SISTEMA DE RECONOCIMIENTO FACIAL CON MASCARILLAS

David Alfredo Torres Montalvo

Junior Raúl Sánchez Suárez

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2022

SISTEMA DE RECONOCIMIENTO FACIAL CON MASCARILLAS

David Alfredo Torres Montalvo

Junior Raúl Sánchez Suárez

Trabajo de Grado para optar al título de Ingeniero Electrónico

Director
Jaime Guillermo Barrero Perez
Mg. en potencia eléctrica

Universidad Industrial de Santander
Facultad de Ingenierías Fisicomecánicas
Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones
Bucaramanga
2022

Agradecimientos

Durante el duro y riguroso camino que se presenta a lo largo de los años de la etapa universitaria se pueden identificar personas con las que se puede disfrutar mucho este momento de la vida. Son estas personas las que merecen un lugar en este apartado ya que gracias a ellas hemos podido avanzar hasta llegar a este punto, al final de esta interesante carrera.

Familiares, profesores, amigos y compañeros merecen un gran agradecimiento ya que han influido de una forma u otra en nuestra vida universitaria, son grandes personas que nos ha otorgado la vida y que han hecho de este lapso, una etapa maravillosa.

El primer agradecimiento especial es para nuestro guía y director Jaime Guillermo Barrero quien nos ha acompañado desde semestres anteriores y nos mostró el camino para realizar esta investigación con paciencia y dedicación.

Un agradecimiento especial debe ser también mencionado para todas aquellas personas que sacaron de su tiempo y esfuerzo para prestarnos sus rostros y darle vida a una parte de nuestro proyecto, su disposición y compromiso estará siempre en nuestros recuerdos mas preciados ya que sin su apoyo, este trabajo de grado no hubiese sido posible.

Dedicatoria

A mis padres Yony Alfredo y Yuly Esthela, en especial a ella, quien después de la partida de mi padre suplió el gran papel de ser ambos a la vez, brindándome la gran fuerza de voluntad y convicción para lograr mis metas.

A mis amigos quienes siempre socializaron y dieron buenos consejos en los momentos de tristeza y familiares quienes me acompañaron y acompañaran en los momentos cruciales de mi vida.

David Alfredo Torres Montalvo

A mis padres Raúl y Luz Marina, quienes con su apoyo forjaron la persona en la que me he convertido hasta este momento de mi vida y quienes fueron el eje fundamental en el que se apoyó mi carrera, sueños y esperanzas.

A mi familia y amigos, quienes nunca dejaron de creer en mi sin importar cuan fuerte soplaran lo vientos y quienes hicieron que este camino fuese mas sencillo de recorrer.

Junior Raúl Sánchez Suárez

Índice general

Índice de figuras

Índice de tablas

Abstract

Title: FACIAL RECOGNITION SYSTEM WITH MASKS

Authors: David Alfredo Torres Montalvo, Junior Raúl Sánchez Suárez

Key Words: COVID-19, face masks, machine learning, statistical analysis.

Description:

COVID-19 has caused problems that have shaken the world in recent years and this has forced many fields of science to adapt to the new reality that is currently being experienced. Many of the technological sectors of the industry have focused on supporting and reinforcing biosecurity standards to prevent the spread of this virus that has claimed the lives of millions of people. In this project an algorithm was designed that, with the help of machine learning and database assembly, was able to determine whether or not an individual is wearing a face mask correctly. This recognition was performed through the camera of embedded devices such as the Raspberry PI 3B+ and the Maix Go by sipeed where these algorithms were implemented. Different machine learning methods were tested and applied, and their accuracy and performance were analyzed using statistical analysis techniques to find an optimal algorithm in accuracy, performance, computational cost and size that best adapted to the limited resources of the embedded device in which it was implemented. Finally, the algorithms were compared and the viability of each of them was observed for applications focused on supporting the identification of people who do not follow these biosafety rules.

Resumen

Titulo: SISTEMA DE RECONOCIMIENTO FACIAL CON MASCARILLAS

Autores: David Alfredo Torres Montalvo, Junior Raúl Sánchez Suárez

Palabras clave: COVID-19, mascarilla, aprendizaje automático, análisis estadístico.

Descripción:

El COVID-19 ha suscitado problemáticas que han sacudido al mundo durante los últimos años y esto ha obligado a muchos campos de la ciencia a adaptarse a la nueva realidad que se está viviendo actualmente. Muchos de los sectores tecnológicos de la industria han puesto su enfoque en apoyar y reforzar las normas de bioseguridad para prevenir la propagación de este virus que ha cobrado la vida de millones de personas. En el presente proyecto se diseñaron tres algoritmos, que con la ayuda del aprendizaje automático y el montaje de bases de datos, pudieron determinar si un individuo lleva puesta correctamente una mascarilla facial o no. Este reconocimiento se realizó a través de la cámara de dispositivos embebidos como la Raspberry PI 3B+ y la Maix Go de Sipeed en donde fueron implementados dichos algoritmos. Se probaron y se aplicaron diferentes métodos de aprendizaje automático y se analizó su precisión y rendimiento mediante técnicas de análisis estadístico para buscar un algoritmo óptimo en cuestiones de precisión, rendimiento, costo computacional y tamaño que se adaptara lo mejor posible a los recursos limitados con los que cuenta el dispositivo embebido en el que se implementó. Finalmente, se compararon los algoritmo y se observó la viabilidad de cada uno de ellos para aplicaciones enfocadas a servir de apoyo para identificar a las personas que no cumplen estas normas de bioseguridad.

Introducción

A lo largo de los últimos meses, la humanidad ha tenido que enfrentar la problemática sanitaria ocasionada por el COVID-19. Dicha problemática ha amenazado la cotidianidad en la vida de todos los países a lo largo y ancho del planeta, obligándolos a tomar medidas drásticas en cuestiones básicas de salubridad, higiene y bioseguridad.

Dadas las repercusiones que ha tenido la pandemia en la salud de las personas, la Organización Mundial de la Salud (OMS) ha realizado una serie de recomendaciones oficiales de bioseguridad que se han venido actualizando a medida que se obtiene más información de este virus Organización Mundial De La Salud [OMS], 2021. Estas recomendaciones dadas por la OMS a todos los países tienen como objetivo mitigar los brotes del virus que se extendieron rápidamente al rededor del mundo y que ya han cobrado la vida de algunos millones de personas según las más recientes estadísticas obtenidas por la universidad de John Hopkins (Johns Hopkins University, 2021).

Dentro de las recomendaciones dadas por la OMS se encuentran: el lavado frecuente de manos, el distanciamiento social, la ventilación adecuada de los lugares habitados y el uso adecuado de las mascarillas, que buscan disminuir el nivel de contagio mediante la higiene en el que viven las personas.

Una de las recomendaciones de salubridad propuesta por la OMS que más ha repercutido dentro de la normalidad de las personas ha sido el uso constante de la mascarilla en sitios públicos de espacio reducido, las cuales evitan la salida de gotículas de agua de entre uno y cien micrómetros de diámetro cargadas de bacterias, patógenos y virus como el COVID-19, desde un paciente infectado que habla, tose o simplemente respira. Por otra parte, las mascarillas también impiden el ingreso de microorganismos como virus y bacterias al sistema respiratorio por diversos métodos que capturan dichas partículas de forma efectiva sin importar mucho su tamaño (Courty y Kierlik, 2020). El uso de las mascarillas, para prevenir el contagio por COVID-19, es un importante requerimiento que se ha hecho tan indispensable en la mayoría de las ciudades del mundo y su uso es obligatorio de forma

permanente mientras se esté en sitios públicos o concurridos. El no cumplimiento de esta norma podría acarrear problemas que pueden ir desde la expulsión de sitios públicos hasta sanciones económicas.

Actualmente existen varias formas de vigilar el uso constante de las mascarillas por parte de los usuarios de sitios públicos. Ya que dicha tarea se convierte en repetitiva y tediosa para el personal de vigilancia encargado, se ha propuesto el uso de la tecnología para realizarla de forma óptima y eficaz, tal y como se propone en este documento. Es así que se piensa en la posibilidad de hacer uso de librerías presentes en lenguajes de programación enfocadas al procesamiento de imágenes (OpenCV team, 2021) y a la inteligencia artificial (Google, 2021) para detectar, mediante cámaras, a los usuarios que no hacen uso de la mascarilla facial. Dichas cámaras pueden ser ubicadas en sitios estratégicos, como entradas de locales o pasillos de centros comerciales, y ser configuradas para enviar una alerta cuando se haga detección de un individuo que no hace uso de la mascarilla y que presenta una amenaza a la salubridad e higiene del lugar.

Al respecto de esta problemática, dentro de los desarrollos tecnológicos e investigaciones que se han dado en pro de la prevención del contagio del COVID-19, se tienen varios ejemplos que buscan dar un paso adelante en la lucha contra la propagación del virus. Uno de ellos es un proyecto desarrollado por la universidad de Barcelona llamado "LogMask" (Universitat de Barcelona, 2020), el cual se basa en técnicas de inteligencia artificial y computación de imágenes en tiempo real para así determinar en zonas con alta concurrencia de personas quienes llevan mascarilla y quienes no la llevan, con el fin de realizar permanente monitorización de estas eventualidades, hacer control de los aforos y emitir alertas de personas sin mascarilla cuando así sea requerido.

Otro de los avances y desarrollos que se ha generado por el tema de la prevención de este virus se dio en la universidad de Guayaquil con un proyecto de grado llamado "Prototipo web detector de mascarilla mediante RTMP y visión artificial para el control dentro del transporte público del norte de Guayaquil en tiempo de pandemia" (Chóez Vera y Palma

Toledo, 2021) el cual haciendo uso de algoritmos de reconocimiento de rostros y modelos de clasificación permitió determinar usuarios del transporte público de Guayaquil que portaban correctamente mascarilla y usuarios que no lo hacían para así ayudar a las autoridades a identificar a quienes infrinjan esta norma esencial de bioseguridad.

Dado el marco expuesto y estudios enunciados, es pertinente ahondar en la proposición de alternativas tecnológicas que aporten a la solución de situaciones que puedan constituir problemas de bioseguiridad, de ahí que el presente proyecto, trata del desarrollo de un algoritmo capaz de determinar si un individuo porta o no su mascarilla facial correctamente por medio de procesamiento de imágenes (Rafael C. González, 2008) técnicas de inteligencia artificial (Bazaraa y col., 2013) y análisis de efectividad (Sun y col., 2019) aplicados al algoritmo en cuestión. Dentro de la documentación del proyecto podemos encontrar seis capítulos que serán detallados a continuación:

Capítulo 1: Describe los objetivos, tanto generales como específicos del proyecto.

Capitulo 2: Presenta los marcos conceptuales y teóricos que sirvieron de apoyo para la ejecución del proyecto.

Capítulo 3: Muestra el paso a paso de la ejecución del proyecto y presenta los resultados de rendimiento de los algoritmos utilizados

Capítulo 4: Presenta los resultados obtenidos en la implementación.

Capítulo 5: Expone las conclusiones que se pueden sacar a partir de los resultados obtenidos del capítulo anterior.

Capítulo 6: Dispone de recomendaciones hechas por los autores del proyecto con el fin de apoyar a futuras investigaciones.

1. Objetivos

1.1. Objetivo general

Diseñar un algoritmo que permita determinar si una persona hace uso de mascarilla, además de identificar si lo hace de manera correcta, para ser implementado en un sistema embebido.

1.2. Objetivos específicos

- Construir una base de datos a partir de imágenes obtenidas en la web, complementada con imágenes adquiridas de manera local con la cual sea posible entrenar y probar el sistema de reconocimiento facial con mascarillas.
- Desarrollar un algoritmo en lenguaje Python haciendo uso de librerías de visión por computadora e inteligencia artificial, que permita determinar si el usuario hace uso correcto o no de una mascarilla.
- Evaluar la precisión del sistema de reconocimiento utilizando imágenes en tiempo real obtenidas a través de la cámara de sistema embebido Raspberry.

2. Marco de Referencia

2.1. Estado del arte

Dentro de los desarrollos tecnológicos e investigaciones que se han desarrollado en pro de la prevención del contagio del COVID-19 se tienen varios ejemplos que buscan dar un paso adelante en la lucha contra la propagación del virus. Uno de ellos es un proyecto desarrollado por la universidad de Barcelona llamado "LogMask" (Universitat de Barcelona, 2020), el cual se basa en técnicas de inteligencia artificial y computación de imágenes en tiempo real para así determinar en zonas con alta concurrencia de personas quienes llevan mascarilla y quienes no la llevan con el fin de realizar permanente monitorización de estas eventualidades, hacer control de los aforos y emitir alertas de personas sin mascarilla cuando así sea requerido.

Otro de los avances y desarrollos que se ha generado por el tema de la prevención de este virus se dio en la universidad de Guayaquil con un proyecto de grado llamado "PROTOTIPO WEB DETECTOR DE MASCARILLA MEDIANTE RTMP Y VISIÓN ARTIFICIAL PARA EL CONTROL DENTRO DEL TRANSPORTE PÚBLICO DEL NORTE DE GUAYAQUIL EN TIEMPO DE PANDEMIA" (Chóez Vera y Palma Toledo, 2021) el cual, haciendo uso de algoritmos de reconocimiento de rostros y modelos de clasificación, permitió determinar cuáles usuarios del transporte público de Guayaquil portaban correctamente mascarilla y cuales usuarios no lo hacían para así ayudar a las autoridades a identificar a quienes infringían esta norma esencial de bioseguridad.

Por otro lado, una de las aplicaciones bastante útiles para la visión artificial es el control de aforo para lugares públicos, tema que ha sido desarrollado por Hanwha Techwin y su "aplicación para control de aforo" (Europe, s.f.) como solución a la problemática del distanciamiento social la cual cuenta simultáneamente el número de personas que entran y salen de un sitio público y ofrece dicha información a directivas del lugar. También una vez que se ha alcanzado el número de personas máximo en el lugar se generan alarmas de aforo,

control de puertas y señales de espera y entrada para los usuarios.

Haciendo una revisión detallada de las investigaciones y trabajos anteriormente mencionados en esta sección, es posible destacar que la mayoría de dichas soluciones son implementadas en dispositivos de alto consumo energético, alto costo y altas capacidades de computo. Debido a esto, se considera pertinente para los autores de este trabajo de grado abordar una solución que permita la portabilidad, el bajo consumo energético y el bajo costo en la implementación de algoritmos computacionales que buscan mitigar el impacto del COVID-19 durante este tiempo de pandemia.

2.2. Procesamiento de imágenes

Durante los últimos años y gracias a la facilidad con la que se han propagado las tecnologías de información y comunicación por el mundo, el procesamiento de imágenes ha pasado a cumplir un papel muy significativo en diferentes áreas de la ciencia, la medicina y la tecnología. Desde la diagnosis médica automática hasta la exploración espacial, las aplicaciones del procesamiento digital de imágenes han comenzado a crecer significativamente hasta el punto de ser una parte común de nuestras vidas, siendo implementadas en muchos dispositivos a nuestro alrededor (A. D. Torres, 1996).

2.2.1. Imágenes y visión humana

Las capacidades visuales del ojo humano están bastante limitadas por la intensidad de luz a la que es sometido y por la longitud de onda de esta luz. La ventana de longitudes de onda que es capaz de detectar el ojo humano va aproximadamente desde los 400 [nm] del color violeta hasta los 700 [nm] del color rojo oscuro y es llamada “espectro visible” tal y como se puede observar en la figura 2.1.

Esta ventana de longitudes de onda que puede percibir el ojo humano de manera natural es bastante pequeña respecto a la gran cantidad de longitudes de onda que se utilizan en la actualidad para diversas aplicaciones que pueden ir desde los rayos X hasta la

transmisión de información por ondas de radio entre muchas otras.

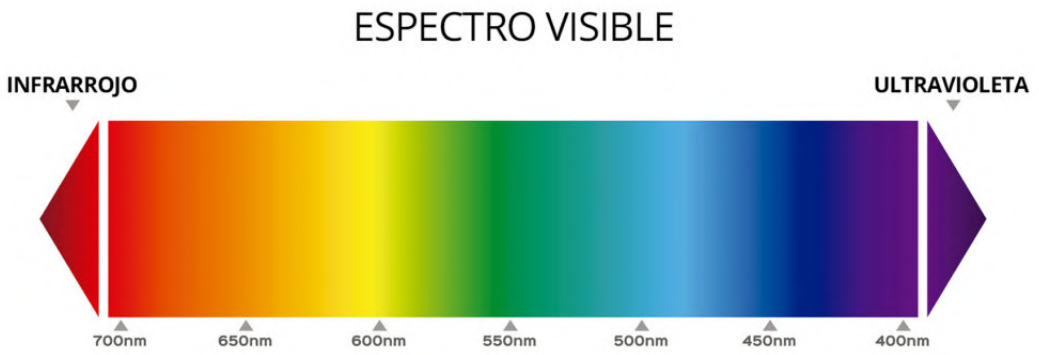


Figura 2.1: Espectro visible.

Tomado de: https://opticabalbuenafuengirola.com/como-afecta-la-luz-azul-de-los-ordenadores-a-nuestra-vision

Gracias a la variedad de longitudes de onda en las que se puede trabajar, es posible crear imágenes a partir de la detección de señales en otras longitudes de onda distintas a las de la franja del espectro visible. Esto ha sido de gran utilidad a lo largo del tiempo para analizar fenómenos que están fuera de las capacidades del ojo humano tales como las radiografías en el ámbito médico, las cuales se trabajan en la banda de los rayos X (de 0.01 [nm] a 10 [nm]), o la radioastronomía que buscar detectar ondas provenientes de otros sistemas estelares con longitudes de onda entre 7 [mm] y 10[m].

2.2.2. Imágenes en la computación

Para entender a plenitud como pueden las computadoras percibir las imágenes se debe tener conocimiento de diversas áreas de la ciencia tales como óptica, matemáticas e informática.

Como primera medida, es necesario entender que una imagen es una representación que contiene información descriptiva de alguna otra cosa u objeto que busca representar, tal como las fotografías, los cuadros y las imágenes digitales con las que se trabaja en este proyecto. Otro concepto importante que debe ser tenido en cuenta es el hecho de que las

imágenes, tal y como las entienden las computadoras, son digitales, y por ende son una colección discreta, matricial y organizada de datos que representan factores de color, brillo e iluminación en rangos predefinidos (Solomon y Breckon, 2011).

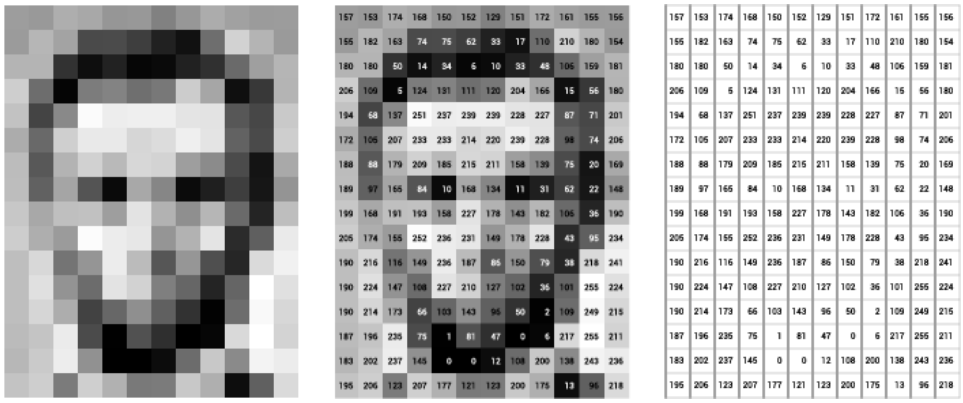


Figura 2.2: Representación de una imagen en una computadora.

Tomado de: https://towardsdatascience.com/understanding-images-with-skimage-python-b94d210afd23

Las imágenes en blanco y negro son usualmente una única capa matricial que representa la intensidad del color blanco en cada píxel de la imagen en cuestión, tal y como se puede ver en la figura 2.2. Por otro lado las imágenes a color suelen tener tres capas que representan características especiales que pueden ser útiles de acuerdo a como se desee ver la imagen. Por un lado tenemos los formatos de los píxeles en espacio RGB (red, green, blue), el cual es un formato que nos proporciona información sobre la intensidad de cada uno de los colores rojo, verde y azul que existe en cada píxel de una imagen y que logra la conformación de otros colores a partir de combinar diferentes intensidades de estos tres colores mencionados. Por otro lado tenemos formatos como el HSI, el cual representa características de los píxeles tales como el tono, la saturación y la intensidad. Por último, para aplicaciones mas especificas existen formatos como el CMYK, el cual permite la conformación de colores mas precisos en aplicaciones de impresión y representación física de imágenes a partir de los colores Cían, Magenta, amarillo y negro.

2.3. Inteligencia artificial

Acuñado inicialmente en la década de los cincuenta, el termino "inteligencia artificial" fue una teoría sobre la inteligencia humana que podían exhibir las maquinas (Helm y col., 2020). Aunque el concepto puede remontarse al siglo XVII con la invención de las maquinas creadas por Pascal y Leibniz, las cuales fueron capaces de simular la capacidad humana para hacer operaciones matemáticas básicas como sumas y restas (Ifrah, 2001), sólo fue hasta mediados del siglo XX que se comenzó a dar relevancia a la capacidad de aprendizaje que podían llegar a tener las maquinas. Junto con los avances de las tecnologías y la creciente capacidad de cómputo que comenzó a existir para esa época fueron creciendo las posibilidades de simular diversas tareas que exigían un aprendizaje humano.

Desde los primeros usos del termino *machine learning*, creado por Arthur Samuel de IBM quien demostró que era posible programar un computador para que aprendiera a jugar damas, hasta los mas recientes algoritmos de aprendizaje distribuido del *deep learning* (El Naqa y Murphy, 2015) las aplicaciones de estas tecnologías se han ido convirtiendo en una parte común y esencial para hacer mas sencilla y cómoda la vida humana con tareas básicas como sugerencias de vídeos de acuerdo a conductas de visualizaciones en plataformas de *streaming*, reconocimientos faciales para acceder de forma segura a los dispositivos o simplemente anuncios personalizados de cosas que se podrían querer o necesitar.

2.3.1. Machine Learning

El aprendizaje automático es un tema del campo de la inteligencia artificial. Utiliza algoritmos para permitir que las computadoras reconozcan patrones en grandes cantidades de datos y hagan predicciones (análisis predictivo). Este aprendizaje permite a la computadora realizar tareas específicas de forma autónoma, es decir, sin programación.

El término se utilizó por primera vez en 1959 pero en los últimos años se ha vuelto cada vez más importante debido a la mejora de la potencia informática y la prosperidad de los datos. De hecho, la tecnología de aprendizaje automático es una parte fundamental del

big data.

Los algoritmos de *machine learning* se pueden dividir en tres categorías como se describen a continuación :

2.3.1.1. Aprendizaje supervisado

Estos algoritmos tienen un aprendizaje a priori basado en el sistema de etiquetado asociado a los datos, lo que les permite tomar decisiones o hacer predicciones. Un ejemplo es un detector de *spam*, que marca los correos electrónicos como *spam* basándose en patrones aprendidos del historial de correo electrónico (remitente, relación texto - imagen, palabras clave en el asunto, etc.).

