarrollo.

En el ámbito global, las tecnologías basadas en reconocimiento facial se están aplicando en varios países, siendo la seguridad ciudadana uno de los campos principales. Un ejemplo notorio de su aplicabilidad es el caso de China donde se instalaron 200 millones de cámaras de video vigilancia con esta tecnología. Estados Unidos es otro país que apostó por esta tecnología en las calles como medida de seguridad (Mozur, 2018).

Otros ejemplos están en el ámbito de las redes sociales. Facebook, por ejemplo, sugiere etiquetar a amigos en base a su rostro en las fotos utilizando reconocimiento facial (Sigal, 2019). El reconocimiento para etiquetados en fotos también se observa en repositorios de fotos

como Amazon Photos y Google Photos buscando facilitar encontrar fotos de una o varias personas en los álbumes.

En el ámbito peruano, uno de los avances más importantes dentro de la verificación o reconocimiento facial ha sido su aplicación por la RENIEC. Este sirve como complemento a su sistema de seguridad biométrica decadactilar que compara los 10 dedos de la mano. Por otro lado, la Policía Nacional del Perú también lo utiliza como apoyo para propósitos criminalísticos (RENIEC, 2014). Sin embargo, dado que la verificación facial está en constante crecimiento, estos sistemas se deben transformar continuamente buscando formas de aumentar su efectividad y así generar una mayor seguridad en su veracidad.

CAPÍTULO III: METODOLOGÍA DE INVESTIGACIÓN

A continuación, se describe el diseño, población y muestra. Además, se explica la metodología de implementación, la cual determina cómo se abordará el problema y las etapas que se siguieron. Asimismo, se describe el cronograma de la implementación.

3.1. Diseño de Investigación

3.1.1. Diseño

La investigación es experimental al tener control en las variables y poder analizar su relación. Además, es una investigación transversal al realizarse en un mismo periodo sin depender de cambios en el tiempo.

3.1.2. Tipo

La investigación es de tipo proyecto solución al generar dos activos que cumplen con un propósito:

- El prototipo desarrollado y utilizado para la recolección de datos
- El prototipo utilizado para las pruebas automatizadas

Asimismo, como parte de la investigación se generan aportes de tipo correlacional al buscar la relación entre la aplicación de diversos métodos de preprocesamiento y la efectividad de la verificación facial, así como el impacto de la fuente de información, el método de detección y el modelo de verificación facial.

3.1.3. Enfoque

La investigación tiene un enfoque cuantitativo al estar centrada en variables numéricas y métodos estadísticos que apoyen en establecer las relaciones planteadas.

3.1.4. Población y Muestra

La población del estudio son todas las imágenes obtenidas para las pruebas a realizar de las tres fuentes de información definidas:

- *Labeled Faces in the Wild* (LFW): Obtenida desde la web oficial de la investigación, se utilizó las imágenes sin preprocesar dado que existe una versión con las imágenes previamente alineadas: http://vis-www.cs.umass.edu/lfw/. Compuesta por 13233 imágenes de rostros obtenidas en entornos no controlados de 5749 personas (Huang, Ramesh, Berg, & Learned-Miller, 2007).
- *YouTube Faces Database*: Obtenida desde la web oficial de la investigación https://www.cs.tau.ac.il/~wolf/ytfaces/. Conformada por imágenes de rostros obtenidas de 3425 videos de Youtube de 1595 personas (Wolf, Hassner, & Maoz, 2011).
- Base de Datos del contexto local (DNI): Parejas de imágenes de DNI y rostro de cada persona obtenida en base al prototipo de investigación desarrollado al cual ingresaron personas de Lima, Perú para registrar sus datos y subir sus fotos. El detalle del prototipo se observa en el Capítulo IV. Compuesta por 102 imágenes de 51 personas.

El muestreo para las pruebas de cada base de datos se realizó de manera probabilística. Debido a que los modelos están preentrenados, se utilizará una metodología recomendada para la medición de los modelos descrita en la sección 3.3. A continuación se detallan las muestras por cada base de datos.

- En el caso de *Labeled Faces in the Wild* (LFW) se obtuvo un subconjunto de probabilísticamente elegido para las pruebas y definido en el mismo *dataset*. Este subconjunto tiene 20 grupos de 300 parejas de imágenes, mitad de la misma persona y mitad de diferentes personas. Esto totaliza 6,000 parejas de imágenes. Sobre esta muestra se realizaron las pruebas mediante *cross-validation* de acuerdo con lo descrito en la sección 3.3.
- En el caso de *YouTube Faces Database* se obtuvo un subconjunto de probabilísticamente elegido para las pruebas siguiendo lo definido como parejas de videos para pruebas en el *dataset*. Esta definición tiene 5,000 parejas de videos al azar, mitad de la misma persona y mitad de diferentes personas. Para cada pareja, se eligió una imagen (fotograma) al azar de cada video y se compararon entre sí. Esto dio un

total de 5,000 parejas de imágenes. Sobre esta muestra se realizaron las pruebas mediante *cross-validation* de acuerdo con lo descrito en la sección 3.3.

- En el caso de la base de datos del contexto local, se obtuvieron todas las combinaciones de parejas entre las imágenes de DNI y las de rostro y, debido a su tamaño, se utilizó en su totalidad para las pruebas. Un total de 2,601 parejas de imágenes. Sobre esta muestra se realizaron las pruebas mediante *cross-validation* de acuerdo con lo descrito en la sección 3.3.

3.1.5. Operacionalización de Variables

A continuación, se muestra la tabla de operacionalización de variables para la investigación.

Tabla 3.

Matriz de Operacionalización de Variables

Variable	Indicadores	Cálculo / Valores
Efectividad del Modelo	Exactitud	(Verdaderos Positivos + Verdaderos Negativos) / Total
	Precisión	Verdaderos Positivos / (Verdaderos Positivos + Falsos Positivos)
	Ratio de Verdaderos Positivos (Recall o Sensibilidad)	Verdaderos Positivos / (Verdaderos Positivos + Falsos Negativos)
	Ratio de Falsos Positivos	Falsos Positivos / (Verdaderos Negativos + Falsos Positivos)
	Curva ROC y Área Bajo la Curva (AUC)	AUC= P(score(x^+) > score(x^-))
Método de Preprocesamiento	Alineamiento	- Sin Alineamiento - Con Alineamiento de Ojos - Con Alineamiento de Ojos + Nariz
	Filtro Gausiano	- Con Filtro Gausiano 5x5 - Con Filtro Gausiano 3x3 - Sin Filtro Gausiano
	Filtro de Agudizamiento	- Con Filtro de Agudizamiento al 5% - Con Filtro de Agudizamiento al 10% - Con Filtro de Agudizamiento al 15% - Con Filtro de Agudizamiento al 20% - Sin Filtro de Agudizamiento
	Filtro de Ecualización	- Con Ecualización en RGB - Con Ecualización en YCbCr - Con Ecualización Local YCbCr - Con Ecualización Local HSV - Sin Ecualización

Fuente de Información	Base de Datos	- DNI - LFW - YouTube Faces DB
Método de Detección facial	Filtro detector de rostros	- Con descriptor HOG - Con descriptor Haar
Modelo de Verificación Facial	Modelo Pre-entrenado	- OpenFace - VGGFace2 - Light CNN

El trabajo busca analizar la relación entre los métodos de preprocesamiento y la efectividad de los modelos de verificación facial. Asimismo, pretende analizar el impacto en caso de variaciones en la fuente de información, el método de detección facial y el modelo de verificación facial aplicado.

En primer lugar, se tienen los indicadores relacionados con la efectividad del modelo estos fueron descritos en la sección 2.2.2. En este caso se tiene la exactitud, la precisión, el ratio de verdaderos positios, el ratio de falsos positios y la curva ROC. Por otro lado, referente a los métodos de preprocesamiento, se tienen varios métodos posibles de acuerdo con la teoría de la sección 2.2.1. La variable medirá si se aplicó o no el preprocesamiento y con qué intensidad o tipo. En el caso del alineamiento se probará con alineamiento de sólo los ojos, alineamiento de ojos y nariz y sin alineamiento. En el filtro gausiano se evaluarán filtros 5x5, 3x3 y sin filtro. Para el agudizamiento se evaluará realizarlo al 5%, 10%, 15%, 20% y sin agudizamiento. En el caso del filtro de ecualización se analizará con ecualización RGB, YCbCr, local YCbCr, local HSV y sin ecualización.

Asimismo, respecto a la fuente de información se compararán los resultados en las 3 bases de datos distintas mencionadas: LFW, YTF y la base de datos del contexto local (DNI). Finalmente, en el método de detección de rostros se comparará HOG y Haar como extractores de características y en el modelo de verificación facial se evaluarán los modelos de *OpenFace*, *Light CNN* y *VGGFace2*.

3.2. Metodología de Implementación

Para la metodología de implementación se tomó en cuenta tanto la perspectiva de analítica de datos como la perspectiva de desarrollo de software dado que ambos tuvieron implicancias en el desarrollo del proyecto planteado.

Por el lado de analítica de datos se decidió utilizar CRISP-DM. Esta metodología forma parte de diversos estudios acerca de este tipo de metodologías y es una de las más populares.

En la Figura 90 se muestra un comparativo de CRISP-DM con otras metodologías para la Gestión de Proyectos de Ciencia de Datos (Saltz, Shamshurin, & Crowston, 2017). Se evaluaron 3 factores: resultados del proyecto, la satisfacción de los individuos de cada equipo y la intención de los individuos de volver a trabajar juntos en otros proyectos.

En la variable de resultados de proyecto, CRISP-DM fue la mejor como se muestra a continuación.

Section	Average Score (1 to 10; 10 is best)
Agile Scrum	6.5
Agile Kanban	7.8
CRISP	8.4
Baseline	7

Figura 90. Comparativo Metodologías Gestión de Proyectos de Ciencia de Datos
Fuente: Saltz, Shamshurin & Crowston (2017). *A Comparing Data Science Project Management Methodologies via a Controlled Experiment*. (p. 1019)

Asimismo, CRISP-DM obtuvo mejores resultados en la satisfacción individual en cada equipo. Agile Kanban fue superior en la variable de colaboración entre los individuos y su intención de volver a trabajar en equipo (Saltz, Shamshurin, & Crowston, 2017).

Adicionalmente, a continuación, se muestra una encuesta realizada acerca de la popularidad de diversas metodologías para analítica de datos, minería de datos o proyectos de ciencia de datos donde se observa que CRISP-DM se ubica en primer lugar (Piatetsky, 2014):

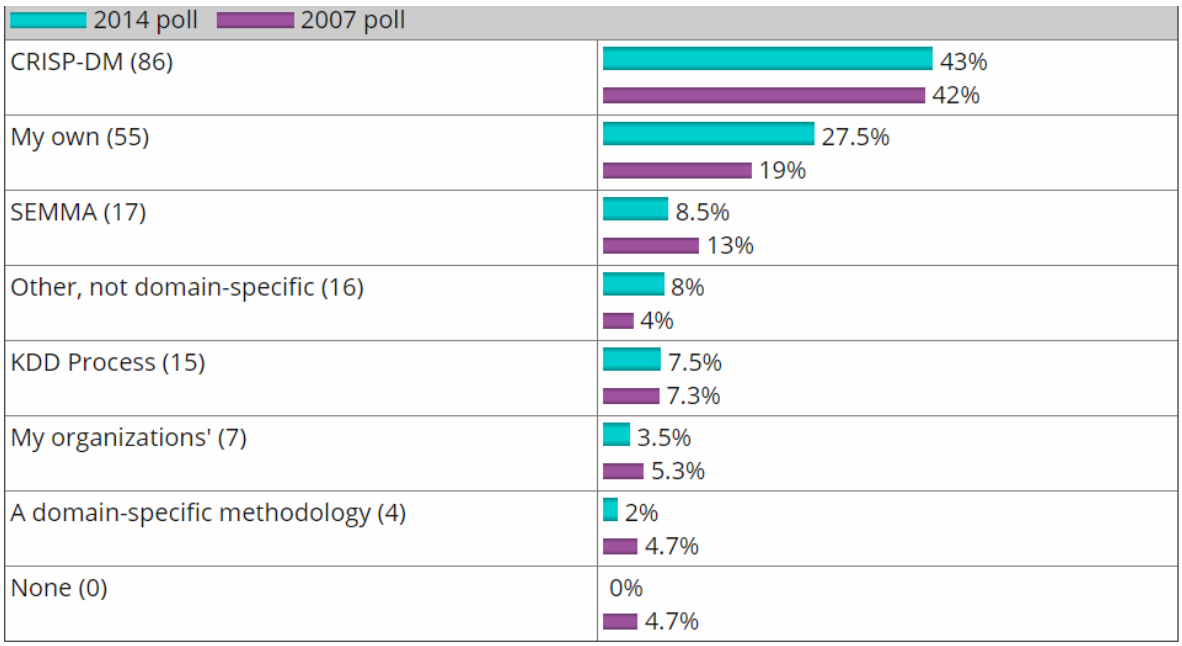


Figura 91. Popularidad de Metodologías de Analítica de Datos
Fuente: Piatesky (2014). *CRISP-DM, still the top methodology for analytics, data mining, or data science projects*

Por otro lado, la metodología CRISP-DM como cualquier metodología tiene sus limitaciones. Es importante denotarlas. En primer lugar, tiende a demandar tiempo empezar a ver resultados en un proyecto, pudiéndose extender las etapas iniciales. Por ejemplo, en el estudio de Saltz, Shamshurin y Crowston (2017), se mencionó que los individuos se demoraron mucho en empezar a codificar viéndose obligados a agilizar el paso en las siguientes etapas.

Otro factor importante detectado en CRISP-DM es que, si bien muchas veces involucra código, este no toma en cuenta los objetivos de desarrollo de software como principales. Estos objetivos son asegurar la calidad del código y asegurar la mantenibilidad del código (Perez, 2017). No se abordan temas de diseño, arquitectura de software ni división de componentes para estos casos. Esto se debe a que para el desarrollo de un sistema existen otras metodologías, las cuales son llevadas a cabo por personas con roles diferentes.

La relación entre la metodología CRISP-DM y las metodologías de desarrollo de software tradicionales se puede ilustrar de la siguiente manera (Haller, 2020).

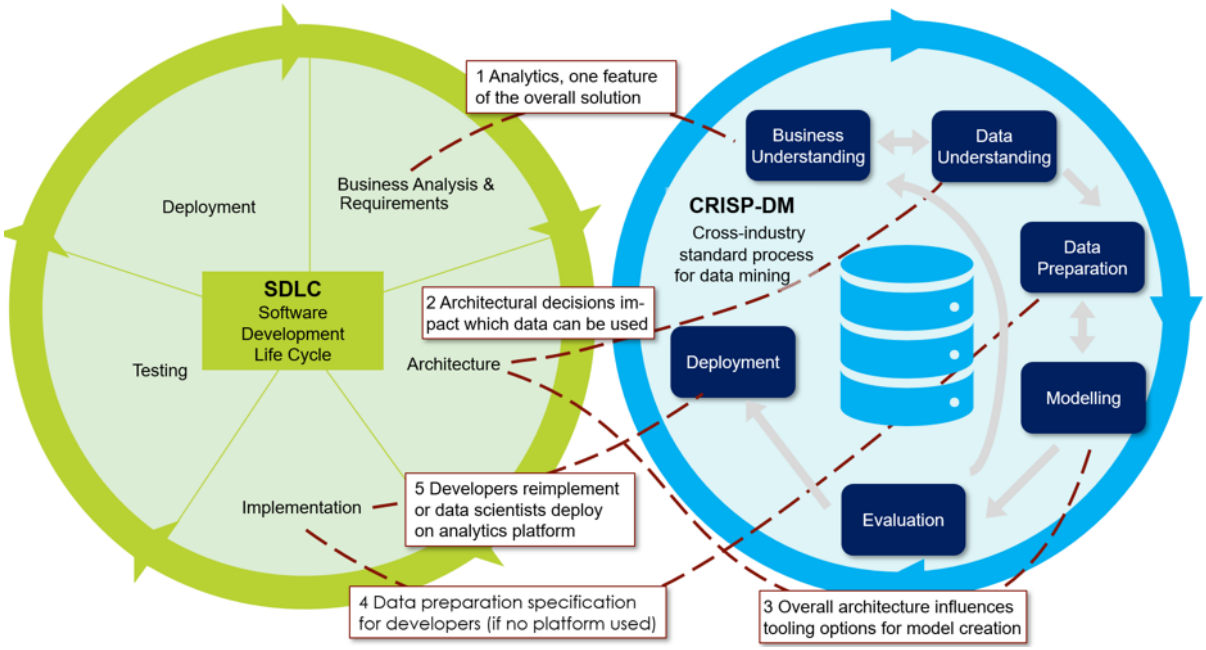


Figura 92. Interacción entre CRISP-DM y el ciclo de desarrollo de software
Fuente: Haller (2020). *Making CRISP-DM Work for Embedded Analytics*

Estas son dos metodologías diferentes con propósitos distintos pero que pueden trabajar en conjunto en un solo proyecto. Tienen similitudes en la fase de análisis de requerimientos de negocio para el desarrollo de una solución. Asimismo, tienen fases que se complementan. Este es el caso de la arquitectura del sistema. El diseño de arquitectura en un desarrollo de software

impacta en la data que se utilizará en el modelo. Además, la implementación de la solución de desarrollo va a verse afectada por la definición de la data así como la manera en la que se ha desplegado el modelo.

Dentro de desarrollo de software, la metodología cascada es la más tradicional y la base para muchas otras nuevas metodologías. Además, es la más utilizada en los proyectos de desarrollo de software, esto se muestra en el siguiente estudio (Vijayasarathy & Butler, 2016):

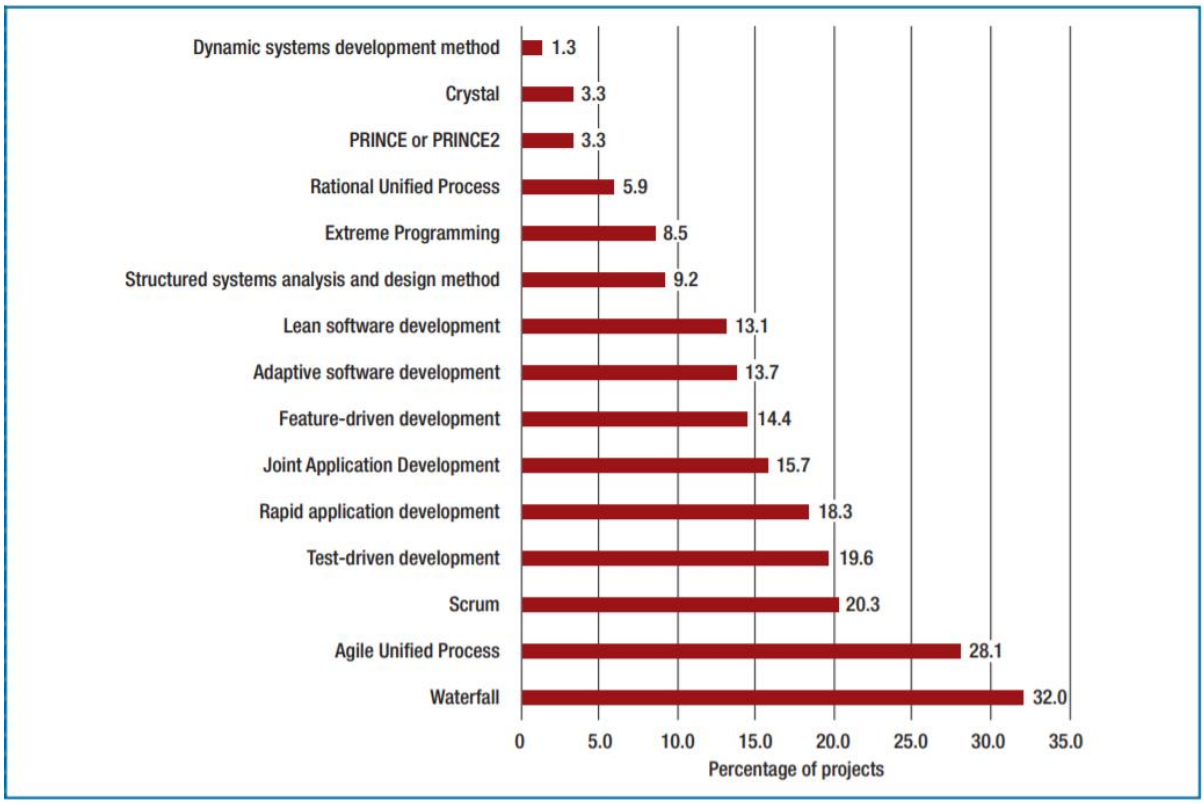


Figura 93. Popularidad de Metodologías de Desarrollo de Software
Fuente: Vijayasarathy & Butler (2016). *Choice of Software Development Methodologies. Do Organizational, Project and Team Characteristics Matter?*. (p. 90)

Asimismo, el estudio destaca que, si bien muchos proyectos pueden utilizar la metodología cascada, se ha encontrado que varios de estos proyectos siguen un modelo iterativo o incremental incluso dentro de las metodologías tradicionales. Esto demuestra que, si bien se puede estar usando la metodología de desarrollo de software cascada, pueden existir ciertos elementos iterativos al momento de su aplicación en las empresas y en la industria.

Sin embargo, como menciona el estudio, la elección de la metodología dependerá de aspectos organizacionales, de proyecto y del equipo. Por ejemplo, una metodología como Scrum está orientada al trabajo de equipos con roles bien marcados dentro de los participantes y con constantes reuniones con el cliente. Consecuentemente, debido a que este estudio es un

desarrollo realizado por una sola persona y con requerimientos bien definidos desde un inicio, se eligió la metodología tradicional de desarrollo de software cascada.

Finalmente, en la literatura se puede destacar el uso de ambas metodologías. Por ejemplo, CRISP-DM es usado en investigaciones relacionadas con reconocimiento de imágenes (Regalado, 2019), análisis de videos en tiempo real (Schulte, 2013) y análisis y reconocimiento de emociones en videos (Narváez, 2016). Todas estas relacionadas con el procesamiento de visión computacional y analítica. Esto se refuerza mediante su orientación a procesos analíticos en los que se deben generar resultados estadísticos a partir del análisis de modelos o algoritmos. Esta metodología suele utilizarse para desarrollar modelos estadísticos que brinden resultados desde la perspectiva de la ciencia de datos y no profundiza mucho en los aspectos prácticos de una implementación de un sistema real desde el lado del software, dado que suelen cumplir propósitos diferentes.

Por otro lado, en el caso de la metodología de desarrollo de software cascada, de igual manera, al ser una metodología popular esta es utilizada en distintos tipos de sistemas. Dentro de los campos relacionados, existen investigaciones desarrollando sistemas biométricos para clasificación de grupos étnicos (Xicali, 2019) y desarrollo de sistemas de reconocimiento facial para vigilancia (Al-Modwahi, Sebetela, Batleng, Parhizkar, & Lashkari, 2012). Estas investigaciones suelen entrar al detalle de la implementación del sistema en sí de manera práctica como se realiza también en esta investigación. Describen los requerimientos funcionales y no funcionales, la arquitectura, el diseño y la parte técnica de la construcción mediante librerías y código personalizado.

Por lo tanto, en la investigación se definió utilizar la metodología CRISP-DM para las actividades de analítica de datos y la metodología cascada para las actividades de desarrollo de software con componentes iterativos. Ambas con gran popularidad en su rubro y con aplicaciones en investigaciones relacionadas.

3.2.1. Metodología de Analítica de Datos

En esta sección se describirá la metodología de analítica de datos aplicada desde una perspectiva teórica. En el capítulo IV, se discutirá el desarrollo de la solución desde la perspectiva práctica de su construcción mediante ejemplos y descripción de las herramientas utilizadas.

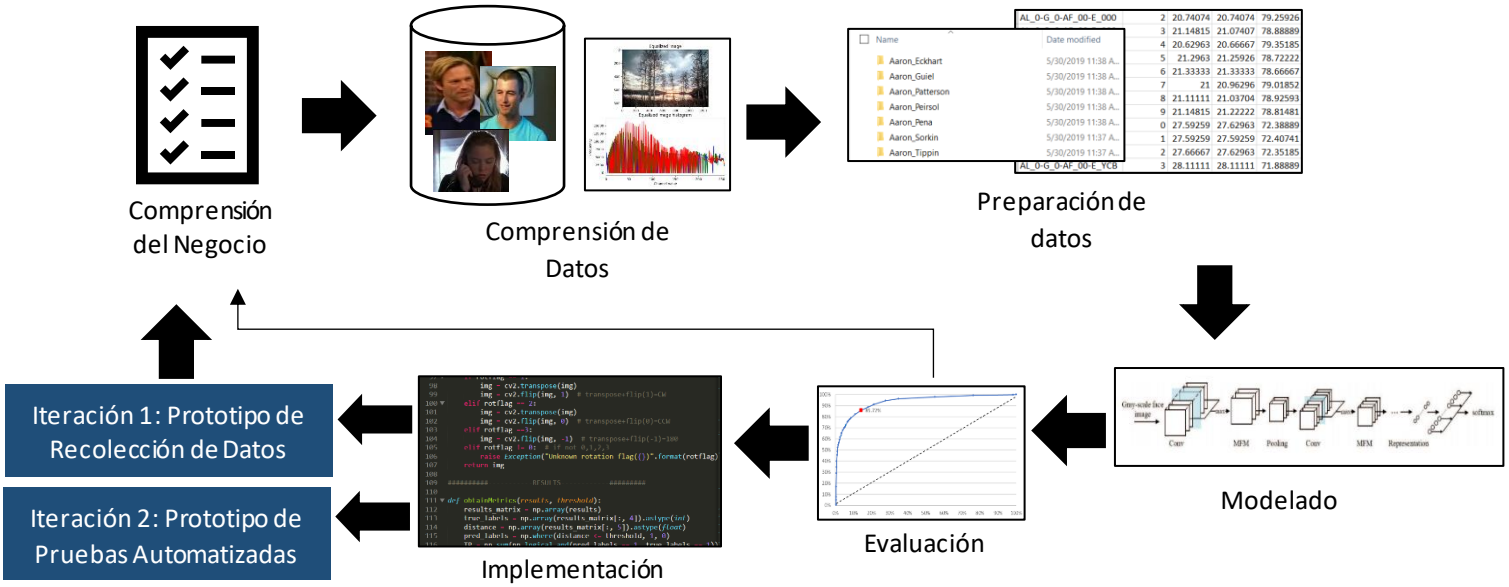


Figura 94. Metodología de Analítica de Datos
Fuente: Elaboración Propia

En la Figura 94 se muestra gráficamente la metodología que está basada en CRISP-DM y se describe en esta sección.

El proceso mostrado se aplicó de forma iterativa para apoyar en la generación de los dos productos o prototipos principales:

- Iteración 1: El prototipo desarrollado y utilizado para la recolección de datos del dataset de DNIs y como prototipo inicial de verificación facial.
- Iteración 2: El prototipo utilizado para las pruebas automatizadas que permiten evaluar las hipótesis de la investigación.

En la primera iteración se realizaron las etapas de analítica de datos para el primer prototipo que tuvo la finalidad de ser utilizado para la recolección de la fuente de información de DNIs. Este luego fue convertido en un prototipo de desarrollo de software. Los datos recolectados por este primer desarrollo se utilizaron en la segunda iteración como una de las tres bases de datos a evaluar. Asimismo, este primer prototipo sirvió como primera implementación de un proceso estándar de verificación facial completo mediante uno de los modelos evaluados (*OpenFace*).

En la segunda iteración se realizaron las etapas de analítica de datos para el prototipo de pruebas automatizadas que permite probar las hipótesis de la investigación. Cabe mencionar que este no es un proceso estático dado que varias veces se dieron saltos entre fases en búsqueda de las mejores alternativas y soluciones para la investigación.

3.2.1.1. Comprensión de Negocio

En la fase de comprensión del negocio se buscó entender el contexto del problema y los factores principales a desarrollar. En este caso, lo primero fue tener claro los objetivos del proyecto. Estos objetivos estaban alineados a los objetivos de la investigación. Por lo tanto, la implementación de los prototipos buscó poder determinar el impacto del preprocesamiento de imágenes en la efectividad de la verificación facial. En base a dicho objetivo se planteó los requerimientos principales que permitiesen cumplir dichos objetivos.

3.2.1.2. Comprensión de Datos

En la fase de comprensión de datos se busca realizar tanto la recolección como la comprensión de los datos. Para la recolección se tomó en cuenta lo siguiente:

Iteración 1 – Prototipo de Recolección de Datos:

Para la construcción del prototipo de recolección de datos no se recolectaron imágenes dado que no eran necesarios para su funcionamiento. Se utilizó el modelo de *OpenFace* que ya está preentrenado con imágenes y estas no eran requeridas para su implementación.

Iteración 2 – Prototipo de Pruebas Automatizadas:

Para el prototipo de pruebas automatizadas, se recopilaron las tres bases de datos que serían evaluadas. Como se observó anteriormente, se obtuvieron las siguientes bases de datos de rostros:

- *Labeled Faces in the Wild* (LFW): Base de datos pública, reconocida y usada en diversas investigaciones (Amos, Bartosz, & Satyanaray, 2016; Schroff, Kalenichenko, & Philbin, 2015).
- *YouTube Faces Database* (YTF): Base de datos pública, reconocida y usada en diversas investigaciones (Schroff, Kalenichenko, & Philbin, 2015; Wu, He, Sun, & Tan, 2018).
- Base de Datos del contexto local (DNI): Obtenida del prototipo de recolección de datos desarrollado.

Asimismo, en la comprensión de datos, se analizó el tipo de imágenes que se tenían ya sea por gráficos o métodos estadísticos preliminares. A través de esto se buscó alcanzar una visión previa de la data, así como validar su contenido para la investigación. Por ejemplo, se

analizaron las bases de datos de imágenes, viendo las características de cada una como por ejemplo las variantes en pose, ruido y rango de color.

3.2.1.3. Preparación de Datos

La preparación de datos implica procesar las imágenes para que estas puedan ser leídas por el sistema. Luego de ser obtenidas las imágenes de cada fuente, estas se estructuraron en carpetas y se comprobó que todas las imágenes dentro de cada *dataset* tengan el mismo tamaño, formato y escala de colores.

Asimismo, se desarrollaron los métodos de preprocesamiento de las imágenes.

Iteración 1 – Prototipo de recolección de datos:

En el caso del prototipo de recolección de datos, se utilizó el método de preprocesamiento de alineamiento estándar ya implementado en *OpenFace* sin realizar cambios mayores. El alineamiento fue descrito en la sección 2.2.1.6. Se aplicó el alineamiento de nariz y ojos, el cual consiste en transformar la imagen para que se alinee a dos puntos de ojos y uno de la nariz. Este se implementó como parte del flujo de recepción de imágenes del formulario web.

Iteración 2 – Prototipo de pruebas automatizadas:

En el caso del prototipo de pruebas automatizadas, se establecieron métodos de preprocesamiento para las imágenes siguiendo los métodos de preprocesamiento planteados para la investigación. Estos fueron los siguientes:

- Alineamiento: El alineamiento se describió en la sección 2.2.1.6. En este caso se hallaron primero 68 puntos de referencia. Luego se utilizaron algunos de estos puntos para alinear el rostro dependiendo del tipo de alineamiento. Se evaluaron 3 opciones de alineamiento:
    - Alineamiento 0: Solo se hace un recorte en base a lo detectado con el método de preprocesamiento. No existe alineamiento.
    - Alineamiento 1 – Solo ojos: Se hace una rotación y centrado en base a dos puntos de los ojos.
    - Alineamiento 2 – Nariz y Ojos: Se hacen rotaciones, escalamientos y transformaciones en base a dos puntos de los ojos y uno de la nariz para realizar el alineamiento.
- Suavizamiento: El suavizamiento se aplica mediante el filtro gausiano. Este fue descrito en la sección 2.2.1.5.1. Se aplican 3 opciones de suavizamiento:
    - Sin Gausiano: No se aplica ningún filtro.
    - Gausiano 3x3: Se aplica un filtro gausiano de tamaño 3x3.

    - Gausiano 5x5: Se aplica un filtro gausiano de tamaño 5x5.
- Ecualización de histograma: La ecualización fue descrita en la sección 2.2.1.4. Se aplican 5 opciones de ecualización:
    - Sin ecualización: No se aplica ecualización
    - Ecualización YCbCr: Se separan los canales y se ecualiza el canal de luminosidad de YCbCr normalizando dicho histograma.
    - Ecualización RGB: Se separan los canales y se ecualiza en todos los canales RGB independientemente normalizando dichos histogramas.
    - Ecualización YCbCr Local: Se separan los canales y se ecualiza por regiones locales el canal de luminosidad de YCbCr normalizando dicho histograma.
    - Ecualización HSV Local: Se separan los canales y se ecualiza por regiones locales el canal de luminosidad de HSV normalizando dicho histograma.
- Agudizamiento: El agudizamiento se aplica también mediante un filtro. Este fue descrito en la sección 2.2.1.5.2. Se aplican 5 opciones de agudizamiento:
    - Sin Agudizamiento: No se aplica ningún filtro.
    - Agudizamiento 5%: Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 5%.
    - Agudizamiento 10%: Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 10%.
    - Agudizamiento 15%: Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 15%.
    - Agudizamiento 20%: Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 20%.

Por otro lado, dentro de la etapa de preprocesamiento de datos, se estructuraron las fuentes de información para cada una de las bases de datos utilizadas.

3.2.1.4. Modelado

En la fase de modelado se procede con la creación de los modelos. En este caso los modelos estaban preentrenados por lo que estos fueron adaptados para ser implementados.

Iteración 1:

En el caso del prototipo de recolección de datos, se implementó el modelo basado en la investigación *OpenFace: A general-purpose face recognition library with mobile applications* (Amos, Bartosz, & Satyanaray, 2016). *OpenFace* es un modelo de redes neuronales convolucionales entrenado con 500 mil imágenes de acceso público de las bases de datos *CASIA-WebFace* (Yi, Lei, Liao, & Li, 2014) y *FaceScrub*. (Ng & Winkler, 2014). Este modelo fue descrito a más detalle en la sección 2.2.4.2.9.

Iteración 2:

En el caso del prototipo de pruebas automatizadas se implementaron tres modelos:

- *OpenFace*: se reutilizó la implementación del modelo en la iteración 1 y descrito en la sección 2.2.4.2.9.
- *VGGFace2*: Se implementó un modelo basado en la investigación *VGGFace2: A dataset for recognising faces across pose and age* (Cao, Shen, Xie, Parkhi, & Zisserman, 2018). *VGGFace2* es un modelo entrenado en la base de datos MS-Celeb-1M y en base a un conjunto de datos creado en la misma investigación denominado *VGGFace2* con imágenes obtenidas de Google. El modelo utilizado está basado ResNet-50 con bloques *Squeeze-and-Excitation* (SeNet) (Hu, Shen, Albanie, Sun, & Wu, 2018) que sirven para recalibrar las características de los canales dentro de la red. Este modelo fue descrito en la sección 2.2.4.2.9.
- *Light CNN*: Se implementó un modelo basado en la investigación *A Light CNN for Deep Face Representation with Noisy Labels* (Wu, He, Sun, & Tan, 2018). *Light CNN* es un modelo de redes neuronales convolucionales que recibe de entrada las imágenes en escala de grises y fue entrenado en la base de datos MS-Celeb-1M. Dicho conjunto de datos contiene 79,077 identidades en total con 5,049,824 imágenes. Asimismo, utiliza Max-Feature-Map (MFM) para realizar depuración de neuronas en cada capa separando el ruido. Otra característica es que utiliza filtros de redes en redes (Network in Network) y bloques residuales para reducir la cantidad de espacio utilizado en los parámetros. Se utilizó *Light CNN-29*, este modelo fue descrito en la sección 2.2.4.2.9.

3.2.1.5. Evaluación

En esta fase se probó el funcionamiento de los modelos de manera independiente previo a su implementación. La evaluación global se realizó luego de tener desarrollado completamente

el prototipo de pruebas automatizadas y se describe en el punto 3.3 Metodología de Medición de Resultados.

3.2.1.6. Implementación

En la implementación se desplegaron los modelos y funciones para poder ser consumidas por cualquier sistema. En este caso los modelos *OpenFace*, *VGG-Face2* y *Light CNN* fueron habilitados para ser utilizados por cualquier aplicación. De la misma manera las funciones de preprocesamiento: alineamiento, ecualización, ecualización y suavizamiento.

3.2.2. Metodología de Desarrollo de Software

Como se mencionó en el punto anterior, para el desarrollo de software se aplicó la metodología cascada. Esta se aplicó de manera iterativa dado que se generaron entregables de forma continua mientras se iba avanzando con el desarrollo. Las fases de esta metodología implican las descritas en el punto 2.2.7 y 2.2.8 de esta investigación: análisis, diseño, construcción, pruebas y mantenimiento. Se entrará a más detalle técnico de estas fases en el Capítulo IV.

Adicionalmente, dentro del desarrollo de software, se desarrollaron los activos principales utilizados en la investigación. Esta estuvo dividida en los dos prototipos principales. Ambos prototipos siguieron el proceso de verificación facial al momento de procesar una imagen. El Proceso de Verificación facial abarca lo descrito en las Bases Teóricas de la investigación de la sección 2.2.

