



**INSTITUTO POLITÉCNICO
NACIONAL**



**ESCUELA SUPERIOR DE
INGENIERÍA MECÁNICA Y
ELÉCTRICA
UNIDAD ZACATENCO**

INGENIERÍA EN COMUNICACIONES Y ELECTRÓNICA

**DETECTOR DE MASCARILLAS MEDIANTE
RASPBERRY PI4**

PROYECTO TERMINAL

M. en C. Galicia Galicia Roberto

PRESENTA:

Perla Aceneth Chávez Barbosa

PYTHON

Es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones de todo tipo. A diferencia de otros lenguajes como Java o .NET, se trata de un lenguaje interpretado, es decir, que no es necesario compilarlo para ejecutar las aplicaciones escritas en Python, sino que se ejecutan directamente por el ordenador utilizando un programa denominado interpretador, por lo que no es necesario “traducirlo” a lenguaje máquina.



Es un lenguaje sencillo de leer y escribir debido a su alta similitud con el lenguaje humano. Además, se trata de un lenguaje multiplataforma de código abierto

OPENCV

Es una biblioteca de código abierto que contiene implementaciones que abarcan más de 2500 algoritmos. Además, está especializada en el sistema de visión artificial y machine learning.

Por ello, este sistema viene a facilitarnos el uso de IA, en este caso es importante mencionar que la visión artificial, siendo esta una parte de la inteligencia artificial que está especializada en el desarrollo de métodos que permiten a las máquinas y sistemas realizar actividades como:

- Ver imágenes de forma similar a como lo haría la visión humana.
- Identificar y procesar estas imágenes para analizar y medir datos con los cuales se puedan supervisar procesos y tomar decisiones.



Su utilización principal abarca la detección de objetos y rostros, especialmente en aspectos como la seguridad, el marketing o incluso la fotografía.

OpenCV ofrece soporte para varios sistemas operativos y varias arquitecturas de hardware, pero

también ofrece el código fuente para que cualquier desarrollador lo compile en cualquier sistema

operativo y arquitectura particular.

OpenCV brinda instrucciones de instalación y opcionalmente binarios para estos sistemas operativos

- GNU/Linux
- Windows

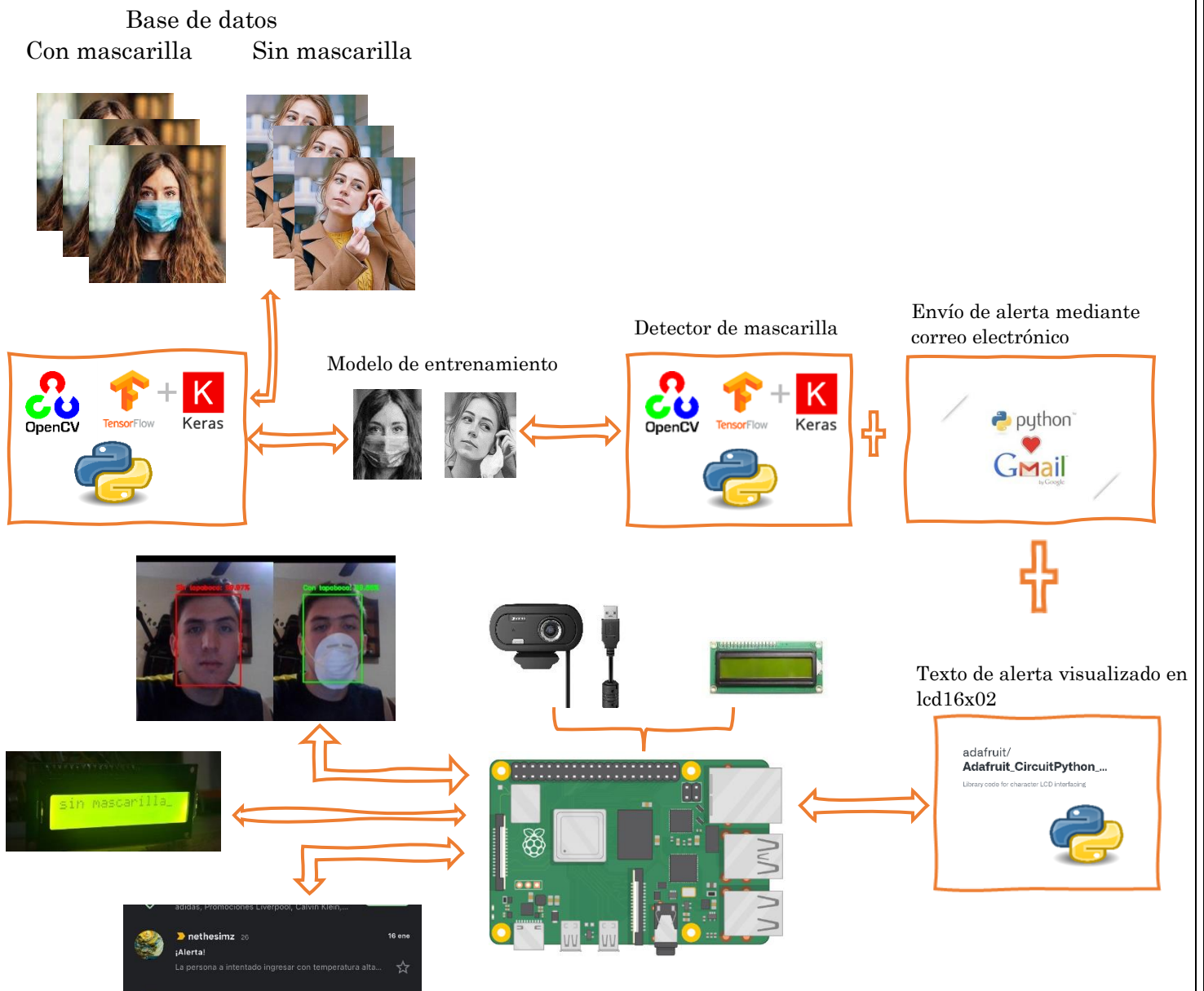
- MacOS
- Android
- iOS

OpenCV se utiliza prácticamente en cualquier plataforma con capacidad suficiente, compilando su código fuente. De este modo se la utiliza en PC, celulares y placas de prototipado como Raspberry Pi o NVidia Jetson.

En OpenCv se puede hacer desde abrir una imagen, modificar a escala de grises una imagen

(Esto nos ayuda para evitar distracciones en la clasificación de imágenes faciales, esto es importante pues más adelante lo mencionaremos a detalle)

DIAGRAMA



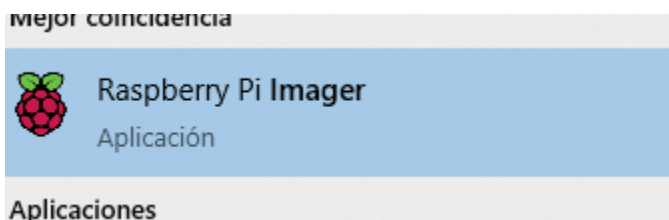
MATERIAL:

- Cámara externa “Acteck”
- Protoboard
- Pantalla LCD 16x02
- Potenciómetro 10kohms
- Resistencias 4.7kohms
- Cables de conexión H-M y M-M
- Sensor De Temperatura Infrarrojo Gy-906 Mlx90614
- Raspberry pi 4
- Memoria microSD
- Teclado y mouse inalámbrico
- Pantalla o monitor con entrada HDMI
- Cable HDMI
- Cable de carga Raspberry pi
- Computadora portátil o de escritorio

Instalación Raspbian OS

Comenzaremos instalando el ambiente Raspbian para poder comenzar a trabajar con nuestra Raspberry pi.

