

Sistema identificador de cubrebocas en personas.

Jorge Irving Cristobal Pichardo

Facultad de Ingeniería. Universidad Autónoma de Guerrero

jorgeirvingcristobalpichardo@gmail.com

Abstract. Actualmente estamos viviendo una contingencia sanitaria mundial, a causa del SARS-CoV-2 [1], comúnmente llamado Coronavirus o Covid-19, que ha provocado que la vida cotidiana se haya paralizado en proporciones nunca antes vistas, debido al confinamiento que se debe tomar para detener el avance de este virus. El presente trabajo trata de un sistema que identifica si una persona situada frente a la cámara cuenta con un cubrebocas o no, ya que si se cumple con la premisa del cubrebocas, este enviará una alerta que hace efectivo el acceso a un espacio público (lugar de trabajo o escuela, etc.)

Palabras clave: Redes Neuronales; Python; OpenCV; TensorFlow; Covid-19.

1 Introducción

Cómo se ha mencionado anteriormente, el gobierno y los sistemas de salud tuvieron que tomar las medidas necesarias para evitar más contagios. Para esto se implementó, el uso obligatorio de cubrebocas en las calles [2] ya que los modelos matemáticos probabilísticos han hecho proyecciones de que sí se puede frenar la propagación como se muestra en la Figura 1 [3].

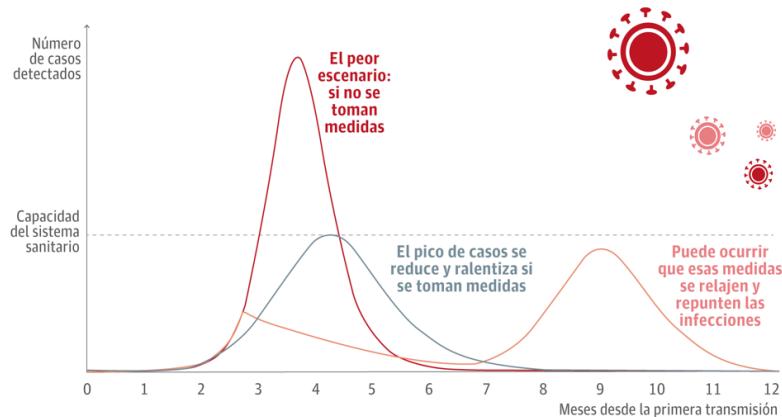


Fig. 1. Número de casos de Coronavirus dependiendo si se acatan o no las medidas.

Desafortunadamente no todas las personas alrededor del mundo pueden estar en confinamiento, ya que se necesitan estar activos los servicios con los que contamos en nuestros hogares (agua, electricidad, comunicaciones, etc.) por otro lado, están los que necesitan salir cada cierto tiempo por provisiones y por último, las personas que sus trabajos son oficios, se ven obligados a salir, por necesidad.

Es por estos motivos es que se ha propuesto el presente proyecto, para automatizar la revisión cubrebocas para tener acceso a lugares públicos. Retomado del proyecto del Doctor Adrián Rosebrock [4] sólo se hizo un afinamiento más al sistema y un acondicionamiento para poder conectarlo al microcontrolador Arduino [5] para la recepción y envío de datos a través de este.

Ya que el uso de cubrebocas es obligatorio, el sistema sólo necesita detectarlo en el rostro de una persona para poder mandar una alerta de que sí se cumple la condición y dar acceso a un espacio público (supermercado, lugares de trabajo, etc.) y si no se cumple con la premisa simplemente el sistema no manda ninguna señal. También esto ayuda a evitar conflictos con personas que quieran entrar a estos espacios por la fuerza y exponer a los que sí están cumpliendo con las reglas.

2 Materiales y métodos

Como se mencionó anteriormente este proyecto fue construido a apartir del programa del Doctor Adrián Rosebrock, que fue creado a partir de la necesidad de contar con un sistema que determine si estamos usando cubrebocas o no.