A continuación, se describe el resultado final de haber realizado el desarrollo de estos prototipos.

Iteración 1:

En la iteración 1 se realizó el prototipo desarrollado y utilizado para la recolección de datos. El proceso se muestra a continuación:

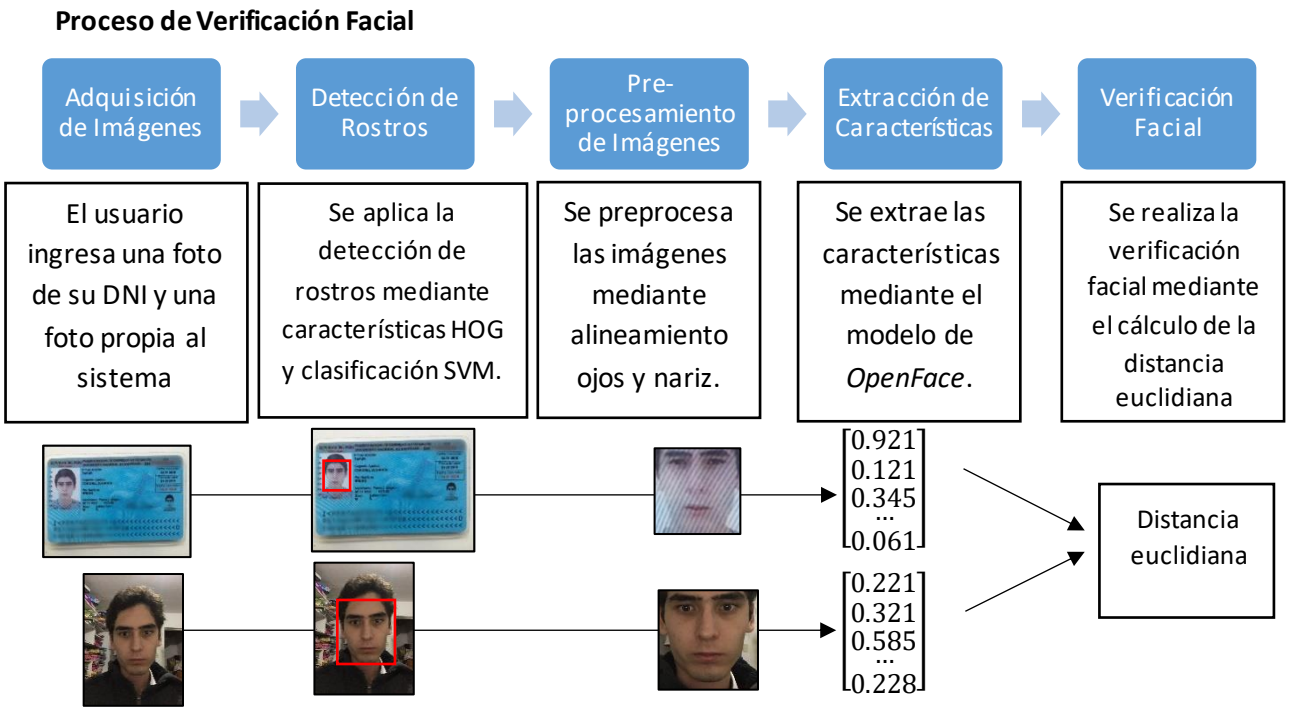


Figura 95. Proceso de Prototipo de Recolección de Datos
Fuente: Elaboración Propia

Este prototipo tuvo como función principal la recolección de la base de datos de DNIs dado que todas las fotos fueron guardadas en una base de datos al realizar el proceso. Sin embargo, también sirvió para implementar el primer proceso estándar de verificación facial.

En este caso, los usuarios ingresan al sistema le toman una foto a su DNI y se toman una foto (*selfie*) subiendo ambas a un formulario a través de su dispositivo móvil. El sistema detecta los rostros mediante características HOG (descrito en la sección 2.3.2.2 – Detección Facial), las alinea mediante alineamiento de ojos y nariz (descrito en la sección 2.2.1.6), extrae las características mediante el modelo de *OpenFace* (descrito en la sección 2.2.4.2.9) y realiza la verificación facial mediante el cálculo de la distancia euclidiana (descrito en la sección 2.2.2). Si la distancia es menor al punto de corte, devuelve un resultado positivo.

Iteración 2:

En el caso del prototipo de pruebas automatizadas dicho proceso fue más complejo dado que existen muchas combinaciones de base de datos, método de detección de rostros, preprocesamiento y extracción de características. A continuación, se muestra este proceso gráficamente.

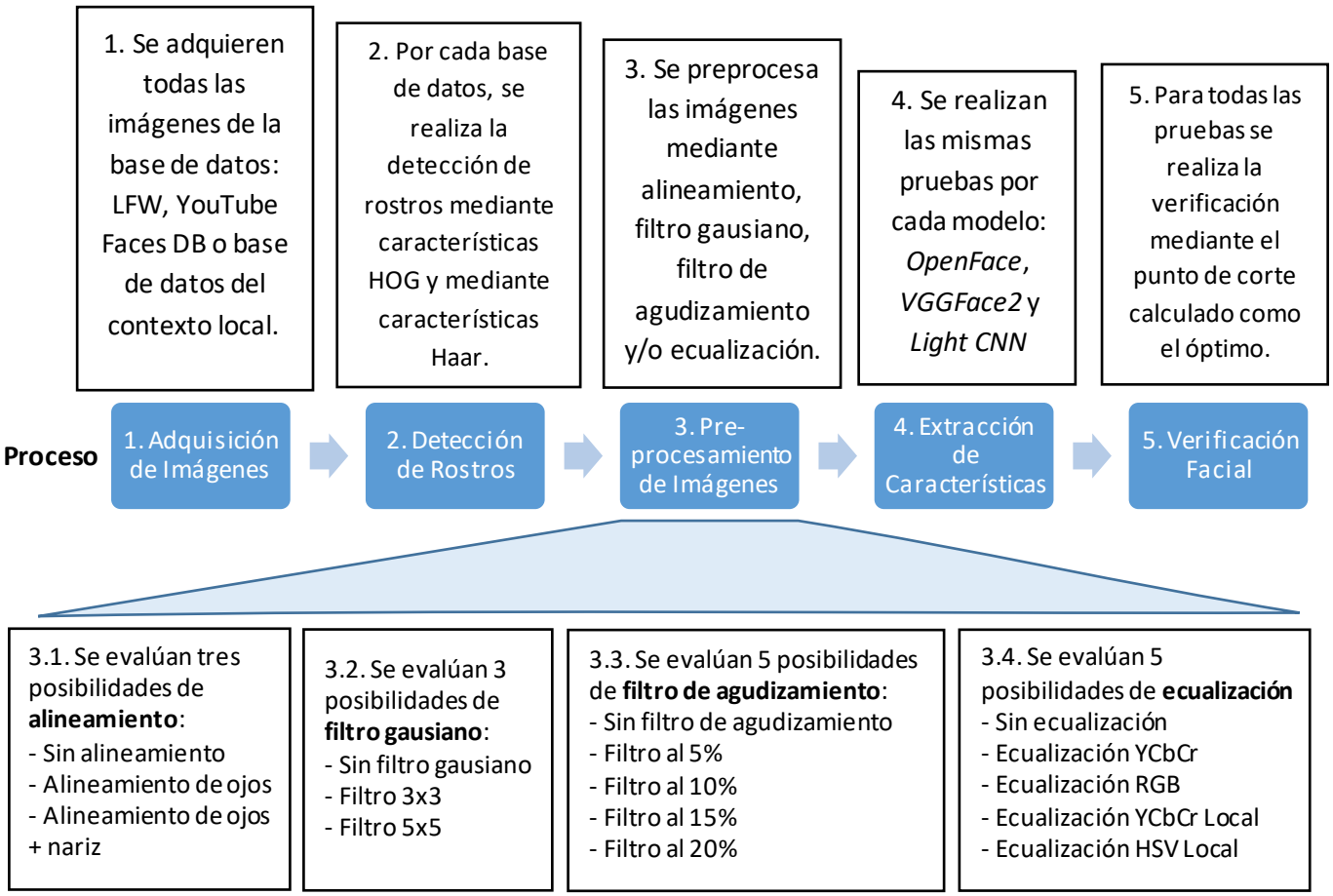


Figura 96. Proceso de Prototipo de Pruebas Automatizadas
Fuente: Elaboración Propia

Como se puede observar, se debió desarrollar el módulo de tal forma que se pueda evaluar las 3 fuentes de información, los 2 métodos de detección facial, los 3 modelos de verificación facial y todas las combinaciones de métodos de preprocesamiento planteadas.

3.3. Metodología para la medición de resultados de la implementación

El análisis de los resultados de la implementación se realizó mediante la evaluación de la relación entre las variables definidas buscando medir el impacto de los métodos de preprocesamiento de imágenes en la efectividad del modelo. Complementariamente se analiza si esta efectividad se ve afectada por la fuente de información, el método de detección facial y/o el modelo de verificación facial utilizado. Por lo tanto, se termina evaluando la totalidad del proceso de verificación facial:

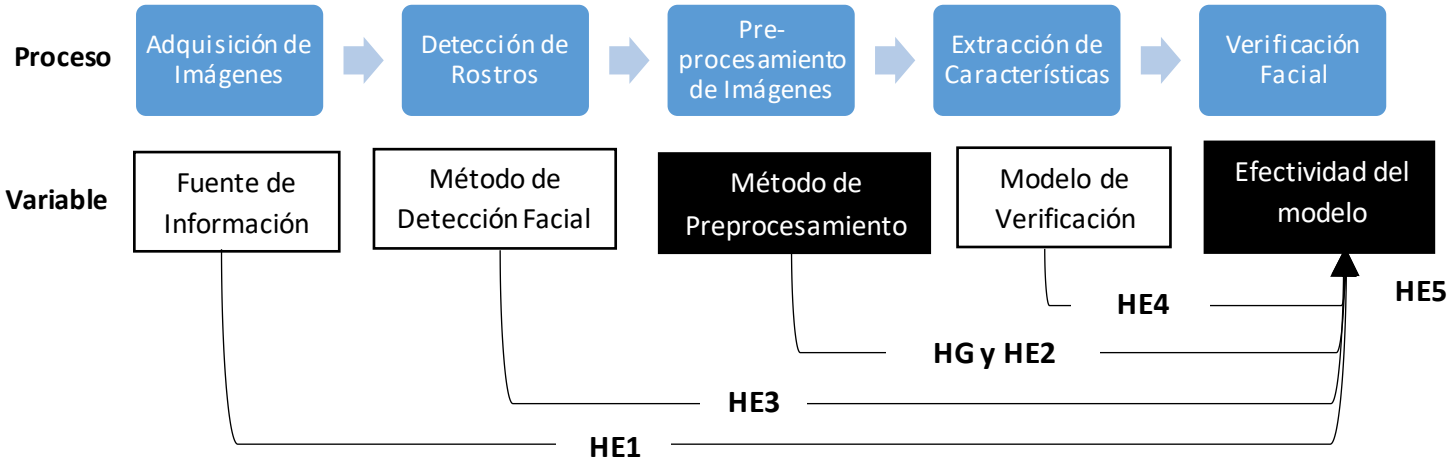


Figura 97. Medición de resultados de implementación
Fuente: Elaboración Propia

En la Figura 97 se puede observar las relaciones que se analizan entre las variables basado en cada una de las 6 hipótesis. Las hipótesis HG, HE1, HE2, HE3 y HE4 toman en cuenta la fuente de información, el método de detección facial, el método de preprocesamiento y el modelo de verificación y su relación con la efectividad de la verificación facial. La hipótesis específica 5 (HE5) evalúa la relación entre las métricas de desempeño determinando si son comparables y si establecen una métrica adecuada para la efectividad en la investigación.

La metodología de medición de resultados se eligió en base a lo presentado en Labeled Faces in the Wild como una de las formas estándar recomendadas de medición (Huang, Ramesh, Berg, & Learned-Miller, 2007). Otra investigación que utiliza dicha metodología como una de sus formas de medición es el estudio relacionado con el modelo VGG-Face (Parkhi, Vedaldi, & Zisserman, 2015). Esta metodología plantea el uso del Equal Error Rate (EER) como métrica y menciona que una ventaja de esta métrica es que es independiente del punto de corte. Por lo tanto, en base a esta revisión de antecedentes y la revisión de su aplicabilidad, se decidió utilizar esta metodología en todas las fuentes de información y modelos, generando pruebas imparciales para la investigación.

Para poder realizar dicho análisis se ejecutaron pruebas de las 3 fuentes de información con 2 métodos de detección facial y 3 modelos de verificación facial para todas las combinaciones de los métodos de preprocesamiento definidas.

La metodología utilizada se basó en la validación cruzada (*cross-validation*) la cual consiste en dividir aleatoriamente un conjunto de datos en *n* subconjuntos iguales y realizar *n* pruebas en las que se separa uno de los subconjuntos para las pruebas y el resto para

entrenamiento. Finalmente se realiza un promedio de los resultados de las *n* pruebas y se utiliza dicho resultado como el final.

Por ejemplo, si se tuviese un conjunto de 1000 parejas de rostros. Se podría dividir en 10 subconjuntos de 100 parejas. Posteriormente se hace una prueba entrenando en 9 de los subconjuntos (900 parejas) y probando en uno (100 parejas). Luego se repite dicho proceso con los 9 subconjuntos restantes. Finalmente, se obtiene el promedio de los resultados de las 10 pruebas.

En este caso los subconjuntos de entrenamiento se usaron para determinar el punto de corte donde el ratio de falsos positivos era igual al ratio de falsos negativos. Por lo tanto, se utiliza el punto de corte donde se tiene el EER (*equal error rate*). De esta forma, para cada prueba se utiliza un punto de corte dinámico y óptimo. Posteriormente, en el subconjunto de pruebas se midió la efectividad en base al punto de corte determinado en los subconjuntos de entrenamiento.

A continuación, se describe a detalle la metodología para la medición de los resultados en el caso de cada fuente de información.

3.3.1. Metodología de medición – *Labeled Faces in the Wild*

En el caso de la base de datos LFW se utilizó como referencia la muestra aleatoria definida por el mismo *dataset*. En esta muestra existen 10 subconjuntos de 600 parejas de rostros cada uno, mitad de las parejas de cada subconjunto son de la misma persona y mitad de las parejas de diferentes personas.

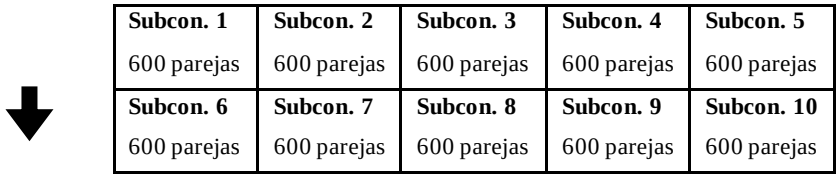
Se utilizó *cross-validation* mediante los 10 subconjuntos de 600 parejas. Es decir, para cada una de las configuraciones de método de preprocesamiento, método de detección facial y modelo de verificación se hizo la misma prueba 10 veces (una en cada subconjunto):

- En los primeros 9 subconjuntos se obtiene el punto de corte donde el ratio de falsos positivos es igual al ratio de falsos negativos (*Equal Error Rate – EER*).
- Luego se evalúa la efectividad del modelo en el subconjunto restante utilizando el punto de corte encontrado en los 9 subconjuntos anteriores.
- Se repite dicho proceso cambiando el subconjunto de pruebas.

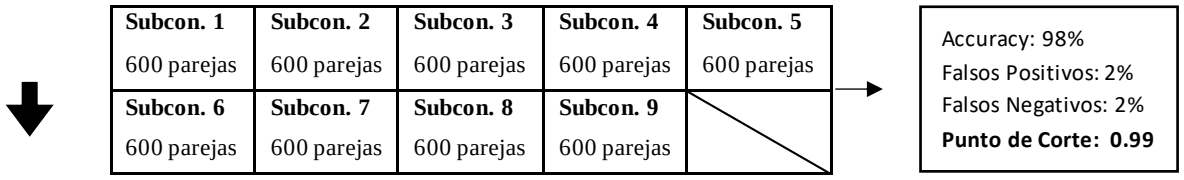
Finalmente se calculó el promedio de los indicadores de efectividad de las 10 pruebas y se usó dicho resultado como el final para cada iteración. Es importante destacar que, utilizando esta metodología, el punto de corte puede variar para cada subconjunto de prueba, utilizando el punto de corte óptimo para cada combinación y evitando el sesgo hacia una combinación u otra.

A continuación, se ilustra este proceso:

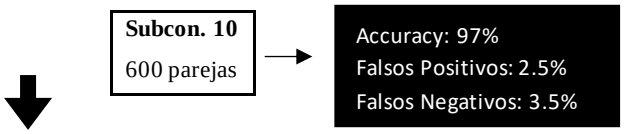
Paso 1: ***Se empieza con 6,000 parejas de imágenes de prueba divididas en 10 subconjuntos***



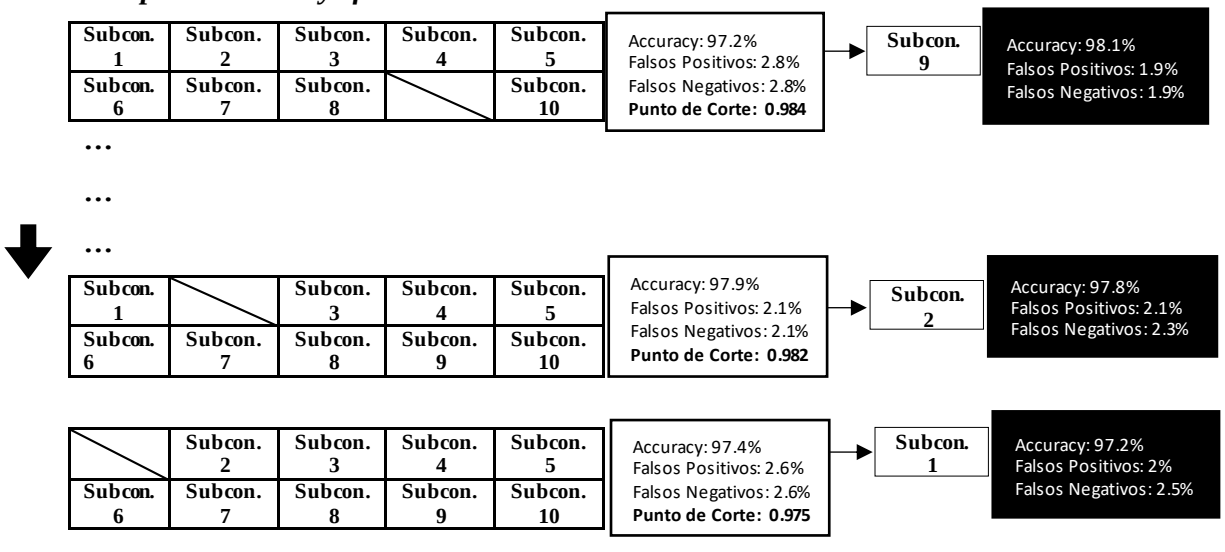
Paso 2: ***9 subconjuntos (5,400 parejas) se evalúan y se obtiene el punto de corte en EER***



Paso 3: ***Se evalúa la efectividad del subconjunto restante utilizando el punto de corte encontrado***



Paso 4: ***Se repite el proceso evaluando la efectividad en cada uno de los otros 9 subconjuntos y hallando su punto de corte y efectividad***



Paso 5: Se ***obtiene el promedio de efectividad de las 10 pruebas***

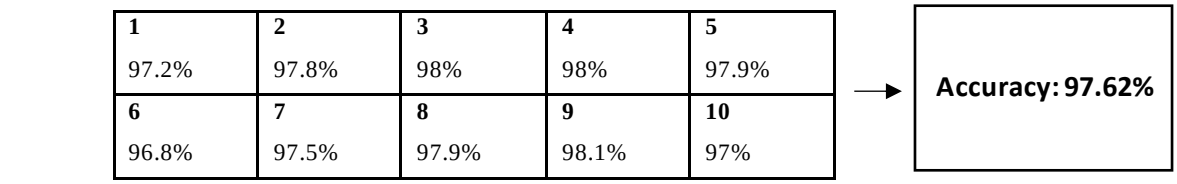


Figura 98. Ejemplo de proceso de cross-validation en LFW
Fuente: Elaboración Propia

3.3.2. Metodología de medición – *YouTube Faces DB*

En el caso de la base de datos YouTube Faces DB se utilizó una muestra elegida aleatoriamente de 5,000 videos definida en el *dataset*. De las 5,000 parejas de videos se eligió una imagen (fotograma) al azar para ser comparada. Se utilizó *cross-validation* mediante 10 subconjuntos de 500 parejas de imágenes de la misma manera como se aplicó en *Labeled Faces in the Wild*. Finalmente se calculó el promedio de los indicadores de efectividad de las 10 pruebas y se usó dicho resultado como el final para cada iteración de método de preprocesamiento, método de detección facial y modelo de verificación.

A continuación, se ilustra este proceso:

Paso 1: ***Se empieza con 5,000 parejas de videos de prueba divididos en 10 subconjuntos.***

Subcon. 1 500 parejas	**Subcon. 2** 500 parejas	**Subcon. 3** 500 parejas	**Subcon. 4** 500 parejas	**Subcon. 5** 500 parejas
Subcon. 6 500 parejas	**Subcon. 7** 500 parejas	**Subcon. 8** 500 parejas	**Subcon. 9** 500 parejas	**Subcon. 10** 500 parejas

Paso 2: ***Se extrae un fotograma aleatorio de cada video obteniendo 5,000 parejas de imágenes. Los fotogramas han sido curados previamente validando que son fotogramas que incluyen un rostro.***

Paso 3: ***9 subconjuntos (4,500 parejas) se evalúan y se obtiene el punto de corte en EER***

Subcon. 1 500 parejas	**Subcon. 2** 500 parejas	**Subcon. 3** 500 parejas	**Subcon. 4** 500 parejas	**Subcon. 5** 500 parejas
Subcon. 6 500 parejas	**Subcon. 7** 500 parejas	**Subcon. 8** 500 parejas	**Subcon. 9** 500 parejas	

→ Accuracy: 98%
Falsos Positivos: 2%
Falsos Negativos: 2%
Punto de Corte: 1.01

Paso 4: ***Se evalúa la efectividad del subconjunto restante utilizando el punto de corte encontrado***

Subcon. 10
500 parejas

→ Accuracy: 97%
Falsos Positivos: 2.5%
Falsos Negativos: 3.5%

Paso 5: ***Se repite el proceso evaluando la efectividad en cada uno de los otros 9 subconjuntos y hallando su punto de corte y efectividad***

Subcon. 1	**Subcon. 2**	**Subcon. 3**	**Subcon. 4**	**Subcon. 5**
Subcon. 6	**Subcon. 7**	**Subcon. 8**		**Subcon. 10**

Accuracy: 97.2%
Falsos Positivos: 2.8%
Falsos Negativos: 2.8%
Punto de Corte: 0984
→ **Subcon. 9**
Accuracy: 98%
Falsos Positivos: 1.9%
Falsos Negativos: 2.1%

…
…
…

Subcon. 1		**Subcon. 3**	**Subcon. 4**	**Subcon. 5**
Subcon. 6	**Subcon. 7**	**Subcon. 8**	**Subcon. 9**	**Subcon. 10**

Accuracy: 97.9%
Falsos Positivos: 2.1%
Falsos Negativos: 2.1%
Punto de Corte: 0.982
→ **Subcon. 2**
Accuracy: 97.1%
Falsos Positivos: 2.1%
Falsos Negativos: 3.3%

	Subcon. 2	**Subcon. 3**	**Subcon. 4**	**Subcon. 5**
Subcon. 6	**Subcon. 7**	**Subcon. 8**	**Subcon. 9**	**Subcon. 10**

Accuracy: 97.4%
Falsos Positivos: 2.6%
Falsos Negativos: 2.6%
Punto de Corte: 0.975
→ **Subcon. 1**
Accuracy: 96.2%
Falsos Positivos: 2%
Falsos Negativos: 4.6%

Paso 6: Se ***obtiene el promedio de efectividad de las 10 pruebas***

1 96.2%	**2** 97.1%	**3** 98%	**4** 98%	**5** 97.3%
6 97.2%	**7** 97.5%	**8** 97.8%	**9** 98.0%	**10** 97%

→ **Accuracy: 97.41%**

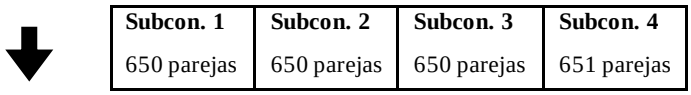
Figura 99. Ejemplo de proceso de cross-validation en YTF
Fuente: Elaboración Propia

3.3.3. Metodología de medición – Base de Datos del contexto local

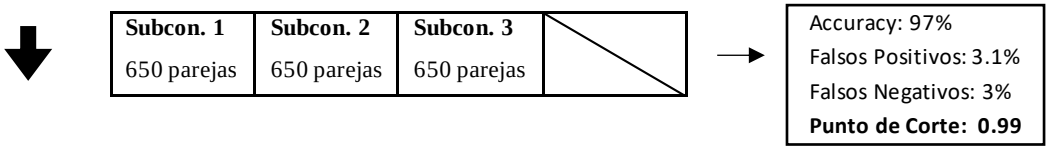
En el caso de la base de datos del contexto local, debido a su tamaño, se utilizó la totalidad de las parejas de imágenes de las 51 personas para las pruebas. Por lo tanto, se tuvieron 2601 parejas de DNI y rostro. Se utilizó *cross-validation* mediante 4 subconjuntos de entre 650 y 651 parejas de imágenes de la misma manera como se aplicó en *LFW y YTF*. Finalmente, se calculó el promedio de los indicadores de efectividad de las 4 pruebas y se usó dicho resultado como el final para cada iteración de método de preprocesamiento, método de detección facial y modelo de verificación.

A continuación, se ilustra este proceso:

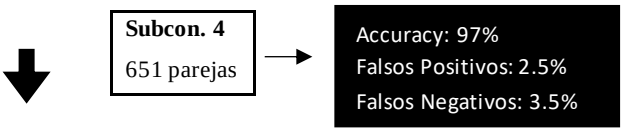
Paso 1: ***Se empieza con 2601 parejas de imágenes de prueba divididas en 4 subconjuntos***



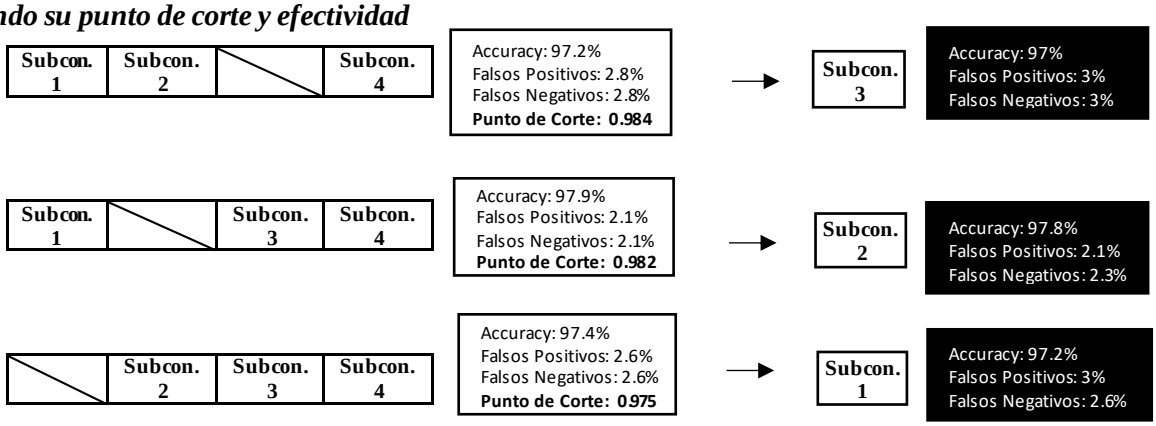
Paso 2: ***3 subconjuntos se evalúan y se obtiene el punto de corte en EER***



Paso 3: ***Se evalúa la efectividad del subconjunto restante utilizando el punto de corte encontrado***



Paso 4: ***Se repite el proceso evaluando la efectividad en cada uno de los otros 3 subconjuntos y hallando su punto de corte y efectividad***



Paso 5: Se ***obtiene el promedio de efectividad de las 4 pruebas***

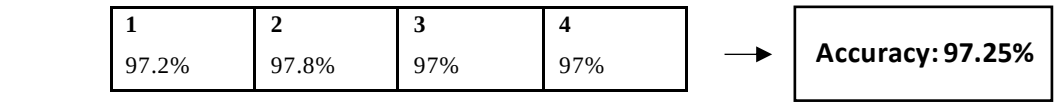


Figura 100. Ejemplo de proceso de cross-validation en base de datos de DNI
Fuente: Elaboración Propia

3.4. Cronograma de actividades y presupuesto

3.4.1. Cronograma de actividades

El cronograma de actividades está dividido en 2 iteraciones y está basado en la metodología descrita en la sección 3.2:

- Iteración 1 – Desarrollo del prototipo de recolección de datos y verificación facial
- Iteración 2 – Desarrollo de prototipo de pruebas automatizadas

A continuación, se muestra el detalle general del cronograma en el año 2016, 2019 y 2020:

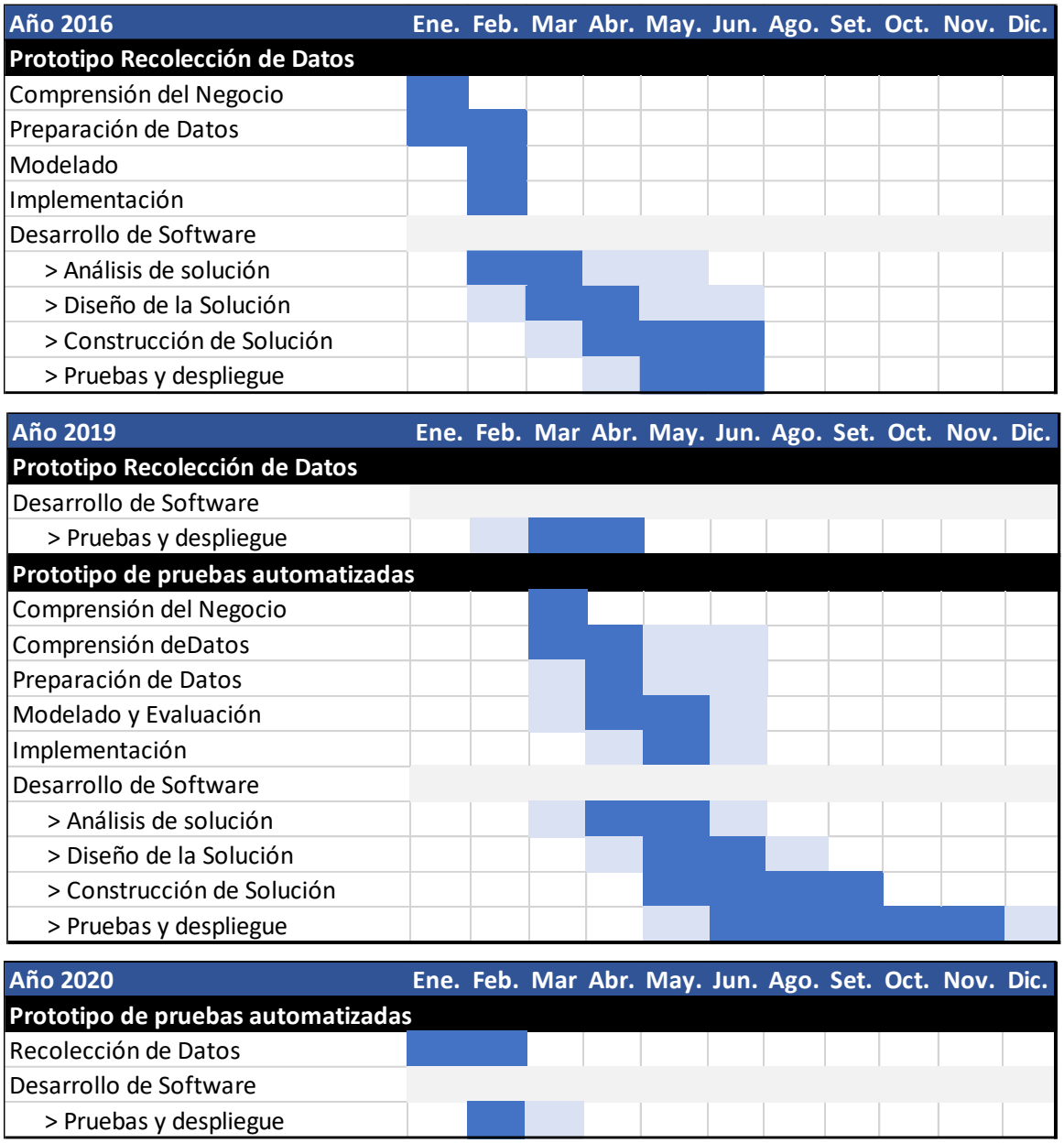


Figura 101. Cronograma de Actividades
Fuente: Elaboración Propia

El prototipo de recolección de datos fue desarrollado inicialmente como parte de una investigación preliminar del autor en el 2016 y presentada en el congreso EQUAA (Universidad ESAN, 2016). No se tuvo que realizar una comprensión y recolección de datos previo al despliegue del prototipo, dado que el fin del prototipo era obtener dichos datos. En la presente investigación dicho prototipo fue nuevamente desplegado y probado con mejoras en el año 2019. El prototipo de pruebas automatizadas fue desarrollado en su totalidad durante el año 2019.

En el año 2020 se tuvo una nueva iteración de recolección de datos y pruebas dado que se decidió ampliar la base de datos de DNIs y se volvieron a realizar las pruebas correspondientes a dicha base de datos.

En color más oscuro se marcan los meses donde se realizó cada fase y en color más claro se marcan los meses donde todavía existieron tareas de dichas fases pero en menor medida.

Dicho cronograma se basa en las metodologías planteada anteriormente y utiliza aspectos de la metodología CRISP-DM de analítica de datos así como de la metodología de desarrollo de software cascada en la construcción. Sin embargo, la investigación ha estado compuesta por de tareas exploratorias y de pruebas, por lo que muchas tareas se suelen traslapar o se suele regresar a fases anteriores.

3.4.2. Presupuesto

El presupuesto se dividirá por ambos proyectos desarrollados. Este toma en cuenta números reales de los componentes de capacidades de cómputo en la nube utilizados. En el proyecto no existieron mayores gastos reales que dichas capacidades. Sin embargo, adicionalmente se invirtió una gran cantidad de tiempo humano en el desarrollo de ambos proyectos.

3.4.2.1. Presupuesto – Prototipo de Recolección de Datos

No se incurrió en costo en el prototipo de recolección de datos a nivel de infraestructura debido a que se utilizó la capa gratuita de todos los servicios de infraestructura y base de datos. Dichas capacidades fueron suficientes para ejecutar los módulos correspondientes. El detalle de los componentes y servicios desplegados se verá a más detalle en el Capítulo IV donde se describe la arquitectura.

3.4.2.2. Presupuesto – Prototipo de Pruebas Automatizadas

A continuación, se detalla el presupuesto de gastos en infraestructura en la nube para la ejecución de los componentes del prototipo de pruebas automatizadas. Esto se basó en la

cantidad de horas de ejecución y almacenamiento para probar cada modelo de verificación facial. Todos estos costos fueron en componentes y servicios de infraestructura de la nube de AWS. El detalle de los componentes de las líneas de costos se verá a más detalle en el Capítulo IV donde se describe la arquitectura.

Módulo de pruebas automatizadas	Costo
OpenFace	
Capacidad Infraestructura AWS - EC2	$ 30.25
Almacenamiento Infraestructura AWS - EBS	$ 4.76
VGGFace2	
Capacidad Infraestructura AWS - EC2	$ 54.45
Almacenamiento Infraestructura AWS - EBS	$ 4.76
LightCNN	
Capacidad Infraestructura AWS - EC2	$ 42.35
Almacenamiento Infraestructura AWS - EBS	$ 4.76
Compartido	
Monitoreo AWS - CloudWatch	$ 5.20
File Server AWS - EFS	$ -
Repositorio Imágenes Docker AWS - ECR	$ 0.73
Total	**$ 147.27**

Figura 102. Costo Componentes de Infraestructura AWS
Fuente: Elaboración Propia

Adicionalmente, ciertos costos que estaban debajo del límite gratuito no se incurrieron. Esto se ve más claramente en la fila del *File Server* (servidor de archivos), que consumió menos de lo mínimo necesario para incurrir en un costo.

CAPÍTULO IV: DESARROLLO DE LA SOLUCIÓN

En esta sección se detalla de mejor manera la solución planteada, sus componentes técnicos y las decisiones tomadas durante su desarrollo.

4.1. Determinación y evaluación de alternativas de solución

Tomando en cuenta la dirección planteada por la investigación en base a los problemas, hipótesis y objetivos, se describirá las alternativas de solución para ambos activos:

4.1.1. Prototipo de recolección de datos y verificación facial

El prototipo de recolección de datos y verificación facial tiene como objetivo recolectar datos del contexto peruano y, adicionalmente, apoyar como prototipo inicial de un modelo de verificación facial siguiendo el proceso completo. Fue importante tomar decisiones y evaluar alternativas como las que se describen a continuación al momento de desarrollar el prototipo.

