Anexo I. Puesta a punto Raspberry Pi 4 Model B.

En este anexo se van a explicar de manera detallada todos los pasos a seguir para poner en marcha la placa procesadora Raspberry Pi, y así llevar a cabo la realización del trabajo de fin de grado.

Instalación del sistema operativo en la Raspberry Pi

Como se ha comentado antes, se ha escogido la tarjeta Raspberry Pi modelo 4b. Una de las características de esta placa procesadora es que necesita de un sistema operativo para su utilización. Este se instalará en la tarjeta micro SD, que en nuestro caso será de 64 gb, normalmente se utilizan tarjetas de esta capacidad, en el caso de que la Raspberry sea utilizada para utilidades más comunes (ordenador auxiliar, reproductor multimedia…). Existen dos posibilidades para descargarlo e instalarlo: NOOBS o Raspbian. NOOBS (New Out Of Box Software) es un instalador de Raspbian (SO propio de la tarjeta Raspberry Pi, actualmente llamado Raspberry Pi OS) que nació para facilitar la instalación del mismo, pero no será necesaria su utilización ya que esta se ha mejorado, de manera que instalaremos Raspbian directamente.

Una vez realizada la elección del sistema operativo, se escoge un ejecutable en el que tenemos la propia interfaz gráfica del escritorio y, además, una serie de software recomendado por la marca Raspberry (Python 3, Thorton IDE, Scratch…). En el que la versión del SO será Raspbian GNU/Linux 10 (buster). Las ventajas de este SO es que es de software libre y de código abierto, con lo que está en constante evolución, además de que es totalmente gratis se ha descargado en la siguiente dirección:

https://www.raspberrypi.org/software/

Antes de la instalación, habrá que formatear la tarjeta micro SD, en este caso se ha realizado con el programa de balenaEtcher 1.15.116, es un software multiplataforma de código abierto que es capaz de guardar carpetas comprimidas en tarjetas SD o microSD. Una vez realizado el formateo, utilizaremos el mismo programa instalar el SO en la tarjeta SD. Enlace de descarga:

https://www.balena.io/etcher/

Una vez se ha elegido el sistema operativo y la manera de formatear la tarjeta se lleva a cabo su instalación en la misma (figura 46).

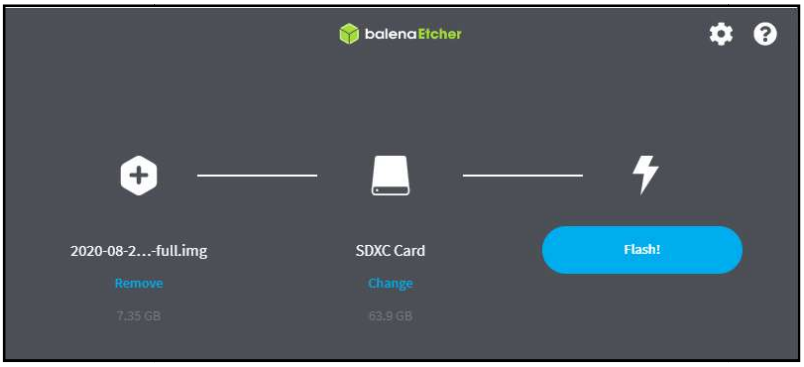

Figura 46-Instalación SO en la tarjeta microSD

En el momento que se formatea y se ejecuta el sistema en la tarjeta, se extrae del dispositivo en el que se ha realizado la instalación, se introduce la tarjeta microSD en la Raspberry y se procede a su configuración.

Inicio de la configuración del sistema operativo y de la Raspberry Pi

Una vez introducida la tarjeta microSD en la entrada adaptada para ello de la Raspberry, habrá que comenzar la configuración de la misma. Pero antes, se conectará un ratón y un teclado a dos de las cuatro entradas que tienen la placa. Además del cable de alimentación tipo C y un cable micro HDMI- HDMI a una de las dos salidas adaptadas para esta finalidad. Con todo esto ya se podrá empezar la configuración.

Iniciada ya la Raspberry PI, la primera pantalla que aparece es la interfaz del escritorio desde donde se empezará la configuración (figura 47).