2.3.1.2. Aprendizaje no supervisado

Estos algoritmos no tienen conocimiento previo. Se enfrentan al caos de los datos para encontrar patrones que les permitan organizarlos de una determinada manera. Por ejemplo, en el campo del *marketing*, se utilizan para extraer patrones de datos masivos de las redes sociales y crear campañas publicitarias altamente segmentadas.

2.3.1.3. Aprendizaje por refuerzo

Su objetivo es permitir que el algoritmo aprenda de su propia experiencia. La mejor decisión se puede tomar en diferentes situaciones según una serie de procesos de prueba y error, y la decisión correcta será recompensada. Actualmente se utiliza para el reconocimiento facial, el diagnóstico médico o la clasificación de secuencias de ADN.

2.3.1.4. *Eigenface*

Cuando se utiliza en problemas de visión por computadora para el reconocimiento facial, los $Eigenfaces$ es el nombre que se le asigna a un conjunto de vectores propios. El método de utilizar $Eigenfaces$ para el reconocimiento fue desarrollado por Sirovich

y Kirby (1987) y utilizado por Matthew Turk y Alex Pentland para la clasificación de rostros. El vector de características proviene de la matriz de covarianza de la distribución de probabilidad en el espacio vectorial de alta dimensión de la imagen de la cara. El $Eigenface$ en sí mismo constituye el conjunto básico de todas las imágenes utilizadas para construir la matriz de covarianza. Al permitir que el conjunto más pequeño de imágenes básicas represente las imágenes de entrenamiento originales, se reduce la dimensionalidad. La clasificación se puede lograr comparando la representación del rostro en un conjunto de bases.

Al realizar un proceso matemático llamado análisis de componentes principales (ACP) en una gran cantidad de imágenes que representan diferentes rostros humanos, se puede generar un conjunto de rostros característicos. De manera informal, $Eigenfaces$ se puede considerar como un conjunto de çomponentes faciales estandarizados"derivados del análisis estadístico de muchas imágenes de rostros. Cualquier cara puede considerarse como una combinación de estas caras estándar. Por ejemplo, la cara de una persona puede constar de un $Eigenface$ promedio más el 10 % del $Eigenface$ #1, el 55 % del $Eigenface$ #2 o incluso el -3 % del $Eigenface$ #3. Además, dado que los rostros no se registran mediante fotografías digitales, sino como una simple lista de valores (un valor para cada rostro propio en la base de datos utilizada), cada rostro ocupa mucho menos espacio.

Los $Eigenfaces$ creados aparecerán como zonas claras y oscuras dispuestas en un patrón específico. Este patrón es la forma en la que se seleccionan los diferentes rasgos de un rostro para ser evaluados y puntuados. Habrá un patrón para evaluar la simetría, si hay algún estilo de vello facial, dónde está la línea del cabello o una evaluación del tamaño de la nariz o la boca. Otros $Eigenfaces$ tienen patrones que son menos sencillos de identificar, y la imagen del $Eigenface$ puede parecerse muy poco a una cara.

La técnica empleada en la creación de $Eigenfaces$ y su uso para el reconocimiento también se utiliza fuera del reconocimiento de rostros: reconocimiento de escritura a

mano, lectura de labios, reconocimiento de voz, interpretación del lenguaje de signos - gestos de la mano y análisis de imágenes médicas. Por ello, algunos no utilizan el término ' $Eigenfaces$ ', sino que prefieren utilizar ' $Eigenimages$ '.

Ahora bien el problema de la representación de la imagen que se tiene es su alta dimensionalidad. Las imágenes bidimensionales en escala de grises $p \times q$ abarcan un espacio vectorial de $m = p \times q$ dimensiones, por lo que una imagen de 100×100 píxeles se encontraría ya en un espacio de imagen de 10.000 dimensiones. La pregunta es: ¿son todas las dimensiones igual de útiles?, sólo podemos tomar una decisión si hay alguna varianza en los datos, así que lo que buscamos son los componentes que representan la mayor parte de la información. El análisis de componentes principales (ACP) fue propuesto de forma independiente por Karl Pearson (1901) y Harold Hotelling (1933) para convertir un conjunto de variables posiblemente correlacionadas en un conjunto más pequeño de variables no correlacionadas. La idea es que un conjunto de datos de alta dimensión suele estar descrito por variables correlacionadas y, por tanto, sólo unas pocas dimensiones significativas representan la mayor parte de la información. El método (ACP) encuentra las direcciones con mayor varianza en los datos, denominadas componentes principales. (Wikipedia contributors, 2021)

2.3.1.4.1. Descripción del algoritmo de *Eigenface*

Sea $X = \{x_1, x_2, \ldots, x_n\}$ un vector aleatorio con observaciones $x_i \in R^d$

1. Se calcula la media μ.

$$\mu = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.1}$$

2. Se calcula la matriz de covarianza S.

$$S = \frac{1}{n} \sum_{i=1}^{n} (x_i - \mu)(x_i - \mu)^T \tag{2.2}$$

3. Se calculan los eigenvalores λ_i y los eigenvectores v_i de S.

$$Sv_i = \lambda_i v_i, i = 1, 2, ..., n \tag{2.3}$$

4. Se ordena los vectores propios descendiendo por su valor propio. Los k componentes principales son los vectores propios correspondientes a los k mayores valores propios.

Los k componentes principales del vector observado x vienen entonces dados por:

$$y = W^T(x - \mu) \tag{2.4}$$

Donde $W = (v_1, v_2, \ldots, v_k)$

La reconstrucción a partir de la base ACP viene dada por:

$$x = Wy + \mu \tag{2.5}$$

Donde $W = (v_1, v_2, \ldots, v_k)$.

El método $Eigenfaces$ realiza entonces el reconocimiento facial se la siguiente forma:

1. Proyectando todas las muestras de entrenamiento en el subespacio ACP.

2. Proyectando la imagen de consulta en el subespacio ACP.

3. Encontrar el vecino más cercano entre las imágenes de entrenamiento proyectadas y la imagen de consulta proyectada.

2.3.1.4.2. Aplicación en el reconocimiento facial.

El reconocimiento facial fue la motivación para la creación de los $Eigenfaces$. Para este uso, los $Eigenfaces$ presentan ventajas sobre otras técnicas disponibles, entre estas la rapidez y la eficacia del sistema. Como el $Eigenface$ es principalmente un método de reducción de dimensiones, un sistema puede representar a muchos sujetos con un conjunto

de datos relativamente pequeño. Como sistema de reconocimiento de rostros, también es bastante invariable a las grandes reducciones de tamaño de las imágenes; sin embargo, empieza a fallar considerablemente cuando la variación entre las imágenes vistas y la imagen de prueba es grande.

Para reconocer los rostros, las imágenes de la galería (las vistas por el sistema) se guardan como colecciones de pesos que describen la contribución de cada $Eigenface$ a esa imagen. Cuando se presenta un nuevo rostro al sistema para su clasificación, se encuentran sus propios pesos proyectando la imagen sobre la colección de $Eigenfaces$. De este modo se obtiene un conjunto de pesos que describen la cara de prueba. A continuación, estos pesos se clasifican con todos los pesos del conjunto de la galería para encontrar la coincidencia más cercana. El método del vecino más cercano (KNN) es un enfoque sencillo para encontrar la distancia euclidiana entre dos vectores, donde el mínimo puede clasificarse como el sujeto más cercano.

Intuitivamente, el proceso de reconocimiento con el método $Eigenface$ consiste en proyectar las imágenes de la consulta en el espacio facial abarcado por los $Eigenfaces$ calculados, y encontrar la coincidencia más cercana con una clase de rostro en ese espacio.

2.3.1.4.3. Pseudocódigo

1. Dado un vector de imagen de entrada $U \in R^n$, teniendo la media del vector de imágenes de la base de datos M, Se calcula el peso del k-ésimo $Eigenface$ como :

$$w_k = V_k^T(U - M) \tag{2.6}$$

   formando un vector de pesos $W = \{w_1, w_2, w_3, ..., w_k, ..., w_n\}$

2. Compare W con los vectores de peso W_m de las imágenes de la base de datos. Encon-

trando la distancia euclidiana.

$$d = ||W - W_m||^2 \tag{2.7}$$

3. Si $d < \epsilon_1$, entonces la m-ésima entrada de la base de datos es un candidato a ser reconocido.

4. Si $\epsilon_1 < d < \epsilon_2$, entonces U puede ser un rostro desconocido y puede añadirse a la base de datos.

5. Si $d > \epsilon_2$, U no es una imagen de un rostro.

Los pesos de cada imagen del conjunto de datos sólo transmiten información que describe esa imagen, mas no a la persona. Una imagen de una persona bajo iluminación frontal puede tener pesos muy diferentes a los del mismo sujeto bajo una fuerte iluminación por la izquierda de la imagen. Esto limita la aplicación de este sistema. Los experimentos realizados en el documento original de $Eigenface$ presentaron los siguientes resultados: una media del 96 % con variación de luz, del 85 % con variación de orientación y del 64 % con variación de tamaño.

Se han realizado varias extensiones del método $Eigenface$, como las $Eigenfiguras$. Este método combina la métrica facial (que mide la distancia entre los rasgos faciales) con la representación Eigenface. (Wikipedia contributors, 2021)

2.3.1.5. *Local binary pattern histogram* (LBPH)

Local binary pattern es un tipo de descriptor visual utilizado para la clasificación en visión por computadoras. El LBP es el caso particular del modelo de Espectro de Textura propuesto en el año de 1990. El LBP se describió por primera vez en 1994. Desde entonces, se ha comprobado que es una potente característica para la clasificación de texturas; además, se ha determinado que cuando el LBP se combina con el descriptor

Histograma de gradientes orientados (HOG), mejora considerablemente el rendimiento de la detección en algunos conjuntos de datos. Básicamente (LBP) es un operador de textura sencillo pero muy eficaz que etiqueta los píxeles de una imagen mediante el umbral de la vecindad de cada píxel y considera el resultado como un número binario.

Utilizando el LBP combinado con histogramas podemos representar las imágenes de rostros con un simple vector de datos. Como el LBP es un descriptor visual también puede ser utilizado para tareas de reconocimiento facial como se mencionará posteriormente.

2.3.1.5.1. Pseudocódigo

1. **Parámetros**. El $LBPH$ usa 4 parámetros característicos:

   - Radio: El radio se utiliza para construir el patrón binario local circular y representar el radio alrededor del píxel central. Normalmente se establece en 1.
   - *Neighbors* (Vecinos): Es el número de puntos de muestra para construir el patrón binario local circular. Se debe tener en cuenta que cuantos más puntos de muestra incluya, mayor será el coste computacional. Normalmente se establece en 8.
   - Grid X: Es el número de celdas en la dirección horizontal. Cuantas más celdas, más fina será la cuadrícula, y mayor será la dimensionalidad del vector de características resultante. Suele fijarse en 8.
   - Grid Y: Es el número de celdas en la dirección vertical. Cuantas más celdas, más fina es la cuadrícula, mayor es la dimensionalidad del vector de características resultante. Normalmente se fija en 8.

2. **Entrenamiento del algoritmo**. En primer lugar, se tiene que entrenar el algoritmo. Para ello, se debe utilizar un conjunto de datos con las imágenes de rostros de las personas que queremos reconocer. También se requiere establecer un ID o etiqueta (puede ser un número o el nombre de una persona) para cada imagen, así el algoritmo

utilizará esta información para reconocer una imagen de entrada y darle una salida. Las imágenes de la misma persona deben tener la misma etiqueta. Con el conjunto de datos de entrenamiento ya construido.

3. **Aplicación de la operación LBP**. El primer paso computacional del $LBPH$ es crear una imagen intermedia que describa mejor la imagen original, resaltando las características del rostro. Para ello, el algoritmo utiliza el concepto de ventana deslizante, basado en los parámetros de radio y *neighbors*.

   La imagen de la figura 2.3 muestra el procedimiento descrito anterioremeten:

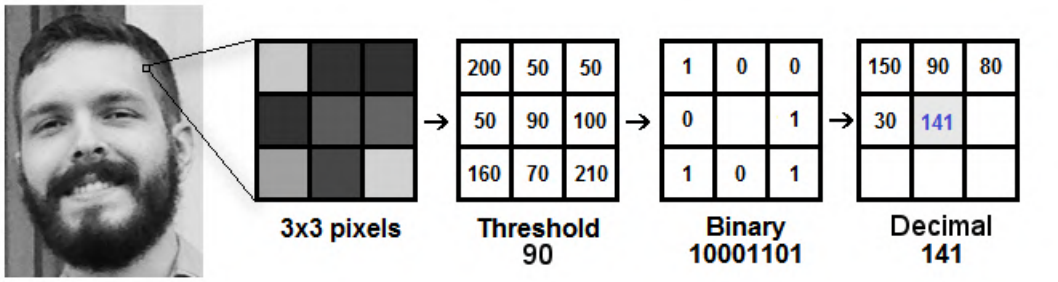


Figura 2.3: Aplicación del operador LBPH.

Tomado de: https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

De acuerdo a la figura 2.3, se explicara en varios pasos para lograr un análisis mas detallado:

- Tenemos una imagen que contiene un rostro representado en escala de grises.
- Se puede obtener una parte de esta imagen como un fragmento de un tamaño de $3x3$ píxeles.
- También se puede representar como una matriz de 3x3 que contiene la intensidad de cada píxel en un el rango (0 255).
- Se tiene que tomar el valor central de la matriz para utilizarlo como umbral.
- Este valor se utilizará para definir los nuevos valores de los 8 *neighbors* (vecinos) del píxel en cuestión.

- Para cada *neighbor* del valor central (umbral), se establece un nuevo valor binario, 1 para los valores iguales o superiores al umbral y 0 para los valores inferiores al umbral.
- Ahora, la matriz contendrá sólo valores binarios (ignorando el valor central). Tenemos que concatenar cada valor binario de cada posición de la matriz línea por línea en un nuevo valor binario (por ejemplo, 10001101). Nota: algunos autores utilizan otros enfoques para concatenar los valores binarios (por ejemplo, en el sentido de las agujas del reloj), pero el resultado final será el mismo.
- A continuación, se convierte este valor binario en un valor decimal y se fija en el valor central de la matriz, que en realidad es un píxel de la imagen original.
- Al final de este procedimiento (procedimiento LBP), tenemos una nueva imagen que representa mejor las características de la imagen original.

El procedimiento LBP se amplió para utilizar un número diferente de radios y *neighbors* (píxeles vecinos), a esto se le conoce con el nombre de LBP *circular* vease la figura 2.4.

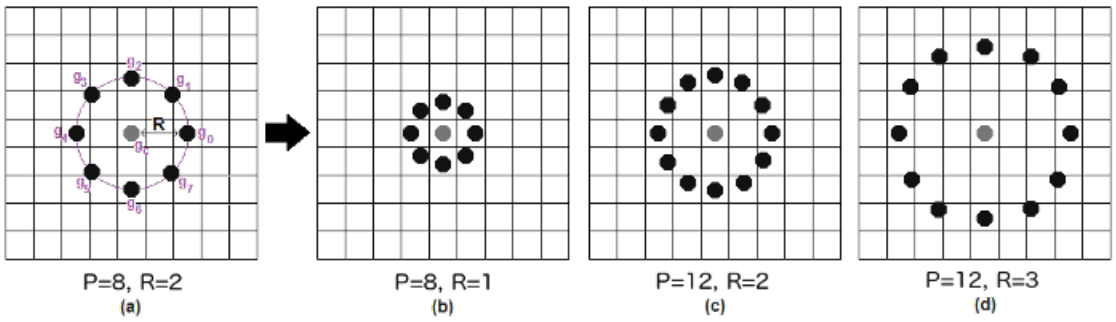


Figura 2.4: LBPH con diferentes valores de radio.

Tomado de: https://updatedcode.wordpress.com/2017/11/26/reconhecimento-facial-como-funciona-o-lbph/

Esto se puede lograr haciendo uso de la interpolación bilineal. Si algún punto de datos está entre las posiciones no entera de los píxeles, utiliza los valores de los 4 píxeles más cercanos (2x2) para estimar el valor del nuevo punto de datos.

4. **Extracción de los histogramas**.Ahora, usando la imagen generada en el último paso, se pueden usar los parámetros *Grid X* y *Grid Y* para dividir la imagen en múltiples cuadrículas, como se visualiza en la siguiente figura.

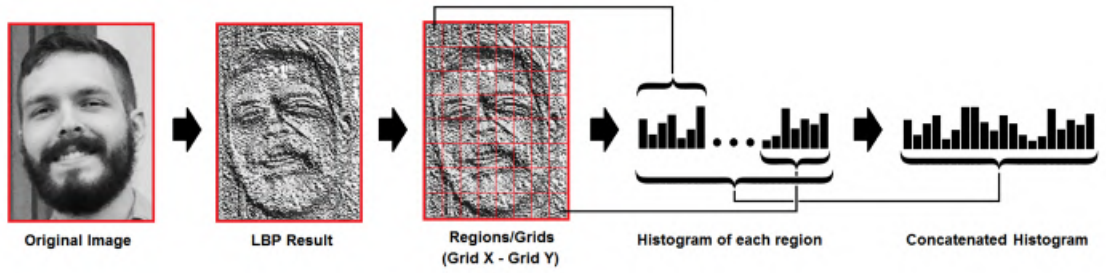


Figura 2.5: Pasos para extraer el Histograma de la imagen LBP.

Tomado de: https://updatedcode.wordpress.com/2017/11/26/reconhecimento-facial-como-funciona-o-lbph/

Se extrae el histograma de cada región de cuadriculas de la siguiente manera:

- Como se obtiene una imagen en escala de grises, cada histograma (de cada cuadrícula) contara con sólo 256 valores (0-255) que representan las ocurrencias de la intensidad de cada píxel.
- Entonces así, se necesita concatenar cada histograma para crear un nuevo y más grande histograma. Suponiendo que tenemos cuadrículas de 8x8, tendremos $8x8x256 = 16384$ posiciones en el histograma final. El histograma final representa las características de la imagen original.

2.3.1.5.2. Realizando el reconocimiento facial.

En este paso, el algoritmo ya debería de estar entrenado. Cada histograma creado se utiliza para representar cada imagen del conjunto de datos de entrenamiento. Así, dada una imagen de entrada, volvemos a realizar los pasos para esta nueva imagen y creamos un histograma que representa la imagen. Así que para encontrar la imagen que coincida con la imagen de entrada sólo tenemos que comparar dos histogramas y devolver la imagen con el histograma más cercano. Podemos utilizar varios enfoques para comparar los

histogramas (calcular la distancia entre dos histogramas), por ejemplo: distancia euclidiana, Chi-cuadrado, valor absoluto, etc. En este caso se describe la forma de comparar dichos histogramas mediante la distancia euclidiana.

$$D = \sqrt{\sum_{i=1}^{n}(hist1_i - hist2_i)^2} \tag{2.8}$$

Así, la salida del algoritmo es el ID o la etiqueta de la imagen con el histograma más cercano. El algoritmo también debe devolver la distancia calculada, esta se puede utilizar como medida de 'confianza'.

Podemos entonces utilizar un umbral y la 'confianza' para estimar automáticamente si el algoritmo ha reconocido correctamente la imagen. Podemos suponer que el algoritmo ha reconocido correctamente la imagen de un rostro si la confianza es inferior al umbral definido.(Prado, 2018)

2.3.2. Mediapipe Face Detection.

Para poder realizar un reconocimiento de rostros correctamente con LBPH y Eigenface es necesario que la imagen entregada a estos clasificadores sea netamente una superficial facial para ellos es necesario tomar de la imagen solamente la región de interés estos lo podemos realizar con $MediaPipe\ Face\ Detection$ la cual es una solución de detección de rostros ultrarrápida que incluye 6 puntos de referencia (ojos, lóbulos de las orejas, nariz y boca) como se muestra en la figura 2.6 y soporte para múltiples rostros. Se basa en $BlazeFace$ (Bazarevsky y col., 2019), un detector de rostros ligero y de buen rendimiento adaptado a $GPU's$ móviles. El rendimiento en tiempo superreal del detector permite aplicarlo a cualquier experiencia de visión por computadora que requiera una región facial de interés precisa como entrada para otros modelos de tareas específicas, como la estimación de puntos clave o geometría facial en 3D (por ejemplo, $MediaPipe\ Face\ Mesh$), la clasificación de rasgos faciales o expresiones, y la segmentación de regiones faciales. $BlazeFace$ utiliza una red ligera de extracción de características inspirada en $MobileNetV1/V2$, pero distinta de ella.

Tomado de («Home», s.f.)

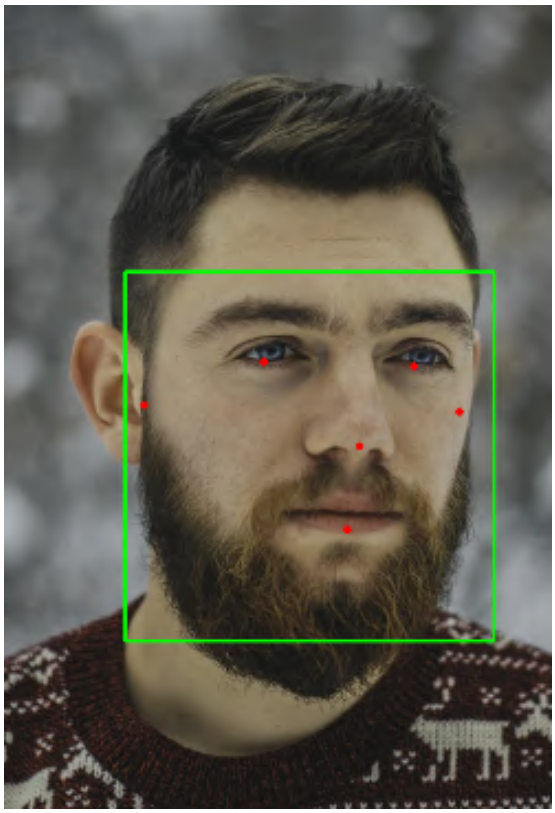


Figura 2.6: Detección del rostro y puntos claves usando Mediapipe Face detection.

Tomado de: https://towardsdatascience.com/write-a-few-lines-of-code-and-detect-faces-draw-landmarks-from-complex-images-mediapipe-932f07566d11

2.3.3. Redes Neuronales

Las *redes neuronales* son modelos simples basados en el funcionamiento del sistema nervioso, son constituidas por su unidades mas básica las cuales son las neuronas, que generalmente se organizan en capas, tal y como se muestra en la figura 2.7.

