UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA LA DETECCIÓN DE FUMADORES EN ÁREAS RESTRINGIDAS MEDIANTE INTELIGENCIA ARTIFICIAL

Trabajo de titulación previo a la obtención del
Título de Ingeniero en Electrónica

AUTORES:

SAMUEL JOSUÉ MOLINARI CALDERÓN

MEDARDO ADRIÁN VELIZ CÓRDOVA

TUTOR:

ING. TEDDY NEGRETE PEÑA

GUAYAQUIL – ECUADOR

2024

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Nosotros, Molinari Calderón Samuel Josué con cédula de identidad N°.0957411143 y Veliz Córdova Medardo Adrián con cédula de identidad N°.0932241763, manifestamos que:

Somos los autores y responsables del presente trabajo; y, autorizamos a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, 1 de febrero del año 2024.

Atentamente,

Samuel Molinari Calderón

0957411143

Adrián Veliz Córdova

0932241763

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Nosotros, Molinari Calderón Samuel Josué con documento de identificación N°.0957411143 y Veliz Córdova Medardo Adrián con documento de identificación N°.0932241763, expresamos nuestra voluntad y por medio del presente documento cedemos a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que somos autores del Proyecto Técnico: "DISEÑO E IMPLEMENTACION DE UN PROTOTIPO PARA LA DETECCION DE FUMADORES EN AREAS RESTRINGIDAS MEDIANTE INTELIGENCIA ARTIFICIAL, el cual ha sido desarrollado para optar por el título de: Ingeniero en Electrónica, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribimos este documento en el momento que hacemos la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 1 de febrero del año 2024.

Atentamente,

Samuel Molinari Calderón

CI: 0957411143

Adrián Veliz Córdova

CI: 0932241763

CERTIFICADO DE DIRECCIÓN DE TRABAJO DE TITULACIÓN

Yo, Geovanny García con documento de identificación N°0912419611, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: DISEÑO E IMPLEMENTACION DE UN PROTOTIPO PARA LA DETECCION DE FUMADORES EN AREAS RESTRINGIDAS MEDIANTE INTELIGENCIA ARTIFICIAL" realizado por Molinari Calderón Samuel Josué con documento de identificación N°.0957411143 y Veliz Córdova Medardo Adrián con documento de identificación N°.0932241763, obteniendo como resultado final el trabajo de titulación bajo la opción de Proyecto Técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 1 de febrero del año 2024.

Ing. Teddy Negrete Peña
0912419611

DEDICATORIA

Agradezco profundamente a Dios por otorgarme el invaluable regalo de la vida, la salud y por colmarme con sus innumerables bendiciones, las cuales han sido mi sostén inquebrantable durante esta etapa crucial de mi trayectoria.

Expreso mi más sincero reconocimiento a mi madre, cuya dedicación y apoyo incansable han sido pilares fundamentales que me han impulsado a perseverar en mi camino académico. A su vez, agradezco a mi padre, a mi hermano y a mi entrañable compañera, Charlotte, por su constante respaldo y aliento en mi búsqueda de conocimiento y crecimiento personal.

Es en este momento de reflexión y gratitud donde reconozco el inmenso valor del amor y el apoyo brindado por mi familia, quienes han sido mi inspiración y motor para alcanzar mis metas y aspiraciones. Con profunda gratitud, celebro su incondicional apoyo y dedicación, que han sido el cimiento de mi desarrollo profesional y personal.

Adrián veliz Córdova

Agradezco sinceramente a quienes me acompañaron en mi viaje universitario y a mis padres por su apoyo inquebrantable. Su aliento fue esencial para mi éxito académico. Mi más profundo agradecimiento por ser mi fuente de inspiración y motivación.

Samuel Molinari Calderón

AGRADECIMIENTO

Agradezco a Dios por permitirme estar con vida, salud y llenándome de sus bendiciones para terminar esta etapa de mi vida.

Agradezco a mi mama que fue el pilar importante para seguir estudiando y también a mi papa, mi hermano y mi gatita Charlotte que fueron de gran apoyo para mi

Adrián veliz Córdova

Quisiera expresar mi máximo agradecimiento para las personas que estuvieron conmigo en este tiempo que estuve en la universidad, también expresar mi gratitud a hacia mis padres por siempre darme un aliento de ánimo para no rendirme y poder llegar a la meta

Samuel Molinari Calderón

RESUMEN

AÑO	ALUMNOS	TUTOR DEL PROYECTO	TEMA DEL PROYECTO
2024	Veliz Córdova Medardo Adrián. Molinari Calderón Samuel Josué	Ing. Teddy Negrete peña	"DISEÑO E IMPLEMENTACIÓN DE UN PROTOTIPO PARA LA DETECCIÓN DE FUMADORES EN ÁREAS RESTRINGIDAS MEDIANTE INTELIGENCIA ARTIFICIAL".

La presente tesis aborda el desarrollo de un prototipo destinado a la detección y notificación de situaciones de fumado en áreas de restricción. Este proyecto se fundamenta en la necesidad de implementar soluciones tecnológicas que promuevan el cumplimiento de normativas y políticas de seguridad en entornos públicos y privados.

Para llevar a cabo esta iniciativa, se ha optado por la utilización de un sistema embebido con capacidades de procesamiento de imagen. Se seleccionó el Raspberry Pi 4 modelo B como plataforma principal debido a su reconocida potencia y versatilidad en el campo de la informática embebida. El sistema operativo oficial de Raspberry Pi, conocido como Raspberry Pi OS o Raspbian, se empleó para el desarrollo del software, utilizando el IDE Geany para la programación en Python. El software desarrollado permite la detección de imágenes a través de una cámara web, ofreciendo visualización en tiempo real de las detecciones en un monitor. En caso de identificar a un individuo fumando, el sistema emite una notificación sonora a través del dispositivo Raspberry Pi, alertando al usuario sobre la prohibición de fumar en el área. Además, se captura una imagen de la detección en tiempo real, la cual se almacena localmente y se envía de forma automática a una dirección de correo electrónico predefinida.

Con el objetivo de facilitar la configuración y personalización del sistema, se ha integrado una interfaz gráfica que permite ajustar las opciones y parámetros de detección según las necesidades del usuario. Esta investigación busca contribuir al desarrollo de soluciones tecnológicas innovadoras que promuevan entornos más seguros y saludables, al mismo tiempo que se fomenta la aplicación de tecnología embebida en el ámbito de la seguridad y la prevención de riesgos.

ABSTRACT

YEAR	STUDENTS	PROJECT TUTOR	PROJECT THEME
2020	Veliz Córdova Medardo Adrián. Molinari Calderón Samuel Josué	Ing. Teddy Negrete Peña	"DESIGN AND IMPLEMENTATION OF A PROTOTYPE FOR THE DETECTION OF SMOKERS IN RESTRICTED AREAS THROUGH ARTIFICIAL INTELLIGENCE"

This thesis addresses the development of a prototype aimed at detecting and notifying smoking situations in restricted areas. This project is based on the need to implement technological solutions that promote compliance with regulations and security policies in public and private environments.

To carry out this initiative, the use of an embedded system with image processing capabilities has been chosen. The Raspberry Pi 4 Model B was selected as the main platform due to its recognized power and versatility in the field of embedded computing. The official Raspberry Pi operating system, known as Raspberry Pi OS or Raspbian, was used for software development, using the Geany IDE for Python programming.

The developed software allows image detection through a webcam, providing real-time visualization of detections on a monitor. In case of identifying an individual smoking, the system emits an audible notification through the Raspberry Pi device, alerting the user about the smoking ban in the area. Additionally, an image of the real-time detection is captured, which is stored locally and automatically sent to a predefined email address. In order to facilitate system configuration and customization, a graphical interface has been integrated that allows adjusting detection options and parameters according to user needs.

This research aims to contribute to the development of innovative technological solutions that promote safer and healthier environments, while also fostering the application of embedded technology in the field of security and risk prevention.

ÍNDICE

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN 2
CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA....3
CERTIFICADO DE DIRECCIÓN DE TRABAJO DE TITULACIÓN....4
DEDICATORIA....5
AGRADECIMIENTO....6
RESUMEN....7
ABSTRACT....8
INTRODUCCIÓN....15
1. PROBLEMA....15
1.1. ANTECEDENTES....15
1.2. IMPORTANCIA Y ALCANCE....16
1.3. DELIMITACIÓN....17
1.3.1. TEMPORAL....17
1.3.2. ESPACIAL....17
1.3.3. ACADÉMICA....17
1.4. INNOVACIÓN....17
2. OBJETIVOS....18
2.1. OBJETIVO GENERAL....18
2.2. OBJETIVOS ESPECÍFICOS....18
3. FUNDAMENTOS TEÓRICOS....18
3.1. Inteligencia artificial....18
3.2. Machine Learning....19
3.1. Reconocimiento de imágenes....19
3.2. CNN (redes neuronales convolucionales)....19
3.3. Detección de objetos....20
3.4. Tensorflow....20
3.5. Tensorflow lite....21
3.5.1. Efficient Det: detección de objetos escalable y eficiente....21
3.6. MediaPipe....22
3.6.1. Detección de palma....23
3.6.2. Detección de cara....24