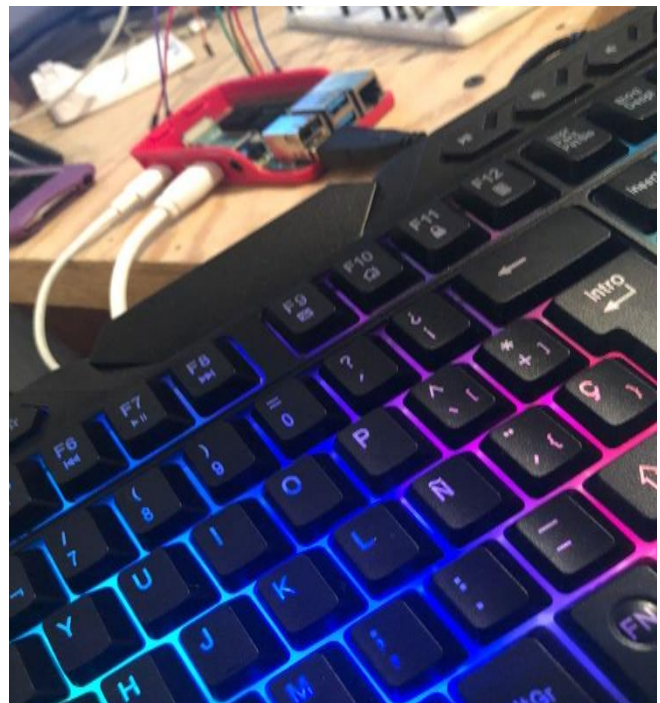
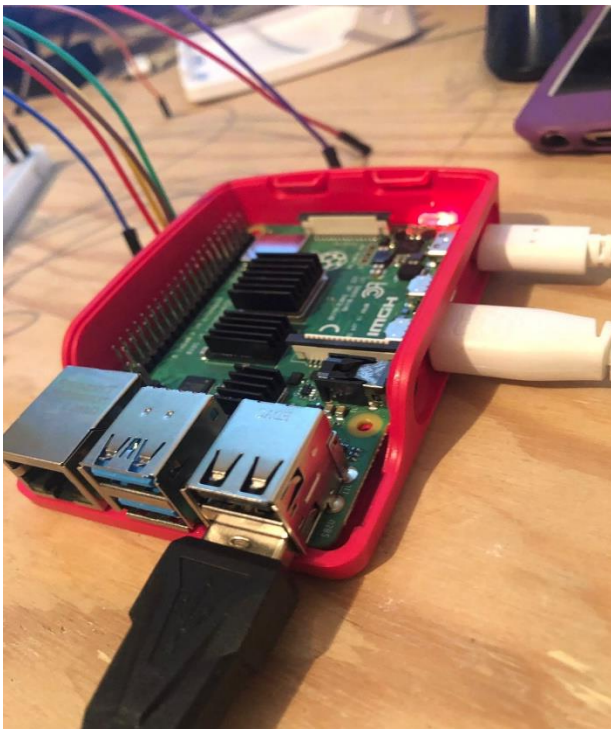
Comenzamos descargando en nuestra computadora el Raspberry pi Imager para poder grabar nuestra tarjeta uSD con Raspbian:



Posteriormente, cargamos la versión que mejor nos acomode, en este caso y para evitar conflictos mayores que se puedan presentar, utilizaremos la versión “Buster” de 32bits, pero es importante mencionar que hasta hoy día, la versión más actualizada es “Bullseye”.



Al terminar la escritura en nuestra uSD, podremos conectarla a la entrada de nuestra Raspberry, así como conectar la Raspberry a un monitor, mediante el cable HDMI y conectarlo a una fuente mediante el cable de fuente de poder.



Para continuar con la configuración, como la implementación de nuestra red wifi, conectamos el teclado y mouse inalámbricos.

En esta ocasión, no indagaremos a profundidad en la instalación de Raspbian puesto que afortunadamente existen videos explicativos en gran cantidad donde podemos visualizarlo mejor:

Descarga de Raspbian:

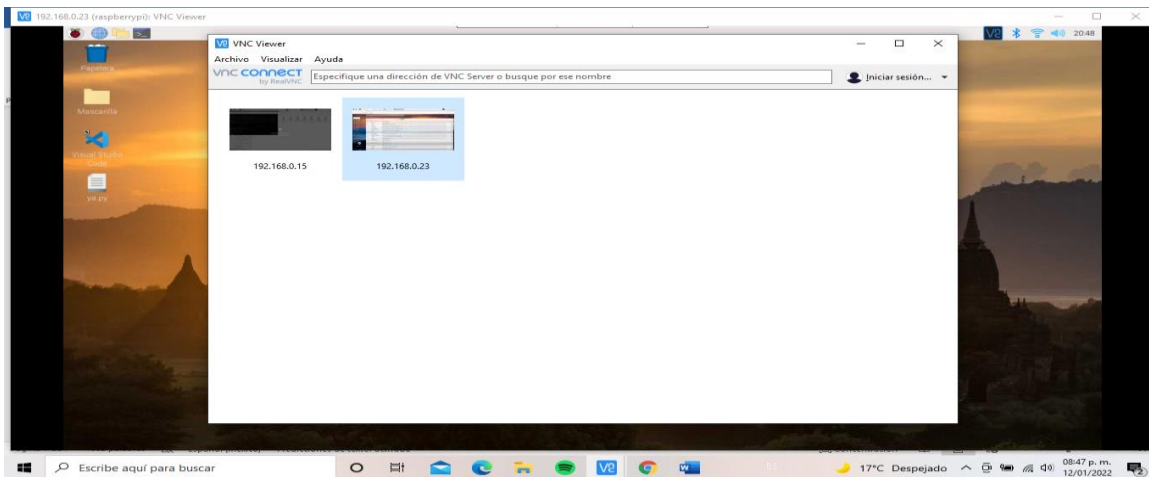
<https://www.raspberrypi.com/software/>

Instalación de Raspbian:

<https://youtu.be/eds99EjKd8M>

INSTALACIÓN DE PAQUETES

Después de concluir con la instalación correcta de nuestra Raspberry pi, comenzamos a instalar los programas a utilizar, para mayor comodidad, se configuró mediante **VNC la computadora portátil** con la Raspberry. Gracias a ello, podremos manipular la interfaz de Raspbian mediante el teclado y mouse de nuestra computadora, por lo que ahora está demás su utilización.



Comenzamos abriendo la terminal, antes de instalar **OpenCV** en su Raspberry Pi 4:

La fundación Raspberry Pi ha lanzado recientemente un software nuevo y mejorado para estas EEPROM. Esto no tiene nada que ver con OpenCV, sino más bien con la disipación de calor. En una de nuestras aplicaciones de visión, el calor de la CPU cae de 65 °C (149 °F) a 48 °C (118 °F) simplemente actualizando el contenido de las EEPROM. Y, como sabe, una temperatura baja de la CPU prolongará la vida útil de su Pi. Para obtener más información, consulte este artículo .

Verificaremos y si es necesario, actualice las EEPROM con los siguientes comandos.

```
sudo rpi-eeprom-update
```

```
if needed, to update the firmware
```

```
sudo rpi-eeprom-update -a
```

```
sudo reboot
```



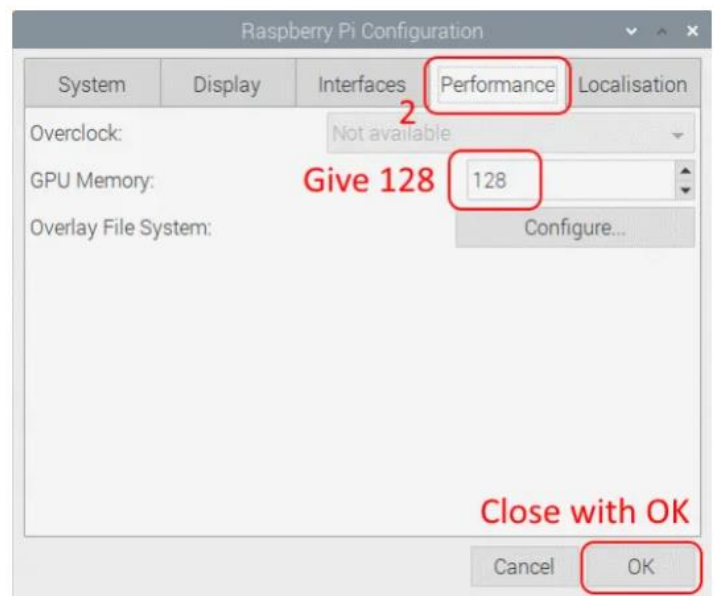
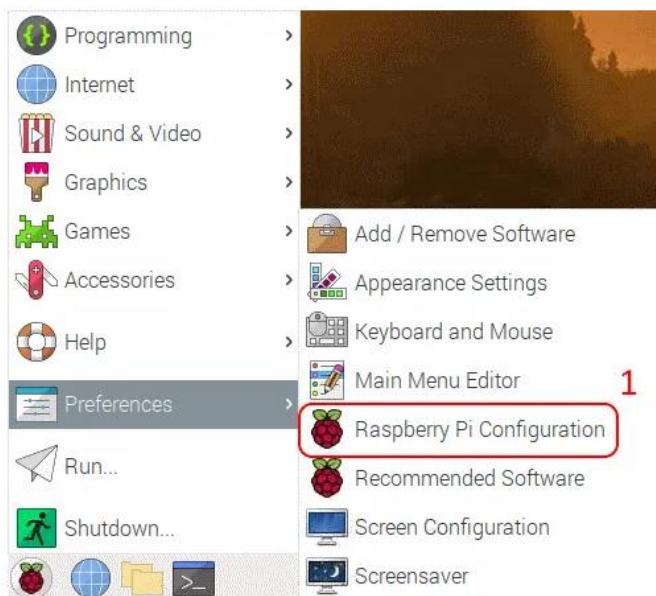
```
pi@raspberrypi: ~  
File Edit Tabs Help  
pi@raspberrypi:~ $ sudo rpi-eeprom-update  
BCM2711 detected  
BOOTLOADER: up-to-date  
CURRENT: Tue 10 Sep 2019 10:41:50 AM UTC (1568112110)  
LATEST: Tue 10 Sep 2019 10:41:50 AM UTC (1568112110)  
FW DIR: /lib/firmware/raspberrypi/bootloader/critical  
VL805: up-to-date  
CURRENT: 000137ad  
LATEST: 000137ad  
pi@raspberrypi:~ $
```

Just fine !