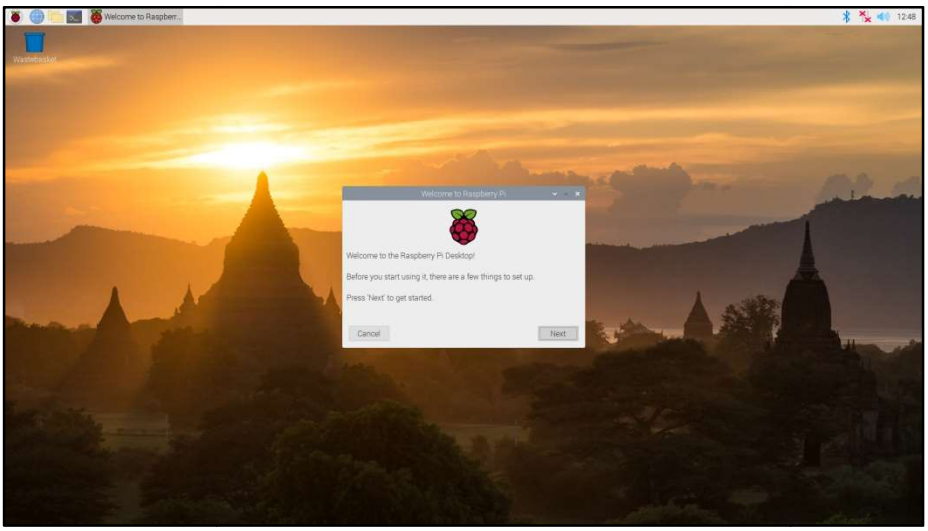

Figura 47-Interfaz gráfica del escritorio

Al haberse escogido la versión del sistema operativo en la que se instala también una serie de software recomendado, se tiene acceso a varios editores de texto (Thonny Python IDE, Scratch, Greenfoot Java IDE…), el LibreOffice característico del sistema operativo Linux, reproductor de video (VLC Media Player)…

Para iniciar la configuración pulsamos 'next' y se establece la zona horaria y el lenguaje del teclado, que en este caso será el inglés (figura 48).

Figura 48-Selección de idioma

El siguiente paso será la definición de la cuenta de pi, que por defecto es 'raspberry', aunque en este caso se cambiará (figura 49).

Figura 49-Definición de la contraseña de la cuenta de Raspberry

Luego habrá que configurar la pantalla para que no presente los bordes de color negro en el monitor, ya que por defecto la Raspberrry tiene una resolución determinada y normalmente no se asemeja a la que se tendrá que utilizar. Por ello, se acepta la opción que comenta que la pantalla presenta bordes negros alrededor del escritorio.

Esta placa procesadora tiene la posibilidad de conectarse a internet mediante una entrada tipo Ethernet, sin embargo, se ha escogido conectarse de manera inalámbrica con Wi-fi (figura 50). Así que en este paso habrá que determinar a cuál de las redes se conectará la Raspberry y posteriormente, su correspondiente contraseña.

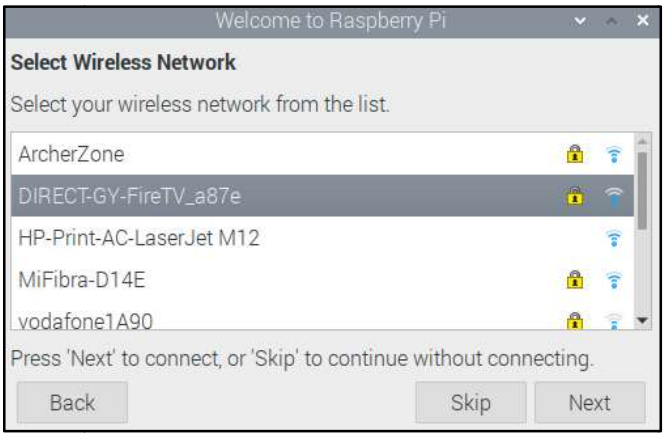

Figura 50-Selección red Wi-fi

Por último se da la posibilidad de actualizar parte del software y aplicaciones si fuera necesario. Pulsando 'next' se activa este proceso, el cual tarda alrededor de unos minutos. Al acabar se reinicia para actualizar todos los cambios que se han establecido en este proceso (figura 51).

Figura 51- Finalización de la configuración inicial

Instalación VNC en Raspberry Pi 4B

Una de las problemáticas que conlleva la utilización de la Raspberry, es que, según para que aplicaciones, se hace complicado tener monitor, teclado y ratón conectados a la placa. Y este es un caso en el que será necesario recurrir a un control remoto desde un ordenador auxiliar, que controle el sistema operativo desde sus propios periféricos. De modo que, sólo será necesario para manejar la Raspberry, un PC auxiliar y el cable de alimentación. Además de la disminución de hardware y todo lo que ello conlleva, al poder acceder por control remoto, se podrá realizar cambios en la configuración, código, aplicaciones de la Raspberry desde cualquier lado, siempre y cuando se tenga acceso a internet. Una vez configurada, funciona sola sin necesidad del ordenador auxiliar.

Esta herramienta es la del servidor VNC (Virtual Network Computing), la cual accede de manera remota a la interfaz gráfica de la placa procesadora. Además cabe la posibilidad de que varios usuarios puedan acceder a la misma Raspberry y, al mismo tiempo se verá el cambio que están realizando ambos usuarios.