A continuación se describirá la división del proyecto y sus diferentes fases, como se muestra en la Figura 2.

```

└── dataset
    ├── with_mask [690 entries]
    └── without_mask [686 entries]
    ├── examples
    │   ├── example_01.png
    │   ├── example_02.png
    │   └── example_03.png
    ├── face_detector
    │   ├── deploy.prototxt
    │   └── res10_300x300_ssd_iter_140000.caffemodel
    ├── detect_mask_image.py
    ├── detect_mask_video.py
    ├── mask_detector.model
    └── plot.png
    └── train_mask_detector.py

```

5 directories, 10 files

Fig 2. Directorio con el que cuenta el proyecto

dataset/ El conjunto de datos de detección de cubrebocas con una subdivisión (como se muestra en la Figura 3) de una carpeta con 690 imágenes con personas con cubrebocas “**with_mask**” y otra carpeta con 686 imágenes con personas sin

cubrebocas “**without_mask**” (cabe mencionar que en ninguna de las dos carpetas se repite una sola imagen de la misma persona con un cubreboca y en otra sin cubreboca, todas las imágenes son diferentes, esto se hizo con el fin de que la salida dé resultados más precisos) para su previo entrenamiento en el script *train_mask_detector.py* ” .

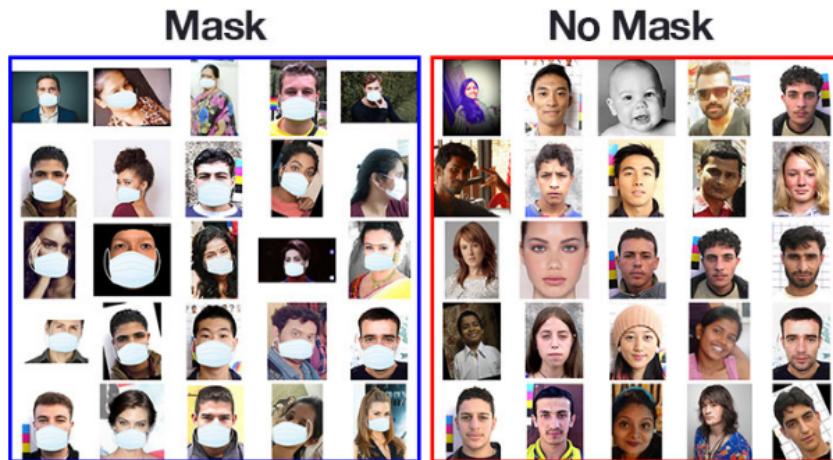


Fig 3. Imágenes de personas con y sin cubrebocas

examples/ Se proporciona para que pueda probar el detector de cubrebocas de una imagen fija.

Tres scripts de Python:

train_mask_detector.py: Acepta nuestro conjunto de datos de entrada y ajusta MobileNetV2 para crear nuestro **mask_detector.model**.

Una historia de **entrenamientoplot.png** que contiene curvas de precisión / pérdida también se produce.

detect_mask_image.py: Realiza la detección de la máscara facial en imágenes estáticas.

detect_mask_video.py: Usando su cámara web, este script aplica la detección de máscara facial a cada cuadro en la transmisión.

En esta ocasión sólo se enfocará en describir en forma detallada del script *detect_mask_video.py* ya que este es el que tendrá comunicación con el microcontrolador *Arduino* ya que su fin es aplicarlo en la vida real, para resolver un problema real.

A continuación la descripción de las *librerías* que se usaron en *detect_mask_video.py*:

TensorFlow: es una biblioteca de código abierto para aprendizaje automático a través de un rango de tareas, y desarrollado por Google para satisfacer sus necesidades de sistemas capaces de construir y entrenar redes neuronales para detectar y descifrar patrones y correlaciones, análogos al aprendizaje y razonamiento usados por los humanos.

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. Sus fuertes se centran en ser amigable para el usuario, modular y extensible.

NumPy: es una extensión de Python, que le agrega mayor soporte para vectores y matrices, constituyendo una biblioteca de funciones matemáticas de alto nivel para operar con esos vectores o matrices.

Argparse: El módulo argparse incluye herramientas para construir procesadores de argumentos y opciones de línea de comando.

Imutils: Una serie de funciones convenientes para hacer funciones básicas de procesamiento de imágenes, como traducción, rotación, cambio de tamaño, esqueletización, visualización de imágenes Matplotlib, clasificación de contornos, detección de bordes y mucho más fácil con OpenCV y Python 2.7 y Python 3.

Time: El módulo time de la biblioteca estándar de Python proporciona un conjunto de funciones para trabajar con fechas y/o horas.

OpenCV: es una biblioteca libre de visión artificial originalmente desarrollada por Intel. OpenCV significa Open Computer Vision (Visión Artificial Abierta).