3.6.3. Blazeface: detección neuronal de rostros..... 24
3.7. Cámara HD.....25
3.8. Parlantes..... 26
3.9. Sistemas embebidos..... 26
3.10. Raspberry PI 4 modelo B..... 26
3.10.1. Case con ventilador..... 27
3.10.2. Fuente de poder..... 28
3.10.3. Unidad de estado sólido (SSD)..... 28
3.10.4. Cable HDMI a microHDMI..... 29
3.1. Google Colab..... 29
3.2. Cuda/Cudnn.....30
3.3. Formato de anotación Pascal VOC (visual object classes)..... 30
3.4. Roboflow.....30
3.5. Ide Geany..... 31
3.6. Python..... 31
3.6.1. Pycharm.....31
3.7. OpenCV.....32
3.7.1. RGB.....32
3.8. SMTP LIVE..... 32
3.8.1. MimeMultipart..... 33
3.9. PYSIMPLEGUI..... 33
4. MARCO METODOLÓGICO..... 34
4.1. Gestión de dependencias..... 34
4.2. Captura de video en tiempo real mediante la librería OpenCV..... 35
4.3. Detección de manos y cara.....37
4.3.1. Instalación de Mediapipe en la Raspberry Pi..... 37
4.3.2. Código con Mediapipe para detección de manos y cara..... 38
4.4. Modelo de aprendizaje automático.....44
4.4.1. Instalación de TensorFlow en la Raspberry Pi.....46
4.4.2. Data Set..... 53
4.4.3. Entrenar y desplegar modelo customizado para la detección de fumadores..... 57
4.5. Lógica para realizar la detección de fumadores..... 61
4.6. Funciones del prototipo al detectar un fumador.....63
4.6.1. Audio de salida.....63

4.6.2. Almacenar el fotograma de la detección dentro de una carpeta local.....65
4.6.3. Enviar el fotograma de la detección por correo electrónico.....65
4.7. Interfaz gráfica del prototipo.....66
5. RESULTADOS.....70
5.1. Resultados entrenamiento.....70
5.2. Resultados tensorflow.....73
5.3. Resultados Mediapipe.....74
5.4. Resultados funciones de salida.....75
5.5. Resultados código final interfaz grafico.....76
5.6. Análisis de resultados.....77
CONCLUSIONES.....83
RECOMENDACIONES.....84
REFERENCIAS.....85
ANEXOS.....88
CRONOGRAMA.....88
PRESUPUESTO.....89
Código OpenCV.....89
Código Mediapipe.....90
Código TensorFlow Lite.....93
Código Final con GUI.....94
Generar una matriz de confusión con valores escalares.....108

ÍNDICE DE IMÁGENES

Figura 1..... 17

Figura 2..... 23

Figura 3..... 24

Figura 4..... 25

Figura 5..... 27

Figura 6..... 27

Figura 7..... 28

Figura 8..... 29

Figura 9..... 34

Figura 10..... 35

Figura 11..... 35

Figura 12..... 36

Figura 14..... 37

Figura 15..... 37

Figura 16..... 38

Figura 17..... 38

Figura 18..... 39

Figura 19..... 40

Figura 20..... 40

Figura 21..... 41

Figura 22..... 42

Figura 23..... 42

Figura 24..... 43

Figura 25..... 44

Figura 26..... 45
Figura 27..... 46
Figura 28..... 47
Figura 29..... 48
Figura 30..... 48
Figura 31..... 49
Figura 32..... 50
Figura 33..... 50
Figura 34..... 52
Figura 35..... 52
Figure 36..... 53
Figura 37..... 53
Figura 38..... 54
Figura 39..... 54
Figura 40..... 55
Figura 41..... 55
Figura 42..... 56
Figura 43..... 56
Figura 44..... 56
Figura 45..... 57
Figura 46..... 58
Figura 47..... 58
Figura 48..... 59
Figura 49..... 60
Figura 50..... 60

Figura 51..... 61
Figura 52..... 62
Figura 53..... 63
Figura 54..... 63
Figura 55..... 64
Figura 56..... 64
Figura 57..... 64
Figura 58..... 65
Figura 59..... 66
Figura 60..... 67
Figura 61..... 67
Figura 62..... 68
Figura 63..... 68
Figura 64..... 69
Figura 65..... 69
Figura 66..... 70
Figura 67..... 73
Figura 68..... 73
Figura 69..... 74
Figura 70..... 75
Figura 71..... 75
Figura 72..... 76
Figura 73..... 77
Figura 74..... 80
Figura 75..... 81

INTRODUCCIÓN

1. PROBLEMA

1.1. ANTECEDENTES

En el Ecuador el Art. 21 de la ley orgánica para la regulación y control del tabaco "Espacios libres de humo" dice que: Se declara la prohibición de fumar y mantener encendidos productos de tabaco en espacios cerrados de instituciones públicas, lugares de trabajo abiertos al público, dependencias de salud y educación, medios de transporte público y en espacios cerrados destinados a actividades deportivas (UTE, 2022)

En muchos entornos públicos y privados, las políticas de no fumar son fundamentales para promover la salud y la seguridad de las personas. (World Health Organization: WHO, 2023)

Sin embargo, en la mayoría de los casos carece de sistemas eficientes y efectivos para hacer cumplir estas políticas Durante la mayor parte del tiempo, los sistemas de vigilancia han sido pasivos y de alcance limitado. En este contexto, se han utilizado cámaras fijas y otros dispositivos de detección como alarmas de seguridad. Estos sistemas son capaces de rastrear personas o detectar algún tipo de evento, sin embargo, no han sido diseñados para detectar comportamientos más complejos (Tsakanikas & Dagiuklas, 2018)

1.2. IMPORTANCIA Y ALCANCE

La importancia de este proyecto radica en la prevención de fumadores en áreas restringidas para un mayor control ya que no existe un nivel seguro de exposición al humo de tabaco ajeno, y provoca enfermedades graves, incluyendo problemas cardíacos, cáncer de pulmón y causa la muerte prematura de alrededor de 1,3 millones de personas anualmente (World Health Organization: WHO, 2023)

También puede dar lugar a situaciones peligrosas, como incendios y daños materiales. Los cigarrillos mal apagados son la principal causa de incendios forestales y residenciales en todo el mundo. En la ciudad de Buenos Aires, los cigarrillos mal apagados provocan el 21% de los incendios. (Tabaco y medio ambiente, 2021).

Por lo tanto, se justifica la necesidad de desarrollar un sistema de detección de personas fumando en áreas de no fumar que utilice tecnologías de visión por computadora y aprendizaje automático.

1.3. DELIMITACIÓN

1.3.1. TEMPORAL

Se considero un tiempo estimado entre 4 meses que esta divido entre la investigación, los estudios, desarrollo del prototipo y sus debidas pruebas para correcto funcionamiento.

1.3.2. ESPACIAL

Este prototipo fue diseñado en una vivienda en la parte norte de la ciudad de Guayaquil provincia del Guayas.

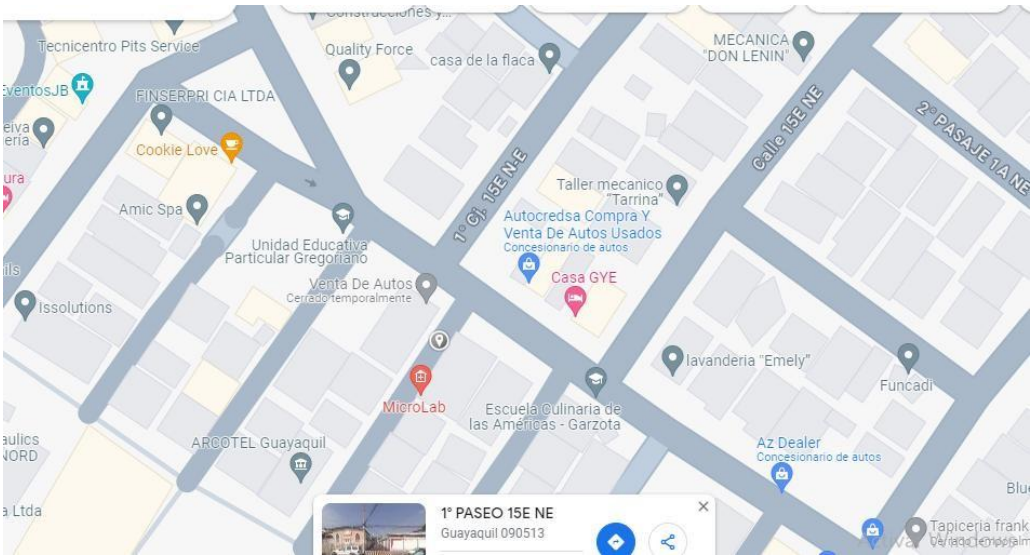


Figura 1. Ubicación satelital de la vivienda

Extraído de https://www.google.com/maps

1.3.3. ACADÉMICA

El proyecto de titulación tendrá lo solicitado por la Universidad Politécnica Salesiana con respecto a los diferentes niveles de investigación y de desarrollo junto a lo requerido.

1.4. INNOVACIÓN

El uso de la inteligencia artificial para detectar fumadores en áreas restringidas

2. OBJETIVOS

2.1. OBJETIVO GENERAL

Diseñar e implementar un prototipo para la detección automática de fumadores en áreas restringidas mediante el uso de inteligencia artificial.

2.2. OBJETIVOS ESPECÍFICOS

- Crear el data set con imágenes de situaciones que presenten a gente fumando.
- Entrenar el modelo de aprendizaje automático para reconocimiento de imágenes e implementarlo en el sistema embebido con el Raspberry Pi en conjunto con la cámara y el parlante.
- Probar el prototipo midiendo su efectividad diferenciando fumadores de no fumadores.

3. FUNDAMENTOS TEÓRICOS

3.1. Inteligencia artificial

La inteligencia artificial es la capacidad de una máquina para mostrar las mismas habilidades que los seres humanos, como el razonamiento, el aprendizaje, la creatividad y las habilidades de planificación. La IA permite a los sistemas tecnológicos percibir su entorno, relacionarse con él, resolver problemas y actuar específicamente. La máquina recibe datos (que ya se han preparado o recogido usando sus propios sensores, como una cámara), los procesa y luego responde a ellos. Los sistemas de IA son capaces de cambiar su comportamiento de una manera determinada, analizando los efectos de las acciones anteriores y trabajando de forma autónoma (Mantaras, 2024).

