Universidad Zaragoza
1542

Trabajo Fin de Grado

Detector de mascarillas empotrado de bajo coste
Low cost embedded mask detector.

Autor/es

Javier Benito Celma

Director/es

Ana María López Torres y Francisco José Torcal Milla

Escuela Universitaria Politécnica de Teruel

2021

Agradecimientos

En estas líneas voy a aprovechar para agradecer a toda la gente que ha estado detrás de no sólo este trabajo, sino de toda mi etapa académica, ya que sin ellos nada de esto hubiera sido posible. Porque el trabajo de fin de grado no solo es el propio trabajo, sino todas las personas que tiran de ti para que puedas realizarlo.

Gracias a mis padres, mi hermano y toda mi familia, por aguantar mis enfados, mis bajones y mis cambios de humor durante todo este tiempo, porque habéis sido, sois y seréis mi gran APOYO.

En segundo lugar, quería agradecer a mi segunda familia, mis amigos, Isabel y Estefanía, que han sido utilizados como modelos en este trabajo en varias ocasiones, gracias por toda la paciencia que han tenido conmigo y toda la fuerza que siempre me han transmitido en cualquier objetivo que me he propuesto.

A Carlos, por confiar en mí desde el principio del proyecto y por dejar que instale un prototipo en la piscina municipal. A Stefan, por ayudarme con la impresión 3D de la caja y por estar siempre dispuesto a ayudarme con cualquier cosa, sin olvidarme de todos mis compañeros de clase que he tenido, todos ellos me han aportado algo de lo que soy ahora tanto como persona como estudiante.

Y por último, expresar mi más sentido agradecimiento a la Universidad Politécnica de Teruel y a todos los profesores que he tenido en este largo recorrido, pero en especial dar las gracias a Ana y Fran, que son los que más de cerca han vivido la realización de este proyecto, gracias por estar ahí y por el apoyo incondicional que me habéis dado.

Resumen

En este trabajo se muestra el desarrollo de un dispositivo compacto de bajo coste para la detección automática del uso de mascarillas, el cual se basa en el ordenador de placa reducida Raspberry Pi.

Para ello, se adapta un modelo de detección de mascarillas basado en redes neuronales convolucionales, dotándole de una mayor robustez frente al caso de mascarilla mal colocada. En el momento en el que se reconoce si el usuario porta correctamente o no la mascarilla, se activará un protocolo de actuación en función del resultado de la detección que consiste en un aviso luminoso aviso y sonoro unido a una comunicación vía email. Cabe decir, que esta detección es sensible a la velocidad a la que pasan las personas delante de la cámara, al tipo de mascarilla y al lugar en el que se sitúa la cámara.

Abstract

This dissertation shows the development of a low-cost, compact device which automatically detects the use of face masks. It is based on the single board computer Raspberry Pi. To this end, a model of face mask detection based on convolutional neural networks is adapted to provide greater robustness in the event of the face mask not being properly positioned.

When the device detects whether the user is wearing the face mask correctly or not, a protocol of action will be activated according to the detection result which consists of sound and light warning attached to a email communication. It must be said that this detection is sensitive to the speed at which people pass in front of the camera, the type of mask and the place where the camera is located.

Índice de contenidos

Anexos

Anexo I. Puesta a punto Raspberry Pi 4 Model B

- Instalación del sistema operativo en la Raspberry Pi
- Inicio de la configuración del sistema operativo y de la Raspberry Pi
- Instalación VNC en Raspberry Pi 4B
- Instalación de bibliotecas

Anexo II. Códigos de recopilación de imágenes

- FotosConMascarilla.py
- FotosSinMascarilla.py

Anexo III. Código de entrenamiento del modelo de predicción (Entrenamiento_Modelo.py)

Anexo IV. Código para la detección de mascarillas en tiempo real (Detector_mascarillas.py)

Índice de tablas

Índice de figuras

1 Introducción

A principios del año 2020 comenzó la crisis sanitaria más grave de los últimos cien años debido a la aparición del SARS-CoV-2 (virus que genera el Covid-19). Este hecho propició que el uso de mascarillas faciales en espacios públicos fuera algo habitual (figura 1) y obligatorio en la mayoría de países del mundo y por supuesto también en España[1], ya que su uso sirve como medio de control para reducir la propagación de este virus de manera significativa[2]. Esta utilización obligatoria de las mismas ha producido un aumento de los controles y revisiones aleatorias en lugares de gran afluencia de personas. Y aunque muchas empresas han estado trabajando en el desarrollo de software de reconocimiento facial y de objetos ya desde hace tiempo, se ha incrementado el interés en su uso en relación a la detección de mascarillas.

Figura 1-Multitud con mascarilla en el exterior[3]

Como se prevé que en los lugares públicos de interior, la mascarilla va a ser una prenda que nos va a acompañar durante bastantes meses, serán necesarios sistemas automáticos para la detección de las mismas, como los basados en el uso de la inteligencia artificial y el procesado digital de imágenes.

En los últimos años se ha popularizado el análisis de imágenes para el reconocimiento de patrones o caracterización de objetos, por ejemplo, en el área de la medicina y más concretamente en la detección de enfermedades, aumentando así, las probabilidades de eliminarlas a tiempo o de prevenirlas antes de que aparezcan. En otras materias como la robótica o la videovigilancia también es útil el uso de este tipo de tecnologías.

De acuerdo con esta necesidad, se ha decidido realizar este trabajo de fin de grado, con el que se reconocerá si una persona lleva puesta la mascarilla facial de manera

correcta, utilizando en todo momento hardware de fácil acceso con el objetivo de que el dispositivo final cumpla su cometido a la vez que no conlleve un coste elevado.

1.1 Objetivo del proyecto

El objetivo de este trabajo es el diseño de un dispositivo compacto de bajo coste de detección de mascarillas faciales en tiempo real. Dos de los puntos clave son que el dispositivo sea compacto y, como se ha comentado en el punto anterior, el hardware no sobrelleve un alto coste, sin renunciar a una elevada tasa de reconocimiento. Para poder diseñarlo con estas características se ha determinado que se utilizará un ordenador de placa reducida como la Raspberry Pi 4B y un algoritmo que se basará en librerías open source (software libre) como OpenCV (Open Computer Vision) y Tensorflow.

Tendrá que ser capaz de poder distinguir entre personas que lleven mascarilla (**Positivo)** y que no lleven mascarilla (**Negativo)**, intentando en todo momento perturbar lo menos posible al usuario el cual es monitorizado en dicho control, además de mantener la privacidad del mismo. Asimismo, se ha creado un protocolo en función de la información recibida, que incluye la comunicación de negativos vía email (adjuntando a imagen del mismo) y una serie de avisos acústicos y lumínicos.

1.2 Resumen de la memoria

Teniendo en cuenta todos estos puntos de trabajo se ha establecido esta organización para la memoria.

- **Estado de la técnica**
- **Análisis del software**
- **Análisis del hardware**
- **Algoritmo del trabajo.** Entrenamiento del modelo, técnicas para el manejo de imágenes y protocolo de actuación
- **Análisis de resultados**
- **Puesta en marcha del modelo en un entorno real**
- **Conclusión y posibles líneas de trabajo futuras**
- **Bibliografía**

El primer paso será estudiar el problema propuesto y los posibles errores que pueden llegar a tener las soluciones ya existentes. Se reslizará una breve introducción teórica, tanto en el tema del aprendizaje profundo, como en el de la detección de objetos, además del estudio de la placa Raspberry Pi, sus ventajas y desventajas. Además, en este punto se presentarán algunos de los modelos de detección dirigidos a este problema ya creados con el objetivo de decidir si se adaptan a los objetivos del trabajo.

El siguiente paso será proceder a la presentación de todos los elementos necesarios para la realización de este trabajo tanto de hardware como de software.

A continuación, se trabajará con la obtención del modelo de predicción. Se explicará la recogida de imágenes de personas con y sin mascarilla, además de la justificación de esta recogida. Posteriormente se escogerá un código de entrenamiento ya diseñado y se justificará cada uno de los parámetros elegidos, que se aplican a los modelos entrenados previos analizados para la obtención del modelo definitivo que se pondrá en marcha en la Raspberry. Una vez determinado el modelo final, se creará un protocolo de actuación del sistema según sea positivo o negativo el usuario detectado por el dispositivo.

Una vez obtenido un prototipo, se realizarán una serie de pruebas de campo en la entrada de la piscina municipal de Teruel, en la que se probarán los distintos modelos y se procederá a la valoración de cada uno de ellos, sobre todo en el caso del modelo definitivo.

Para finalizar, se expondrán las conclusiones sobre el trabajo realizado y posibles líneas de investigación futuras.

2 Evolución de las aplicaciones de reconocimiento facial

Como se ha comentado en el punto anterior, uno de los objetivos del proyecto es diferenciar si las personas llevan o no llevan puestas mascarillas faciales. Pero para ello, se tendrá primero que detectar la presencia de la cara de una persona.

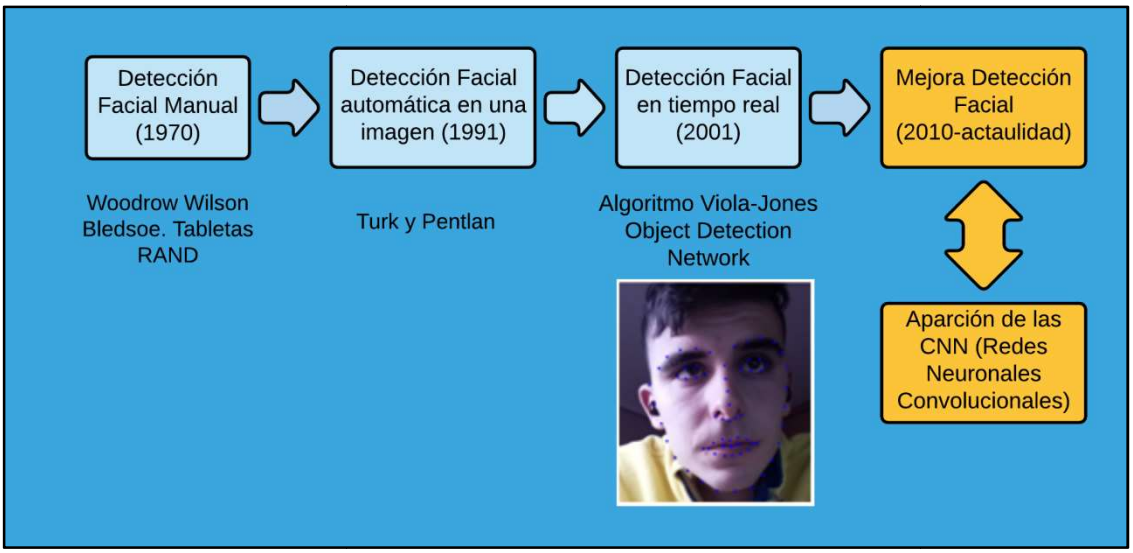

Figura 2-Evolución Reconocimiento Facial

El reconocimiento facial (figura 2), aunque parece una tecnología relativamente nueva, no lo es. El primero que puso en marcha esta disciplina fue Woodrow Wilson Bledsoe a principios de los años 70, con un dispositivo basado en tabletas RAND (tableta digitalizadora que permite dibujar al usuario en la pantalla de la computadora[4]), que era capaz de recrear los diferentes rasgos faciales a través de coordenadas con la ayuda de un lápiz que transmitía impulsos electromagnéticos a una cuadrícula. Aunque esta tecnología produjo un gran avance, tenía grandes carencias ya que por ejemplo era todavía manual. Una década después se consiguió detectar un rostro en una imagen, comenzando así el reconocimiento facial automático Se siguió investigando y desarrollando durante los siguientes años, pero no fue hasta el año 2001 gracias al trabajo de **Paul Viola y Michael Jones,** cuando se obtuvieron las mejores tasas de éxito en relación a la detección de objetos. Y aunque su objetivo principal era la detección de rostros, esta tecnología es capaz también de distinguir una gran variedad de objetos a **tiempo real**[5]**.**

Su algoritmo se basa en la comparación de intensidades luminosas de las distintas regiones rectangulares (Haar-Like features) de una imagen (figura 3), que se calculan de una manera rápida y fácil con el método integral. Estas actúan como clasificadores débiles ya que por sí mismos tienen una probabilidad de dar una predicción correcta de, más o menos, la del azar.

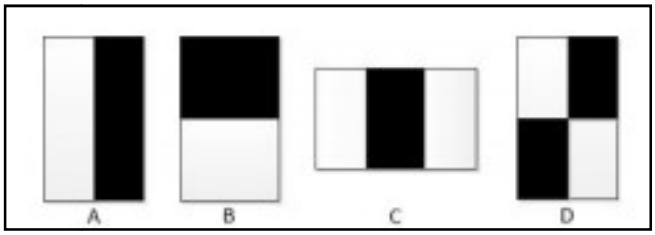

Figura 3- Filtros basados en Haar-like features[6]

Para conseguir elevar la probabilidad de acierto es necesario realizar un entrenamiento supervisado para poder así, crear la cascada de clasificadores. Este entrenamiento se procesa mediante un algoritmo basado en Adaboost (meta-algoritmo que se utiliza para acelerar diferentes tipos de aprendizaje[7][8]), el cual elimina la mayoría de los features (características modeladas), cogiendo sólo las necesarias y construyendo de esta manera una cascada de clasificadores fuertes, adaptados al patrón a identificar, que es lo que determina si en una imagen existe un rostro u objeto determinado[8].