En primer lugar, se debió decidir cómo sería la interacción con el usuario. Se prefirió darle flexibilidad a la interfaz para ser móvil. Tomando esto en cuenta se decidió que la interacción con el usuario sea mediante un formulario sencillo que finalizando retorne un resultado estándar de un modelo y guardase las fotos y datos en una base de datos.

Por otro lado, respecto al modelo de verificación facial, se tomó como alternativa los modelos basados en redes neuronales convolucionales dado que son de los más avanzados dentro de la industria y brindan buenos resultados de efectividad. Asimismo, dentro de los modelos de redes neuronales convolucionales, se debió decidir el modelo de verificación facial a implementar como prueba. Se podría haber usado cualquiera de los tres modelos planteados en esta investigación. Sin embargo, siguiendo el propósito de mantener el desarrollo lo más estándar posible, se decidió utilizar *OpenFace* debido a que tiene componentes demostrativos ya documentados que permiten desplegarlo con menores modificaciones en un entorno web, incluyendo, por ejemplo, el preprocesamiento de la alineación de rostro. Sin embargo, dicho prototipo podría conectarse a cualquiera de los modelos de verificación planteados.

Igualmente, para las técnicas de detección se utilizó el método de detección facial ya probado en *OpenFace* y que además es uno de los más comunes en la industria: el método de extracción de características basado en descriptor HOG.

Otra decisión importante se centra en dónde desplegar los componentes. En este caso, basándonos en que *OpenFace* es una librería que se despliega como un contenedor Docker; se prefirió desplegarlo en una infraestructura en la nube que tuviese dichas capacidades. Esto se podría haber realizado en Google Cloud Platform, Microsoft Azure, entre otras. Sin embargo, por motivos económicos y de elasticidad en el manejo de las cargas, se prefirió utilizar IBM Cloud para el componente web y AWS de Amazon para el componente de verificación facial.

Acerca de los lenguajes de programación utilizados, se decidió utilizar Python en el componente de verificación facial de la aplicación debido a que dicho lenguaje es el utilizado por *OpenFace*. Asimismo, en el componente del formulario web se decidió usar Python para mantener un lenguaje estándar entre ambos componentes. Sin embargo, existe flexibilidad en el formulario para ser desarrollado en otros lenguajes.

4.1.2. Prototipo de pruebas automatizadas

El prototipo de pruebas automatizadas está compuesto por una variedad de componentes más complejos. El propósito inicial es poder realizar la evaluación automatizada de todos los métodos de preprocesamiento de la investigación en todos los modelos de verificación facial, bases de datos y métodos de detección facial mencionados.

Previo a iniciar el desarrollo, una decisión importante recaía en la elección de las alternativas de bases de datos a evaluar. Se tuvo como propósito evaluar en bases de datos reconocidas, así como también en el contexto local. Por lo tanto, se decidió realizar las pruebas en las bases de datos *YouTube Faces DB* y LFW, ambas bases de datos reconocidas y utilizadas en investigaciones del rubro (Amos, Bartosz, & Satyanaray, 2016; Schroff, Kalenichenko, & Philbin, 2015). Adicionalmente, se utilizó la base de datos del contexto local recolectada durante la presente investigación.

Además, se debió evaluar los métodos de detección facial a utilizar como parte de las pruebas. La detección facial ganó mucha atención entre los investigadores al ver los grandes resultados del algoritmo de detección de Viola y Jones basado en descriptores Haar, este fue un gran paso para las aplicaciones de visión computacional dado que fue el primer algoritmo en tener una alta efectividad acompañado de una detección veloz (Paisitkriangkrai, Shen, & Zhan, 2010). Por otro lado, el detector basado en características HOG es también popular y brinda buenos resultados. Este fue elegido por la investigación de *OpenFace* como parte de la detección facial utilizada en su implementación (Amos, Bartosz, & Satyanaray, 2016). La velocidad del algoritmo es muy importante al realizar implementaciones y esto permite su uso en investigaciones con detección de rostros en video en tiempo real o aplicaciones móviles (Adouani, Wiem, & Zied, 2019; Amos, Bartosz, & Satyanaray, 2016). Estos factores hacen que ambos extractores de características sean reconocidos para su aplicación en sistemas de detección facial. Como consecuencia, ambos algoritmos están disponibles en las librerías más populares de visión computacional *OpenCV* y *dlib*. Por lo tanto, se decidió por ambos extractores dado que son algunos de los métodos de extracción de características más comunes para este tipo de implementaciones, cuentan con alta efectividad, tienen una buena velocidad de detección y cuentan con aplicaciones *open source*.

Por otro lado, en los métodos de preprocesamiento se buscó abarcar una variedad de posibilidades y combinaciones dentro de los métodos más comunes en la literatura. Entre estos se planteó el uso de suavizamiento, ecualización, alineamiento y agudizamiento. Como se pudo observar, tanto el suavizamiento, agudizamiento y ecualización son métodos de preprocesamiento tradicionales dentro de la visión computacional. Asimismo, el alineamiento ha sido usado en muchos de los modelos de reconocimiento facial abarcados en la literatura como *OpenFace*, *FaceNet* y *Light CNN* (Wu, He, Sun, & Tan, 2018; Amos, Bartosz, & Satyanaray, 2016; Schroff, Kalenichenko, & Philbin, 2015).

Asimismo, respecto a los modelos de verificación facial, se buscó modelos modernos y con buenos resultados en la industria, esto incluía los modelos de redes neuronales convolucionales

que han tenido mucho éxito en los últimos años. Además, se tomó la decisión de utilizar *OpenFace*, *VGGFace2* y *Light CNN* dado que los tres modelos tienen alta efectividad y cuentan con implementaciones con buen rendimiento computacional, pudiéndose utilizar en sistemas reales. Asimismo, dichos modelos cuentan con disponibilidad de acceso libre para su uso.

Un gran reto fue encontrar soluciones que logren integrar esta gran variedad de posibilidades en flujos de datos y que generen estos resultados de forma consistente y sin complicaciones. Esta dificultad venía acompañada de complejidades que se describen a continuación:

- El código de cada uno de los 3 modelos de verificación facial estaba desarrollado con dependencias de diferentes librerías y versiones lo que hacía muy complejo poder integrar todas las pruebas en un solo componente y que pudiesen conversar entre ellos.
- El código de estos modelos de verificación facial ya venía integrado con métodos de preprocesamiento o detección facial por defecto con la librería. Se debía buscar una manera de desacoplarlo sin dañar la funcionalidad.
- Las dependencias del código de los modelos de verificación facial podían generar conflicto con las dependencias del código de los nuevos métodos de preprocesamiento o detección facial que se deseaban evaluar.

Para solucionar las complejidades de integración se tuvo que tomar acciones a través de varios frentes importantes evaluando diferentes alternativas:

- Los métodos de preprocesamiento se generaron utilizando funciones que pudiesen ser reutilizables entre todos los modelos de verificación.
- Las pruebas de las métricas, *cross-validation* y su exportación a archivos también fueron desarrollados de tal forma que pudiesen ser reutilizables entre todos los tipos de pruebas intentando limitar el retrabajo.
- Se buscó desacoplar y separar funciones en base a componentes. En los casos de cada modelo de verificación se decidió ejecutarlo en un componente diferente para mantener sus dependencias sin mayores conflictos.

Por otro lado, se encontraron complejidades con los tiempos de ejecución de las pruebas. Se debían ejecutar miles de iteraciones para recrear todas las combinaciones planteadas, las cuales a su vez tenían que ejecutar funciones que requerían capacidades de cómputo y tiempo. Estas funciones debían realizar el preprocesamiento, detección de rostros y verificación facial. Para contrarrestar esto, se crearon ciertos *caches* intermedios para almacenar las imágenes ya detectadas o preprocesadas y reutilizarlas en las siguientes iteraciones.

Finalmente, de igual manera a como se realizó con el prototipo anterior, se decidió utilizar Python como lenguaje de programación en este componente debido a que dicho lenguaje es común en los modelos implementados. Asimismo, se prefirió utilizar componentes Docker en la nube de AWS por razones de flexibilidad y costo.

4.2. Propuesta de Solución

4.2.1. Planeamiento y descripción de actividades

Las siguientes actividades siguen la metodología presentada en la sección 3.2. desde una perspectiva práctica para el desarrollo de la solución. Dichas actividades se desarrollaron en dos iteraciones: una iteración para el prototipo de recolección de datos y otra iteración para el prototipo de pruebas automatizadas.

4.2.1.1. Comprensión del Negocio

Como se describió anteriormente, se basó en comprender y tener claros los objetivos de la investigación para poder ponerlos en práctica en las siguientes fases.

4.2.1.2. Comprensión de datos

En la fase de recolección de datos se busca realizar la adquisición y análisis de las imágenes recolectadas. Esto se realizó en dos iteraciones:

Iteración 1 – Prototipo de Recolección de Datos: Para la construcción del prototipo de recolección de datos no se recolectaron imágenes dado que no eran necesarios para su funcionamiento y el modelo ya estaba entrenado.

Iteración 2 – Prototipo de Pruebas Automatizadas: Para el prototipo de pruebas automatizadas, se recopilaron las tres bases de datos. La recopilación se realizó mediante las siguientes actividades:

1. Descarga de base de datos de *Labeled Faces in the Wild* (LFW)
2. Descarga de base de datos *YouTube Faces Database* (YTF)
3. Recopilación y descarga de Base de Datos del contexto local (DNI)

Adicionalmente, en la iteración 2, en la etapa de comprensión de datos se observó la data de rostros de las 3 bases de datos y se analizó su complejidad en términos de pose, iluminación y expresión.

1. Analizar base de datos de DNI
2. Analizar base de datos LFW
3. Analizar base de datos YTF

4.2.1.3. Preparación de Datos

Para la preparación de datos se aplicaron los métodos de preprocesamiento de la siguiente manera:

Iteración 1 – Prototipo de Recolección de Datos: Se aplicó el método de preprocesamiento estándar utilizado en *OpenFace*, el cual es el alineamiento en base a ojos y nariz.

1. Preprocesar con Alineamiento

Iteración 2 – Prototipo de Pruebas Automatizadas: Para el prototipo de pruebas automatizadas, se estructuró la data para ser procesada y se evaluaron los 4 métodos de preprocesamiento mencionados, siguiendo las actividades a continuación en este orden:

1. Estructurar data de Base de Datos de Labeled Faces in the Wild
2. Estructurar data de YouTube Faces Database
3. Estructurar data de Base de Datos de Contexto Local (DNI)
4. Preprocesar con Alineamiento
5. Preprocesar con Ecualización
6. Preprocesar con Suavizamiento
7. Preprocesar con Agudizamiento

4.2.1.4. Modelado y Evaluación

En la etapa de algoritmos de Modelado se realizaron las siguientes actividades para cada prototipo:

Iteración 1 – Prototipo de Recolección de Datos:

1. Descarga del modelo de OpenFace
2. Validación de dependencias del modelo
3. Prueba del modelo

Iteración 2 – Prototipo de Pruebas Automatizadas: El modelo de *OpenFace* ya fue descargado y probado en la iteración anterior por lo que esta iteración se centró en *VGGFace2* y *Light CNN* desarrollando las siguientes actividades:

1. Descarga del modelo de *VGGFace2*
2. Validación de dependencias del modelo de *VGGFace2*
3. Prueba del modelo de *VGGFace2*
4. Descarga del modelo *Light CNN*
5. Validación de dependencias del modelo de *Light CNN*

6. Prueba del modelo de *Light CNN*

4.2.1.5. Implementación

En esta etapa se pusieron en práctica y estructuraron los componentes de analítica de datos desarrollados en las fases anteriores para poder construirse los dos prototipos a partir de estos componentes.

4.2.1.6. Desarrollo de Software

La construcción de los prototipos fue realizada mediante el desarrollo de software cascada utilizando como punto de partida lo planteado anteriormente. Esto se realizó mediante las siguientes actividades:

1. Análisis

Para el prototipo de recolección de datos, se analizó las necesidades referentes en usabilidad e interfaz de usuario que permitan una correcta recopilación de imágenes. Asimismo, para ambos prototipos, se analizó las necesidades relacionadas con la arquitectura técnica de la solución de tal forma que puedan ejecutarse cumpliendo los objetivos de la investigación.

2. Diseño

En la etapa de diseño se definió la estructura de los sistemas y la relación entre sus componentes. Además, se determinó los ambientes donde se iban a ejecutar los prototipos y el detalle técnico de las tecnologías a utilizar en cada uno. Asimismo, se determinó la interfaz del prototipo de recolección de datos para que sea amigable al usuario y utilizable en dispositivos móviles.

3. Construcción

La etapa de construcción involucró el desarrollo de ambos prototipos: el prototipo de recolección de datos y el prototipo de pruebas automatizadas.

4. Pruebas

En la etapa de pruebas se probó el funcionamiento de cada prototipo. Dentro de cada prototipo las pruebas se hicieron primero a nivel unitario de código. Luego, se pasaron a pruebas de integración para evaluar el resultado obtenido de grupos de componentes integrados. Asimismo, en el prototipo de recolección de datos, se hicieron pruebas para las validaciones de los campos del formulario, así como del formato de las imágenes.

4.2.2. Desarrollo de Actividades. Aplicación de Herramientas de Solución

Esta sección detalla las actividades descritas en el planeamiento de actividades (sección 4.2.1) y las herramientas que se utilizaron para el desarrollo de cada módulo.

4.2.2.1. Prototipo de Recolección de Datos

Esta sección detalla las actividades y herramientas para el desarrollo del prototipo de recolección de datos.

4.2.2.1.1. Comprensión del Negocio

El prototipo de recolección de datos tiene como objetivo principal lograr la recolección de fotos de DNI y rostro de los usuarios a partir de un prototipo básico de verificación facial.

4.2.2.1.2. Comprensión de Datos

Para el prototipo de recolección de datos no se necesitó recopilar datos dado que el sistema no los requirió para poder desarrollarse. Este prototipo sirve para recolectar los datos de DNIs que se usarán en el prototipo de pruebas automatizadas como parte de una de las fuentes de información.

4.2.2.1.3. Procesamiento de Datos

En este prototipo sólo se aplicó el método de preprocesamiento estándar utilizado en *OpenFace*, el cual es el alineamiento en base a ojos y nariz.

1. Preprocesar con Alineamiento

Figura 103. Aplicación de Alineamiento
Fuente: Elaboración Propia

En la Figura 103 se observa un ejemplo de la aplicación de este tipo de alineamiento

- Alineamiento 0: No existe alineamiento, solo se hace un recorte en base a lo detectado con el método de preprocesamiento.
- Alineamiento 2: Se hacen rotaciones, escalamientos y transformaciones en base a los puntos 36, 46 y 33 de la Figura 122 que se muestra más adelante.

4.2.2.1.4. Modelado y evaluación

Para el prototipo de recolección de datos se utilizó el modelo entrenado de *OpenFace*. Para aplicar el modelo de verificación facial se realizó lo siguiente:

1. Descarga del modelo de *OpenFace*

Este se descargó del siguiente enlace usando git https://github.com/cmusatyalab/openface. Una imagen del repositorio se observa a continuación:

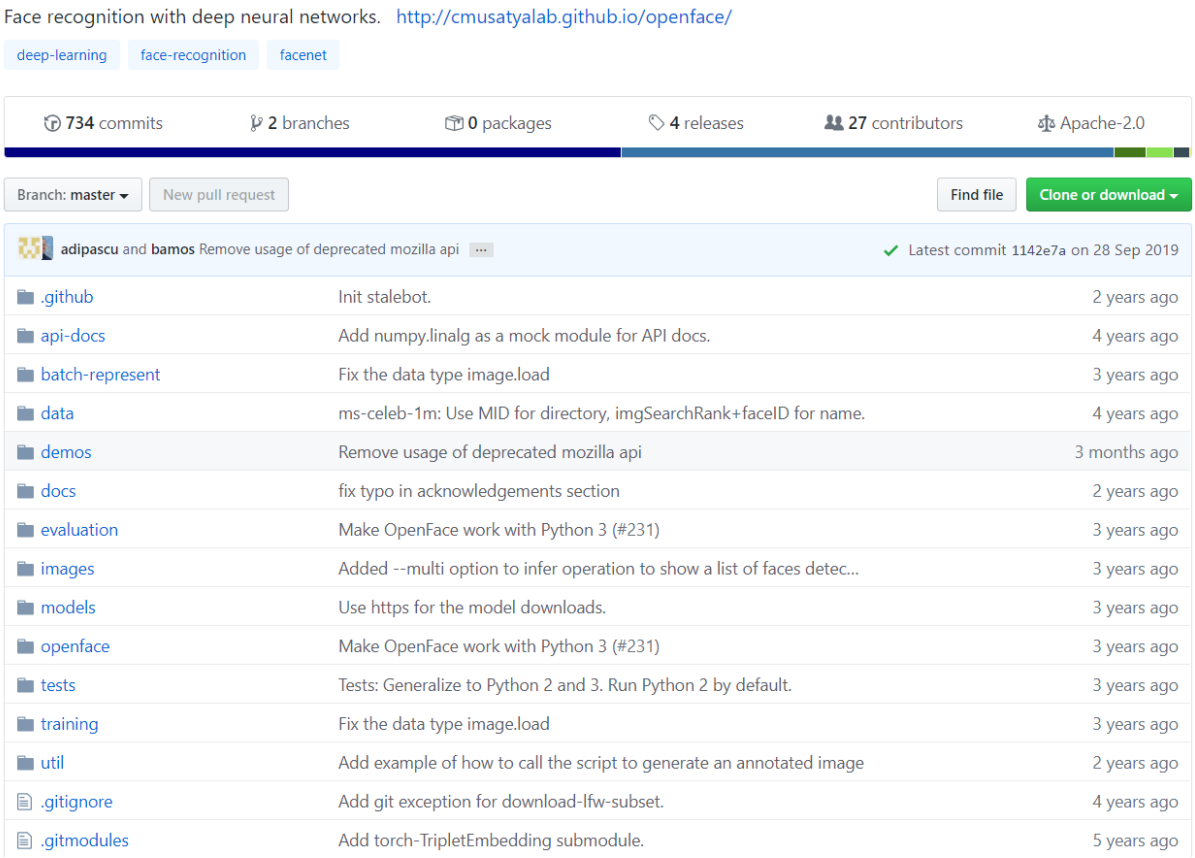


Figura 104. Detalle de repositorio del modelo OpenFace
Fuente: Amos, Bartosz, & Satyanarayanan (2016). *OpenFace*

Como se observa en la Figura 105, dicho modelo tiene diferentes variantes que se desarrollaron. Se utilizó la versión *nn4.small2.v1* dado que es el que es el que tiene mayor efectividad en las pruebas tal como se observa en la tabla.

Model	Accuracy	AUC
nn4.small2.v1 (Default)	0.9292 ± 0.0134	0.973
nn4.small1.v1	0.9210 ± 0.0160	0.973
nn4.v2	0.9157 ± 0.0152	0.966
nn4.v1	0.7612 ± 0.0189	0.853
FaceNet Paper (Reference)	0.9963 ± 0.009	not provided

Figura 105. Detalle de modelos de OpenFace
Fuente: Amos, Bartosz, & Satyanarayanan (2016). *OpenFace*

2. Validación de dependencias del modelo de *OpenFace*

Las dependencias están documentadas dentro del mismo repositorio del modelo. Sin embargo, el despliegue recomendado es mediante Docker. Esto ayuda a encapsular las dependencias. Por lo tanto, se pudo instalar cumpliendo el requisito de tener instalado Docker.

3. Prueba del modelo de *OpenFace*

Se descargó la imagen Docker y se creó el contenedor en base a dicha imagen. Posteriormente, se realizó pruebas con parejas de imágenes para ver que los resultados obtenidos sean positivos y que el modelo no requiera dependencias adicionales.

4.2.2.1.5. Implementación

Se estructuraron y se hicieron disponibles los componentes de código de analítica de datos de forma reutilizable para pasar al proceso de desarrollo de software del prototipo.

4.2.2.1.6. Desarrollo de Software

La construcción del prototipo de recolección de datos se realizó siguiendo la metodología de desarrollo de software cascada.

1. Análisis

Dentro del análisis, para prototipo de recolección de datos se recabaron los requerimientos principales en base a los objetivos. A continuación, se muestran los principales requerimientos funcionales y no funcionales en la Tabla 4 y Tabla 5:

Tabla 4.

Requerimientos Funcionales - Prototipo de recolección de datos

#	Descripción
RF01	El sistema debe incluir un formulario que permita a los usuarios subir una foto de su rostro (*selfie*) y una foto de su DNI.
RF02	El sistema debe procesar y generar un resultado estándar de verificación facial entre ambas fotos subidas por el usuario.
RF03	El sistema debe permitir la descarga posterior de todas las imágenes enviadas para ser utilizadas posteriormente en la investigación.

Tabla 5.

Requerimientos No Funcionales - Prototipo de recolección de datos

#	Descripción
RNF01	El sistema debe tener una interfaz sencilla e intuitiva para el usuario.
RNF02	El ingreso de datos debe incluir validaciones correspondientes que restrinjan que se ingrese la información e imágenes correctamente.
RNF03	El sistema debe funcionar correctamente en dispositivos móviles de diferentes plataformas y sistemas operativos.
RNF04	El tiempo de respuesta del formulario no debe exceder los 6 segundos con una velocidad de Internet estándar.

2. Diseño

Dentro del diseño se debió tomar en cuenta la parte de arquitectura como la parte de diseño al usuario final.

La arquitectura desarrollada se muestra a continuación:

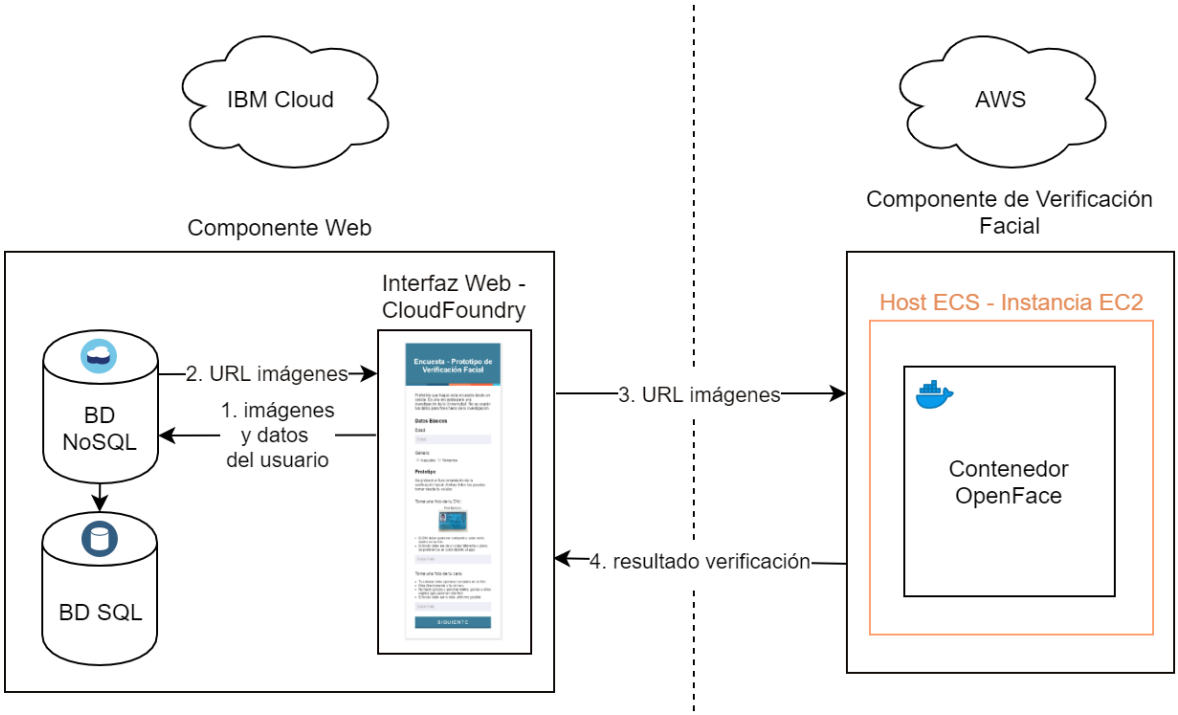


Figura 106. Arquitectura del Prototipo de Recolección de Datos
Fuente: Elaboración Propia

Como se observa en la Figura anterior, el prototipo está dividido en dos componentes principales:

- El primer componente representa la aplicación web mediante un formulario hacia el usuario final. Se encuentra desplegado en IBM Cloud.
- El segundo componente incluye el servicio de verificación facial. Se encuentra desplegado en AWS.

La separación viene de la mano con facilitar la distinción entre funciones de cada componente y facilitar la reutilización del servicio, así como reducir la complejidad de las pruebas.

El primer componente recibe los datos del usuario y se ejecuta como una instancia de *Cloud Foundry* de 128MB de memoria en IBM Cloud. Los datos recibidos los almacena en una base de datos no relacional (NoSQL) en la que no existe una estructura estricta para almacenamiento y brinda flexibilidades para almacenamiento de imágenes. Una vez almacenadas las imágenes, el formulario envía la URL de las imágenes al servicio de verificación facial. En caso sea exitoso, se recibirá la distancia euclidiana desde dicho servicio. Luego de tener la distancia, se toma la decisión en base al punto de corte. Finalmente, se almacenan los resultados en la base de datos NoSQL y se muestra el mensaje de envío exitoso al usuario con su resultado.

Este componente fue alojado sobre IBM Cloud con una base de datos NoSQL denominada Cloudant y una base de datos relacional DB2. Los datos almacenados en la base de datos NoSQL son enviados continuamente a un *data warehouse* transaccional. Desde esa base de datos transaccional se pueden extraer fácilmente los registros, filtrarlos, ordenarlos y exportarlos a Excel.

El segundo componente está relacionado con la red neuronal convolucional de *OpenFace*. Este servicio recibe peticiones del primer componente y responde las consultas automáticamente. Las consultas se hacen enviando las URLs de las dos imágenes faciales para que devuelva los resultados de la comparación. Este componente fue alojado sobre una instancia de Amazon EC2 (instancia *t2.micro* de AWS). Sobre esa instancia se ejecutó el contenedor Docker con el servicio y su código fuente.

Instancia	CPU virtual*	Créditos por hora de CPU	Memoria (GiB)	Almacenamiento	Rendimiento de red
t2.nano	1	3	0,5	Solo EBS	Bajo
t2.micro	1	6	1	Solo EBS	De bajo a moderado
t2.small	1	12	2	Solo EBS	De bajo a moderado
t2.medium	2	24	4	Solo EBS	De bajo a moderado

Figura 107. Características de instancia t2.micro
Fuente: Amazon Web Services (2020). *Tipos de instancias de Amazon EC2*

La Figura 107 muestra el detalle de esta instancia (t2.micro) en AWS. Como se puede observar, es una instancia con características económicas. Estas se describen a continuación:

- Instancia: Representa el nombre de la instancia. En este caso es t2.micro.
- CPU virtual: Representa la cantidad de vCPUs que tiene la instancia. En este caso es 1.
- Créditos por hora de CPU: Es una medida de AWS para cuántos créditos se acumulan por cada hora de uso del CPU. Estos créditos se pueden utilizar para canjear mayor rendimiento en la máquina virtual sin un costo adicional. En este caso son 6 créditos.
- Memoria (GiB): Representa la cantidad de memoria RAM que tiene la máquina virtual. En este caso es 1 GB.
- Almacenamiento: Representa el tipo de almacenamiento que viene con la instancia. En este caso viene con EBS (*Elastic Block Storage*).
- Rendimiento de Red: Representa las capacidades de velocidad de red que tiene la instancia. En este caso es de bajo a moderado.

Por el lado del diseño del formulario, se decidió hacer una interfaz sencilla que pudiese verse en un móvil. A continuación, se muestra la interfaz:

Figura 108. Diseño de Interfaz de Prototipo de Recolección de Datos
Fuente: Elaboración Propia

Los campos principales dentro del formulario son los de ambas fotos. Los campos de edad y género se incluyeron por motivos de emular el envío de más data en los formularios. Sin embargo, no fueron utilizados en la presente investigación. Asimismo, las instrucciones se hicieron lo más claras posibles para facilitar la comprensión y evitar fallas en los envíos. Además, se procuró que estas sean cortas para que los usuarios las lean completas.

3. Construcción

En la construcción del prototipo de recolección de datos se desarrolló todo lo planteado para poder lograr el resultado deseado de cara al usuario que enviaría sus fotos.

El prototipo de recolección de datos está compuesto por tres elementos principales:

- La imagen Docker que contiene el código necesario para que se ejecute el modelo de verificación facial *OpenFace*. Esta tiene como lenguaje principal Python. Sin embargo, la red neuronal convolucional de *OpenFace* fue desarrollada en el lenguaje de programación Lua. Dicha red se encuentra dentro de la imagen Docker.
- El directorio del código fuente del componente web. Este fue desarrollado en Python e incluye el código del formulario web. Además, para la parte visual se utilizó HTML, CSS y JavaScript. Este código realiza lo siguiente:
    - Obtener datos enviados por el usuario
    - Almacenar imágenes
    - Enviar *URL*s al componente de verificación facial
    - Esperar respuesta
    - Recibir la distancia euclidiana
    - Tomar una decisión en base a la distancia euclidiana
    - Almacenar resultado
    - Mostrar mensaje de éxito

  Las peticiones del módulo web al módulo de verificación facial se hacen cada vez que un usuario envía el formulario. La conexión entre ambos módulos se hizo a través de un *Websocket.*
- El directorio del código fuente del componente de verificación facial. Este fue desarrollado en Python y realiza lo siguiente:
    - Obtener ambas imágenes de las *URL*s enviadas por el usuario
    - Detectar rostros de ambas imágenes
    - Alinear Rostros
    - Ejecutar el modelo de *OpenFace* del contenedor Docker para extraer características
    - Calcular distancia entre rostros y enviarla

En este caso, este código reutilizó la mayor parte de la detección de rostro, alineamiento, ejecución del modelo y cálculo de la distancia en base a funciones estándar de *OpenFace.* Esto se debe a que la función central fue almacenar las fotos para la base de datos y hacer una prueba inicial del proceso estándar.

Por otro lado, respecto al despliegue del prototipo de recolección de datos, se tomaron las siguientes consideraciones. El despliegue al ambiente productivo no forma parte de la construcción, este se hizo luego de terminado el desarrollo del sistema. Sin embargo, es clave describir el modelo de despliegue para que se tenga un mejor entendimiento de los repositorios de almacenamiento durante el proceso de codificación.

A continuación, se muestra el diagrama de despliegue de código e imágenes entre entornos.

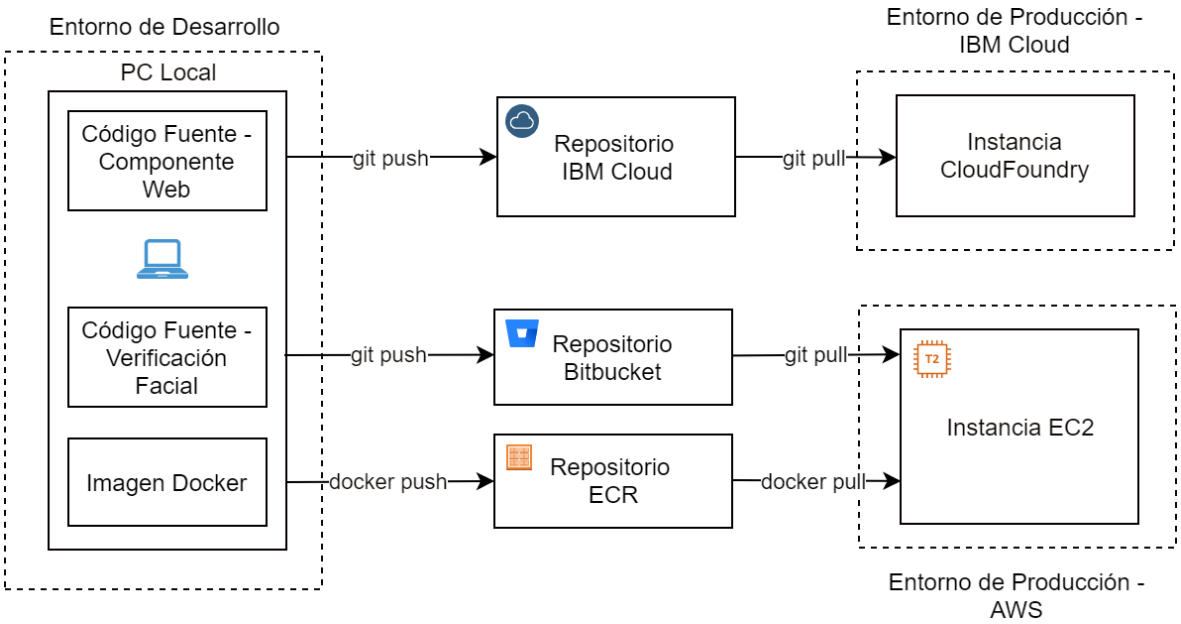


Figura 109. Despliegue de código de prototipo de pruebas automatizadas
Fuente: Elaboración Propia

Por un lado, se tiene el entorno de desarrollo que es donde se encuentra la PC o laptop local donde se construyen los algoritmos y se hacen las pruebas del sistema. Por otro lado, se tiene el entorno de producción dividido en IBM Cloud y AWS. Este ambiente es donde se despliega y posteriormente, ejecuta el prototipo de recolección de datos.

Las características de la PC de desarrollo local utilizada son las siguientes:

- Laptop Core i5-7300
- 8GB de RAM
- 250GB de disco SSD

Los tres elementos que se debieron desplegar desde el entorno de desarrollo hacia el entorno de producción fueron el código fuente del componente web, el código fuente del componente de verificación facial y la imagen Docker.

Como se puede observar en el diagrama de despliegue, se utilizó *git* como software de control de versiones. En el caso del código fuente del componente web se utilizó el repositorio *git* de IBM Cloud. En el caso del código fuente del componente de verificación facial, se utilizó el repositorio de Bitbucket para el almacenamiento. Los comandos que se muestran en el gráfico representan los comandos de subida y de descarga que son estándares de *git* (*git push*, *git pull*). El código fue desarrollado en una computadora local y almacenado en repositorios *git* en la nube de tal forma que pueda ser migrado a producción. Cualquier actualización en el código del componente web era enviado al repositorio de IBM Cloud y desplegado en la instancia de Cloud Foundry. Cualquier actualización en el código de verificación facial era enviada al repositorio de Bitbucket y desplegado en la instancia EC2.

Para la imagen Docker se realizó un proceso similar, pero utilizando el repositorio de imágenes de Amazon Web Services (AWS) denominado Elastic Container Registry (ECR). Cada actualización se subió a dicho repositorio para luego ser consumidas y ejecutadas por la instancia EC2.

Finalmente, para la ejecución del prototipo se tomaron en cuenta ciertos factores. En primer lugar, como se mencionó anteriormente, se tenían tres elementos a desplegar:

- La imagen Docker de *OpenFace*
- El directorio del código fuente del componente de verificación facial
- El directorio del código fuente del componente web

En el caso del entorno local, el proceso de ejecución era en dicha computadora. Todos los componentes se ejecutaban en un solo servidor local.

En el caso del entorno productivo, la ejecución del componente web se realizó en IBM Cloud. Este necesitó de un servidor para ejecutar código Python en cualquiera de los dos entornos.

La ejecución del componente de verificación facial incluyendo la imagen Docker se realizó en AWS. En ambos entornos, primero se construía el contenedor Docker desde la imagen haciendo un montaje de la carpeta del código fuente. Esta carpeta se montó como un volumen dentro del contenedor. Luego, dentro del contenedor se ejecutó el código fuente de las pruebas el cual estaba en lenguaje Python.

4. Pruebas

En la etapa de pruebas se realizaron las pruebas del funcionamiento correcto del prototipo. Estas se ejecutaron en el entorno local.

Las pruebas primero fueron unitarias a nivel de código. Entre estas se encuentran las siguientes:

- Pruebas del código de conexión a la base de datos
- Pruebas del código de almacenamiento en base de datos
- Pruebas del código de carga de archivos de imágenes.
- Pruebas del código de ejecución del modelo de verificación facial.
- Pruebas de exportación de datos.

Luego se realizaron pruebas de integración que buscaban verificar la integración entre lo probado en las pruebas unitarias:

- Pruebas del flujo de envío de datos: carga de imágenes → detección facial → preprocesamiento → verificación → descarga de base de datos de imágenes recolectadas.

4.2.2.2. Prototipo de pruebas automatizadas

Esta sección detalla las actividades y herramientas para el desarrollo del prototipo de pruebas automatizadas.

4.2.2.2.1. Comprensión del Negocio

Al desarrollar el prototipo de pruebas automatizadas se tiene como objetivo principal obtener un sistema capaz de hacer miles de pruebas del proceso de verificación facial eficientemente y obtener los resultados necesarios para poder responder las hipótesis planteadas en esta investigación.

4.2.2.2.2. Comprensión de Datos

Para el prototipo de pruebas automatizadas, se recopilaron las tres bases de datos establecidas. A continuación, se describen las actividades realizadas.