3.2. Machine Learning

El aprendizaje automático o más comúnmente llamado como “machine learning” se basa en la información, la ciencia de datos y la estadística. Este subcampo de la inteligencia artificial toma observaciones del mundo real y permite a las computadoras (o más específicamente a los algoritmos) identificar patrones a partir de esos datos. De esta forma, el aprendizaje automático nos permite generalizar el comportamiento de estos casos (Alvear, 2022).

3.1. Reconocimiento de imágenes

Gracias a la inteligencia artificial, la tecnología de reconocimiento de imágenes se está haciendo realidad. ¿Cómo funciona? Este es el proceso mediante el cual las computadoras aprenden a reconocer objetos y patrones en imágenes y clasificarlos en categorías especifica. El reconocimiento de imágenes mediante inteligencia artificial es un gran avance en la sociedad tecnológica actual, y la recopilación y la clasificación de grandes cantidades de información visual es esencial para la toma de decisiones en áreas como la seguridad, el marketing y la medicina (Perez, 2023).

3.2. CNN (redes neuronales convolucionales)

En los últimos años, el aprendizaje profundo ha recibido mucha atención y se ha aplicado con éxito en diversos campos. Se han utilizado diversas técnicas de aprendizaje profundo en áreas como robótica, empresas, ciberseguridad, asistentes virtuales, reconocimiento de imágenes y atención médica. También se aplican en análisis de sentimientos y procesamiento del lenguaje natural.

Una técnica de aprendizaje profundo bien establecida es la red neuronal convolucional (CNN), que ha dominado tareas de visión por computadora desde los impresionantes resultados en la competición ImageNet en 2012. Las CNN son útiles en imágenes médicas, ya que pueden detectar tumores y anomalías de manera más precisa en imágenes de rayos X y resonancias magnéticas. Pueden analizar componentes del cuerpo humano, como los pulmones, identificando

ubicaciones potenciales de tumores y otras anormalidades basándose en imágenes previamente procesadas por redes CNN (Yamashita, 2018).

3.3. Detección de objetos

La identificación y localización de objetos, un problema común en la visión artificial consiste en reconocer y ubicar objetos de clases especificas en una imagen, la detección de objetos antes era del aprendizaje profundo era un proceso de varios pasos. En la identificación de objetos se utilizan redes neuronales convolucionales(CNN) que implementa diversos modelos y algoritmos con el fin de realizar la tarea de detección de objetos (Perez j. A., 2020).

3.4. Tensorflow

En estos últimos años de aprendizaje automático se han hecho grandes avances en los campos esto se debe gracias a los modelos de aprendizaje avanzado, enorme conjunto de datos y plataformas de software que ayuda a hacer más sencillo la potencia de cómputo para entrenar estos modelos. Tensor Flow se desarrolló con el propósito de experimentar modelos nuevos y entrenarlos con grandes cantidades de dato y llevarlo a la práctica, tensor Flow ofrece soporte tanto para entrenar modelos a una gran escala como para la inferencia, tensor Flow haciendo uso de sus potentes servidores (viene equipado GPU) para la fluidez de su entrenamiento y ejecutando los modelos que ya están entrenados en diversas plataformas. El elemento que distingue a tensor Flow es su representación unificada que mediante un grafo de flujo de datos (M, 2016).

3.5. Tensorflow lite

Tensor Flow lite constituye una colección de recursos que está diseñada para facilitar a los desarrolladores la ejecución de sus modelos en dispositivo integrados, móviles o de internet de las cosas. Tensor Flow lite presenta un formato especial con el nombre FlatBuffers que se identifica con la extensión de archivo, este se destaca por su portabilidad y eficiencia. Un modelo de tensor Flow lite puede contener metadatos que proporciona una descripción legible por humanos de los modelos, así como datos legibles por maquinas existen diversas formas de generar un modelo de TensorFlow Lite:

- Utilizar un modelo de Tensorflow lite existente: explora los ejemplos disponibles en tensor Flow lite para seleccionar un modelo ya creado
- Crear un modelo de tensor Flow lite: emplea tensorflow lite modelo maker para generar un modelo personalidad con un propio conjunto de datos
- Convierte un modelo de tensor Flow en un modelo de tensor Flow lite: utiliza el conversor de tensor Flow lite para transformar un modelo de tensor Flow en uno que sea compatible con tensor Flow lite (Lite, n.d.).

3.5.1. Efficient Det: detección de objetos escalable y eficiente

si se basa en BiFPN se ha creado una serie de modelos de detección que se denomina efficent det la arquitectura del efficient det se alinea en gran medida con el paradigma de detectores de una sola etapa y se opta por utilizar efficient Nets pre entrenados en ImageNet efficient det es un modelo de detección de modelo e imágenes que se destaca por su eficiencia y precisión, utiliza un enfoque de red neuronal profunda para identificar y poder clasificar los objetos en una imagen (Tan, 2019).

3.6. MediaPipe

Mediapipe presenta un marco integral hecho para la construcción de tuberías de procesamiento que realizan inferencias sobre una variedad de datos sensoriales. Este enfoque permite la creación de tuberías de percepción mediante la interconexión de componente modulares, abarcando desde la interferencia de modelos hasta algoritmos de procesamiento de medios y transformación de datos.

Los datos sensoriales como flujos de audio y video ingresan al sistema, mientras las descripciones resultantes, como la localización de objetos y flujos de puntos de referencias facial, se generan como salida

Media pipe esta específicamente diseñado para las profesionales del aprendizaje automático, incluyendo investigadores, estudiantes y desarrolladores de software. Su utilidad se manifiesta en la implementación de aplicaciones de aprendizaje automático lista para la producción, medida pipe radica en facilitar el prototipo ágil de tuberías de percepción, integrando modelos de inferencia y otros componentes reutilizables. Además, Mediapipe simplifica la implementación de tecnologías de percepción de diversos dispositivos y aplicaciones, abarcando una gran gama de plataformas de hardware. Este marco fomenta mejoras incrementables en las tuberías de percepción mediante su lenguaje de configuración avanzado y herramientas de evaluación.

Así mismo el desarrollo de la misma aplicación para distintas plataformas suele ser un proceso laborioso, implicando la optimización de las inferencias y los pasos de procesamiento para garantizar un rendimiento correcto y eficiente de un dispositivo especifico (lugaresi, 2019).

3.6.1. Detección de palma

Para identificar las posiciones iniciales de las manos, se desarrollan con un modelo de detector de un solo disparo especialmente optimizando para aplicaciones en tiempo real en dispositivos móviles. La detección de manos propone retos considerables, los modelos tanto ligero como completo deben ser capaces trabajar en una gran variedad de tamaños de manos enseñando una escala sustancial a comparación con el marco de la imagen, deben tener la capacidad de detectar manos que están parcialmente ocultas o que se superponen, a comparación de los rostros que tienen un nivel alto de contraste. La detección que se hace en las manos es esta basada únicamente en sus regiones visuales se invita a visualizar la figura 2 (Fernandez J. , 2023).

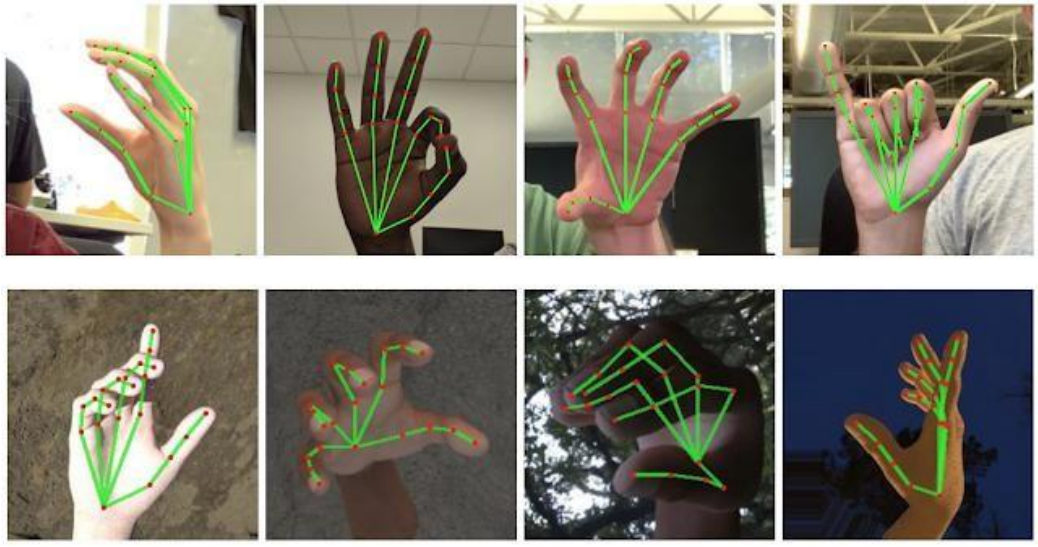


Figura 2 modelo de detección de manos (Fernandez J. , 2023)

3.6.2. Detección de cara

Media pipe face detection es una solución rápida para la detección facial que incluye 6 puntos de referencia y brinda un soporte para identificar múltiples rostros a la vez. Esta tecnología se fundamente en el blazeface, es un detector de rostros eficiente y liviano creado específicamente para la inferencia en unidades de procesamiento grafico (GPU) móviles. La velocidad de detección en tiempo real es excepcional, este detector lo convierte en una elección ideal para aplicaciones que requieren una región facial precisa como entrada para otros modelos especializados tales como la estimación tridimensional de puntos clave faciales se invita a observar la figura 3 (Fernandez J. , 2023).