OS: El módulo os nos permite acceder a funcionalidades dependientes del Sistema Operativo. Sobre todo, aquellas que nos refieren información sobre el entorno del mismo y nos permiten manipular la estructura de directorios.

Serial: para poder comunicarnos con Arduino necesitamos la librería PySerial.

Por otro lado, también se tomará en cuenta el material que se utilizó para la alerta que el sistema enviará al Arduino desde nuestro proyecto previamente construido en Python, cabe destacar que la mención y participación del microcontrolador tomará más importancia en la sección de **Propuesta**:

Material de Arduino:

- 1 Placa de Arduino Uno
- 1 cable USB para Arduino
- Cables o Jumpers para las conexiones
- 1 protoboard
- Leds y resistencias (330, 220 o de 100 Ohms)
- 1 matriz de leds 8x8 para Arduino

2.1 Estructura del detector de cubrebocas

A continuación se describe como fue la estructuración que tomó el proyecto y su posterior obtención de resultados como lo muestra las Figura 4.



Figura 4: División en dos fases; entrenamiento y aplicación

2.2 Fase # 1: Entrenamiento del detector de cubrebocas

En primera instancia se sitúa en el archivo del proyecto ***train_mask_detector.py*** que es la columna vertebral en el que se sostendrá este, ya en el archivo se explicará lo que hace cada línea de código.

En este archivo se ejecutará desde la Terminal (En sistemas Mac Os X y/o Linux, y en Windows Os desde CMD, también conocido como Símbolo del Sistema) con el siguiente comando: **python train_mask_detector.py --dataset dataset** (*Si tienes Mac Os y Linux, es muy importante que le agregres el número '3' al final de la palabra 'python', si no el programa no se ejecutará*) y mostrará la siguiente información:

```
[INFO] cargando imágenes ...
[INFO] compilando modelo ...
[INFO] jefe de entrenamiento ...
Entrene para 34 pasos, valide en 276 muestras
Epoch 1 / 20
34 / 34 [=====] - 30s 885ms / paso - pérdida: 0. 6431 -
Precisión: 0 . 6676 - val_loss: 0. 3696 - val_accuracy: 0. 8242
Epoch 2 / 20
34 / 34 [=====] - 29s 853ms / paso - pérdida: 0. 3507 -
Precisión: 0 . 8567 - val_loss: 0. 1964 - val_accuracy: 0. 9375
Epoch 3 / 20
34 / 34 [=====] - 27s 800ms / paso - pérdida: 0. 2792 -
Precisión: 0 . 8820 - val_loss: 0. 1383 - val_accuracy: 0. 9531
Epoch 4 / 20
34 / 34 [=====] - 28s 814ms / paso - pérdida: 0. 2196 -
Precisión: 0 . 9148 - val_loss: 0. 1306 - val_accuracy: 0. 9492
Epoch 5 / 20
34 / 34 [=====] - 27s 792ms / paso - pérdida: 0. 2006 -
Precisión: 0 . 9213 - val_loss: 0. 0863 - val_accuracy: 0. 9688
...
Epoch 16 / 20
34 / 34 [=====] - 27s 801ms / paso - pérdida: 0. 0767 -
Precisión: 0 . 9766 - val_loss: 0. 0291 - val_accuracy: 0. 9922
Epoch 17 / 20
34 / 34 [=====] - 27s 795ms / paso - pérdida: 0. 1042 -
Precisión: 0 . 9616 - val_loss: 0. 0243 - val_accuracy: 1. 0000
Epoch 18 / 20
34 / 34 [=====] - 27s 796ms / paso - pérdida: 0. 0804 -
Precisión: 0 . 9672 - val_loss: 0. 0244 - val_accuracy: 0. 9961
Epoch 19 / 20
34 / 34 [=====] - 27s 793ms / paso - pérdida: 0. 0836 -
Precisión: 0 . 9710 - val_loss: 0. 0440 - val_accuracy: 0. 9883
Epoch 20 / 20
34 / 34 [=====] - 28s 838ms / paso - pérdida: 0. 0717 -
Precisión: 0 . 9710 - val_loss: 0. 0270 - val_accuracy: 0. 9922
[INFO] evaluando la red ...
soporte de precisión de puntuación f1
with_mask 0. 99 1. 00 0. 99 138
```

```

sin cubrebocas 1.00 0.99 0.99 138
precisión 0.99 276
promedio macro 0.99 0.99 0.99 276
promedio ponderado 0.99 0.99 0.99 276

```

Y posteriormente se desplegará una grafica previamente programada del entrenamiento, como se muestra en la Figura 5. Las curvas de precisión / pérdida de entrenamiento del detector de cubrebocas demuestran una alta precisión y pequeños signos de sobreajuste en los datos.