1. Descarga de base de datos de *Labeled Faces in the Wild* (LFW). Esta descarga se realizó entrando a la URL oficial de la base de datos: http://vis-www.cs.umass.edu/lfw/

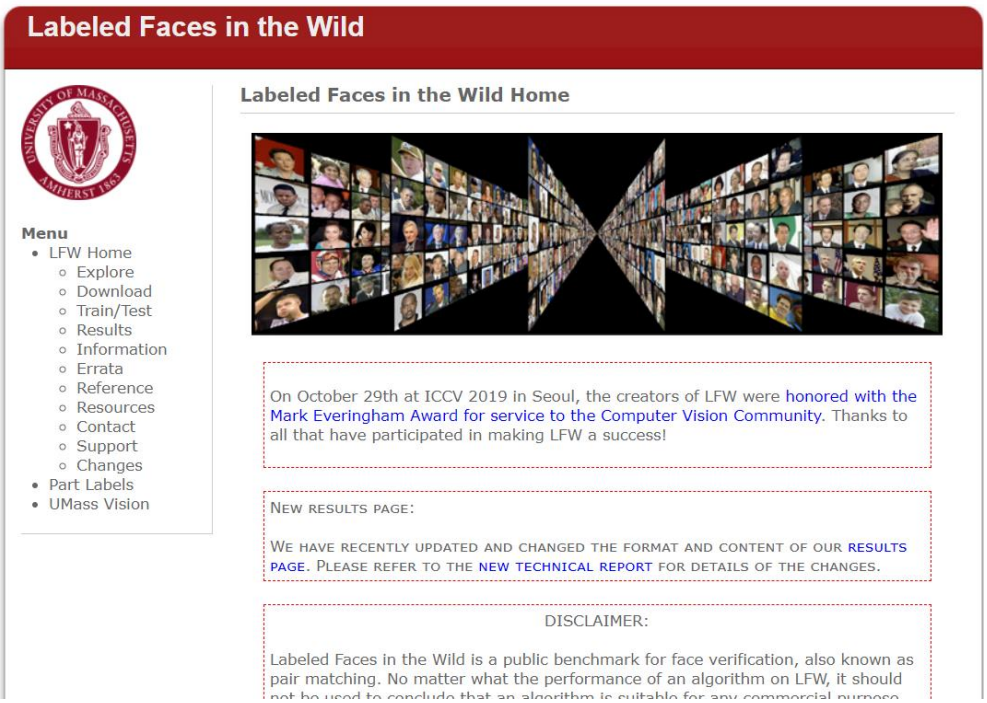


Figura 110. Página web oficial de fuente de información LFW
Fuente: Huang *(2019)*. *Labeled Faces in the Wild Home*

La descarga se realizó desde el enlace titulado *"All images gzipped tar file"*. Esto se muestra en la Figura 111.

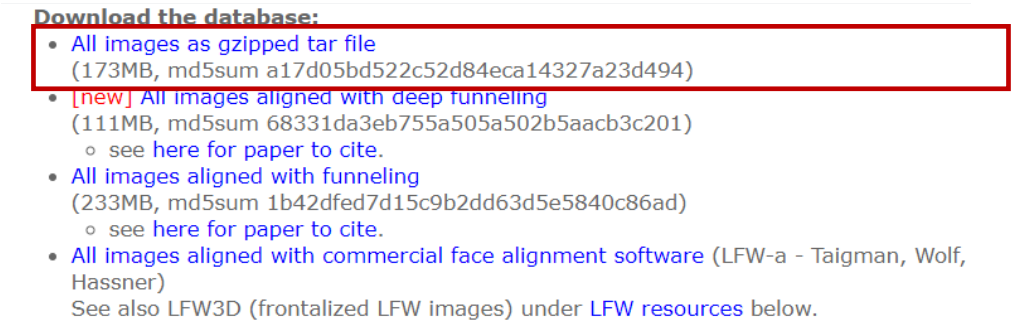


Figura 111. Enlace de descarga de fuente de información LFW
Fuente: Huang *(2019)*. *Labeled Faces in the Wild Home*

2. Descarga de base de datos *YouTube Faces Database* (YTF). Esta descarga se realizó entrando a la URL oficial de la base de datos: https://www.cs.tau.ac.il/~wolf/ytfaces/.

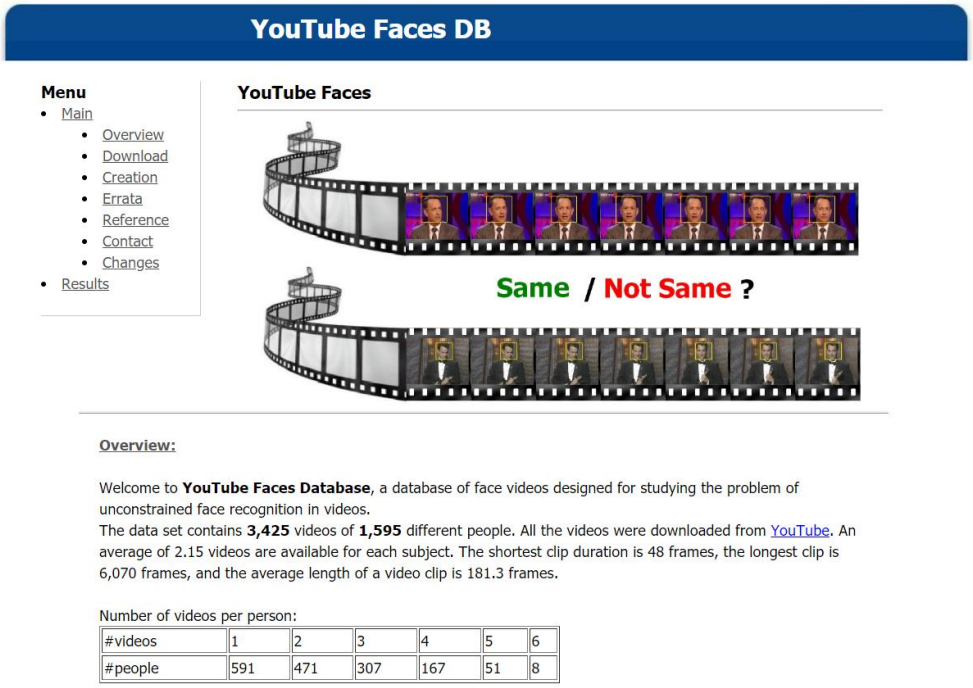


Figura 112. Página web oficial de fuente de información YTF
Fuente: Wolf, Hassner, & Maoz (2012). *YouTube Faces*

La descarga se realizó llenando el formulario de "Download the Database" que se encuentra en la misma página. Este formulario se muestra en la Figura 113.

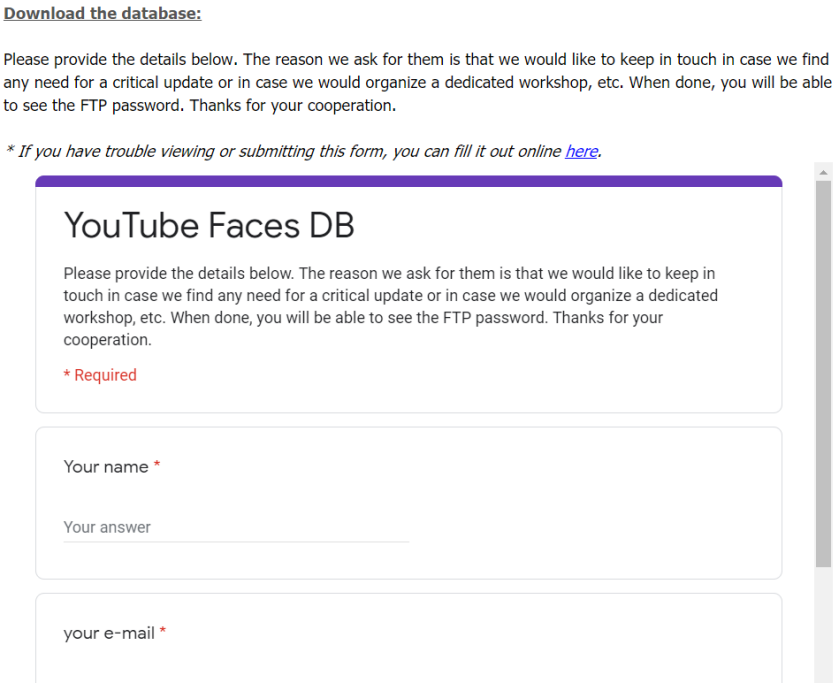


Figura 113. Formulario de descarga de fuente de información YTF
Fuente: Wolf, Hassner, & Maoz (2012). *YouTube Faces*

3. Recopilación y Descarga de Base de Datos del contexto local (DNI). La recopilación fue realizada solicitando a personas que llenen el formulario web del prototipo de recolección de datos.

   Una vez recopilada la información, esta se debió descargar. Esta descarga se realizó accediendo a las imágenes mediante su URL privada y almacenándolas en un repositorio local. A continuación, se muestra un ejemplo de un registro en la base de datos:


```
{
  "_id": "321a4bfb6c14a6aeaed9d0af50b288fc",
  "_rev": "2-97d7e522630cd8530f19ecbda7b2f1a8",
  "genero": "Femenino",
  "edad": "87",
  "resultado": 1.010233277306596,
  "fecha_hora": "2020-02-03 22:03:02",
  "_attachments": {
    "imagen2.jpg": {
      "content_type": "image/jpeg",
      "revpos": 2,
      "digest": "md5-/HTrjSvv/kWwymwoNKbuTA==",
      "length": 39805,
      "stub": true
    },
    "imagen1.jpg": {
      "content_type": "image/jpeg",
      "revpos": 2,
      "digest": "md5-i/KxMUK560GfK10U7foBQQ==",
      "length": 38724,
      "stub": true
    }
  }
}
```


Figura 114. Detalle registro de base de datos DNI
Fuente: Elaboración Propia

Se puede observar los siguientes campos principales:

- _id: El identificador único del registro.
- _rev: El identificador único de la versión del registro.
- genero: El género de la persona.
- edad: La edad de la persona.
- resultado: El resultado de la distancia entre ambas fotos de la persona.
- fecha_hora: El día y la hora que fue creado el registro

- attachments: Los dos archivos adjuntos "imagen1.jpg" e "imagen2.jpg". Estos archivos representan la foto del DNI y la foto del rostro de la personas.

A partir de cada registro, las imágenes se pudieron descargar, previa autenticación, mediante la URL que tenía la siguiente estructura:

- Imagen DNI:
  https://**{{instancia}}**.cloudant.com/**{{base_de_datos}}** /**{{id_registro}}**/imagen1.jpg
- Imagen Rostro:
  https://**{{instancia}}**.cloudant.com/**{{base_de_datos}}** /**{{id_registro}}**/image2.jpg

Donde:

- {{instancia}}: Es el identificador de la instancia del servicio de base de datos Cloudant.
- {{base_de_datos}}: Es la base de datos dentro de la instancia.
- {{id_registro}}: Es el campo _id que representa el identificador único del registro enviado.

Adicionalmente, en la etapa de comprensión de datos se realizó un análisis de las imágenes recolectadas:

1. Analizar base de datos de DNI

Se observó que la base de datos de DNI cuenta con imágenes en las cuales la posición del rostro de las personas generalmente es mirando al frente en ambas fotos. Sin embargo, la base de datos tiene cierta complejidad en las fotos del DNI dado que el rostro aparece pequeño y a un lado. Adicionalmente, el DNI agrega ruido a la foto del rostro del documento, haciéndolo más difícil de reconocer. Asimismo, la iluminación no está estandarizada en las imágenes, siendo algunas más oscuras o con luces de color.

2. Analizar base de datos LFW

Se observó que la base de datos de LFW tiene en su mayoría, mayor nitidez en las imágenes y menos ruido que la base de datos de DNI. Sin embargo, esta base de datos tiene una mayor variabilidad en la posición de los rostros, pudiendo no estar mirando de frente y en los ambientes en los que se encuentran las personas.

3. Analizar base de datos YTF

La base de datos de YTF es incluso más compleja en la posición de rostros que la base de datos de LFW. Además, al ser extractos de videos, algunas suelen tener mayor ruido y menor calidad que las imágenes de LFW. Por lo tanto, es una base de datos con una complejidad mayor. Asimismo, cuenta con variaciones en iluminación frecuentes.

4.2.2.2.3. Preparación de Datos

En primer lugar, dentro de la fase de preparación de datos, se detalla cómo se estructuraron las imágenes para cada una de las bases de datos utilizadas:

1. Estructurar data de Labeled Faces in the Wild

Esta base de datos se encuentra estructurada en base a carpetas con el nombre de cada persona. En cada carpeta se encuentran las imágenes con el nombre de la persona y una numeración. En la siguiente imagen se observa mejor la estructura:

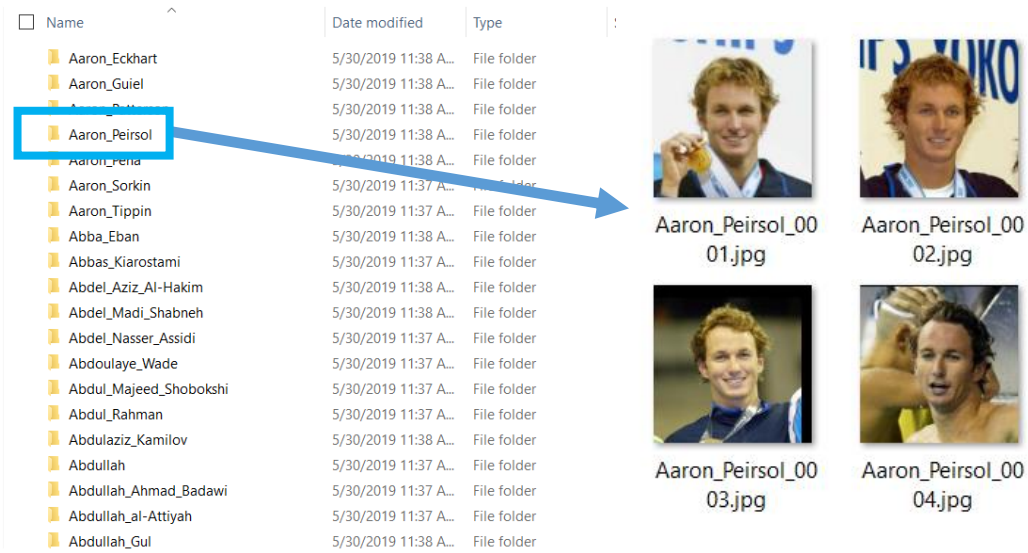


Figura 115. Detalle base de datos LFW
Fuente: Elaboración Propia

Asimismo, este tiene ya definido un set de imágenes probabilísticamente elegidas para ser usadas como muestra para las pruebas. Por lo tanto, se tomó en consideración la información de dicho set de imágenes y se configuró al sistema para leer la definición del *dataset* de prueba.

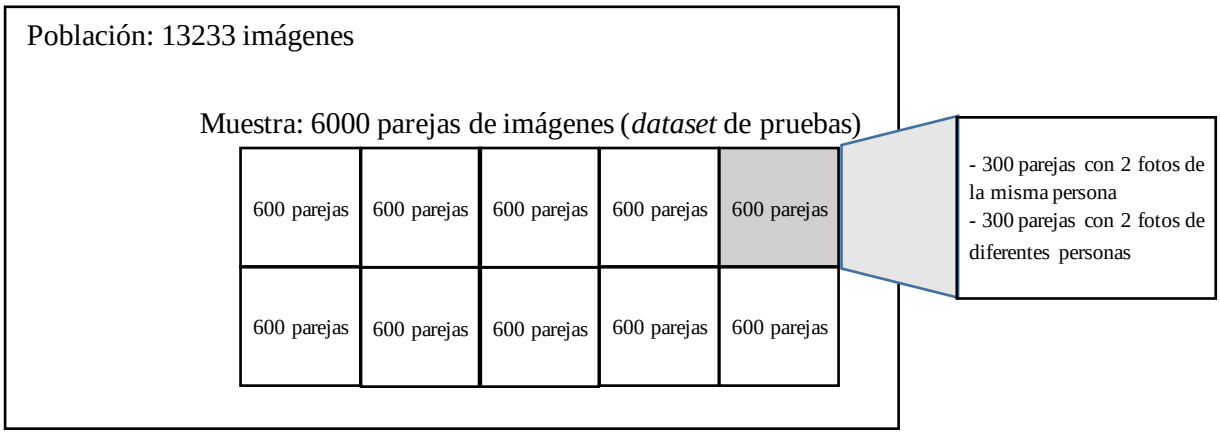


Figura 116. Estructura de la muestra de pruebas de LFW
Fuente: Elaboración Propia

En la Figura 116 se muestra las cantidades del subconjunto de pruebas. La definición del *dataset* de pruebas viene dividida en 10 cortes de 600 parejas de imágenes donde 300 parejas son de la misma persona y 300 parejas son de diferentes personas. Esta definición se encuentra en un archivo denominado *pairs.txt*. En la Figura 117 se observa un extracto del archivo.

```
Abel_Pacheco    1       4
Akhmed_Zakayev  1       3
Akhmed_Zakayev  2       3
Amber_Tamblyn   1       2
Anders_Fogh_Rasmussen   1       3
Anders_Fogh_Rasmussen   1       4
Angela_Bassett  1       5
Angela_Bassett  2       5
Angela_Bassett  3       4
Ann_Veneman     3       5
Ann_Veneman     6       10
Ann_Veneman     10      11
Anthony_Fauci   1       2
Antony_Leung    1       2
Antony_Leung    2       3
Anwar_Ibrahim   1       2
Augusto_Pinochet        1       2
Barbara_Brezigar        1       2
Benjamin_Netanyahu      1       4
Benjamin_Netanyahu      4       5
Bernard_Law     2       3
Bernard_Law     3       4
Bertrand_Bonello        1       2
Bill_Frist      2       9
Bill_Frist      4       5
Bob_Graham      2       4
Bob_Graham      3       6
Bob_Graham      4       6
Bob_Graham      5       6
Boris_Becker    2       6
Brad_Johnson    1       3
Brad_Johnson    2       3
Brad_Johnson    2       4
Brian_Griese    1       2
```

Figura 117. Detalle archivo pruebas.txt de LFW
Fuente: Elaboración Propia

Abajo se muestra la descripción de cada columna en caso sean pruebas de la misma persona:

1. Columna 1: Nombre de la primera persona
2. Columna 2: Número de imagen de la primera persona
3. Columna 3: Número de la otra imagen de la primera persona

Abajo se muestra la descripción de cada columna en caso sean pruebas de imágenes de diferentes personas:

1. Columna 1: Nombre de la primera persona
2. Columna 2: Número de imagen de la primera persona
3. Columna 3: Nombre de la segunda persona
4. Columna 4: Número de la otra imagen de la segunda persona.

2. Estructurar data de YouTube Faces Database

YouTube Faces DB provee imágenes extraídas de videos de *YouTube* en los cuales se encuentran rostros de distintas personas. Estas imágenes están estructuradas en base a carpetas con la siguiente estructura:

/{nombre_persona}/{número_video}/{número_video}.{número_de_fotograma}.jpg

Dentro de dicha estructura, {nombre_persona} es el nombre de la persona del video, {número_video} es el número del video, {número_de_fotograma} es el número del fotograma dentro del video. A continuación, se muestra esta estructura:

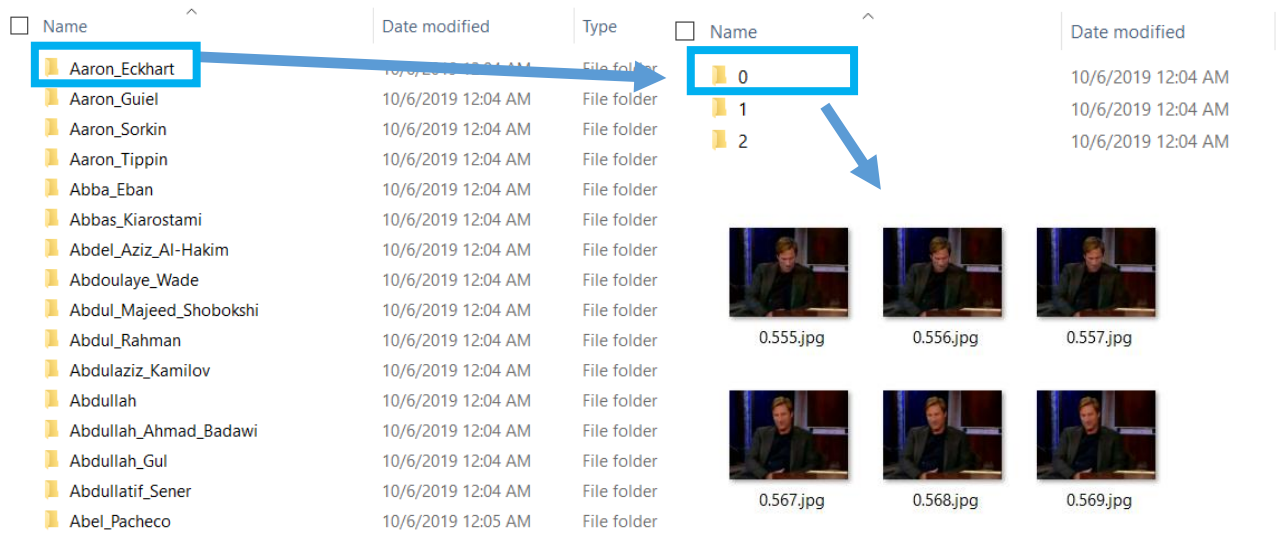


Figura 118. Detalle Base de Datos YouTube Faces DB
Fuente: Elaboración Propia

Por otro lado, la base de datos provee un archivo donde se define probabilísticamente una muestra para las pruebas siguiendo como ejemplo lo planteado por LFW. Este archivo define 5,000 parejas de videos.

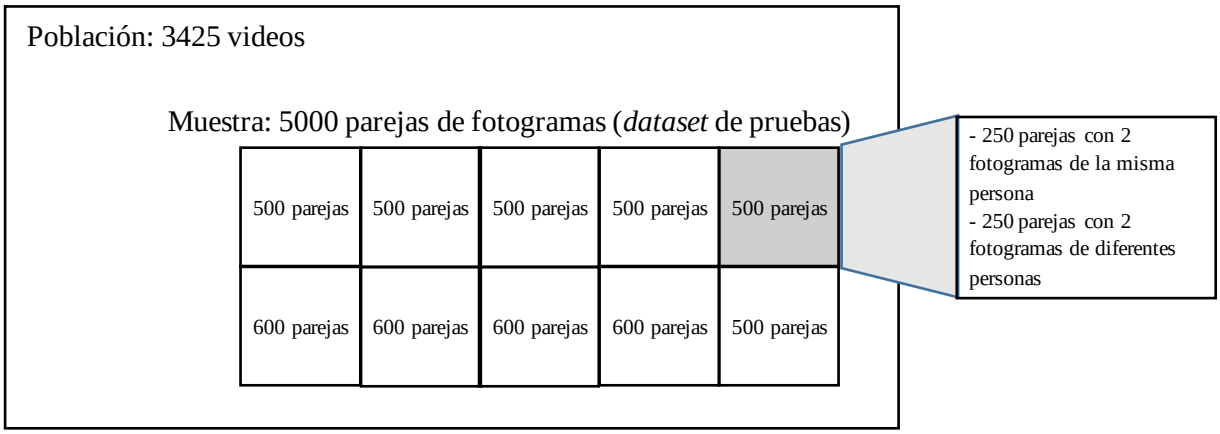


Figura 119. Estructura de la muestra de pruebas de YTF
Fuente: Elaboración Propia

Mitad son parejas de videos de la misma persona y mitad de videos de diferentes personas. Estas parejas fueron divididas en 10 cortes de 500 parejas. 250 parejas de cada corte de la misma persona y 250 parejas son de diferentes personas. De cada video, se eligió un fotograma al azar que lo representa. A continuación, se observa un extracto del archivo denominado *splits.txt* que contiene la definición de las 5,000 parejas:

```
split number, pair number in split, first name, second name, is same:
1, 1, Sadie_Frost/1, Sadie_Frost/5, 1
1, 2, Daniel_Bruehl/2, Daniel_Bruehl/3, 1
1, 3, James_Sensenbrenner/0, James_Sensenbrenner/1, 1
1, 4, James_Carville/0, James_Carville/3, 1
1, 5, Hana_Sadiq/0, Hana_Sadiq/1, 1
1, 6, Rachel_Hunter/4, Rachel_Hunter/5, 1
1, 7, Alicia_Silverstone/1, Alicia_Silverstone/2, 1
1, 8, Kevin_Borseth/2, Kevin_Borseth/4, 1
1, 9, Maria_Callas/1, Maria_Callas/2, 1
1, 10, Dave_Campo/1, Dave_Campo/2, 1
1, 11, Kristin_Chenoweth/1, Kristin_Chenoweth/5, 1
1, 12, Francis_Collins/0, Francis_Collins/3, 1
1, 13, Elijah_Wood/0, Elijah_Wood/3, 1
1, 14, William_McDonough/0, William_McDonough/2, 1
1, 15, Colleen_Atwood/3, Colleen_Atwood/5, 1
1, 16, Cecilia_Cheung/1, Cecilia_Cheung/2, 1
1, 17, Paul_Wilson/0, Paul_Wilson/4, 1
1, 18, Yasser_Arafat/0, Yasser_Arafat/4, 1
1, 19, Richard_Lugar/1, Richard_Lugar/2, 1
1, 20, Christine_Ebersole/2, Christine_Ebersole/5, 1
1, 21, Jan_Peter_Balkenende/0, Jan_Peter_Balkenende/1, 1
1, 22, Richard_Lugar/0, Richard_Lugar/5, 1
1, 23, James_Comey/3, James_Comey/4, 1
1, 24, Jan_Peter_Balkenende/0, Jan_Peter_Balkenende/5, 1
1, 25, Dave_Campo/1, Dave_Campo/3, 1
1, 26, George_W_Bush/0, George_W_Bush/1, 1
1, 27, George_W_Bush/1, George_W_Bush/4, 1
1, 28, Lyle_Lovett/2, Lyle_Lovett/5, 1
1, 29, Paul_Bremer/2, Paul_Bremer/5, 1
1, 30, Kristin_Chenoweth/4, Kristin_Chenoweth/5, 1
1, 31, Ludivine_Sagnier/0, Ludivine_Sagnier/3, 1
1, 32, Pedro_Duque/1, Pedro_Duque/4, 1
1, 33, Edgar_Savisaar/1, Edgar_Savisaar/2, 1
1, 34, Barry_Williams/2, Barry_Williams/3, 1
1, 35, William_McDonough/0, William_McDonough/4, 1
1, 36, Jan_Peter_Balkenende/1, Jan_Peter_Balkenende/5, 1
1, 37, Paul_Bremer/3, Paul_Bremer/5, 1
1, 38, Narendra_Modi/1, Narendra_Modi/3, 1
1, 39, Nicanor_Duarte_Frutos/2, Nicanor_Duarte_Frutos/5, 1
1, 40, Carey_Lowell/0, Carey_Lowell/2, 1
1, 41, William_McDonough/0, William_McDonough/3, 1
1, 42, Alicia_Silverstone/0, Alicia_Silverstone/5, 1
```

Figura 120. Detalle archivo splits.txt de YouTube Faces DB
Fuente: Elaboración Propia

A continuación se detalla el contenido de cada columna del archivo:

1. Partición de prueba
2. Número de pareja en la partición
3. Nombre y número de video de la primera persona
4. Nombre y número de video de la segunda persona
5. Son iguales: número binario que representa si son de la misma persona o de diferente persona

3. Estructurar data de Base de Datos de Contexto Local (DNI)

La base de datos del contexto local es una combinación de DNI y rostro de cada persona obtenido en base al prototipo de recolección de datos desarrollado. Las parejas de imágenes se almacenaron en una sola carpeta. Para diferenciarlas se les dio un nombre único por pareja y con un *1* o *2* al final dependiendo si eran DNI o imagen de rostro. Por ejemplo:

Imagen DNI:

- *fsawb141fsb_1.jpeg*

Imagen rostro:

- *fsawb141fsb_2.jpeg*

A continuación, se observa un extracto de la base de datos:

Figura 121. Extracto de base de datos de contexto local
Fuente: Elaboración Propia

Por otro lado, de acuerdo con lo descrito en la metodología, se desarrollaron todos los métodos de preprocesamiento mostrados. Estos se utilizan luego para realizar las pruebas.

4. Preprocesar con alineamiento

Para realizar el preprocesamiento con alineamiento, primero se hallaron los 68 puntos de referencia. Estos se pueden ver a continuación:

Figura 122. Puntos de referencia obtenidos de los rostros para la alineación
Fuente: Amos, Bartosz & Satyanaray (2016). *OpenFace: A general-purpose face recognition library with mobile applications.* (p. 6)

Luego, se aplicó el alineamiento. A continuación, se puede ver un ejemplo del alineamiento:

Figura 123. Aplicación de Alineamiento
Fuente: Elaboración Propia

Alineamiento 0:

- Solo se hace un recorte en base a lo detectado con el método de preprocesamiento. No existe alineamiento.

Alineamiento 1:

- Se hace una rotación y centrado en base a la posición de los ojos. Se utilizan los puntos 36 y 46 de la Figura 122, para realizar el alineamiento.

Alineamiento 2:

- Se hacen rotaciones, escalamientos y transformaciones en base a los puntos 36, 46 y 33 de la Figura 122 para realizar el alineamiento.

5. <u>Preprocesar con suavizamiento</u>

Adicionalmente, la Figura 124 muestra una comparación de los distintos filtros de suavizamiento:

Figura 124. Aplicación del filtro gausiano
Fuente: Elaboración Propia

En este caso es menos notorio que el alineamiento dado que con el filtro gausiano solo se hace un suavizamiento a la imagen.

Sin Gausiano:

- No se aplica ningún filtro.

Gausiano 3x3:

- Se aplica un filtro gausiano de tamaño 3x3.

Gausiano 5x5:

- Se aplica un filtro gausiano de tamaño 5x5.

6. Preprocesar con agudizamiento

La Figura 125 hace una comparación similar con el filtro de agudizamiento:

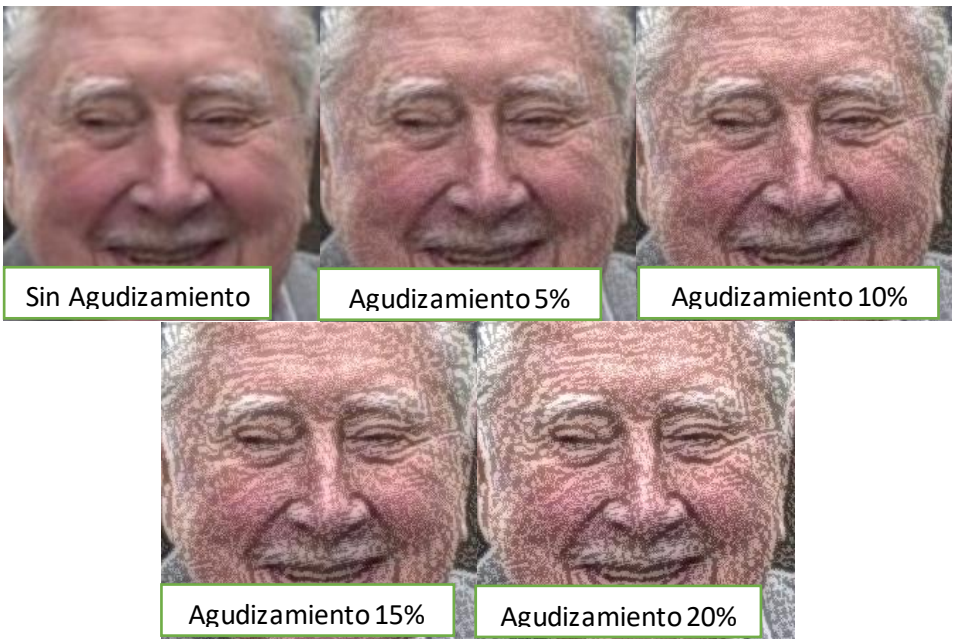


Figura 125. Aplicación de Filtro de Agudizamiento
Fuente: Elaboración Propia

Sin Agudizamiento:

- No se aplica ningún filtro.

Agudizamiento 5%:

- Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 5%.

Agudizamiento 10%:

- Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 10%.

Agudizamiento 15%:

- Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 15%.

Agudizamiento 20%:

- Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 20%.

7. Preprocesar con ecualización

Finalmente, la Figura 126 muestra los tipos de filtro de ecualización utilizados en la investigación:

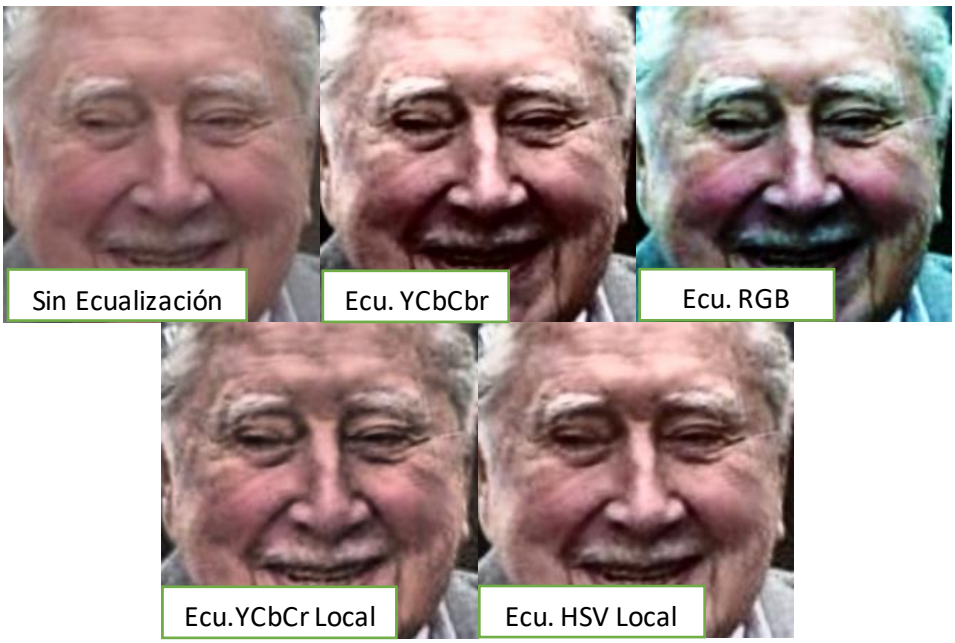


Figura 126. Aplicación de Filtro de Ecualización
Fuente: Elaboración Propia

Sin Ecualización:

- No se aplica ningún filtro.

Ecualización YCbCr:

- Se separan los canales y se ecualiza el canal de luminosidad de YCbCr normalizando dicho histograma.

Ecualización RGB:

- Se separan los canales y se ecualiza en todos los canales RGB independientemente normalizando dichos histogramas.

Ecualización YCbCr Local:

- Se separan los canales y se ecualiza por regiones locales el canal de luminosidad de YCbCr normalizando dicho histograma.

Ecualización HSV Local:

- Se separan los canales y se ecualiza por regiones locales el canal de luminosidad de HSV normalizando dicho histograma.

4.2.2.2.4. Modelado y Evaluación

En la etapa de modelado se desarrollaron las siguientes actividades:

1. Descarga del modelo de *VGGFace2*

Este se descargó del siguiente enlace usando git https://github.com/rcmalli/keras-vggface. Una imagen del repositorio se observa a continuación:

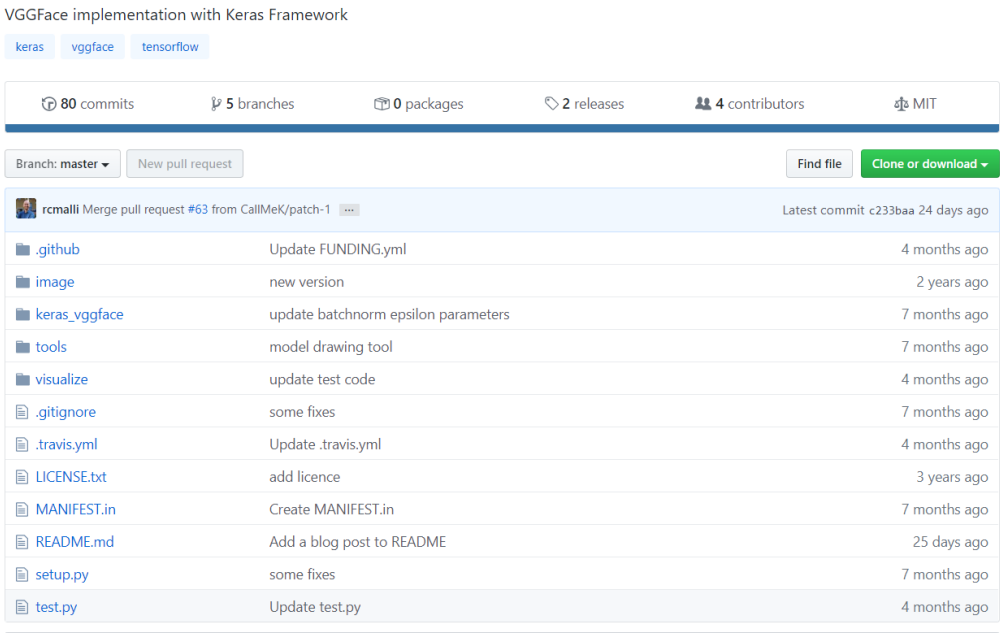


Figura 127. Detalle de repositorio del modelo VGGFace
Fuente: Malli (2017). *Keras VGGFace*

Dicho repositorio contiene tres variantes de *VGGFace* que se pueden utilizar incluyendo la versión de *VGGFace* del paper original. En este caso se utiliza la variante del paper de *VGGFace2* con bloques *Squeeze-and-Excitation* (SeNet).