Para el desarrollo de este proyecto utilizaremos OpenCV lanzó la versión 4.5.5. pues es la versión que nos permite eliminar el mayor número de errores a la hora de programar y compilar en Python, pero hay un problema a mencionar. El script de instalación de Python3 se modifica incorrectamente. Colocará todas las bibliotecas en el directorio incorrecto y Python no las encontrará. Al agregar -D PYTHON3_PACKAGES_PATH=/usr/lib/python3/dist-packages a la compilación, las bibliotecas se almacenan en la carpeta correcta.

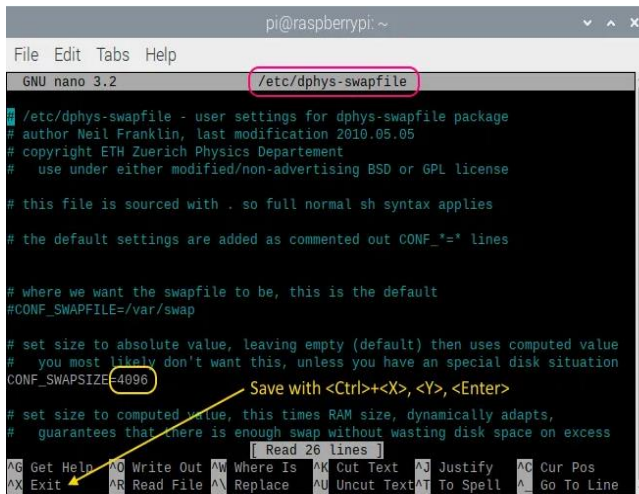
Para su correcta instalación seguiremos los siguientes pasos:

El chip de RAM físico es utilizado tanto por la CPU como por la GPU. En una Raspberry Pi 2 o 3, el valor predeterminado es de 64 Mbytes asignados para la GPU. El Raspberry Pi 4 tiene un tamaño de memoria GPU de 76 Mbytes. Puede resultar algo pequeño para proyectos de visión, por lo que lo cambiaremos a 128 Mbyte. Después de esta acción, el sistema quiere reiniciarse.

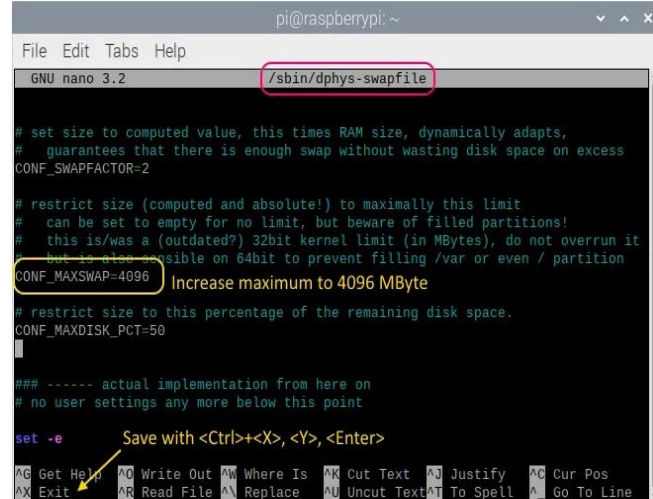


El siguiente paso es aumentar su espacio de intercambio. OpenCV necesita mucha memoria para compilar. Las últimas versiones quieren ver un mínimo de 6,5 GB de memoria antes de construir. Su espacio de intercambio está limitado a 2048 MB por defecto. Para superar este límite de 2048 MByte, deberá aumentar este máximo en el archivo `/sbin/dphys-swapfile`
`sudo nano /sbin/dphys-swapfile`

`sudo nano /etc/dphys-swapfile`
`sudo reboot`



The screenshot shows the nano text editor editing the file `/etc/dphys-swapfile`. The file contains configuration for the dphys-swapfile package. A yellow box highlights the line `CONF_SWAPSIZE=4096`. A yellow arrow points from the text "Save with <Ctrl>+<X>, <Y>, <Enter>" to the bottom status bar of the nano editor.



The screenshot shows the nano text editor editing the file `/sbin/dphys-swapfile`. The file contains configuration for the dphys-swapfile package. A yellow box highlights the line `CONF_MAXSWAP=4096`. A yellow arrow points from the text "Save with <Ctrl>+<X>, <Y>, <Enter>" to the bottom status bar of the nano editor.

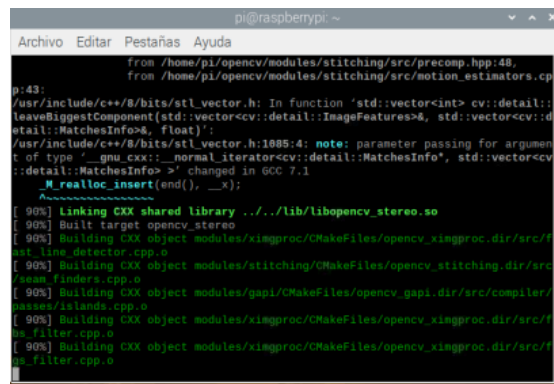
Ahora vamos a **instalar opencv**, tecleando en la terminal script de instalación que ejecuta todos los comandos de esta guía a la vez. Toda la instalación tardará una hora y media en completarse.

`free -m`

`wget https://github.com/Qengineering/Install-OpenCV-Raspberry-Pi-32-bits/raw/main/OpenCV-4-5-5.sh`

`sudo chmod 755 ./OpenCV-4-5-5.sh`

`./OpenCV-4-5-5.sh`



The screenshot shows the terminal output of the OpenCV installation script. It displays the progress of building and linking various OpenCV modules, including `opencv_stereo`, `opencv_ximgproc`, and `opencv_ximgproc.dir/src/line_detector.cpp.o`. The output shows that the installation is progressing well, with many modules being built successfully.

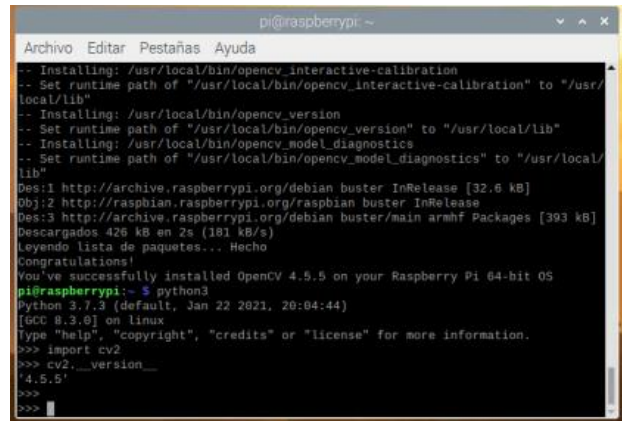
Al terminar, simplemente verificamos que la instalación haya sido exitosa tecleando:

`Python3`

`>>Import cv2`

`>>cv2.__version__`

Si todo está correcto, debe aparecer de esta forma



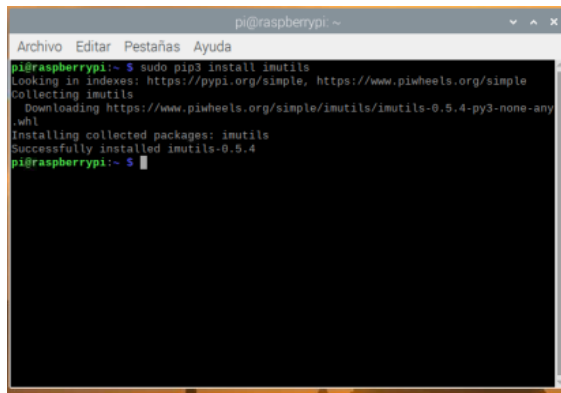
```
pi@raspberrypi:~$ sudo apt-get install python3-opencv
-- Installing: /usr/local/bin/opencv_interactive-calibration
-- Set runtime path of "/usr/local/bin/opencv_interactive-calibration" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_version
-- Set runtime path of "/usr/local/bin/opencv_version" to "/usr/local/lib"
-- Installing: /usr/local/bin/opencv_model_diagnostics
-- Set runtime path of "/usr/local/bin/opencv_model_diagnostics" to "/usr/local/lib"
Des:1 http://archive.raspberrypi.org/debian buster InRelease [32.6 kB]
Obj:2 http://raspbian.raspberrypi.org/raspbian buster InRelease
Des:3 http://archive.raspberrypi.org/debian buster/main armhf Packages [393 kB]
Descargados 426 kB en 2s (181 kB/s)
leyendo lista de paquetes... Hecho
Congratulaciones!
You've successfully installed OpenCV 4.5.5 on your Raspberry Pi 64-bit OS
pi@raspberrypi:~$ python3
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'4.5.5'
>>>
```

Instalación de *Imutils*.