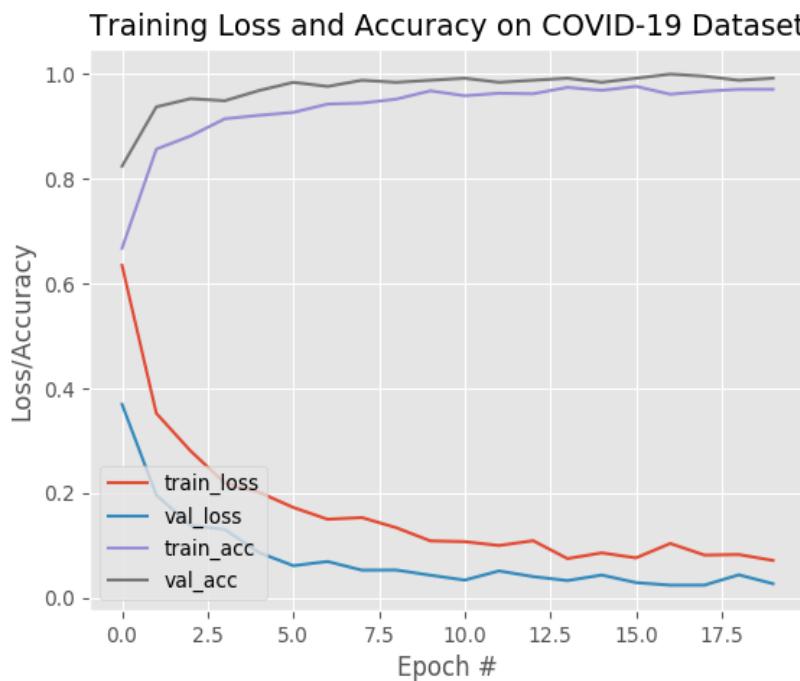


Fig. 5: Gráfica de los datos entrenados.

2.3 Fase # 2 Aplicar el detector de cubrebocas

Anteriormente se construyeron las redes neuronales entrenadas para clasificar a las personas que tienen en el rostro un cubrebocas o de la misma manera, si no lo tienen. Para comprobar que todo está funcionando de la forma correcta ejecutamos el siguiente archivo llamado ***detect_mask_image.py*** y lo hacemos con el siguiente comando desde la Terminal del sistema: ***python detect_mask_image.py --image examples/example_01.png*** cambiamos los nombres de las imágenes en la parte del comando para ver los ejemplos que se encuentran en la carpeta **/examples** y los resultados se pueden visualizar como la Figura 6.