En los últimos años, el reconocimiento facial ha mejorado de una manera ostensible gracias al uso del aprendizaje automático (machine learning). Un caso particular de este conjunto de algoritmos de aprendizaje son las **redes neuronales convolucionales** (CNN), muy populares en el procesamiento de imagen y vídeo[9], ya que dan la capacidad de ver al ordenador. Pueden llegar a encontrar características de las imágenes como bordes, texturas, sombreados… Para poder llegar a entrenar este tipo de algoritmos, habitualmente se utilizan una modalidad del aprendizaje automático como es el **aprendizaje supervisado**, cuyo objetivo es la obtención de un modelo, en este caso los parámetros de una red neuronal, que a partir de una serie de características de una imagen, indique la categoría a la que esa imagen pertenece. Para ello se necesita un conjunto de imágenes conocidas, es decir, etiquetadas según la categoría a la que pertenecen. De esta manera, el algoritmo realiza una serie de predicciones asignando cada una de las imágenes una de las etiquetas dadas. Luego estas predicciones se comparan con los datos reales y se corrigen los parámetros del modelo hasta conseguir

que el error de predicción sea suficientemente bajo. Una vez explicado cómo pueden aprender este tipo de redes, en el siguiente punto se explica el funcionamiento de las mismas.

Todo comienza a partir de los pixeles de una imagen que actúan como las características de entrada del algoritmo, a los cuales se le irán aplicando una serie de filtros (Kernel) adaptados a la detección de unas determinadas características de la misma (figura 4). Se realiza el producto de convolución de cada uno de estos filtros con los píxeles de la imagen obteniéndose una matriz de salida o mapa de características. Esta recorre toda la imagen, es decir, si la imagen de entrada tiene 256 pixeles de ancho y de largo y además 3 colores, RGB, se recorrerán las 256x256x3=196608 neuronas o variables de entrada. Estos mapas de características serán la entrada de una nueva capa de la red neuronal dando lugar a su vez a nuevos mapas de características (figura 5).

En este sentido, el factor diferencial del uso del aprendizaje profundo (deep learning) con estas redes, es que estas van aprendiendo el valor de estos filtros según los patrones que vaya detectando de las propias imágenes de entrada. De esta manera se conseguirá diferenciar si alguien lleva o no lleva gafas, tiene ojos azules u ojos verdes y así con infinitas posibilidades.

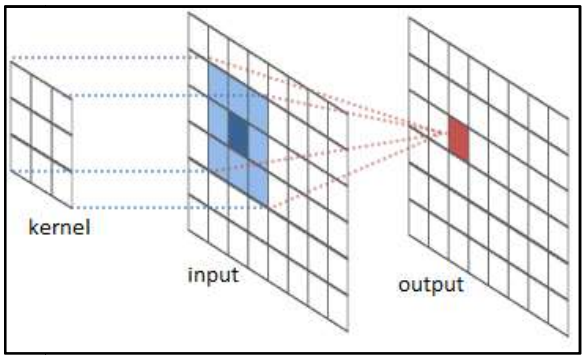

Figura 4- Filtros Kernel[10]

Estas redes se forman aplicando cientos de filtros en las distintas capas de la red neuronal. Además, conforme vaya aumentando el número de capas, también se tiene que ir disminuyendo el tamaño de la imagen, con el objetivo de reducir el número de neuronas (Subsampling o pooling). Así se obtendría la red neuronal convolucional para poder clasificar las imágenes[11](figura 5).

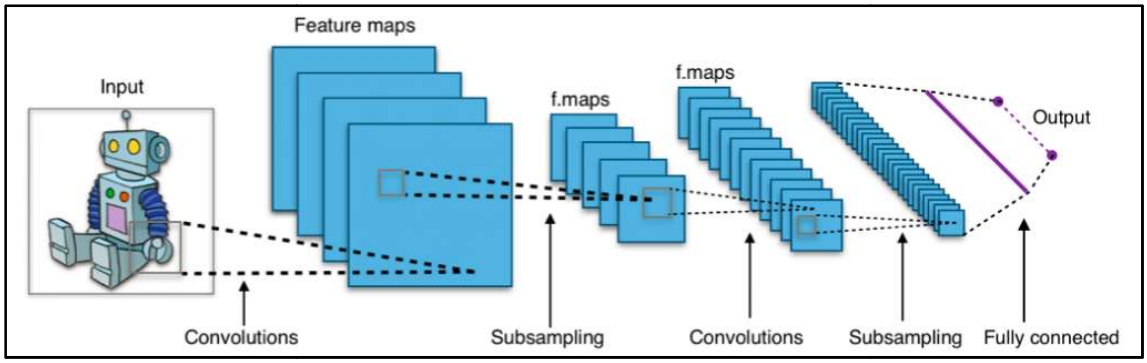

Figura 5-Red Neuronal Convolucional (CNN)[11]

3 Análisis del software

En este apartado se va a proceder a explicar el software utilizado durante el proyecto.

Software	Detalle
Sistema operativo	Raspbian GNU 10
Lenguaje de programación	Python 3.7.3
Entorno de programación	Thonny Python IDE

Tabla 1- Listado de software utilizado

En primer lugar, en la Tabla 1 se describe el software instalado en la Raspberry Pi, además del entorno de programación y lenguaje utilizado en este proyecto. En el Anexo I, en el punto de instalación del sistema operativo se detallan los pasos que se han seguido para poder instalar el sistema operativo seleccionado. Además, se han utilizado una serie de bibliotecas necesarias para la realización del trabajo que son las que se observan en la Tabla 2.

Bibliotecas	Uso
OpenCV (4.1.0), Imutils(0.5.4)	Procesado de imagen y vídeo
Tensorflow(2.1.0),Scikit-learn(0.24.1)	Aprendizaje automático
Numpy	Manejo Matrices
Matplotlib	Dibujo gráficas
RPLCD	LCD
Smtplib,imghdr	Correo Electrónico
PiCamera	Cámara

Tabla 2- Bibliotecas instaladas en la Raspberry Pi

En el Anexo I, en la parte de Instalación de bibliotecas en Raspberry quedan explicadas la manera y versión a utilizar de cada una de ellas. Además de la puesta en marcha de la cámara Raspi, del bus I2C (LCD) y VNC server(necesario para el control remoto de la placa).

Por otro lado, el coste computacional de entrenar un modelo de deep learning es muy alto, ya que hay que almacenar todas las imágenes del entrenamiento, realizar el preprocesamiento de los datos a utilizar y el entrenamiento de la última parte de la red neuronal (clasificador), como se va a explicar posteriormente. Todos estos pasos conllevan un gran uso de recursos, de los cuales carece un ordenador monoplaca como la Raspberry, que a partir de una cantidad de datos pequeña (más o menos unas 200 imágenes), puede calentar en demasía el procesador haciendo que este se recaliente (T>= 80ºC) y disminuya su velocidad o incluso, pueda llegar a quemarse (T > =85ºC). Por lo que, es necesario realizar la tarea de entrenamiento del modelo de predicción en un ordenador auxiliar a la Raspberry.

3.1 Organización del algoritmo

El algoritmo del trabajo queda dividido en varios archivos que son necesarios para la realización completa del trabajo y su posterior puesta en marcha. Su contenido se explicará en posteriores apartados y queda reflejado en la figura 6.

- **Detector_mascarilla.py**. Programa para la detección de personas con mascarilla y sin mascarilla
- **Modelo_mascarillas.model.** Modelo de predicción entrenado
- **Entrenamiento_modelos.py.** Programa para el entrenamiento del modelo
- **FotosConMascarilla.py.** Programa para la colección de fotos de positivos, que se guardarán en el dataset_aux.
- **FotosSinMascarilla.py.** Programa para la colección de fotos de negativos que se guardarán en el dataset_aux.
- **Imágenes resultados.** Capturas de imágenes de negativos.
- **Detector_caras.** Modelo de detección de caras
- **Historial.txt.** Archivo txt en el que se guarda un historial de todos los positivos y negativos que se hayan detectado
- **Dataset.** Conjunto de imágenes que serán entrenadas desde **Entrenamiento_modelos.py**
- **Dataset_aux.** Carpeta dónde se guardan todas las imágenes recolectadas para su posterior selección.

En la figura 6 se pueden observar los pasos que se han seguido, para la realización de este proyecto:

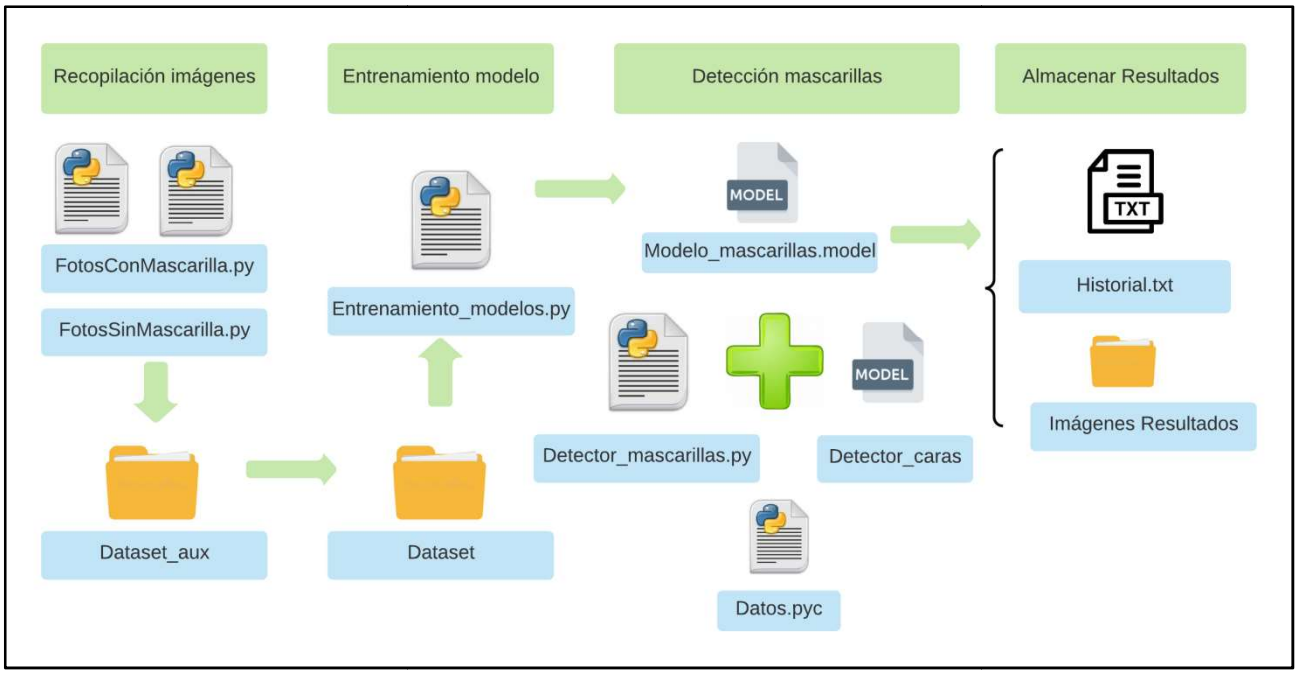

Figura 6-Diagrama de flujo del software

En resumen, se han recopilado las imágenes con los archivos 'FotosConMascarilla.py' y 'FotosSinMascarilla.py' y automáticamente se guardan en la carpeta 'Dataset_aux'. Estas imágenes han sido copiadas a la carpeta 'Dataset' para entrenar los distintos modelos de detección de mascarillas, en este caso se genera Modelo_mascarillas.model. Con el modelo entrenado, el Detector_caras.model y el archivo donde se han guardado el correo desde el que se envía el aviso y su contraseña (Datos.pyc), se ejecuta el programa 'Detector_mascarillas.py' con el que se detecta la presencia de mascarillas en las caras de los usuarios y se pone en marcha el protocolo de actuación. Por último, se genera un historial de positivos y negativos ('Historial.txt') además de sus correspondientes capturas de pantalla ('Imágenes Resultados').

4 Análisis del hardware

En este se explicarán todos los componentes hardware utilizados en el proyecto.

4.1 Conexión del prototipo

Ya que uno de los objetivos de este trabajo es diseñar un dispositivo de bajo coste, se ha tenido en cuenta el precio de cada uno de los componentes. Además, se especificará para cada uno de ellos si se necesita para el modelo final o sólo para su puesta en marcha y/o ensayo.

En este caso, el prototipo final que se ha realizado para hacer las pruebas se muestra en la figura 7.