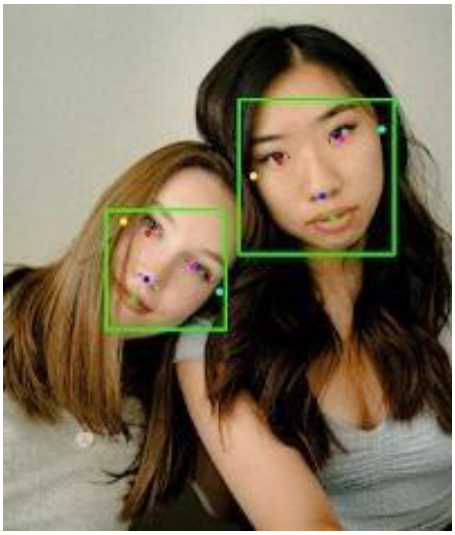


Figura 3 detección de la cara (Omes, omes , 2021)

3.6.3. Blazeface: detección neuronal de rostros

En las redes neuronales profundas se ha logrado conseguir la detección de objetos en tiempo real, lo cual es esencial en el contexto de aplicaciones móviles. Este proceso constituye típicamente el primer paso en el procesamiento de video en estas aplicaciones, seguido por componentes específicos de tarea como la segmentación, el seguimiento o la inferencia geométrica. Por ende, resulta crucial que la inferencia del modelo de detección de objetos se realice de manera ágil, superando

significativamente el estándar de referencias de tiempo real, para la detección facial llamado blazeface, diseñado para optimizar la inferencia en unidades de procesamiento grafico (GPU)móviles, este marco se adapta del modelo single shot multibox detector (SSD) las contribuciones principales incluyen

- Relación con la velocidad de inferencia

Un extractor de características sumamente compacto basado en una red neuronal convolucional con una estructura similar a MobileNet esto está enfocada específicamente al diseño para la detección eficiente de objetos

- Relación con calidad de la predicción

Una estrategia de resolución de empates alternativa a la supervisión, no logros una resolución de empates más estables y suave entre predicciones superpuestas (Bazarevsky, 2019).

3.7. Cámara HD

Una cámara HD es un dispositivo que se utiliza para la captura de imágenes con una resolución más amplia que las cámaras habituales la cámara que se utilizó cuenta con un sensor de imágenes de 720HD Pixel CMOS y con una resolución máxima de 1MP/1280 x 720/ 640 x 480 Pixeles, se puede utilizar en Windows XP, Vista, 7 or late como se muestra en la figura 4 (genius, n.d.).

Figura 4 cámara HD (genius, n.d.)

3.8. Parlantes

Los parlantes o también conocido por altavoces son dispositivos creados para par convertir señales en onda sonoras los parlantes, como se muestra en la figura 3 son los componentes encargados de proveer la salida de audio desde tu computadora. Gracias a estos dispositivos, puedes disfrutar del sonido de la música o videos que reproduzcas. Su conexión varía según el modelo, algunos se conectan al puerto "USB", mientras que otros lo hacen al puerto de audio c (basica, n.d.).

3.9. Sistemas embebidos

Los sistemas embebidos también se los conoce como sistemas empotrados son sistemas que están a cargo de cumplir un deber en específico, por lo general estos sistemas tienen recursos limitados y sus primeras etapas de desarrollo estos dispositivos tenían opciones de comunicación que se limitaban con su entorno exterior como los otros dispositivos. Sin embargo, debido a los avances tecnológicos en la industria estos disipativos ahora poseen la capacidad de establecer comunicación directa con el usuario e incluso tener la capacidad de realizar intercambios de información entre maquinas mediante el internet.

Los sistemas embebidos que son sistemas electrónicos digitales basados en microcontroladores tienen la función específica de llevar a cabo una tarea determinada, comúnmente esta tarea tiende a ser estable a lo largo de tiempo, este enfoque inicial ha evolucionado con el tiempo y los avances de la industria (LUNA, 2021).

3.10. Raspberry PI 4 modelo B

Este modelo se anunció su salió en junio del 2019, en lugar de los puertos HDMI de tamaño completos se los sustituyeron por dos de puertos MicroHDMI, tiene la capacidad de soportar una pantalla con resolución de 4k a 60Hz o dos pantallas de 4K a 30Hz, se ha incorporado USB 3.0 por primera vez, y el puerto Ethernet ya no una velocidad máxima de 300Mbps, también cuenta con un procesador Broadcom

nuevos y tres veces más efectivo que el anterior (Raspberry PI 3 modelo A) como se puede ver en la figura 5 (vazquez, 2022).

Figura 5 Raspberry pi 4 modelo B (Raspberry , n.d.)

3.10.1. Case con ventilador

El case con ventilador para el Raspberry es una carcasa diseñada específicamente para proteger y que se mantenga refriada la placa del Raspberry PI, es un tipo de estuche viene incluido con un ventilador para ayudar a dispersar el calor que genera por el funcionamiento del Raspberry, este case ayuda a mejorar el rendimiento del dispositivo como se puede observar en la figura 6 (Rus C. , 2020).

Figura 6 Case con ventilador

3.10.2. Fuente de poder

La fuente del poder para el Raspberry se refiere al componente que le suministra la energía eléctrica para hacer funcionar el Raspberry. Esta fuente en casos suele ser un adaptador que suministra 5 voltios y 3 amperios, todos los modelos del Raspberry PI reciben energías a través de un conector microUSB como se puede observar en la figura 7 (Factory, 2024).

Figura 7 fuente de poder

3.10.3. Unidad de estado sólido (SSD)

Una unidad de estado sólido (por sus siglas en ingles solid state drive) un SSD constituye un dispositivo de almacenamiento que emplea una memoria flash para preservar datos de forma permanente. Estas unidades se ofrecen como una segunda opción para los discos duros convencionales el SSD que escogido tiene una capacidad de almacenamiento de 120GB, como se puede observar en la figura 8 (Fernandez Y. , 2023).

Figura 8 Unidad de estado solido

3.10.4. Cable HDMI a microHDMI

El cable HDMI a micro HDMI es un cable que tiene un puerto HDMI en un extremo y un conector HDMI en el otro. El conector HDMI es un estándar para la transmisión de un audio y video de alta calidad, este cable hace fácil la conexión entre los dispositivos que utilizan diferentes tamaños de conectores HDMI, dando paso a la transmisión de señales de audio y video entre ellos (FotoAccess, 2022).

3.1. Google Colab

Google colab o colaboratory es un servicio que tiene dos funciones una que es gratis y la otra que es de servicio pagado esta herramienta que ofrece Google a sus usuarios para incentivar la investigación sobre el estudio relacionado sobre inteligencia artificial (Google, n.d.).

3.2. Cuda/Cudnn

Cuda (por sus siglas en ingles compute unified device architecture) es una plataforma fue desarrollada por NVIDIA, permite realizar procesamientos gráficos es muy utilizado en las librerías de Python (Tensor Flow, Pytorch,etc) y Cudnn (por sus siglas en ingles NVIDIA Cuda Deep Neural Network)es una plataforma que permite agilizar el proceso de GPU (NVIDIA, 2023).

3.3. Formato de anotación Pascal VOC (visual object classes)

Pascal (VOC) cuenta con dos partes esenciales, un grupo de datos de imágenes y anotaciones accesibles al público junto con software de evaluaciones estandarizado, y una competición y taller que tienen lugar anualmente el conjunto de datos de VOC2007 se compone de fotografías de consumidores con anotaciones recopiladas del sitio web de intercambio de fotos, desde el año 2006, se ha lanzado anualmente un nuevo conjunto de datos de anotaciones de verdad. (Everingham, n.d.).

3.4. Roboflow

Roboflow introduce 100 puntos de referencias específicos del dominio que van más allá de los objetos convencionales en un contexto determinado. Este enfoque evalúa como se desempeñan los modelos en gran variedad de problemas, incluyendo aplicaciones médicas, imágenes aéreas, videojuegos y otros escenarios. En comparación con los otros puntos de referencia tradicionales para la detección de objetos, que suelen enfocarse en un solo conjunto de datos, como Microsoft COCO y Pascal VOC, estos nuevos puntos de referencia buscan una adaptación más amplia aunque los conjuntos de datos convencionales son semánticamente diversos en su propio dominio, se centran en la optimización de métricas específicas para un único conjunto de datos, lo que no proporciona una representación completa del nivel de datos generalización que ha alcanzado este modelo (RoboFlow, 2023).

3.5. Ide Geany

El IDE Geany es entorno de desarrollo integrado (IDE sus siglas en el idioma inglés) que sirve como un editor de texto ligero con funciones de IDE para programar también se lo puede utilizar para el resaltado de sintaxis, finalización automática de código, gestión de proyecto, plugins entre varias funciones más que tiene este editor de texto. El Geany se puede utilizar en diferentes sistemas operativos como GNU/Linux, Mac Os x y Microsoft Windows tiene soporte para diversos lenguajes (C++, PASCAL, C, JAVA, Python, PHP, HTML, etc) (GEANY, 2023).

3.6. Python

Python principalmente destaca como un lenguaje de programación extraordinariamente robusto y accesible, cuenta con un lenguaje de programación amigable y fácil. Su potencia radica en su eficiencia de estructuras de datos de muy alto nivel, esto está respaldado por un sistema de programación orientado a objetos que, aunque simple demuestra ser sumamente efectivo. Todas estas características lo posicionan como una elección ideal para realizar tareas de scripting y el desarrollo ágil de aplicaciones en diversas áreas. Es un lenguaje versátil que demuestra su eficacia en la mayoría de las plataformas, ofreciendo una herramienta excepcional para profesionales de la programación y desarrolladores en general (Python, 2024).