El primer paso para configurar el servidor (VNC server) en la Raspberry, es comprobar si la versión instalada en la misma es la más nueva. Normalmente en las versiones actuales ya viene preinstalado por lo que no ha sido necesaria su instalación (figura 52).

```
pi@raspberrypi:~ $ sudo apt-get update
Get:1 http://raspbian.raspberrypi.org/raspbian buster InRelease [15.0 kB]
Get:2 http://archive.raspberrypi.org/debian buster InRelease [32.9 kB]
Get:3 http://archive.raspberrypi.org/debian buster/main armhf Packages [375 kB]
Fetched 423 kB in 1s (323 kB/s)
Reading package lists... Done
pi@raspberrypi:~ $ sudo apt-get install realvnc-vnc-server
Reading package lists... Done
Building dependency tree
Reading state information... Done
realvnc-vnc-server is already the newest version (6.7.2.42622).
The following packages were automatically installed and are no longer required:
  gconf-service gconf2-common libbluetooth3 libexiv2-14 libgconf-2-4
  libgfortran3 libgmime-2.6-0 libncurses5 libssl1.0.2 lxplug-volume uuid-dev
Use 'sudo apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Figura 52-Comandos Instalación VNC Server

Una vez realizada esta comprobación habrá que activar el servidor VNC. Se hará desde la línea de comandos, aunque también se puede realizar desde la interfaz. Para ello habrá que abrir la herramienta de configuración (figura 53):

```
pi@raspberrypi:~ $ sudo raspi-config
```

Figura 53- Comando configuración Raspberry

Ya con el menú abierto se pulsará el botón de 'Interface options' y después se activará el servidor de VNC presionando en el botón que tiene el nombre del servidor. Existe otra opción para activar el protocolo VNC de la Raspberry, accediendo desde el MenuPrincipal>Preferences>Rasberry Pi Configuration, en Interfaces también se puede activar el mismo. En ese momento desde la terminal aparecerá el aviso de que el servidor VNC se ha activado con éxito y además en la barra de herramientas de la parte derecha habrá un logo del propio servidor.

Para conectarse desde dispositivos externos será necesario conocer la dirección IP de la Raspberry, la cual podemos conocer desde la terminal escribiendo el siguiente comando: "hostname –I".

El siguiente paso será instalar VNC-Viewer, en los dispositivos desde los cuales se quiera acceder a la Raspberry PI. Existe la posibilidad de descargar este software desde cualquier tipo de sistema operativo (en este caso Windows, pero con la posibilidad de descargarlo en android, IOS, Linux).

Enlace de descarga:

https://www.realvnc.com/es/connect/download/viewer/

Una vez descargada la aplicación, para conectarse a la Raspberry, habrá que introducir la dirección IP de la misma (sólo en la primera conexión) y el nombre con el que se identificará a esa IP (figura 54).

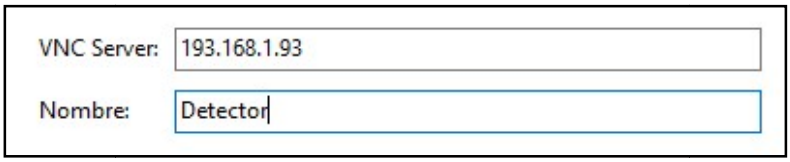

Figura 54-Dirección IP Raspberry Pi

Además, cada vez que se precise el control remoto de la placa, hay que introducir el usuario y la contraseña de la misma.

Instalación de bibliotecas

OpenCV

Es la biblioteca desarrollada por Intel para la realización de visión artificial (OpenCV: Open Computer Vision), necesaria para toda modificación y procesado de imágenes. Está generado sobre el lenguaje Python, es un software totalmente libre, con lo se encuentra en un continuo desarrollo, además de que su uso es muy frecuente en todos los proyectos relacionados con visión artificial.

Hay muchas maneras de instalar esta biblioteca, en este caso se ha realizado de la siguiente manera. Primero se han descargado e instalado una serie de paquetes necesarios para la utilización de esta herramienta.

Después se ha obtenido la versión de OpenCV 4.1.0.25. Aunque hay versiones posteriores de la herramienta, esta es una de las más estables. La instalación de la biblioteca tan sólo dura alrededor de unos 5 minutos, una vez instalada se comprueba que la instalación se ha realizado de manera satisfactoria. Los pasos que se deben seguir para esta instalación quedan determinados en el siguiente enlace:

https://omes-va.com/como-instalar-opencv-en-raspberry-pi/

Tensorflow

Tensorflow es una plataforma de código abierto necesaria para el aprendizaje automático (formación y carga del modelo, adaptación y generación de las capas del mismo y procesamiento de las imágenes adaptándolas a la generación del modelo). Además esta herramienta, es capaz de ejecutar la biblioteca Keras, que está diseñada para la experimentación de redes de aprendizaje profundo de una manera bastante rápida. Así que entre estas dos bibliotecas se podrán formar los modelos deseados (figura 55).

Figura 55-Tensorflow+keras

La versión de tensorflow a instalar va a ser la 2.1.0, como la versión de python es la 3.7.3 no habrá ningún problema con la instalación de tensorflow. Además para la

instalación de esta herramienta se requiere que el administrador de paquetes (pip) tenga una versión mayor que la 19.0, así que como por defecto estaba instalada la versión 18.1, se ha actualizado a la versión más actual 21.1.1.

Una vez cumplidos estos requisitos, se procederá a la instalación de la herramienta según los pasos que determina el siguiente enlace:

https://qengineering.eu/install-tensorflow-2.1.0-on-raspberry-pi-4.html

Imutils

Con el objetivo de realizar una serie de funciones básicas de OpenCV de una manera mucho más sencilla, se descargará la biblioteca imutils. La instalación es extremadamente sencilla, tan solo habrá que ejecutar el siguiente desde la terminal:

pip3 install imutils

Tal y como determina el siguiente enlace:

https://pypi.org/project/imutils/

Entonces, se descargará automáticamente la versión 0.5.4 de esta biblioteca.

RPLCD

En este trabajo se necesita la utilización de una pantalla LCD, con el objetivo de comunicar si son positivos o negativos. Para que en la instalación del hardware sea más sencilla, se ha escogido una pantalla con el expansor Entrada/Salida I2C PCF8574. De esta manera, se podrá conectar al bus I2C de la raspberry pi.

Para controlar esta pantalla se ha descargado la biblioteca RPLCD en su versión 1.3.0 (compatible con el módulo expansor), que es una de las más intuitivas de todas las que existen. Para descargarla simplemente habrá que ejecutar el siguiente comando desde la terminal:

pip3 install RPLCD

Tal y como determina el siguiente enlace:

https://pypi.org/project/RPLCD/

Además habrá que instalar la biblioteca python-smbus para poder así controlar la pantalla desde el bus I2C de la Raspberry.

sudo apt-get install python-smbus

Y por último, habrá que habilitar el puerto I2C, de esta manera ya se podrá escribir en la LCD. Para ello, desde la interfaz del escritorio se accede a Menu>Preferences>Raspberry Pi Configuration y en interfaces dónde anteriormente se ha activado pi Camera, se activa el puerto I2C (figura 56).

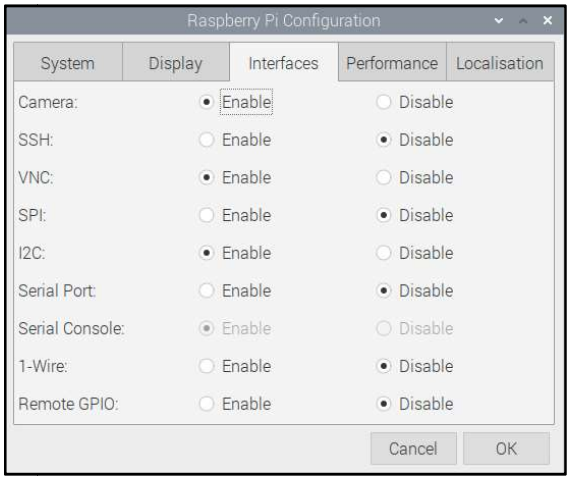

Figura 56-Activación bus I2C

Scikit-learn

La biblioteca Scikit-learn se utiliza frecuentemente en el ámbito del aprendizaje automático (machine learning). Es de una biblioteca de open-source para el lenguaje de progamación de Python y en este caso, se utilizará para generar el informe de clasificación del entrenamiento, dividir el dataset en dos bloques y binarizar las etiquetas de cada imagen.

Se descargará la versión 0.24.1, y aunque en este apartado se refiera a esta biblioteca como Scikit-learn, en el entorno de programación de Python se refiere a ella como Sklearn.

Todas las bibliotecas que se describen en este anexo, se instalan con el administrador de paquetes pip3, ya que está asociado a python3 que es el que se utiliza en este trabajo, más concretamente la versión 3.7.3. Las demás bibliotecas y funciones que se importan en los distintos programas no necesitan instalación ya que vienen por

defecto, solo hará falta importar las funciones y/o bibliotecas que dispongamos desde el propio código.

Anexo II. Códigos de recopilación de imágenes.

FotosConMascarilla.py