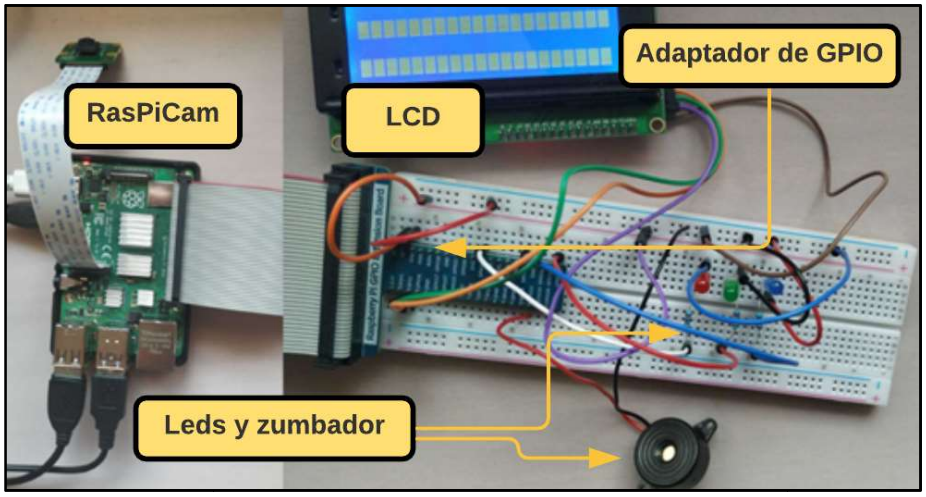

Figura 7- Prototipo de pruebas

En relación a la placa principal del trabajo, se ha escogido la última versión de los ordenadores de placa reducida de Raspberry Pi, el modelo 4B, con 4 gigas de RAM (figura 9).

Por otro lado, para la toma de imágenes, se ha elegido la RasPiCam V2 de 8Mpx (figura 8), la cual se conecta a un zócalo existente en la zona de salidas de la placa. Este tipo de cámara se fabrica esencialmente para su uso con Raspberry y son compatibles en todo tipo de placas de esta marca (Raspberry 1,2,3 y 4)[12].

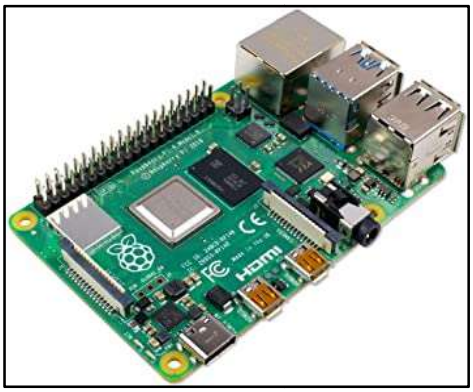

Figura 9- Placa Raspberry Pi 4B

Figura 8- Instalación RaspiCam

Además, se han puesto en la misma 4 disipadores en los componentes que pueden llegar a calentarse y un estuche con un ventilador para refrigerar la placa (figura 10).

Figura 10-Instalación estuche y ventilador

Para la información de salida, se han utilizado unos leds (Verde=Positivo, Rojo=Negativo y Azul=Led de activación de la cámara) informativos, se ha instalado una pantalla LCD y un zumbador para tener tanto aviso sonoro como visual (figura 7). Los tres leds y el zumbador se han conectado a 4 salidas digitales y por el contrario, la pantalla se ha conectado a la placa por el bus I2C, tal y como se aclara en el Anexo I del presente trabajo.

En la tabla 3 se recogen todos los elementos hardware utilizados en este proyecto. Se diferencia entre los componentes usados para el prototipo final, para el prototipo de pruebas y para la configuración de la placa base. Se incluyen los precios y unidades de cada producto. Se han utilizado los materiales más baratos debido a que uno de los objetivos del proyecto es que el dispositivo sea de bajo coste.

Componentes	Observaciones	Cantidad	Precio total	Prot.Final	Prot.Pruebas
Placa Procesadora	Raspberry Pi Model 4B 4gb RAM	1	58 €	Sí	Sí
Cámara	Módulo Cámara Raspberry Pi V2.1 8Mpx	1	30 €	Sí	Sí
Tarjeta SD	64 gb	1	11€	Sí	Sí
Teclado	Conexión USB	1	12 €	Config	Config
Ratón	Conexión USB	1	8 €	Config	Config
Interfaz Multimedia	Micro HDMI- HDMI	1	4,25 €	Config	Config
Cables	Adaptador GPIO Raspi + Cables	1 Adaptador + 10 Cables	4 €		Sí
Resistencias	220 Ω	3	1 €	Sí	Sí
Leds	1 Verde 1 Rojo 1 Normal	3	2 €	Sí	Sí
Pantalla auxiliar	LCD 4x20	1	10 €	Sí	Sí
Placa PCB perforada		1	2 €	Sí	
Zumbador	Z de continua	1	1 €	Sí	Sí
Ventiladores		3	6 €	Sí	

Tabla 3-Elementos hardware

Ni el dispositivo final ni el prototipo tienen la necesidad de lo cables USB del teclado y del ratón, ni tampoco del cable micro-HDMI HDMI para su uso habitual, ya que el detector se podrá iniciar desde un ordenador o smartphone auxiliar gracias al protocolo VNC. Estos tres accesorios sólo se han utilizado durante la realización del proyecto.

En resumen el **coste** de fabricación del dispositivo final ha sido de un total de **125 euros**.

4.2 Conexión del dispositivo final

Con el objetivo de poder instalarlo en un entorno real, además de mejorar su compactibilidad, se ha diseñado una caja para que en ella quepan todos los componentes, además de una serie de ventiladores con el objetivo de que la placa procesadora esté totalmente refrigerada, ya que esta puede sobrecalentarse,

disminuyendo notablemente el rendimiento de la misma. Esta caja se ha fabricado mediante impresión 3D y ha sido diseñada específicamente para este trabajo.

5 Modelo de detección

El objetivo inicial de este proyecto es construir un sistema de bajo coste capaz de diferenciar en tiempo real si una persona lleva una mascarilla facial puesta o no. Para ello, se localizó un modelo matemático de detección de mascarillas y se procedió a su implementación en la placa procesadora.

Como se va a explicar en el siguiente apartado, el modelo seleccionado, un modelo binario que considera salida positiva rostro con mascarilla y salida negativa rostro completamente sin mascarilla, no resulta satisfactorio ya que considera que una persona lleva mascarilla, aunque su nariz sea visible. El crear un nuevo modelo a partir del entrenamiento de una red neuronal convolucional, no fue uno de los objetivos iniciales de este proyecto, ya que este proceso conlleva un gran coste tanto computacional como temporal. Esto es debido a que se necesitaría una gran cantidad de datos, del orden de varios millones de imágenes en este caso y, por ello el tiempo que se tardaría en entrenar un solo modelo sería de varios días. Aunque con una buena GPU (unidad de procesamiento gráfico) se podría disminuir este tiempo.

Se podría haber implementado el sistema con este modelo siendo consciente de sus limitaciones, pero se ha optado por mejorarlo modificando los parámetros de las últimas capas de neuronas para que reconozca como nuevos casos negativos el llevar la mascarilla a medio poner. Por tanto, se ha necesitado estudiar el comportamiento básico de las redes neuronales convolucionales y su entrenamiento.

Se utiliza la técnica de **aprendizaje por transferencia** (transfer learning), la cual es una técnica de aprendizaje en la que se selecciona una red pre-entrenada, y se vuelve a entrenar con un nuevo conjunto de imágenes que incluya los positivos y negativos que se desean. En este entrenamiento solo se modificarán los parámetros asociados a las últimas capas que son las que son más dependientes del problema concreto a modelar.

Esta parte de la red es mucho menos costosa de entrenar ya que se parte de una red ya entrenada con millones de datos. En el campo de la visión por computador se utiliza de manera habitual la base de datos **ImageNet,** que contiene un total de 15 millones de imágenes etiquetadas con más de 22000 categorías[13].

Así que, por todo esto, será necesaria una recogida de imágenes de personas con y sin mascarillas. A continuación, se analiza un modelo de partida con el objetivo de

encontrar los fallos más comunes y proponer mejoras que se podrían implementar en el mismo.

5.1 Modelo previo

Como se ha comentado anteriormente y dada la situación pandémica que está viviendo la sociedad actual, se han implementado algunos modelos de detección de mascarillas, uno de los cuales se va a presentar en el siguiente apartado, descargado desde el siguiente enlace:

https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/

La mayoría de estos modelos parten de los pesos proporcionados por el entrenamiento de los datos de la base de datos ImageNet, ya que las primeras capas de la red se centran en el reconocimiento de parámetros básicos de la imagen: líneas con diferente orientación, esquinas…, que son comunes a todos los modelos de predicción sobre imágenes. Sin embargo, para entrenar la última parte de la red neuronal, se utilizan imágenes de personas sin mascarilla, es decir, totalmente con la cara al descubierto y con la mascarilla perfectamente puesta, tal y como se aprecia en la figura 12.

Figura 12-Imágenes utilizadas en el modelo previo

En cambio, estas no son las dos únicas posiciones en las que la gente lleva puesta su mascarilla habitualmente. Existen otras tres posibilidades, por debajo de la barbilla, por debajo de la boca o entre la boca y la nariz (figura 13), todas ellas conllevan la misma consecuencia: No protección frente a inhalación de posibles virus, con lo que todos estos ejemplos serían igual a no llevar la mascarilla ya que se dejaría nariz y/o boca al descubierto.

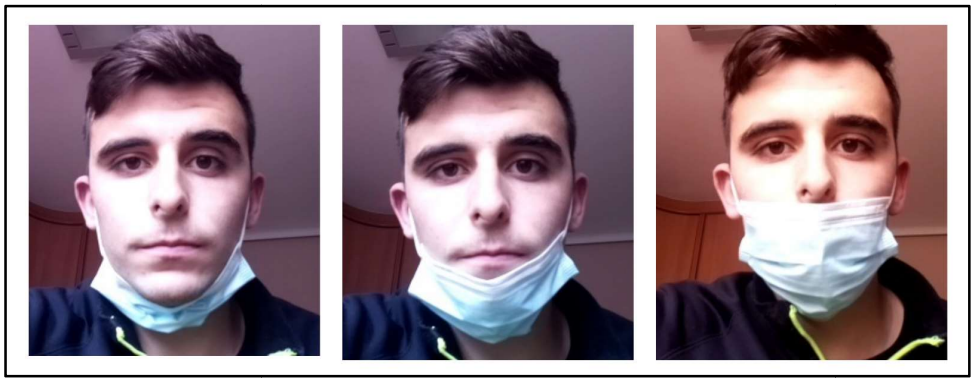

Figura 13-Ejemplos mascarillas mal puestas

Estos ejemplos sumados a la persona sin mascarilla totalmente deberían dar como resultado negativo y en el siguiente punto se verá como se comporta el modelo inicial frente a esa casuística.

- Resultados y conclusiones del modelo previo

Haciendo diferentes pruebas obtenemos los siguientes resultados:

Figura 14-Resultados modelo previo

Después de realizar varias pruebas (figura 14) con diferentes usuarios (variando sexo y edad), entornos y tipos de mascarillas, se puede observar que en el momento que esta última queda entre el labial superior y la parte inferior de la nariz (que como se acaba de comentar, estaría mal colocada), se obtiene un resultado POSITIVO, razón por

la que se ha decidido modificar la red con el entrenamiento de dos nuevas capas, con el objetivo de mejorar los resultados obtenidos.

Además, conforme la distancia entre la cámara y el usuario aumenta, lo hace también este problema de una forma considerable, pero esto es debido a la calidad de la cámara y a la placa procesadora, con lo que ocurrirá en todos los modelos entrenados.

En los siguientes apartados se explicará cómo se ha modificado la red y se ha renovado la base de datos para entrenar el clasificador con estas nuevas imágenes.

Primero es necesario seleccionar las imágenes con las que entrenar el clasificador que diferencia entre usuarios con mascarilla y sin mascarilla. Después se ha escogido el código que se ha utilizado para su entrenamiento. **El enlace para acceder a este algoritmo es el siguiente:**

https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/

5.2 Adquisición de imágenes y justificación

Para mejorar el clasificador y darle más robustez al modelo en los casos en los que el usuario lleva la mascarilla mal puesta se ha decidido renovar las imágenes del dataset.

5.2.1 Elección de imágenes

- **Posición de la mascarilla**

Para ello, se han ido recogiendo imágenes de distintas personas con la mascarilla mal puesta (por debajo de la nariz) y con la misma bien puesta (por encima de la nariz). De esta manera, como el clasificador identifica los patrones de las imágenes etiquetadas (ConMascarilla y SinMascarilla), en el momento que el usuario tenga la mascarilla por debajo de la nariz, el detector debe comunicar el negativo, si el modelo está bien entrenado. Es decir, la respuesta debe ser igual en el caso en el que el usuario no lleve la mascarilla o la lleve mal puesta (figura 15).

Figura 15-Recopilación imágenes

- **Fondo de la imagen**

Además, para aumentar la calidad del clasificador se han recogido las imágenes con el mismo fondo blanco para todas las imágenes (figura 15), disminuyendo así la complejidad de las capturas en relación a bordes, colores… Esto es válido porque se va a tener el mismo fondo cuando el dispositivo se instale en un entorno real y no se tendrán problemas con el mismo. Si no fuera el caso, habría que entrenar el sistema con personas delante de fondos diferentes, ya que puede ocurrir que, si el fondo es parecido o igual al de una serie de imágenes de personas sin mascarilla, el dispositivo comunique que ha sido un negativo independientemente de si lleva o no lleva la mascarilla, ya que lo que se identificaría sería el fondo.