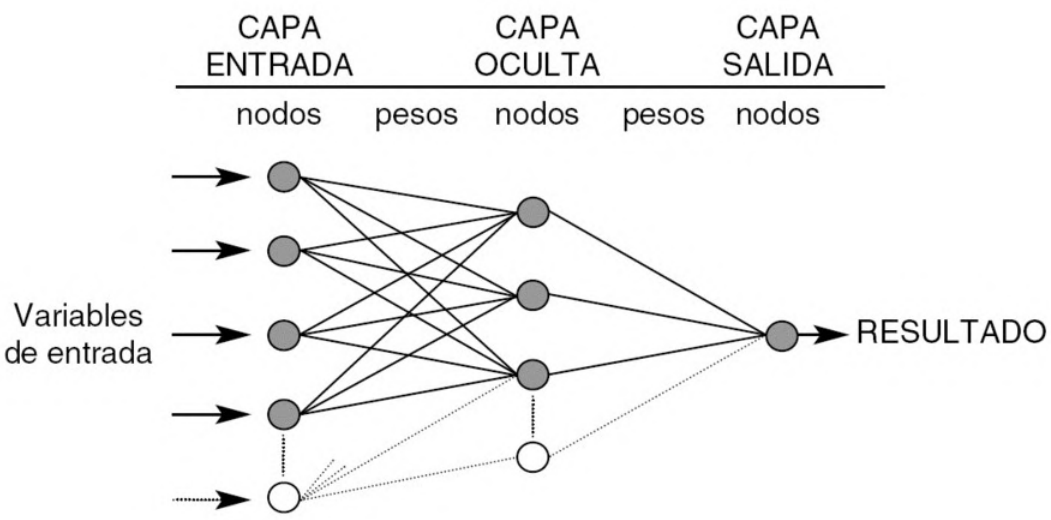


Figura 2.7: Estructura de una red neuronal artificial

Tomado de: https://www.medintensiva.org/en-redes-neuronales-artificiales-medicina-intensiva–articulo-13071859

Una red neuronal es un modelo simplificado de cómo el cerebro humano procesa la información: funciona conectando simultáneamente una gran cantidad de unidades de procesamiento que parecen versiones abstractas de neuronas.

Las unidades de procesamiento se organizan en capas. Una red neuronal normalmente consta de tres partes: una capa de entrada, con unidades que representan campos de entrada, una o más capas ocultas y una capa de salida, con una o más unidades que representan campos de destino. Las unidades están conectadas con fuerza de conexión variable (o peso). Los datos de entrada se presentan en la primera capa y los valores se propagan de cada neurona a cada neurona en la siguiente capa. Finalmente, se envía un resultado desde la capa de salida.

La red aprende examinando registros individuales, generando predicciones para cada registro y ajustando los pesos cuando se hacen predicciones incorrectas. Este proceso se repite muchas veces y la red continúa mejorando sus predicciones hasta que se cumplen uno o más criterios de parada.

Inicialmente, todos los pesos son aleatorios y las respuestas generadas por la red pueden ser un sin sentido, la red aprende entrenando. Los ejemplos de resultados conocidos se presentan continuamente a la red y las respuestas que proporciona se comparan con los resultados conocidos. La información de esta comparación se devuelve a través de la red, cambiando gradualmente los pesos. A medida que avanza el entrenamiento, la red se vuelve cada vez más precisa para replicar los resultados conocidos. Una vez entrenada, la red se puede aplicar a casos futuros en los que se desconoce el resultado. («El modelo de redes neuronales», 2021)

2.3.4. Redes neuronales convolucionales.

Una red neuronal convolucional es un algoritmo de *Deep learning* diseñado para procesar imágenes, tomándolas como entrada, asignando importancia (peso) a ciertos elementos de la imagen para distinguirlos de otros. Este es uno de los principales algoritmos que contribuye al desarrollo y mejora del campo de la visión artificial.

Las redes convolucionales contienen varias capas ocultas, la primera de las cuales puede detectar líneas, curvas, etc... por lo que se especializan hasta que pueden reconocer formas complejas como caras, contornos, etc. Las tareas comunes para dichas redes son: detección o clasificación de objetos, clasificación de escenas y clasificación de imágenes.

La red toma como entrada los píxeles de una imagen, por ejemplo, si tenemos una imagen de 224×224 píxeles de alto y ancho como las usadas en el presente proyecto, eso equivale a 50176 neuronas, eso es, si solo tenemos un color (imagen en escala de grises). Si tuviéramos una imagen a color, necesitaríamos 3 canales (R, G, B), por lo que usaríamos 224x224x3 = 150528 neuronas de entrada. Es de suma importancia normalizar los datos, es decir, el valor del píxel de la imagen suele estar entre 0 y 255, y debería estar entre 0 y 1. dicho proceso de normalización podemos lograrlo dividiendo cada uno de los píxeles entre el valor más alto que estos tienen, es decir 255.

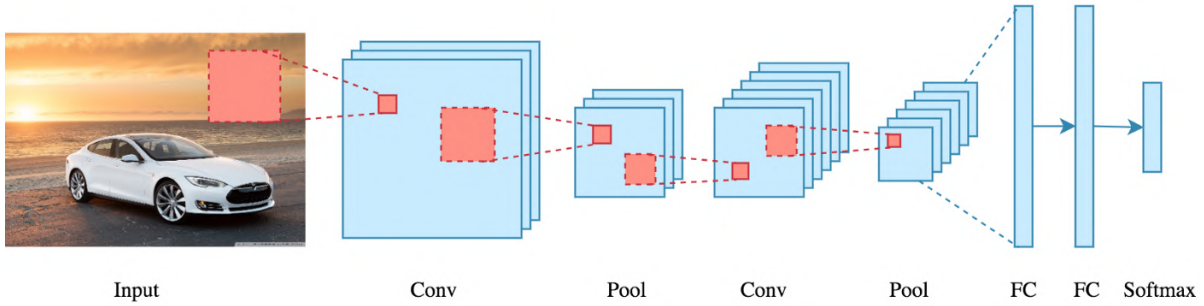


Figura 2.8: Capas de convolución de una Red neuronal convolucional.

Tomado de: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

2.3.4.1. Kernel

El *kernel* en las redes convolucionales se considera como el filtro que se aplica a una imagen para sustraer ciertas propiedades relevantes o patrones de esta. Dentro de las propiedades relevantes para la cual sirven los *kernels* son; identificar bordes, enfoque, desenfoque, entre otros. Esto se consigue al hacer la convolución entre la imagen y el kernel.

2.3.4.2. Convolución

Uno de los procesos más distintivos de estas redes neuronales son las convoluciones. La cual se basa en tomar un conjunto de píxeles de la imagen de acceso e ir llevando a cabo un producto escalar con un *kernel*. El *kernel* recorrerá cada una de las neuronas de acceso y obtendremos una matriz totalmente nueva, la cual va a ser una de las capas escondidas de la red. En el debido caso de que la imagen sea a color se van a tener tres *kernels* del mismo tamaño que se sumarán para obtener una imagen de salida.

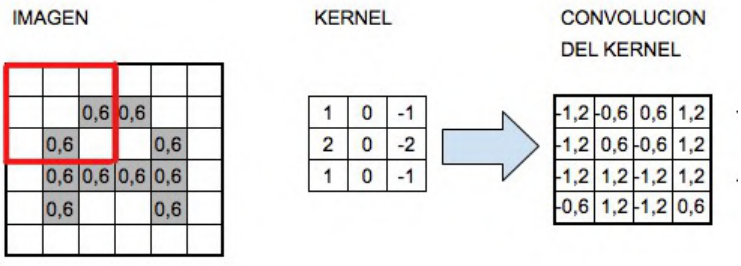


Figura 2.9: Proceso de convolución con un *kernel*.

Tomado de: https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/

2.3.4.3. *Padding*

Es una operación que se usa en las redes neuronales convolucionales. Este se aplica agregando píxeles de valor cero alrededor de la imagen original. Tiene dos usos: El primero es para que al realizar la convolución la imagen resultante sea de igual tamaño que la imagen original. El segundo es cuando se tiene información relevante en las esquinas de la imagen por lo que al realizar la convolución el filtro pasa más por el centro de la imagen que en las esquinas, en este caso que se aplica el *Padding* para tener la información más relevante cerca del centro. (Ai, 2021)

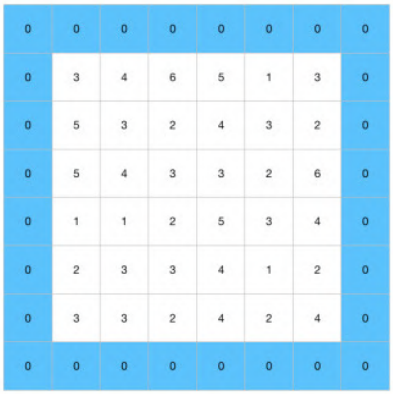


Figura 2.10: *Padding* en una imagen.

Tomado de: https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8

2.3.4.4. Estructura de una red neuronal convolucional.

En general, las redes neuronales convolucionales están construidas con una estructura que contiene 3 tipos distintos de capas:

1. Una capa convolucional, que es la que le da el nombre a la red.
2. Una capa de reducción o de *pooling*, la cual va a reducir la cantidad de parámetros al quedarse con las características más comunes.
3. Una capa clasificadora totalmente conectada, la cual nos va dar el resultado final de la red.

2.3.4.4.1. Capa convolucional.

Lo que distingue a las redes neuronales convolucionales de cualquier otra red neuronal es que usan una operación llamada convolución en alguna de sus capas en vez de usar la multiplicación de matrices que se aplica originalmente. La operación de convolución obtiene como ingreso o *input* la imagen y después aplica sobre ella un filtro o *kernel* que nos regresa un mapa de las propiedades de la imagen original, así logramos minimizar el tamaño de parámetros.

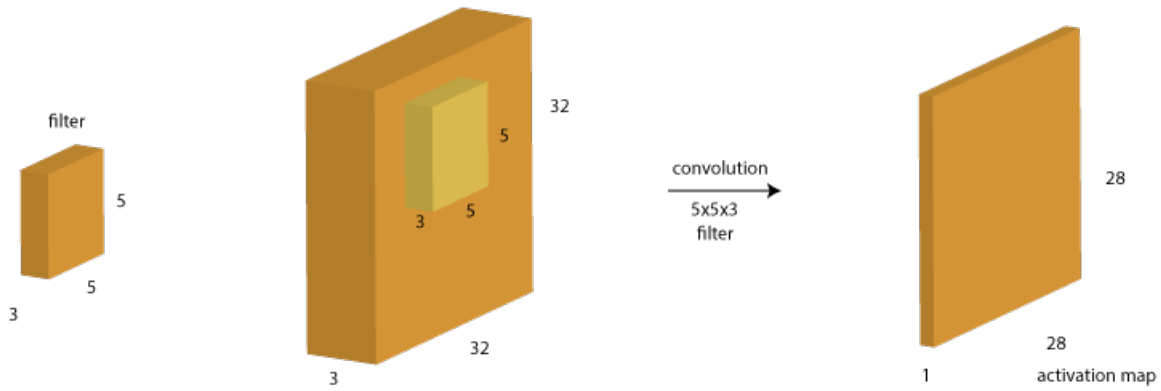


Figura 2.11: Capa de convolución de una CNN.

Tomado de: https://www.analyticsvidhya.com/blog/2016/04/deep-learning-computer-vision-introduction-convolution-neural-networks/

2.3.4.4.2. Capa de *pooling* o reducción.

La capa de reducción o *pooling* se coloca generalmente después de la capa convolucional. Su utilidad principal es la reducción de las dimensiones espaciales (ancho x alto) del volumen de entrada para la siguiente capa convolucional. La reducción no afecta a la dimensión de la profundidad del volumen. La operación realizada por esta capa también se llama reducción de muestreo, ya que la reducción de tamaño conduce también a la pérdida de información. Sin embargo, una pérdida de este tipo puede ser beneficioso para la red por dos razones:

- la disminución en el tamaño conduce a una menor sobrecarga de cálculo para las próximas capas de la red
- También se logra reducir los sobreajustes.

La operación que se suele utilizar en esta capa es *max-pooling*, que divide a la imagen de entrada en un conjunto de rectángulos y, respecto de cada subregión, se va seleccionando el máximo valor de esta, como se ilustra en la siguiente figura.

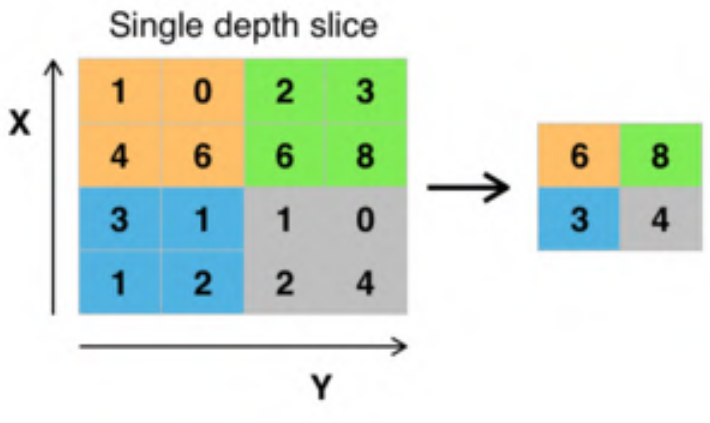


Figura 2.12: Operación de *Max-pooling*

Tomado de: https://towardsdatascience.com/deep-learning-personal-notes-part-1-lesson-3-cnn-theory-convolutional-filters-max-pooling-dbe68114848e

2.3.4.4.3. Capa clasificadora totalmente conectada.

Al final de las capas convolucional y de *pooling*, las redes utilizan generalmente capas completamente conectadas en la cual cada píxel se considera como una neurona separada al igual que en una red neuronal regular. Esta última capa clasificadora tendrá tantas neuronas como el número de clases que se debe predecir. (Briega, 2016)

2.3.4.5. Red *YOLO* versión 2

Los algoritmos de detección $YOLO$ ($You\ Only\ Look\ Once$) son métodos desarrollados recientemente que permiten la detección de objetos con velocidades suficientes como para tener aplicaciones en vídeos en tiempo real.

El método de detección $YOLO$ funciona mediante la subdivisión de una imagen en múltiples cuadros en donde calcula probabilidades de contención de objetos por dichos cuadros mediante la información recolectada de las clases con las que fue entrenado. $YOLO$ se entrena con imágenes completas siendo capaz de optimizar el rendimiento de la detección de las clases en las imágenes por medio de una red de 24 capas convolucionales que reducen los espacios de características de las capas anteriores (Pérez y col., 2019).

En términos de velocidad, la red $YOLO$ es capaz de reconocer imágenes y procesar cuadros a casi 150 frames por segundo, lo cual es sin duda útil para aplicaciones en tiempo real, sin embargo la precisión de las redes entrenadas con este método se quedaron escasas en cuanto a precisión se trataba, llegando a conseguir tan solo un 63 %. Por tal motivo en el 2016 fue lanzada por J. Redmon y A. Farhadi la segunda versión de la red $YOLO$ la cual buscaba mantener las velocidades existentes aparte de realizar mejoras en la detección y la clasificación de objetos proporcionando cambios tales como mejoras en la resolución de las imágenes de entrenamiento, cambios en los cuadros delimitadores, entrenamientos con dimensiones aleatorias, entre otros (Redmon y Farhadi, 2017).

2.4. Fundación Raspberry Pi

La Fundación Raspberry Pi es una organización con sede en Reino Unido cuyo principal objetivo es trabajar para poner el poder de la informática y la creación digital en manos de personas de todo el mundo. Su principal producto es la serie sistemas embebidos de la familia Raspberry Pi, la cual son tarjetas de desarrollo con hardware suficiente como para poder correr un sistema operativo propio proporcionado por la organización. Estas tarjetas se pueden programar en lenguaje Python y Micropython. («Raspberry Pi Foundation - About Us», 2021)

2.4.1. MicroPython

MicroPython es un analizador sintáctico basado en la gramática de Python 3. Contiene la mayor parte de la gramática básica de esta versión del lenguaje. Se ejecuta principalmente en chips embebidos con rendimiento y memoria limitados. Cabe recalcar que Micropython no incluye toda la sintaxis de Python 3.

2.4.2. Tarjetas Raspberry Pi

Existen diferentes modelos de tarjetas Raspberry Pi, unas mas potentes e idóneas para ciertas tareas que otras, entre ellas podemos encontrar: Raspberry Pi Zero, Raspberry Pi 2, Raspberry Pi 3, Raspberry Pi 4. Estas ultimas son actualizaciones de las anterior cuyas mejoras recaen principalmente en el hardware en características como el procesador y la memoria RAM.

La tarjeta que se usó para implementar los algoritmos presentados en el presente proyecto fue la Raspberry Pi 3B +, la cual es es una modificación de la placa original Raspberry Pi 3, el aspecto de la *main board* se observa en la figura 2.13 y las especificaciones en la tabla 2.1

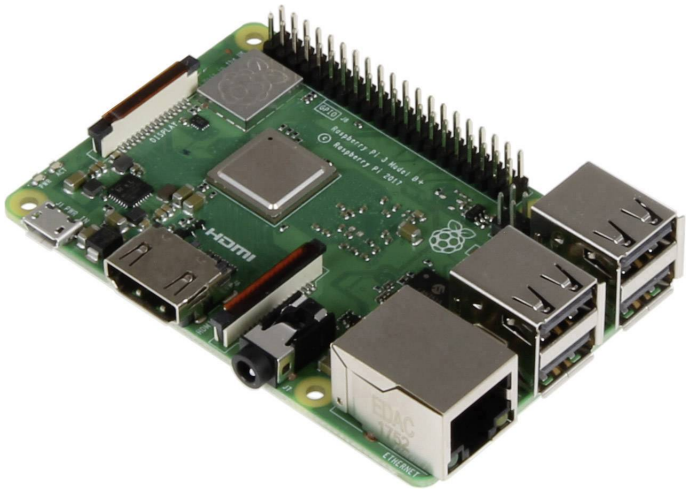


Figura 2.13: Raspberry Pi3B+

Tomado de: https://www.conrad.com/p/raspberry-pi-3-b-1-gb-4-x-14-ghz-raspberry-pi-1668026

Raspberry PI 3 B+	Description
CPU:	Broadcom bcm2837b0 cortex-a53 (armv8) 64-bit soc @ 1.4ghz
Memory:	1GB LPDDR2 SDRAM
Storage:	Support micro expansion storage (max 64GB)
Ports:	GPIO 40 pins HDMI 4 x USB 2.0 CSI (Raspberry Pi Camera, 5 MP) DSI (Touch Screen) Headphone/composite video jack Micro SD Micro USB (Power) Power-over-Ethernet (PoE)

Tabla 2.1: Especificaciones técnicas de la Raspberry Pi 3B +.

Tomado de: https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

2.5. Sipeed Maix

2.5.1. MaixPy

MaixPy es un *port* de Micropython para los chips K210 de Sipeed, el cual es una CPU RISC-V de doble núcleo de 64 bits con FPU (*Floating point unit*) de *hardware*, acelerador de convolución entre otros. Es un proyecto que soporta el funcionamiento normal de un MCU (*Microcontroller unit*) e integra la aceleración por hardware. La visión artificial de la IA, el conjunto de micrófonos, el módulo central de potencia de cálculo de 1 TOPS (*Trillions or Tera Operations per Second*) convierten a este producto una buena herramienta para el desarrollo rápido de aplicaciones inteligentes en el campo de la AIOT (*Artificial intelligence of things*) con un coste extremadamente bajo y un tamaño práctico.

Las tareas que se pueden realizar con MaixPy haciendo uso del chip K210 va desde la detección facial , ejecutar la red *MobileNet*, hacer el calculo de la transformada rápida de Fourier a señales de audio, hasta correr emuladores de videojuegos retros como lo son la NES y el Game Boy Advance («What can MaixPy do», s.f.). Las tareas de detección de rostros son particularmente laboriosas debido a que se debe realizar etiquetado de sectores de las imágenes de forma manual para identificar las regiones de interés y así realizar el entrenamiento de las redes neuronales. Con el fin de facilitar esta tarea, Sipeed pone a disposición de los usuarios herramientas que hacen mas amigable al usuario este etiquetado tales como Vott y LabelImg («Maixhub Model Training Platform Instructions for Use», s.f.).

2.5.2. LabelImg

Es un aplicativo que posee una interfaz gráfica interactiva que permite mediante el uso del *mouse* de un computador dibujar el área de interés dentro de una imagen y etiquetar con un nombre a que clase pertenece dicha imagen en conjunto al área seleccionada. Lo anterior con el fin de obtener un modelo entrenado mas preciso al momento de la ejecución del mismo. El etiquetado de las regiones de interés con esta herramienta permite genera

archivos xml en los que se pueden encontrar los datos necesarios para entrenar de forma correcta un modelo en la nube.

Lo antes descrito se puede observar en la figura 2.14 y en el *script* de la lista 1 , donde se visualiza la interfaz de *LabelImg*, la selección de la región de interés dentro de una imagen, su respectivo etiquetado y el contenido de los archivos .xml.

Aspectos característicos del contenido del archivo .xml generado por *labelimg* son: la ruta y el archivo imagen al que están ligados los datos, así como las coordenadas de la imagen de la región de interés y su respectiva etiqueta

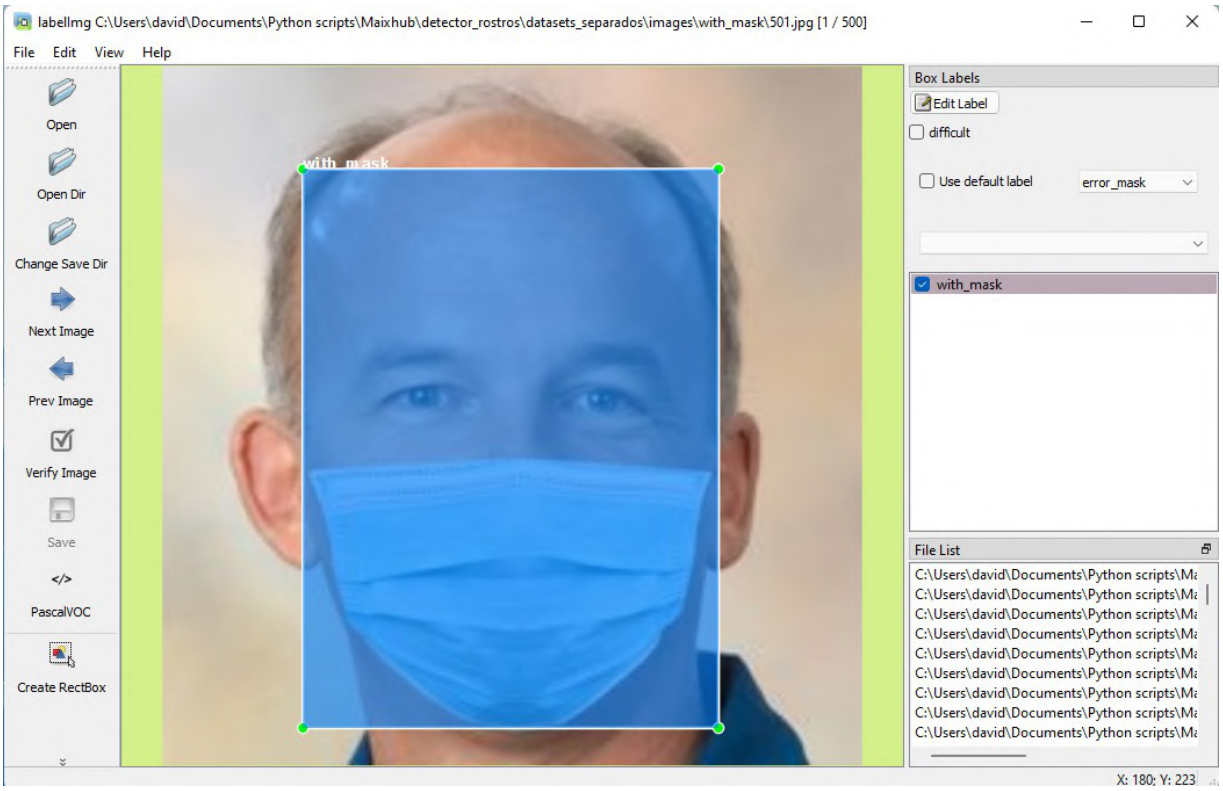


Figura 2.14: Interfaz y etiquetado de la ROI en imágenes del *dataset*

```
<annotation verified="yes">
        <folder>with_mask</folder>
        <filename>501.jpg</filename>
        <path>C:\Users\david\Documents\Python scripts\Maixhub\detector_rostros
                        \datasets_separados\images\with_mask\501.jpg</path>
        <source>
                <database>Unknown</database>
        </source>
        <size>
                <width>224</width>
                <height>224</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>with_mask</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>45</xmin>
                        <ymin>33</ymin>
                        <xmax>178</xmax>
                        <ymax>212</ymax>
                </bndbox>
        </object>
</annotation>
```

Listing 1: Archivo .xml generador por LabelImg

2.5.3. Tarjetas Maix

Existen diversas tarjetas dentro de la familia Maix que cuentan con el chip K210, entre ellas están: Maix Dock, Maix Bit, Maix Amigo, Maix Duino, Maix Cube, Maix Go y Maix nano.