3.6.1. Pycharm

Pycharm es un entorno de Python diseñado principalmente diferentes programadores con el fin de brindarles todas las herramientas que se necesita para un desarrollo del software de Python (Pycharm, 2023).

3.7. OpenCV

OpenCV es una biblioteca de visión por computadora de código abierto que se distingue como un recurso invaluable al incorporar una amplia gama de algoritmos, que suman varios cientos, está diseñada para abordar retos específicos en el campo de la visión de las computadoras. Esta biblioteca presenta como una interfaz esencialmente basada en C++, marcando una evolución trascendental en la funcionalidad y el rendimiento de la biblioteca (OpenCV, 2024).

3.7.1. RGB

Una imagen es un grupo de pixeles que se encuentra distribuido en una figura de cuadricula, cada uno de los pixeles a su misma vez está conformado por diferentes componentes, que cuando son combinados forman colores que se puede visualizar en una imagen

RGB (Red Green Blue / Rojo Verde Azul) es uno de los espacios de color que más se utiliza cuando se habla de figura o imágenes y cuando se usa OpenCV lee una imagen o dibuja figuras geométricas (Omes, 2020).

3.8. SMTP LIVE

Se trata de un protocolo de comunicación comúnmente en internet que se emplea para enviar mensajes de correo electrónico entre diferentes servidores de correo. Su función principal radica en dirigir y hacer llegar el correo electrónico al servidor correspondiente del destinatario (Zelezny, 2023).

3.8.1. MimeMultipart

Las extensiones multipropósito de correo de internet (MIME) representa un gran estándar de internet utilizado para facilitar la transferencia de uno o más archivos adjuntos, ya sean de texto o no. Los archivos adjuntos no textuales pueden abarcar archivos de gráficos, audio y video, permitiendo así la inclusión de diversos tipos de contenido en los mensajes. La implementación MIME en B2B advanced comunications da la posibilidad de mandar múltiples archivos adjuntos en un solo mensaje ebMS (tipo de contenido especifico) (IMB, 2021).

3.9. PYSIMPLEGUI

Una de las alternativas más accesibles para el desarrollo de aplicaciones con una interfaz gráfica de usuario (GUI) en el software de Pythom es simpleGUI. Esta biblioteca permite crear GUI funcionales de manera eficientes para sistemas operativos como Windows, macOS y Linux, todo ello con unas pocas líneas de código esta característica asegura que las aplicaciones desarrolladas sean altamente portables entre distintas plataformas (Rodriguez, 2022).

4. MARCO METODOLÓGICO

Para el desarrollo del prototipo se optó por la combinación del modelo customizado de **TensorFlow** lite con la detección de manos y cara de **Mediapipe** esto para reducir el número de falsos positivos que se presenten en las detecciones. Se utilizando la librería **OpenCV** para capturar el video en tiempo real y, al aplicar la detección de fumadores mande un audio de aviso por un parlante, así como guardar el fotograma de la detección en una carpeta local y enviarlo por correo electrónico.

Figura 9. Conexiones

4.1. Gestión de dependencias

Como se muestra en la figura 10 Para instalar las librerías y sus dependencias se pueden instalar por la consola como administrador con el comando **sudo apt-get install "libreria"**, con el comando **pip3 install "libreria"**, o como se aprecia en la figura 18 utilizar el **IDE Thonny Python** que cuenta con una herramienta para administrar paquetes la cual se puede utilizar para elegir que versiones de las librerías y sus dependencias a instalar.

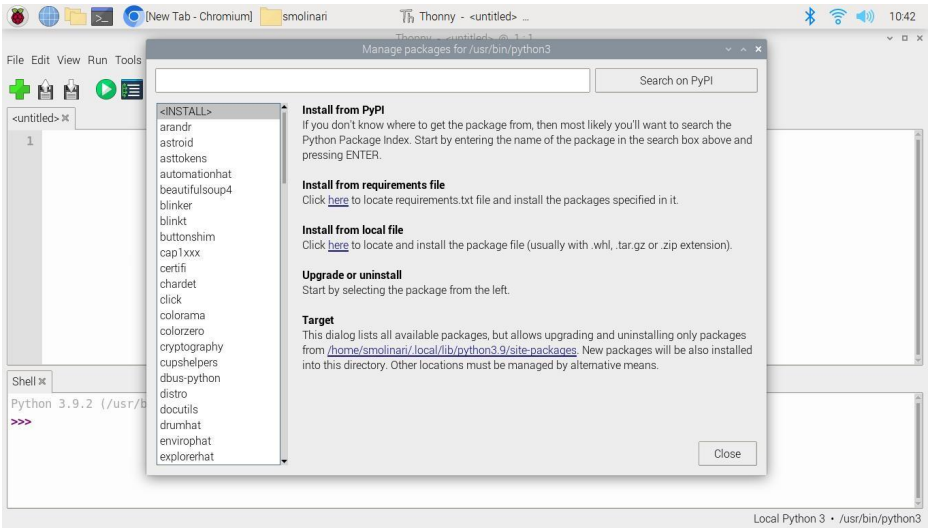


Figura 10. IDE Thonny Python administrador de paquetes

4.2. Captura de video en tiempo real mediante la librería OpenCV

Para instalar la librería **OpenCV** se utilizó la herramienta de administración de paquetes del **IDE Thonny Python** como se muestra en la figura 11. En la que se instaló la versión 4.5.3 de la librería.

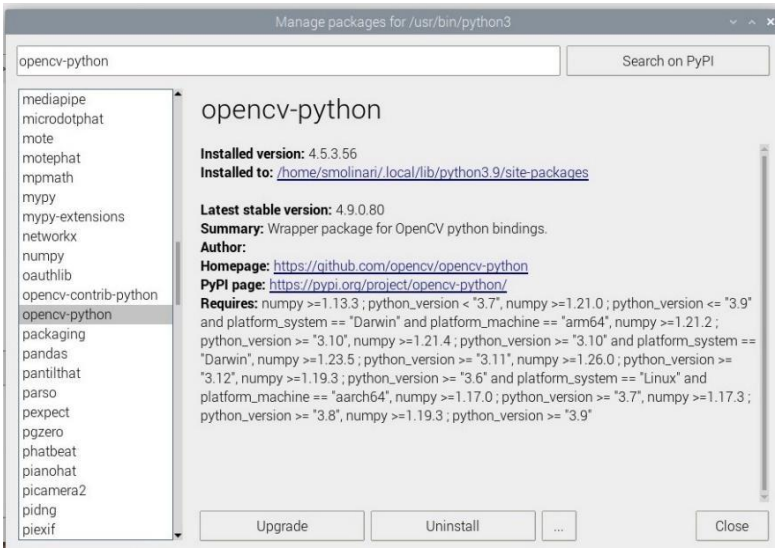


Figura 11. Instalación de la librería OpenCV

Para capturar el video de la webcam conectada a la Raspberry Pi, se debe llamar a la librería de **OpenCV** como, **import cv2 as cv**, luego se le asigna una variable al objeto y le dice que abra la

cámara, **cam=cv.VideoCapture(0),** donde 0 se refiere a la primera entrada de video que encuentre, después se utilizó un ciclo **while** en el que se guardan los valores de la función **cam.read()** dentro de las variables **ret** e **im** y se procede a mostrar la variable **im** que contiene la imagen capturada en una ventana para poder visualizarla como se logra observar en la figura 12 y 13

```
1   import cv2 as cv
2
3
4
5   #abrir camara
6   cam = cv.VideoCapture(0)
7
8   while True:
9       ret, im = cam.read()
10
11      #mostrar webcam
12      cv.imshow('ObjectDetection', im)
13      key = cv.waitKey(1)
14      if key == 27:
15          break
16  cv.destroyAllWindows()
17
```

Figura 12. Código para obtener la imagen de la webcam a través de la librería OpenCV

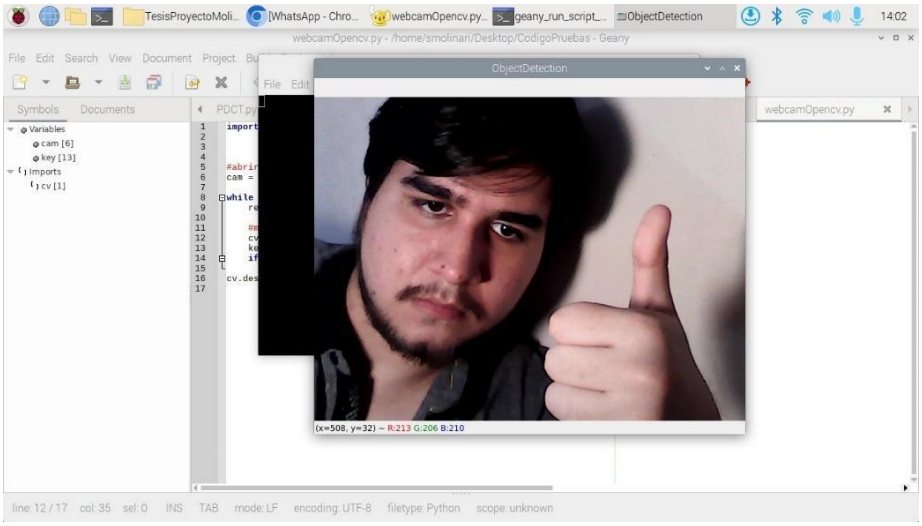


Figura 13**. Imagen obetenida con la librería OpenCV**

4.3. Detección de manos y cara

Para detectar las manos y la cara de una persona, se empleó la librería de Google **Mediapipe**. Esta librería ofrece modelos de machine learning en varias funciones llamadas **"solutions**" para diversas aplicaciones y en este caso se utilizó una de sus funciones para identificar y seguir las manos y los dedos de las personas, así como otra función que permite identificar las caras de las personas como se logra observar en la figura 14.