- **Ángulo de visión**

Otro parámetro que se ha tenido en cuenta es el ángulo desde el cual el usuario pasará por el control. Se ha tenido en cuenta que, en cualquier entrada, aunque se entre por una puerta fijada y el dispositivo esté justo en frente de la misma, los usuarios no siempre entran mirando al frente. Debido a esta situación se ha optado por recopilar imágenes también con las personas mirando ligeramente (aproximadamente unos 15º) hacia ambos lados para tener cubierta esta posibilidad tal y como se aprecia en la figura 16.

Figura 16- Ángulo de imagen

- **Tipo de mascarilla**

Por último, se ha considerado que los usuarios pueden llegar a llevar distintas mascarillas de distintas formas y colores. Se ha tenido en cuenta que la mayoría de personas llevan mascarillas higiénicas y FPP2, y los colores más comunes de las mismas son colores azul, blanco y negro, por ello las personas que aparecen en las imágenes recogidas llevan estos tipos de mascarilla.

5.2.2 Recopilación imágenes

Con el objetivo de automatizar la recopilación de imágenes se ha diseñado dos tipos de código.

- **FotosConMascarilla.py**
- **FotosSinMascarilla.py**

Los dos algoritmos se han diseñado para capturar imágenes en el momento que el usuario lo indique. La única diferencia está en el directorio en el que se han guardado las imágenes capturadas.

5.2.2.1 Programa de toma de imágenes

Para la captura de imágenes, se ha utilizado la librería picamera, propia de las cámaras de Raspberry, la versión 1.10[14].

Para realizar capturas de forma continua (figura 17) se ha utilizado el comando **capture_continuos()** el cual va capturando imágenes (que se eliminan después de mostradas y si se da el caso, después de guardadas) desde la cámara de forma infinita, se van guardando en la salida de la función y se va mostrando con la función de OpenCV **cv2.imshow(),** no sin antes producir la matriz tridimensional RGB de la captura no codificada para poder mostrarla por pantalla. Este proceso se realiza con la función **PiRGBArray()**.

Además, para poder mostrar de forma infinita la captura de imágenes, se van eliminando las salidas de la cámara (captura) después de la muestra por pantalla de las mismas, con el comando **truncate(0).**

```
while True:
    #Captura de imágenes continua
    for frame in Camara.capture_continuous(Captura_Camara, format="bgr", use_video_port=True):

        frame = frame.array

        #Muestra por pantalla
        cv2.imshow("Presiona la c para tomar una captura de imagen", frame)

        #Eliminación de la salida después de su muestra por pantalla
        Captura_Camara.truncate(0)
```

Figura 17-Captura continua de imágenes

En relación a la captura de imagen y guardado en el directorio deseado, se ha optado por utilizar el comando **cv2.waitKey(1)** que muestra cada fotograma durante 1ms todo el tiempo hasta que se presiona a la 'c' para realizar una captura de la imagen mostrada (**imwrite()**) y el guardado en el directorio o si, por el contrario se presiona 'esc' para salir del bucle y de la captura de imágenes (figura 18).

```
        k = cv2.waitKey(1)

        #Si se presiona Escape se sale del bucle
        if k == 27:
            break

        #Si se preta c se realiza la captura y guardado de la imagen
        elif k == ord('c'):

            #Nombre de la imagen y directorio en el que se guarda
            img_name = "dataset_aux/FotosSinMascarilla/image_{}.jpg".format(Numero)

            #Captura de la imagen
            cv2.imwrite(img_name, frame )
            print("{} se ha guardado de forma satisfactoria".format(img_name))

            #Aumento del número de archivos en el directorio
            Numero += 1
            print(f"El numero de archivos en el directorio es de {Numero}")

    if k == 27:
        print("Saliendo...")
        break


cv2.destroyAllWindows()
```

Figura 18-Captura y guardado de la imagen

Los códigos completos de ambos algoritmos se encuentran en el Anexo II del presente proyecto.

5.2.3 Imágenes recopiladas

En total se han recogido un total de 1245 imágenes. Sin embargo se han descartado un porcentaje considerable debido a que algunas salen movidas, en otras se supera el

grado de inclinación que se ha establecido… Otras imágenes recopiladas han sido eliminadas porque en principio, estas capturas se realizaban con unos fondos que no eran completamente iguales. De manera que el total de imágenes recopiladas en cada categoría que se han utilizado para el entrenamiento del modelo son:

- **Personas Con Mascarilla:** 414 imágenes
- **Personas Sin Mascarilla (mal puesta):** 462 imágenes

5.3 Entrenamiento del modelo

Con el código que se indica en el apartado 5.1 del presente proyecto y con las imágenes recogidas anteriormente se procede al proceso de entrenamiento en el que se han variado, respecto del programa de entrenamiento original los siguientes parámetros: el número de **epoch** o veces que el conjunto de datos completo pasa por el algoritmo para su entrenamiento, la **tasa de aprendizaje** o ritmo de cambio de los diferentes parámetros, el **número de imágenes** y el **tamaño del grupo**(batch size) de imágenes que pasan por la red neuronal antes de actualizar los parámetros.

A continuación, se van a explicar los distintos pasos de este programa de entrenamiento

5.3.1 Estructura del modelo implementado

Se ha cargado la red neuronal convolucional (53 capas) MobilenetV2 entrenada con los datos de la base de datos Imagenet (figura 20),esta estructura está diseñada y optimizada para su uso con dispositivos móviles [15]. Se ha especificado el tamaño de la imagen de entrada (**input_tensor**) que en este caso ha sido (224,224,3), que son el alto, ancho (número de píxeles) y número de colores (RGB). De esta red, no se va a utilizar la última capa, una capa completamente conectada correspondiente a la parte superior de la red (**input_top**).

```
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224,3)))
```

Figura 19- Utilización arquitectura MobileNetV2

Además, como esta parte del modelo ya está entrenada previamente, se indica que no habrá que entrenar estas capas (Figura 21).

```
for layer in baseModel.layers:
    layer.trainable = False
```

Figura 20-No entreno capas MobilenetV2

Una vez ha quedado definida la parte de la red neuronal convolucional a partir de la red de partida, se han definido las capas superiores de nuestra red (figura 22), que serán un total de 6 capas. Primero se ha utilizado una capa Pooling, que se basa esencialmente en reducir las dimensiones de las imágenes de entrada con el objetivo de reducir el tiempo de entrenamiento. En este caso se han reducido la escala del alto y del ancho 7 veces (**AveragePooling2D**). Después se ha generado una capa la cual ha aplanado la matriz de entrada, obteniendo así una matriz de una sola dimensión o vector (**Flatten)**. La siguiente capa que se ha creado tiene 128 neuronas totalmente conectadas (**Dense**) que utiliza una función de activación tipo 'relu'. Esta lo que hace es obviar las entradas negativas a esta capa, dándoles valor nulo y en cambio los valores positivos de entrada se mantienen con el mismo valor. En resumen, es una función de activación de rectificación lineal. En el siguiente paso, se ha establecido una capa regularizadora que elimina la mitad de las neuronas (**Drouput)** para evitar el sobreajuste (efecto de sobreentrenar un algoritmo de aprendizaje con unos ciertos datos para los que se conoce el resultado deseado) de los datos de entrenamiento. Después de esta capa se establece una capa de salida con dos neuronas, que usa la función de activación 'Softmax' para realizar la clasificación entre ambas clases.

```
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel)
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(128, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(2, activation="softmax")(headModel)
```

Figura 21-Estructura del modelo

Una vez se ha establecido la estructura de las 6 capas del clasificador, se han situado a la salida de la parte convolucional de la red neuronal MobilenetV2 tal y como se aprecia en la figura 23.

```
model = Model(inputs=baseModel.input, outputs=headModel)
```

Figura 22-Colocación capas del clasificador

De forma que la estructura del modelo que se utilizará es la reflejada en la figura 24.

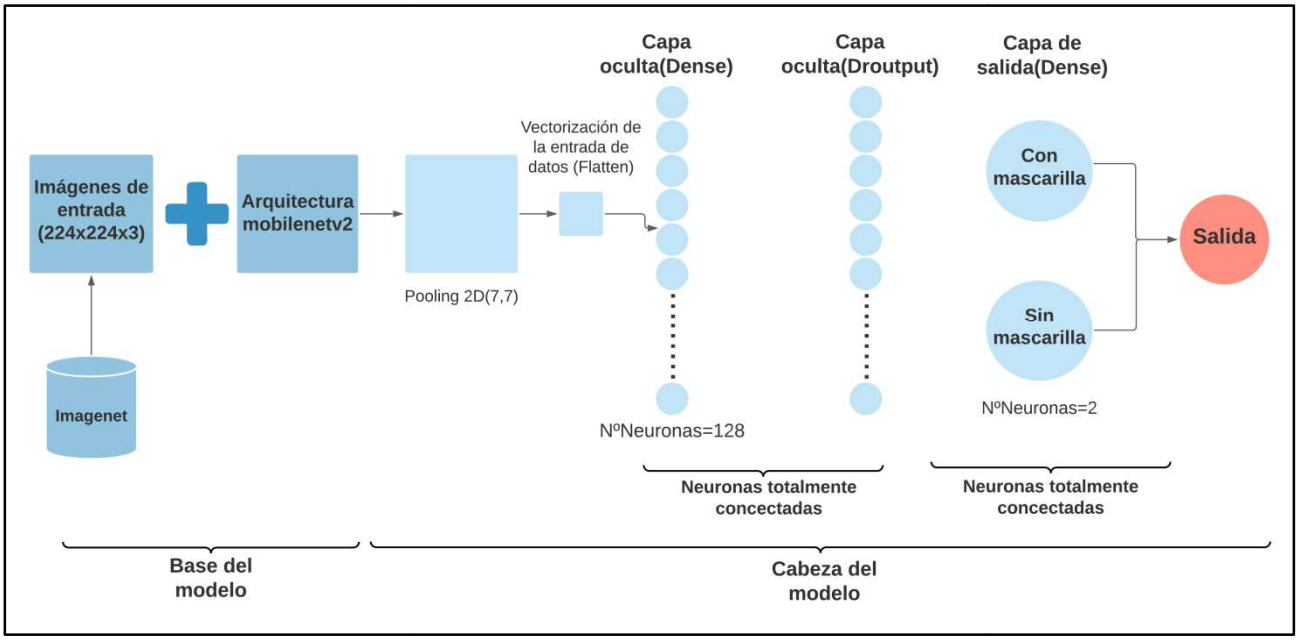

Figura 23-Estructura del modelo

5.3.2 Entrenamiento del modelo

En los algoritmos de machine learning el objetivo principal es reducir la diferencia entre el resultado previsto y el resultado real. A partir de esta diferencia se define la función de pérdidas o de coste del algoritmo. Para reducir esta pérdida se utiliza un algoritmo **optimizador**, que lo que hace es, como su propio nombre indica optimizar los valores de los pesos (o parámetros), modificando su valor para reducir el valor de la función de pérdidas. En este caso, se ha escogido el optimizador **Adam** debido a que es el más eficiente en lo que se refiere a modelos relacionados con visión por computador, además de que es muy fácil de configurar[16].

5.3.2.1 Definiciones previas

Por otro lado, para comprobar que los modelos funcionan correctamente, a parte de las posteriores pruebas de campo, después del entrenamiento se han obtenido cuatro parámetros que determinan la fiabilidad del mismo: precision, recall, accuracy y F1-score. Estos resultados se han obtenido con la función propia de la librería Keras

classification_report(), pero para la interpretación de estos se necesitan comprender las siguientes medidas que definen lo que se denomina la matriz de confusión.

- **True Positives (TP):** Valores **correctamente clasificados como valores positivos**, es decir, una foto de una persona con mascarilla ha sido clasificada como persona con mascarilla.
- **False Positives (FP):** Valores **erróneamente clasificados como valores positivos**, en este caso, una foto de persona sin mascarilla se ha clasificado como persona con mascarilla
- **True Negatives (TN):** Valores **correctamente clasificados como valores negativos**, es decir, una foto de una persona sin mascarilla ha sido elegida en la clasificación de persona sin mascarilla.
- **False Negatives (FN):** Valores que **erróneamente clasificados como valores negativos**, una foto de persona con mascarilla ha sido clasificada como una imagen de persona sin mascarilla.

Visto el significado de estos parámetros, se pueden explicar los parámetros que determinan la calidad del modelo de predicción entrenado.