```
import cv2
import glob
from picamera import PiCamera
from picamera.array import PiRGBArray

#Inicialización de la PiCamera
Camara= PiCamera()
Camara.resolution = (512, 304)
Captura_Camara= PiRGBArray(Camara, size=Camara.resolution)

#Numero de archivos en el directorio donde se guardan las imágenes
Numero=len(glob.glob("/home/pi/Entrega_TFG/dataset_aux/FotosConMascarilla/*jpg"))


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)

        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/FotosConMascarilla/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 57-Algoritmo recopilación imágenes 1

FotosSinMascarilla.py

```python
import cv2
import glob
from picamera import PiCamera
from picamera.array import PiRGBArray


#Inicialización  de la PiCamera
Camara = PiCamera()
Camara.resolution = (512, 304)
Captura_Camara = PiRGBArray(Camara, size=Camara.resolution)

#Numero de archivos en el directorio donde se guardan las imágenes
Numero=len(glob.glob("/home/pi/Entrega_TFG/dataset_aux/FotosSinMascarilla/*jpg"))


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)

        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 58-Algoritmo recopilación imágenes 2

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

```
# Funcion para generar lotes de datos de imágenes de tensores con aumento de datos en tiempo real
from tensorflow.keras.preprocessing.image import ImageDataGenerator
# Funcion para utilizar la  arquitectura de red neuronal preentrenada
from tensorflow.keras.applications import MobileNetV2
# Funcion necesaria para reducir el tamaño de la entrada vertical y horizontalmente
from tensorflow.keras.layers import AveragePooling2D
# Funcion necesaria para evitar el sobreajuste de la red neuronal
from tensorflow.keras.layers import Dropout
# Funcion necesaria para aplanar la matriz de la imagen (vector)
from tensorflow.keras.layers import Flatten
# Funcion necesaria para incluir capas ocultas, de entrada y de salida a la red neuronal
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
# Algoritmo de optimizacion
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
# Funcion para convertir una imagen PIL en una matriz
from tensorflow.keras.preprocessing.image import img_to_array
# Funcion para cargar una imagen de un directorio
from tensorflow.keras.preprocessing.image import load_img
# Funcion para conseguir matrices binarias
from tensorflow.keras.utils import to_categorical
 # Funcion para binarizar las etiquetas (asignar 1 o 0 segun etiqueta)
 from sklearn.preprocessing import LabelBinarizer
 # Funcion para dividir un dataset en dos bloques
 from sklearn.model_selection import train_test_split
 # Funcion para generar el informe de clasificacion
 from sklearn.metrics import classification_report
 from imutils import paths
 # Biblioteca necesaria para el dibujo de graficas
 import matplotlib.pyplot as plt
 # Biblioteca necesaria para el manejo de matrices
 import numpy as np
 # Biblioteca necesaria para definir el analizador de argumentos
 import argparse
 import os

 # Se contruye el analizador de argumentos
 ap = argparse.ArgumentParser()
 ap.add_argument("-d", "--dataset", required=True,
     help="path to input dataset")
 ap.add_argument("-p", "--plot", type=str, default="plot.png",
     help="path to output loss/accuracy plot")
 ap.add_argument("-m", "--model", type=str,
     default="mask_detector.model",
     help="path to output face mask detector model")
 args = vars(ap.parse_args())

 # Se establecen los hiperparámetros
 INIT_LR = 1e-4
 EPOCHS = 20
 BS = 30

print("[INFO] loading images...")
# Se escoge la lista de imagenes del directorio seleccionado
imagePaths = list(paths.list_images(args["dataset"]))
# Se inicializa la lista de datos y etiquetas
data = []
labels = []

#loop sobre las imagenes
for imagePath in imagePaths:

    # Se extrae la etiqueta del nombre del archivo, si le hemos dicho que lleva o no lleva mascarilla
    label = imagePath.split(os.path.sep)[-2]

    # Se carga la imagen de entrada y se establece el tamaño
    image = load_img(imagePath, target_size=(224, 224))
    # Se convierte una instancia de imagen PIL(Python Image Library) en una matriz
    image = img_to_array(image)
    # Se preprocesa la matriz que codifica a nuestra imagen
    image = preprocess_input(image)
```

Figura 59-Algoritmo del entrenamiento del modelo 1

```
    # Se actualiza la lista de datos, añadiendo a la matriz que codifica a nuestra imagen
    data.append(image)
    # Se actualiza la lista de etiquetas
    labels.append(label)

# Se convierten los datos y etiqutas en matrices, y se sobreescribe
data = np.array(data, dtype="float32")
labels = np.array(labels)

# Se codifica unidireccionalmente en las etiquetas, 1 positivo 0 negativo
# fit_transform= Ajuste binarizador de etiquetas y transformador de etiquetas tipo 'str' en etiquetas binaria
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
# Se convierte el vector en una matriz binaria
labels = to_categorical(labels)

# Proporcion de datos de 75% de datos para el entrenamiento y el restante para pruebas
(trainX, testX, trainY, testY) = train_test_split(data, labels,
    test_size=0.20, stratify=labels, random_state=42)
    # data = imagenes a dividir
    # labels = con mascarilla o sin mascarilla (1 o 0)
    # test_size= define el tamano del conjunto de prueba
    # stratify= se utiliza para mantener la proporcion de datos de cada clase
    # random_state= fija una semilla para la generacion de numeros aleatorios


# Se genera lotes de datos de imágenes de tensores con aumento de datos en tiempo real
aug = ImageDataGenerator(
    rotation_range=20,     # Rango de grados para rotaciones aleatorias
    zoom_range=0.15,       # Rango para zoom aleatorio
    width_shift_range=0.2, # Fraccion del total del ancho
    height_shift_range=0.2,# Fraccion del total del alto
    shear_range=0.15,      # Intensidad de corte
    horizontal_flip=True,  # Se voltea las entradas horizontalmente de manera aleatoria
    fill_mode="nearest")


# Se carga la arquitectura de la red neuronal MobileNetV2
baseModel = MobileNetV2(weights="imagenet", include_top=False,
    input_tensor=Input(shape=(224, 224, 3)))
    # weights= utilizamos los pesos que se entrenaron previamente en el conjunto de datos de Imagenet
    # include top= no utilizamos la capa superior de la red neuronal
    # input_tensor = necesario para utilizar una imagen como entrada del modelo


# Se contruye el clasificador que será puesto al principio del mismo
headModel = baseModel.output
headModel = AveragePooling2D(pool_size=(7, 7))(headModel) # Se reduce 7 veces la entrada vertical y horizontal
headModel = Flatten(name="flatten")(headModel)            # Se convierte los elementos de una matriz de imagen
headModel = Dense(128, activation="relu")(headModel)      # Se añade una capa oculta a la red neuronal con 128
headModel = Dropout(0.5)(headModel)                       # Se que se produzca un sobreajuste de la red neuron
headModel = Dense(2, activation="softmax")(headModel)     # Se establece la capa de salida con dos nodos

# Se define la entrada y salida del modelo
model = Model(inputs=baseModel.input, outputs=headModel)

# loop sobre todas las capas del modelo y las congelamos,
# asi no seran actualizadas durante el primer proceso de entrenamiento

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

print("[INFO] compiling model...")