Imutils es una serie de funciones de conveniencia para acelerar la computación OpenCV en la Raspberry Pi.

Tecleamos el siguiente comando en la terminal de la Raspberry pi:

sudo pip3 install imutils



```
pi@raspberrypi:~$ sudo pip3 install imutils
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting imutils
  Downloading https://www.piwheels.org/simple/imutils/imutils-0.5.4-py3-none-any.whl
Installing collected packages: imutils
Successfully installed imutils-0.5.4
pi@raspberrypi:~$
```

Instala *TensorFlow*.

Tensorflow es una plataforma de aprendizaje automático de código abierto. Facilita la creación de modelos de aprendizaje automático para computadoras de escritorio, dispositivos móviles, la web y la nube, sin importar si eres principiante o experto.

Para la instalación de *TensorFlow*, tecleamos el siguiente comando en la terminal:

sudo pip3 install https://github.com/lhelontra/tensorflow-on-arm/releases/download/v2.1.0/tensorflow-2.1.0-cp37-none-linux_armv7l.whl

Yo ya lo tenía instalado por lo que sólo verificamos que esté dentro de Python



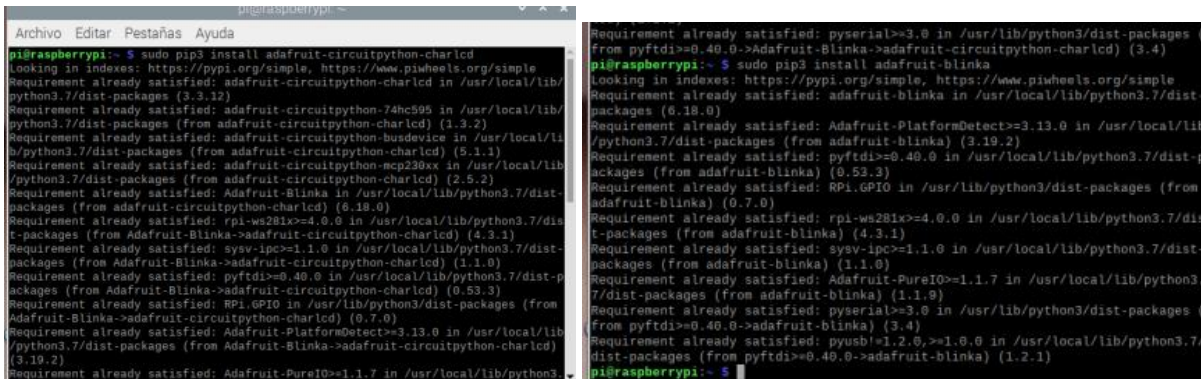
```
pi@raspberrypi:~$ python3
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import tensorflow
>>>
```

Instalar *Adafruit_CircuitPython_CharLCD* y *Adafruit_Blinka*.

Es fácil usar una pantalla LCD de caracteres con CircuitPython o Python y el módulo Adafruit CircuitPython CharLCD . Este módulo le permite escribir fácilmente código Python que controla una pantalla LCD de caracteres (ya sea retroiluminación simple o retroiluminación RGB). Puede usarlos con cualquier placa de microcontrolador CircuitPython o con una computadora que tenga GPIO y Python gracias a Adafruit_Blinka, nuestra biblioteca de compatibilidad CircuitPython-for-Python.

En este código hacemos uso de los paquetes de Python3 *Adafruit_CircuitPython_CharLCD* y *Adafruit_Blinka*. Para instalarlos ejecuta los siguientes comandos en la terminal:

```
sudo pip3 install adafruit-circuitpython-charlcd
sudo pip3 install adafruit-blinka
```

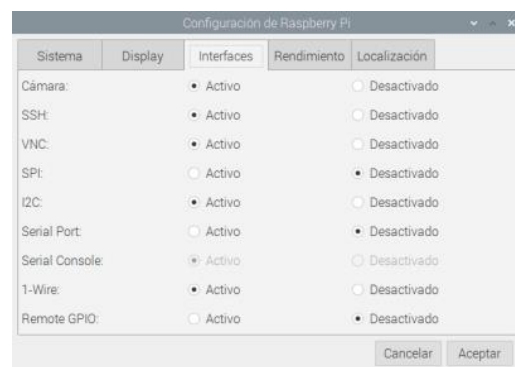


```
pi@raspberrypi:~$ sudo pip3 install adafruit-circuitpython-charlcd
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: adafruit-circuitpython-charlcd in /usr/local/lib/python3.7/dist-packages (3.3.12)
Requirement already satisfied: adafruit-circuitpython-74hc595 in /usr/local/lib/python3.7/dist-packages (from adafruit-circuitpython-charlcd) (3.3.2)
Requirement already satisfied: adafruit-circuitpython-busdevice in /usr/local/lib/python3.7/dist-packages (from adafruit-circuitpython-charlcd) (5.1.1)
Requirement already satisfied: adafruit-circuitpython-mcp230xx in /usr/local/lib/python3.7/dist-packages (from adafruit-circuitpython-charlcd) (2.5.2)
Requirement already satisfied: Adafruit-Blinka in /usr/local/lib/python3.7/dist-packages (from adafruit-circuitpython-charlcd) (6.18.0)
Requirement already satisfied: rpi-ws281x>=4.0.0 in /usr/local/lib/python3.7/dist-packages (from Adafruit-Blinka->adafruit-circuitpython-charlcd) (4.3.3)
Requirement already satisfied: sysv-ipc>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from Adafruit-Blinka->adafruit-circuitpython-charlcd) (1.1.0)
Requirement already satisfied: pyftdi>=0.40.0 in /usr/local/lib/python3.7/dist-packages (from Adafruit-Blinka->adafruit-circuitpython-charlcd) (0.53.3)
Requirement already satisfied: RPi.GPIO in /usr/lib/python3/dist-packages (from Adafruit-Blinka->adafruit-circuitpython-charlcd) (0.7.0)
Requirement already satisfied: Adafruit-PureIO>=1.1.7 in /usr/local/lib/python3.7/dist-packages (from Adafruit-Blinka->adafruit-circuitpython-charlcd) (3.19.2)
Requirement already satisfied: Adafruit-PureIO>=1.1.7 in /usr/local/lib/python3.7/dist-packages (from Adafruit-Blinka->adafruit-circuitpython-charlcd) (3.19.2)
pi@raspberrypi:~$ sudo pip3 install adafruit-blinka
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: adafruit-blinka in /usr/local/lib/python3.7/dist-packages (6.18.0)
Requirement already satisfied: Adafruit-PlatformDetect>=3.13.0 in /usr/local/lib/python3.7/dist-packages (from adafruit-blinka) (3.19.2)
Requirement already satisfied: pyftdi>=0.40.0 in /usr/local/lib/python3.7/dist-packages (from adafruit-blinka) (0.53.3)
Requirement already satisfied: RPi.GPIO in /usr/lib/python3/dist-packages (from adafruit-blinka) (0.7.0)
Requirement already satisfied: rpi-ws281x>=4.0.0 in /usr/local/lib/python3.7/dist-packages (from adafruit-blinka) (4.3.1)
Requirement already satisfied: sysv-ipc>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from adafruit-blinka) (1.1.0)
Requirement already satisfied: Adafruit-PureIO>=1.1.7 in /usr/local/lib/python3.7/dist-packages (from adafruit-blinka) (1.1.0)
Requirement already satisfied: pyserial>=3.0 in /usr/lib/python3/dist-packages (from pyftdi>=0.40.0->adafruit-blinka) (3.4)
Requirement already satisfied: pyusb>=1.2.0,>=1.0.0 in /usr/local/lib/python3.7/dist-packages (from pyftdi>=0.40.0->adafruit-blinka) (1.2.1)
```

En este caso, ya se encuentran instalados, por lo que aparece del modo anterior.

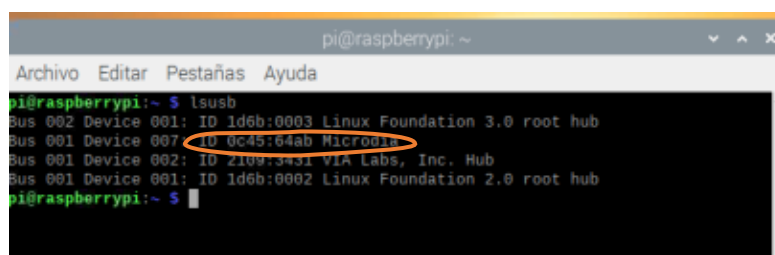
Detección de cámara

Después de instalar las librerías pertinentes, se hará la detección de la cámara externa, iniciamos habilitando la cámara en nuestra Raspberry:



Si no está “activo” el apartado de “cámara”, actívelo y después reinicie el sistema.