- **Accuracy:** relación entre predicciones correctas y predicciones totales del modelo.
- **Precision:** relación entre el número de resultados positivos y el número de resultados positivos predichos por el clasificador. Esta definición es para las imágenes de personas con la mascarilla:

$$\boldsymbol{Precision} = \frac{TP}{TP + FP}$$

  - Si se refiere al dataset de personas sin mascarilla y al tratarse de un modelo binario se define de la siguiente manera:

$$\boldsymbol{Precision} = \frac{TN}{TN + FN}$$

- **Recall:** relación entre el número de resultados positivos y el número de resultados previstos como positivos.

$$\boldsymbol{Recall} = \frac{TP}{TP + FN}$$

  - Si se refiere al dataset de personas sin mascarilla (negativos), se define de la siguiente manera

$$\boldsymbol{Recall} = \frac{TN}{TN + FP}$$

- **F1-score:** es la media armónica entre la precisión y recall, informa acerca de la calidad del clasificador, conforme este valor sea mayor, mejor será el **rendimiento del modelo.**

$$\boldsymbol{F1 - score} = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

5.3.2.2 Resultados de los entrenamientos

- **Entrenamiento nº1**

Se ha optado por realizar todos los entrenamientos con un tamaño del grupo de 30, ya que cuanto mayor sea mayor será el tiempo de entrenamiento, y al no tener un GPU puede ser contraproducente aumentar este valor. Por otro lado no se disminuye más, porque cuanto más pequeño sea, menos precisa será la estimación del gradiente. La tasa de aprendizaje se ha establecido en el valor que venía por defecto. En posteriores entrenamiento se han modificado estos valores. De manera que los hiperparámetros escogidos han sido los reflejados en la tabla 4.

Hiperparámetros	
Nº de epoch	10
Tasa de aprendizaje	1E-04
Tamaño de grupo	30

Tabla 4-Hiperparámetros del entrenamiento nº1

Los resultados del entrenamiento 1 han sido los mostrados en la tabla 5:

Resultados	Precision	Recall	F1-score
ConMascarilla	0,84	0,88	0,86
Sin mascarilla	0,89	0,80	0,84
Accuracy			0,85

Tabla 5-Resultados del entrenamiento nº1

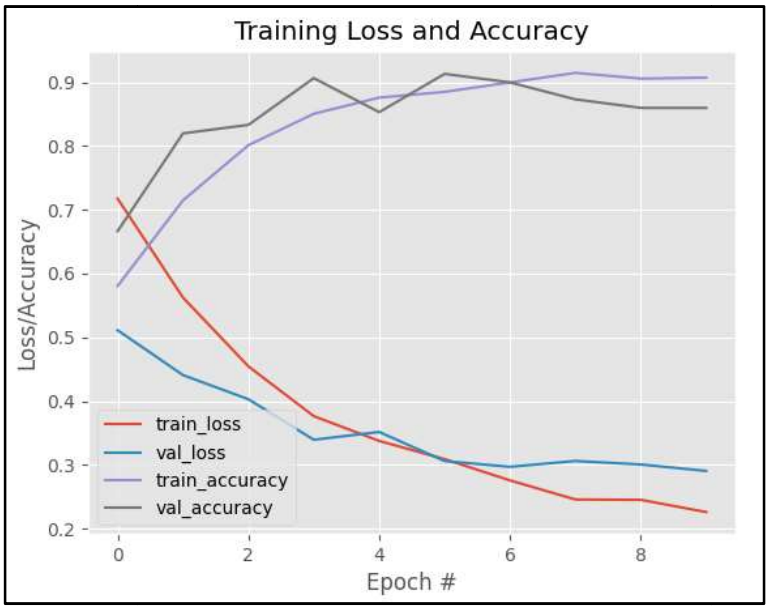

Figura 24-Resultados del entrenamiento nº1

Como se aprecia en los resultados y gráfica del entrenamiento (figura 24), la red neuronal necesita aprender, para mejorar la calidad del modelo y para ello se aumentan el número de veces que los datos recorren esta red hasta 15 en el segundo entrenamiento.

- **Entrenamiento nº2**

Hiperparámetros	
Nº de epoch	15
Tasa de aprendizaje	1E-04
Tamaño de grupo	30

Tabla 6-Hiperparámetros del entrenamiento nº2

Los resultados del entrenamiento 2 han sido los mostrados en la tabla 7:

Resultados	Precision	Recall	F1-score
ConMascarilla	0,87	0,89	0,88
Sin mascarilla	0,9	0,88	0,89
Accuracy			0,88

Tabla 7-Resultados del entrenamiento nº2

La gráfica del presente entrenamiento muestra los valores de pérdida y precisión a lo largo del mismo (figura 25). A través de esta gráfica se puede ver que el entrenamiento es correcto, ya que tanto para el entrenamiento como para la validación la exactitud sigue creciendo y el error disminuyendo.

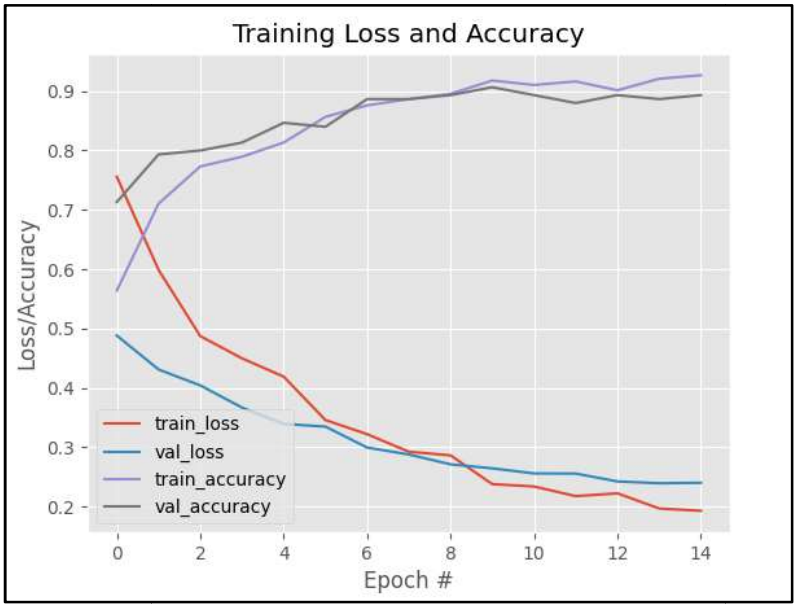

Figura 25-Gráfica del entrenamiento nº2

- **Entrenamiento nº3**

Para mejorar los datos del entrenamiento anterior se ha aumentado el número de epoch hasta 20 para que las imágenes pasen más veces más por la red neuronal y pueda aprender más (tabla 8).

Hiperparámetros	
Nº de epoch	20
Tasa de aprendizaje	1E-04
Tamaño de grupo	30

Tabla 8-Hiperparámetros del entrenamiento nº 3

Los resultados del entrenamiento 3 han sido los mostrados en la tabla 9:

Resultados	Precision	Recall	F1-score
ConMascarilla	0,92	0,96	0,94
Sin mascarilla	0,97	0,92	0,95
Accuracy			0,94

Tabla 9-Resultados del entrenamiento nº3

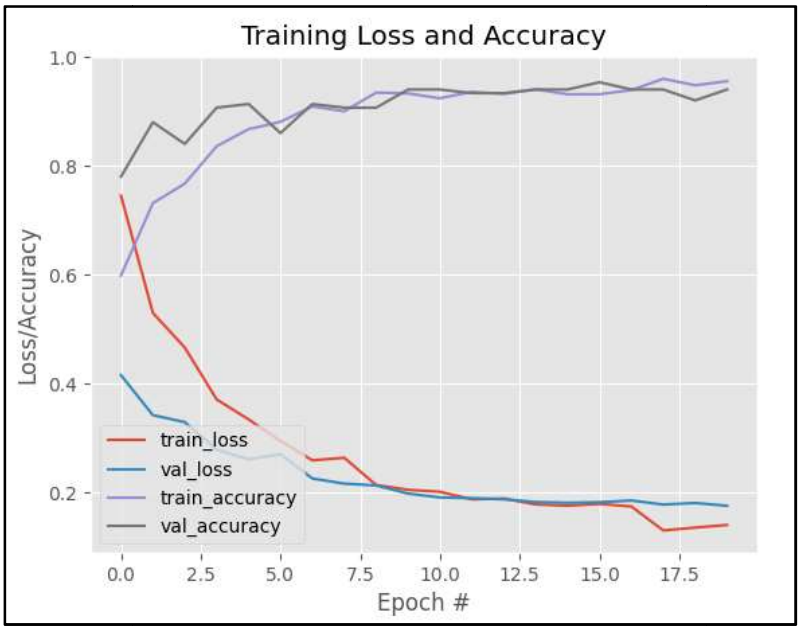

Figura 26-Gráfica del entrenamiento nº3

La gráfica del presente entrenamiento (figura 26) y los resultados del mismo muestran una mejora de la calidad del modelo entrenado.

- **Entrenamiento nº4**

Observando los resultados de los anteriores entrenamientos se ha llegado a la conclusión de que la tasa de aprendizaje escogida es buena, ya que la fúnción de pérdida es muy pequeña. Sin embargo, cabe la posibilidad de hacerla más insignificante, disminuyendo la tasa de aprendizaje, pero el entrenamiento durará más tiempo

Si por el contrario incrementamos el valor de este hiperparámetro, el gradiente aumentará y de esta manera, es posible que la función de pérdida aumente de una manera considerable. Para demostrar este comportamiento, se ha aumentado el valor de la tasa de aprendizaje a 1E-3 para ver como se comporta el optimizador si aumentamos este hiperparámetro (tabla 10).

Hiperparámetros	
Nº de epoch	20
Tasa de aprendizaje	1E-03
Tamaño de grupo	30

Tabla 10-Hiperparámetros del entrenamiento nº4

Los resultados del entrenamiento 4 han sido los mostrados en la tabla 11:

Resultados	Precision	Recall	F1-score
ConMascarilla	0.82	0.73	0.77
Sin mascarilla	0.79	0.85	0.82
Accuracy			0.79

Tabla 11-Resultados del entrenamiento nº4

Tal y como se ha adelantado, el rendimiento de este modelo ha disminuido de forma notable debido al aumento del valor de la tasa de aprendizaje. En la figura 27 se puede observar cómo ha evolucionado la función de validación de la pérdida, en relación a la de entrenamiento. Por lo que se puede deducir que el entrenamiento es malo debido al aumento de la tasa de aprendizaje sobre todo en la parte final del entrenamiento. Generalmente, en los procesos de entrenamiento se utilizan tasas de aprendizaje

elevadas al inicio para reducir el error rápidamente. Posteriormente, esta tasa se reduce para adaptarse al tamaño del error a corregir

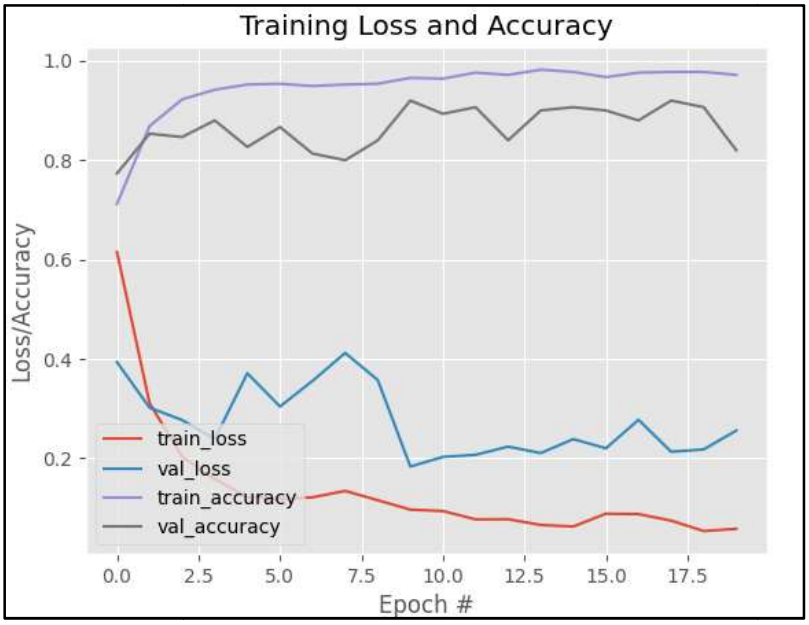

Figura 27-Gráfica del entrenamiento nº4

Elección del modelo de predicción entrenado

Después de la realización de los entrenamientos y ante los resultados obtenidos, se ha optado por la elección del modelo de predicción correspondiente al **entrenamiento número 3**, el cual se ha llamado **Modelo_mascarilla.model**

5.4 Detección de cara y mascarilla

Una vez que se cuenta con el modelo de predicción definitivo, se implementa en la raspberryPi para obtener en tiempo en real la información sobre las mascarillas, a partir de la cual se aplica el protocolo de actuación que se expone en el punto siguiente del presente proyecto.

El enlace del que se ha obtenido parte del código de predicción es el mismo que se ha indicado en el apartado anterior.

El diagrama de flujo del proceso es el reflejado en la figura 28.