La tarjeta que se uso principalmente en el presente proyecto fue la Maix Go, cuyas especificaciones las podemos encontrar en la tabla 2.2 y el aspecto de la main board es el mostrado en la figura 2.15. («Maix Go - Sipeed Wiki», s.f.)

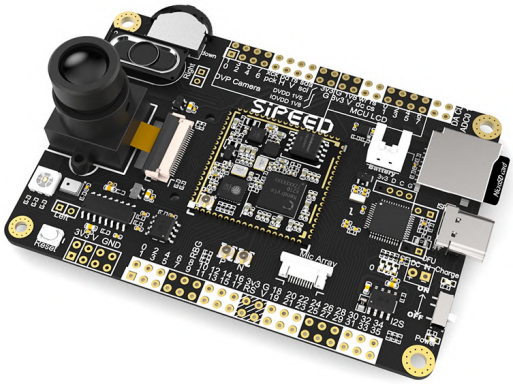


Figura 2.15: Maix Go

Tomado de: https://wiki.sipeed.com/soft/maixpy/en/develop_kit_board/maix_go.html

Maix Go	Description
CPU:	**Dual-core 64 bits RISC-V / 400MHz (double-precision FPU integration)**
Memory:	**8 MiB 64bit on-chip SRAM**
Storage:	**16 MiB Flash, support micro SDXC expansion storage (max 128GB)**
Screen:	**2.4 inch TFT, capacitive touch screen resolution: 320*240**
Camera (package):	**200W pixels (actual use 30W), 0V2640 model M12 camera**
TF card slot:	**Multimedia resource expansion, support large-capacity storage**

Tabla 2.2: Especificaciones técnicas de la Maix Go.

Tomado de: https://wiki.sipeed.com/soft/maixpy/en/develop_kit_board/maix_go.html

3. Metodología

3.1. Modelos con Machine Learning

Para llevar a cabo la realización de este proyecto se han planteado varias soluciones que conllevan el uso de técnicas de inteligencia artificial pertenecientes al campo de *Machine learning* y modelos asociados a este mismo proporcionados por la librería de *OpenCV* cuyos modelos de clasificación de imágenes utilizados son *Eigenface* y LBPH (*local binary patern histograms*), uno con sus ventajas y desventajas respecto al otro.

Para ilustrar la parte de procesamiento de imágenes y la clasificación, en la siguiente figura se representa de forma general los pasos que realiza el algoritmo desarrollado.

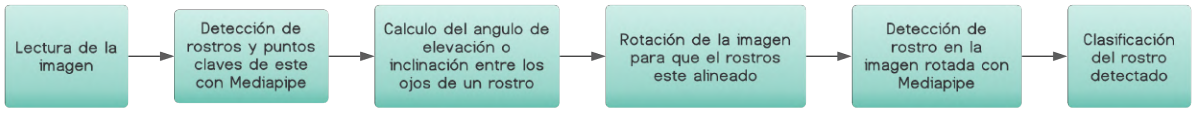


Figura 3.1: Procesamiento y clasificación del algoritmo.

3.1.1. Preprocesado de las imágenes

Como podemos observar en el flujo grama anterior es necesario realizar un procesamiento de imágenes para que los modelos de *Machine learning* puedan hacer una clasificación correcta, teniendo esto en cuenta, lo primero es detectar el rostro en la imagen mediante *Mediapipe Face Detection*, este identifica varios puntos claves en el rostro como lo son los ojos, los lóbulos de las orejas, la nariz, la boca, el mentón entre otros. Para este primer procesamiento se utiliza la ubicación de los ojos y mediante el trazado de un triangulo rectángulo entre ellos se calcula el ángulo de inclinación o elevación entre estos últimos con el fin de poder rotar la imagen para conseguir que la altura entre ellos siempre sea la misma, el objetivo de esta practica es volver robusto el reconocimiento de rostros y lograr mas eficiencia en el clasificado, pues en ultimas se necesitarían menos imágenes que contengan caras con distintas facetas y posiciones en el conjunto de datos de entrenamiento.

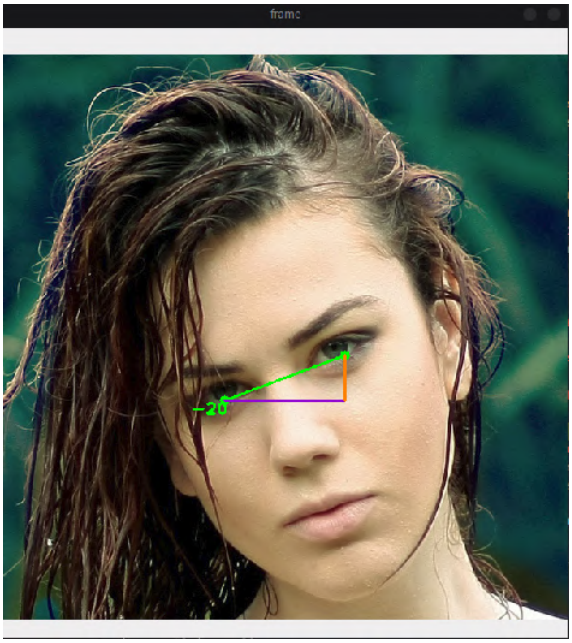


Figura 3.2: Visualización del ángulo de elevación o inclinación entre los ojos de un rostro.

Una vez rotada la imagen respecto al rostro (Rosebrock, 2021) se vuelve a aplicar el reconocimiento de caras de *mediapipe* para poder recortar la región de interese dentro de la imagen rotada.

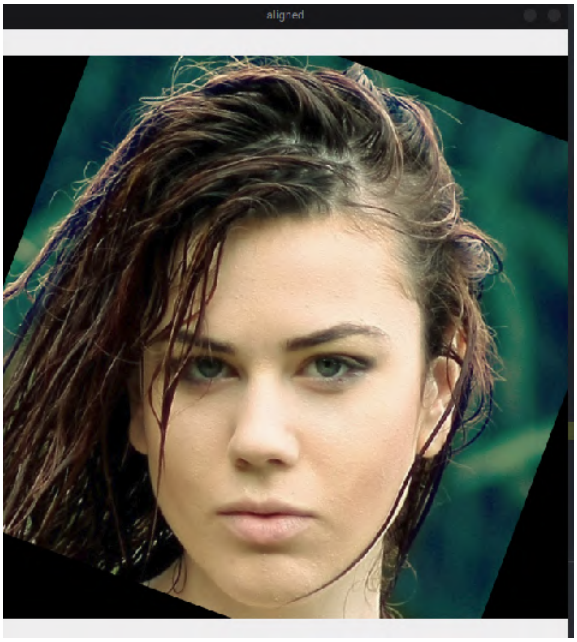


Figura 3.3: Imagen alineada respecto a los ojos de un rostro.

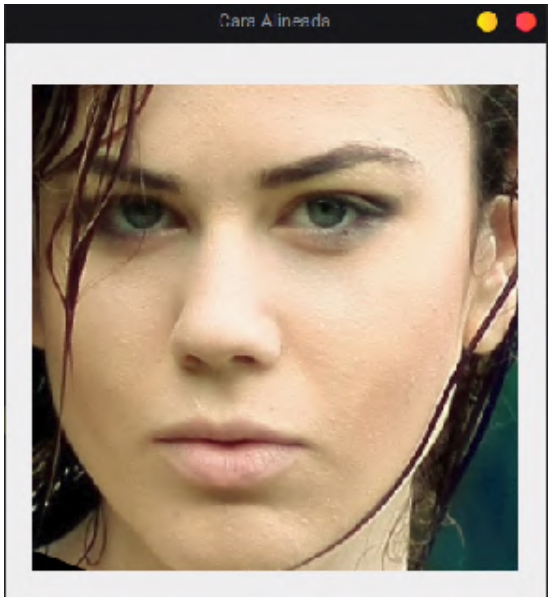


Figura 3.4: Recorte del rostro alineado como entrada al clasificador.

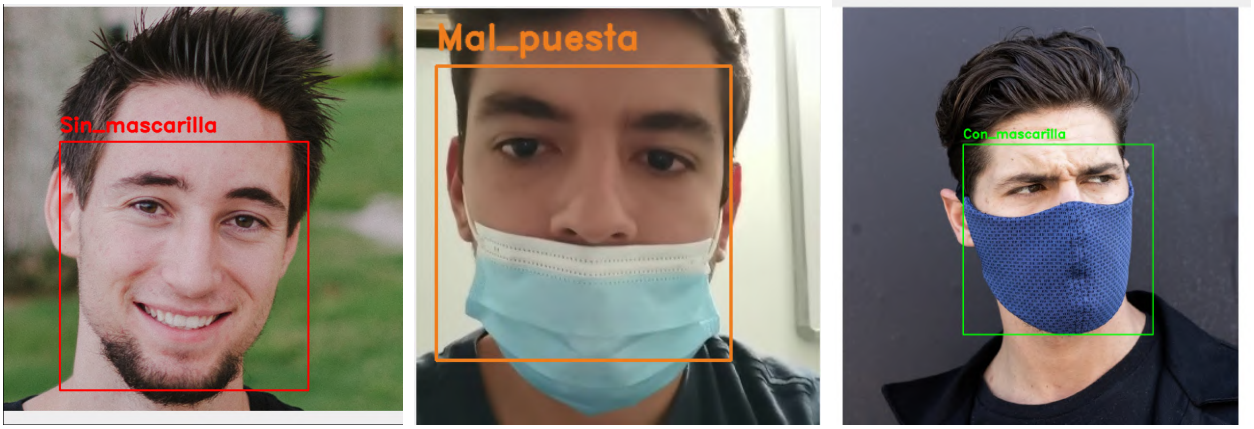
3.1.2. Entrenamiento de los modelos

Como se ha mencionado anteriormente para esta solución se utilizaron 2 modelos proporcionados por la librería de *OpenCV*, *Eigenface* y LBPH (SuperDataScience Team, 2017), para el *script* de entrenamiento, en primera instancia se lee de forma iterativa las imágenes pertenecientes a 3 carpetas distintas, con mascarilla, Error mascarilla y Sin mascarilla, a medida que se van leyendo las imágenes se van agregando en una lista de forma ordenada , en otra lista se va agregando las etiquetas alusivas al clasificado, en este caso se usaron los números 0, 1 y 2, 0 para Con mascarilla, 1 para error mascarilla y 2 para sin mascarilla. Cabe resaltar que dichas imágenes a medida que son leídas son escaladas a una resolución de 72x72 píxeles para aliviar el pesos final del modelo y transformadas a escala de grises debido a que dichos modelos clasifican imágenes en esta representación de color. Por ultimo se pasa la lista que contiene las imágenes y las etiquetas a la función de entrenamiento y se guarda el modelo en un formato .xml.

El código que describe la anterior secuencia de pasos se puede encontrar en el repositorio de GitHub (Torres y Sánchez, 2022) , siguiendo la ruta /Machine learning codes /train_full.py.

3.1.3. Código del algoritmo

Hay varias formas de utilizar la aplicación del código, la mas básica seria a través de un vídeo pre-grabado o mediante *video streaming* con la cámara del dispositivo que este ejecutando el algoritmo, la otra seria mediante una serie de imágenes que contengan rostros de personas con, sin o tapabocas mal puestos. Para poder conformar el código que se evaluó, se unió el método de preprocesado de las imágenes que se describió en sub-secciones anteriores cuya esencia es identificar los rostros y de ser necesario rotarlos para que el ángulo de elevación o inclinación entre los ojos sea cero, con la clasificación de los modelos ya entrenados, dependiendo de la etiqueta arrojada por el clasificador se encierra el rostro de la persona en un cuadrado de color verde si porta tapabocas, anaranjado si lo lleva mal puesto y rojo en caso de que no haga uso de este, la respuesta del algoritmo se ve tal y como se muestra en las figuras 3.5a, 3.5b y 3.5b.



(a) Clasificación de: Sin mascarilla

(b) Clasificación de: Mascarilla mal puesta

(c) Clasificación de: Con mascarilla

Figura 3.5: Detección con Raspberry pi 3B+.

Con respecto al algoritmo se puede encontrar en el repositorio de GitHub (Torres y Sánchez, 2022), en la ruta /Machine learning codes/test_face_mask_face_aligment_full.py.

3.2. Modelos de redes neuronales

Debido a la falta de fluidez y hardware presentada por las soluciones propuestas para la Raspberry, se migró de plataforma a las tarjetas Maix que ofrece la compañía Sipeed para poder conseguir el objetivo del presente proyecto en esta familia de dispositivos llamados sistemas embebidos.

Dentro de la documentación de la familia de tarjetas Maix, existen dos tipos de algoritmos de IA; detección de objetos y clasificación de imágenes. Con el fin de hacer el reconocimiento de mascarillas, la opción que se tomo en cuenta fue la de detección de objetos, se hizo pasar las tres clases previamente mencionadas (sin, con y mascarilla mal puesta) como si fueran objetos a detectar por parte del algoritmo. Como se profundizará en las subsecciones posteriores, la plataforma de Sipeed ofrece entrenamiento de modelos con redes neuronales en la nube, tan solo se debe de tener la base de datos de imágenes organizada seleccionando una de las jerarquía que ellos dictaminan en su documentación. Dentro de esta jerarquía deben ir las imágenes de los objetos en conjunto a los datos con la región de interés y etiqueta referente a la imagen en cuestión, guardados en un archivo .xml escrito en lenguaje HTML. Para la obtención de dichos datos se uso él aplicativo ya desarrollado en Python llamado *LabelImg* del que se habló en la sección 2.5.2.

3.2.1. Entrenamiento en Maixhub

Para el entrenamiento en la nube basta con entrar en la pagina web de Maixhub e ingresar con una cuenta registrada previamente, se encontrara una interfaz similar a la mostrada en la figura 3.6.

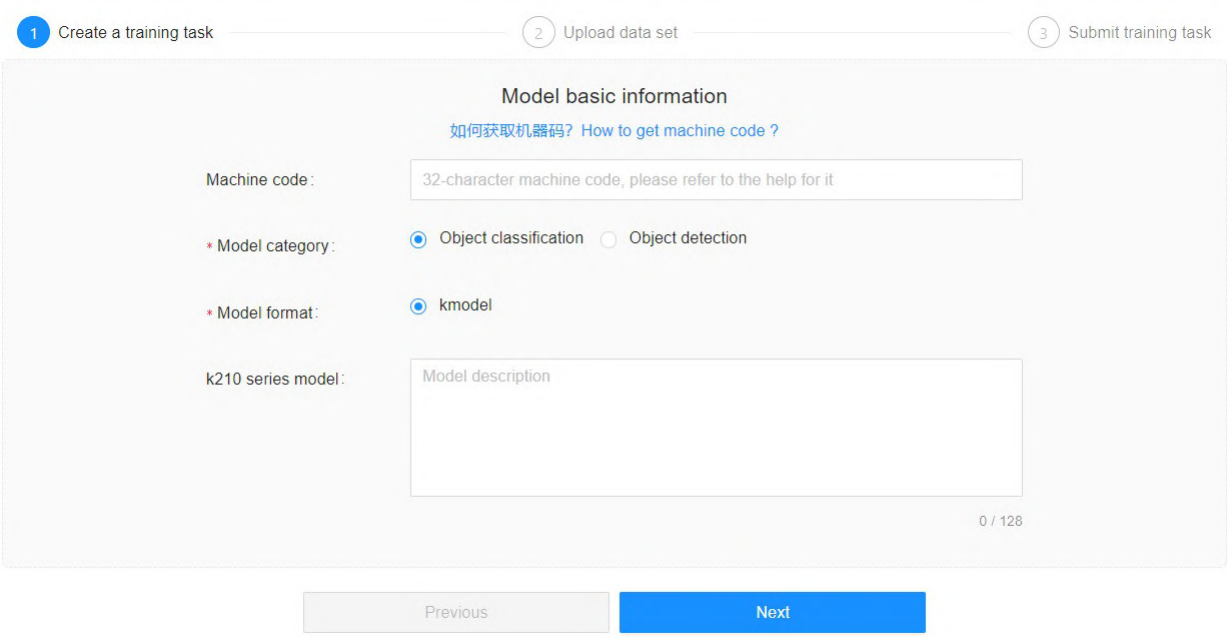


Figura 3.6: Interfaz de entrenamientos de modelos en la nube con $MaixHub$

Uno de los requerimientos para usar este servicio es contar con un serial valido único de cada tarjeta proporcionada por Sipeed ya sea Maix Go , Maix Bit etc . . . , seleccionar la categoría del modelo y el formato del modelo que sera .kmodel, sumado a esto añadir una breve descripción de lo que hará el modelo.

Al momento de subir la base de datos se debe seguir una de las jerarquías de archivos que admite $MaixHub$, la estructura jerárquica que se usó en el presente proyecto es la presentada en la figura 3.7, la cual fue basada en la que se encuentra en la documentación de la plataforma. Por último lo que se debe subir es un archivo en formato .zip (Torres y Sanchez, 2022b) de no mas de 20 MB ($MegaBytes$).

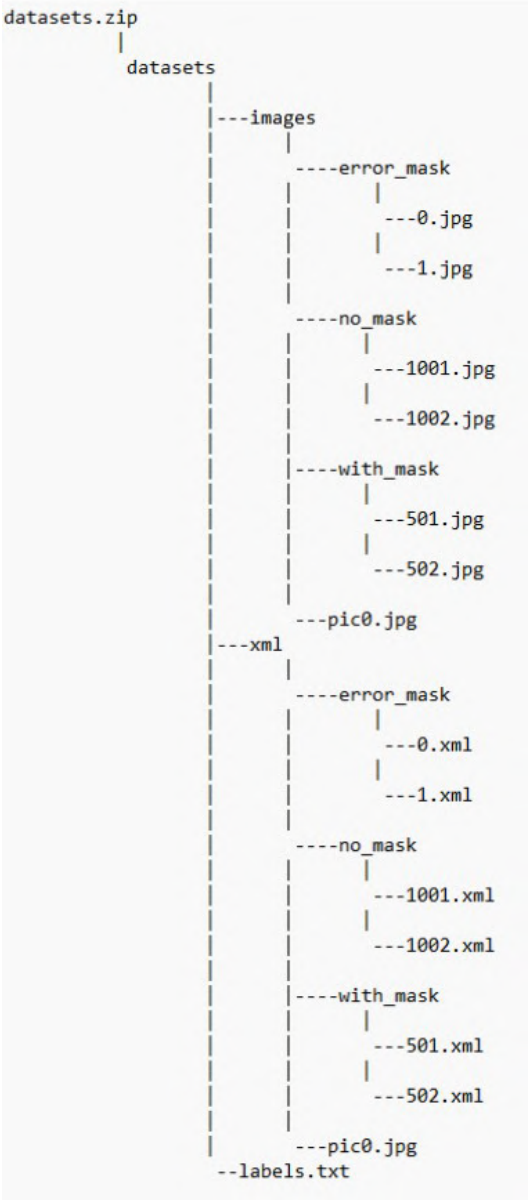


Figura 3.7: Jerarquía de archivos

Tomado de: https://www.maixhub.com/ModelTrainingHelp_en.html

3.2.2. Código del algoritmo.

Una vez subido el *dataset* organizado según la jerarquía y en un archivo .zip comenzará el entrenamiento en la nube, podremos ver el progreso de este en el apartado de información de la cuenta, el tiempo de entrenamiento varía dependiendo del tamaño del *dataset*, en el caso del presente proyecto demoró alrededor de unos cincuenta minutos.

Una vez finalizado el proceso podremos descargar un archivo .zip, dentro de su contenido, lo mas relevante es un archivo de texto que contiene advertencias que se presentan durante el entrenamiento, el modelo en formato .kmodel, una imagen que contiene las gráficas de las funciones de perdida durante el entrenamiento y la validación, y por último un *script* en MaixPy para ejecutar el detector de objetos.

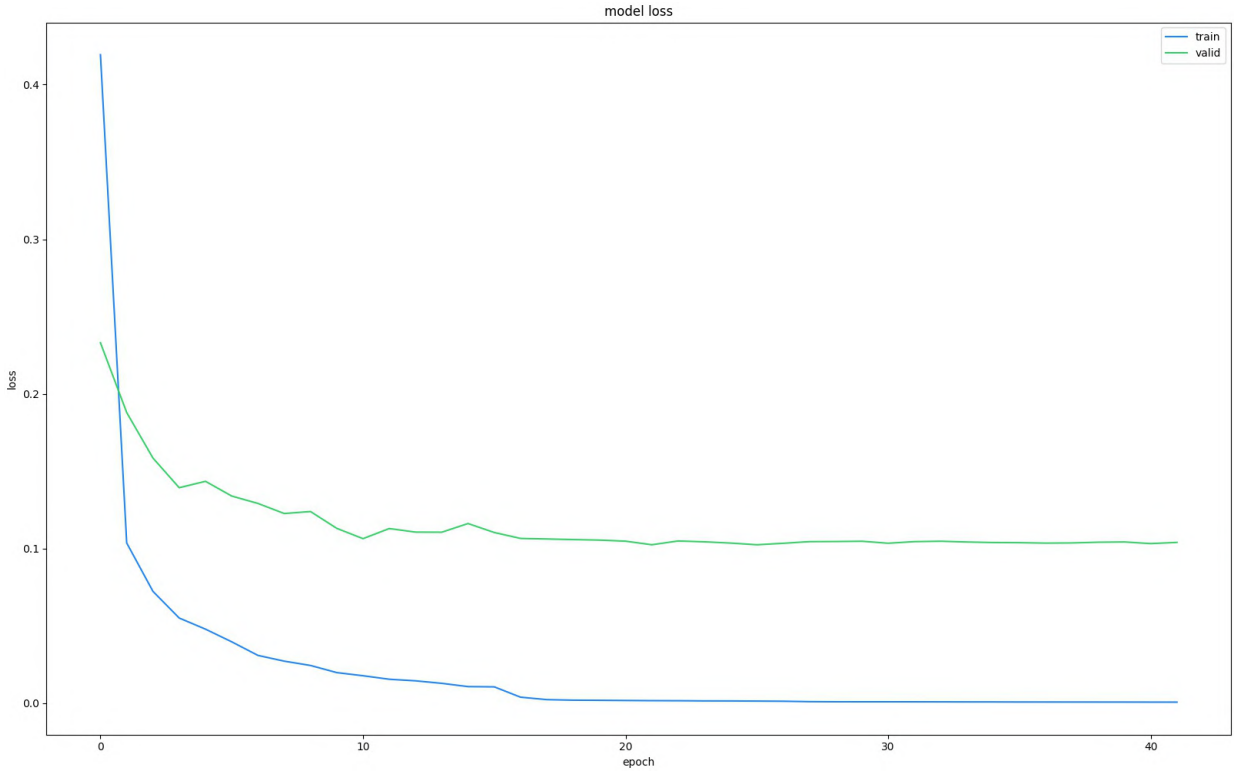
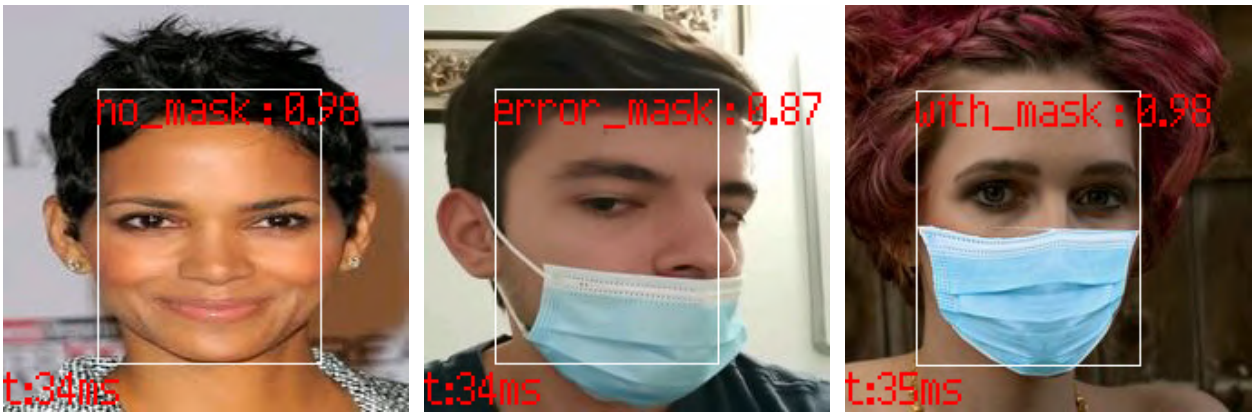


Figura 3.8: Función de perdida en el entrenamiento y validación.