Seguido de ello, detectamos el puerto donde se encuentra la cámara, mediante el código de *lsusb*



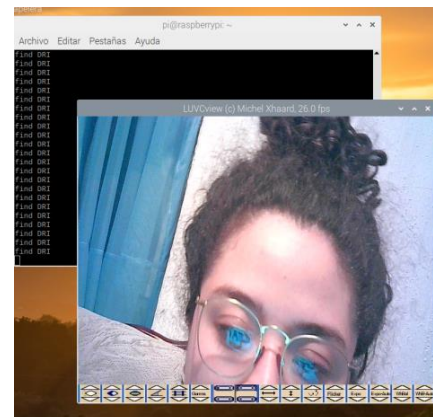
Al tener detectada la cámara mostraremos más información del puerto USB con el comando:

lsusb -d0c45:64ab -v

```
pi@raspberrypi: ~  
Archivo Editar Pestañas Ayuda  
-V, --version  
    Show version of program  
-h, --help  
    Show usage and help  
pi@raspberrypi:~$ lsusb -d0c45:64ab -v  
Bus 001 Device 007: ID 0c45:64ab Microdia  
Couldn't open device, some information will be missing  
Device Descriptor:  
  bLength                18  
  bDescriptorType        1  
  bcdUSB                  2.01  
  bDeviceClass            239 Miscellaneous Device  
  bDeviceSubClass         2  
  bDeviceProtocol         1 Interface Association  
  bMaxPacketSize0         64  
  idVendor                0x0c45 Microdia  
  idProduct               0x64ab  
  bcdDevice               6.22  
  iManufacturer          1  
  iProduct                2  
  iSerial                 0  
  bNumConfigurations      1  
  Configuration Descriptor:
```

Se agregará el comando `sudo apt-get install luvview` para que instale el visor de la cámara USB en la Raspberry y ejecutamos la cámara con el comando *luvcview*

```
pi@raspberrypi: ~  
Archivo Editar Pestañas Ayuda  
Synch Type               Asynchronous  
Usage Type               Data  
wMaxPacketSize           0x13fc 3x 1020 bytes  
bInterval                1  
pi@raspberrypi:~$ sudo apt-get install luvview  
Leyendo lista de paquetes... Hecho  
Creando árbol de dependencias  
Leyendo la información de estado... Hecho  
Se instalarán los siguientes paquetes NUEVOS:  
  luvview  
0 actualizados, 1 nuevos se instalarán, 0 para eliminar y 9 no actualizados.  
Se necesita descargar 50.1 kB de archivos.  
Se utilizarán 122 kB de espacio de disco adicional después de esta operación.  
Des:1 http://raspbrian.mirrors.lucidnetworks.net/raspbian buster/main armhf luvview  
  armhf 1:0.2.6-6 [50.1 kB]  
Descargados 50.1 kB en 1s (37.4 kB/s)  
Seleccionando el paquete luvview previamente no seleccionado.  
(Leyendo la base de datos ... 111894 ficheros o directorios instalados actualmen  
te.)  
Preparando para desempaquetar .../luvcview_1:0.2.6-6_armhf.deb ...  
Desempaquetando luvview (1:0.2.6-6) ...  
Configurando luvview (1:0.2.6-6) ...  
Procesando disparadores para man-db (2.8.5-2) ...  
pi@raspberrypi:~$
```



DESARROLLO

¿Cómo funciona el entrenamiento?

Al hacer zoom infinito en una imagen se podría observar que está conformada de pequeños recuadros a los cuales llamamos pixeles, que es la construcción fundamental de cualquier imagen, cuando decimos que una imagen tiene resolución de 1280x1200 quiere decir que tiene 1280 pixeles en el eje de las “x”, y 1200 en el eje de las “y”.

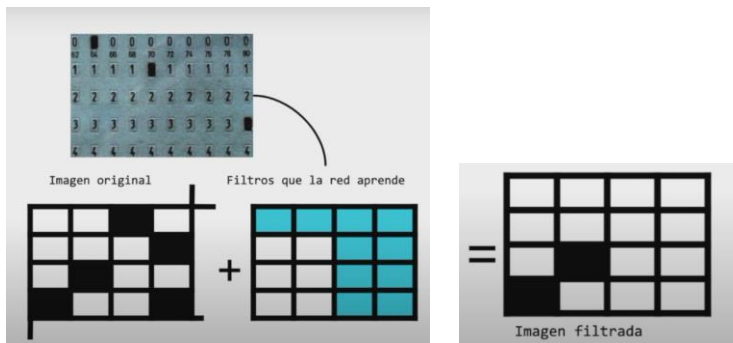
Un píxel es un valor numérico, en el caso de una imagen en blanco y negro, el pixel es un valor entre 0 y 255, (blanco 0 y negro 255) y lo de en medio es la escala de grises.

Cuando las imágenes son a color, los pixeles se forman de tres superpuestos, uno en color roja, verde y azul. Y al hacer juegos entre cada uno de ellos genera toda la gama de colores.

Pixel values

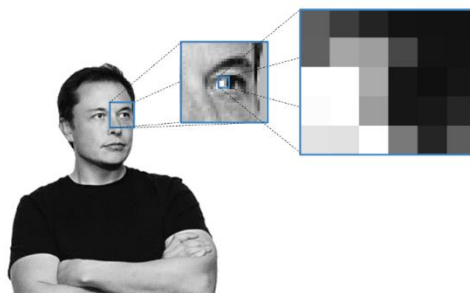


Para que un algoritmo de IA aprenda una clasificación, necesita las características, de esta matriz se extraen esas características mediante la convolución, esta consiste en super poner una imagen sobre la imagen que tenemos, de forma que solo dejara pasar alguna serie de pixeles. Es un filtro.



Con ello podremos obtener:

El tamaño de los filtros es siempre menor al de la imagen, y se aplican de manera progresiva, es decir si tenemos una imagen de 1000 pixeles, y el filtro es de 3x3 pues el filtro se aplicará de 3 x3 hasta haber recorrido toda la imagen.

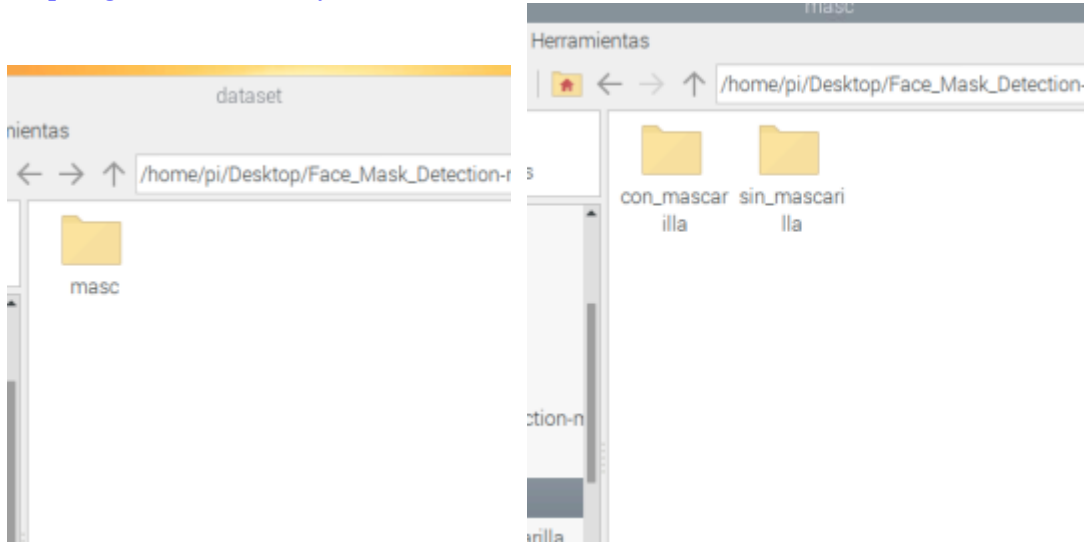


Desarrollaremos en Python, **vamos a entrenar un clasificador:**

Creamos una base de datos (dataset) con dos clases, por un lado, tendremos imágenes a color de rostros con mascarillas, y en otro directorio imágenes de rostros de personas sin mascarillas.

La cual fue obtenida de Github

<https://github.com/GabySol/OmesTutorials2021/tree/main/Mascarillas%20dataset>



#Importamos los paquetes necesarios

#Keras es una biblioteca de Redes Neuronales de Código Abierto escrita en Python. Es capaz de ejecutarse sobre *TensorFlow*, *Microsoft Cognitive Toolkit* o *Theano*.