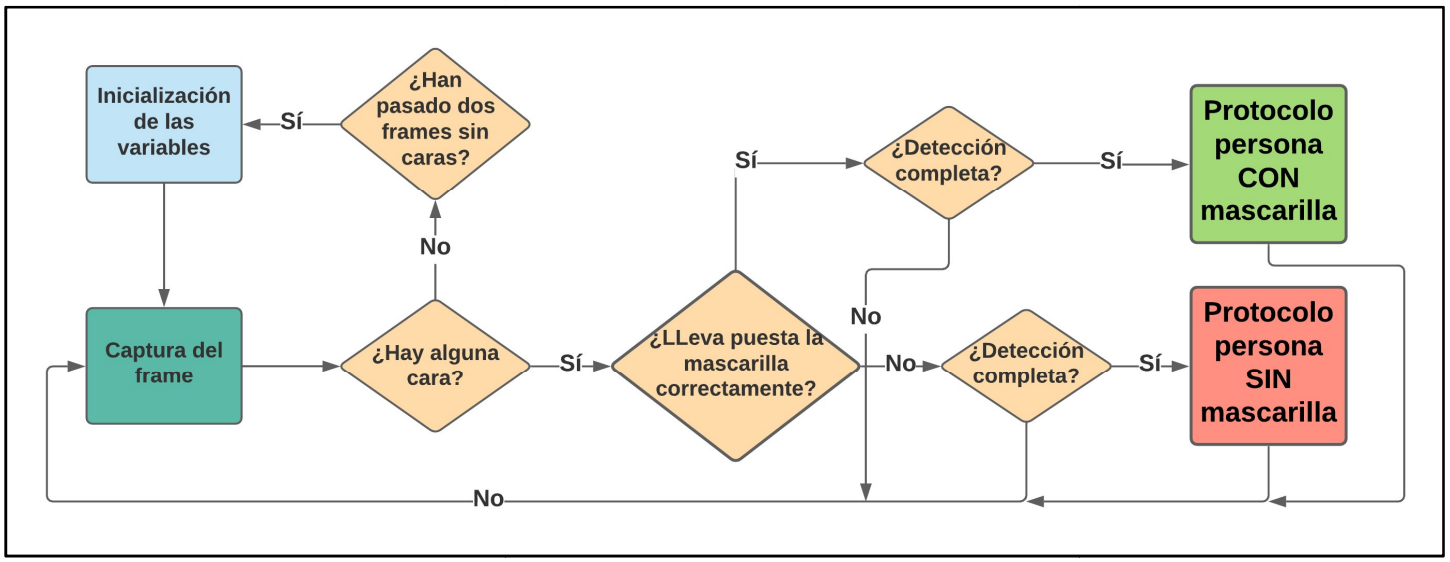

Figura 28-Diagrama de flujo del programa principal

La aplicación se ha diseñado para que en tiempo real se pueda detectar la presencia de mascarillas. En primer lugar, a parte de la inicialización de las variables, se ha iniciado la **retransmisión en tiempo real**. Se ha diseñado para que haya dos opciones de captura de imagen, desde la cámara de la Raspberry (prototipo final) o desde una cámara conectada a uno de los puertos USB de la placa. Así, se puede cambiar el tipo de cámara según las necesidades de cada usuario. Además, se le añade un tiempo de retraso para que el sensor de la cámara pueda activarse (figura 29).

```
# vs = VideoStream(src=0).start()           #Utilizacion WEBCAM
vs = VideoStream(usePiCamera=True).start()  #Utilizacion PiCAM
time.sleep(5.0)
```

Figura 29-Inicio de la retrasmisión en tiempo real

El siguiente paso es capturar el frame, y a partir de esta imagen realizar un preprocesamiento para poder analizar las características de la misma. Primero se guarda el frame con la función **read()** y luego se le cambia el tamaño con **resize(),** propia de la librería imutils.

Una vez capturado este frame, antes de pasarlo por la red neuronal que se ha entrenado, se detectan las caras que hay en el mismo. Para la detección de las caras, se ha implementado un modelo ya entrenado el cual se encuentra en el siguiente enlace:

https://github.com/gopinath-balu/computer_vision/tree/master/CAFFE_DNN

Para poder utilizar este modelo hay que carga la estructura de la red neuronal y sus pesos, además el modelo formado por esta red neuronal. Para poder leer la red neuronal de aprendizaje profundo se ha utilizado el comando **cv2.dnn.readNet()** (figura 30).

```
# Se carga el modelo de detecction de caras
faceNet = cv2.dnn.readNet("deploy.prototxt", "Modelo_deteccion_caras.caffemodel")
```

Figura 30-Modelo detección de caras

Para la detección de las caras se pasa la imagen (frame) a través de la red neuronal (**faceNet.setInput()**) y se obtiene la detección obtenida aplicando ese modelo además de la confianza asociada a esta detección. Se ha establecido una confianza mínima, la cual depende de la situación del detector, en este caso en 0.4, evitando así las detecciones débiles. Una vez se ha detectado la cara, se genera el cuadro que la delimita y se asegura de que no esté fuera de la imagen. Se extraen la ROI (región de interés) de la cara que se corresponde con ese cuadrado delimitador y se le cambia el tamaño para que se corresponda con el esperado a la entrada de la red neuronal del modelo de detección de mascarillas (224,224). Después se pasa la imagen a una matriz (**img_to_array()**), se adapta la imagen al formato que requiere el modelo de mascarillas (**preprocess_input()**) y se guardan en dos listas diferentes, las coordenadas de donde se sitúa la cara (‘locs’) en la imagen original y todo lo que hay en el cuadro que define la cara (‘faces’).

Por otro lado, en relación a la detección de la mascarilla, para cargar el modelo entrenado se ha utilizado **load_model().** Así que, después de la detección de la cara, se realiza una predicción con ese modelo (**predict()**) de si lleva mascarilla o no. Los resultados de la predicción se guardan en una lista, ‘preds’.

Con las predicciones del modelo de detección de mascarillas, se determina si la persona lleva puesta la mascarilla de una manera correcta. Y según esta información se activa el protocolo de actuación que se va explicar en el siguiente apartado. Además para poder visualizar dónde se sitúa la cara del usuario se obtienen de la lista las coordenadas dónde se encuentra la cara para poder así dibujar un cuadro que delimita la posición de la misma.

5.5 Protocolo de actuación

Tal y como se ha comentado en puntos anteriores, uno de los objetivos ha sido crear un protocolo de actuación (figura 31) según la información recibida del detector de mascarillas. Este protocolo se ha diseñado dentro del código del apartado anterior.

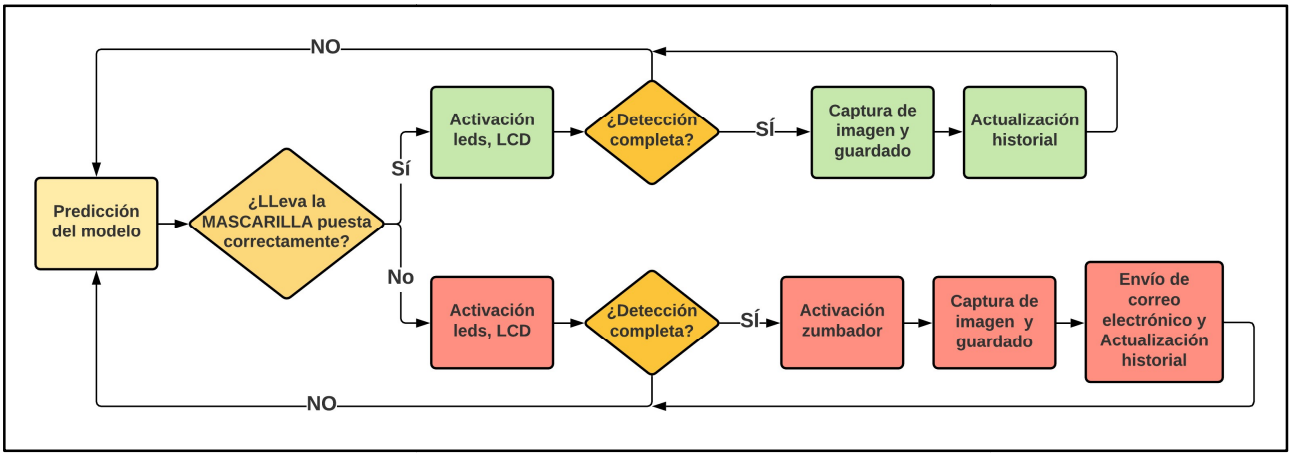

Figura 31-Resumen protocolo activación

Esencialmente, el protocolo de activación comunica con el exterior si el usuario porta o no la mascarilla correctamente, ya sea comunicación hacia el propio usuario, hacia una segunda persona que vigile el control mediante el envío de correo electrónico, activación de unos leds (rojo y verde), muestra por pantalla (LCD) del resultado y el sonido de un zumbador.

5.5.1 Detección completa

Por otro lado, se ha tenido en cuenta que cuando el control detecta una cara, realiza una predicción del resultado. Este se genera uno por cada 'frame', es decir uno por cada vez que se recorre todo el loop. Cada frame tiene una duración de 0.5 segundos debido a todas las funciones que se realizan, sin embargo cuando el detector detecte una cara no siempre va a activar el protocolo de activación ya que sería incompatible que en todos los frames que detecte una cara esté enviando un correo electrónico.Porque por ejemplo puede ser que en un control pueda detectar un negativo durante 6 segundos consecutivos y se enviarían 12 correos electrónicos. Para evitar esta situación, se ha introducido el término de **detección completa**, la cual activa el protocolo de activación.

Esta detección completa se produce si se detecta una persona con o sin mascarilla durante unos frames consecutivos, el número de los mismos depende de la situación de cada control (distancia al usuario y tiempo en el que la cámara capta la presencia del usuario). Con la implementación de esta seguridad se ha conseguido reducir de forma notoria el número de falsos negativos y positivos, ya que, en muchas ocasiones, puede ser que cuando la persona con mascarilla pasa por el control, en principio el detector comunica que es un negativo y al siguiente frame, detecta que la persona lleva la mascarilla correctamente. Esta situación, además de la comentada en el párrafo anterior, han justificado la puesta en marcha de esta seguridad.

En el caso general, se ha establecido que para que se produzca una detección completa hay que detectar a un positivo o a un negativo durante 4 frames consecutivos, dicho lo cual**, el protocolo se activa a los 2 segundos de la llegada del usuario al control**. Pero como ya se ha remarcado anteriormente, esta cantidad de tiempo dependerá de la situación del control en el que se haya implementado este proyecto.

Esta cantidad de frames son contados utilizando un variable (vector) 'Fases' **Fases[0,0,0]**. Estas variables determinan cuando se produce la detección completa, además de comunicar que la persona ya ha pasado por el control.

Fases[0] = variable que cuenta los frames en los que se detecta una persona con la mascarilla puesta correctamente. En el momento en el que pasa el usuario por el detector se inicializa la variable.

Fases[1] = variable que cuenta los frames en los que se detecta una persona sin mascarilla o la lleva de manera incorrecta. En el momento en el que pasa el usuario por el detector se inicializa la variable.

Fases[2] = variable que cuenta los frames cuando no se detecta ninguna cara, se utiliza para inicializar las dos variables anteriores. En el momento en el que hay dos frames consecutivos en el que no se detecta una cara se inicializa esta variable.

Ante la problemática de que dos usuarios pasen por el control al mismo tiempo y se produzcan predicciones distintas para cada uno de ellos, siempre prevalece el protocolo de activación de un negativo, es decir, se enciende el zumbador, envío de correo electrónico…

5.5.2 Activación leds, zumbador y LCD

- Leds y zumbador

Los leds y el zumbador se activan mediante el control de salidas digitales (**12,13 y 19 respectivamente**). Para la activación y desactivación de las mismas, se ha utilizado la biblioteca GPIO (figura 32) de python, tal y como se anota en el Anexo I.

```
#Se evita que al reiniciar el programa, salten
# advertencias informando de que los pines estan en uso
GPIO.setwarnings(False)

#Número de salidas digitales correspondientes al GPIO
Led_positivo=12
Led_negativo=13
Buzzer=19

#Configuración de los pines como salidas digitales
GPIO.setup(Led_positivo,GPIO.OUT)
GPIO.setup(Led_negativo,GPIO.OUT)
GPIO.setup(Buzzer,GPIO.OUT)
```

Figura 32-Configuración salidas digitales

Que como ya se ha comentado anteriormente se activan una vez se produzca la detección completa y se apagan (figura 33) en el momento termine todo el protocolo de activación.

```
GPIO.output(Buzzer,True)
GPIO.output(Led_positivo,True)
GPIO.output(Led_negativo,False)
```

Figura 33-Control leds y zumbador

Se ha tenido en cuenta que los leds no tienen casi resistencia por eso se ha colocado una resistencia de 220Ω en el ánodo de los leds para reducir el voltaje de alimentación.

- LCD

En relación a la pantalla LCD, esta se ha utilizado para comunicar en todo momento el progreso del proceso de predicción, además de su resultado (si se ha detectado la mascarilla o no) y la hora actual. Asimismo, se aporta información al inicio de la ejecución del programa con respecto a la carga de cada uno de los modelos (mascarilla y cara).

Como se apunta en el ANEXO I, se ha utilizado la librería RPLCD para el control de la pantalla.

5.5.3 Captura de imagen

En el momento en que se produce una detección completa de alguno de los usuarios de forma negativa, es decir, que la persona que pasa por el control no lleva la mascarilla de la manera correcta, se realiza una captura de lo que la cámara está captando en ese mismo instante y se guarda en la carpeta de 'Imágenes Resultados'. Este guardado se efectúa para, posteriormente, enviar la imagen del negativo mediante un correo electrónico (figura 34).