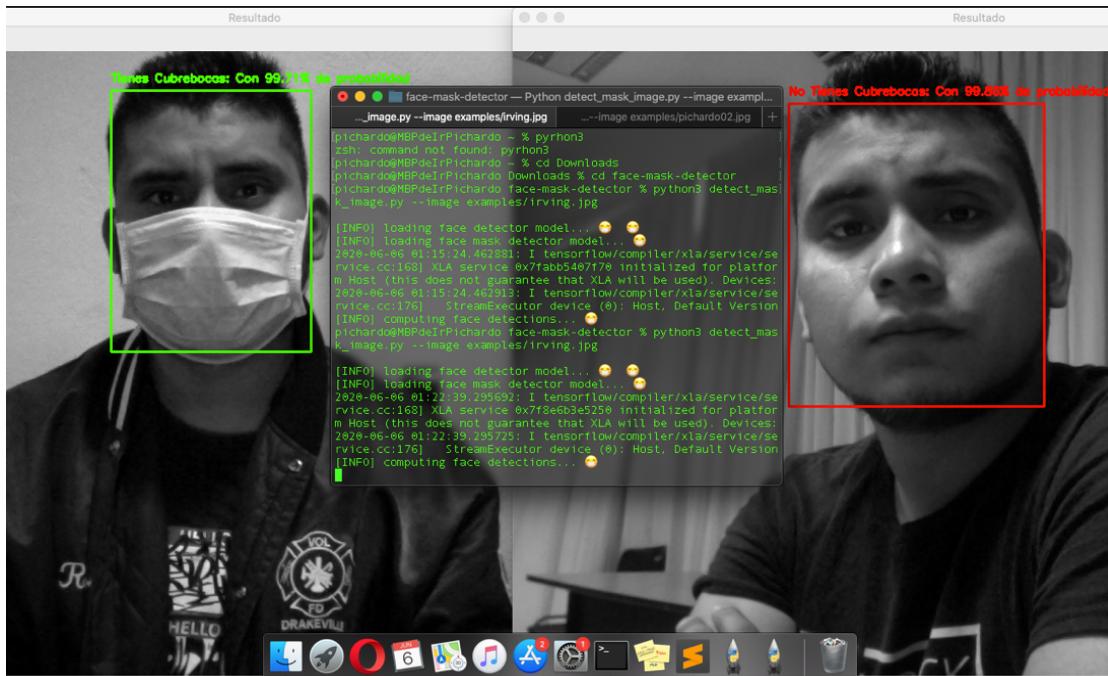


Fig. 6: Muestra el resultado de ejecutar el archivo ***detect_mask_image.py***

3 Propuesta

En esta parte se centrará en dar la propuesta del proyecto y su funcionamiento, a continuación en la Figura 7 se mostrará el proceso que se llevará a cabo.



Fig. 7. Proceso de implementación del programa.

3.1 Representación del programa.

De acuerdo con la figura 3, nos centraremos en la parte 3 del bloque de procesos a realizar.

Pero antes se realiza el diagrama de bloques para entender más que se pretende realizar en este proyecto.