#Está especialmente diseñada para posibilitar la experimentación en más o menos poco

#tiempo con redes de Aprendizaje Profundo.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import AveragePooling2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.utils import to_categorical
from sklearn.preprocessing import LabelBinarizer
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import os
```



```

#Construimos el analizador de argumentos y analizamos los mismos 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_detector1k.model",
help="path to output face mask detector model")
args = vars(ap.parse_args())

# tomamos la lista de imágenes en nuestro directorio de conjunto de datos, luego inicializamos
# la lista de datos (es decir, imágenes) e imágenes de clase
print("[INFO] loading images...")
imagePaths = list(paths.list_images(args["dataset"]))
data = []
labels = []

# Recorremos las rutas de la imagen
for imagePath in imagePaths:
#extraemos la etiqueta de clase del nombre del archivo
label = imagePath.split(os.path.sep)[-2]

#Cargamos la imagen de entrada (224x224) y preprocesamos
image = load_img(imagePath, target_size=(224, 224))
image = img_to_array(image)
image = preprocess_input(image)

#Actualizamos las listas de datos y etiquetas, respectivamente
data.append(image)
labels.append(label)

#Convertir los datos y las etiquetas en matrices NumPy
data = np.array(data, dtype="float32")
labels = np.array(labels)

# realizar una codificación one-hot en las etiquetas
lb = LabelBinarizer()
labels = lb.fit_transform(labels)
labels = to_categorical(labels)

#dividimos los datos en divisiones de entrenamiento y prueba usando el 75% de
# los datos para entrenamiento y el 25% restante para prueba
(trainX, testX, trainY, testY) = train_test_split(data, labels,
test_size=0.20, stratify=labels, random_state=42)

#construimos el generador de imágenes de entrenamiento
aug = ImageDataGenerator(

```

```

rotation_range=20,
zoom_range=0.15,
width_shift_range=0.2,
height_shift_range=0.2,
shear_range=0.15,
horizontal_flip=True,
fill_mode="nearest")

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

#construimos la cabeza del model que se colocará encima del model base
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)

# colocamos el model FC de la cabeza encima del model base (esto se convertirá
#en el modelo real que entrenaremos)
model = Model(inputs=baseModel.input, outputs=headModel)

#recorremos todas las capas en el model base y congelarlas para
#no ser actualizado durante el primer proceso de entrenamiento
for layer in baseModel.layers:
    layer.trainable = False

#Compilamos nuestro modelo
print("[INFO] compiling model...")
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
metrics=["accuracy"])
print("[INFO] training head...")
H = model.fit(
aug.flow(trainX, trainY, batch_size=BS),
steps_per_epoch=len(trainX) // BS,
validation_data=(testX, testY),
validation_steps=len(testX) // BS,
epochs=EPOCHS)

# hacemos predicciones sobre el conjunto de prueba
print("[INFO] evaluating network...")
predIdxs = model.predict(testX, batch_size=BS)

# para cada imagen en el conjunto de prueba, necesitamos encontrar el índice de la
# etiqueta con la mayor probabilidad predicha correspondiente
predIdxs = np.argmax(predIdxs, axis=1)

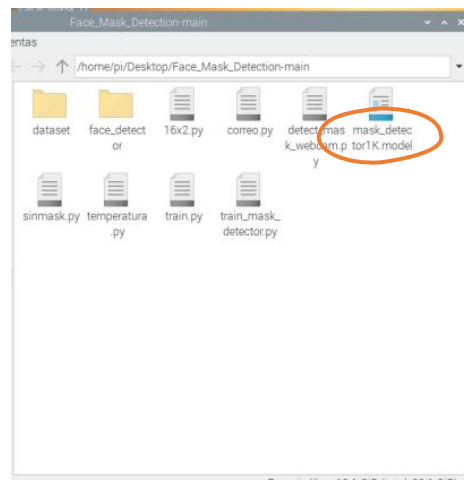
```

```
#mostrar un informe de clasificación bien formado
print(classification_report(testY.argmax(axis=1), predIdxs,
target_names=lb.classes_))

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

#Trazar la pérdida de entrenamiento y la precisión
N = EPOCHS
plt.style.use("ggplot")
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")
plt.title("Training Loss and Accuracy")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```

Al ejecutar nuestro entrenamiento, se generará un archivo XML que será nuestro “ejecutable” para el programa principal.



Ahora comenzamos a programar nuestro detector:

```
#Importamos los paquetes necesarios

from tensorflow.keras.applications.mobilenet_v2 import preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model
from imutils.video import VideoStream
```

```

import numpy as np
import argparse
import imutils
import time
import cv2
import os

def detect_and_predict_mask(frame, faceNet, maskNet):
# toma las dimensiones del marco y luego construye un blob de eso
(h, w) = frame.shape[:2]
blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300),
(104.0, 177.0, 123.0))

# pasar el blob a través de la red y obtener las detecciones de rostros
faceNet.setInput(blob)
detections = faceNet.forward()

# inicializar nuestra lista de caras, sus ubicaciones correspondientes, y la lista de predicciones de nuestra
#red de mascarillas
faces = []
locs = []
preds = []

#generamos un bucle sobre las detecciones
for i in range(0, detections.shape[2]):
# extraemos la confianza (es decir, la probabilidad) asociada con
# la detección
confidence = detections[0, 0, i, 2]

#filtramos las detecciones débiles asegurándonos de que la confianza sea
# mayor que la confianza mínima
if confidence > args["confidence"]:
# calculamos las coordenadas (x, y) del 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")

# Aseguramos que los cuadros delimitadores estén dentro de las dimensiones de
# el marco
(startX, startY) = (max(0, startX), max(0, startY))
(endX, endY) = (min(w - 1, endX), min(h - 1, endY))

#Extraemos el ROI de la cara, conviértalo de BGR a canal RGB
# ordenamos, redimensionamos a 224x224 y preprocesarlo
face = frame[startY:endY, startX:endX]
face = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
face = cv2.resize(face, (224, 224))
face = img_to_array(face)
face = preprocess_input(face)

```

```

#Agregamos la cara y los cuadros delimitadores a sus respectivas listas
faces.append(face)
locs.append((startX, startY, endX, endY))

#Sólo hará predicciones si se detectó al menos una cara
if len(faces) > 0:
# para una inferencia más rápida, haremos predicciones por lotes en todas las
# caras al mismo tiempo en lugar de predicciones una por una
# en el ciclo anterior `for`
faces = np.array(faces, dtype="float32")
preds = maskNet.predict(faces, batch_size=32)

#Devolvemos una tupla de 2 de las ubicaciones de las caras y sus correspondientes ubicaciones
return (locs, preds)

# construimos el analizador de argumentos y analizamos los argumentos
ap = argparse.ArgumentParser()
ap.add_argument("-f", "--face", type=str,
default="face_detector",
help="path to face detector model directory")
ap.add_argument("-m", "--model", type=str,
default="mask_detector.model",
help="path to trained face mask detector model")
ap.add_argument("-c", "--confidence", type=float, default=0.5,
help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# cargamos nuestro modelo serializado de detector de rostros desde el disco
print("[INFO] loading face detector model...")
prototxtPath = os.path.sep.join([args["face"], "deploy.prototxt"])
weightsPath = os.path.sep.join([args["face"],
"res10_300x300_ssd_iter_140000.caffemodel"])
faceNet = cv2.dnn.readNet(prototxtPath, weightsPath)

# cargamos el modelo de detector de máscara facial desde el disco
print("[INFO] loading face mask detector1k model...")
maskNet = load_model(args["model"])

# Inicializamos la transmisión de video y permitimos que el sensor de la cámara se caliente
print("[INFO] starting video stream...")
vs = VideoStream(src=0).start()
time.sleep(2.0)

#Iniciamos un bucle sobre los fotogramas de la transmisión de video
while True:
# Tomamos el marco de la secuencia de video encadenada y cambie su tamaño a un ancho máximo de 400
#píxeles
frame = vs.read()

```



```

frame = imutils.resize(frame, width=500)

#Detectamos rostros en el marco y determinamos si llevan puesto una
# mascarilla o no
(locs, preds) = detect_and_predict_mask(frame, faceNet, maskNet)

#Recorremos las ubicaciones de rostros detectados y sus correspondientes
# ubicaciones
for (box, pred) in zip(locs, preds):
#Descomprima el cuadro delimitador y las predicciones
(startX, startY, endX, endY) = box
(mask, withoutMask) = pred

#Determinar la etiqueta de clase y el color que usaremos para dibujar
# el cuadro delimitador y el texto
if mask > withoutMask:
label = "Con mascarilla"
color = (0, 255, 0)

else:
label = "Sin mascarilla"
color = (0, 0, 255)
exec(open("correo.py").read()) #Código que envía alerta si detecta sujeto sin mascarilla
exec(open("sinmask.py").read())#código de texto en lcd

#Mostramos la etiqueta y el rectángulo del cuadro delimitador en la salida
# marco
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)

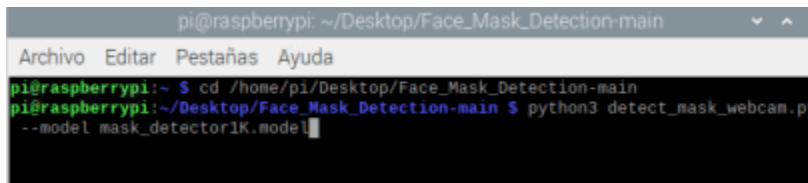
#Mostramos el cuadro de salida
cv2.imshow("Face Mask Detector", frame)
key = cv2.waitKey(1) & 0xFF

# Si se presionó la tecla `q`, sal del bucle
if key == ord("q"):
break

#Hacemos limpieza
cv2.destroyAllWindows()
vs.stop()