Para observar el código que ejecuta la detección, este se encuentra en el repositorio del proyecto (Torres y Sánchez, 2022) dentro de la carpeta Sipeed Maix, el archivo llamado “boot.py”

En las figuras 3.9a, 3.9b y 3.9c podemos observar como se ve el reconocimiento facial de mascarillas en la pantalla TFT de la Maix Go, dentro de ella podemos observar un recuadro encerrando el rostro dentro de la imagen, la etiqueta , el valor de confiabilidad que no es mas que; que tan segura esta la red de que la etiqueta que arrojó es verídica y el tiempo de respuesta que usualmente ronda los 34-35 mS



(a) Clasificación de: Sin mascarilla

(b) Clasificación de: Mascarilla mal puesta

(c) Clasificación de: Con mascarilla

Figura 3.9: Detección con Sipeed Maix Go.

4. Resultados

Para poder evaluar los resultados obtenidos tanto con los modelos que hacen uso de *Machine learning* como de Redes neuronales, se emplean y se comparan diferentes términos y parámetros de rendimiento tales como: la precisión, el error, tiempo de clasificación y el tiempo de ejecución de los diversos algoritmos que hacen uso de diferentes clasificadores entrenados de forma independiente con cierta cantidad en ascenso de imágenes.

4.1. Resultados con Machine learning

Como se había mencionado en el capitulo anterior, los modelos de clasificación usados con *Machine learning* son : $Eigenface$ y *Local Binary Pattern Histogram*. La forma en la que se obtuvieron los distintos parámetros de rendimiento comienza con la organización previa de imágenes, etiquetadas mediante una revisión visual por parte de los estudiantes desarrolladores del proyecto, en la cual se clasificaron 150 imágenes por clase, dando como resultado 450 imágenes examinadas por parte de cada modelo (Torres y Sanchez, 2021a). Es importante mencionar que la cantidad de modelos de $Eigenface$ y LBPH que se evaluaron suman en total 10 modelos diferentes, 5 modelos $Eigenface$ y 5 modelos LBPH. Para cada uno de estos modelos la cantidad de imágenes con las que fueron entrenados van desde las 800 a 1200 (Torres y Sanchez, 2021b) imágenes por clase con incrementos de imágenes de 100 en 100. Se aclara a manera de ejemplo que las imágenes de la base de datos de entrenamiento usadas para obtener un modelo de 1100 imágenes por clase son las mismas tanto para $Eigenface$ como para LBPH. En las siguientes tablas i/o figuras se representa los parámetros de rendimiento para cada clase de clasificación con una base de datos de prueba con la cantidad de imágenes por clase antes mencionada.

4.1.1. Plataforma: Raspberry

Las pruebas en dispositivo embebido se realizaron en una placa Raspberry Pi 3B+ la cual ejecutó los algoritmos en lenguaje Python bajo las siguientes especificaciones:

- Procesador: Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC
- Ram: 1GB LPDDR2 SDRAM
- SO: Raspbian GNU/Linux 10

4.1.1.1. Rendimiento sin mascarilla

Sin mascarilla					
Modelo / Imágenes de entrenamiento por clase	**Precisión [%]**	**Error [%]**	**Tiempo de clasificación [mS]**	**Tiempo de ejecución [mS]**	**Peso del modelo (MB)**
Eigenface 800	**98.63**	**1.36**	**121.95**	**229.14**	**466**
Eigenface 900	**98.63**	**1.36**	**168.67**	**287.77**	**545.4**
Eigenface 1000	**99.32**	**0.68**	**197.54**	**321.77**	**629.6**
Eigenface 1100	**99.32**	**0.68**	**348.92**	**526.04**	**718.5**
Eigenface 1200	**99.32**	**0.68**	**477.88**	**695.21**	**812.2**
LBPH 800	**98.64**	**1.36**	**1969.18**	**2160.63**	**188.5**
LBPH 900	**99.32**	**0.68**	**2149.49**	**2337.63**	**212**
LBPH 1000	**99.32**	**0.68**	**2012.21**	**2163.52**	**235.6**
LBPH 1100	**98.63**	**1.36**	**2291.16**	**2429.95**	**259.3**

Tabla 4.1: Tabla de parámetros de rendimiento en Raspberry para clase de: sin mascarilla

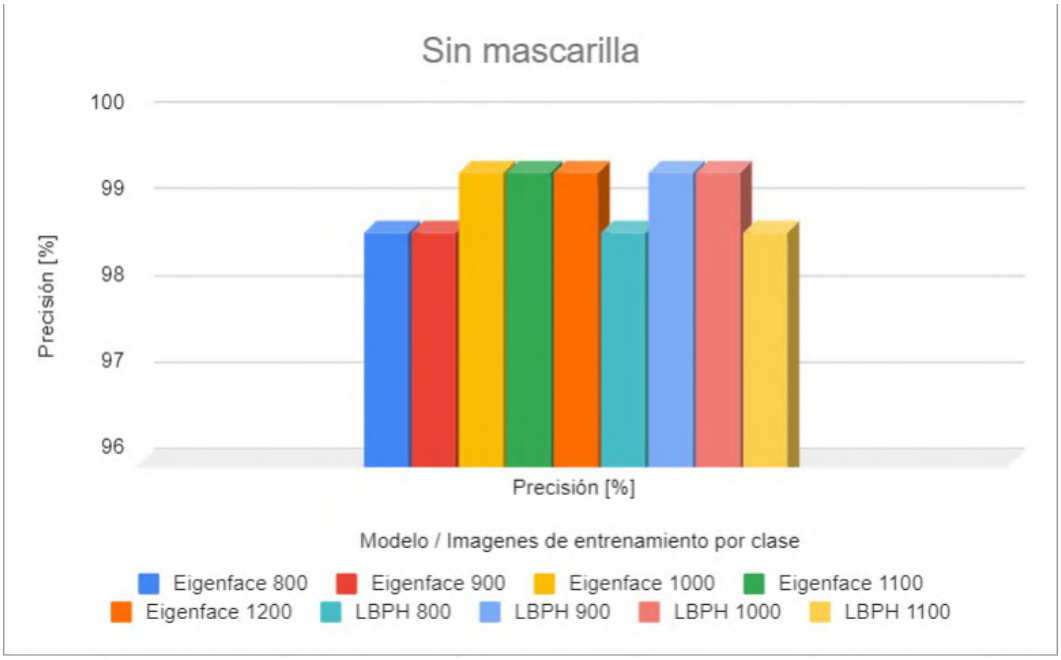


Figura 4.1: Gráfica de precisión en Raspberry para clase de: sin mascarilla

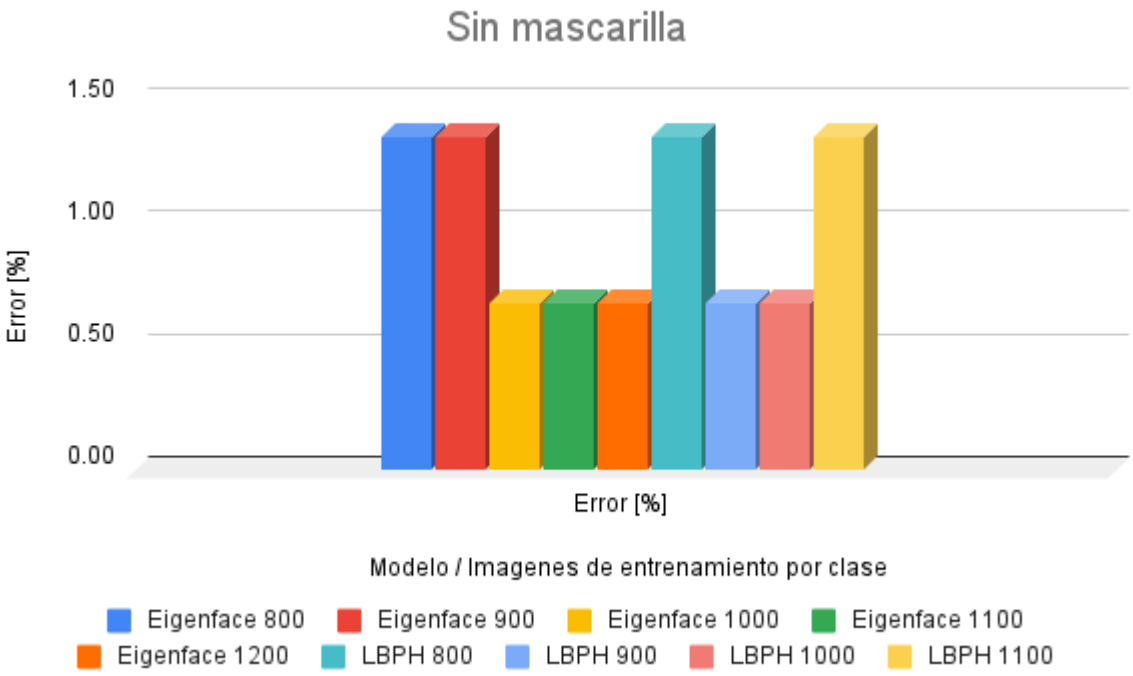


Figura 4.2: Gráfica de Error en Raspberry para clase de: sin mascarilla

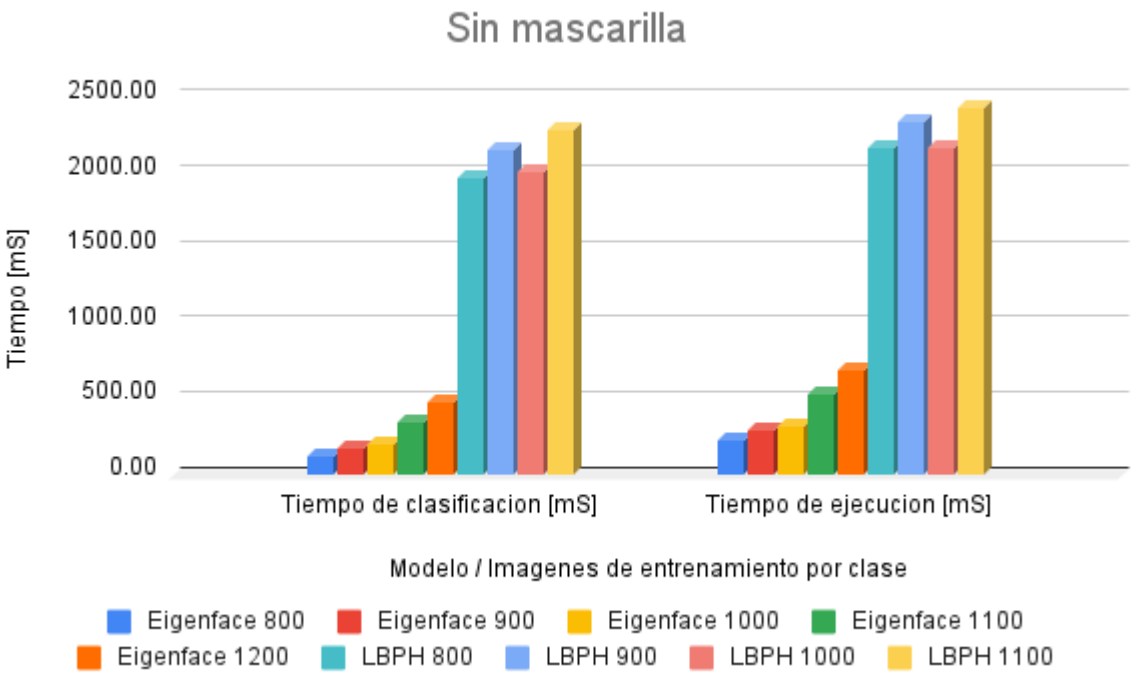


Figura 4.3: Gráfica de tiempos en Raspberry para clase de: sin mascarilla

4.1.1.2. Rendimiento con mascarilla mal puesta

Error mascarilla					
Modelo / Imágenes de entrenamiento por clase	Precisión [%]	Error [%]	Tiempo de clasificación [mS]	Tiempo de ejecución [mS]	Peso del modelo (MB)
Eigenface 800	93.28	6.71	254.47	537.01	466
Eigenface 900	93.28	6.71	306.41	578.63	545.4
Eigenface 1000	91.94	8.05	172.59	318.70	629.6
Eigenface 1100	94.63	5.36	226.74	400.70	718.5
Eigenface 1200	94.63	5.36	258.83	431.54	812.2
LBPH 800	93.96	6.04	1212.13	1362.70	188.5
LBPH 900	93.96	6.04	1312.40	1459.83	212
LBPH 1000	93.96	6.04	1297.73	1439.28	235.6
LBPH 1100	93.29	6.71	1440.71	1682.35	259.3

Tabla 4.2: Tabla de parámetros de rendimiento en Raspberry para clase de: Mascarilla mal puesta

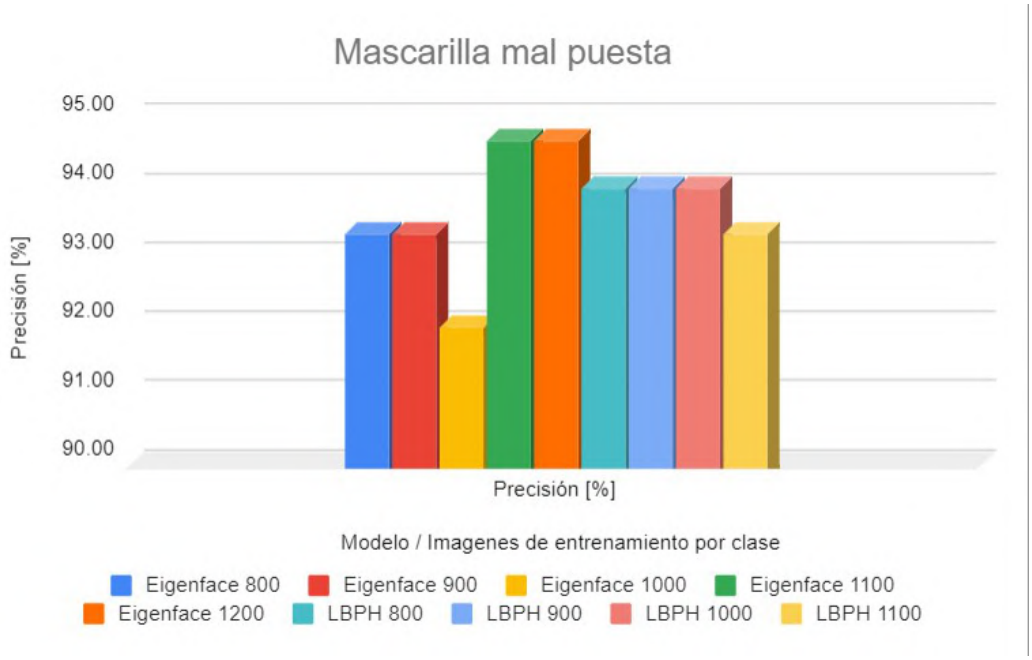


Figura 4.4: Gráfica de precisión en Raspberry para clase de: mascarilla mal puesta

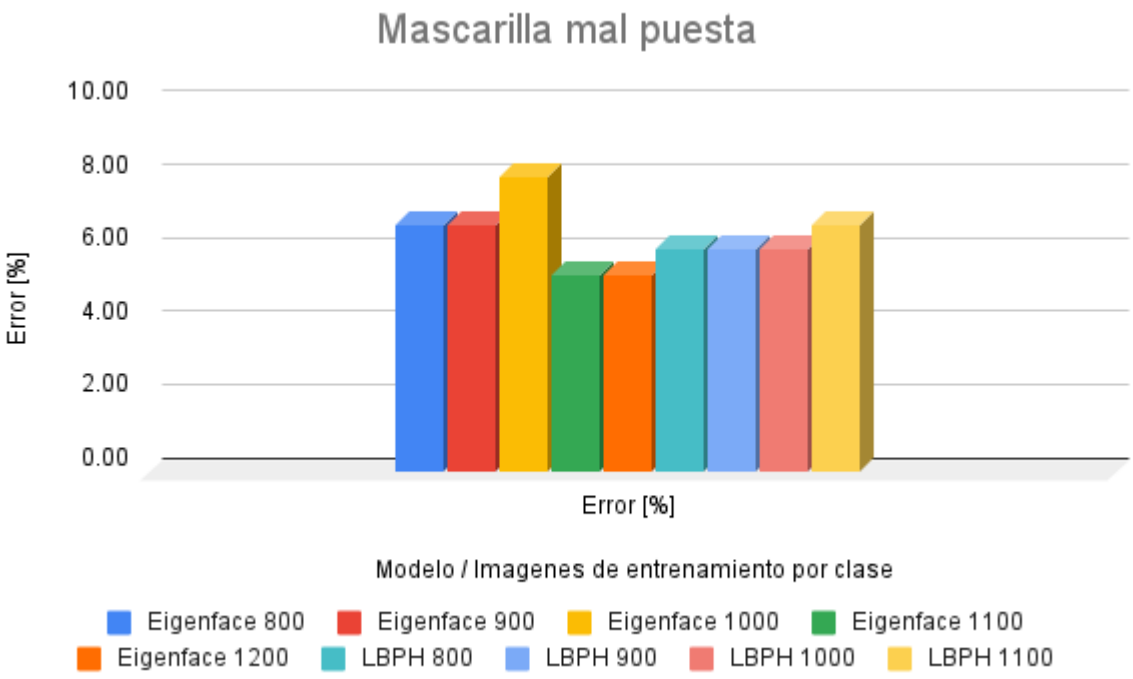


Figura 4.5: Gráfica de Error en Raspberry para clase de: mascarilla mal puesta

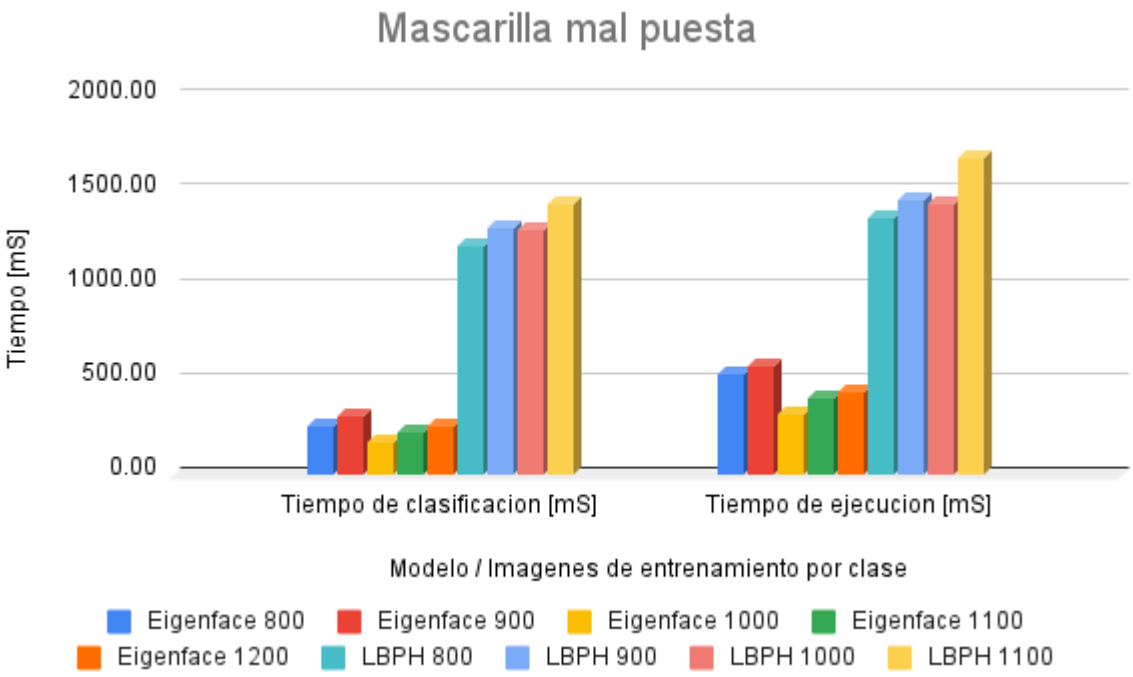


Figura 4.6: Gráfica de tiempos en Raspberry para clase de: mascarilla mal puesta

4.1.1.3. Rendimiento con mascarilla

Con mascarilla					
Modelo / Imágenes de entrenamiento por clase	Precisión [%]	Error [%]	Tiempo de clasificación [mS]	Tiempo de ejecución [mS]	Peso del modelo (MB)
Eigenface 800	93.10	6.89	198.06	492.80	466
Eigenface 900	91.03	8.96	149.37	340.17	545.4
Eigenface 1000	90.35	9.65	197.76	393.29	629.6
Eigenface 1100	90.35	9.65	228.50	435.94	718.5
Eigenface 1200	91.72	8.27	258.73	463.14	812.2
LBPH 800	96.55	3.44	1044.66	1205.99	188.5
LBPH 900	97.24	2.75	1172.11	1329.44	212
LBPH 1000	97.24	2.75	1366.71	1530.67	235.6
LBPH 1100	97.24	2.75	1549.96	1747.89	259.3

Tabla 4.3: Tabla de parámetros de rendimiento en Raspberry para clase de: con mascarilla