# Se define el algoritmo de optimizacion
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# Se compila el modelo
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])
    #loss=función de perdida ( modelo de clasificacion binaria)
    #optimizer= algoritmo de optimizacion escogido
    #metrics = guardamos los valores de precision en accuracy
```

Figura 60-Algoritmo del entrenamiento del modelo 2

```
# Se entrena el modelo nuestro modelo ejectuando la funcion fit
print("[INFO] training head...")
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),   # Definición del lote y  de los dos grupos de datos
    steps_per_epoch=len(trainX) // BS,         # Se definen los pasos  por epoca
    validation_data=(testX, testY),            # Se definen los datos sobre los cuales evaluar la perdida
    validation_steps=len(testX) // BS,         # Se definen los pasos antes de cada epoca
    epochs=EPOCHS)                             # Se definen  el numero de epocas ( pasada a traves de
                                               # las filas del conjunto de datos)

# Se realizan las predicciones del modelo
print("[INFO] evaluating network...")
#Genera predicciones de salida para muestras de entrada
predIdxs = model.predict(testX, batch_size=BS)

# Para cada imagen testeada se necesita encontrar el inidice de la etiquta
#con su correspondiente maxima probabilidad
# argmax devuelve el argumento maximo en el eje 1
predIdxs = np.argmax(predIdxs, axis=1)

# Se genera el informe de clasificacion
print(classification_report(testY.argmax(axis=1), predIdxs,
    target_names=lb.classes_))

# Se guarda el modelo
print("[INFO] saving mask detector model...")
model.save(args["model"], save_format="h5")


N = EPOCHS
plt.style.use("ggplot")
# Abrimos grafica
plt.figure()
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["accuracy"], label="train_accuracy")
plt.plot(np.arange(0, N), H.history["val_accuracy"], label="val_accuracy")
# Definimos titulo de la grafica, de los ejes y leyenda
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
# Guardamos la grafica, como hemos determinado en el analizador de argumentos
plt.savefig(args["plot"])
```

Figura 61-Algoritmo del entrenamiento del modelo 3

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

```
# Bibliotecas importadas
# Función para la adaptar el tipo de imagen a la entrada del modelo
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
# Funcion para convertir una imagen PIL en una matriz
from tensorflow.keras.preprocessing.image import img_to_array
# Biblioteca para cargar el modelo de predicción
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
# Biblioteca para el envio de correos electrónico (Mensaje)
from email.message import EmailMessage
# Biblioteca para mostrar hora y dia del negativo
from datetime import datetime
# Biblioteca para conecatar el bus i2c a una LCD
from RPLCD import i2c
from Datos import Email,password
# Configuracion de pines
import RPi.GPIO as GPIO
# Biblioteca para poder ver el tipo de foto y asi enviarla en un archivo adjunto
import imghdr
# Biblioteca  para el envio de correos electronicos (Servidor)
import smtplib
# Biblioteca para el manejo de matrices
import numpy as np
import imutils
# Función para los retrasos
import time
# OpenCV
import cv2
# Biblioteca para ver el numero de archivos que hay en un directorio determinado
import glob

 # Funcion de envio de correo electronico
 def Send_Email(Numero):

     #Enviar una imagen adjunta al correo, la leemos con rb(read byte)
     with open("Imagenes_Resultados/Imagenes_Negativos/image_{}.jpg".format(Numero),'rb') as f:
         file_data=f.read()
         # Tipo de imagen
         file_type=imghdr.what(f.name)
         file_name=f.name

     # Se adjunta imagen al correo
     msg.add_attachment(file_data,maintype='image',subtype=file_type, filename=file_name)

     #Puerto estandar de ssl 465, le damos el nombre smtp
     with smtplib.SMTP_SSL('smtp.gmail.com', 465) as smtp:
         smtp.login(email_adress,email_password)
         smtp.send_message(msg)
         smtp.quit()

     print("E-mail enviado")
     return

# Funcion de deteccion de caras y mascarillas
def detect_and_predict_mask(frame, faceNet, maskNet):

    # Se guarda el alto y ancho del frame
    (h, w) = frame.shape[:2]
    # Preprocesamiento del frame
    blob = cv2.dnn.blobFromImage(frame, 1, (300, 300),
        (104.0, 177.0, 123.0))
    # Resta media, normalizacion e intercambiar valores
    # frame= frame que se preprocesara antes de pasarla a través de nuestra red neuronal  para su clasificación
    # 1= Factor de escala
    # (300, 300) Tamaño de la red neuronal
    # RGB en ese orden, no RBG
    # Resta media se realiza con el objetivo de intercambiar valores
    # Blob devuelve la imagen despues de la resta media, normalizacion e intercambiar valores