```

Para ejecutar nuestro programa final con nuestro modelo, tecleamos lo siguiente en la terminal



```
pi@raspberrypi: ~/Desktop/Face_Mask_Detection-main
Archivo  Editar  Pestañas  Ayuda
pi@raspberrypi:~ $ cd /home/pi/Desktop/Face_Mask_Detection-main
pi@raspberrypi:~/Desktop/Face_Mask_Detection-main $ python3 detect_mask_webcam.py
--model mask_detector1K.model
```

Ahora realizaremos el código del correo electrónico a enviar.

Abrimos un script de Python, y tecleamos el siguiente código:

```
# Importamos los paquetes necesarios
import smtplib,ssl # se puede usar para comunicarse con los servidores de correo para enviar correo
from email.mime.multipart import MIMEMultipart #Este módulo es parte de la Compat32API de correo
electrónico heredada ( ). Su funcionalidad se reemplaza parcialmente por la contentmanagerde la nueva
API
from email.mime.text import MIMEText #obtiene una estructura de objeto de mensaje al pasar un
archivo o algún texto a un analizador, que analiza el texto y devuelve el objeto de mensaje raíz
from email.mime.image import MIMEImage #obtiene una estructura de objeto de mensaje al pasar un
archivo o alguna imagen a un analizador, que analiza la imagen y devuelve el objeto de mensaje raíz

enviador = 'nethesimz@gmail.com'
receptor = 'perlaaceneth@gmail.com'

msgRoot = MIMEMultipart('related')
msgRoot['Subject'] = '¡Alerta!'
msgRoot['From'] = enviador
msgRoot['To'] = receptor
#msgRoot.preamble = 'Esto es el preambulo.'

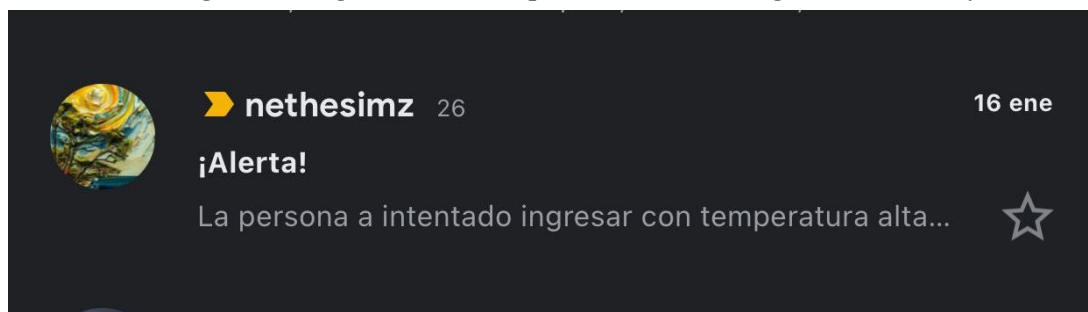
msgAlternative = MIMEMultipart('alternative')
msgRoot.attach(msgAlternative)

# Envío en texto plano
msgText = MIMEText('La persona a intentado ingresar sin cubrebocas. Mantener medidas pertinentes
para la desinfección. ')

msgAlternative.attach(msgText)
```

```
s = smtplib.SMTP('smtp.gmail.com', 587)
s.ehlo()
s.starttls()
s.login(user = enviador, password = 'Huelum12')
s.sendmail(enviador, receptor, msgRoot.as_string())
s.quit()
print (" Ya se ha enviado el correo :)")
```

Al ser ejecutado nuestro código, nos llegará al correo preestablecido el siguiente mensaje:



Ahora vamos a realizar el circuito correspondiente para el sensor de temperatura, para ello necesitaremos instalar una librería con los siguientes pasos:

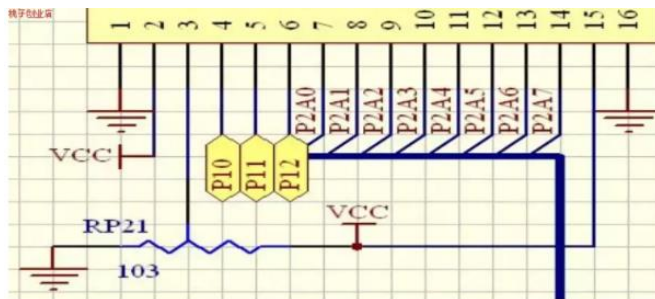
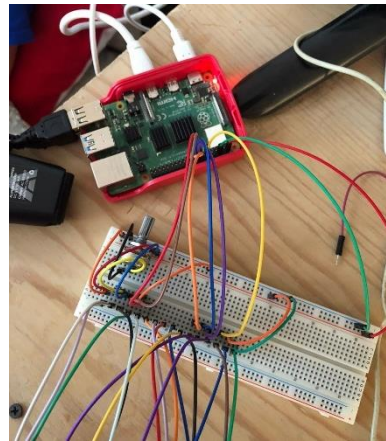
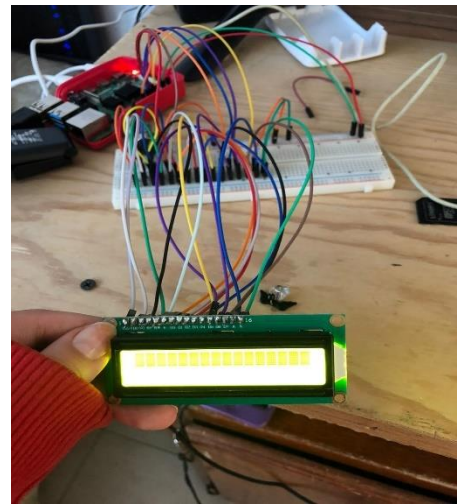
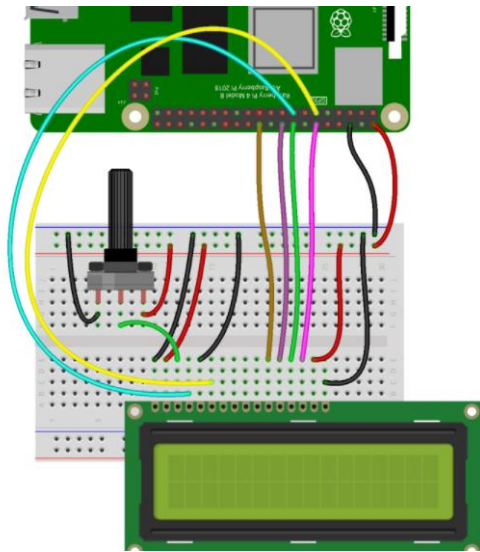
Haremos uso del paquete de Python3 *wlthermsensor*. Para instalarlo ejecuta el siguiente comando en la terminal:

sudo pip3 install wlthermsensor

```
pi@raspberrypi ~
Archivo  Editar  Pestañas  Ayuda
pi@raspberrypi:~$ sudo pip3 install wlthermsensor
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting wlthermsensor
  Downloading https://files.pythonhosted.org/packages/05/28/58bff5a46f00772e5f67c740777cb347c129d70673b1ad87b4ccdfa866a7/wlthermsensor-2.0.0-py2.py3-none-any.whl
Requirement already satisfied: click in /usr/lib/python3/dist-packages (from wlthermsensor) (7.0)
Installing collected packages: wlthermsensor
Successfully installed wlthermsensor-2.0.0
pi@raspberrypi:~$
```

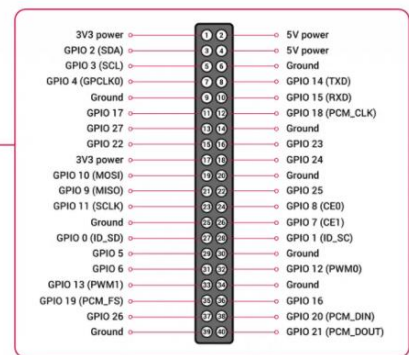
Para poder visualizar el texto sin mascarilla en nuestra pantalla LCD, proseguimos con:

Circuito



Pin interface specifications table
Numbers symbols pin pin that Numbers symbols that

Numbers	symbols	Pin that	Numbers	symbols	Pin that
1	VSS	To power	9	D2	data
2	VDD	power positive	10	D3	data
3	VL	Liquid crystal display bias	11	D4	data
4	RS	Data/command options	12	D5	data
5	R/W	Read/write choice	13	D6	data
6	E	By using the signal	14	D7	data
7	D0	data	15	BLA	Backlighting power positive
8	D1	data	16	BLA	Backlighting power negative



Para el circuito utilizamos los pines:

Raspberry pi 4	Pantalla LCD16x02
2 (5v de poder)	15 luz trasera positiva
6(Tierra)	16 luz trasera negativa
12(GPIO18 PCM_CLK) regula el ancho del pulso	14 D0 ~ D7 para ocho cables de datos de dos vías.
16(GPIO23) normales, se pueden programar mediante códigos	13 D0 ~ D7 para ocho cables de datos de dos vías.
18(GPIO24) normales, se pueden programar mediante códigos	12 D0 ~ D7 para ocho cables de datos de dos vías.
22(GPIO25) normales, se pueden programar mediante códigos	11 D0 ~ D7 para ocho cables de datos de dos vías.
11(gpio17) normales, se pueden programar mediante códigos	6 can end extremo E para hacer que, cuando E es conducido por alto nivel saltar a baja electricidad en tiempos normales, módulo LCD Orden Ejecutiva.
15(GPIO22) normales, se pueden programar mediante códigos	4 RS para elegir registros, high electric normalmente elige registros de datos, low electric normalmente elige el registro de instrucciones.

Funciones de los GPIO

→ Pueden ser configurados tanto como entrada como de salida.

→ Los pines GPIO también pueden ser activados y desactivados mediante código. Es decir, pueden ponerse a 1 (alto nivel de voltaje) o a 0 (bajo nivel de voltaje).

→ Por supuesto pueden leer datos binarios, como los unos y ceros, es decir, señal de voltaje o ausencia de ella.

→ Valores de salida de lectura y escritura.

→ Los valores de entrada pueden ser configurados en algunos casos como eventos para que generen algún tipo de acción sobre la placa o sistema. Algunos sistemas embebidos los usan como IRQ. Otro caso es configurar que cuando uno o varios pines estén activos por ciertos sensores realice alguna acción...

→ En cuanto al voltaje e intensidad, debes saber bien las capacidades máximas aceptables por la placa, en este caso la Raspberry Pi 4 o la 3. No debes pasarlos para no dañarla.

Código

```
# Importamos los paquetes necesarios
```

```
import time
```

```
import board
```

```
import digitalio
```

```
import adafruit_character_lcd.character_lcd as characterlcd
```



```
#Declaramos el tamaño de la lcd
```

```
lcd_columns = 16
```

```
lcd_rows = 2
```

```
# Configuración de pines de Raspberry Pi:
```

```
lcd_rs = digitalio.DigitalInOut(board.D22)
```

```
lcd_en = digitalio.DigitalInOut(board.D17)
```

```
lcd_d4 = digitalio.DigitalInOut(board.D25)
```

```
lcd_d5 = digitalio.DigitalInOut(board.D24)
```

```
lcd_d6 = digitalio.DigitalInOut(board.D23)
```

```
lcd_d7 = digitalio.DigitalInOut(board.D18)
```

```
# Inicializamos la clase lcd
```

```
lcd = characterlcd.Character_LCD_Mono(
```

```
    lcd_rs, lcd_en, lcd_d4, lcd_d5, lcd_d6, lcd_d7, lcd_columns, lcd_rows
```

```
)
```

```
#declaramos la presentación de un mensaje con movimiento de izquierda a derecha con una duración
```

```
#de dos segundos y al final limpiamos la pantalla
```

```
lcd.text_direction = lcd.LEFT_TO_RIGHT
```

```
lcd.cursor = True
```

```
lcd.blink = False
```

```
scroll_msg = "sin mascarilla"
```

```
lcd.message = scroll_msg
```

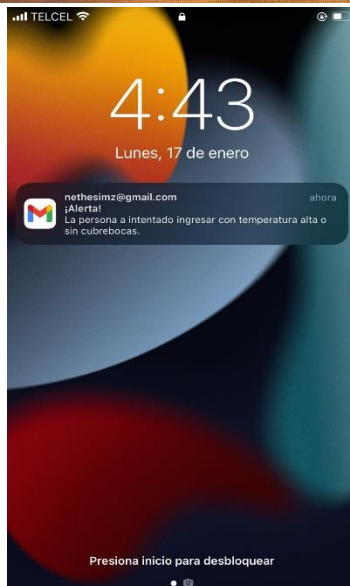
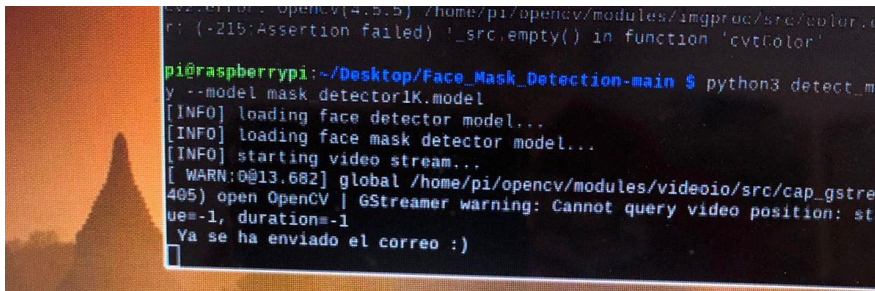
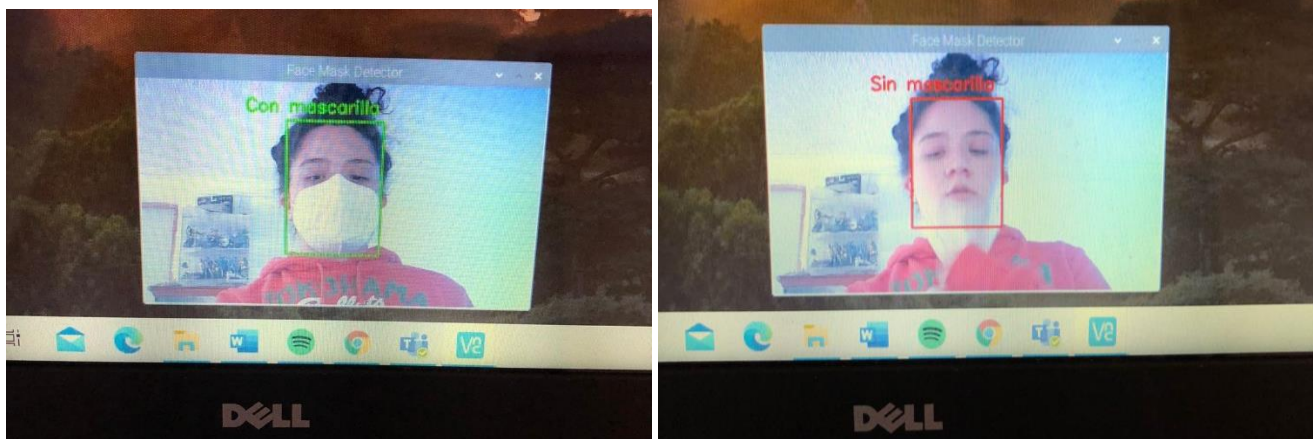
```
for i in range(len(scroll_msg)):
```

```
    time.sleep(0.2)
```

```
    lcd.move_left()
```

```
lcd.clear()
```

En conjunto, los tres programas se ven como:



REFERENCIAS:

- [1] <https://medium.com/metadatos/gu%C3%ADa-r%C3%A1pida-sobre-preprocesamiento-de-im%C3%A1genes-faciales-para-redes-neuronales-usando-opencv-en-c68599ca08dd>
- [2] <https://qengineering.eu/install-opencv-4.5-on-raspberry-pi-4.html>
- [3] http://kio4.com/raspberry/37_enviar_correo.htm
- [4] <https://www.raspberrypi.com/documentation/computers/configuration.html>
- [5] <https://rplcd.readthedocs.io/en/stable/usage.html>
- [6] <https://omes-va.com/deteccion-mascarilla-opencv-python/>
- [7] <https://spanish.alibaba.com/product-detail/lcd1602-lcd-1602-1602a-display-module-lcd-16x02-character-green-backlight-screen-3-3v-5v-60752190091.html>
- [8] https://www.hwlibre.com/gpio-raspberry-pi/?_gl=1%2A35x2ww%2A_ga%2AYW1wLUh0RFEwcUsxRV9fdDJTU3VzUHHGSEftU3pQbGoxeVRuWkNEUE1SZzV3ek1sSTlVEh3bElwT1A5aTdRWhXQjk