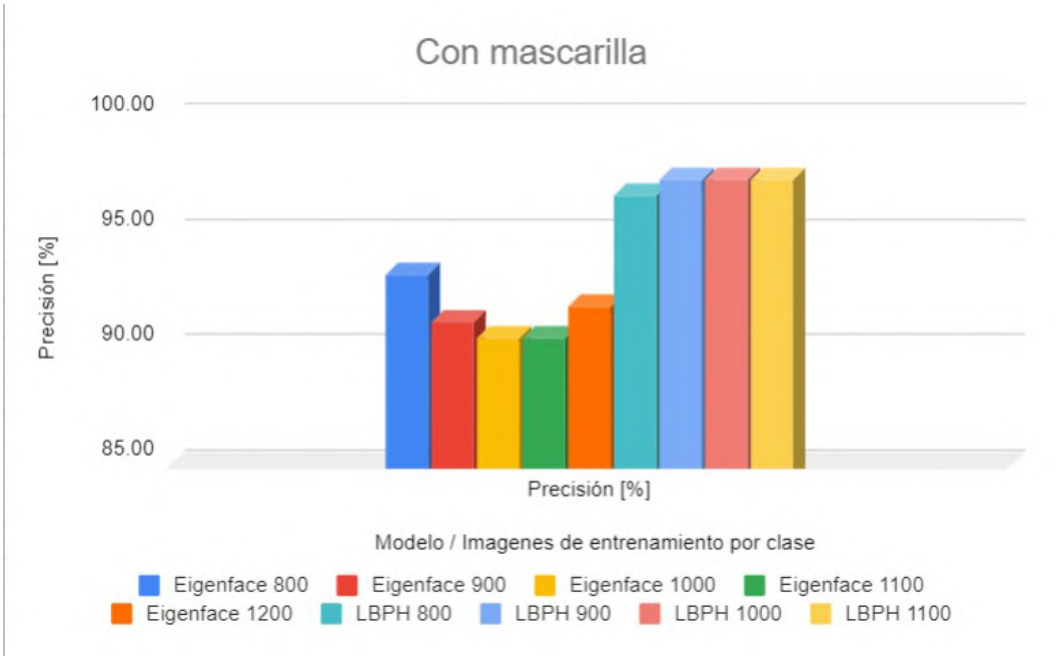


Figura 4.7: Gráfica de precisión en Raspberry para clase de: con mascarilla

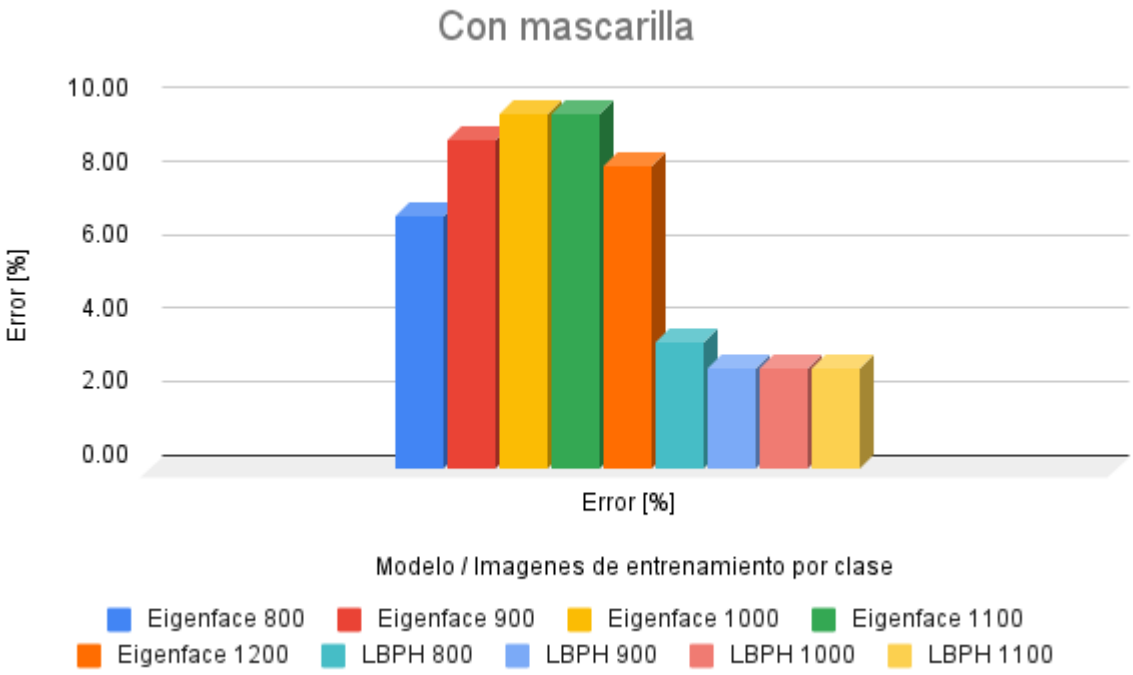


Figura 4.8: Gráfica de Error en Raspberry para clase de: con mascarilla

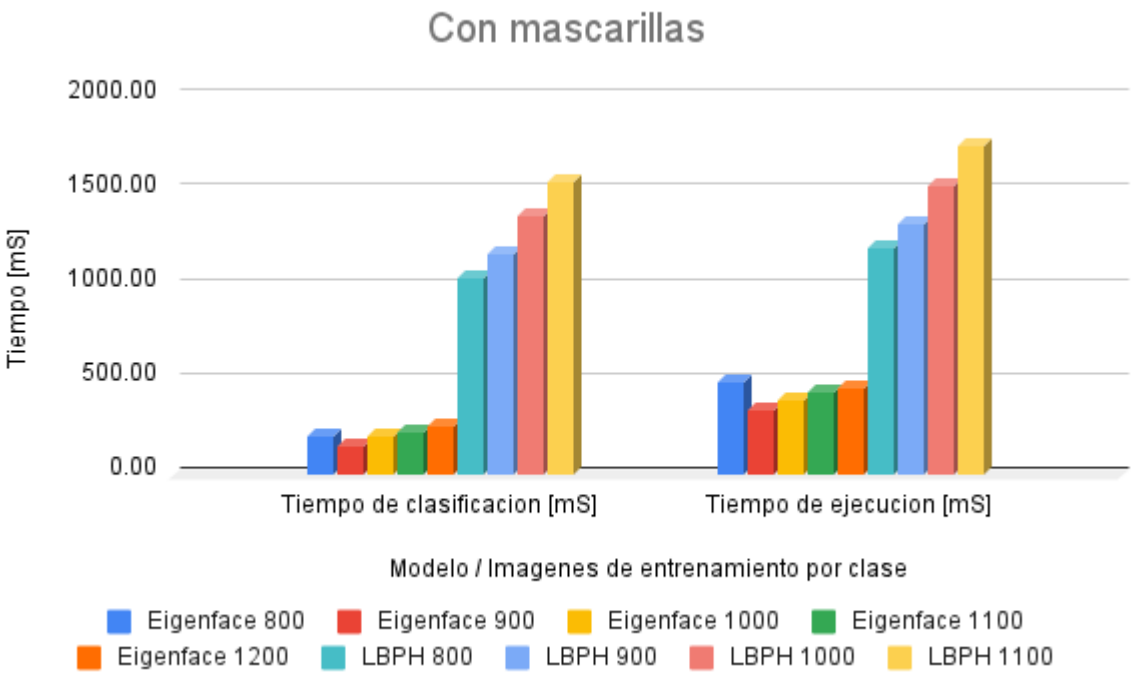


Figura 4.9: Gráfica de tiempos en Raspberry para clase de: con mascarilla

4.1.1.4. Matrices de confusión en Raspberry.

Matriz de Confusión Modelo LBPH 800 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	140	1	1
	Mascarilla erronea	5	140	1
	Sin mascarilla	0	8	145

Tabla 4.4: Matriz de confusión para modelo LBPH de 800 imágenes

Matriz de Confusión Modelo LBPH 900 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	141	0	0
	Mascarilla erronea	4	140	1
	Sin mascarilla	0	9	146

Tabla 4.5: Matriz de confusión para modelo LBPH de 900 imágenes

Matriz de Confusión Modelo LBPH 1000 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	141	0	0
	Mascarilla erronea	4	140	1
	Sin mascarilla	0	9	146

Tabla 4.6: Matriz de confusión para modelo LBPH de 1000 imágenes

Matriz de Confusión Modelo LBPH 1100 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	141	0	0
	Mascarilla erronea	4	140	2
	Sin mascarilla	0	9	145

Tabla 4.7: Matriz de confusión para modelo LBPH de 1100 imágenes

Matriz de Confusión Modelo EigenFaces 800 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	135	7	2
	Mascarilla erronea	9	139	0
	Sin mascarilla	1	3	145

Tabla 4.8: Matriz de confusión para modelo EigenFace de 800 imágenes

Matriz de Confusión Modelo EigenFaces 900 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	132	7	2
	Mascarilla erronea	11	139	0
	Sin mascarilla	2	3	145

Tabla 4.9: Matriz de confusión para modelo EigenFace de 900 imágenes

Matriz de Confusión Modelo EigenFaces 1000 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	131	6	1
	Mascarilla erronea	9	137	0
	Sin mascarilla	5	6	146

Tabla 4.10: Matriz de confusión para modelo EigenFace de 1000 imágenes

Matriz de Confusión Modelo EigenFaces 1100 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	131	3	1
	Mascarilla erronea	9	141	0
	Sin mascarilla	5	5	146

Tabla 4.11: Matriz de confusión para modelo EigenFace de 1100 imágenes

Matriz de Confusión Modelo EigenFaces 1200 imagenes		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	133	5	1
	Mascarilla erronea	7	141	0
	Sin mascarilla	5	3	146

Tabla 4.12: Matriz de confusión para modelo EigenFace de 1100 imágenes

Para el caso de la plataforma Raspberry, los métodos de obtención de tiempos y rendimientos de los modelos fueron similares a los métodos usados en la plataforma PC explicados en la sección 2.1.1 del libro de anexos (Torres y Sanchez, 2022a), la cual haciendo mención; corresponden a una precisión y error que proviene de una suma de etiquetas arrojadas por el modelo en cuestión, por supuesto dividiendo sobre el numero total de imágenes evaluadas por clase. Para obtener los datos de tiempo, la primera bandera que se usa es para el tiempo de clasificación y se toma justo antes de ejecutar la función de predicción del modelo correspondiente. Por otra parte, la segunda bandera se toma una vez el algoritmo obtiene una respuesta por parte del modelo.

En el caso del tiempo de ejecución total, se toma la primera bandera justo antes de leer el $frame$ y la segunda bandera se toma, al igual que en el caso anterior, justo después de obtener una respuesta por el clasificador, de esta forma se han calculado los anteriores parámetros de rendimiento.

Dentro de los datos obtenidos se puede observar que los valores de precisión y error en las tablas 4.1, 4.2, 4.3 son muy similares a los valores de precisión y error en la plataforma PC (Torres y Sanchez, 2022a). Esto debido a que los modelos usados en ambas plataformas fueron exactamente los mismos y por ende, los rendimientos deberán ser similares.

Tal y como se puede observar en los datos presentados en las gráficas de las figuras 4.3, 4.6, 4.9, el uso de estos modelos clasificatorios es considerablemente mas lento en plataformas embebidas como la Raspberry en comparación a plataformas con mayor capacidad de procesamiento como lo es un computador (Torres y Sanchez, 2022a). A pesar de este hecho, modelos con menor precisión en cambios de luminosidad de la imagen como los modelos $Eigenface$, presentan tiempos de ejecución que pueden llegar a ser sostenibles en aplicaciones cotidianas y en el uso practico de este algoritmo para detección de mascarillas en vídeo streaming.

4.1.2. Reconocimiento en vivo. Plataforma: Raspberry

4.1.2.1. Rendimiento sin mascarilla

Sin mascarilla					
Modelo / Imágenes de entrenamiento por clase	Precisión [%]	Error [%]	Tiempo de clasificación [mS]	Tiempo de ejecución [mS]	Peso del modelo (MB)
Eigenface 800	100	0.00	171.76	366.06	466
Eigenface 900	99.336	0.66	216.96	412.23	545.4
Eigenface 1000	96.309	3.69	216.04	375.57	629.6
Eigenface 1100	95.349	4.65	271.21	437.83	718.5
Eigenface 1200	93.355	6.65	342.03	517.36	812.2
LBPH 800	96.346	3.65	1068.41	1182.93	188.5
LBPH 900	94.02	5.98	1799.74	1957.06	212
LBPH 1000	99.66	0.34	1865.32	2015.35	235.6
LBPH 1100	98.63	1.36	2291.16	2429.95	259.3

Tabla 4.13: Tabla de parámetros de rendimiento en Raspberry para clase de: sin mascarilla

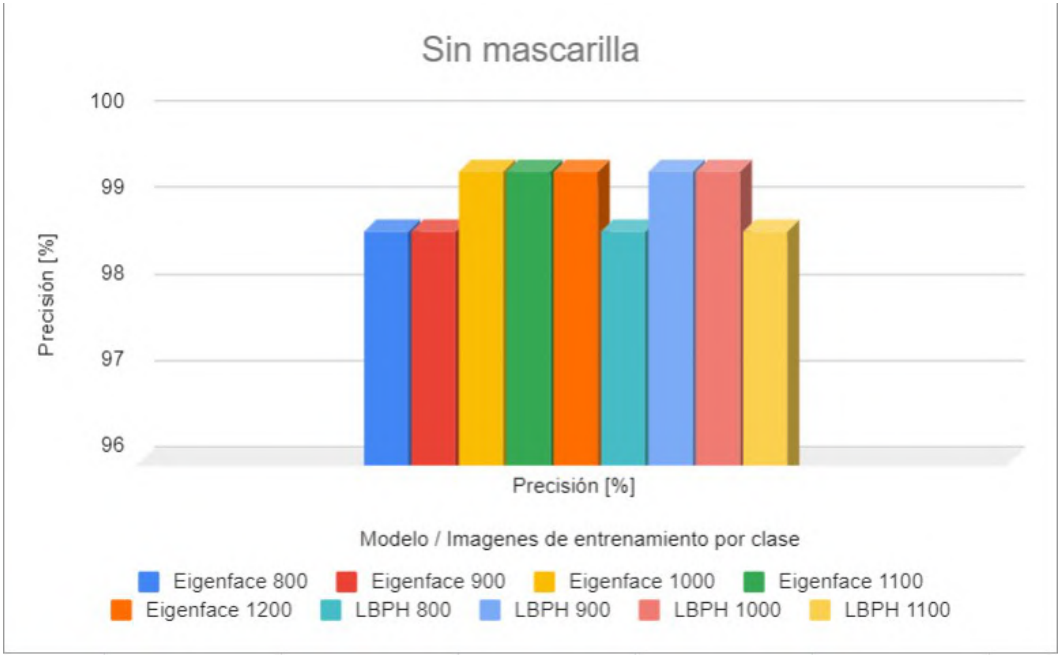


Figura 4.10: Gráfica de precisión en Raspberry para clase de: sin mascarilla

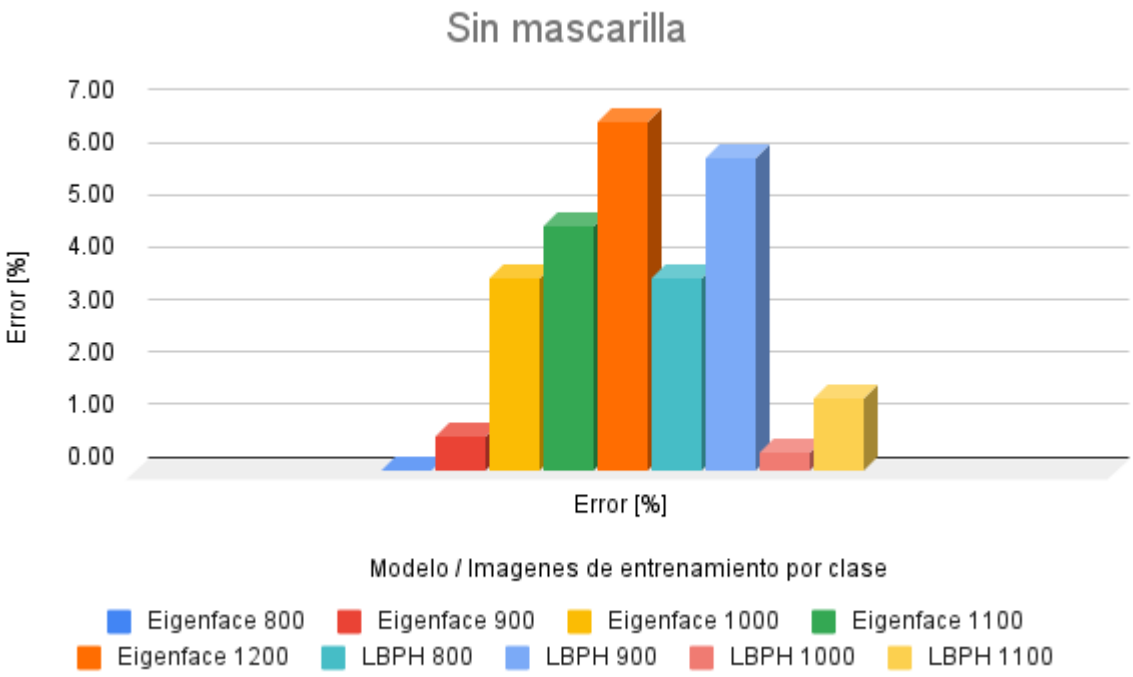


Figura 4.11: Gráfica de Error en Raspberry para clase de: sin mascarilla

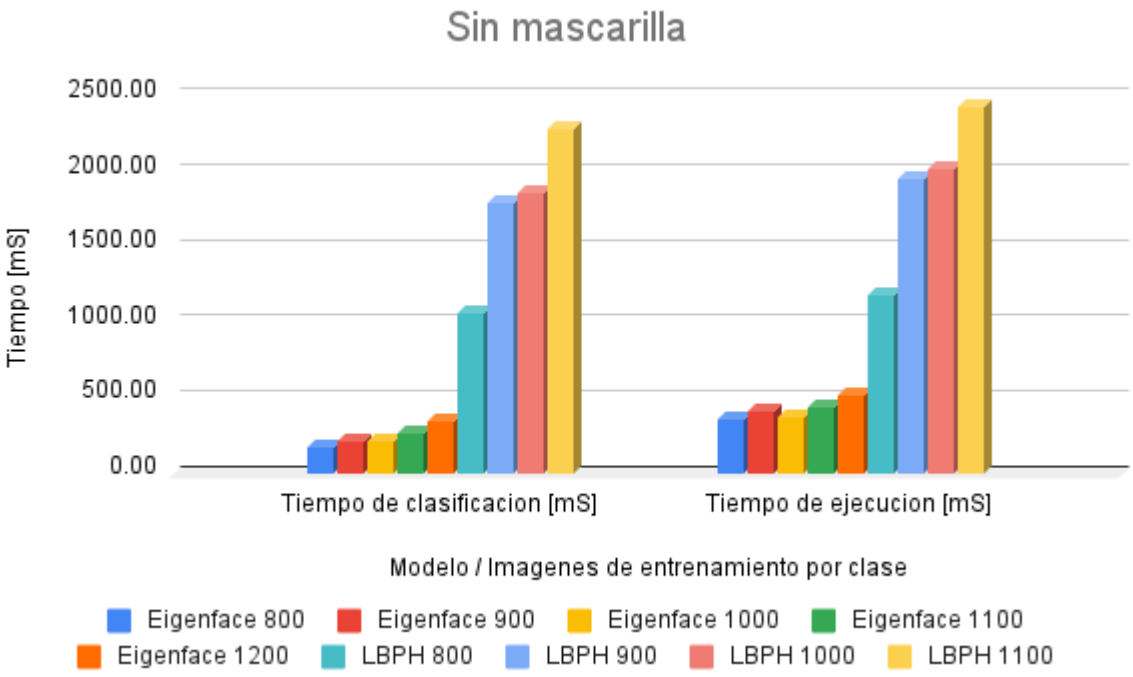


Figura 4.12: Gráfica de tiempos en Raspberry para clase de: sin mascarilla

4.1.2.2. Rendimiento con mascarilla mal puesta

Error mascarilla					
Modelo / Imágenes de entrenamiento por clase	Precisión [%]	Error [%]	Tiempo de clasificación [mS]	Tiempo de ejecución [mS]	Peso del modelo (MB)
Eigenface 800	100.00	0.00	163.20	348.17	466
Eigenface 900	100.00	0.00	207.54	391.21	545.4
Eigenface 1000	99.34	0.66	263.77	456.71	629.6
Eigenface 1100	94.86	5.14	309.14	496.89	718.5
Eigenface 1200	96.67	3.33	367.73	557.62	812.2
LBPH 800	99.67	0.33	1677.64	1848.85	188.5
LBPH 900	99.34	0.66	1698.36	1985.36	212
LBPH 1000	95.349	4.65	1958.75	2115.95	235.6
LBPH 1100	100.00	0.00	2098.65	2175.82	259.3

Tabla 4.14: Tabla de parámetros de rendimiento en Raspberry para clase de: mascarilla mal puesta

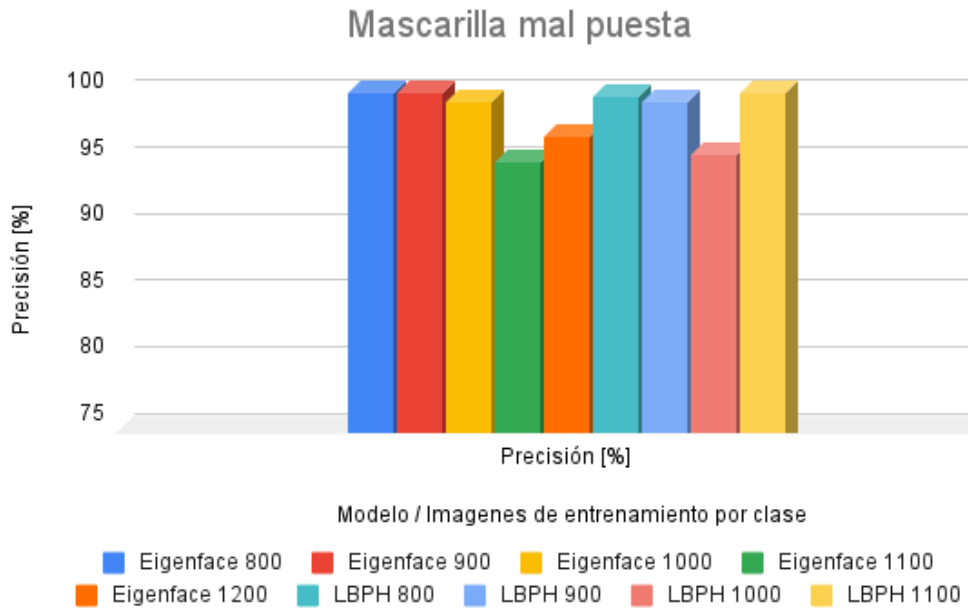


Figura 4.13: Gráfica de precisión en Raspberry para clase de: mascarilla mal puesta