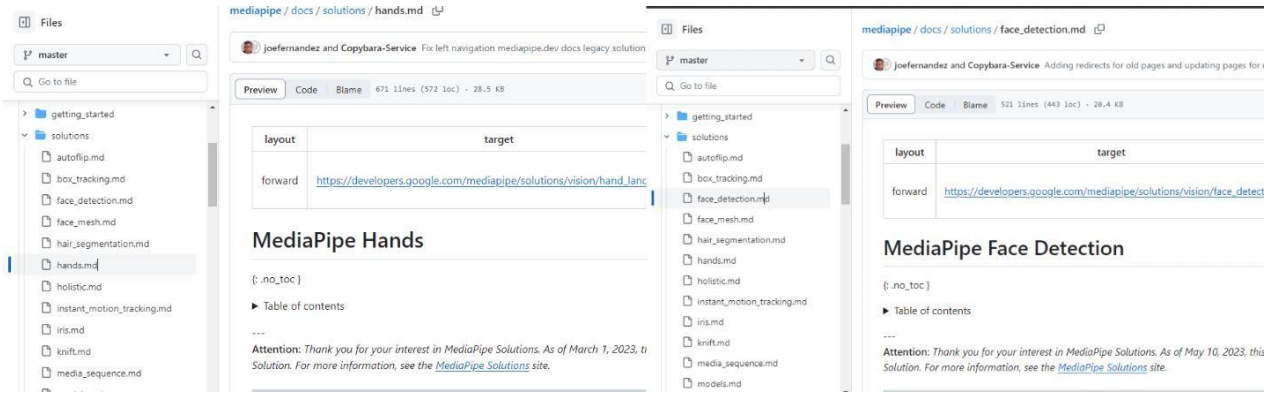


Figura 14. Funciones o "solutions" de mediapipe (mediapipe, 2023)

4.3.1. Instalación de Mediapipe en la Raspberry Pi

Para instalar la librería **Mediapipe** se utilizó la herramienta de administración de paquetes del **IDE Thonny Python** como se muestra en la figura 15. En la que se instaló la versión 0.8.9.1 de la librería.

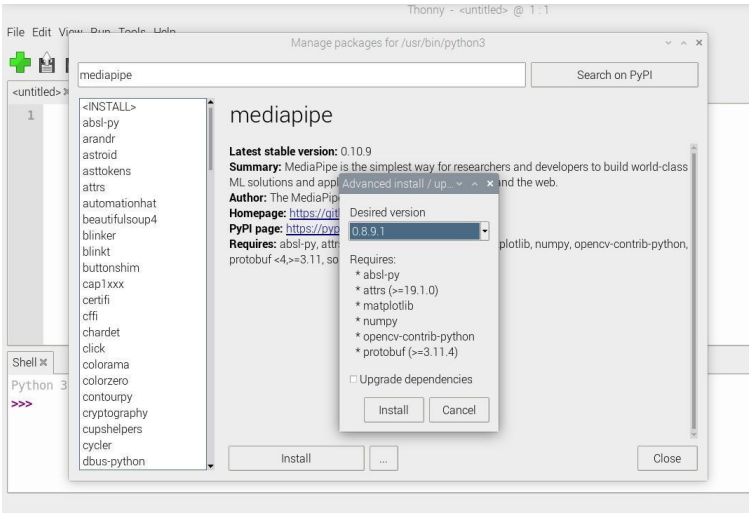


Figura 15. instalación de librería Mediapipe y sus dependencias

Debido a un error con la versión predeterminada de la librería **protobuf** en la instalación de **Mediapipe** como se muestra en la figura 24, es necesario degradar la librería a una versión en el rango de 3.20.X como se indica en la figura 16.

Figura 16. Error al importar la librería Mediapipe

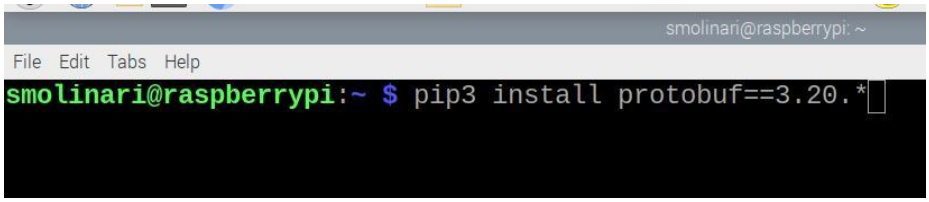


Figura 17. instalación de una versión inferior de la librería protobuf

4.3.2. Código con Mediapipe para detección de manos y cara

Como se muestra en la figura 18, para llamar a la librería **Mediapipe** se utiliza **import mediapipe as mp**, luego se llama a los modelos para la detección de manos y cara con **mp_face** y **hands**, a los cuales se les asigna parámetros iniciales, la confianza de detección en el caso de la cara, y 3 parámetros en el caso de las manos que son: confianza de detección, confianza de seguimiento y numero de manos detectadas. Se declara una variable para almacenar el módulo que dibuja las detecciones llamado **mp_drawing** y otra variable que almacena el módulo que da estilo y color llamada **mp_drawing_styles**.

```python
import cv2 as cv
import mediapipe as mp

# Configuración del modelo de detección de caras
mp_face_detection = mp.solutions.face_detection
mp_face = mp_face_detection.FaceDetection(min_detection_confidence=0.8)

# Configuración del modelo de detección de manos
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=2,
                       min_detection_confidence=0.8,
                       min_tracking_confidence=0.8)
```

Figura 18. Parámetros iniciales para modelos de detección de manos y cara

Después de asignar nombres más manejables a los módulos de **Mediapipe**, se procede a crear una función que procesa la imagen y aplica la detección de cara, como se muestra en la Figura 19. Esta función toma como entrada el fotograma de la imagen capturada con **OpenCV** y la procesa aplicándole el modelo de detección de cara de **Mediapipe**. La detección se almacena en la variable **result_face**. Si se detecta una o más caras, se procede a dibujarlas en la imagen capturada con **mp_drawing** y a obtener los **landmarks** y guardarlos en **face_landmarks**, que son los valores de las coordenadas de los puntos de la cara, como se muestra en la Figura 20. El punto que se desea almacenar es el del labio, por lo que se guarda el **landmark** 3. Para ello, se multiplican sus coordenadas en X y en Y por el largo y el ancho de la imagen, los cuales se obtienen mediante la función **img.shape**, donde los subíndices **[1]** y **[0]** corresponden al ancho y al largo de la imagen, respectivamente.

Para finalizar, se retorna el fotograma con la detección dibujada

```python
# Deteccion cara
def detect_and_draw_Face(img):
    global point_3_face  # Declarar como variable global
    # Procesamiento de caras
    result_face = mp_face.process(img)

    # Dibujo de la cara si se detecta
    if result_face.detections:
        for detection in result_face.detections:
            # Dibujo de la detección de la cara
            mp_drawing.draw_detection(img, detection)

            # Obtener las coordenadas de los landmarks de la cara
            face_landmarks = detection.location_data.relative_keypoints

            # Almacenar las coordenadas del punto 3 de face_landmark
            if len(face_landmarks) > 3:
                point_3_face = (int(face_landmarks[3].x * img.shape[1]), int(face_landmarks[3].y * img.shape[0]))
    return img
```

Figura 19. Función de detección de cara

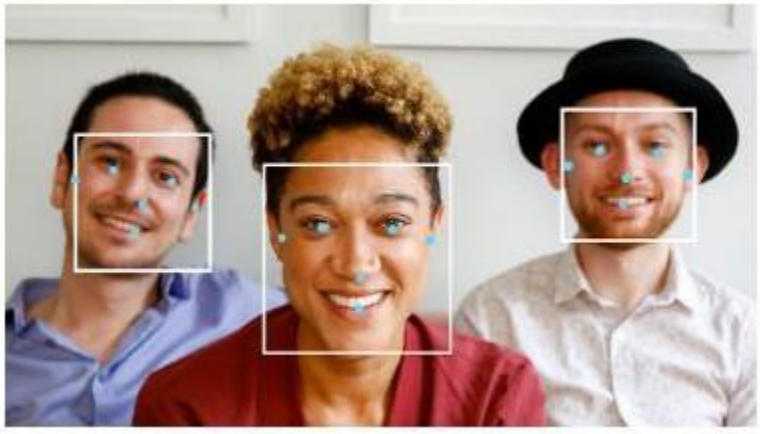


Figura 20. Landmarks de la cara (mediapipe, 2023)

A continuación, se procederá a crear una función que procese y aplique la detección de manos y dedos, como se muestra en la Figura 21. Esta función toma como entrada el fotograma de la imagen capturada con **OpenCV** y lo procesa aplicándole el modelo de detección de manos de **Mediapipe**. La detección se almacena en la variable **result_hands**. Si se identifican una o varias manos, se procede a localizar los **landmarks** de los dedos y se dibujan en el fotograma mediante **mp_drawing**.

Los puntos y conexiones de los **landmarks** se representan con los parámetros **hand_landmarks** y **HAND_CONECTIONS**, a los cuales se les aplica un estilo diferenciado por colores utilizando el parámetro **mp_drawing_styles**.

Las coordenadas de los **landmarks** se almacenan en la variable **landmarksHand**, luego almacenamos los **landmarks** de las puntas de los dedos índice y pulgar en variables individuales siendo los landmarks 8 y 4 respectivamente como se muestra en la figura 30. Para ello, se multiplican sus coordenadas en X y en Y por el ancho y el largo de la imagen respectivamente. Luego, se establece una condición que será verdadera cuando todas las coordenadas de los **landmarks** de los dedos se encuentren más abajo que las coordenadas de las puntas de los dedos índice y pulgar. Se tiene en cuenta que el tamaño por defecto de la imagen capturada por la cámara web es de 640 píxeles de ancho por 480 píxeles de largo. A medida que las puntas de los dedos se acercan a la parte superior de la imagen, sus coordenadas en Y se acercan a 0, y hacia la derecha, sus coordenadas en X también se acercan a 0, como se muestra en la Figura 22. La condición para determinar cuál **landmark** está más arriba se basa en que su valor sea menor al del otro **landmark**.