2. Validación de dependencias del modelo de *VGGFace2*

En este caso las dependencias están documentadas dentro del mismo repositorio del modelo. El detalle se observa en la Figura 128.

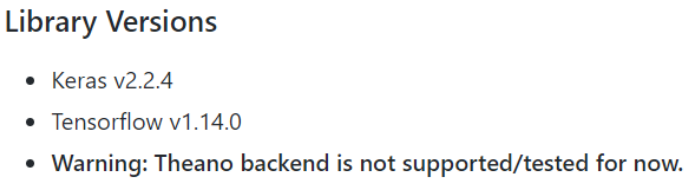


Figura 128. Detalle de dependencias del modelo VGGFace
Fuente: Malli (2017). *Keras VGGFace*

Como se puede observar se requirió las librerías Keras y TensorFlow. Estas librerías a su vez tienen otras dependencias para su instalación que se tuvieron que tomar en consideración.

3. Prueba del modelo de *VGGFace2*

Se creó un contenedor Docker con las dependencias necesarias instaladas y se realizó pruebas con parejas de imágenes para ver que los resultados obtenidos sean positivos y que el modelo no requiera dependencias adicionales.

4. Descarga del modelo *Light CNN*

El modelo *Light CNN* descargó del siguiente enlace usando git https://github.com/AlfredXiangWu/LightCNN. Una imagen del repositorio se observa a continuación:

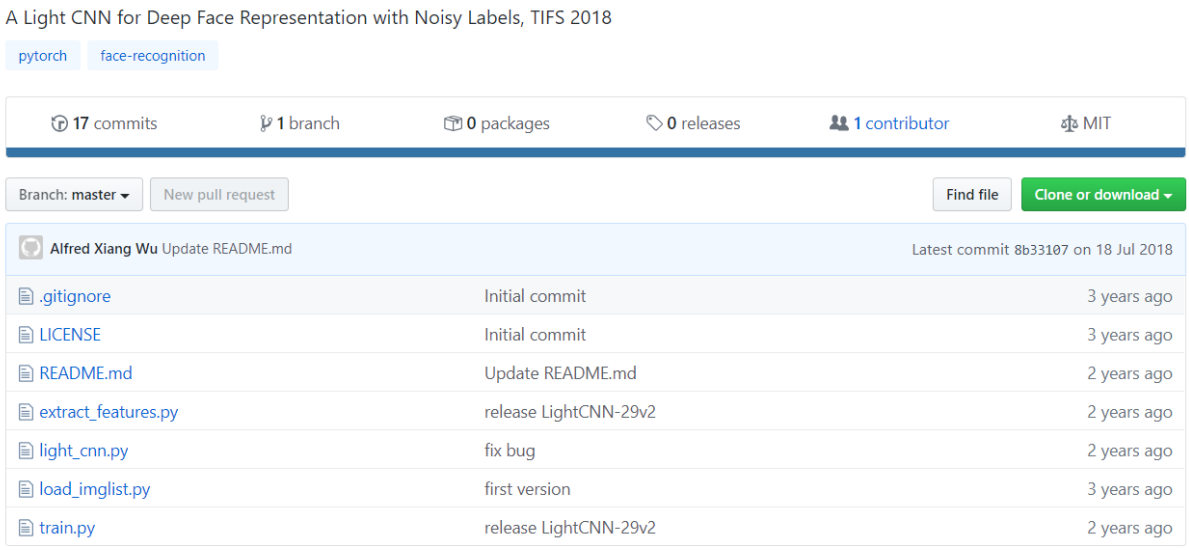


Figura 129. Detalle de repositorio del modelo Light CNN
Fuente: Xiang (2018). *Light CNN*

5. Validación de dependencias del modelo de Light CNN

En este caso las dependencias también estaban documentadas dentro del mismo repositorio del modelo. El detalle se observa en la Figura 130.

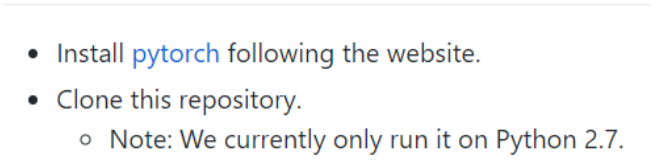


Figura 130. Detalle de dependencias del modelo Light CNN
Fuente: Xiang (2018). *Light CNN*

Como se puede observar se requirió Pytorch y la versión 2.7 de Python. Pytorch a su vez tiene otras dependencias para su instalación que se tuvieron que tomar en consideración.

6. Prueba del modelo de Light CNN

Se creó un contenedor Docker con las dependencias necesarias instaladas y se realizó pruebas con parejas de imágenes para ver que los resultados obtenidos sean positivos y que el modelo no requiera dependencias adicionales.

4.2.2.2.5. Implementación

Se estructuraron y se hicieron disponibles los componentes de código de analítica de datos de forma reutilizable para pasar al proceso de desarrollo de software del prototipo.

4.2.2.2.6. Desarrollo de Software

Aplicando la metodología de desarrollo de software cascada, se construyó el producto de datos del prototipo de pruebas automatizadas en base a las 4 etapas: análisis, diseño, construcción y pruebas.

1. Análisis

En la fase de análisis del desarrollo de software del prototipo de pruebas automatizadas se recabaron los requerimientos principales en base a los objetivos. A continuación, se muestran los principales requerimientos funcionales y no funcionales:

Tabla 6.

Requerimientos Funcionales – Prototipo de Pruebas Automatizadas

#	Descripción

RF01	El sistema debe procesar iterativamente todas las combinaciones de pruebas definidas.
RF02	El sistema debe cargar y procesar todas las imágenes almacenadas en las tres bases de datos: *LFW, YouTube Faces DB,* Base de datos de contexto local (DNI).
RF03	El sistema debe ejecutar los métodos de detección facial mediante características HOG y características HAAR.
RF04	El sistema debe ejecutar todos los tipos de métodos de preprocesamiento definidos: alineamiento, ecualización, filtro de suavizamiento y filtro de agudizamiento.
RF05	El sistema debe ejecutar los tres modelos de verificación facial: OpenFace, VGGFace2 y Light CNN.
RF06	El sistema debe generar los resultados de las pruebas y exportarlos a archivos.

Tabla 7.

Requerimientos No Funcionales – Prototipo de Pruebas Automatizadas

#	Descripción
RNF01	El sistema debe tener capacidad computacional de procesar gran cantidad de datos no estructurados.
RNF02	El sistema debe tener capacidad para almacenar gran cantidad de datos no estructurados.
RNF03	El sistema debe tener capacidades de ejecutar tareas en paralelo.
RNF04	El sistema debe permitir compatibilidad de integración entre componentes de código con dependencias diferentes.
RNF05	El sistema debe tener alta disponibilidad para evitar interrupciones durante la ejecución de las pruebas.

2. Diseño

Dentro del diseño del desarrollo de software del prototipo de pruebas automatizadas, se tomó en cuenta principalmente la arquitectura y cómo se ejecutarían las pruebas de forma eficiente. Esto tenía una complejidad adicional dado que la cantidad de pruebas es cuantiosa y se requieren varios días y capacidades computacionales para realizar todas las pruebas. Por lo tanto, se decidió buscar implementar paralelismo en la ejecución aprovechando que las pruebas son independientes. Además, se debía lograr este paralelismo sin perder control y rastro de cada componente.

Otro detalle importante era que cada modelo de verificación facial se ejecutaba con dependencias y código diferentes. Por este motivo, la primera separación de las pruebas fue a nivel del modelo de verificación facial. Se decidió tener tres componentes diferentes encargados de ejecutar cada prueba por modelo: *OpenFace*, *VGGFace2* y *Light CNN*.

A continuación, se muestra la arquitectura del componente encargado de la ejecución de las pruebas de *OpenFace*:

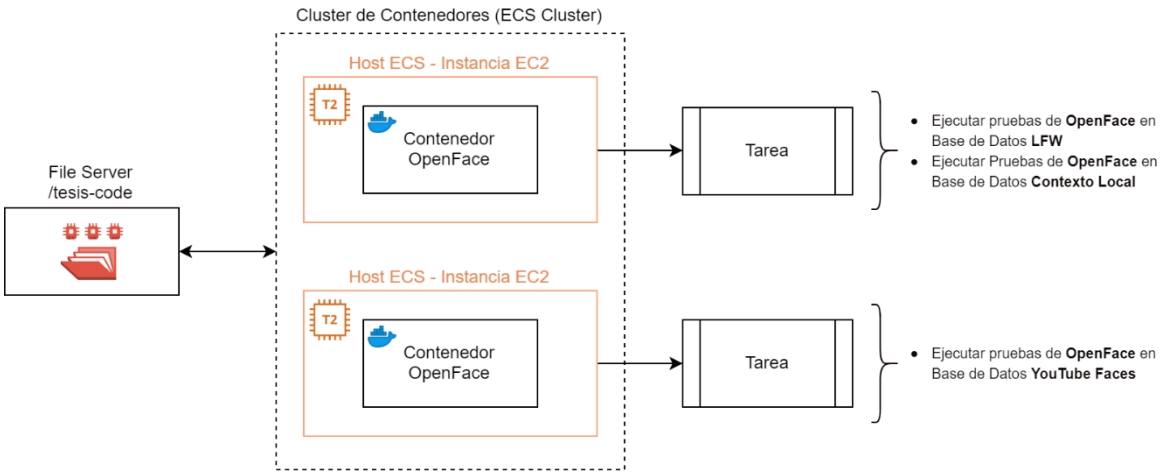


Figura 131. Arquitectura de Prototipo de Pruebas Automatizadas – OpenFace
Fuente: Elaboración Propia

Como se puede observar en la Figura, se decidió tener un clúster de contenedores Docker ejecutándose sobre distintos hosts (instancia EC2 de AWS). En este caso la arquitectura se basó en dos hosts dentro del clúster, ambos ejecutaban una instancia de la misma imagen Docker con el código de *OpenFace*. Sin embargo, cada uno ejecuta tareas diferentes. El primer host ejecuta pruebas de *OpenFace* en la base de datos LFW y en la del contexto local y el segundo host ejecuta pruebas de OpenFace en la base de datos de YouTube Faces DB. Cada host es una máquina virtual de baja capacidad (*t2.micro* de AWS) con el propósito de ahorrar costos y ambos host se ejecutan en paralelo. Adicionalmente, ambos host del clúster tienen acceso al

File Server (servidor de archivos) compartido donde se encuentra el código para ejecutar las pruebas.

Amazon EC2 (Elastic Compute Cloud) brinda las capacidades de infraestructura en la nube habilitando un servidor Linux sobre el cual se ejecuta el contenedor Docker. Además, para administrar y ejecutar los contenedores en la capa de infraestructura se utilizó Amazon ECS (Elastic Container Service). Asimismo, para almacenar y registrar la imagen Docker personalizada se utilizó Amazon ECR (Elastic Container Registry). Finalmente, para almacenar el código personalizado que se desarrolló para la ejecución del servicio se utilizó Amazon EFS (Elastic File Server) que permite almacenar archivos en un servidor independiente que puede montarse sobre cualquier instancia de Amazon EC2.

Una estructura similar se utilizó creando un clúster para las pruebas del modelo de verificación *VGGFace2*. Se utilizó un clúster independiente de servidores que se conectó al mismo *File Server* para obtener el código de las pruebas. Este clúster ejecutaba contenedores de *VGGFace2*. Esto se puede observar en la Figura a continuación:

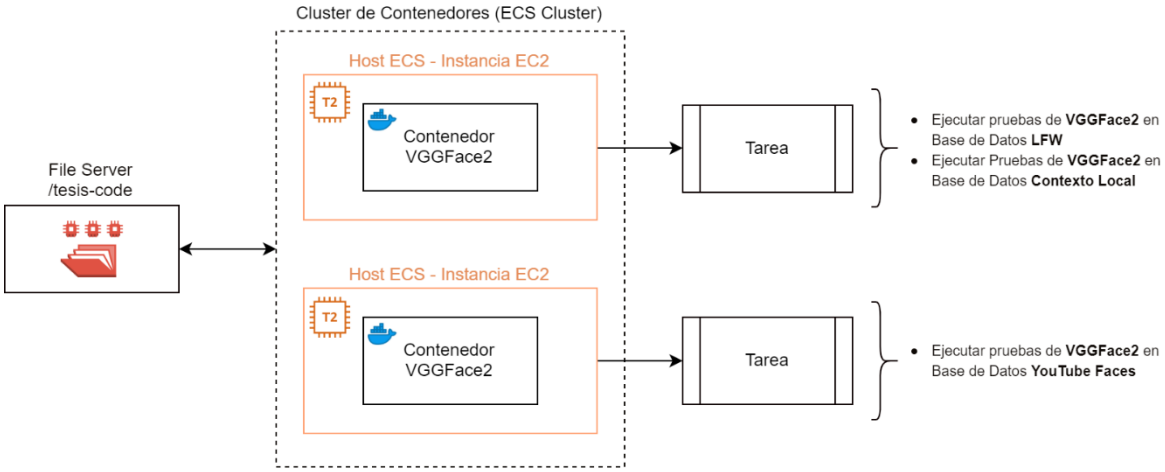
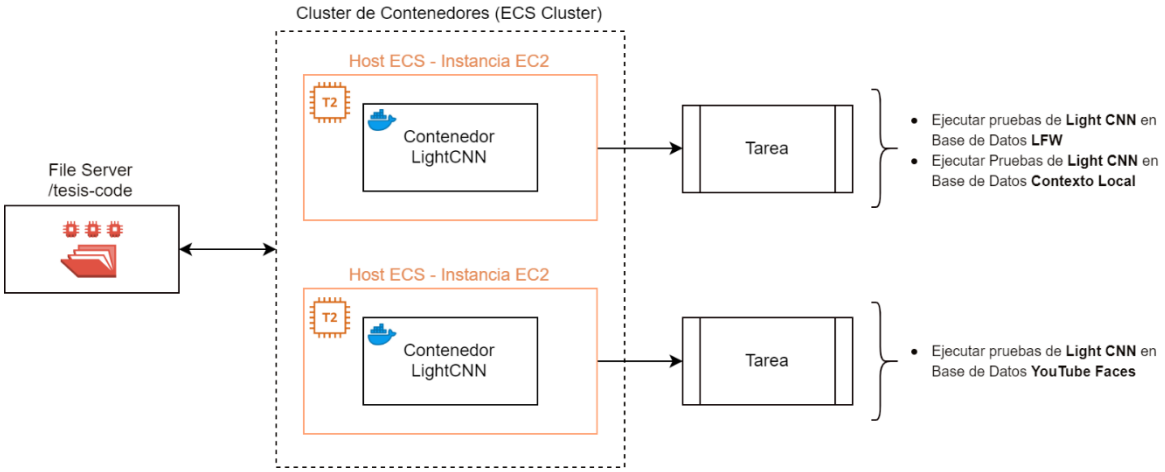


Figura 132. Arquitectura de Prototipo de Pruebas Automatizadas – VGGFace2
Fuente: Elaboración Propia

Finalmente, para el modelo de verificación *Light CNN*, se hizo de la misma manera:



Figuran 133. Arquitectura de Prototipo de Pruebas Automatizadas – Light CNN
Fuente: Elaboración Propia

En base a lo planteado, se ejecutaron en paralelo las pruebas divididas en hosts independientes. Estos hosts independientes se agruparon en clústeres que ejecutaban una imagen Docker de uno de los modelos de verificación facial. Lo realizado facilitó la eficiencia en recursos con máquinas virtuales de bajo costo y mayor velocidad al realizar tareas en paralelo. Asimismo, se mantuvo control centralizado al tener un solo *File Server* compartido y una subdivisión en tres clústeres administrados.

3. Construcción

En la construcción del prototipo de pruebas automatizadas se desarrolló todo lo planteado para poder lograr el resultado deseado.

El prototipo de pruebas automatizadas está compuesto por dos elementos principales:

- Las imágenes Docker implican el código necesario para que se ejecute el modelo de verificación facial. Se tiene una imagen por cada modelo de verificación: *OpenFace*, *VGGFace2* y *Light CNN*.
- El directorio del código fuente. Este incluye el código de las pruebas automatizadas que fue desarrollado en Python y se encarga de lo siguiente:
    - Cargar las imágenes de las bases de datos
    - Ejecutar todas las combinaciones de preprocesamiento
    - Ejecutar el modelo de verificación facial de la imagen Docker
    - Calcular las métricas de resultados de cada prueba
    - Exporta dichas métricas a archivos ".csv" con los resultados

Adicionalmente, este directorio incluye las imágenes para las pruebas de las 3 bases de datos y los resultados que se generaron de la ejecución de las pruebas.

Por otro lado, cabe describir el flujo de despliegue del prototipo de pruebas automatizadas. De igual manera que en el prototipo anterior, el despliegue en sí no forma parte de la etapa de construcción dado que este se hizo luego de terminado el desarrollo del sistema. Sin embargo, es clave describir el modelo de despliegue para que se tenga un mejor entendimiento de los repositorios de almacenamiento durante el proceso de codificación.

A continuación, se describe a mayor detalle el despliegue de código e imágenes entre entornos.

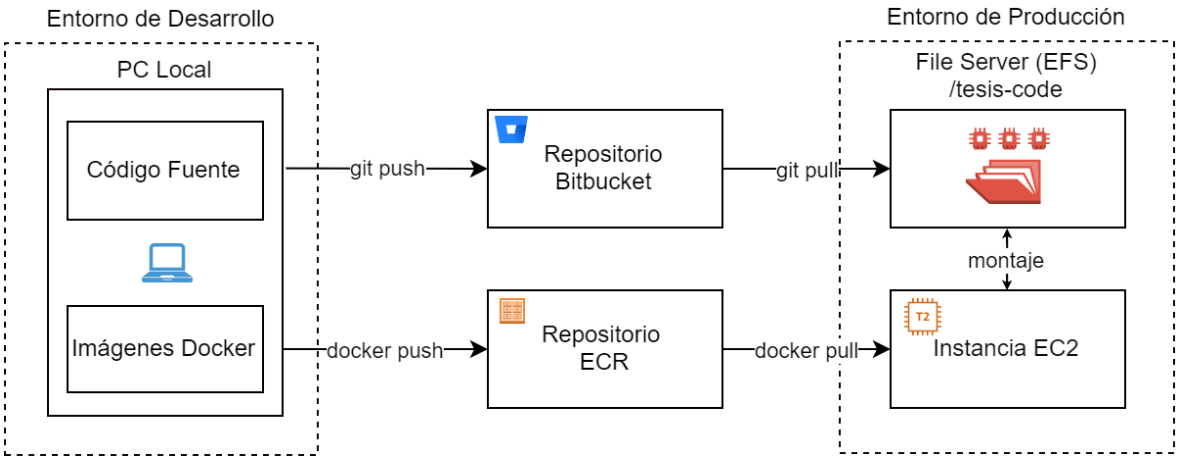


Figura 134. Despliegue de código de prototipo de pruebas automatizadas
Fuente: Elaboración Propia

Por un lado, se tiene el entorno de desarrollo que es donde se encuentra la PC o laptop local donde se construyen los algoritmos y se hacen las pruebas del sistema. Por otro lado, se tiene el entorno de producción que es el ambiente donde se despliega y posteriormente, ejecuta el prototipo de pruebas automatizadas.

Las características de la PC de desarrollo local son las mismas que la utilizada para el prototipo de recolección de datos.

Los dos elementos que se desplegaron desde el entorno de desarrollo hacia el entorno de producción fueron el código fuente y las imágenes Docker.

Como se puede observar en la Figura anterior, para el código fuente se utilizó *git* como software de control de versiones y Bitbucket como el repositorio de almacenamiento. El código en producción se almacenó en una carpeta /tesis-code en el File Server. Este código fue desarrollado en una computadora local y almacenado en un repositorio de Bitbucket en la nube

de tal forma que pueda ser migrado al *File Server* a través de dicho repositorio. Cualquier actualización en el código era enviado al repositorio Bitbucket. Luego, desde cualquier host que tuviese conexión al *File Server*, se podían descargar los cambios en el directorio.

Asimismo, siguiendo lo realizado en el prototipo de recolección de datos, para las imágenes Docker se utilizó el repositorio de imágenes de Amazon Web Services (AWS) denominado Elastic Container Registry (ECR). Cada actualización se subió a dicho repositorio para luego ser consumidas y ejecutadas por la instancia EC2.

Una vez ejecutadas las pruebas, se debía descargar los archivos de resultados al entorno de desarrollo local para su análisis. Para lograr esta descarga, el proceso se realizó a la inversa. Desde el servidor host del entorno productivo se subieron los archivos del *File Server* al repositorio *git* de Bitbucket (*git push*) para luego ser descargados desde la computadora local (*git pull*).

Finalmente, se describe las consideraciones que se tomaron para poder ejecutar este prototipo en cada uno de los entornos. En primer lugar, como se mencionó anteriormente, se tenían dos elementos a desplegar:

- Las imágenes Docker, una por cada modelo de verificación facial
- El directorio del código fuente

En el caso del entorno local, el proceso de ejecución era en dicha computadora. En el caso del entorno productivo, el proceso de ejecución se hizo por cada servidor host donde se desplegó una imagen. Como se mostró en la arquitectura se tuvieron seis servidores host diferentes, dos por cada imagen y en cada host se ejecutó un subconjunto diferente de las pruebas.

Sea en el entorno local o el productivo, primero se construía el contenedor Docker desde la imagen haciendo un montaje de la carpeta del código fuente. Esta carpeta se montó como un volumen dentro del contenedor. Luego, dentro del contenedor se ejecutó el código fuente de las pruebas.

4. Pruebas

En la etapa de pruebas se realizaron las pruebas del funcionamiento correcto del prototipo. Estas se ejecutaron en el entorno local.

Las pruebas primero fueron unitarias a nivel de código. Entre estas se encuentran las siguientes:

- Pruebas del código de carga de cada base de datos: LFW, YTF, DNI.

- Pruebas de ejecución del código de cada tipo de preprocesamiento.
- Pruebas de ejecución del código de cada tipo de método de detección facial.
- Pruebas de ejecución del código del modelo de verificación facial.
- Pruebas del código del cálculo de la distancia euclidiana.
- Pruebas del código del cálculo de las métricas de efectividad.
- Pruebas del código de la validación cruzada.
- Pruebas del código de exportación de resultados.

Luego se realizaron pruebas de integración que buscaban verificar la integración entre lo probado en las pruebas unitarias. Entre estas se encuentran las siguientes:

- Pruebas del flujo de ejecución de una imagen: carga de imagen → detección facial → preprocesamiento → verificación
- Pruebas del flujo de ejecución de una base de datos: carga de base de datos → detección facial → preprocesamiento → verificación → cálculo de efectividad → exportar resultados.

4.3. Medición de la Solución

4.3.1. Análisis de Indicadores Cuantitativos

A continuación, se analizarán los resultados cuantitativos encontrados en la investigación.

A partir de las pruebas realizadas por el prototipo de pruebas automatizadas se obtuvo un conjunto de resultados. El detalle de los resultados obtenidos, debido a la cantidad de información, se pueden descargar del siguiente repositorio: https://github.com/btafur/Tesis-2019.

Se dividió la evaluación en las 3 fuentes de información diferentes que a su vez tuvieron miles de pruebas diferentes.

Tabla 8.

Resumen de pruebas realizadas

Base de Datos	Tamaño Muestra	Modelo	# de Pruebas	# de Verificaciones Faciales
LFW	6,000 parejas	OpenFace	450 (x 10 cortes)	2,700,000
		VGGFace2	450 (x 10 cortes)	2,700,000
		Light CNN	450 (x 10 cortes)	2,700,000

YouTube Faces DB	5,000 parejas	OpenFace	450 (x 10 cortes)	2,250000
		VGGFace2	450 (x 10 cortes)	2,250000
		Light CNN	450 (x 10 cortes)	2,250000
Contexto Local	2,601	OpenFace	450 (x 4 cortes)	1,170,450
		VGGFace2	450 (x 4 cortes)	1,170,450
		Light CNN	450 (x 4 cortes)	1,170,450
TOTAL				18,361,350

Por cada base de datos y modelo se hicieron 450 pruebas, algunas de las cuales se hicieron en 10 cortes para el *cross-validation*. Las 450 pruebas se debieron a la combinación de métodos de detección y de preprocesamiento.

450 = 2 x 3 x 3 x 5 x 5*

* 2 métodos de detección, 3 métodos de alineamiento, 3 métodos de suavizamiento, 5 métodos de ecualización, 5 métodos de agudizamiento

Para calcular la cantidad de verificaciones faciales se multiplicó la cantidad de parejas de cada muestra por la cantidad de pruebas a realizar. Por ejemplo, para el caso de *OpenFace* en *LFW* se hicieron 450 pruebas en una muestra de 6000 parejas dando un total de 2,700,000 verificaciones faciales (450 x 6000 =2,700,000).

Por lo tanto, en total se hicieron 15,503,400 verificaciones faciales para realizar todas las pruebas.

Adicionalmente, se muestran los tiempos aproximados que tardó la ejecución total de todas las pruebas por cada modelo, en caso se hubiesen hecho de manera secuencial:

- *OpenFace*: 20 días
- *VGGFace2*: 36 días
- *Light CNN*: 28 días

El modelo de *OpenFace* tardó menos en realizar las pruebas mientras que *Light CNN* tuvo tiempos de ejecución más lentos. Como se comentó anteriormente, la ejecución se realizó en servidores paralelos por lo que la duración fue de aproximadamente 2 veces menos por cada modelo.

En los resultados que se mostrarán a continuación, la efectividad del modelo hará referencia al indicador de exactitud (*accuracy*) de la matriz de operacionalización de variables, dado que es el más representativo. Asimismo, más adelante se analizarán a detalle los otros indicadores de efectividad en las combinaciones más relevantes encontradas.

Se logró una efectividad de hasta 98.18% en LFW, 85.72% en *YouTube Faces DB* y 93.62% en la base de datos de DNIs. En el caso de LFW y la base de datos de DNIs, se logró dicha efectividad mediante el modelo de *Light CNN*. En el caso de YTF fue mediante el modelo *VGGFace2*. Todos estos resultados fueron mediante el alineamiento de ojos. Las combinaciones de los otros métodos de preprocesamiento difirieron.

A continuación, se muestra un resumen de los máximos resultados de efectividad obtenidos por fuente de información y modelo de verificación facial. Se detalla también los métodos de preprocesamiento que facilitaron lograr dicha efectividad en cada resultado.

Máxima Efectividad por Modelo de Verificación y Fuente de Info.			
FUENTE INFO.	**MODELO DE VERIFICACIÓN FACIAL**		
	OpenFace	**VGG-Face2**	**LightCNN**
DNI	**87.93%** **Detección:** HAAR **Alineamiento:** Ojos - Nariz **Suavizamiento:** Filtro 5x5 **Afinamiento:** 0% **Ecualización:** Ninguna	**91.19%** **Detección:** HAAR **Alineamiento:** Ninguno **Suavizamiento:** Ninguno **Afinamiento:** 0% **Ecualización:** Local YCbCr	**93.62%** **Detección:** HAAR **Alineamiento:** Ojos **Suavizamiento:** Filtro 5x5 **Afinamiento:** 0% **Ecualización:** Ninguna
LFW	**92.97%** **Detección:** HOG **Alineamiento:** Ojos - Nariz **Suavizamiento:** Ninguno **Afinamiento:** 0% **Ecualización:** Ninguna	**97.63%** **Detección:** HOG **Alineamiento:** Ojos **Suavizamiento:** Ninguno **Afinamiento:** 20% **Ecualización:** YCbCr	**98.18%** **Detección:** HOG **Alineamiento:** Ojos **Suavizamiento:** Ninguno **Afinamiento:** 0% **Ecualización:** Ninguna
YTF	**80.36%** **Detección:** HOG **Alineamiento:** Ojos - Nariz **Suavizamiento:** Ninguno **Afinamiento:** 0% **Ecualización:** Ninguna	**85.72%** **Detección:** HOG **Alineamiento:** Ojos **Suavizamiento:** Filtro 5x5 **Afinamiento:** 20% **Ecualización:** Ninguna	**85.04%** **Detección:** HOG **Alineamiento:** Ojos **Suavizamiento:** Filtro 5x5 **Afinamiento:** 0% **Ecualización:** Ninguna

Figura 135. Resumen de los mejores resultados de las pruebas de efectividad realizadas por fuente de información y modelo de verificación
Fuente: Elaboración Propia

Para ver los valores más a detalle del resumen de resultados, se puede ver el Anexo 2. Asimismo, para ver toda la volumetría completa de resultados, estos se pueden descargar del repositorio https://github.com/btafur/Tesis-2019.

En las siguientes subsecciones, se entrará a detalle en los resultados de efectividad por cada método de preprocesamiento, modelo de verificación facial, método de detección facial y

fuente de información. Luego, se procederá a mostrar los mejores resultados a nivel de efectividad y, finalmente, se analizarán los mejores resultados.

4.3.1.1. Resultados de Método de Preprocesamiento – Alineamiento – HE2

En primer lugar, se muestran los resultados del método de preprocesamiento de alineamiento versus la efectividad del modelo. Los resultados están diferenciados por las bases de datos LFW, YTF y DNI. Asimismo, están diferenciados por modelo de verificación facial *OpenFace*, *VGGFace2* y *Light CNN*. Finalmente, se diferencian por descriptores utilizados para la detección facial HOG o Haar. Los resultados de alineamiento se referencian mediante *AL_0*, *AL_1* y *AL_2*:

- *AL_0*: Sin alineamiento
- *AL_1*: Con alineamiento de ojos y nariz
- *AL_2*: Con alineamiento de ojos

Estos resultados se resumen a continuación:

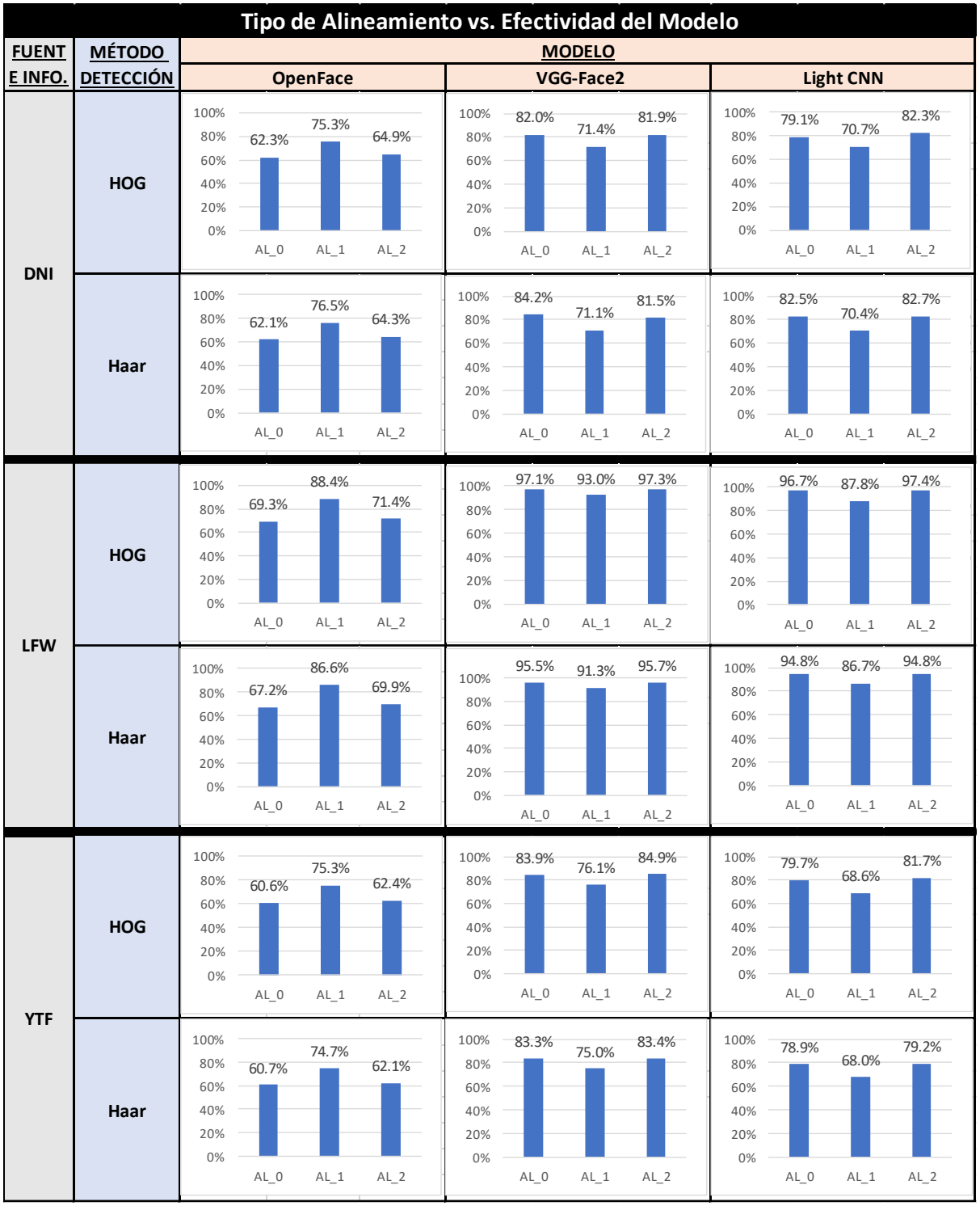


Figura 136. Resultados de Alineamiento vs. Efectividad del Modelo
Fuente: Elaboración Propia

Como se puede observar en la tabla, en el caso del modelo de *OpenFace*, tanto el alineamiento de ojos y nariz (*AL_1*) como el alineamiento de ojos (*AL_2*) mostraron una mayor efectividad promedio.

En el caso de *VGGFace2*, se logró una mayor efectividad promedio con el alineamiento de ojos (*AL_2*) y una menor efectividad con el alineamiento de ojos y nariz (*AL_1*).

Además, con *Light CNN*, se tuvo también una mayor efectividad promedio con el alineamiento de ojos (*AL_2*) y una menor efectividad con el alineamiento de ojos y nariz (*AL_1*).

Una razón por la que el método de alineamiento *AL_1* genera menor efectividad se puede deber a la mayor distorsión de la imagen al transformarla en base a 3 puntos (nariz y dos ojos). La razón por la que *OpenFace* genera una mayor efectividad mediante este método se debe posiblemente a que dicho modelo tomó ese alineamiento en consideración como parte de su entrenamiento.

A continuación, se evaluó la relación entre las variables mediante un análisis bivariado. En caso se obtenga un resultado de medias distintas, quiere decir que existe relación entre variables. Se plantea las siguientes hipótesis para la prueba:

H_0: Todos los tipos de alineamiento tienen media de efectividad iguales.

H_1: Todos los tipos de alineamiento no tienen media de efectividad iguales.

Utilizando el análisis de comparación de medias con nivel de confianza de 0.95 (alfa de 0.05) se obtiene lo siguiente:

ANOVA						
	Suma de cuadrados	*gl*	*Media cuadrática*	*F*	*valor P*	*F crit*
Inter-grupos	2002.8484	2	1001.42419	7.845212	0.000398	2.997951
Infra-grupos	516590.69	4047	127.6478115			
Total	518593.54	4049				

Figura 137. Comparación entre medias de efectividad de los tipos de alineamiento – Análisis ANOVA
Fuente: Elaboración Propia

- El valor *p-value* es de casi 0 y es menor a alfa (0.05)
- Por lo tanto, se rechaza H0 y se determina que los promedios de los grupos no son iguales.

A partir de esta prueba, se demuestra estadísticamente que el alineamiento está relacionado con la efectividad del modelo.

4.3.1.2. Resultados de Método de Preprocesamiento – Suavizamiento – HE2

En el caso del método de preprocesamiento de suavizamiento versus la efectividad del modelo se realizó el mismo análisis.

Los resultados de suavizamiento se referencian mediante *G_0*, *G_3* y *G_5*:

- *G_0*: Sin suavizamiento. No se aplica este filtro.
- *G_3*: Se aplica un filtro gausiano de tamaño 3x3.
- *G_5*: Se aplica un filtro gausiano de tamaño 5x5.