    # Pasa la imagen a traves de la red neuronal
    faceNet.setInput(blob)
    # Se obtienen las probabilidades de que haya una cara segun la red de caras
    detections = faceNet.forward()
```

Figura 62-Algoritmo de detección de mascarilla 1

```
    # Inicialización lista de caras, sus correspondientes localizaciones,
    # y la lista de predicciones desde nuestra red de mascarillas
    faces = []
    locs = []
    preds = []

    for i in range(0, detections.shape[2]):

        # Se extrae la confianza asociada con la deteccion
        confidence = detections[0, 0, i, 2]

        # Se desechan las detecciones debiles de las caras
        if confidence > 0.5:

            # computa la x e y- que cordina el cuadro delimitador para el objeto
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
            (startX, startY, endX, endY) = box.astype("int") #Funcion Numpy

            # Se asegura que el cuadro delimitador esta dentro de las dimensiones del frame
            (startX, startY) = (max(0, startX), max(0, startY))
            (endX, endY) = (min(w - 1, endX), min(h - 1, endY))

            # Se extrae los puntos de referencia (ROI) de la cara
            face = frame[startY:endY, startX:endX]
            # Se cambia el orden de los colores
            face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
            # Se cambia el tamaño de la imagen para poder pasarla por la red neuronal
            face = cv2.resize(face, (224, 224))
            # Se convierte una instancia de imagen PIL(Python Image Library) en una matriz
            face = img_to_array(face)
            # Se cambia el formato de la imagen para poder pasarlo por la red neurnal
            face = preprocess_input(face)

            # Se añade la cara y el cuadro delimitante con su respectivas listas
            faces.append(face)
            locs.append((startX, startY, endX, endY))

    # Solo se realiza las predicciones si al menos una cara ha sido detectada
    if len(faces) > 0:

        # Para una mas rapida deduccion se realiza un grupo de predicciones
        # en todas las caras al mismo tiempo en vez de una predicion detras de otra
        faces = np.array(faces, dtype="float32")
        preds = maskNet.predict(faces, batch_size=32)

    #devuelve dos variables, la localizacion de las caras y su correpondiente localización
    return (locs, preds)

# Inicialización de las variables

#Modo al que nos referimos para definir las salidas y entradas de la Raspberry
GPIO.setmode(GPIO.BCM)

#Inicializacion constantes del envio de correo electronico

email_adress= Email                    #Definir correo desde el que se envia los correos
email_password= password               #Definir contrasea del correo
# Se inicializa fecha para enviar correo electronico
now=datetime.now()
fecha_actual=now.strftime('Persona sin mascarilla el dia %d del %m del %Y, a las %H:%M:%S')
msg= EmailMessage()

msg['Subject']='Imagen'
msg['From']=email_adress
msg['To']='imagenes.mascarilla@gmail.com'
msg.set_content('{}'.format(fecha_actual)) #Mensaje del correo (Cuerpo)
```

Figura 63-Algoritmo de detección de mascarilla 2

```
#Inicializar LCD
lcdmode = 'i2c'           #Puerto de entrada i2c
cols = 20                 #Numero de columnas LCD
rows = 4                  #Numero de filas LCD
charmap = 'A00'
i2c_expander = 'PCF8574'  #Adaptador puerto i2c

address = 0x27            #Direccion LCD
port = 1                  #Puerto

#Establecer inicio LCD
lcd = i2c.CharLCD(i2c_expander, address, port=port, charmap=charmap,
                  cols=cols, rows=rows)

lcd.cursor_mode='hide'                    #Establecer modo del cursor (Invisible)
lcd.cursor_pos=(0,0)                      #Establecer posicion del cursor
lcd.write_string('DETECTOR MASCARILLAS')
lcd.crlf()

print("[INFO] loading face detector model...")
lcd.write_string('Modelo caras...')
lcd.crlf()

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

print("[INFO] loading face mask detector model...")
lcd.write_string('Modelo mascarilla...')
lcd.crlf()


# Se carga el modelo de deteccion de mascarillas
maskNet = load_model("Modelo_mascarillas.model")

# Contadores de frames
Fases=[0,0,0,0]
# Fases[0], cuenta los frames cuando se detecta una cara CON mascarilla
# Fases[1], cuenta los frames cuando se detecta una cara SIN mascarilla
# Fases[2], cuenta los frames cuando no hay ninguna cara,
# así se inicializa las variables (Led,zumbador,LCD y porcentaje de reconocimiento)

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

# Se determina las salidas a la que se unen los LED y el zumbador
Led_activacion_camara=21
Led_positivo=12
Led_negativo=13
Buzzer=19

# Se configuran de los pines como salidas digitales
GPIO.setup(Led_activacion_camara,GPIO.OUT)
GPIO.setup(Led_positivo,GPIO.OUT)
GPIO.setup(Led_negativo,GPIO.OUT)
GPIO.setup(Buzzer,GPIO.OUT)

lcd.close(clear=True)
lcd.write_string('DETECTOR MASCARILLAS')
lcd.crlf()