Finalmente, la función retorna el fotograma con las detecciones dibujadas.

```python
# Deteccion manos
def detect_and_draw_Hands(img):
    global point_8_hand, point_4_hand, condition

    # Procesamiento de manos con Mediapipe
    result_hands = hands.process(img)
    if result_hands.multi_hand_landmarks:

        # Dibujo de los puntos para cada mano detectada
        for hand_landmarks in result_hands.multi_hand_landmarks:
            mp_drawing.draw_landmarks(img,
                                      hand_landmarks,
                                      mp_hands.HAND_CONNECTIONS,
                                      mp_drawing_styles.get_default_hand_landmarks_style(),
                                      mp_drawing_styles.get_default_hand_connections_style()
                                      )

        # Obtener las coordenadas de los landmarks
        landmarksHand = hand_landmarks.landmark

        # Almacenar las coordenadas del punto 8 de landmarksHand
        if len(landmarksHand) > 8:
            point_8_hand = (int(landmarksHand[8].x * img.shape[1]), int(landmarksHand[8].y * img.shape[0]))

        # Almacenar las coordenadas del punto 4 de landmarksHand
        if len(landmarksHand) > 4:
            point_4_hand = (int(landmarksHand[4].x * img.shape[1]), int(landmarksHand[4].y * img.shape[0]))

        # Condición manos puntos 12 y 8
        condition = all(
            landmarksHand[i].y < landmarksHand[j].y
            for i, j in [(12, 20), (8, 20), (12, 16), (8, 16), (12, 4), (8, 4)])

    return img
```

Figura 21. Función de detección de manos

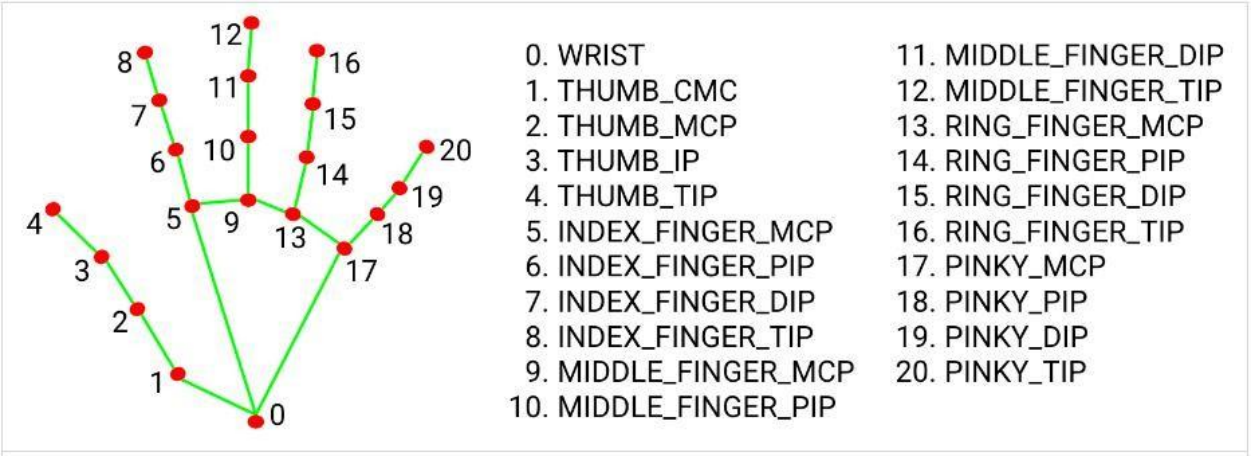


Figura 22. Landmarks de los dedos de las manos (mediapipe, 2023)

La librería de **OpenCV** convierte internamente la imagen de **RGB** a **BGR**, pero la librería **Mediapipe** necesita trabajar con formato **RGB** por lo tanto se debe realizar la conversión antes de entregarle el fotograma a las funciones de detección de cara y manos así que se hizo otra función que junte estas 2 funciones ya mencionadas para convertir el fotograma 1 sola vez en el código, esto se hace mediante la función **cv.cvtColor()** cuyos parámetros son la imagen a modificar y el tipo de conversión que se quiere obtener como se muestra en la figura 23, luego de realizar la conversión se procesa la imagen pasándola por las 2 funciones las cuales le dibujaran las detecciones de cara y manos para finalmente devolver al formato **BGR** y la función devuelva el fotograma con las detecciones.

```
# Proceso principal para deteccion de cara y manos
def detect_and_draw(img):
    # Conversión de BGR a RGB para el procesamiento de Mediapipe
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    detect_and_draw_Face(img)
    detect_and_draw_Hands(img)
    # Conversión de RGB a BGR para mostrar la imagen con OpenCV
    img = cv.cvtColor(img, cv.COLOR_RGB2BGR)
    return img
```

Figura 23. Función para detectar y dibujar las detecciones de manos y cara

Una vez se tienen definidas las funciones se procede a abrir la cámara con **cv.VideoCapture(0)**, para luego crear un ciclo **while** en el que iniciamos las variables los **landmarks** y de las distancias que se quieren calcular, en este caso se usó la **fórmula de la distancia euclidiana**.

$$(_{1,\,2}) = \qquad \sqrt{(_{2} - _{1}{}^{2} +)(_{2} - _{1})^{2}}$$

para calcular la distancia entre 2 puntos que en este caso vienen a ser la distancia de los **landmarks** 4 y 8 de las manos, esto representa la distancia entre la punta del dedo índice con la punta del pulgar, y la distancia entre el **landmark** 3 de la cara y el **landmark** 8 de la mano que representan el centro de la boca con la punta del dedo índice, tal como se indica en la figura 24. Se guardan los valores de **cam.read()** en las variables **ret** e **im**. La variable **im** es la imagen capturada por la librería **OpenCV** esta se la pasa por la función principal que contiene a las otras 2 funciones de detección de manos y cara llamada **detect_and_draw.**

```
cam = cv.VideoCapture(0)

while True:

    # Inicializar las distancias
    distance_3_8 = None
    distance_4_8 = None

    # Variables para almacenar las coordenadas de los puntos 3 de face_landmark y los puntos 8 y 4 de landmarksHand
    point_3_face = None
    point_8_hand = None
    point_4_hand = None
    condition = False

    ret, im = cam.read()

    # Procesamiento de la imagen
    img_proces = detect_and_draw(im)

    # Calcular la distancia si ambos puntos están disponibles
    if point_4_hand is not None and point_8_hand is not None:
        distance_4_8 = ((point_4_hand[0] - point_8_hand[0]) ** 2 + (
                point_4_hand[1] - point_8_hand[1]) ** 2) ** 0.5

    # Calcular la distancia si ambos puntos están disponibles
    if point_3_face is not None and point_8_hand is not None:
        distance_3_8 = ((point_3_face[0] - point_8_hand[0]) ** 2 + (
                point_3_face[1] - point_8_hand[1]) ** 2) ** 0.5
```

Figura 24. Cálculo de las distancias de los landmarks

Para visualizar las coordenadas de los **landmarks**, el estado de la condición que indica si están levantados los dedos índice y medio más arriba que el resto de **landmarks** y los resultados de los cálculos de las distancias se imprimen utilizando la función **print()**, como se indica en la figura 25.

Para finalizar se muestra las detecciones que están almacenadas en la variable i**mg_proces** mediante la función **cv.imshow()**, cuyos parámetros son el nombre de la ventana que se abre y variable que contiene la imagen a mostrar.

Para salir del ciclo **while** se debe esperar a una tecla con el comando cv.waitKey(), si dentro de la función hay un número mayor a 0 se esperara a un delay en milisegundos y si el número es 0 se tomará el valor de una tecla que se presione en este caso la tecla ESC la cual es la numero 27 en el código **ASCII**.

```
    print(f"punto3_Cara: {point_3_face}")
    print(f"punto4_Mano: {point_4_hand}")
    print(f"punto8_Mano: {point_8_hand}")
    print(f"distancia de la boca a la punta del dedo indice: {distance_3_8}")
    print(f"distancia entre las puntas de los dedos pulgar e indice: {distance_4_8}")
    print(f"los dedos indice y medio estan levantados: {condition}")
    cv.imshow('Mediapipe', img_proces)
    key = cv.waitKey(1)
    if key == 27:
        break
cv.destroyAllWindows()
```

Figura 25. Visualizar detecciones, coordenadas y distancias

4.4. Modelo de aprendizaje automático

La decisión de adoptar un modelo de **detección de objetos** se fundamenta en la necesidad de enfocarse en la identificación precisa de un objeto específico, en este caso, el cigarrillo, para determinar la presencia de un fumador en las imágenes. A diferencia de los métodos de clasificación de imágenes que evalúan la totalidad de la imagen, la **detección de objetos** permite localizar con precisión regiones de interés, proporcionando ventajas significativas en términos de eficiencia y precisión. Es esencial reconocer que todas las imágenes de fumadores contendrán el objeto de interés, el cigarrillo. La elección de un modelo de detección de objetos busca optimizar la eficiencia y precisión del sistema al abordar directamente la identificación del objeto relevante. Esta adaptación específica del enfoque permite concentrarse en la tarea principal de determinar la presencia de un fumador en una imagen, ofreciendo así una aproximación más efectiva para alcanzar el objetivo propuesto.