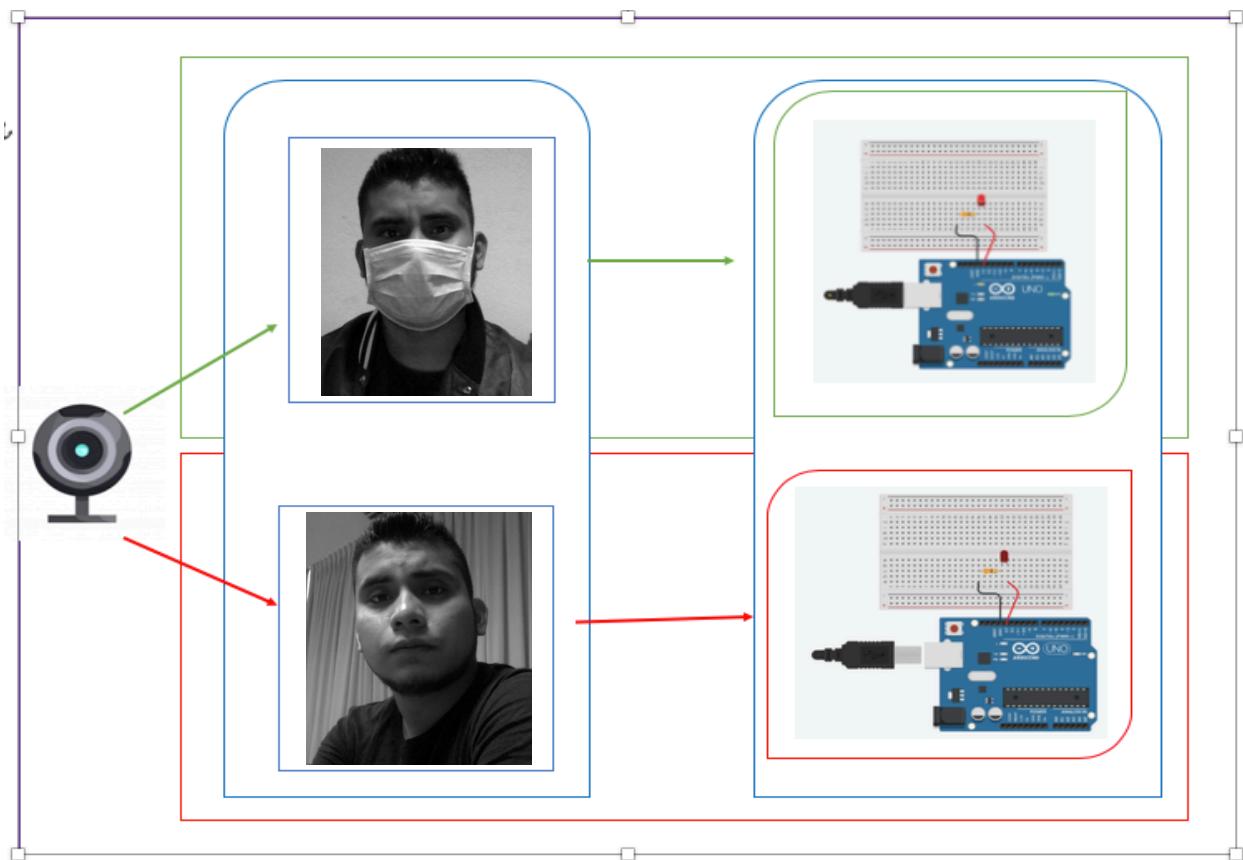


Fig. 8: Representación de cómo funcionaría el programa.

Lo que se pretende conseguir es que el sistema detecte a través de la cámara si una persona tiene o no un cubrebocas, en caso de que sea positivo, manda una alerta al microcontrolador para que este accione una alarma (se puede interpretar como el abrir de una puerta, etc.) y de lo contrario el sistema no hace nada, ya que se supone que sí una persona quiere entrar a un espacio público, debe llevar puesto el cubrebocas obligatoriamente.

3.2 Modificación del código *detect_mask_video.py*

```

125      # determinar la etiqueta de clase y el color que usaremos para dibujar
126      # el cuadro delimitador y el texto
127      label = "Tienes cubreboca" if mask > withoutMask else "No Tienes cubreboca"
128      color = (0, 255, 0) if label == "Tienes cubreboca" else (0, 0, 255)
129
130      # incluye la probabilidad en la etiqueta
131      label = "{}: Con {:.2f}%".format(label, max(mask, withoutMask) * 100)+" de probabilidad"
132
133
134      import serial
135      arduino=serial.Serial(port='/dev/cu.usbmodem14201', baudrate=9600)
136      if mask==1:
137          arduino.write(mask)
138      else:
139          withoutMask==0
140          arduino.write(withoutMask)
141
142
143      # muestra la etiqueta y el rectángulo del cuadro delimitador en la salida
144      # marco
145      cv2.putText(frame, label, (startX, startY - 10),
146                  cv2.FONT_HERSHEY_SIMPLEX, 0.45, color, 2)
147      cv2.rectangle(frame, (startX, startY), (endX, endY), color, 2)
148
149      # muestra el marco de salida
150      cv2.imshow("Detector de Cubrebocas *2020*",frame)
151      key = cv2.waitKey(1) & 0xFF
152

```

Fig. 9: Modificación de líneas de código

Como especifica el título de este apartado, aquí se van hacer unas modificaciones al programa y en la Figura 9 señala en que líneas de código será, y éstas son las que comprenden de la línea 135 a la 141.

1.- Importamos aquí mismo la librería de PySerial

2.- Declaramos una variable llamada arduino y le pasamos los parametros del nombre del puerto y configuración de los datos a 9600 bps. [Nota importante: lo que muestra la linea 136 es exclusivo para sistemas Mac Os X, ya que en Windows sólo se colocaría: `arduino = serial.Serial('COM3', 9600)` por eso tuve unas complicaciones al conectar los puertos a Python porque desconocía las configuraciones típicas para el sistema].

3.- Se programa un if/else, por si se cumple con la premisa del cubre bocas se escriba sobre el Arduino y si no pues el sistema se queda estático o se apaga en caso de que la persona se quitó el cubrebocas.

4 Results and discussion

Después de la sección anterior el programa está listo para ejecutarse, el Arduino ya tiene el código que va a ejecutar y entonces se ejecuta el archivo detect_mask_video.py y muestra la imagen de la Figura 10.