Estos resultados se resumen a continuación:

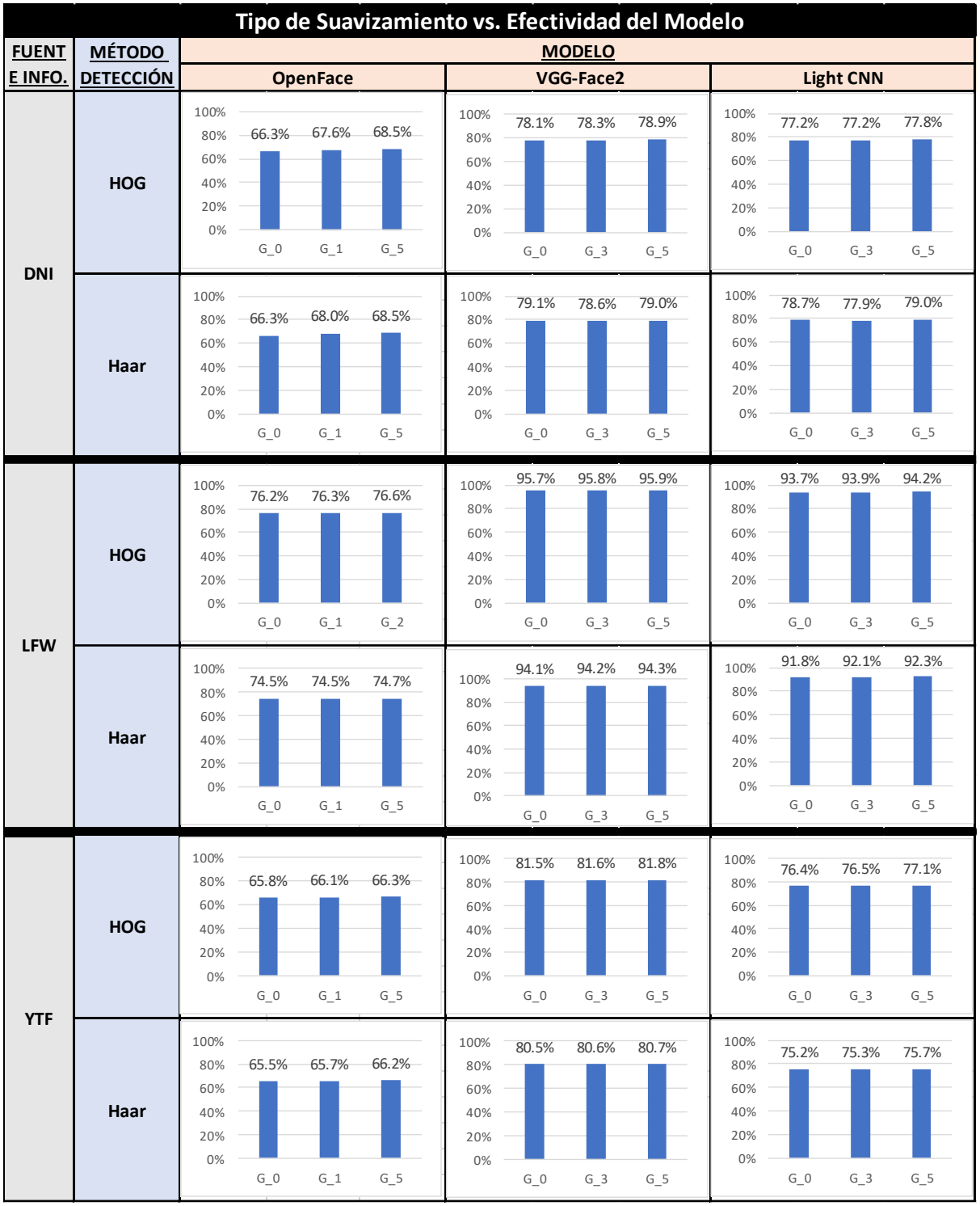


Figura 138. Resultados de Suavizamiento vs. Efectividad del Modelo
Fuente: Elaboración Propia

En base a lo mostrado en la tabla, en el caso del modelo de *OpenFace*, el suavizamiento genera poca variación en la efectividad. Sin embargo, se observa que tiende ligeramente a generar una efectividad mayor en promedio. Siendo el suavizamiento con filtro gausiano de 5x5 el que tuvo una mayor efectividad promedio.

En el caso de *VGGFace2*, se genera una mayor efectividad promedio en LFW y YTF mediante el suavizamiento. Sin embargo, sigue siendo por una pequeña diferencia.

Adicionalmente, con *Light CNN*, sigue existiendo poca variación en la efectividad, pero también se obtuvo una mayor efectividad promedio con el suavizamiento en casi todos los casos, siendo el filtro gausiano 5x5 el que generó una mayor efectividad promedio.

Como se observó en los ejemplos de gausiano, tiende a ser un filtro con un impacto leve en la imagen. Esto puede que contribuya a que no afecte negativamente los resultados siendo un filtro que genera poca distorsión.

Por otro lado, se evaluó la relación entre el método de preprocesamiento de suavizamiento y la efectividad mediante un análisis bivariado. Se plantea las siguientes hipótesis para la prueba:

H_0: Todos los tipos de suavizamiento tienen media de efectividad iguales.

H_1: Todos los tipos de suavizamiento no tienen media de efectividad iguales.

Utilizando el análisis de comparación de medias con nivel de confianza de 0.95 (alfa de 0.05) se obtiene lo siguiente:

ANOVA						
Source of Variation	*SS*	*df*	*MS*	*F*	*P-value*	*F crit*
Between Groups	251.91286	2	125.9564291	0.983416	0.37412043	2.997951
Within Groups	518341.63	4047	128.0804618			
Total	518593.54	4049				

Figura 139. Comparación entre medias de efectividad de los tipos de suavizamiento – Análisis ANOVA
Fuente: Elaboración Propia

- El valor *p-value* es de 0.3741 y es mayor a alfa (0.05)
- Por lo tanto, se acepta H0 y se determina que los promedios de los grupos son iguales.

A partir de esta prueba, se establece que existe poca variación entre las medias de ambos grupos por lo que el suavizamiento no tiene una relación determinante con la efectividad del modelo. Esto se debe a la poca diferencia entre los resultados obtenidos por cada tipo de suavizamiento.

4.3.1.3. Resultados de Método de Preprocesamiento – Agudizamiento – HE2

Para el análisis del método de preprocesamiento de agudizamiento versus la efectividad del modelo realizó un análisis similar. Los resultados de agudizamiento se referencian mediante *AF_00*, *AF_05*, *AF_10*, *AF_15* y *AF_20*:

- *AF_00*: Sin agudizamiento. No se aplica este filtro.
- *AF_05*: Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 5%.
- *AF_10*: Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 10%.
- *AF_15*: Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 15%.
- *AF_20*: Se obtienen los cambios en contraste de la imagen mediante un filtro de agudizamiento y se promedia con el resto de la imagen ponderando dicho resultado al 20%.

Estos resultados se resumen a continuación:

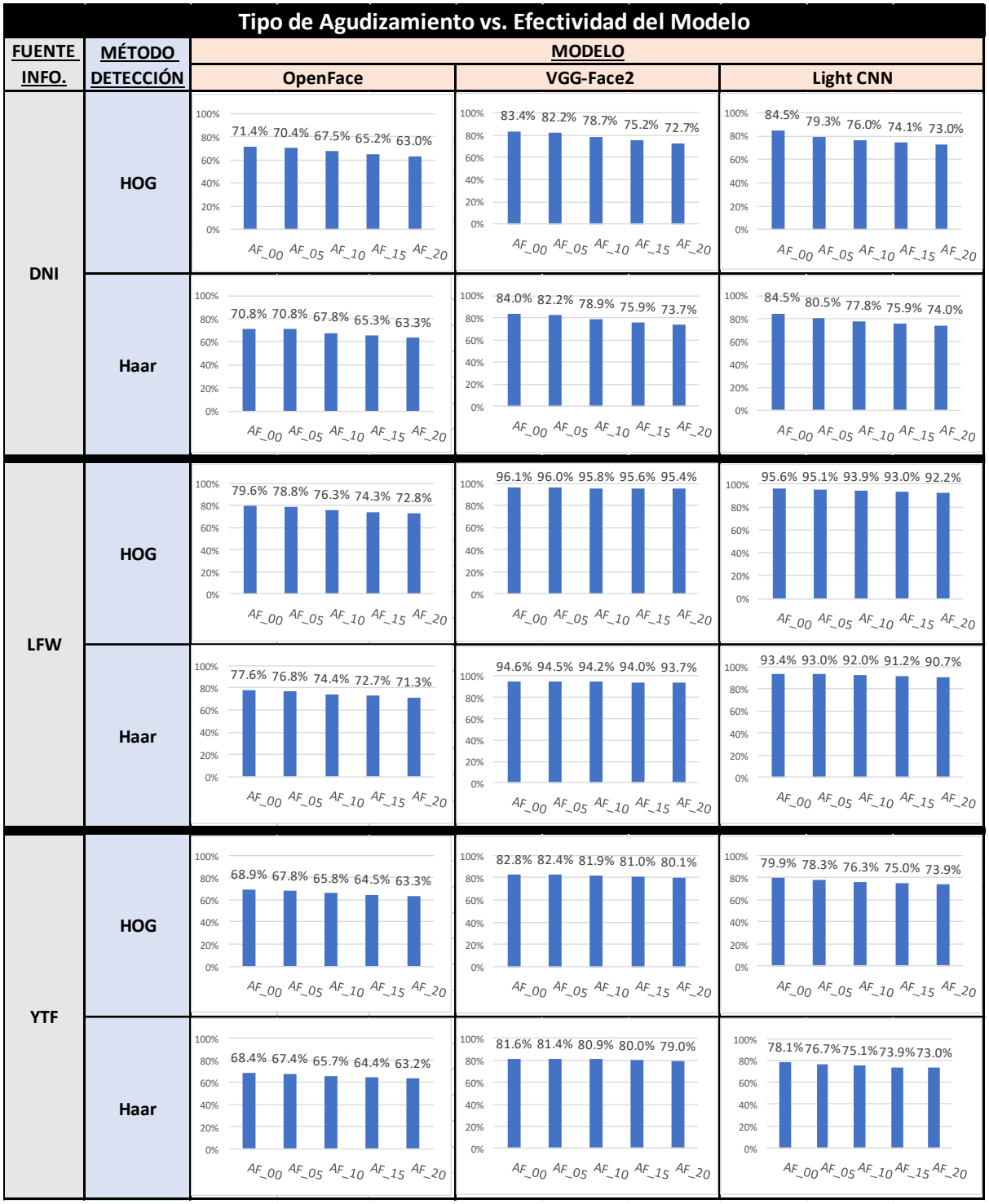


Figura 140. Resultados de Agudizamiento vs. Efectividad del Modelo
Fuente: Elaboración Propia

Como se puede observar en la tabla, en el caso del modelo de *OpenFace*, el agudizamiento tiende a generar una menor efectividad en promedio. Incluso la tendencia es que mientras mayor es el porcentaje de agudizamiento, mayor es el decrecimiento en efectividad promedio. En el caso de *VGGFace2* y *Light CNN*, se observan resultados muy similares.

El decrecimiento en efectividad se puede deber a que el agudizamiento genera cambios fuertes en los colores y los bordes de la imagen, los cuales pueden afectar al resultado del algoritmo de verificación.

Adicionalmente, se evaluó la relación entre el método de preprocesamiento de agudizamiento y la efectividad mediante un análisis bivariado. Se plantea las siguientes hipótesis para la prueba:

H_0: Todos los tipos de agudizamiento tienen media de efectividad iguales.

H_1: Todos los tipos de agudizamiento no tienen media de efectividad iguales.

Utilizando el análisis de comparación de medias con nivel de confianza de 0.95 (alfa de 0.05) se obtiene lo siguiente:

ANOVA						
	Suma de cuadrados	*gl*	*Media cuadrática*	*F*	*valor P*	*F crit*
Inter-grupos	19176.94	4	4794.234943	38.83067	6.06958E-32	2.374129
Infra-grupos	499416.6	4045	123.4651674			
Total	518593.54	4049				

Figura 141. Comparación entre medias de efectividad de los tipos de agudizamiento – Análisis ANOVA
Fuente: Elaboración Propia

- El valor *p-value* es de casi 0 y es menor a alfa (0.05)
- Por lo tanto, se rechaza H0 y se determina que los promedios de los grupos no son iguales.

A partir de esta prueba, se demuestra estadísticamente que el agudizamiento está relacionado con la efectividad del modelo.

4.3.1.4. Resultados de Método de Preprocesamiento – Ecualización – HE2

Como último método de preprocesamiento se planteó la ecualización. Los resultados de la ecualización se referencian mediante *E_00, E_LHH, E_LHY, E_RGB* y *E_YCB*:

- *E_000*: Sin ecualización. No se aplica este procesamiento.
- *E_YCB*: Se realiza ecualización YCbCr. Se separan los canales y se ecualiza el canal de luminosidad de YCbCr normalizando dicho histograma.
- *E_RGB*: Se realiza ecualización RGB. Se separan los canales y se ecualiza en todos los canales RGB independientemente normalizando dichos histogramas.

- *E_LHY*: Se aplica ecualización YCbCr local. Se separan los canales y se ecualiza por regiones locales el canal de luminosidad de YCbCr normalizando dicho histograma.
- *E_LHH*: Se aplica ecualización HSV local. Se separan los canales y se ecualiza por regiones locales el canal de luminosidad de HSV normalizando dicho histograma.

Estos resultados se resumen a continuación:

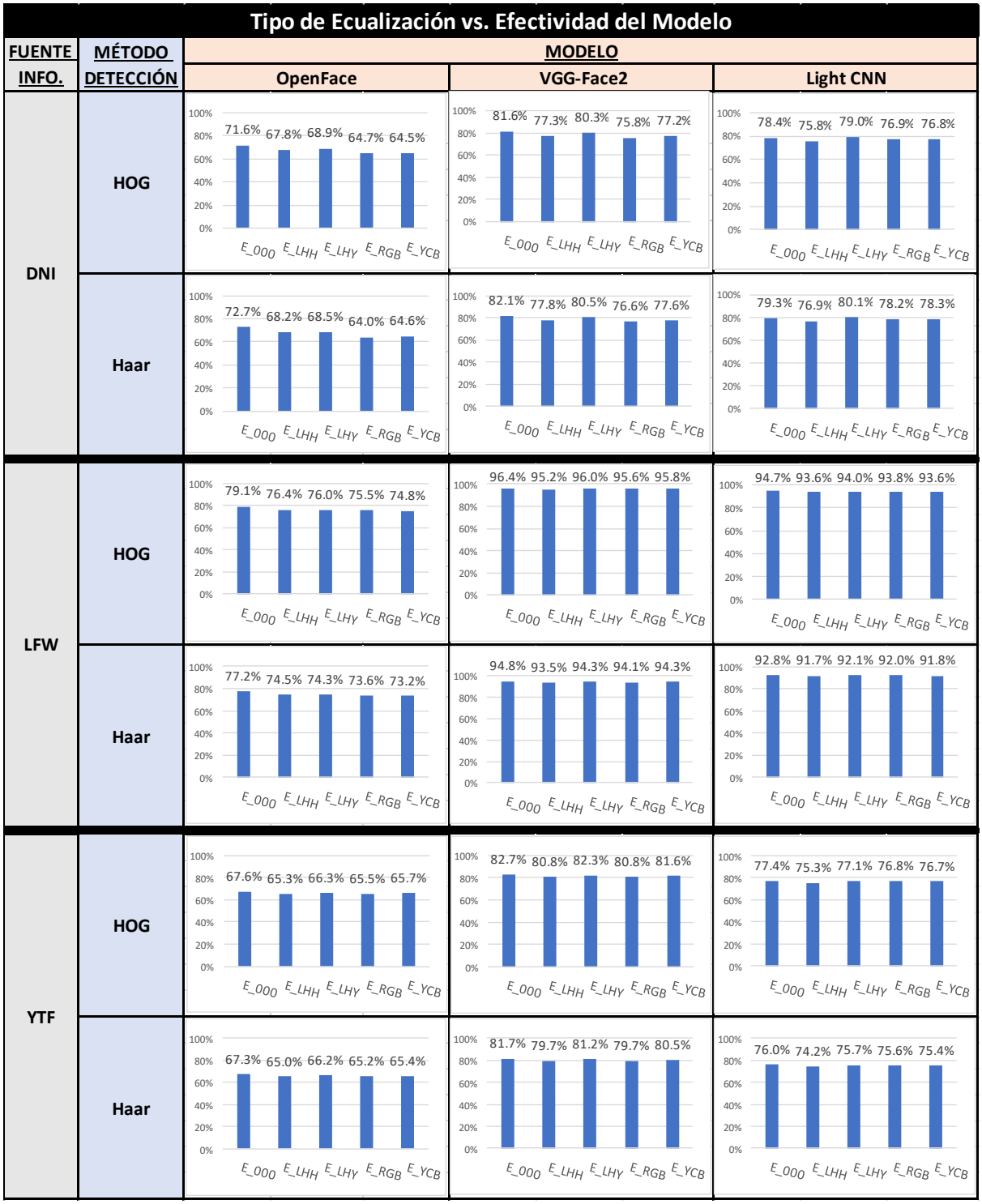


Figura 142. Resultados de Ecualización vs. Efectividad del Modelo
Fuente: Elaboración Propia

Como se puede observar, en el caso del modelo de *OpenFace*, la ecualización tiende a generar una menor efectividad en promedio que no aplicarla.

En el caso de *VGGFace2* y *Light CNN* de igual manera se obtuvo una menor efectividad promedio con cualquiera de los tipos de ecualización.

Adicionalmente, en caso de aplicar alguno de los métodos de ecualización, los que generan mejores resultados en promedio generalmente son la ecualización YCbCr local (*E_LHY*) o la ecualización HSV local (*E_LHH*). El hecho de brindar una ecualización local suele generar menos distorsión en la imagen al hacer más uniforme los cambios en los colores. Por lo tanto, se podría intuir que, de igual manera que con la alineación, una menor distorsión, en este caso en los colores, suele brindar mejores resultados hacia la efectividad.

Por otro lado, se evaluó la relación entre la ecualización como método de preprocesamiento y la efectividad mediante un análisis bivariado. Se plantea las siguientes hipótesis para la prueba:

H_0: Todos los tipos de ecualización tienen media de efectividad iguales.

H_1: Todos los tipos de ecualización no tienen media de efectividad iguales.

Utilizando el análisis de comparación de medias con nivel de confianza de 0.95 (alfa de 0.05) se obtiene lo siguiente:

ANOVA						
	Suma de cuadrados	*gl*	*Media cuadrática*	*F*	*valor P*	*F crit*
Inter-grupos	4420.1561	4	1105.039029	8.693338	5.52488E-07	2.374129
Infra-grupos	514173.39	4045	127.1133215			
Total	518593.54	4049				

Figura 143. Comparación entre medias de efectividad de los tipos de ecualización – Análisis ANOVA
Fuente: Elaboración Propia

- El valor *p-value* es de casi 0 y es menor a alfa (0.05)
- Por lo tanto, se rechaza H0 y se determina que los promedios de los grupos no son iguales.

A partir de esta prueba, se demuestra estadísticamente que la ecualización está relacionada con la efectividad del modelo.

4.3.1.5. Método de Preprocesamiento y la Efectividad del Modelo – HG

A partir de los análisis anteriores se obtuvo lo siguiente:

- El tipo alineamiento está relacionado con la efectividad del modelo
- El tipo de suavizamiento no está relacionado con la efectividad del modelo.
- El tipo de agudizamiento está relacionado con la efectividad del modelo.
- El tipo de ecualización está relacionado con la efectividad del modelo.

Por lo tanto, casi todos los métodos de preprocesamiento evaluados tienen relación estadística con la efectividad del modelo.

Adicionalmente, a continuación, se muestra el incremento porcentual en efectividad luego de aplicar las técnicas de preprocesamiento. La Figura 144 muestra el incremento en efectividad al comparar la mejor combinación con preprocesamiento versus la mejor combinación sin preprocesamiento para cada fuente de información y modelo de verificación:

Diferencia entre efectividad con Preprocesamiento vs. Sin Preprocesamiento			
FUENTE INFO.	MODELO DE VERIFICACIÓN FACIAL		
	OpenFace	VGG-Face2	LightCNN
DNI	**20.91%** Con Preprocesamiento 87.93% Sin Preprocesamiento 67.02%	**1.00%** Con Preprocesamiento 91.19% Sin Preprocesamiento 90.19%	**5.58%** Con Preprocesamiento 93.62% Sin Preprocesamiento 88.04%
LFW	**17.07%** Con Preprocesamiento 92.97% Sin Preprocesamiento 75.90%	**0.30%** Con Preprocesamiento 97.63% Sin Preprocesamiento 97.33%	**0.33%** Con Preprocesamiento 98.18% Sin Preprocesamiento 97.85%
YTF	**15.44%** Con Preprocesamiento 80.36% Sin Preprocesamiento 64.92%	**0.58%** Con Preprocesamiento 85.72% Sin Preprocesamiento 85.14%	**1.56%** Con Preprocesamiento 85.04% Sin Preprocesamiento 83.48%

Figura 144. Diferencia entre efectividad con preprocesamiento y sin preprocesamiento
Fuente: Elaboración Propia

Se puede observar que se obtiene un incremento en efectividad en todos los casos. Asimismo, este incremento varía dependiendo de la fuente de información y modelo de verificación facial. La mayor mejora de efectividad se da en el modelo de *OpenFace*

alcanzando mejoras de más del 20% y dentro de las fuentes de información, la más beneficiada es la base de datos de DNIs.

En base a lo explicado anteriormente, se puede determinar que el método de preprocesamiento está relacionado con la efectividad del modelo.

4.3.1.6. La Fuente de información y la Efectividad del Modelo – HE1

Como se observó en los anteriores resultados, existe diferencias entre la efectividad entre las fuentes de información. *YouTube Faces DB* suele ser la base de datos más complicada de generar una alta efectividad y LFW suele ser la base de datos con mayor efectividad. Abajo se muestra un resumen promedio de efectividad obtenida por cada fuente de información en cada modelo de detección facial y modelo de verificación facial:

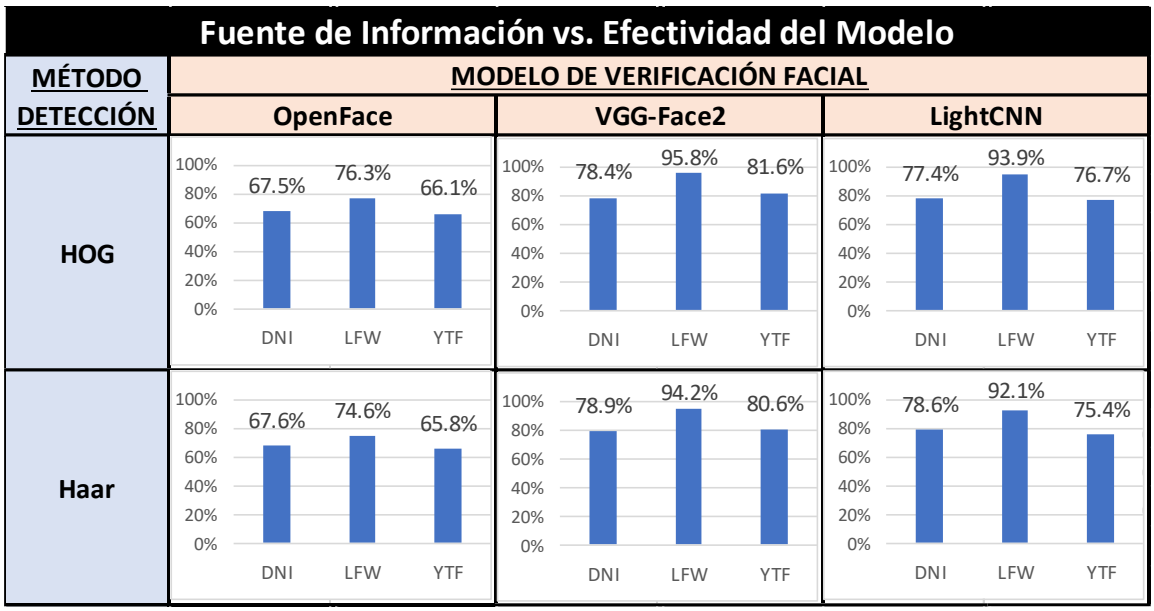


Figura 145. Resultados de Fuente de Información vs. Efectividad del Modelo
Fuente: Elaboración Propia

En los gráficos se muestra la efectividad promedio de todas las pruebas por fuente de información, disgregado por modelo de verificación facial y modelo de detección facial.

Como se puede observar, en todos los casos la fuente de información LFW obtuvo un promedio superior de efectividad mientras que la fuente de información YTF suele obtener una menor efectividad promedio en las pruebas la mayoría de las veces. Igualmente, como se observó al inicio del análisis de resultados (Figura 135), la mejor efectividad obtenida por la mejor combinación de cada prueba fue en LFW, seguida de la efectividad en la fuente de información de DNIs y finalmente en YTF.

Por lo tanto, se puede inferir que existe una mayor dificultad en realizar la verificación facial en la fuente de información YTF, posiblemente debido a la variedad de poses de los rostros y poca estandarización de las imágenes con alta variabilidad en colores, posiciones, iluminación, lejanía del rostro, entre otros.

Asimismo, de igual manera que con el resto de las pruebas, se realizó una prueba bivariada para analizar la relación entre la fuente de información y la efectividad del modelo.

H_0: Todas las fuentes de información tienen media de efectividad iguales.

H_1: Todas las fuentes de información no tienen media de efectividad iguales.

Utilizando el análisis de comparación de medias con nivel de confianza de 0.95 (alfa de 0.05) se obtiene lo siguiente:

ANOVA

	Suma de cuadrados	*gl*	*Media cuadrática*	*F*	*valor P*	*F crit*
Inter-grupos	158583.47	2	79291.73408	891.3463	0	2.997951
Infra-grupos	360010.07	4047	88.95727046			
Total	518593.54	4049				

Figura 146. Comparación entre medias de efectividad de las fuentes de información – Análisis ANOVA
Fuente: Elaboración Propia

- El valor *p-value* es de casi 0 y es menor a alfa (0.05)
- Por lo tanto, se rechaza H0 y se determina que los promedios de los grupos de fuentes de información no son iguales.

A partir de esta prueba, se demuestra estadísticamente que la fuente de información está relacionada con la efectividad del modelo.

4.3.1.7. El Método de Detección Facial y la Efectividad del Modelo – H3

En los resultados, se pudo observar ligeras diferencias en la efectividad entre los métodos de detección facial. El método de detección facial basado en descriptores HOG ha tenido en su mayor parte una mejor efectividad. A continuación, se muestra un resumen promedio de efectividad por método de detección facial disgregado por modelo de verificación facial y fuente de información.

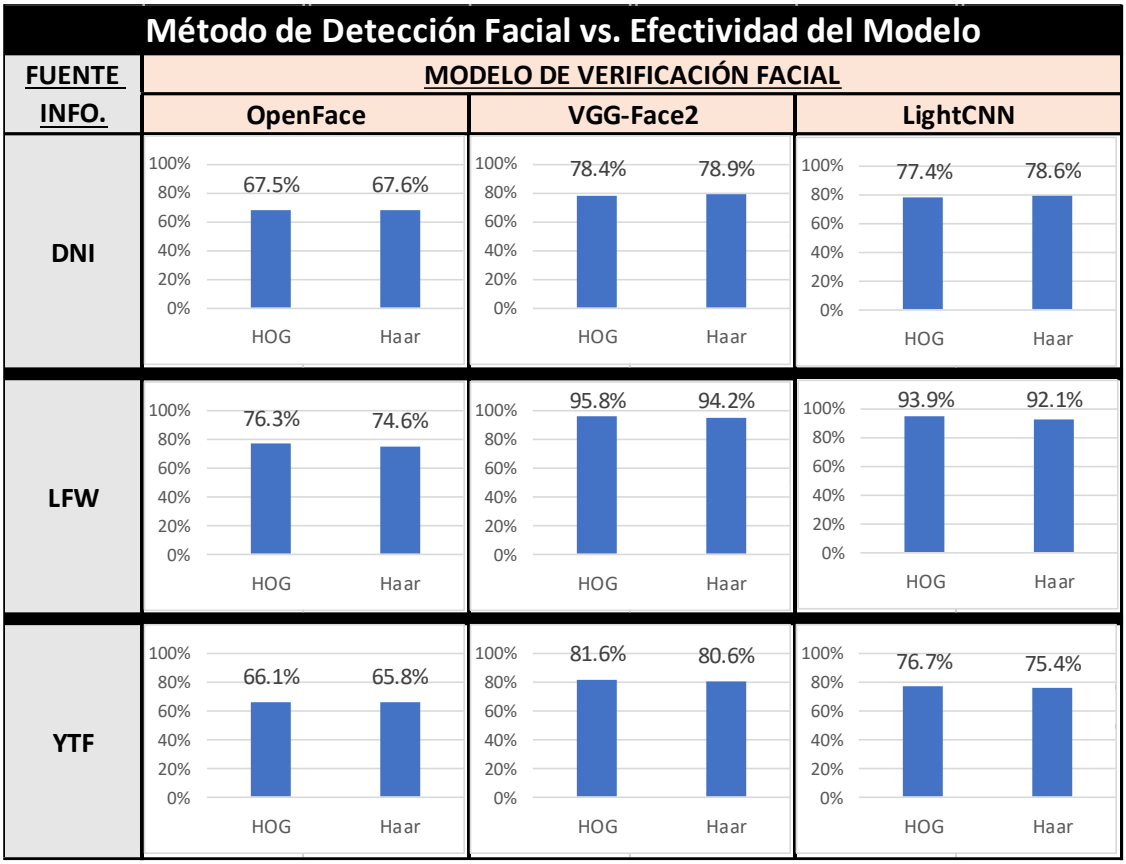


Figura 147. Resultados de Método de Detección Facial vs. Efectividad del Modelo
Fuente: Elaboración Propia

Se puede observar que HOG como extractor de características tiende a tener una mayor efectividad promedio en la mayoría de los casos pero de forma ligera. Esto puede deberse a su capacidad de detección, es decir en que detecte o no correctamente el rostro. Otro motivo de esta mayor efectividad puede ser el tamaño de recorte que se utiliza al momento de detectar el rostro.

Por otro lado, de la misma manera que con el resto de las pruebas, se realizó una prueba bivariada para analizar la relación entre el método de detección y la efectividad del modelo.

H_0: Todos los métodos de detección tienen media de efectividad iguales.

H_1: Todos los métodos de detección no tienen media de efectividad iguales.

Utilizando el análisis de comparación de medias con nivel de confianza de 0.95 (alfa de 0.05) se obtiene lo siguiente:

ANOVA						
	Suma de cuadrados	*gl*	*Media cuadrática*	*F*	*valor P*	*F crit*
Inter-grupos	462.16659	1	462.1665858	3.610764	0.057477713	3.843757
Infra-grupos	518131.38	4048	127.9968812			
Total	518593.54	4049				

Figura 148. Comparación entre medias de efectividad de los métodos de detección facial – Análisis ANOVA
Fuente: Elaboración Propia

- El valor *p-value* es ligeramente mayor a alfa (0.05)
- Por lo tanto, se acepta H0 y se determina que los promedios de los grupos de métodos de detección facial son iguales.

A partir de esta prueba, se determina que estadísticamente el método de detección facial no está relacionado con la efectividad del modelo. Por lo tanto, si bien ambos grupos de datos tienen resultados ligeramente distintos, la diferencia no se considera estadísticamente significativa.

4.3.1.8. El Modelo de Verificación Facial y la Efectividad del Modelo – H4

En el caso del modelo de verificación facial, se hicieron pruebas similares para evaluar su relación con la efectividad del modelo. Como se pudo observar en los resultados anteriores *VGGFace2* y *LFW* han tenido mayor efectividad que el resto. A continuación, se muestra un resumen promedio de efectividad por cada modelo de verificación facial.

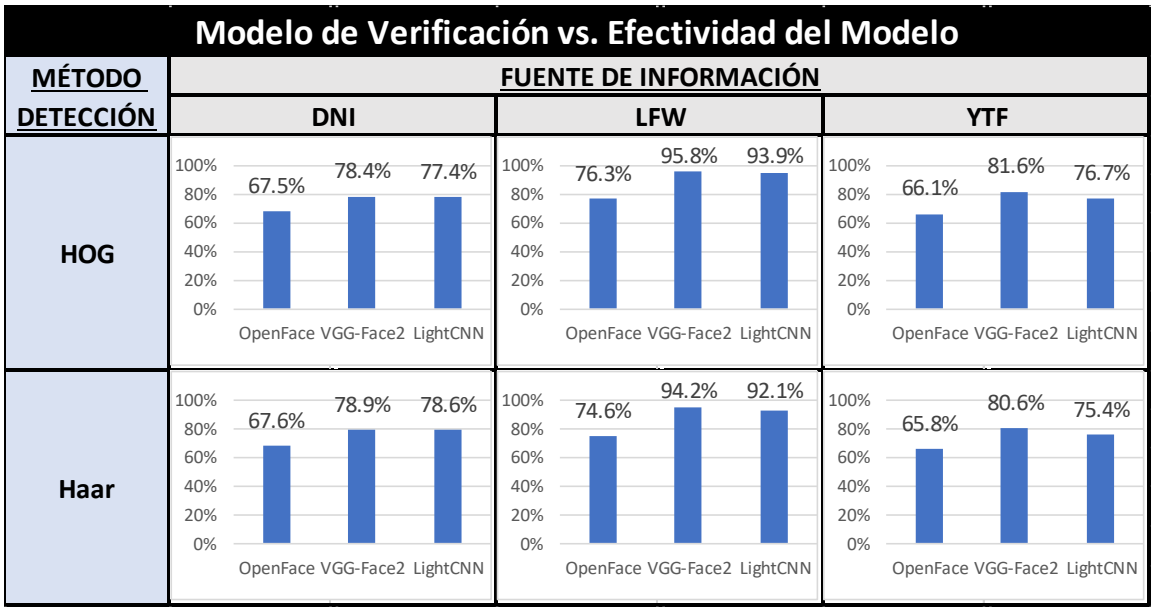


Figura 149. Resultados de Modelo de Verificación Facial vs. Efectividad del Modelo
Fuente: Elaboración Propia

Cuando se obtiene el promedio de todas las pruebas, *VGGFace2* es el más consistente teniendo una efectividad promedio mayor en todos los casos.

Esto contrasta con el hecho que *Light CNN* genera la combinación con la mejor efectividad en el caso de DNI y LFW (Figura 135). Esto se debe a que las combinaciones de *Light CNN* están más dispersas, teniendo así efectividades bien altas pero también más bajas que *VGGFace2* mientras que *VGGFace2* puede tener resultados un poco más compactos generando un promedio más alto pero sin llegar a los mismos máximos que *Light CNN*.

Además, siguiendo el método de análisis que se utilizó en las otras pruebas, se realizó una prueba bivariada para analizar la relación entre el modelo de verificación facial y la efectividad del modelo.

H_0: Todos los modelos de verificación facial tienen media de efectividad iguales.

H_1: Todos los modelos de verificación facial no tienen media de efectividad iguales.

Utilizando el análisis de comparación de medias con nivel de confianza de 0.95 (alfa de 0.05) se obtiene lo siguiente:

ANOVA						
	Suma de cuadrados	*gl*	*Media cuadrática*	*F*	*valor P*	*F crit*
Inter-grupos	180352.31	2	90176.15616	1078.943	0	2.997951
Infra-grupos	338241.23	4047	83.57826276			
Total	518593.54	4049				

Figura 150. Comparación entre medias de efectividad de modelos de verificación facial – Análisis ANOVA
Fuente: Elaboración Propia

- El valor *p-value* es de 0 y es menor a alfa (0.05)
- Por lo tanto, se rechaza H0 y se determina que los promedios de modelos de verificación facial no son iguales.

Por lo tanto, se demuestra que el modelo de verificación facial está relacionado con la efectividad del modelo.

4.3.1.9. Métricas de la Efectividad del Modelo – H5

En esta sección, se analizará las métricas de desempeño para medir la efectividad del modelo y su relación entre ellos. Consecuentemente, el análisis se centrará en la mejor combinación para cada una de las 3 fuentes de información.

En el caso de la fuente de información de DNIs, el modelo con mejor efectividad fue el de *Light CNN* con método de detección Haar, alineamiento de ojos, con suavizamiento 5x5 y sin agudizamiento ni ecualización. Este modelo alcanzó 93.62% de exactitud y una precisión de 23%. Este porcentaje de precisión relativamente bajo se debe a que la data es bien desproporcionada en cantidad de parejas positivas versus parejas negativas. Existen muchas más posibilidades de tener una pareja negativa que positiva y la cantidad de falsos positivos es grande en relación a la cantidad de verdaderos positivos. Sin embargo, para la desproporción de la data, se puede considerar como una precisión adecuada.