Una vez se ha enviado este correo, se procede a la eliminación de la imagen para así mantener la privacidad del usuario (**remove()**).

```
# Directorio en el  que guardamos la captura
img_name_neg="Imagenes_Resultados/image_0.jpg"
cv2.imwrite(img_name_neg,frame )
```

Figura 34-Realización de la captura de pantalla y guardado

5.5.4 Envío de correo electrónico

El envío de correo electrónico se ha utilizado con el fin de que el dispositivo se comunique de manera remota, ya sea al responsable del control o a otra persona, en el momento en el que llegue un usuario con la mascarilla facial puesta de manera incorrecta o directamente no la lleve. De esta manera, el uso de este tipo de comunicación sirve como alarma ante la detección de un negativo que pasa por el control. Además, en el e-mail se ha adjuntado la captura de imagen de la persona que no cumple esta norma y la hora en la que se ha producido esta detección.

Para poder enviar el correo electrónico hay que iniciar sesión en un correo electrónico y para ello hay que mostrar por pantalla el usuario del correo y la contraseña (en el programa se han guardado como 'e-mail' y 'password'). Para evitar la filtración de estos datos, se han guardado en otro de los archivos de este trabajo, 'Datos.pyc'. Se ha generado el archivo ejecutable (.pyc) de Datos.py y se ha eliminado este, con el objetivo de que el usuario no pueda acceder a estos datos. En el programa principal se ha importado este archivo, de manera que así queden ocultos para el usuario y puedan ser utilizados para el envío del correo.

Por otro lado, al inicio del programa se inicializan la información necesaria para el envío de cualquier mail (figura 35): correo electrónico emisor (y su contraseña) y

receptor. Es necesario también la biblioteca referida al envío de mensajes vía correo electrónico, email.message, más concretamente la función EmailMessage.

```
#Inicializacion constantes envio de correo electronico
email_adress= Email                    #Correo emisor
email_password= password               #Contraseña del correo emisor

#Definición del mensaje
msg= EmailMessage()
msg['Subject']='Imagen'
msg['From']=email_adress
msg['To']='imagenes.mascarilla@gmail.com'
```

Figura 35-Inicialización contantes para el envío de email

Además se ha creado una función llamada Send_Email en la que se define la imagen que se adjunta en el correo, además de adjuntarla, la hora en la que se ha producido el negativo y la conexión al servidor SMTP de gmail.

- Adjuntar imagen al correo electrónico (figura 36)

Para adjuntar la imagen al correo, se ha tenido que leer byte a byte la imagen, se ha obtenido el tipo de imagen (que es, además del nombre de la misma). Por último se ha adjuntado al mensaje con la función '**add_attachment'**.

```
#Envío de imagen
with open("Images_Negatives/image_{}.jpg".format(Numero),'rb') as f:
    file_data=f.read()
    file_type=imghdr.what(f.name)
    file_name=f.name

#Adjunto de imagen al mensaje
msg.add_attachment(file_data,maintype='image',subtype=file_type, filename=file_name)
```

Figura 36-Adjunto de imagen al e-mail

- Conexión al servidor SMTP(protocolo de transferencia de correo) de gmail

En referencia al inicio de sesión en un gmail, ha sido necesaria configurar la conexión del servidor SMTP (figura 37). En primer lugar, se ha tenido que activar el acceso a la cuenta de correo que emite el mensaje desde aplicaciones poco seguras, ya que por defecto este apartado está desactivado.

Con respecto al código principal (Detector_mascarillas.py) se ha importado la biblioteca de python 'smptlib' y así configurar el inicio de sesión y el envío del mensaje el cual ya está definido. El nombre del servidor SMTP de gmail es smtp@gmail.com y el puerto usado es el 465, de esta manera el cifrado es el SSL que es más seguro que el TSL. Una vez definido el nombre del servidor, se inicia sesión (**login()**)y se envía el correo(**send_message()**).

```
#Puerto estandar  465 con cifrado ssl
with smtplib.SMTP_SSL('smtp.gmail.com', 465) as smtp:
    smtp.login(email_adress,email_password)
    smtp.send_message(msg)
    smtp.quit()
```

Figura 37-Configuración SMTP

5.5.5 Actualización del historial

En último lugar, se ha establecido un historial en el que se guardan las detecciones completas de cada uno de los usuarios, además de si son positivos o negativos y la fecha en el que se producen. Este historial se genera en archivo de texto (.txt) llamado 'Historial.txt'.

Para poder añadir la información al archivo (figura 38) se abre con el comando **open()** y el 'append'. Después se escribe que la persona lleva o no lleva mascarilla, la fecha y hora con el comando **write()** y, posteriormente se cierra el archivo (**close()**).

```
#Generar archivo txt, Historial Positivos y negativos mas hora de la incidencia
Historial=open("/home/pi/face_mask_detection/Archivos_txt/Historial.txt","a")
Historial.write(now.strftime("Persona SIN mascarilla %d/%m/%Y, a las %H:%M:%S \n"))
Historial.close()
```

Figura 38-Generación del historial

En la figura 39 se observa el diseño del historial.