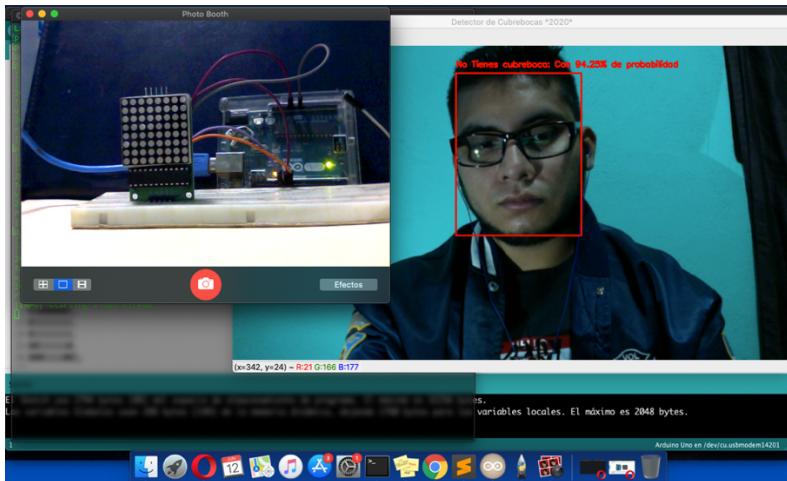


Fig. 10: Se muestra la ejecución y cómo el programa no responde a una persona sin cubrebocas.

Consecuentemente la persona debe tener cubrebocas, así como se muestra en la Figura 11.



Fig. 11: El microcontrolador responde a la premisa que cumple del cubrebocas entonces la matriz8x8 manda un mensaje o alerta.

¡Listo puedes ingresar de forma segura al espacio público!

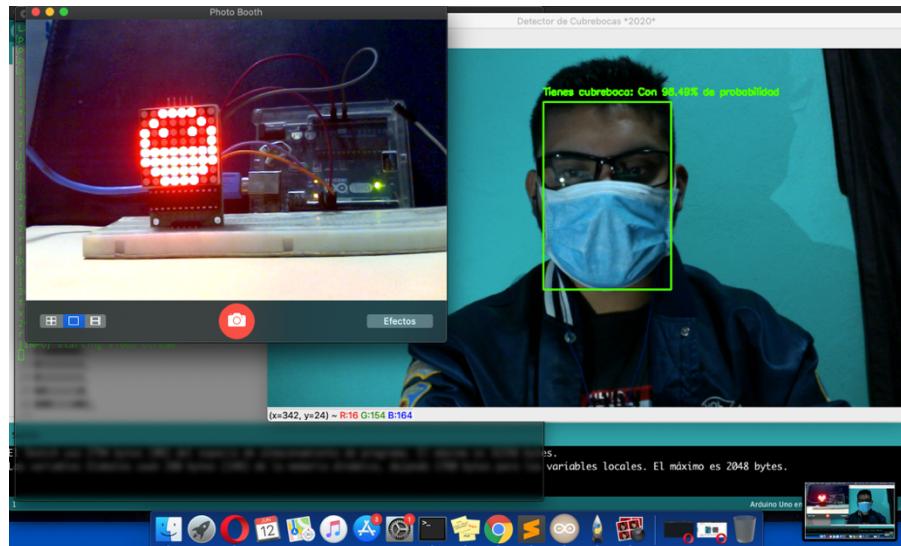


Fig. 12: La matriz manda un emoji con cubrebocas después de 5 segundos transcurridos.

5 Conclusiones

Para cerrar este reporte, se llegó a la conclusión de que el proyecto es 100% viable para la implementación en la vida real, ya que los tiempos actuales así lo requieren. El programa *detect_mask_video.py* sí manda la señal en tiempo real para poner en movimiento un accionador (puede ser un rele, servomotor, alertas, etc.) No pude implementarlo al 100% en una puerta, ya que necesito servomotores o reles para llevarlo a cabo y lamentablemente, no los tengo disponibles, más adelante espero implementarlo.

Referencias

1. <https://gacetamedica.com/investigacion/descifrando-los-origenes-del-sars-cov-2/> [Consultada el día 10 de junio de 2020]
2. <https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/advice-for-public/when-and-how-to-use-masks> [Consultada el día 7 de junio de 2020]
3. <https://www.elnortedecastilla.es/sociedad/salud/proyecciones-llevan-cierre-20200313215232-ntrc.html> [Consultada el día 5 de junio de 2020]
4. <https://www.pyimagesearch.com/2020/05/04/covid-19-face-mask-detector-with-opencv-keras-tensorflow-and-deep-learning/> [Consultada el día 2 de junio de 2020]
5. <https://arduino.cl/que-es-arduino/> [Consultada el día 8 de junio de 2020]