A continuación, se muestra la gráfica de la curva *ROC*:

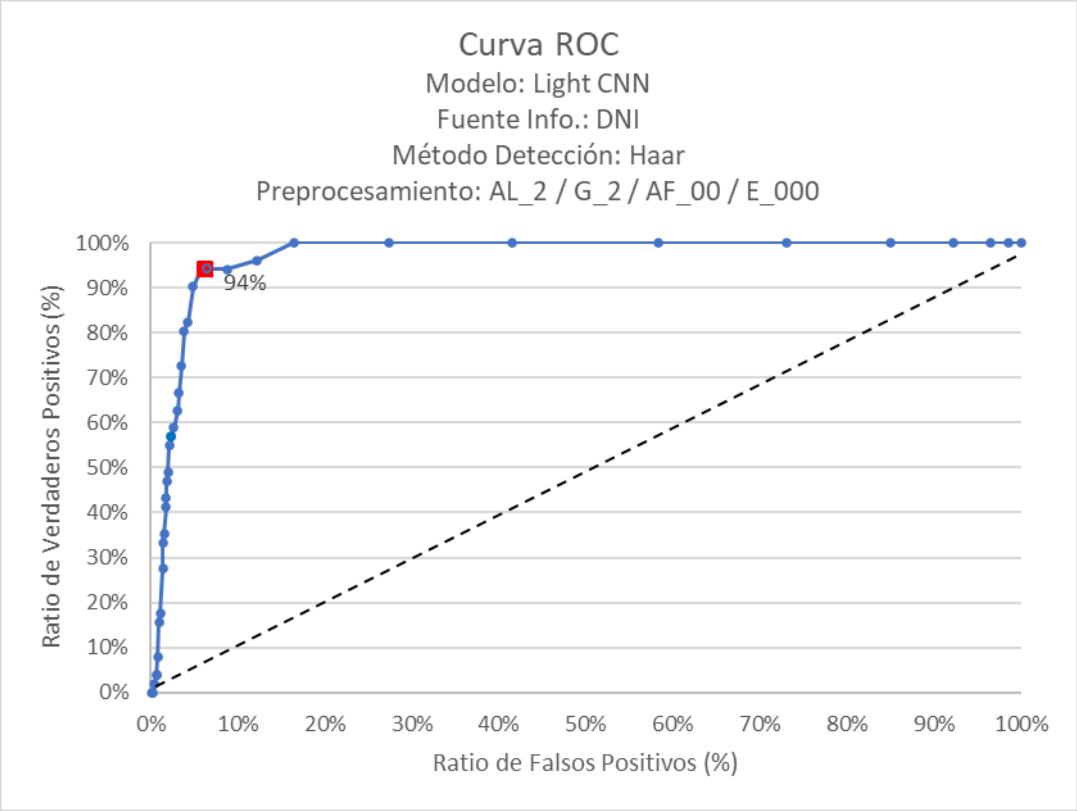


Figura 151. Mejor combinación para verificación facial en Fuente de Información DNI – Curva ROC
Fuente: Elaboración Propia

En la gráfica se puede observar el cambio del ratio de verdaderos positivos y el ratio de falsos positivos ante distintos puntos de corte. Asimismo, la línea punteada ilustra el resultado en caso se hubiesen elegido resultados al azar sin utilizar un modelo estadístico. Por lo tanto, una forma de encontrar el punto de corte óptimo es que este debería encontrarse aproximadamente en la ubicación en la que la curva se acerca lo más posible a la esquina superior izquierda. En este caso, el punto de corte hallado se encuentra bien posicionado, es en

el que la exactitud alcanza el 93.62% y donde el ratio de falsos positivos es igual al ratio de falsos negativos.

Además, se calculó el área bajo la curva (AUC) y se obtuvo un aproximado de 97.01%. Al tener un *AUC* alto, demuestra que el modelo puede obtener óptimos resultados.

Por otro lado, en el caso de la fuente de información de LFW, el modelo con mejor efectividad fue también el de *Light CNN*. En este caso con método de detección HOG, alineamiento de ojos y sin ningún suavizamiento, agudizamiento ni ecualización. Esta combinación alcanzó 98.18% de exactitud y una precisión de 98.18%. La similitud entre ambos indicadores se debe a que la data es totalmente proporcional con la misma cantidad de valores positivos como negativos.

A continuación, se muestra la gráfica de la curva ROC:

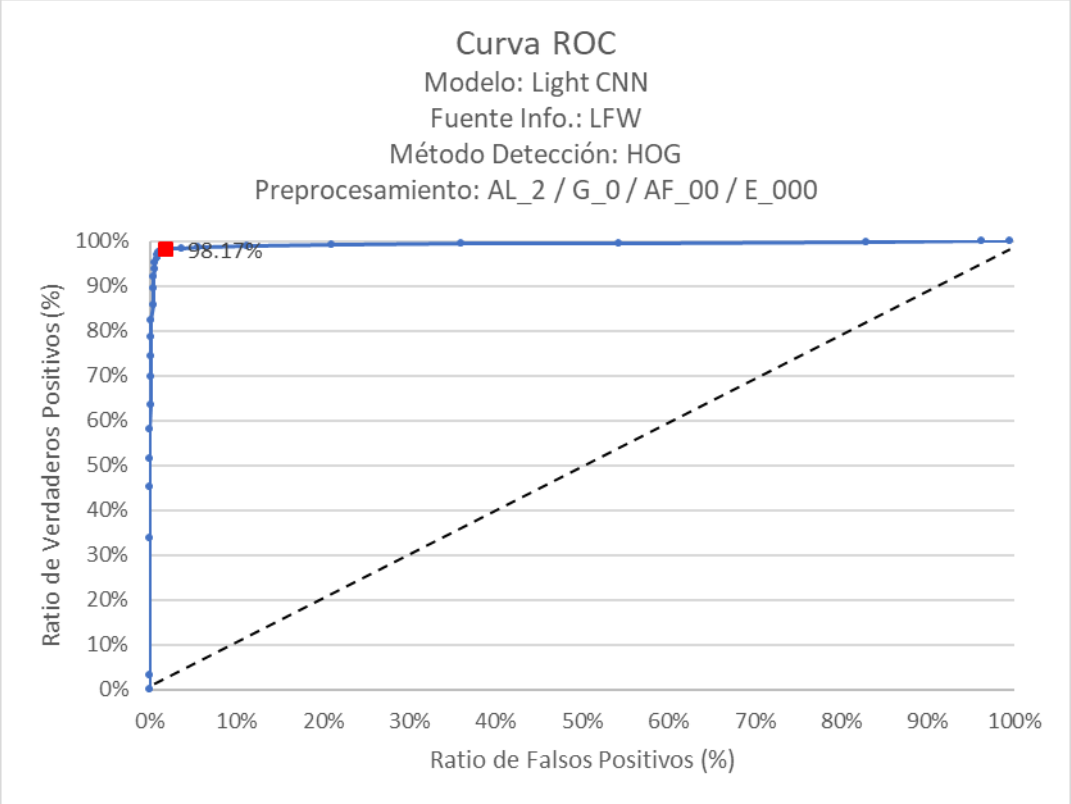


Figura 152. Mejor combinación para verificación facial en Fuente de Información LFW – Curva ROC
Fuente: Elaboración Propia

Asimismo, se calculó el área bajo la curva, obteniendo resultados igualmente positivos con un valor de 99.1%.

Adicionalmente, en el caso de la fuente de información de YTF, el modelo con mejor efectividad fue el de *VGGFace2*, con método de detección HOG, alineamiento de ojos, filtro

de suavizamiento 5x5, agudizamiento al 20% y sin ecualización. Esta combinación alcanzó 85.72% de efectividad y una precisión de 85.72%. De igual manera que con LFW, la similitud entre ambos indicadores se debe a que la data es totalmente proporcional con la misma cantidad de valores positivos como negativos.

A continuación, se muestra la gráfica de la curva ROC:

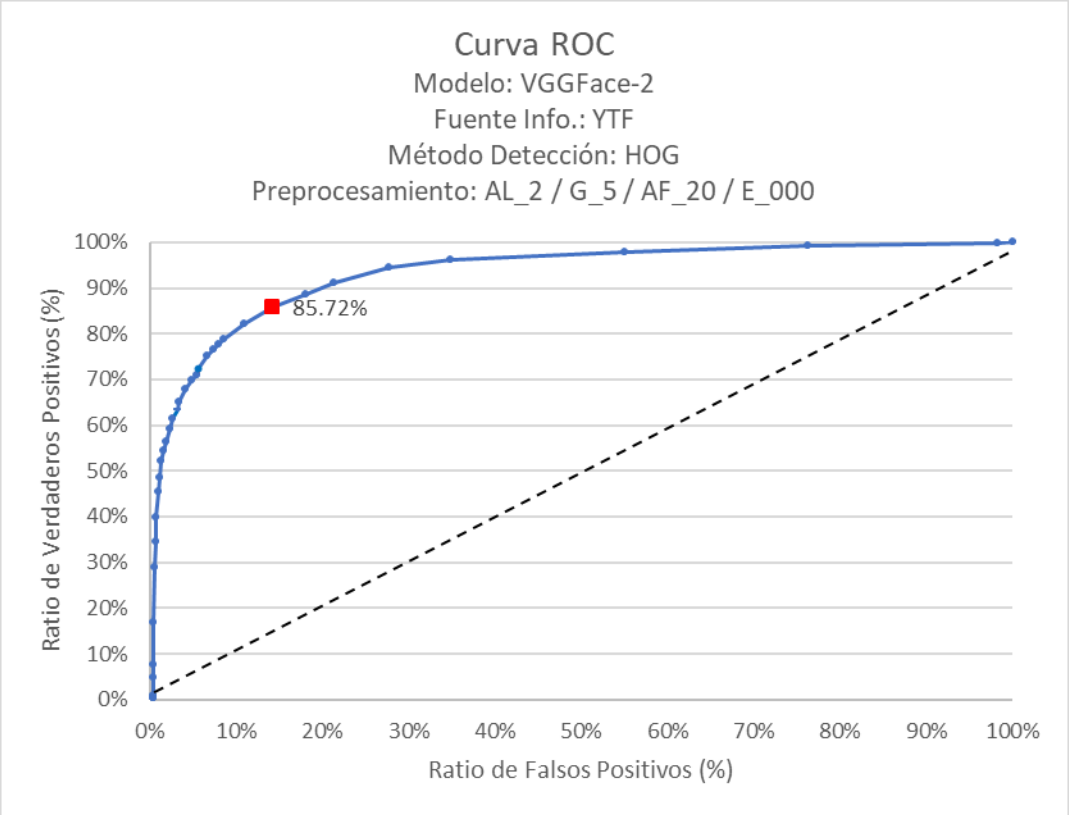


Figura 153. Mejor combinación para verificación facial en Fuente de Información YTF – Curva ROC
Fuente: Elaboración Propia

A partir de la curva, se calculó el valor del AUC, dando como resultado 93.12%.

Como se puede observar, en todos los casos el punto de efectividad encontrado se encuentra cercano a la esquina superior izquierda, dado que son los casos con mayor efectividad. Asimismo, se observa la relación entre el ratio de verdaderos positivos, el ratio de falsos positivos y cómo generan la curva ROC. Además, se observa cómo la proporcionalidad de la data genera similitudes y diferencias entre la precisión y la exactitud. Finalmente, se demostró cómo el área bajo la curva también es un indicador importante de efectividad.

4.3.2. Simulación de la Solución. Aplicación de Software

A continuación, se muestra el funcionamiento de ambos prototipos.

4.3.2.1. Simulación del Prototipo de Recolección de Datos

En el caso del prototipo de recolección de datos, a continuación, se detalla el flujo de su funcionamiento.

El usuario primero ingresa a la URL del prototipo desde su celular. Por ejemplo: https://face-verification.mybluemix.net. Una vez ingresado, el usuario visualizará un formulario como el siguiente:

Figura 154. Simulación de Formulario de Recolección de Datos - Paso 1
Fuente: Elaboración Propia

Se simuló un formulario de registro a una web por lo que se solicitan datos extra adicionales a las fotografías. El usuario llena los datos solicitados de género y edad:

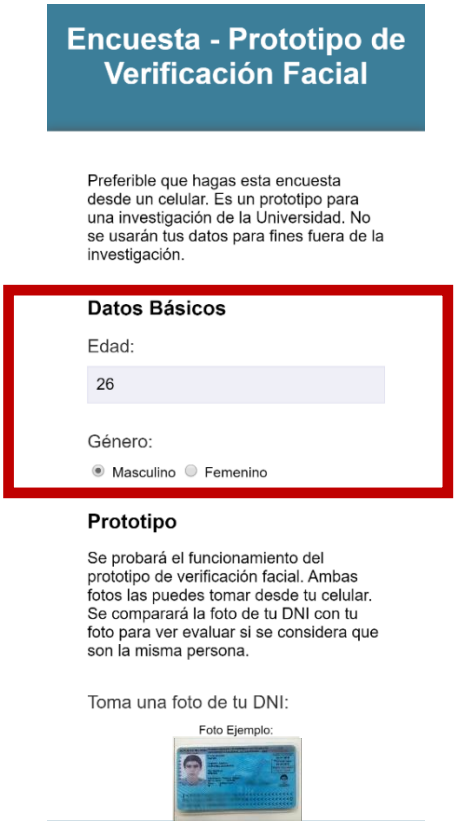


Figura 155. Simulación de Formulario de Recolección de Datos - Paso 2
Fuente: Elaboración Propia

Luego se le solicita al usuario una imagen de su rostro y una imagen de su DNI.

Figura 156. Simulación de Formulario de Recolección de Datos - Paso 3
Fuente: Elaboración Propia

El usuario provee ambas imágenes como se muestra a continuación.

Figura 157. Simulación de Formulario de Recolección de Datos - Paso 4
Fuente: Elaboración Propia

Una vez llenado el formulario, se presiona el botón de "ENVIAR". Esto llevará al usuario a una ventana de resultado mostrando tanto un resultado negativo como positivo de la verificación facial en los rostros. En la imagen a continuación se muestra un ejemplo del mensaje de éxito.

Figura 158. Simulación de Formulario de Recolección de Datos - Paso 5
Fuente: Elaboración Propia

4.3.2.2. Simulación del prototipo de pruebas automatizadas

En el caso del prototipo de pruebas automatizadas, debido a que no se trata de un prototipo para el contacto con el usuario final, la simulación es limitada. Sin embargo, a continuación se mostrarán ejemplos de cómo se da su funcionamiento al interno.

Si se desea entrar a una de las máquinas virtuales que ejecutan las pruebas se puede acceder utilizando el software Putty u otro que permita acceso SSH:

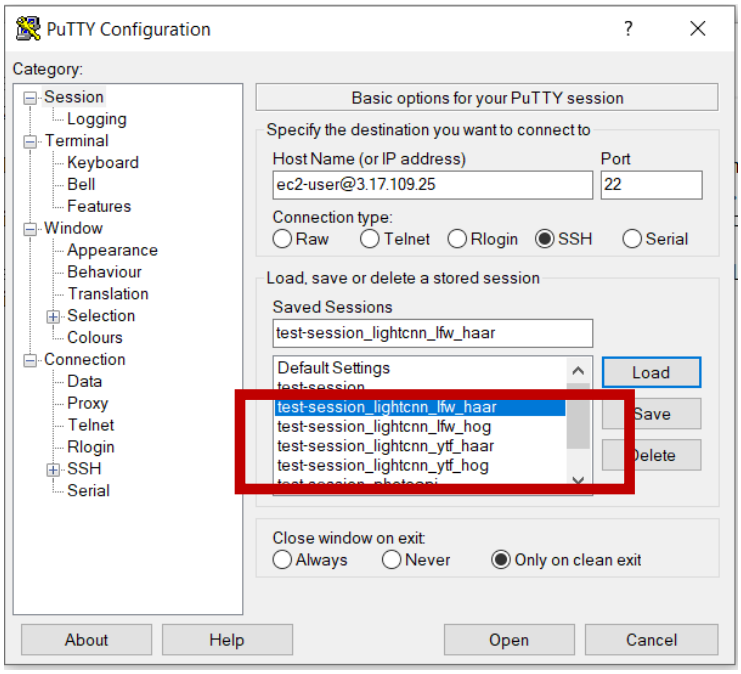


Figura 159. Simulación de Prototipo de Pruebas Automatizadas - Paso 1
Fuente: Elaboración Propia

En el sistema se referenciaron todos los accesos a las máquinas de las pruebas. Estos se muestran resaltados en el recuadro rojo. De esta forma era fácil acceder en cualquier momento a cada máquina dentro de los clústeres y visualizar el proceso de ejecución. Por ejemplo, *test-session_lightcnn_lfw_haar* es el servidor virtual encargado de ejecutar las pruebas de *Light CNN* con método de detección Haar y base de datos LFW.

Al ingresar al servidor virtual, se abre una consola SSH con acceso a la máquina tal como se muestra a continuación:

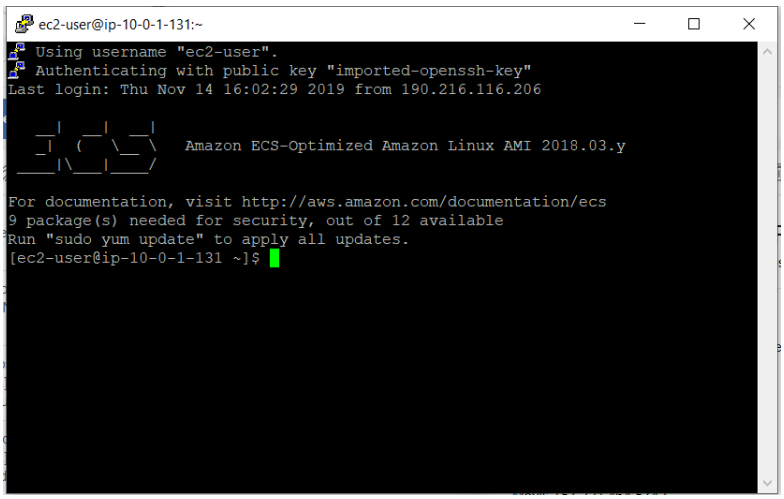


Figura 160. Simulación de Prototipo de Pruebas Automatizadas - Paso 2
Fuente: Elaboración Propia

La máquina virtual tiene acceso al File Server compartido. Esto se puede verificar ejecutando el comando *mount | grep efs*:

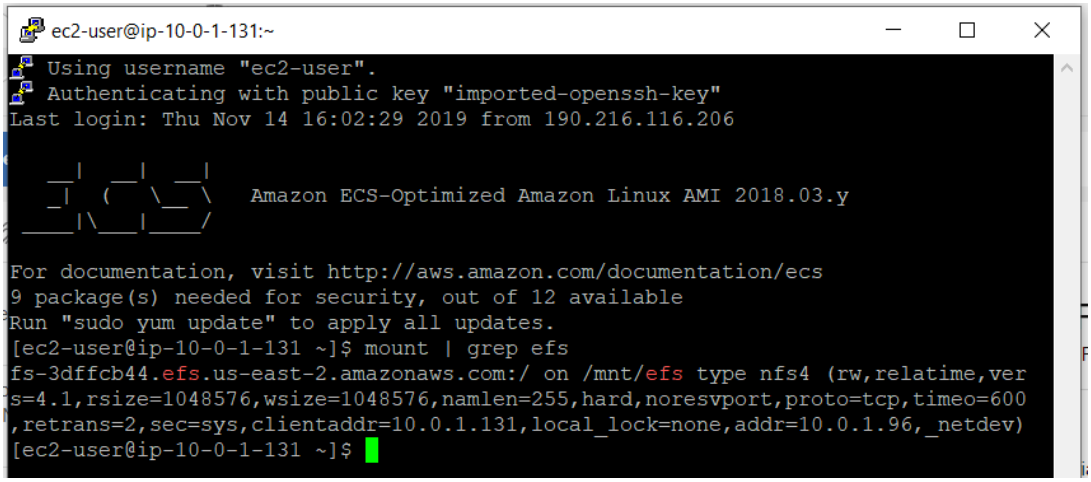


Figura 161. Simulación de Prototipo de Pruebas Automatizadas - Paso 3
Fuente: Elaboración Propia

Asimismo, se puede validar accediendo a la carpeta compartida */mnt/efs/tesis-code/tesis-api* y listando los archivos:

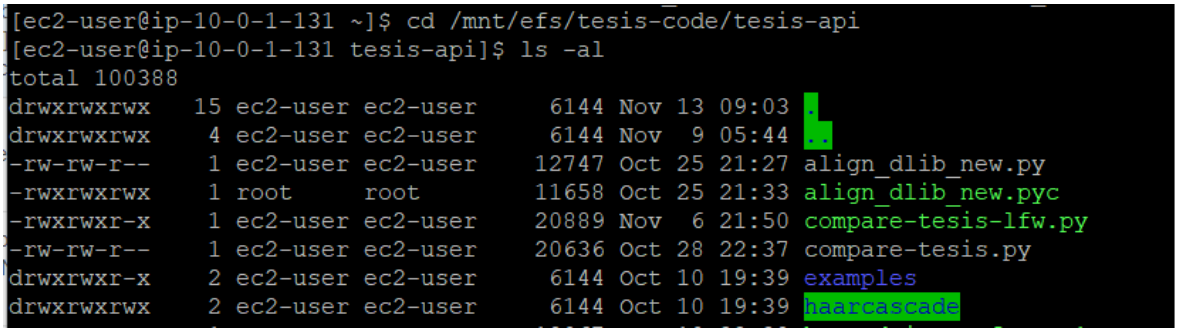


Figura 162. Simulación de Prototipo de Pruebas Automatizadas - Paso 4
Fuente: Elaboración Propia

Dentro de cada servidor se tiene en ejecución el contenedor Docker de uno de los modelos. Esto se verifica listando los procesos activos de Docker (comando *docker ps*):

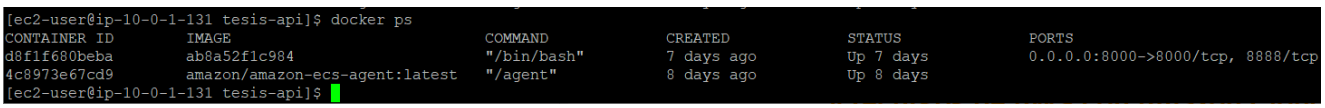


Figura 163. Simulación de Prototipo de Pruebas Automatizadas - Paso 5
Fuente: Elaboración Propia

Adicionalmente, se puede ingresar al contenedor Docker en ejecución con el comando *docker exec -it <id del contenedor> /bin/bash*

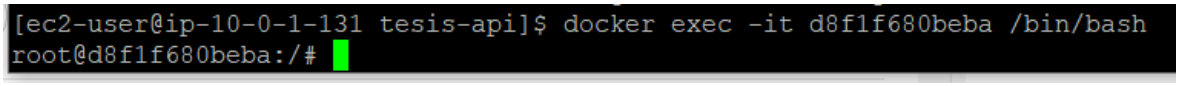


Figura 164. Simulación de Prototipo de Pruebas Automatizadas - Paso 6
Fuente: Elaboración Propia

Una vez dentro del contenedor se valida que se tiene acceso a la carpeta compartida del *File Server* listando los archivos de la carpeta /tesis-api:

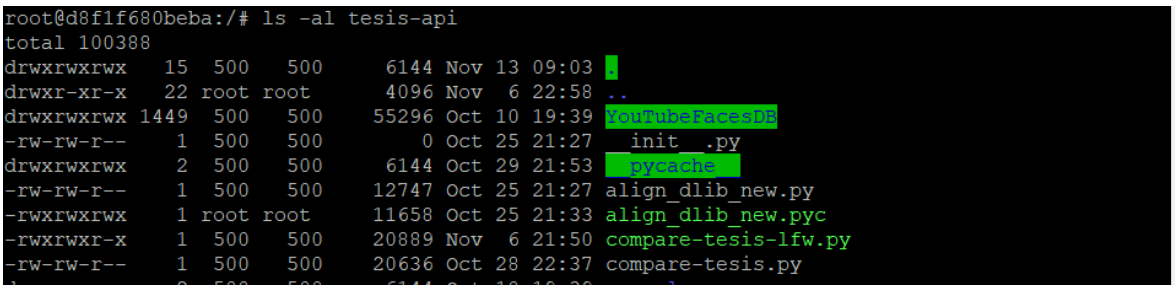


Figura 165. Simulación de Prototipo de Pruebas Automatizadas - Paso 7
Fuente: Elaboración Propia

Por lo tanto, desde el contenedor se puede ejecutar el código de dicha carpeta ejecutando un archivo Python que se desee. Por ejemplo, mediante el comando *python compare-tesis.py*.

Una vez que se tienen las pruebas en ejecución, estas se quedan en ejecución por varias horas o días imprimiendo registros en la consola. Un ejemplo de pruebas en ejecución es el siguiente:

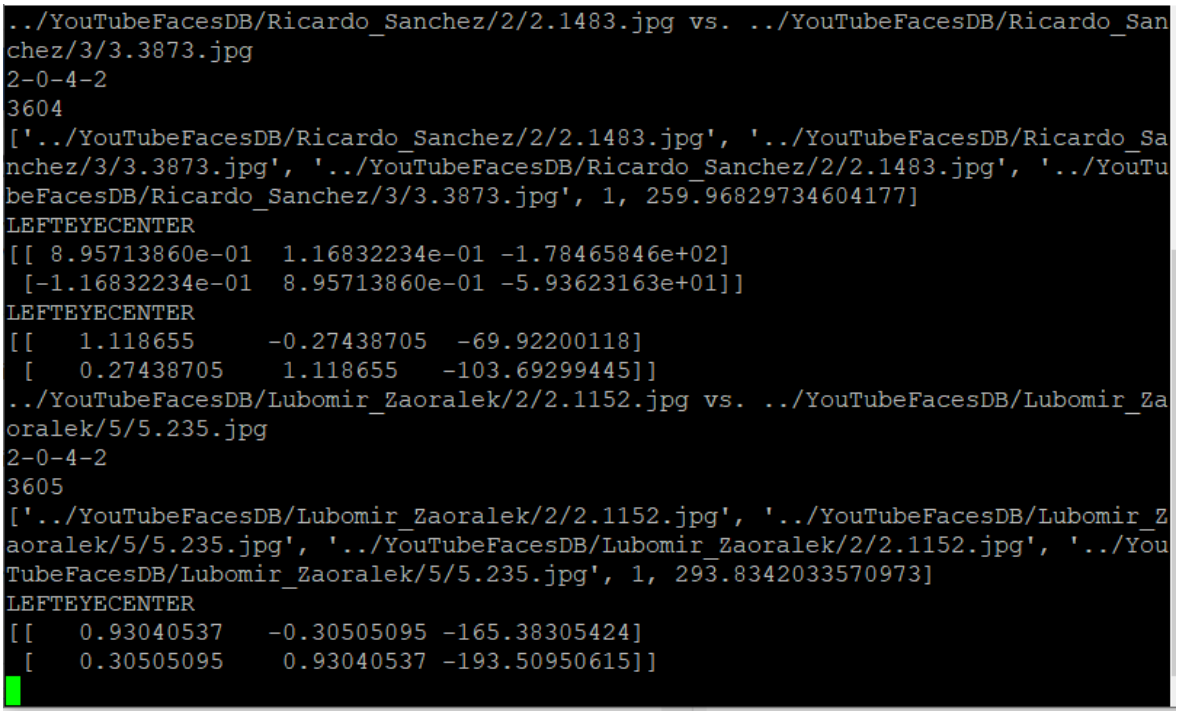


Figura 166. Simulación de Prototipo de Pruebas Automatizadas - Paso 8
Fuente: Elaboración Propia

Una vez culminado un set de pruebas y guardado en un archivo, el código deja de imprimir el proceso en la consola y muestra los resultados de la última prueba:

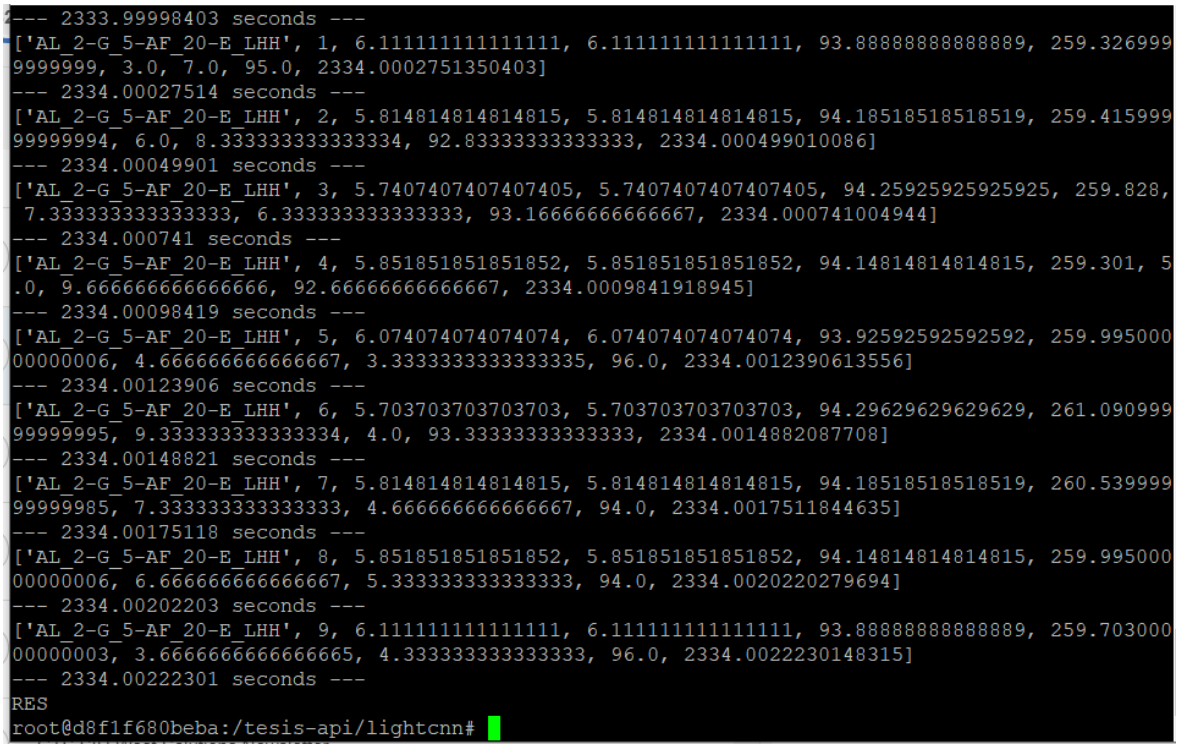


Figura 167. Simulación de Prototipo de Pruebas Automatizadas - Paso 9
Fuente: Elaboración Propia

Finalmente, cada set de pruebas registra los resultados en un archivo. Por ejemplo, en el caso del modelo de verificación *Light CNN* con método de detección HOG y en la base de datos YTF, un extracto del archivo de resultados se observa de la siguiente manera:

1	2	3	4	5	6	7	8	9	10
AL_0-G_0-AF_00-E_000	0	16.35556	16.35556	83.64444	263.375	17.6	18.4	82	3013.937
AL_0-G_0-AF_00-E_000	1	16.57778	16.57778	83.42222	263.248	15.2	17.6	83.6	3013.937
AL_0-G_0-AF_00-E_000	2	16.66667	16.66667	83.33333	263.416	14.8	14.8	85.2	3013.937
AL_0-G_0-AF_00-E_000	3	16.8	16.8	83.2	262.295	10.8	20.8	84.2	3013.938
AL_0-G_0-AF_00-E_000	4	16.4	16.4	83.6	263.893	18.8	14	83.6	3013.938
AL_0-G_0-AF_00-E_000	5	16.53333	16.53333	83.46667	263.609	17.2	15.2	83.8	3013.938
AL_0-G_0-AF_00-E_000	6	16.75556	16.75556	83.24444	263.585	15.2	13.6	85.6	3013.938
AL_0-G_0-AF_00-E_000	7	16.84444	16.84444	83.15556	263.615	14.4	12	86.8	3013.939
AL_0-G_0-AF_00-E_000	8	15.86667	15.86667	84.13333	263.7	23.2	20.4	78.2	3013.939
AL_0-G_0-AF_00-E_000	9	16.31111	16.31111	83.68889	263.436	18.4	18	81.8	3013.939
AL_0-G_0-AF_00-E_YCB	0	17.77778	17.77778	82.22222	270.627	19.2	24.4	78.2	6823.189
AL_0-G_0-AF_00-E_YCB	1	17.95556	17.95556	82.04444	270.938	18.4	21.6	80	6823.189
AL_0-G_0-AF_00-E_YCB	2	18.17778	18.17778	81.82222	271.483	17.6	16	83.2	6823.19
AL_0-G_0-AF_00-E_YCB	3	18.4	18.4	81.6	270.549	12.8	20.4	83.4	6823.19
AL_0-G_0-AF_00-E_YCB	4	18.22222	18.22222	81.77778	271.99	19.6	11.2	84.6	6823.19
AL_0-G_0-AF_00-E_YCB	5	18.13333	18.13333	81.86667	271.16	17.2	17.6	82.6	6823.19
AL_0-G_0-AF_00-E_YCB	6	18.4	18.4	81.6	270.982	14.4	16.8	84.4	6823.191
AL_0-G_0-AF_00-E_YCB	7	18.35556	18.35556	81.64444	271.735	17.2	13.2	84.8	6823.191
AL_0-G_0-AF_00-E_YCB	8	17.51111	17.51111	82.48889	271.799	25.2	20.4	77.2	6823.191
AL_0-G_0-AF_00-E_YCB	9	17.95556	17.95556	82.04444	271.254	19.6	18.8	80.8	6823.191
AL_0-G_0-AF_00-E_RGB	0	17.28889	17.28889	82.71111	271.4	19.6	24	78.2	2821.194
AL_0-G_0-AF_00-E_RGB	1	17.64444	17.64444	82.35556	271.51	16.8	19.6	81.8	2821.194
AL_0-G_0-AF_00-E_RGB	2	17.77778	17.77778	82.22222	271.664	16.8	17.2	83	2821.194
AL_0-G_0-AF_00-E_RGB	3	17.86667	17.86667	82.13333	271.015	13.2	19.2	83.8	2821.194
AL_0-G_0-AF_00-E_RGB	4	17.77778	17.77778	82.22222	272.401	18.4	10.8	85.4	2821.195
AL_0-G_0-AF_00-E_RGB	5	17.73333	17.73333	82.26667	271.596	17.2	18.4	82.2	2821.195
AL_0-G_0-AF_00-E_RGB	6	17.95556	17.95556	82.04444	271.595	14.8	16.4	84.4	2821.195

Figura 168. Simulación de Prototipo de Pruebas Automatizadas - Paso 10
Fuente: Elaboración Propia

A continuación, la descripción de cada columna:

1. Prueba: Representa la prueba realizada mediante la codificación de cada método de alineación, suavizamiento, agudizamiento y ecualización.
2. Iteración: Representa la iteración de la prueba. Las pruebas en las fuentes de información LFW y YTF se realizaron mediante *cross-validation* con 10 iteraciones, probando en 10 conjuntos diferentes.
3. Ratio de Falsos Positivos: Representa el ratio de falsos positivos en el punto de corte encontrado en los 9 primeros conjuntos.
4. Ratio de Falsos Negativos: Representa el ratio de falsos negativos en el punto de corte encontrado en los 9 primeros conjuntos.
5. Efectividad: Representa la exactitud (*accuracy*) en el punto de corte encontrado en los 9 primeros conjuntos.

6. Punto de Corte: Representa el punto de corte donde el ratio de falsos positivos es igual al ratio de falsos negativos en los 9 primeros conjuntos.
7. Ratio de Falsos Positivos de Prueba: Es el ratio de falsos positivos en el conjunto de pruebas restante utilizando el punto de corte encontrado.
8. Ratio de Falsos Negativos de Prueba: Es el ratio de falsos negativos en el conjunto restante de pruebas utilizando el punto de corte encontrado.
9. Efectividad de Prueba: Es la exactitud (*accuracy*) en el conjunto restante de pruebas utilizando el punto de corte encontrado.
10. Tiempo: Es el tiempo en segundos que tardó la ejecución de la prueba.

CAPÍTULO V: CONCLUSIONES Y RECOMENDACIONES

5.1. Discusión y Conclusiones

La investigación tuvo como objetivo principal demostrar la relación entre los métodos de preprocesamiento y la efectividad de la verificación facial empleando visión computacional. Consecuentemente, buscó mejorar la efectividad y reducir los efectos generados por los problemas denominados PIE referentes a la posición del rostro, la iluminación de la imagen y la expresión del rostro. Esto se realizó mediante la construcción de dos prototipos, uno que permitió la recolección de una base de datos y otro que realizó la gran cantidad de pruebas necesarias en búsqueda de las mejores combinaciones de método de preprocesamiento, fuente de información, método de detección facial y modelo de verificación facial. Mediante la evaluación de estas pruebas, la investigación generó resultados que sirven para futuras implementaciones de este tipo de sistemas. Como se observó en la sección *4.3 - Medición de la Solución* los resultados de las pruebas alcanzaron una exactitud de hasta 98.18% en LFW, 85.72% en *YouTube Faces DB* y 93.62% en la base de datos de DNIs.

En el caso de LFW, se llegó a la máxima efectividad mediante el modelo de *Light CNN*. Este tuvo una exactitud de 98.18%, precisión de 98.18% y área bajo la curva de 99.1%. En este caso la precisión es la misma debido a que la data es proporcional entre parejas de rostros positivas y parejas de rostros negativas. La combinación incluyó detección basado en descriptores HOG y métodos de preprocesamiento con alineamiento de ojos, sin suavizamiento, sin afinamiento y sin ecualización.

En el caso de la base de datos de DNIs, se logró la máxima efectividad mediante el modelo *Light CNN*, utilizando los métodos de preprocesamiento de alineamiento de ojos, con suavizamiento 5x5, sin afinamiento y sin ecualización. Se obtuvo una exactitud de hasta

93.62%, área bajo la curva de 97.01% y área precisión de 23%. La precisión en este caso fue baja debido a la gran desproporción que existe entre cantidad de parejas positivas y cantidad de parejas negativas.

Por otro lado, en el caso de la base de datos YTF, se logró una exactitud de 85.72%, una precisión de 85.72% y un área bajo la curva de 93.12%. Para lograr este resultado se utilizó detector basado en características HOG, el alineamiento de ojos, el suavizamiento con filtro 5x5, el afinamiento al 20% y sin ecualización.