Historial.txt: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
*****Historial*****
Persona CON mascarilla 10/04/2021, a las 19:41:36
Persona CON mascarilla 10/04/2021, a las 19:41:42
Persona CON mascarilla 10/04/2021, a las 19:41:49
Persona SIN mascarilla 10/04/2021, a las 19:41:59
Persona SIN mascarilla 10/04/2021, a las 19:42:07
Persona CON mascarilla 10/04/2021, a las 19:44:17
Persona CON mascarilla 10/04/2021, a las 19:44:23
Persona SIN mascarilla 11/04/2021, a las 10:25:35
Persona SIN mascarilla 11/04/2021, a las 10:25:43
```

Figura 39-Historial de detecciones

6 Análisis de resultados

En relación a la **detección de mascarillas**, los resultados son idóneos para esta aplicación, ya que se ha mejorado la robustez del modelo ante la problemática de que una persona lleve la mascarilla facial de manera incorrecta, tal y como se puede apreciar en la figura 40. Para comprobar la efectividad del modelo de detección seleccionado, se han usado distintas mascarillas (FFP2, higiénicas y de tela) y personas.

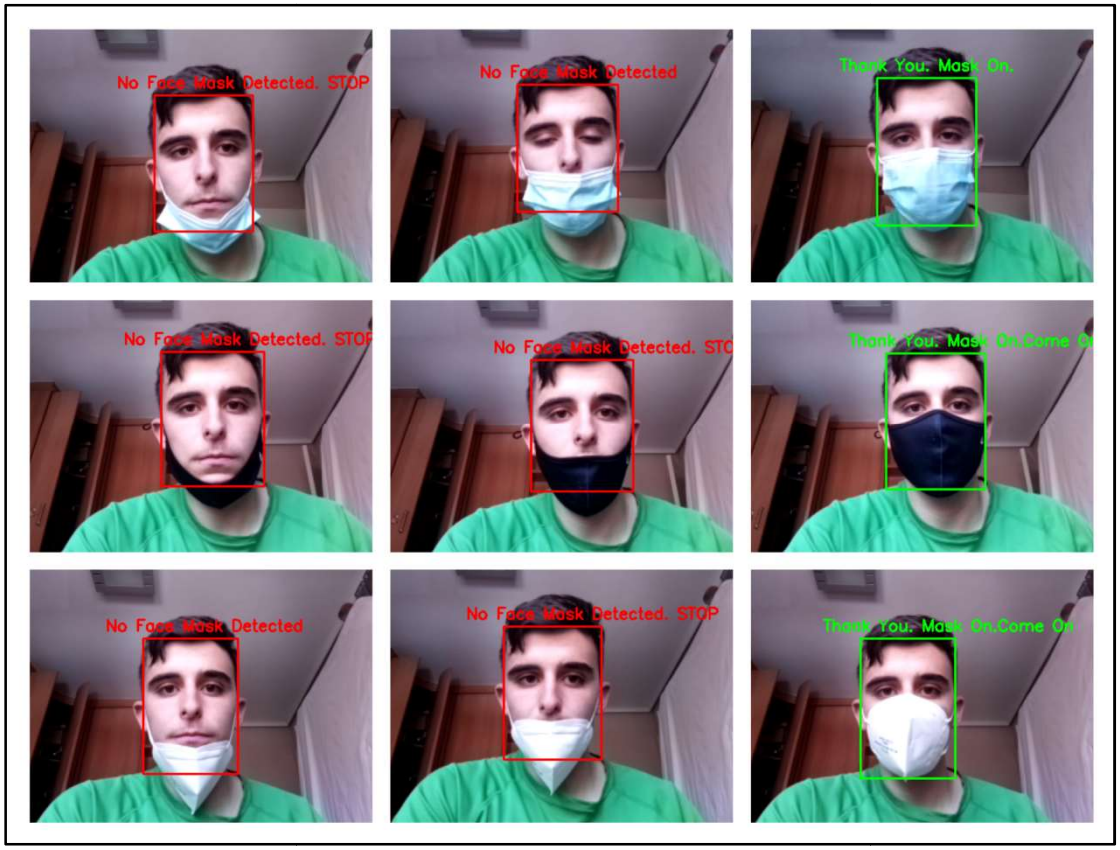

Figura 40-Resultados del modelo 1

Además, también se ha tenido en cuenta si una persona se pone la mano en la boca, ocultando boca y nariz o directamente si no lleva ningún tipo de mascarilla (figura 41).

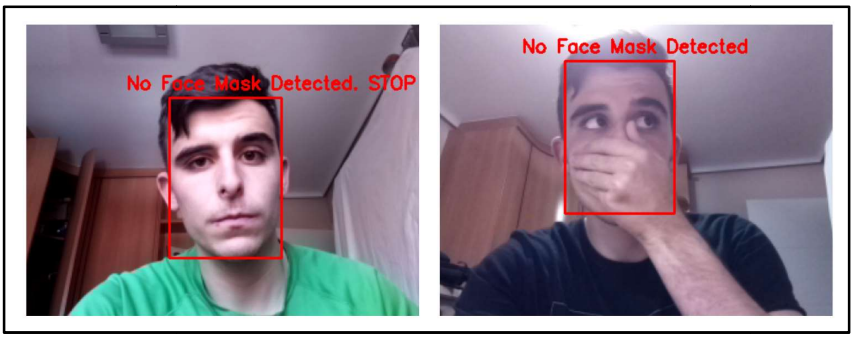

Figura 41-Resultados del modelo 2

Aunque la mayoría de situaciones problemáticas se han solucionado, existen algunas que no han podido llegar a una solución. Como por ejemplo, cuando un usuario utiliza la mascarilla que tiene el dibujo de la boca y nariz o utiliza una transparente (figura 42).

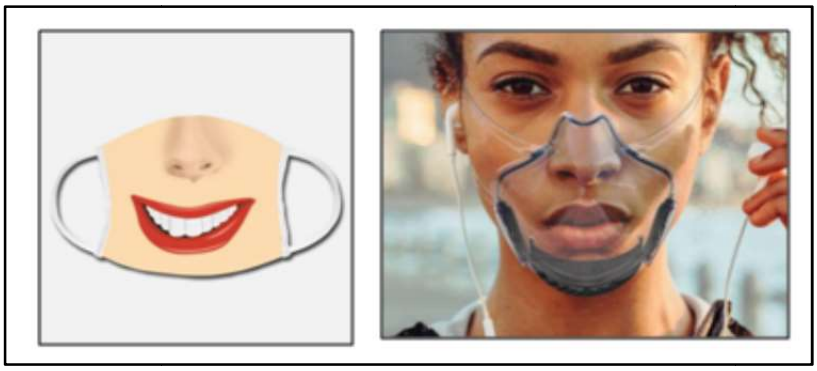

Figura 42- Tipos de mascarillas no detectadas[17][18]

En estos dos casos, el modelo realizaría una predicción incorrecta, ya que estimaría que el usuario que pasa por el control no lleva mascarilla. Para poder mejorar este fallo se tendría que usar una cámara y un procesador mucho más potente, ya que se debería aumentar el conjunto de imágenes de entrenamiento.

Por otro lado, con respecto al protocolo de actuación, no se tarda el mismo tiempo en ejecutar el relacionado con la presencia de positivos y de negativos, ya que este último tiene el envío de correo electrónico y este proceso conlleva un retraso en la ejecución del programa.

Además de los dos protocolos de actuación, el detector de mascarillas funciona correctamente, el modelo realiza un **95% de las predicciones de manera satisfactoria** de un total de 200 pruebas que se han realizado, tan sólo fallando con algún tipo de mascarilla muy específico.

7 Puesta en marcha del dispositivo en un entorno real

Para poder probar los distintos modelos de predicción que se han ido entrenando mientras se recopilaban las imágenes, así como la robustez del sistema, se ha instalado un prototipo del dispositivo en la Piscina Municipal de Teruel (en la entrada a la piscina).

El hardware se ha introducido de manera provisional en una caja de empalme, a la que por dos de los agujeros se han conectado la cámara webcam y la alimentación de la Raspberry.

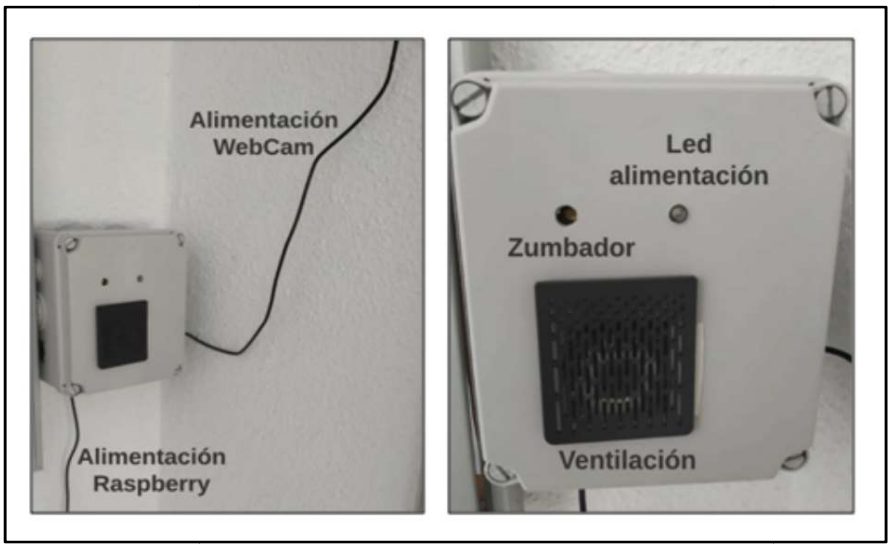

Figura 43- Instalación del dispositivo

Además, también ha sido necesaria la instalación de un zumbador para avisar de los posibles negativos y un led de 'alimentación' el cual informa de si el programa principal de detección está en funcionamiento (figura 43).

En relación a la refrigeración de la placa, se han instalado 3 ventiladores con el objetivo de disminuir la temperatura del procesador. En principio se había colocado sólo uno de ellos, pero debido a que se está trabajando en un entorno con ya de por sí, poca ventilación y además la temperatura es considerablemente elevada, la temperatura de la Raspberry llegó a aumentar hasta los 75ºC a la hora de la puesta en marcha del dispositivo. Sin embargo, cuando se instalaron los dos ventiladores de 4x4 cm se logró que la temperatura sólo llegara hasta los 64,2ºC, lejos de los 80 en los que el procesador empieza a ser más lento, tal y como se ha comentado en el punto 4 (figura 44).

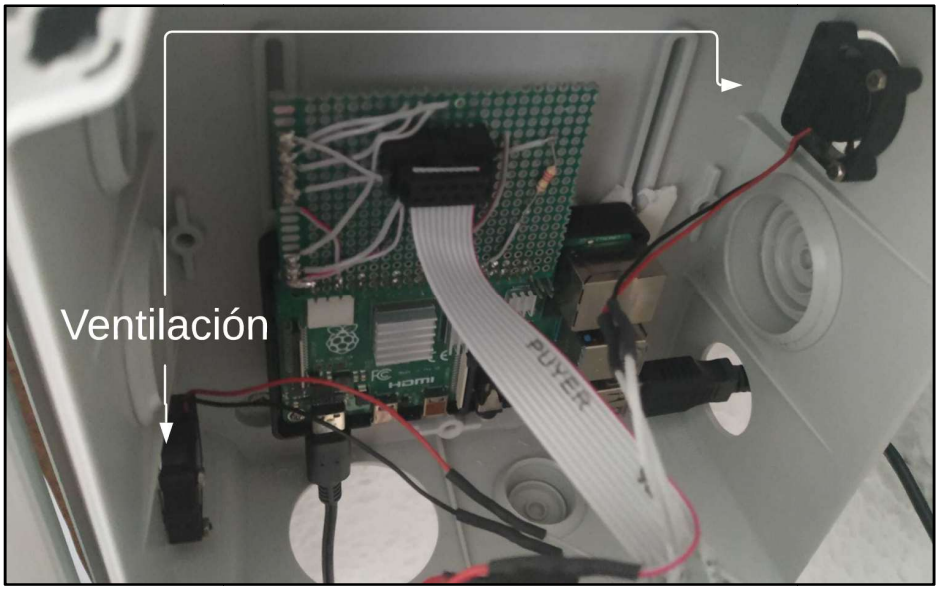

Figura 44-Ventilación interior

Además, se ha diseñado una placa a la que se han soldado todos los elementos que son alimentados por la raspberryPi (Led, zumbador y ventiladores).

Este dispositivo se ha colocado en un lugar en el que la luz del exterior no sea directa, ya que los cambios de luz son muy desfavorables para el rendimiento de la cámara. Una vez han sido solucionadas todas estas problemáticas, la caja de empalmes se ha anclado en la pared y la cámara se ha colocado en frente del torno de entrada, para tener más tiempo a detectar al usuario. La detección se produce más o menos en 1 segundo en este control, por lo que el contador de frames que activa el protocolo de activación será 2 (figura 45).

Figura 45-Colocación del dispositivo

Por último, con respecto a los resultados obtenidos, **aproximadamente el 90% de los usuarios son detectados de manera correcta.** Sin embargo, el 10% restante de las personas no se les ha detectado la cara, es decir, el modelo de predicción de detección de mascarillas no ha realizado ninguna predicción, porque según el modelo de detección de caras, no ha habido ninguna en los frames analizados. Y esta situación se debe a que mucha gente entra mirando al suelo ya que, por ejemplo, tienen que coger la tarjeta de entrada en el bolso. Asimismo, las caras de personas con el pelo largo y suelto son mucho más difíciles de detectar ya que el pelo, en algunas ocasiones, les tapa la cara.

Quitando estos errores residuales, la mayoría de las personas han sido detectadas según mostrando si llevaban o no mascarilla facial.

8 Conclusiones y líneas de trabajo futuras

8.1 Conclusiones

Con este proyecto se ha conseguido diseñar y poner en marcha un dispositivo compacto de bajo coste de detección de mascarillas en tiempo real, el cual proporciona una respuesta inmediata según la información recibida por el modelo de predicción entrenado.

Además, el trabajo que se ha realizado ha permitido ajustar los modelos ya existentes de detección de mascarillas, mejorando de una forma notable la problemática de las personas que llevan la mascarilla mal puesta. En cuanto a esta detección se ha tenido en cuenta que el número de falsos positivos (todos ellos en el momento que la llevan mal puesta) aumenta conforme incrementan estos tres factores:

1. Distancia de la cámara al usuario
2. Diferencia de la altura de la cámara y la altura del usuario
3. Grado de inclinación de la cámara conforme a la cara del usuario

Otro de los objetivos logrados en este proyecto es la implementación de este modelo en un dispositivo compacto como es la Raspberry Pi, asimismo se consigue que sea de bajo coste gracias a los pocos elementos hardware que se necesitan para la implementación del modelo en el ordenador monoplaca.

Asimismo, se ha conseguido diseñar un protocolo de actuación ante la posible detección de negativos o positivos, teniendo la opción de guardar un historial de los usuarios detectados en el control, del envío de un correo electrónico con imagen adjunta del negativo y la implementación de avisos tanto lumínicos como sonoros. Además se ha conseguido empaquetar el dispositivo en una caja diseñada en 3D, haciendo que sólo sea visible a la persona del usuario el cable de alimentación de la Raspberry pi, la caja y la cámara.

Este dispositivo se ha probado en un entorno real, obteniendo unos resultados excelentes en relación a la detección de mascarillas y a la ejecución del protocolo de actuación. Con lo que se podría implementar en otras situaciones, como la entrada de una escuela o de una piscina ya que al ser tan compacto es de fácil instalación en cualquier entorno.

Por último, cabe decir que durante la realización de este trabajo he adquirido una serie de conocimientos y destrezas. En primer lugar, el manejo de la raspberryPi y algunas de las miles de posibilidades que tiene este ordenador monoplaca. He aprendido a usar el aprendizaje automático para aplicaciones de reconocimiento facial, además del uso y entrenamiento de las redes neuronales convolucionales, y el posterior análisis de los resultados del modelo de predicción generado. Por otro lado también se ha aprendido a enviar de correos electrónicos a través de python, a realizar una retrasmisión en tiempo real, a capturar y manipular una imagen.

8.2 Líneas de trabajo futuras

De las conclusiones expuestas se han desprendido dos importantes líneas de trabajo futuras.

En relación al modelo se puede cambiar el diseño con el objetivo ajustar todavía más los resultados obtenidos, diferenciando entre persona sin mascarilla, con mascarilla y con mascarilla mal puesta, añadiendo una tercera categoría a las posibles salidas del modelo. Para ello se tienen que realizar los siguientes pasos:

- Cambiar la salida del clasificador, sustituyendo la última capa por otra que tenga 3 neuronas.
- Para etiquetar los datos guardados en cada directorio, no se podrá realizar binarizándo las etiquetas como en este trabajo, ya que serían 3 tipos de datos así que habrá que utilizar el **LabelEncoder()** en vez del LabelBinarizer()
- Crear la carpeta dentro del directorio dataset dónde se guardarán las imágenes de personas con mascarillas mal puestas
- Aumentar el número actual de imágenes tomadas con la mascarilla mal puesta, con y sin ella, ya que al ser un modelo más preciso se necesitará una mayor cantidad de imágenes
- Rediseñar el protocolo de actuación (añadir a las predicciones del modelo la posibilidad de que la mascarilla esté mal puesta)

Y la segunda y última vía de trabajo es el diseño de una placa de circuito impreso que lleve la Raspberry incrustada además de todos los circuitos existentes, disminuyendo la posibilidad de fallos en el hardware, además de la disminución del tamaño del dispositivo, mejorando así su compactibilidad.

9 Bibliografía

[1] BOE, "Boletón Oficial del Estado," *Boletín Of. del Estado*, pp. 26798–26800, 2021.

[2] Organización Mundial de la Salud, "Recomendaciones sobre el uso de mascarillas en el contexto de la COVID-19," *Organ. Mund. la salud*, pp. 1–5, 2020, [Online]. Available: https://extranet.who.int/iris/restricted/handle/10665/331789.

[3] Publico, "El alcalde de Ordizia contradice a Urkullu y dice que no pueden garantizarse las elecciones del domingo." https://www.publico.es/politica/alcalde-ordizia-contradice-urkullu-dice-no-garantizarse-elecciones-domingo.html (accessed May 22, 2021).

[4] Techlandia, "¿Qué es una tableta digital?" https://techlandia.com/tableta-digital-info_227810/ (accessed May 24, 2021).

[5] Mobbeel, "Historia del Reconocimiento Facial." https://www.mobbeel.com/blog/historia-del-reconocimiento-facial/ (accessed May 24, 2021).

[6] W. Zhou, Y. Chen, and S. Liang, "Sparse Haar-like feature and image similarity-based detection algorithm for circular hole of engine cylinder head," *Appl. Sci.*, vol. 8, no. 10, 2018, doi: 10.3390/app8102006.

[7] S. Gupta, "A short introduction to heavy-ion physics," vol. 14, no. 5, pp. 771–780, 2015, [Online]. Available: http://arxiv.org/abs/1508.01136.

[8] R. Klette, *Concise Computer Vision: Stereo Matching*. 2014.

[9] Q. Fang *et al.*, "Detecting non-hardhat-use by a deep learning method from far-field surveillance videos," *Autom. Constr.*, vol. 85, no. January, pp. 1–9, 2018, doi: 10.1016/j.autcon.2017.09.018.

[10] E. L. DE FALKEN, "Visión artificial con redes convolucionales." https://www.ellaberintodefalken.com/2019/10/vision-artificial-redes-convolucionales-CNN.html (accessed May 30, 2021).

[11] P. COSTA, "Redes neuronales convolucionales explicadas." https://pochocosta.com/podcast/redes-neuronales-convolucionales-explicadas/

(accessed May 30, 2021).

[12] Raspberry, "Camera Module V2." https://www.raspberrypi.org/products/camera-module-v2/ (accessed May 26, 2021).

[13] Y. Zhang, J. Gao, and H. Zhou, "Breeds Classification with Deep Convolutional Neural Network," *ACM Int. Conf. Proceeding Ser.*, pp. 145–151, 2020, doi: 10.1145/3383972.3383975.

[14] "PiCamera." https://picamera.readthedocs.io/en/release-1.10/index.html# (accessed Jun. 01, 2021).

[15] M. S. A. H. M. Z. A. Z. L.-C. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks."

[16] DataSmarts, "¿Qué es un Optimizador y Para Qué Se Usa en Deep Learning?" https://datasmarts.net/es/que-es-un-optimizador-y-para-que-se-usa-en-deep-learning/ (accessed Jun. 15, 2021).

[17] SpreadShirt, "No Title." https://www.spreadshirt.es/shop/accesorios/mascarillas/-+boca+nariz/ (accessed Jun. 15, 2021).

[18] B. INSIDER, "Transparentes, capaces de autodesinfectarse o conectadas a tu móvil: así serán las mascarillas que llevarás en el futuro." https://www.businessinsider.es/seran-mascarillas-llevaras-futuro-690947 (accessed Jun. 15, 2021).