# Se inicia el video en directo y se activa  la camara
print("[INFO] starting video stream...")
# vs = VideoStream(src=0).start()              #Utilizacion WEBCAM
vs = VideoStream(usePiCamera=True).start()   #Utilizacion PiCAM
time.sleep(5.0)
lcd.write_string('   DETECTOR LISTO')

GPIO.output(Led_activacion_camara,True)
```

Figura 64-Algoritmo de detección de mascarilla 3

```
# loop sobre los frames del video en directo
while True:

    Fases[2]+=1

    #En el momento recorra dos veces esta parte del programa inicializamos las variables
    if Fases[2]==2:

        lcd.close(clear=True)
        lcd.write_string('DETECTOR MASCARILLAS')

        Fases[0]=0
        Fases[1]=0
        GPIO.output(Buzzer,False)
        GPIO.output(Led_positivo,False)
        GPIO.output(Led_negativo,False)

    now=datetime.now()
    lcd.cursor_pos=(3,0)
    lcd.write_string(now.strftime('            %H:%M:%S'))

    # Se escoge el frame del video para analizarlo
    frame = vs.read()
    # Se cambia la anchura
    frame = imutils.resize(frame, width=500)

    # Se detecta las caras en el frame y determina si estos llevan puestas las mascarillas o no
    (locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)
    # loop sobre las caras detectadas y sus correspondientes localizaciones
    for (box, pred) in zip(locs, preds):

        # Se desempaqueta el cuadro delimitante y las predicciones
        (startX, startY, endX, endY) = box
        (mask, withoutMask) = pred

        if mask > withoutMask:

            label = "Mascarilla bien colocada"
            color = (0, 255, 0)
            print("Mask")

            #Contador de frames
            Fases[0]+=1
            Fases[1]=0
            Fases[2]=0

            if Fases[0]==1:

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

                lcd.close(clear=True)
                lcd.write_string('DETECTOR MASCARILLAS')
                lcd.crlf()
                lcd.cursor_pos=(1,0)
                lcd.write_string('Mascarilla Detectada')
                lcd.cursor_pos=(2,0)
                lcd.write_string('50%')

            if Fases[0]==2:

                lcd.cursor_pos=(2,0)
                lcd.write_string('100%')
```

Figura 65-Algoritmo de detección de mascarilla 4

```
                #Se inicializa el contador de frames
                Fases[0]=0

                #Texto que aparece en pantalla cuando se detecta el positivo
                label = "Detección Completada"

                lcd.close(clear=True)
                lcd.write_string('DETECTOR MASCARILLAS')

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

        else:

            label = "No se detecta mascarilla"
            color = (0, 0, 255)
            print("No mask")

            #Contador de frames
            Fases[1]+=1
            Fases[0]=0
            Fases[2]=0

            if Fases[1]==1:

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

                lcd.close(clear=True)
                lcd.write_string('DETECTOR MASCARILLAS')
                lcd.crlf()
                lcd.cursor_pos=(1,0)
                lcd.write_string('Mask No Detectada')

            lcd.cursor_pos=(2,0)
            lcd.write_string('50%')

            #Frames tenemos que estar delante de la camara para que nos haga la captura (se puede modificar)
            if Fases[1]==2:

                lcd.cursor_pos=(2,0)
                lcd.write_string('100%')

                # Se inicializa el contador de frames
                Fases[1]=0

                # Texto que aparece en pantalla cuando se detecta el positivo
                label = "Deteccion Completa.No mascarilla"

                # Envio correo elecetronico
                # Actualizamos fecha y hora, para enviarla de manera adjunta
                now=datetime.now()
                fecha_actual=now.strftime('Persona sin mascarilla el dia %d del %m del %Y, a las %H:%M:%S ')
                print(fecha_actual)

                # Directorio en el  que guardamos la captura
                img_name_neg="Imagenes_Resultados/image_0.jpg"
                cv2.imwrite(img_name_neg,frame )
                lcd.cursor_pos=(3,0)
                lcd.write_string('Enviando correo...  ')

                Send_Email(img_name_neg)

                remove("Imagenes_Resultados/image_0.jpg")

                lcd.close(clear=True)
                lcd.write_string('DETECTOR MASCARILLAS')

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

Figura 66-Algoritmo de detección de mascarilla 5

```
        #Posicion del texto y rectangulo en el frame
        cv2.putText(frame, label, (startX-50, startY - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)
        cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)

    #Mostrar el frame en una ventana
    cv2.imshow("Detector de mascarillas", frame)

    # Si se pulsa el boton 'escape' se cierra la ventana y el programa
    # Se el waitKey(1) ya que así mostrará un marco durante 1ms,
    # después de lo cual la pantalla se cerrará automáticamente
    # 1 Ms es el minimo que nos da el SO, si hay mas programas ejecutandose este tiempo aumentara
    k=cv2.waitKey(1)

    if k==27:
        break



# Se apaga el led activación
GPIO.output(Led_activacion_camara,False)
# Limpieza antes del apagado
lcd.close(clear=True)    #Apagamos LCD
GPIO.cleanup()           #Limpiamos pines
cv2.destroyAllWindows()  #Cerramos ventana de los frames
vs.stop()
```

Figura 67-Algoritmo de detección de mascarilla 6