Dentro de las fuentes de información, la que obtuvo los mejores resultados fue LFW llegando a 92.97% (*OpenFace*), 97.63% (*VGGFace2*) y 98.18% (*Light CNN*). Luego DNI con 87.93% (*OpenFace*), 91.19% (*VGGFace2*) y 93.62% (*Light CNN*). Finalmente, en último lugar estuvieron los resultados de YTF, este obtuvo 80.36% (*OpenFace*), 85.72% (*VGGFace2*) y 85.04% (*LightCNN*). Por lo tanto, LFW ha tenido mayores resultados posiblemente por la data que tiene rostros menos distorsionados y con menor rotación.

Adicionalmente, en el estudio se demostró la relación entre los métodos de preprocesamiento y su impacto en la efectividad del sistema. Además, se comprobó el impacto de la fuente de información y el modelo de verificación facial. Asimismo, estadísticamente el impacto del método de detección facial no fue muy significativo pero sí generó variaciones en los resultados obtenidos. Ordenando en base al mayor impacto utilizando el valor de p de la tabla ANOVA, se tiene lo siguiente:

Tabla 9.

Impacto de las variables ordenado de mayor a menor

#	Variable	ANOVA - Valor p
1	Modelo de Verificación Facial	0
2	Fuente de Información	0
3	Método de Preprocesamiento	Agudizamiento – 0 Ecualización – 0 Alineamiento – 0.0003976 Suavizamiento – 0.3741
4	Método de Detección Facial	0.0575

En el caso del método de preprocesamiento se colocó al medio de la lista dado que hay métodos con alto impacto (agudizamiento y ecualización) como con muy bajo impacto (suavizamiento). Esto da una idea referencial de qué variables tienen un mayor impacto en los resultados. El mayor impacto detectado en la investigación fue por el lado del modelo de verificación facial y la fuente de información. El menor impacto estuvo por el lado del método de detección facial.

En primer lugar, se observa que lo establecido en la hipótesis general (HG) respecto a la relación entre los métodos de preprocesamiento y la efectividad de la verificación facial queda comprobado estadísticamente. Esto se debe a que se evaluó y validó la relación entre la efectividad del modelo y el alineamiento, el suavizamiento, el agudizamiento y la ecualización. Con esto, se cumple el objetivo principal de evaluar dicha relación y su impacto.

En segundo lugar, se buscó obtener y generar fuentes de información que permitan realizar el análisis de los modelos de verificación facial. Consecuentemente, se tomó en consideración bases de datos reconocidas, así como del contexto local. Las bases de datos reconocidas fueron obtenidas de repositorios públicos (LFW y YTF) y la base de datos del contexto local fue generada para la investigación (DNIs) en base a un prototipo. Además, se analizó la relación entre la fuente de información y la efectividad del modelo. Esto también dio un resultado positivo validando la hipótesis específica 1 (HE1). La efectividad entre las tres fuentes de información es diferente, esto demuestra que ciertas fuentes de información tienden a tener una mayor complejidad que otras sin importar el método de preprocesamiento aplicado. Además, esto comprueba que es importante haber realizado las pruebas en fuentes de información diferentes. La base de datos de *YouTube Faces DB* suele tener una menor efectividad, sus imágenes normalmente son más difíciles de reconocer al tener rostros con mayor variedad de poses y ambientes mientras que la base de datos LFW resultó ser la más fácil en identificación. La base de datos de DNI tuvo una complejidad media dado que la pose del rostro generalmente estaba centrada y sin expresiones muy diferentes, pero las fotografías de los DNIs eran de poca calidad, ruidosas, con una escala de color azulada y la foto a un costado en tamaño pequeño. Por lo tanto, la fuente de información tiene un impacto en el análisis de la relación entre los métodos de preprocesamiento y la efectividad de la verificación facial.

Adicionalmente, se buscó analizar los métodos preprocesamiento de forma independiente y su impacto en la efectividad para probar la hipótesis específica 2 (HE2). En este caso se decidió evaluar los métodos de preprocesamiento de forma individual: el alineamiento, el suavizamiento, el agudizamiento y la ecualización. Se descubrió que, si bien el preprocesamiento impacta en los resultados, el mayor impacto lo tiene la ecualización y el

agudizamiento con valores p mucho menores a *alpha*. Sin embargo, este impacto no será necesariamente siempre positivo. Por ejemplo, esto se puede observar en los valores máximos de efectividad donde el alineamiento ha sido el que consistentemente ha dado mejores resultados, sin tener un impacto tan grande en los resultados como la ecualización y el agudizamiento. Por lo tanto, no necesariamente tener un gran impacto quiere decir el impacto sea positivo. Esto va a depender de diversos factores dentro del problema a analizar. Asimismo, el suavizamiento tuvo un menor impacto en los resultados, posiblemente por su poco nivel de distorsión y transformación en las imágenes. En la Tabla 10 se muestra el impacto de cada tipo de preprocesamiento en orden de mayor a menor.

Tabla 10.

Tipos de preprocesamiento ordenados en base al impacto de mayor a menor

#	Variable	ANOVA - Valor p
1	Agudizamiento	0
2	Ecualización	0
3	Alineamiento	0.000397608
4	Suavizamiento	0.3741

En el caso del alineamiento, se observaron resultados muy positivos del alineamiento de solo ojos (AL_2). Esto se puede deber a que dicho alineamiento genera una menor distorsión al simplemente rotar la imagen. En cambio, el alineamiento ojos nariz (AL_1), al hacer una transformación en base a 3 puntos, suele generar un mayor movimiento o deformación en los rostros. Si bien *OpenFace* sí tuvo resultados positivos con el alineamiento ojos nariz, esto se debe a que dicho modelo fue entrenado tomando en cuenta dicho alineamiento. Por lo tanto, esto refleja otro punto importante, los métodos de preprocesamiento pueden aplicarse previos al entrenamiento, haciendo que el modelo sepa actuar ante los efectos de este y brinde mejores resultados. En este caso, debido a que se está aplicando el preprocesamiento en modelos previamente entrenados, los métodos con mucha distorsión pueden generar resultados adversos dado que el modelo puede no estar preparado para recibir imágenes tan diferentes a lo ya conocido.

Por otro lado, se buscó analizar los métodos de detección facial mediante descriptores HOG y descriptores Haar y cómo afectan la evaluación de efectividad de la verificación facial.

Respecto a la hipótesis específica 3 (HE3) que busca analizar dicha relación entre estos métodos de detección facial y la efectividad del modelo, se concluye que estos no tienen un impacto estadístico significativo. La efectividad entre el método HOG y Haar tiende a variar muy poco, lo cual se comprobó mediante el análisis estadístico que muestra un valor p mayor a *alpha*. Por lo tanto, esto no genera un impacto en el análisis entre el preprocesamiento y la efectividad de la verificación facial. Sin embargo, en los resultados numéricos igual existieron variaciones ligeras. Posiblemente las variaciones sean ligeras dado que los detectores faciales tienen un porcentaje de error bajo en la actualidad. Las diferencias leves entre HOG y Haar puede deberse tanto a detecciones fallidas como al tipo de recorte que se genera en la imagen. En este caso, el extractor de características HOG tuvo los mejores resultados en promedio con menos detecciones fallidas.

Otro objetivo importante fue obtener y configurar modelos de verificación facial que se encuentren disponibles y entrenados para realizar las evaluaciones planteadas. En este caso los modelos modernos y reconocidos entrenados con grandes bases de datos públicas y en redes neuronales convolucionales fueron *OpenFace*, *VGGFace2* y *Light CNN*. Estos fueron configurados para poder realizar las pruebas. Además, respecto a la hipótesis específica 4 (HE4) se concluyó que existe relación entre el modelo de verificación facial y la efectividad del modelo. Por lo tanto, genera un impacto en el análisis entre el preprocesamiento de imágenes y la efectividad de la verificación facial. Hay modelos que tienden a tener una mayor efectividad utilizando cualquier tipo de preprocesamiento. En este caso *Light CNN* y *VGGFace2* generaron los mejores resultados. Estos mejores resultados se deben a que fueron entrenados con una mayor cantidad de datos así como a sus arquitecturas más modernas basadas en bloques residuales. Otro hecho importante es que las combinaciones de resultados de *Light CNN* están más dispersas, teniendo así efectividades bien altas pero también más bajas que *VGGFace2* mientras que *VGGFace2* tuvo resultados un poco más compactos generando un promedio más alto pero sin llegar a los mismos máximos que *Light CNN*.

Asimismo, se buscó establecer las métricas de desempeño que determinen una correcta evaluación de un modelo de verificación facial. En esta investigación se utilizaron algunos de los indicadores más comunes de efectividad como la exactitud y la curva ROC. Igualmente, la hipótesis específica 5 (HE5) analiza cómo las métricas de efectividad son representativos y comparables entre sí. Esta también se corroboró mediante el análisis de las curvas ROC, el área bajo la curva y la precisión de las combinaciones con mejores resultados. Por lo tanto, estas métricas permitieron hacer las comparaciones entre las combinaciones evaluadas en esta investigación de manera imparcial.

Finalmente, para lograr realizar las pruebas, se desarrolló prototipos que permitieron evaluar la relación de los resultados de las hipótesis planteadas y siguieron los procesos estándares de la verificación facial. Se configuró dichos prototipos para que utilicen los modelos de verificación facial, métodos de detección facial y métodos de preprocesamiento de imágenes planteados en la investigación. En este caso el prototipo de recolección de datos sirvió para la recolección de la base de datos del contexto local. Asimismo, el prototipo de pruebas automatizadas permitió realizar las más de 18 millones de verificaciones faciales necesarias para poder realizar las 4,050 combinaciones de pruebas necesarias para las 3 bases de datos diferentes, 3 modelos diferentes, 2 tipos de detección facial, 3 tipos de alineamiento, 3 tipos de suavizamiento, 5 tipos de ecualización y 5 tipos de agudizamiento.

5.2. Recomendaciones

A partir de la presente investigación fue posible determinar que los métodos de preprocesamiento de imágenes tienen influencia en la efectividad de la verificación facial. Además, esta efectividad se ve afectada por la fuente de información, el método de detección facial y el modelo de verificación facial utilizado. Por lo tanto, cada eslabón del proceso está conectado y tiene una influencia directa en cómo este genera la efectividad final del sistema. Por ejemplo, si se hace un preprocesamiento que distorsiona mucho la imagen, todo el proceso siguiente se ve afectado. Asimismo, si la extracción de características no da el resultado correcto, el impacto en el resultado podría ser mayor. Por lo tanto, es importante considerar que todos los elementos en el proceso son significativos y tomar en cuenta cómo estos pueden afectar el resultado al momento de construir un sistema de verificación facial.

Un factor importante al realizar las pruebas entre tantas variables diferentes es buscar la mejor alternativa de estandarizarlas. Por ejemplo, se debe tener en consideración que el punto de corte de cada modelo puede afectar directamente el resultado final de cada configuración del proceso. Para lograr un estándar, se tomó en cuenta que el punto de corte utilizado sería en el cual el ratio de falsos positivos sea igual al ratio de falsos negativos (el *equal error rate*).

Por otro lado, es importante notar que ciertos modelos están mejor entrenados que otros ante ciertas variantes en las imágenes. Por lo tanto, al momento de crear un modelo es necesario tomar la decisión si este tendrá como entrada imágenes alteradas o imágenes del entorno real sin alterar. Normalmente, si se cuenta con una amplia gama de imágenes se decide ir por el camino de no distorsionarlas, pero si se cuenta con una menor cantidad se decide entrenar en base a imágenes alineadas o adaptadas. Sin embargo, si se adaptan las imágenes en la etapa de entrenamiento, estas tendrán que ser adaptadas cada vez que se ejecuta dicho modelo utilizando

el mismo método de preprocesamiento con el que fue entrenado para generar la efectividad deseada. La investigación demostró que ciertos métodos de preprocesamiento, sin necesidad de estar entrenados dentro de un modelo, pueden generar mejores resultados si se aplican previo a la verificación. En especial, el alineamiento de ojos fue un claro ejemplo de esto dando resultados independientes del modelo y fuente de información al que fue aplicado.

En lo referente a la fuente de información, sería interesante recrear estos análisis en otras bases de datos ya sean públicas o privadas. Adicionalmente, se podría realizar estudios que analicen más métodos de preprocesamiento y su impacto de tal forma que se tenga un mayor detalle este tipo de resultados. Por ejemplo, estos métodos pueden ser nuevas formas de alineamiento o aumento de valores como el contraste y/o brillo. En el ámbito del alineamiento se abren nuevas posibilidades de investigación buscando alineamientos que no distorsionen mucho el rostro y que generen rostros más fáciles de reconocer tomando en cuenta los resultados positivos del estudio.

Asimismo, se podrían evaluar otros modelos de verificación facial y cómo estos resultados cambian en el caso de utilizarse modelos que no estén basados en redes neuronales convolucionales. Algunos ejemplos de estos modelos son los basados en *fisherfaces* o *eigenfaces* y algunos más modernos como los basados en redes generativas antagónicas (*GAN - Generative Adversarial Networks*).

Otro factor de interés sería analizar a detalle si el cambio de efectividad del método de detección facial se debe a fallas en detectar el rostro o se debe en mayor medida al tamaño del recorte realizado. Además, se podría evaluar los tamaños de recorte y su impacto, considerándolo como otro método de preprocesamiento en la efectividad de la verificación facial.

Adicionalmente, dentro de la implementación, se debe buscar la mejor solución de arquitectura de cara a generar los resultados de las pruebas, teniendo los objetivos en mente. En este caso, se utilizó una arquitectura enfocada en lograr paralelismo en las pruebas manteniendo una estructura en base a métodos y funciones de código individuales y reutilizables.

Asimismo, se buscó utilizar recursos en el despliegue que permitan una implementación de bajo costo y escalable. Esto se pudo observar en ambos prototipos. En el caso del prototipo de recolección de datos se logró mediante la separación modular del componente de verificación facial y el componente web. En el caso del prototipo de pruebas automatizadas esto se logró mediante el paralelismo de varias instancias de servidores virtuales de bajo costo. Ambos

prototipos tienen total factibilidad de escalar en el futuro o que se utilicen arquitecturas similares en investigaciones futuras.

Por otro lado, se realizó el estudio mediante un proceso basado en metodologías orientadas a analítica de datos y desarrollo de software. Un proceso en el cual se mezclan actividades relacionadas de ambos rubros para poder lograr cumplir con el proceso necesario de inicio a fin tanto por el lado teórico como por el lado práctico de la implementación. Por lo tanto, se logró un balance entre ambas metodologías de tal forma que se desarrollen estos prototipos exitosamente. Esto también puede ser útil para investigaciones con un enfoque similar en el futuro. Existen amplias posibilidades de investigación futura acerca de metodologías conjuntas entre ambos campos: ciencia de datos y desarrollo de software.

Finalmente, el presente estudio es útil como base para futuras implementaciones de este tipo de sistemas, estableciendo procesos, herramientas y arquitecturas. Asimismo, sirve como base para investigaciones que quieran abarcar temas relacionados con los métodos de preprocesamiento y la verificación facial. Además, planteó procesos y actividades basadas en metodologías reconocidas que pueden ser utilizados en investigaciones con fines similares a los de la presente investigación. Asimismo, brindó un análisis completo del proceso y de cómo cada fase se interrelaciona y apoyo al resultado de una efectividad final en un sistema de verificación facial.

REFERENCIAS

ACM. (2018). Obtenido de A.M. Turing Award: https://amturing.acm.org/

Adouani, A., Wiem, M., & Zied, L. (2019). Comparison of Haar-like, HOG and LBP approaches for face detection in video sequences. *International Multi-Conference on Systems, Signals & Devices* (pp. 266-271). Istanbul: IEEE.

Al-Modwahi, A. A., Sebetela, O., Batleng, L. N., Parhizkar, B., & Lashkari, A. H. (2012). Facial Expression Recognition Intelligent Security System for Real Time Surveillance. *World Congress in Computer Science, Computer Engineering, and Applied Computing*, (pp. 1-8). Nevada.

Alom, M. Z., Taha, T. M., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, M. S., . . . Asari, V. K. (2018). The History Began From AlexNet: A Comprehensive Survey on Deep Learning Approaches. *arXiv*, 1-39.

Amazon Web Services. (2020). *Tipos de instancias de Amazon EC2*. Obtenido de Amazon Web Services: https://aws.amazon.com/es/ec2/instance-types/

Amos, B., Bartosz, L., & Satyanaray, M. (2016). OpenFace: A general-purpose face recognition library with mobile applications.

Amos, B., Bartosz, L., & Satyanarayanan, M. (2016). OpenFace. Github. Obtenido de https://github.com/cmusatyalab/openface

Bassil, Y. (2012). A Simulation Model for the Waterfall Software Development Life Cycle. *International Journal of Engineering & Technology, 2*(5).

Ben-Hur, A., & Weston, J. (2010). A User's Guide to Support Vector Machines. *Methods in Molecular Biology, 609*, 223-239.

Benitez, G., Olivares, J., Aguilar, G., & Sanchez, G. (2012). Face Identification Based on Contrast Limited Adaptive Histogram Equalization (CLAHE).

Burges, C. J. (1998). A Tutorial on Support Vector Machines for Pattern Recognition. *Data Mining and Knowledge Discovery, 2*(2), 121-167.

Calvo, G., Baruque, B., & Corchado, E. (2013). Study of the Pre-processing Impact in a Facial Recognition System. *International Conference on Hybrid Artificial Intelligence Systems*, (pp. 334-344).

Cao, Q., Shen, L., Xie, W., Parkhi, O. M., & Zisserman, A. (2018). VGGFace2: A dataset for recognising faces across pose and age. *International Conference on Automatic Face & Gesture Recognition* (págs. 67-74). Xi'an, China: IEEE.

Caraig, L. M. (Diciembre de 2017). *Understanding Image Histograms with OpenCV*. Obtenido de Lou Marvin Caraig: https://lmcaraig.com/image-histograms-histograms-equalization-and-histograms-comparison/

Castaldo, F., & Palmieri, F. A. (2013). *Camera system: pinhole model, calibration and reconstruction.* Campania, Italia.

Chang, P. C., Chen, Y.-S., Lee, C. H., Cheng-Chang, L., & Han, C. C. (2018). Illumination Robust Face Recognition Using Spatial Expansion Local Histogram Equalization and Locally Linear Regression Classification. *International Conference on Computer and Communication Systems*, (pp. 249-253).

Chang, Y., Jung, C., Ke, P., Song, H., & Hwang, J. (2018). Automatic Contrast-Limited Adaptive Histogram Equalization With Dual Gamma Correction. *IEEE Access, 6*, 11782-11792.

Chen, L. (Enero de 2019). *Support Vector Machine-Simply Explained.* Obtenido de Towards Data Science: https://towardsdatascience.com/support-vector-machine-simply-explained-fee28eba5496

Coventry, L., De Angeli, A., & Johnson, G. (2003). Usability and Biometric Verification at the ATM Interface. *Usability of Large Scale Public Systems, 5*(1), 153-160.

Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. *Computer Vision and Pattern Recognition.* California, Estados Unidos: IEEE.

De la Escalera, A. (2001). *Visión por Computador: Fundamentos y métodos.* Madrid: Prentice Hall.

Denyer, S. (7 de Enero de 2018). *China's watchful eye: Beijing bets on facial recognition in a big drive for total surveillance*. Obtenido de The Washington Post: https://www.washingtonpost.com/news/world/wp/2018/01/07/feature/in-china-facial-recognition-is-sharp-end-of-a-drive-for-total-surveillance

Destruels, V. (2015). Información Digital. I.E.S. Albal. Obtenido de http://www.aulainformatica.eu/datos/dise%C3%B1o_grafico/gimp/capitulo2/Teoria2.pdf

Dharavath, K., Talukdar, F. A., & Laskar, R. H. (2014). Improving Face Recognition Rate with Image Preprocessing. *Indian Journal of Science and Technology, 7*(8), 1170-1175.

Felzenszwalb, P. F., Girshick, R. B., McAllester, D., & Ramanan, D. (2009). Object Detection with Discriminatively Trained Part Based Models. *IEEE Transactions on*

Pattern Analysis and Machine Intelligence, 32(9), 1627 - 1645. doi:10.1109/TPAMI.2009.167

Fotiadis, E. P., Garzón, M., & Barrientos, A. (2013). Human Detection from a Mobile Robot Using Fusion of Laser and Vision Information. *Sensors, 13*, 11603-11635.

Gallardo, J. A. (2009). *Metodología para la Definición de Requisitos en Proyectos de Data Mining (ER-DM).* Tesis Doctoral, Universidad Politécnica de Madrid, Madrid.

Gao, H. (2017). *A Walk-through of AlexNet*. Obtenido de Medium: https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637

Gibiansky, A. (Febrero de 2014). *Convolutional Neural Netowrks*. Obtenido de Andrew Gibiansky: http://andrew.gibiansky.com/blog/machine-learning/convolutional-neural-networks/

Gómez, R. (Abril de 2019). *Understanding Ranking Loss, Contrastive Loss, Margin Loss, Triplet Loss, Hinge Loss and all those confusing names*. Obtenido de Raúl Gómez.

Haller, K. (15 de Enero de 2020). *Making CRISP-DM Work for Embedded Analytics*. Obtenido de The Data Administration Newsletter: https://tdan.com/making-crisp-dm-work-for-embedded-analytics/25884

Han, H., Shan, S., Chen, X., & Gao, W. (2013). A Comparative Study on Illumination Preprocessing in Face Recognition. *Pattern Recognition*, 1691-1699.

Haykin, S. (2009). *Neural Networks and Learning Machines.* New Jersey: Pearson Education.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. *arXiv*, 1-12.

Hearst, M. A. (1998). Support Vector Machines. *IEEE Intelligent Systems, 13*(4), 18-28.

Heusch, G., Rodriguez, Y., & Marcel, S. (2006). Local binary patterns as an image preprocessing for face authentication. *7th International Conference on Automatic Face and Gesture Recognition*. Southampton: IEEE.

Hu, J., Shen, L., Albanie, S., Sun, G., & Wu, E. (2018). Squeeze-and-Excitation networks. *IEEE Conference on Computer Vision and Pattern Recognition*. Utah: IEEE.

Huang, G. (2019). *Labeled Faces in the Wild Home*. Obtenido de Labeled Faces in the Wild: http://vis-www.cs.umass.edu/lfw/

Huang, G. B., Ramesh, M., Berg, T., & Learned-Miller, E. (2007). *Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments.*

Intel. (2015). *Intel Developer Zone*. Obtenido de Histogram of Oriented Gradients (HOG) Descriptor: https://software.intel.com/en-us/node/529070

Jack, K. (2005). *Video Demystified: A Handbook for the Digital Engineer* (4th ed.). Oxford, United Kingdom: Newnes.

Jeon, G., & Lee, Y.-S. (2013). Histogram Equalization-Based Color Image Processing in Different Color Model. *Advanced Science and Technology Letters, 28*, 54-57.

KalaiSelvi , R., Kavitha , P., & Shunmuganathan , K. L. (2014). Capturing Facial Actions in Video to Revive Expressions of Humans. *International Journal of Innovative Research in Science, Engineering and Technology*, 2021-2025.

Kandan, H. (30 de Agosto de 2017). *Understanding the kernel trick*. Obtenido de Towards Data Science: https://towardsdatascience.com/understanding-the-kernel-trick-e0bc6112ef78

Karaaba, M., Surinta, O., Schomaker, L., & Wiering, M. (2015). In-plane Rotational Alignment of Faces by Eye and Eye-pair Detection. *Conferencia VISAPP*.

Karpathy, A., Johnson, J., & Fei-Fei, L. (2016). CS231n Convolutional Neural Networks for Visual Recognition. California: Stanford University. Obtenido de http://cs231n.stanford.edu/

Klette, R. (2014). *Concise Computer Vision: An Introduction into Theory and Algorithms*. Londres: Springer.

Krig, S. (2014). Image Pre-Processing. In S. Krig, *Computer Vision Metrics* (pp. 39-83). California, Estados Unidos: Springer.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in neural information processing systems*.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. *Nature, 521*, 436-444.

LeCun, Y., Boser, B., Denker, J. S., Howard, R. E., Hubbard, W., & Jacket, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, *1*(4), 541-551.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 2278-2324.

Lienhart, R., & Maydt, J. (2002). An extended set of Haar-like features for rapid object detection. *International Conference on Image Processing*. Nueva York: IEEE.

Linna, M., Kannala, J., & Rahtu, E. (2015). Online Face Recognition System based on Local. *Advanced Concepts for Intelligent Vision Systems*, 403-414.

Malli, R. C. (2017). Keras VGGFace. Github. Obtenido de https://github.com/rcmalli/keras-vggface

Masi, I., Wu, Y., Hassner, T., & Natarajan, P. (2018). Deep Face Recognition: A Survey. *Conference on Graphics, Patterns and Images (SIBGRAPI).* Parana, Brazil.

Matyás, V., & Ríha, Z. (2002). Biometric Authentication — Security and Usability. *Proc. 6th IFIP TC6/TC11 Conf. Commun. Multimedia Security*, 227-239.

Mohammed, N., Munassar, A., & Govardhan, A. (2010). Comparison Between Five Models Of Software Engineering. *International Journal of Computer Science, 7*(5), 94-101.

Morosan, C. (2012). Biometric solutions for today's travel security problems. *Journal of Hospitality and Tourism Technology, 3*(3), 176-195.

Mozur, P. (8 de Julio de 2018). *Inside China's Dystopian Dreams: A.I., Shame and Lots of Cameras*. Obtenido de The New York Times: https://www.nytimes.com/2018/07/08/business/china-surveillance-technology.html

Narváez, M. M. (2016). *Análisis y reconocimiento de la expresión facial de la emoción en video de personas con demencia*. Loja.

Ng, H.-W., & Winkler, S. (2014). A Data-driven Approach to Cleaning Large Face Datasets. *IEEE International Conference on Image Processing (ICIP).* Paris, France.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning.* Determination Press.

OpenCV. (2012). *Affine Transformations*. Obtenido de https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/warp_affine/warp_affine.html

Ortiz, N., Hernández, R. D., Jimenez, R., Mauledeoux, M., & Avilés, O. (2018). Survey of Biometric Pattern Recognition via Machine Learning Techniques. *Contemporary Engineering Sciences, 11*(34), 1677-1694.

Paisitkriangkrai, S., Shen, C., & Zhan, J. (2010). Face Detection with Effective Feature Extraction. *Asian Conference on Computer Vision* (pp. 460-470). Berlin: Springer.

Parkhi, O., Vedaldi, A., & Zisserman, A. (2015). Deep Face Recognition. *British Machine Vision Conference*, (pp. 41.1-41.12). Swansea.

Perez, L. (2017). When Data Science Becomes Software Engineering. *9th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, (pp. 226-232). Madeira.

Piatetsky, G. (Octubre de 2014). *CRISP-DM, still the top methodology for analytics, data mining, or data science projects*. Obtenido de KD Nuggets: https://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html

Pröve, P. L. (17 de Octubre de 2017). *Squeeze-and-Excitation Networks*. Obtenido de Medium: https://towardsdatascience.com/squeeze-and-excitation-networks-9ef5e71eacd7

Regalado, D. J. (2019). *Reconocimiento de imágenes con técnicas de minería de datos.* Quito.

Remanan, S. (28 de Abril de 2019). *Beginner's Guide to Object Detection Algorithms*. Obtenido de Medium: https://towardsdatascience.com/beginners-guide-to-object-detection-algorithms-6620fb31c375

RENIEC. (7 de agosto de 2014). RENIEC mejora sistema de identificación. *RENIEC*. Obtenido de https://www.reniec.gob.pe/portal/detalleNota.htm?nota=924

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., . . . Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision, 115*(3), 211–252.

Saltz, J. S., Shamshurin, I., & Crowston, K. (2017). Comparing Data Science Project Management Methodologies via a Controlled Experiment. *Hawaii International Conference on System Sciences*, (pp. 1013-1022). Waikoloa.

Santoso, H., & Pulungan, R. (2013). A Parallel Architecture for Multiple-Face Detection Technique Using AdaBoost Algorithm and Haar Cascade. *Information Systems International Conference (ISICO)*, (pp. 592-597).

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 815-823.

Schulte, M. (2013). *Real-time feature extraction from video stream data for stream segmentation and tagging*. Dortmund.

Schutt, R., & O'Neil, C. (2013). *Doing Data Science: Straight Talk from the Frontline*. California: O'Reilly.

Shamik, S., Qian, G., & Pramanik, S. (2002). Segmentation and Histogram Generation Using the HSV Color Space For Image Retrieval. *International Conference on Image Processing*, *2*, pp. 589-592. New York, USA.

Shklar, L., & Rosen, R. (2003). *Web Application Arquitecture: Principles, protocols and practices*. West Sussex, England: John Wiley & Sons Ltd.

Sigal, S. (4 de Setiembre de 2019). Facebook will finally ask permission before using facial recognition on you. *Vox*. Obtenido de https://www.vox.com/future-perfect/2019/9/4/20849307/facebook-facial-recognition-privacy-zuckerberg

Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv 1409.1556*.

Stutz, D. (2014). *Understanding Convolutional Neural Networks.* Aquisgrán: RWTH Aachen University.

Sucar, L. E., & Gómez, G. (2008). *Visión Computacional.* Múnich, Alemania.

Sugata, T. L., & Yang, C. K. (2017). Leaf App: Leaf recognition with deep convolutional neural networks. *IOP Conference Series: Materials Science and Engineering,, 273*.

Suleiman, A., & Sze, V. (2014). Energy-Efficient HOG-based Object Detection at 1080HD 60 fps with Multi-Scale Support. *Proceedings of the 2014 IEEE Workshop on Signal Processing Systems (SiPS)*.

Sun, Y., Wang, X., & Tang, X. (2014). Deep Learning Face Representation from Predicting 10,000 Classes. *CVPR*, 1891–1898.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., . . . Rabinovich, A. (2014). Going deeper with convolutions. *Computer Vision and Pattern Recognition* .

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications.* Londres: Springer.

Taigman, Y., Yang, M., Ranzato, M. A., & Wolf, L. (2014). DeepFace: Closing the Gap to Human-Level Performance in Face Verification. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1701-1708.

Toledano, D. T., Pozo, R. F., Trapote, A. H., & Gómez, L. H. (2006). Usability evaluation of multi-modal biometric. *Interacting with Computers, 18*, 1101-1122.

Universidad ESAN. (2016). Proyectos de pregrado en encuentros de investigación. Obtenido de https://www.informesan.edu.pe/proyectos-de-pregrado-en-encuentros-de-investigacion-2/

Valenzuela, M. (12 de Agosto de 2015). Regla de Aprendizaje de Retropropagación. *Sistemas Conexionistas y Evolutivos (IA–5005)*.

Van den Broek, E. L. (2005). Human-centered content-based image retrieval. Utrecht, Holanda: Utrecht University.

Vijayasarathy, L. R., & Butler, C. W. (2016). Choice of Software Development Methodologies. Do Organizational, Project and Team Characteristics Matter? *IEEE Software*, 86-94.

Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *Computer Vision and Pattern Recognition (CVPR).*

Voulodimos, A., Doulamis, N., Doulamis, A., & Protopapadakis, E. (2018). Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience*, 13.

Wang, M., & Deng, W. (2018, 4 24). Deep Face Recognition: A Survey. *ArXiv*, 1-17.

Wang, X., Han, T. X., & Yan, S. (2009). An HOG-LBP Human Detector with Partial Occlusion Handling. *Proc. 12th IEEE Int'l Conf.*

Wikimedia Commons. (22 de Noviembre de 2015). EM spectrum. Wikimedia Commons, the free media repository. Obtenido de https://commons.wikimedia.org/w/index.php?title=File:EM_spectrum.svg&oldid=179921766

Wikimedia Commons. (6 de Agosto de 2018). RGB color solid cube. Wikimedia Commons, the free media repository. Obtenido de https://commons.wikimedia.org/w/index.php?title=File:RGB_color_solid_cube.png&oldid=313837060.

Wolf, L., Hassner, T., & Maoz, I. (2011). Face Recognition in Unconstrained Videos with Matched Background Similarity. *Computer Vision and Pattern Recognition (CVPR)*.

Wolf, L., Hassner, T., & Maoz, I. (31 de Octubre de 2012). *YouTube Faces*. Obtenido de YouTube Faces DB: https://www.cs.tau.ac.il/~wolf/ytfaces/

Wu, X., He, R., Sun, Z., & Tan, T. (2018). A light CNN for deep face representation with noisy labels. *IEEE Transactions on Information Forensics and Security, 13*(11), 2884-2896.

Xiang, A. (2018). Light CNN. Github. Obtenido de https://github.com/AlfredXiangWu/LightCNN

Xicali, J. A. (2019). *Sistema Biométrico para la Clasificación de Grupos Étnicos*. Puebla: Universidad Autónoma de Puebla.

Yi, D., Lei, Z., Liao, S., & Li, S. Z. (2014). Learning Face Representation from Scratch.

Zhao, W., Chellappa, R., Phillips, J. P., & Rosenfeld, A. (2003, Diciembre). Face Recognition: A Literature Survey. *ACM Computing Surveys, 35*(4), 399–458.

Zhao, Z.-Q., Zheng, P., Xu, S.-t., & Wu, X. (2018). Object Detection with Deep Learning: A Review. *IEEE transactions on neural networks and learning systems*.

ANEXOS

Anexo 1 – Matriz de Consistencia

Problemas	Objetivos	Hipótesis	Variables
PG: ¿Cómo el preprocesamiento de imágenes impacta en la efectividad de la verificación facial empleando visión computacional?	OG: Evaluar el impacto del preprocesamiento de imágenes en la efectividad de la verificación facial empleando visión computacional.	HG: El preprocesamiento de imágenes impacta en la efectividad de los modelos de verificación facial empleando visión computacional.	V1: Método de preprocesamiento V2: Efectividad del modelo
PE1: ¿El impacto del preprocesamiento en la efectividad de la verificación facial se verá alterado al analizarse en diferentes bases de datos de rostros disponibles?	OE1: Obtener y generar datasets de rostros que permitan realizar el análisis de los modelos de verificación facial.	HE1: El impacto del preprocesamiento en la efectividad de la verificación facial se verá afectado por los datasets a evaluar.	V1: Fuente de información V2: Efectividad del modelo
PE2: ¿El impacto del preprocesamiento en la efectividad de la verificación facial variará al analizarse con distintas técnicas de preprocesamiento?	OE2: Aplicar las técnicas de preprocesamiento mediante suavizamiento, agudizamiento, ecualización y alineación para realizar el análisis de la efectividad de la verificación facial.	HE2: El impacto del preprocesamiento en la efectividad de la verificación facial variará en base al tipo de preprocesamiento utilizado.	V1: Método de preprocesamiento V2: Efectividad del modelo

PE3: ¿El impacto del preprocesamiento en la efectividad de la verificación facial se verá afectado al analizarse con diferentes métodos de detección facial?	OE3: Utilizar los métodos HOG y Haar para la extracción de los vectores característicos de la detección facial.	HE3: El impacto del preprocesamiento en la efectividad de la verificación facial dependerá del método de detección facial utilizado.	V1: Método de detección facial V2: Efectividad del modelo
PE4: ¿El impacto del preprocesamiento en la efectividad de la verificación facial variará al analizarse con diferentes modelos de verificación facial basados en redes neuronales convolucionales?	OE4: Aplicar los modelos de redes neuronales convolucionales OpenFace, VGG-Face2 y Light CNN para realizar las evaluaciones de efectividad de la verificación facial.	HE4: El impacto del preprocesamiento en la efectividad de la verificación facial dependerá de los modelos de redes neuronales convolucionales utilizados.	V1: Modelo de verificación facial V2: Efectividad del modelo
PE5: ¿Las métricas para medir el desempeño dan una visión comparable mediante una correcta evaluación del impacto del preprocesamiento en la efectividad de la verificación facial?	OE5: Analizar el impacto del preprocesamiento de imágenes en la efectividad de la verificación facial mediante métricas de desempeño de las redes OpenFace, Vgg-Face2 y Light CNN.	HE5: Las métricas de desempeño para medir la efectividad serán comparables y permitirán evaluar el impacto del preprocesamiento en la efectividad de la verificación facial.	V1: Efectividad del modelo

Anexo 2 – Resumen de resultados

Combinación	Efectividad	Efectividad en conjuntos de Prueba	Punto de Corte
LightCNN			
HOG			
DNI	77.35	77.40	243.23
LFW	93.96	93.95	244.84
YTF	76.67	76.67	234.61
HAAR			
DNI	78.53	78.55	248.35
LFW	92.10	92.07	239.59
YTF	75.39	75.39	236.14
OpenFace			
HOG			
DNI	67.44	67.50	1.03
LFW	76.34	76.34	0.95
YTF	66.08	66.08	1.01
HAAR			
DNI	67.66	67.62	0.95
LFW	74.57	74.57	0.93
YTF	65.84	65.83	1.01
VGGFace2			
HOG			
DNI	78.63	78.44	187.96
LFW	95.79	95.79	188.37
YTF	81.66	81.64	193.94
HAAR			
DNI	79.29	78.92	186.64
LFW	94.20	94.20	186.45
YTF	80.59	80.57	191.57

Para ver todo el detalle de resultados, se pueden descargar de https://github.com/btafur/Tesis-2019.