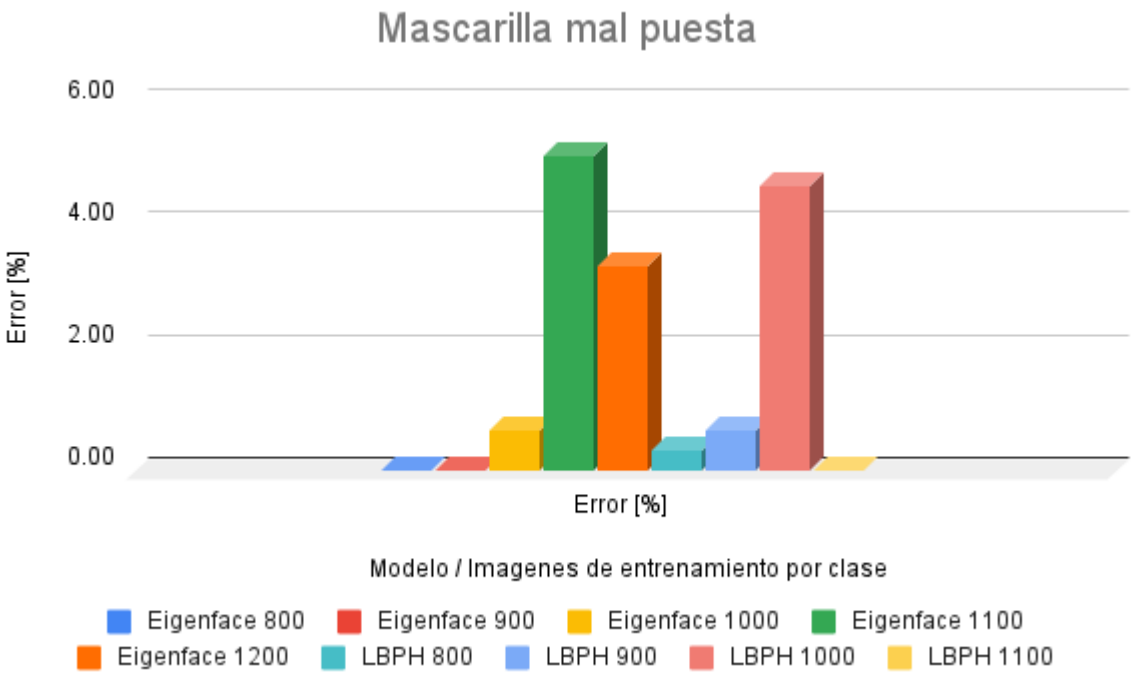


Figura 4.14: Gráfica de Error en Raspberry para clase de: mascarilla mal puesta

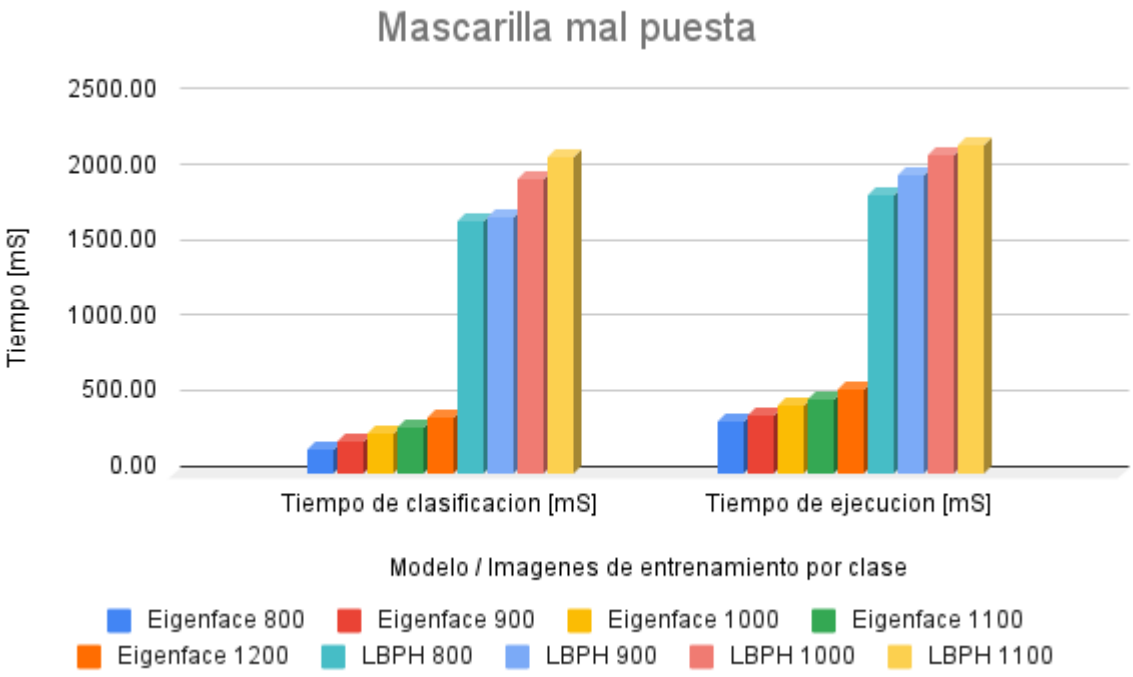


Figura 4.15: Gráfica de tiempos en Raspberry para clase de: mascarilla mal puesta

4.1.2.3. Rendimiento con mascarilla

Con mascarilla					
Modelo / Imágenes de entrenamiento por clase	Precisión [%]	Error [%]	Tiempo de clasificación [mS]	Tiempo de ejecución [mS]	Peso del modelo (MB)
Eigenface 800	96.49	3.40	171.40	364.60	466
Eigenface 900	93.27	6.73	114.72	223.89	545.4
Eigenface 1000	88.67	11.33	132.43	237.02	629.6
Eigenface 1100	90.35	9.65	154.36	251.32	718.5
Eigenface 1200	98.67	1.33	186.50	288.94	812.2
LBPH 800	100.00	0.00	782.48	878.49	188.5
LBPH 900	98.94	1.06	916.30	1013.21	212
LBPH 1000	100.00	0.00	1043.49	1138.65	235.6
LBPH 1100	99.66	0.34	1181.58	1281.46	259.3

Tabla 4.15: Tabla de parámetros de rendimiento en Raspberry para clase de: con mascarilla

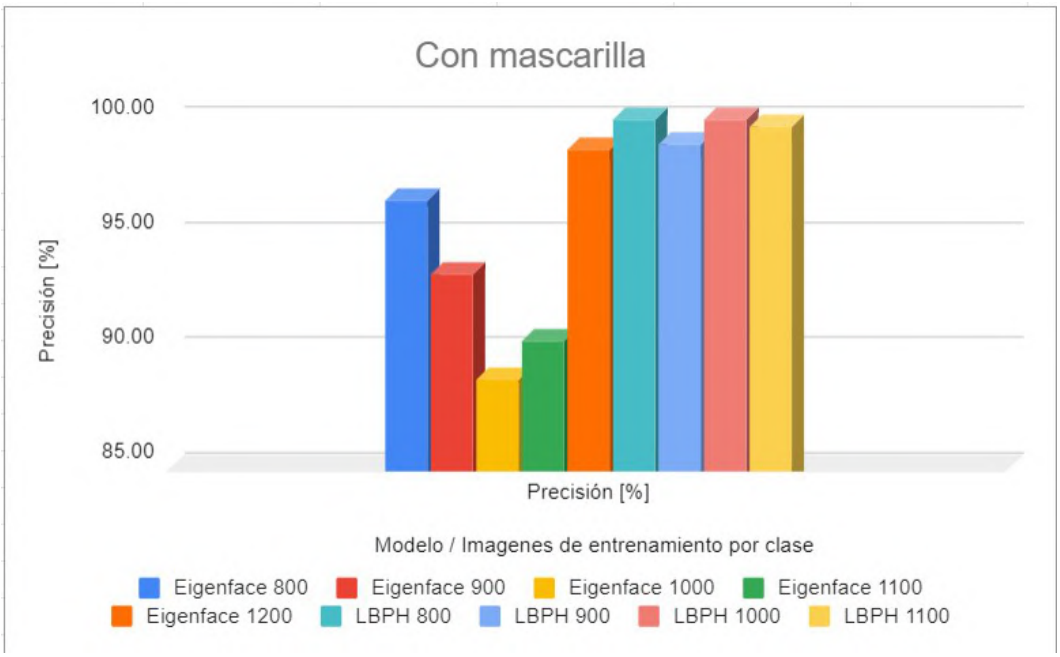


Figura 4.16: Gráfica de precisión en Raspberry para clase de: con mascarilla

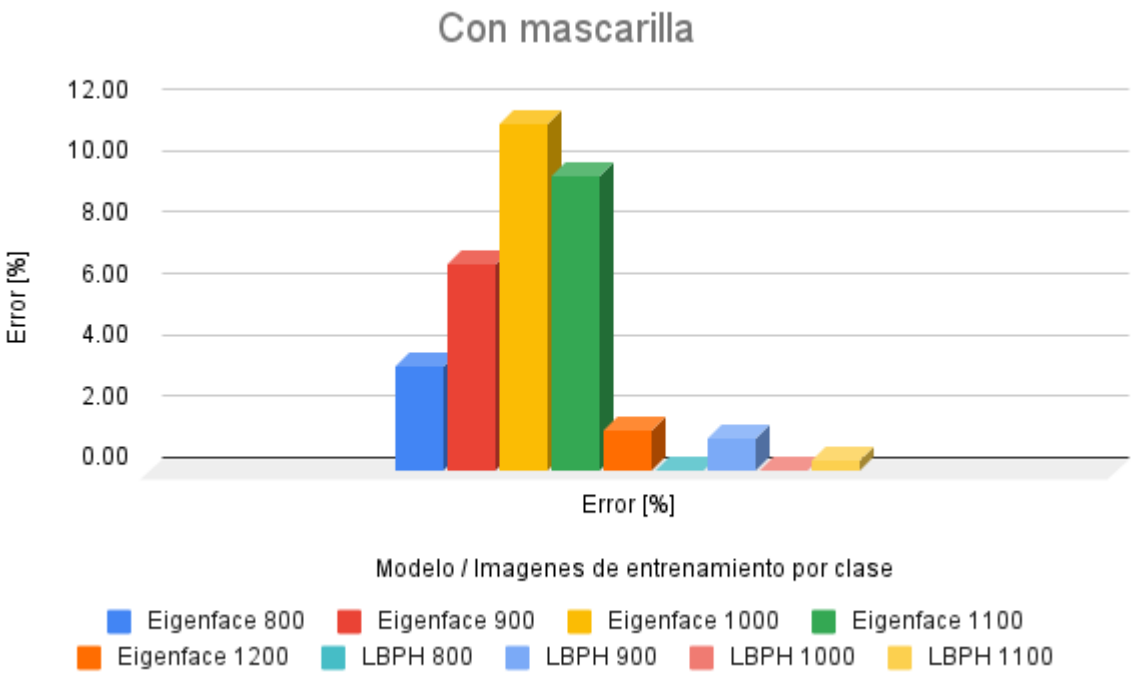


Figura 4.17: Gráfica de Error en Raspberry para clase de: con mascarilla

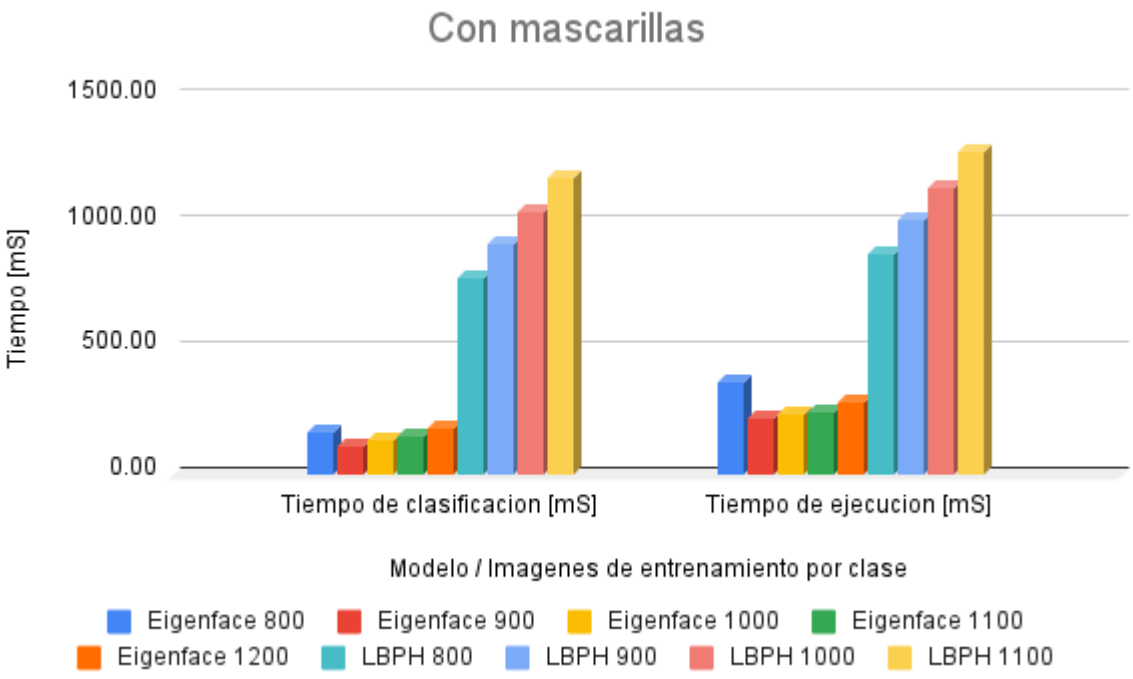


Figura 4.18: Gráfica de tiempos en Raspberry para clase de: con mascarilla

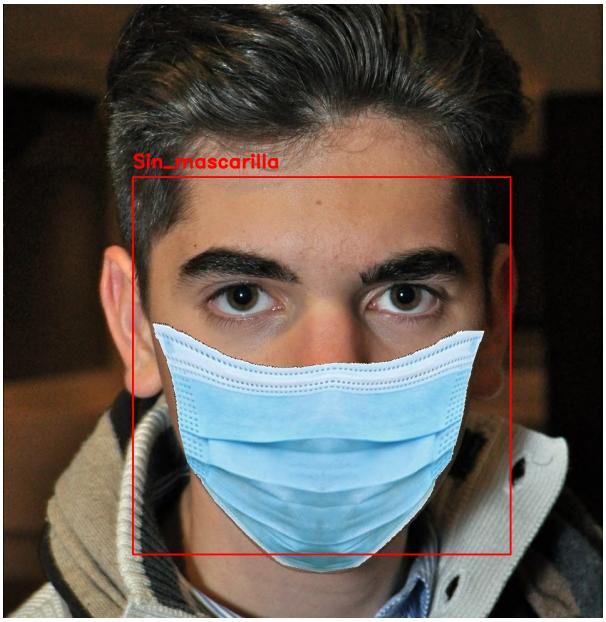
Para el caso del rendimiento de los modelos con vídeo en vivo, se utilizó una cámara compatible con Raspberry desde la cual se capturaron trecientos frames de un sujeto de prueba con una mascarilla facial en una posición acorde a la categoría que se estaba evaluando obteniendo los resultados contenidos en las tablas 4.13, 4.14, 4.15.

Para la obtención de los datos de rendimiento, error y tiempos de ejecución se tomaron varias muestras por cada categoría y se promediaron los datos obtenidos observando que dichos resultados presentaban una variación considerable entre prueba y prueba que era determinada por diversos factores como la iluminación; la posición corporal, el ángulo de inclinación y el ángulo de rotación del sujeto de pruebas; el movimiento durante la prueba y las variaciones en alguno de los parámetros antes mencionados.

Con lo anterior en cuenta, se intentó realizar las pruebas de rendimiento con una variación mínima en los parámetros que podían afectar los resultados obteniendo valores satisfactorios de hasta cien por ciento de éxito de clasificación en todas las pruebas de algunos modelos en varias categorías tal y como se puede observar en las tablas de rendimiento y en las figuras 4.10, 4.13, 4.16, las cuales expresan gráficamente dichos resultados.

Por otra parte, los tiempos de clasificación y de ejecución fueron, en la mayoría de los casos, ligeramente mas pequeños que los tiempos de clasificación y ejecución en bancos de imágenes de prueba. Aunque siguen siendo considerablemente altos y dicha mejora no vuelve a los modelos completamente viables para la práctica, es un hecho a considerar que el tiempo de clasificación en vídeo podría permitir al algoritmo funcionar de forma mas fluida y eficaz para algoritmos rápidos como los que utilizan el modelo de $Eigenface$.

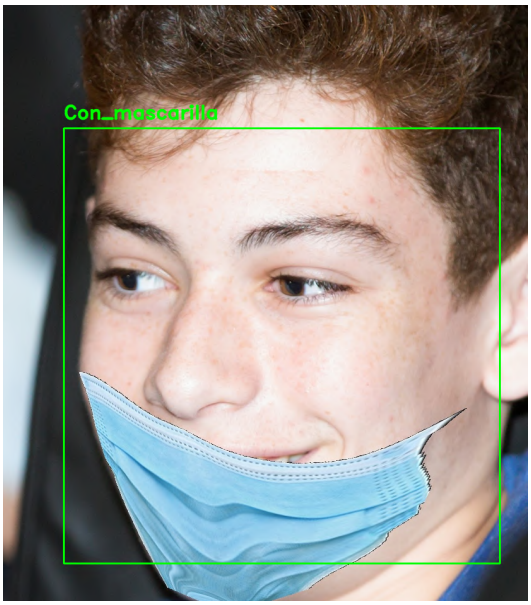
4.1.3. Errores encontrados en el clasificado de imágenes



(a) Error 1

(b) Error 2

Figura 4.19: Errores encontrados dentro de la clase de: con mascarilla



(a) Error 3

(b) Error 4

Figura 4.20: Errores encontrados dentro de la clase de: mascarilla mal puesta

Tal y como se puede observar en las figuras 4.19a y 4.19b, los clasificadores tanto Eigenface y LBPH han designado sin mascarilla y con mascarilla mal puesta a imágenes que realmente contienen rostros con mascarillas, estos es principalmente a que en primera instancia estas imágenes no se usaron en el momento de realizar el entrenamiento, de igual forma también se le puede atribuir a la varianza de la intensidad de la luz que produce una convergencia en los datos parecidos de otras clases.

Lo descrito en el inciso anterior también coincide para los errores de clasificado mostrados en las figuras 4.20a y 4.20b pero en este caso también se le puede atribuir a la rotación de los rostros y la posición de la mascarilla que de cierta forma llega a mostrar la boca combinando propiedades de otras clases que nuevamente producen errores de convergencia en los datos.

4.2. Resultados con Redes neuronales

Es importante mencionar que antes de proceder a usar las plataformas de Sipeed Maix Go y Maix Bit se planteó el uso de una red neuronal convolucional en PC con el objetivo de poder ejecutarla después en la plataforma de Raspberry, desafortunadamente el modelo de Raspberry con el que contábamos no posee el hardware necesario para poder ejecutar con facilidad o solvencia aceptable los cálculos que hace una red neuronal convolucional así que no fue posible correr el algoritmo que se desarrolló en PC en un principio con dicha red en la Raspberry. Si el lector esta interesado en conocer mas acerca de los resultados de dicha red neuronal convolucional en la plataforma de PC puede consultar con mas detalle la sección 2.2.1 del libro de anexos (Torres y Sanchez, 2022a). Esta sección contiene los resultados de las pruebas. De igual forma, la sección 1.1 contiene la fundamentación y metodología usada en el entrenamiento y ejecución de la Red.

Por otra parte, se realizaron pruebas de la red neuronal entrenada por Maixhub en las tarjetas Maix Go y Maix Bit de la misma forma como se hicieron las pruebas sobre la Raspberry en la sección 4.1. En primer lugar se probó el algoritmo con un banco de imágenes

de prueba que contenía 150 elementos por cada clase de clasificación y se observó la cantidad de fallos y aciertos que tenía el algoritmo al clasificar cada una de estas imágenes para calcular la precisión y el error. Por otra parte, se realizaron pruebas de vídeo en vivo en el que se enfocó un sujeto de prueba con mascarilla, sin mascarilla y con la mascarilla usada de forma incorrecta y se le permitió al algoritmo clasificar las imágenes que recibía por medio de la cámara durante 150 fotogramas para así contar fallos y aciertos y calcular también precisión y error.

4.2.1. Reconocimiento con imágenes. Plataforma: Maix Go

4.2.1.1. Rendimiento sin mascarilla

Sin Mascarilla				
Modelo	**Precisión [%]**	**Error [%]**	**Tiempo de Clasificación [mS]**	**Peso del modelo [MB]**
YOLOv2	**100**	**0.00**	**34.269**	**1.82**

Tabla 4.16: Tabla de parámetros de rendimiento en Maix GO para clase de: sin mascarilla

4.2.1.2. Rendimiento con mascarilla mal puesta

Error Mascarilla				
Modelo	**Precisión [%]**	**Error [%]**	**Tiempo de Clasificación [mS]**	**Peso del modelo [MB]**
YOLOv2	**98.667**	**1.333**	**34.309**	**1.82**

Tabla 4.17: Tabla de parámetros de rendimiento en Maix GO para clase de: mascarilla mal puesta

4.2.1.3. Rendimiento con mascarilla

Con Mascarilla				
Modelo	**Precisión [%]**	**Error [%]**	**Tiempo de Clasificación [mS]**	**Peso del modelo [MB]**
YOLOv2	**98.667**	**1.333**	**34.322**	**1.82**

Tabla 4.18: Tabla de parámetros de rendimiento en Maix GO para clase de: con mascarilla

4.2.1.4. Matriz de confusión

Matriz de Confusión Modelo Maixhub		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	148	3	0
	Mascarilla erronea	1	148	0
	Sin mascarilla	1	3	150

Tabla 4.19: Matriz de confusión para banco de imágenes de pruebas

Tal como se puede observar en las tablas 4.16, 4.17 y 4.18 los valores de precisión que presenta este algoritmo son bastante satisfactorios para el reconocimiento y clasificación de rostros y mascarillas superando el 98 % del ya mencionado parámetro de rendimiento.

Dentro de los problemas que se presentaron con los algoritmos de *machine learning* vistos en la sección 4.1 de este libro se encontraba el hecho de que los modelos usualmente debían escoger entre precisión o velocidad al momento de ejecutar los algoritmos de clasificación. Este problema no se presenta con este modelo de red neuronal ya que el tiempo de clasificación es de poco mas de 34 [mS] lo que permite clasificar muchas mas imágenes en un menor tiempo.

Una particularidad que se debe notar de estas pruebas realizadas al algoritmo se encuentra en la tabla 4.19, la cual nos presenta la matriz de confusión del modelo al realizar las pruebas de rendimiento. Como se expuso en la sección 4.2, los bancos de prueba fueron únicamente de 150 imágenes por clase y al observar la matriz de confusión se puede observar en la clase de "Mascarilla Errónea" que se tiene un total de 154 imágenes clasificadas. Esto se presenta ya que el algoritmo puede detectar y clasificar múltiples rostros en las imágenes que le son presentadas. Esta clasificación de múltiples rostros se realiza a pesar de que todas las imágenes de los bancos de prueba tienen únicamente un rostro para clasificar, lo que representa un error del algoritmo al realizar múltiples

detecciones y clasificaciones sobre un solo rostro.

4.2.2. Reconocimiento en vivo. Plataforma: Maix Go

4.2.2.1. Rendimiento sin mascarilla

Sin Mascarilla				
Modelo	**Precisión [%]**	**Error [%]**	**Tiempo de Clasificación [mS]**	**Peso del modelo [MB]**
YOLOv2	**99.329**	**0.671**	**34.436**	**1.82**

Tabla 4.20: Tabla de parámetros de rendimiento en Maix GO para clase de: sin mascarilla

4.2.2.2. Rendimiento con mascarilla mal puesta

Error Mascarilla				
Modelo	**Precisión [%]**	**Error [%]**	**Tiempo de Clasificación [mS]**	**Peso del modelo [MB]**
YOLOv2	**100**	**0.00**	**34.66**	**1.82**

Tabla 4.21: Tabla de parámetros de rendimiento en Maix GO para clase de: mascarilla mal puesta

4.2.2.3. Rendimiento con mascarilla

Con Mascarilla				
Modelo	**Precisión [%]**	**Error [%]**	**Tiempo de Clasificación [mS]**	**Peso del modelo [MB]**
YOLOv2	**95.333**	**4.667**	**34.826**	**1.82**

Tabla 4.22: Tabla de parámetros de rendimiento en Maix GO para clase de: con mascarilla

4.2.2.4. Matriz de confusión

Matriz de Confusión Modelo Maixhub		Predicción		
		Con mascarilla	Mascarilla erronea	Sin mascarilla
Observación	Con mascarilla	143	0	1
	Mascarilla erronea	3	150	0
	Sin mascarilla	4	0	149

Tabla 4.23: Matriz de confusión para videostream

Tal y como los resultados de reconocimiento en vivo de los modelos de *machine learning*, los resultados obtenidos para estos modelos son dependientes en gran medida de diversos factores ambientales y del entorno en el que se está ejecutando el algoritmo de clasificación. A pesar de los datos de rendimiento que se pueden observar en las tablas 4.20, 4.21 y 4.22 es necesario mencionar que dichas pruebas de vídeo en vivo fueron realizadas en ambientes óptimos de iluminación, con variaciones considerables de movimiento en el sujeto de prueba que no afectaron las métricas de rendimiento y con un ángulo de visión adecuado hacia el objetivo.

Tambien debe mencionarse que la calidad de la cámara genérica que se utilizó para las pruebas no fue la mas alta, lo que dificultó un poco la adquisición de los datos de rendimiento. A pesar de todo lo anterior, las métricas de rendimiento fueron bastante favorables permitiendo superar en algunas clases las medidas de rendimiento de los bancos de imágenes. Por otra parte, los tiempos de clasificación del algoritmo fueron también de menos de 35 [mS] lo que hace este método muy viable para el reconocimiento facial de mascarillas en tiempo real.

4.3. Comparativa de resultados.

Para esta comparativa se ha seleccionado el modelo con mejor desempeño dentro de los diferentes tipos de modelos (*Eigenface*, LBPH, YOLO V2) y se han realizado las

siguientes gráficas:

4.3.1. Reconocimiento de Imágenes.

4.3.1.1. Precisión

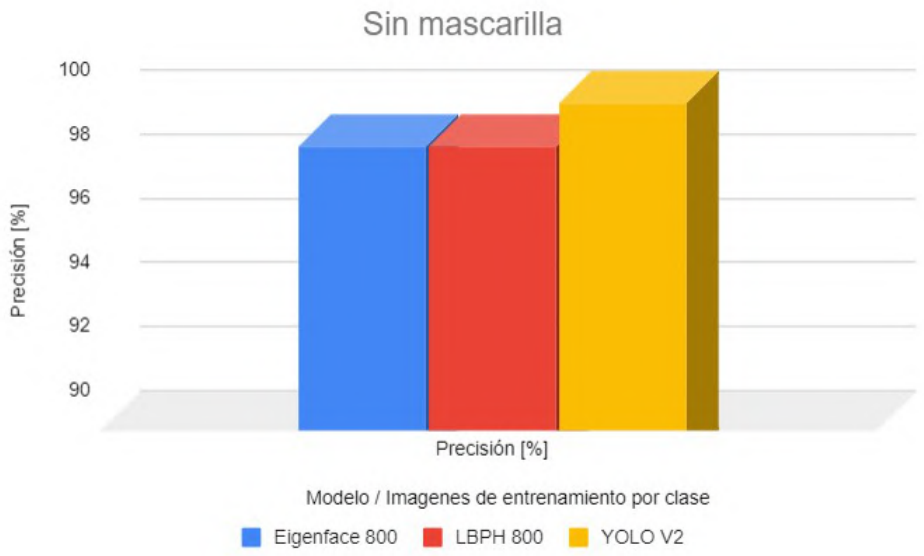


Figura 4.21: Gráfica de comparación en la precisión para clase de: Sin mascarilla.

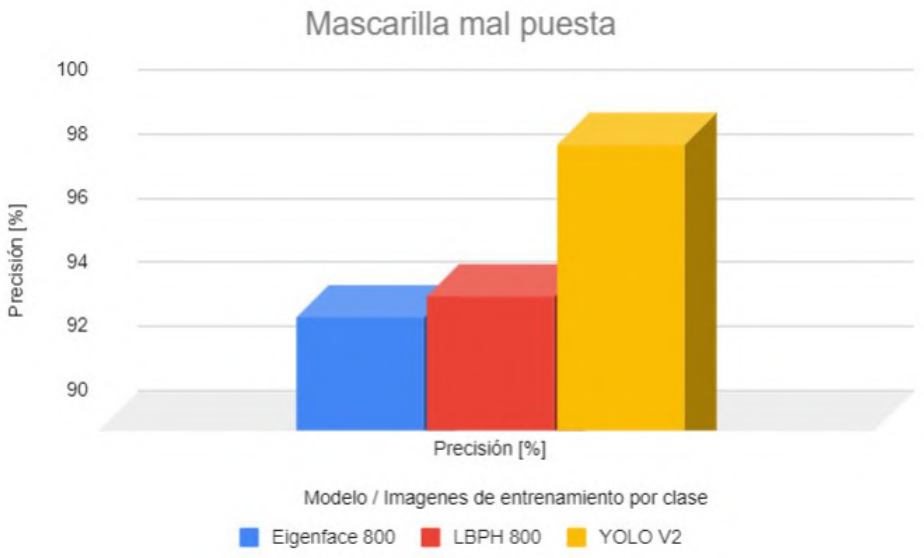


Figura 4.22: Gráfica de comparación en la precisión para clase de: Mascarilla mal puesta.

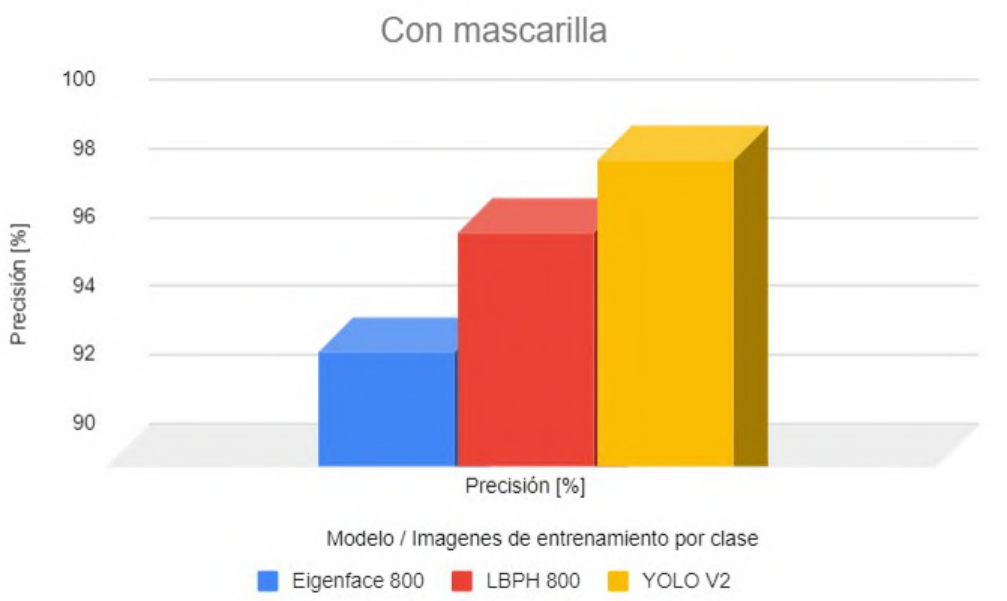


Figura 4.23: Gráfica de comparación en la precisión para clase de: Con mascarilla.

4.3.1.2. Tiempo de clasificación

Figura 4.24: Gráfica de comparación de tiempos para clase de: Sin mascarilla.

Figura 4.25: Gráfica de comparación de tiempos para clase de: Mascarilla mal puesta.

Figura 4.26: Gráfica de comparación de tiempos para clase de: Con mascarilla.

4.3.2. Reconocimiento en vivo.

4.3.2.1. Precisión

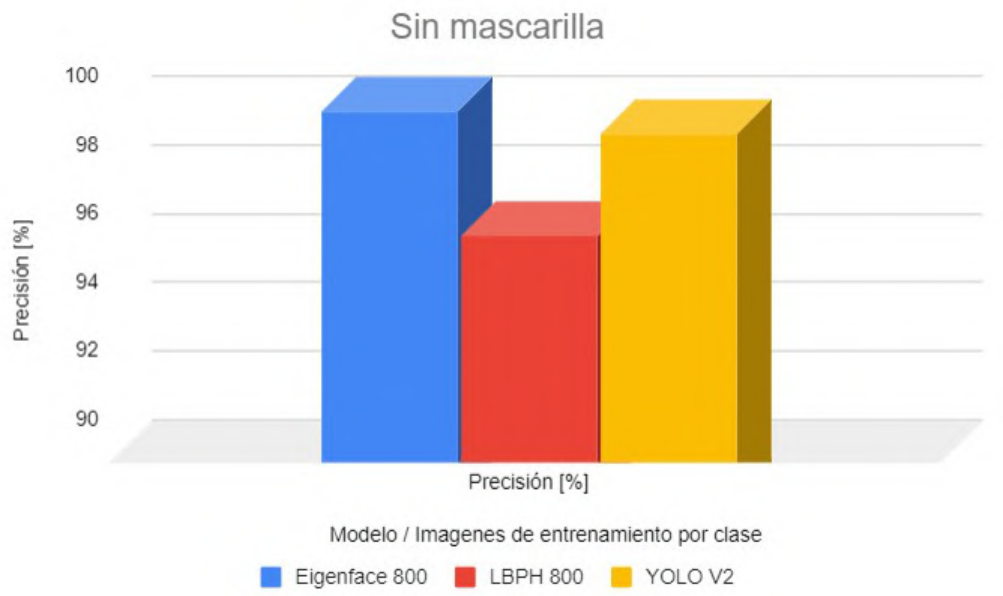


Figura 4.27: Gráfica de comparación en la precisión para clase de: Sin mascarilla.

Figura 4.28: Gráfica de comparación en la precisión para clase de: Mascarilla mal puesta.

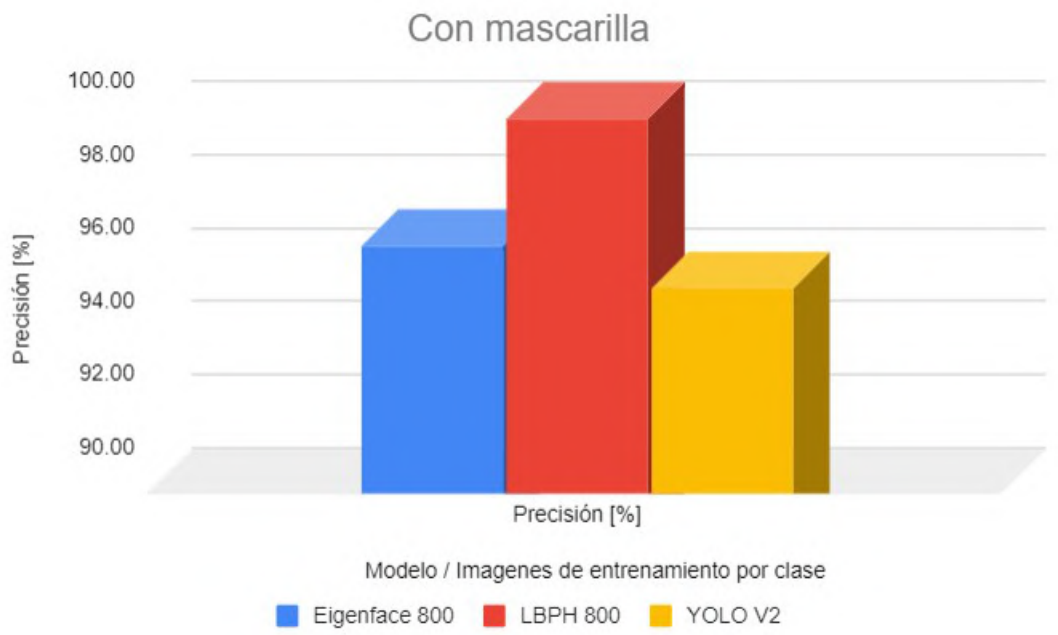


Figura 4.29: Gráfica de comparación en la precisión para clase de: Con mascarilla.

4.3.2.2. Tiempos de clasificación

Figura 4.30: Gráfica de comparación de tiempos para clase de: Sin mascarilla.

Figura 4.31: Gráfica de comparación de tiempos para clase de: Mascarilla mal puesta.

Figura 4.32: Gráfica de comparación de tiempos para clase de: Con mascarilla.

5. Conclusiones

- Con el fin de desarrollar un algoritmo adecuado para la detección y clasificación de rostros con mascarillas se logró hacer una construcción de una base de datos adecuada para el entrenamiento de los modelos en los que se basaron dichos algoritmos. Estas bases de datos fueron creadas a partir de una recopilación de imágenes de diversas fuentes como bases de datos existentes en la web (C., 2021) e imágenes de los rostros de personas voluntarias que aceptaron participar en la creación de este banco de imágenes y que ayudaron a hacer posible este proyecto.

- Tal y como se puede observar en el capítulo 3, se realizó el entrenamiento de diversos modelos basados en tres métodos enfocados en la detección y clasificación. Dos de ellos basados en *Machine learning* y apoyados por la librería de detección *mediapipe* y clasificación *OpenCV* mientras que el tercero de ellos se basó en redes neuronales convolucionales. Todos los algoritmos que se desarrollaron a partir de estos métodos cumplieron satisfactoriamente su propósito de detectar y clasificar rostros con mascarillas y obtuvieron tasas de precisión satisfactorias.

- A pesar de que todos los métodos utilizados tuvieron sus propias características, el modelo de redes neuronales que se implementó en las tarjetas Maix presentó los mejores parámetros de rendimiento en cuestión de precisión y tiempo, tal y como se puede observar en la sección 4.3. Esta superioridad de rendimiento en este tercer modelo convierte este método en el mas viable para ser implementado en la práctica.

- Es importante tener en cuenta el hecho de que las tarjetas Maix poseen una ventaja muy grande frente a las tarjetas *Raspberry* debido a que cuentan con *hardware* dedicado específicamente al procesamiento de redes neuronales (FPU). Esto les permite realizar tareas de clasificación y detección de forma mas óptima y por ende mas rápida. También es necesario observar los modelos de redes neuronales, como el que se utilizó en este proyecto, pueden ser implementados en las demás tarjetas de la familia Maix de bajo

costo y consumo energético.

- Las características propias de cada plataforma de desarrollo y cada modelo entrenado permitieron obtener distintos tiempos de ejecución y clasificación al momento de ejecutar los algoritmos. Dichos tiempos fueron determinantes para considerar la viabilidad en la practica de cada uno de los métodos que se trabajaron.Tal como se puede observar en la sección 4.3.2, los tiempos de clasificación de las redes neuronales hacen de este método el mas óptimo para llevar a la practica. A pesar de esto se debe observar que tiempos resultantes de modelos de *machine learning* implementados en la *Raspberry* como lo es $Eigenface$ puede llegar a ser aceptables en un reconocimiento en vivo dentro de la Raspberry Pi examinada en este proyecto. Ahora bien, se podría llegara a mejorar la precisión de dichos modelos entrenándolos con una base de datos mas selecta y surtida en cuanto a condiciones de perspectiva del los sujetos e iluminación ambiental.

6. Recomendaciones

- Debido a que las tarjetas Maix son un sistema relativamente nuevo y no tan conocido, la cantidad de información que es posible encontrar sobre estas tarjetas es bastante limitada. por tal motivo, se ha llevado a cabo la producción de recursos escritos (Torres y Sánchez, 2022) que permitan al lector comprender mejor el funcionamiento de elementos útiles en el desarrollo de algoritmos con las tarjetas Maix como lo es la plataforma Maixhub.

- Uno de los inconvenientes mas frecuentes que se tuvieron al momento de realizar las clasificaciones en tiempo real con las diferentes plataformas con las que se trabajó fue el factor ambiental en el que se encontraba el sujeto en observación. Los algoritmos de *machine learning* fueron especialmente afectados por la iluminación, el ángulo en el que se realizaba la observación y el movimiento del sujeto observado. Por otro lado, el algoritmo de redes neuronales se vio principalmente afectado únicamente por la iluminación, presentando bastante tolerancia al movimiento del objetivo y a variaciones en los ángulos de observación. A pesar de que los resultados de los parámetros de rendimiento en los algoritmos fueron satisfactorios, se especula en este proyecto que dichos valores podrían mejorar utilizando cámaras de mejor calidad al momento de implementar los modelos.

Bibliografía

Ai, B. . (2021, 12 de diciembre). *Intro a las redes neuronales convolucionales - Bootcamp AI.* Consultado el 12 de febrero de 2022, desde https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8

Bazaraa, M. S., Sherali, H. D. & Shetty, C. M. (2013). *Nonlinear programming: theory and algorithms*. John Wiley & Sons.

Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K. & Grundmann, M. (2019). BlazeFace: Sub-millisecond Neural Face Detection on Mobile GPUs.

Briega, L. . R. . E. . (2016, 2 de agosto). *Redes neuronales convolucionales con TensorFlow.* Consultado el 12 de febrero de 2022, desde https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/

, C. (2021, 28 de abril). *GitHub - cabani/MaskedFace-Net: MaskedFace-Net is a dataset of human faces with a correctly and incorrectly worn mask based on the dataset Flickr-Faces-HQ (FFHQ).* Consultado el 25 de febrero de 2022, desde https://github.com/cabani/MaskedFace-Net

Chóez Vera, M. N. & Palma Toledo, C. M. (2021). *Prototipo Web detector de mascarilla mediante RTMP y visión artificial para el control dentro del transporte público del norte de Guayaquil en tiempo de pandemia.* (B.S. thesis). Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Fıésicas ...

Courty, J.-M. & Kierlik, É. (2020). ¿ Cómo funcionan las mascarillas de protección respiratoria? *Investigación y Ciencia*, *524*.

El modelo de redes neuronales. (2021, 17 de agosto). Consultado el 11 de febrero de 2022, desde https://www.ibm.com/docs/es/spss-modeler/SaaS?topic=networks-neural-model

El Naqa, I. & Murphy, M. J. (2015). What is machine learning? *machine learning in radiation oncology* (pp. 3-11). Springer.

Europe, H. T. (s.f.). *APLICACIÓN PARA EL CONTROL DE AFORO*. Consultado en 2022, desde https://www.hanwha-security.eu/es/soluciones-distancia-social/#.

Google. (2021). *API Documentation*. https://www.tensorflow.org/api_docs

Helm, J. M., Swiergosz, A. M., Haeberle, H. S., Karnuta, J. M., Schaffer, J. L., Krebs, V. E., Spitzer, A. I. & Ramkumar, P. N. (2020). Machine learning and artificial intelligence: definitions, applications, and future directions. *Current reviews in musculoskeletal medicine*, *13*(1), 69-76.

Home. (s.f.). Consultado el 21 de febrero de 2022, desde https://google.github.io/mediapipe/

Ifrah, G. (2001). *The universal history of computing: From the abacus to the quantum computer*. John Wiley & Sons, Inc.

Johns Hopkins University. (2021). *COVID-19 Dashboard*. https://coronavirus.jhu.edu/map.html

Maix Go - Sipeed Wiki. (s.f.). Consultado el 22 de febrero de 2022, desde https://wiki.sipeed.com/soft/maixpy/en/develop_kit_board/maix_go.html

Maixhub Model Training Platform Instructions for Use. (s.f.). Consultado el 22 de febrero de 2022, desde https://www.maixhub.com/ModelTrainingHelp_en.html

OpenCV team. (2021). *OpenCV*. https://opencv.org/

Organización Mundial De La Salud [OMS]. (2021). *Brote de enfermedad por coronavirus (COVID-19): orientaciones para el público*. https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/advice-for-public.

Pérez, R. M., Arias, J. S. & Porras, A. M. (2019). Introducción al Aprendizaje Automático con YOLO. *Tecnologıéa Vital*, *3*(6).

Prado, K. S. D. (2018). Face Recognition: Understanding LBPH Algorithm. https://onx.la/3b842

Rafael C. González, R. E. W. (2008). *Digital Image Processing*. Prentice Hall.

Raspberry Pi Foundation - About Us. (2021, 5 de julio). Consultado el 4 de marzo de 2022, desde https://www.raspberrypi.org/about/

Redmon, J. & Farhadi, A. (2017). YOLO9000: better, faster, stronger. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 7263-7271.

Rosebrock, A. . (2021, 17 de abril). *Face Alignment with OpenCV and Python*. Consultado el 25 de marzo de 2022, desde https://pyimagesearch.com/2017/05/22/face-alignment-with-opencv-and-python/

Solomon, C. & Breckon, T. (2011). *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab*. John Wiley & Sons.

Sun, S., Cao, Z., Zhu, H. & Zhao, J. (2019). A survey of optimization methods from a machine learning perspective. *IEEE transactions on cybernetics*, *50*(8), 3668-3681.

SuperDataScience Team. (2017, 3 de agosto). *Face recognition using OpenCV*. https://www.superdatascience.com/blogs/opencv-face-recognition

Torres, A. D. (1996). Procesamiento digital de imágenes. *Perfiles Educativos*, (72).

Torres & Sánchez. (2022, 17 de enero). *GitHub - davidto098/Facemask-detection-codes: In this repository you will find the codes used in Face mask detection project.* Consultado el 17 de enero de 2022, desde https://github.com/davidto098/Facemask-detection-codes

Torres & Sanchez. (2021a, 22 de diciembre). *Dataset de prueba, Google Drive*. Consultado el 18 de febrero de 2022, desde https://onx.la/5aa5d

Torres & Sanchez. (2021b, 22 de diciembre). *Datasets de entrenamiento Machine Learning - Google Drive*. Consultado el 4 de marzo de 2022, desde https://drive.google.com/drive/folders/1iW0GftwkhbwVbuB1LgrIg1Ntnfg0G85p?usp=sharing

Torres & Sanchez. (2022a, 18 de febrero). *Anexos de: Reconocimiento facial con mascarillas.pdf*. Consultado el 18 de febrero de 2022, desde https://onx.la/a0482

Torres & Sanchez. (2022b, 14 de febrero). *datasets_separados.zip*. Consultado el 4 de marzo de 2022, desde https://onx.la/c4c81

Universitat de Barcelona. (2020). *LogMask, una nueva tecnología de inteligencia artificial que detecta el uso de mascarillas*. https://www.ub.edu/web/ub/es/menu_eines/noticies/2020/06/011.html

What can MaixPy do. (s.f.). Consultado el 22 de febrero de 2022, desde https://wiki.sipeed.com/soft/maixpy/en/what_maix_do.html

Wikipedia contributors. (2021, 13 de enero). *Eigenface*. https://en.wikipedia.org/wiki/Eigenface