En la exploración de herramientas para el entrenamiento y despliegue de modelos de **detección de objetos**, se llevó a cabo una fase inicial utilizando la librería **Ultralytics**. Durante esta etapa, se enfocó en la implementación de la arquitectura **YOLO**, específicamente la versión **YOLOv8**, en el entorno de la Raspberry Pi.

No obstante, al evaluar el rendimiento del modelo predeterminado proporcionado por **YOLOv8** en la Raspberry Pi como se muestra en la figura 26, se evidenció que esta arquitectura no está optimizada para operar exclusivamente en entornos con recursos limitados de CPU, sino más bien, está diseñada para un desempeño óptimo con el respaldo de tarjetas gráficas o GPUs.

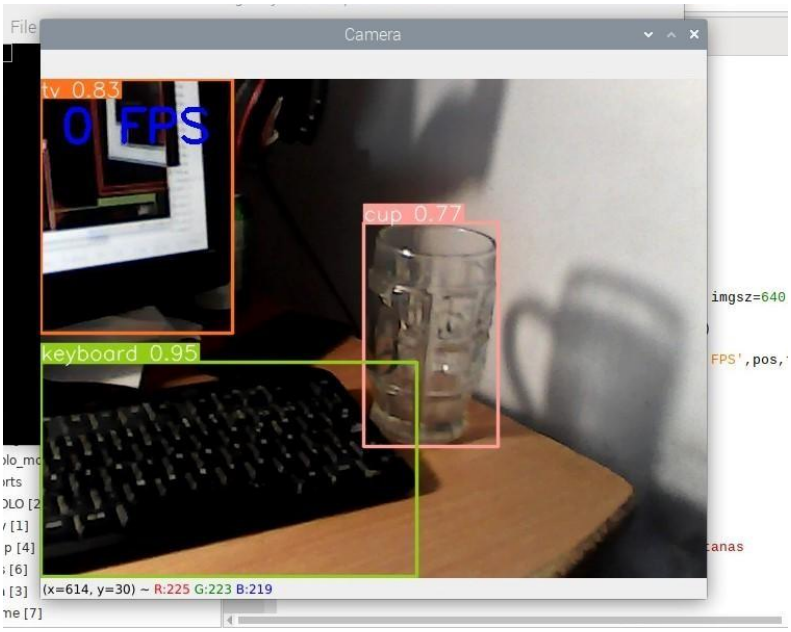


Figura 26. Modelo de prueba de YOLOv8

En busca de un enfoque más adecuado para el entorno de la Raspberry Pi, se tomó la decisión de optar por un modelo entrenado con las librerías de **TensorFlow**, específicamente **TensorFlow Lite** con la arquitectura **EfficientDet_Lite0** presenta características fundamentales para nuestros requisitos, tales como una orientación hacia el rendimiento en CPU, un tamaño compacto y una velocidad eficiente de detección.

La elección de **TensorFlow Lite** se respalda además por la existencia de un repositorio dedicado en GitHub, proporcionando recursos específicos y herramientas para el entrenamiento y despliegue de modelos diseñados para la Raspberry Pi. Este enfoque no solo se alinea con la necesidad de un modelo optimizado para entornos con recursos limitados, sino que también facilita la implementación y gestión del sistema de **detección de objetos**.

4.4.1. Instalación de TensorFlow en la Raspberry Pi

Para instalar la librería de **TensorFlow** y sus dependencias en la Raspberry Pi se debe ir a su GitHub oficial y copiar el comando para clonar el repositorio como se indica en la figura 27.

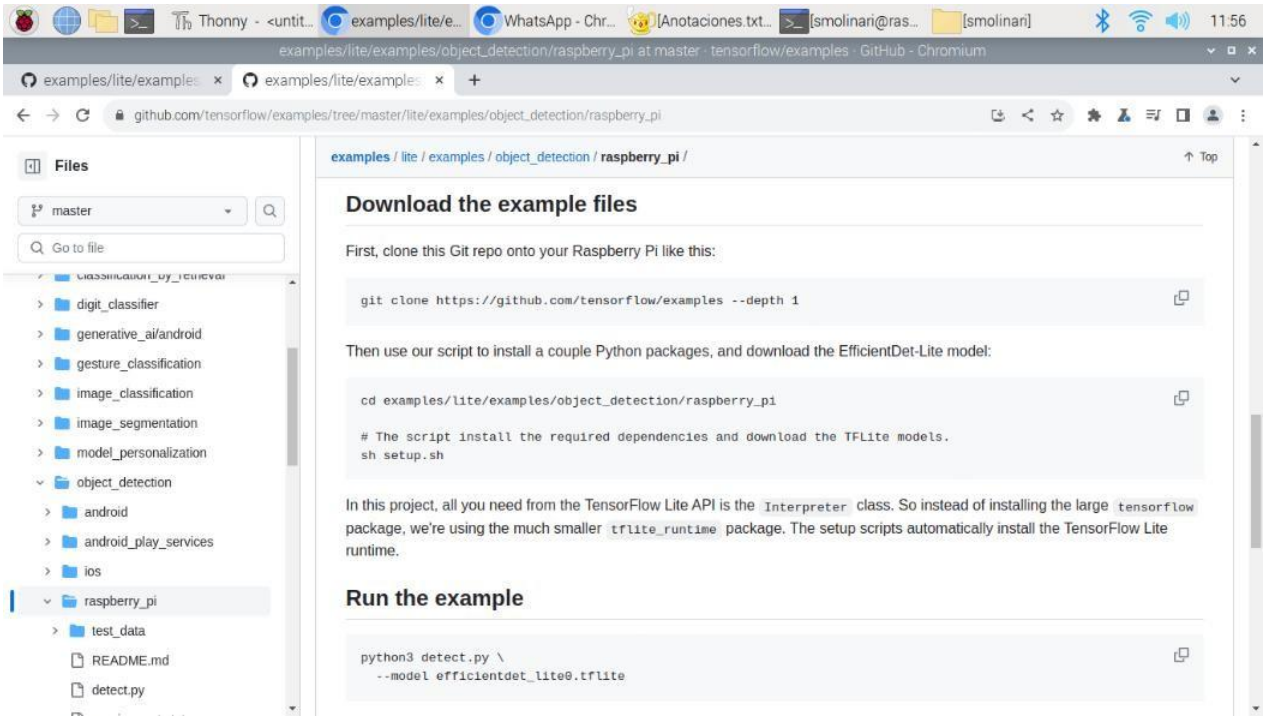


Figura 27. GitHub oficial de tensorflow (tensorflow, 2023)

Como se detalla en la Figura 28, primero se crea una carpeta llamada Python con el comando **mkdir Python**. Luego, se procede a acceder a dicha carpeta mediante el comando **cd Python**. Dentro de esta carpeta, se crea otra llamada TFL con el comando **mkdir TFL**, y se utiliza **cd TFL** para ingresar a ella. Este proceso se realiza con el fin de facilitar el acceso a los archivos y

carpetas que se encuentran en el repositorio de GitHub. Posteriormente, se utiliza el comando **git** para clonar el repositorio (que se muestra en la figura 35) dentro de la nueva carpeta.

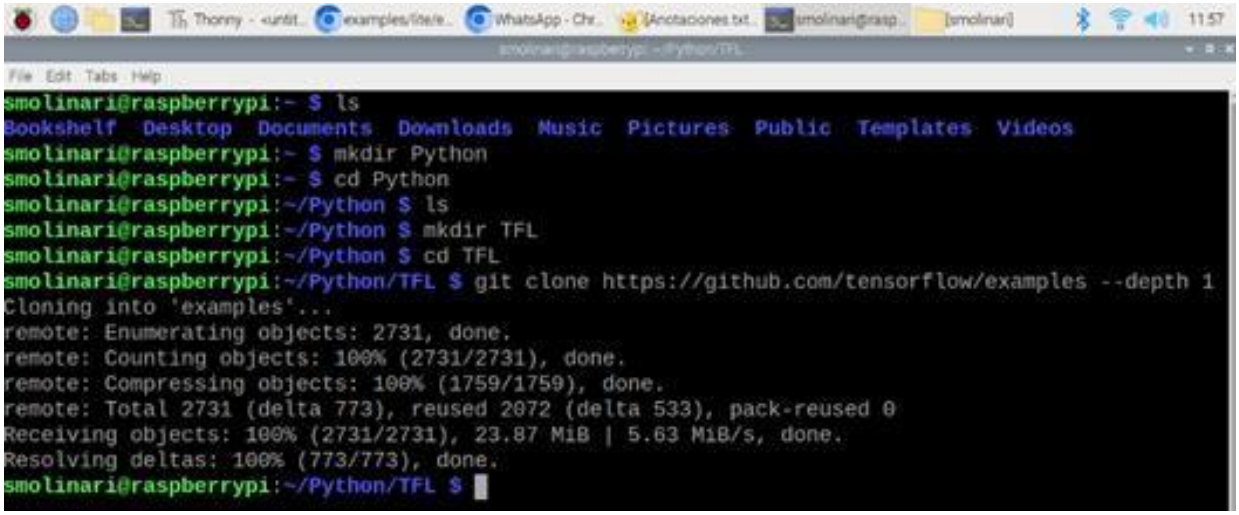


Figura 28. Clonación del repositorio de la librería tensorflow, carpetas y dependencias para raspberry pi

Cuando acabe la clonación del **github** de **TensorFlow** dentro de la carpeta TFL se creará otra carpeta llamada **examples**, para acceder a el ejemplo de detección de objetos en Raspberry Pi se debe utilizar el comando **cd examples/lite/examples/object_detection/raspberry_pi** (que se muestra en la figura 35), dentro de esta dirección se encuentra el archivo setup.sh el cual debemos ejecutar utilizando el comando **sh setup.sh como se muestra en la figura 29**

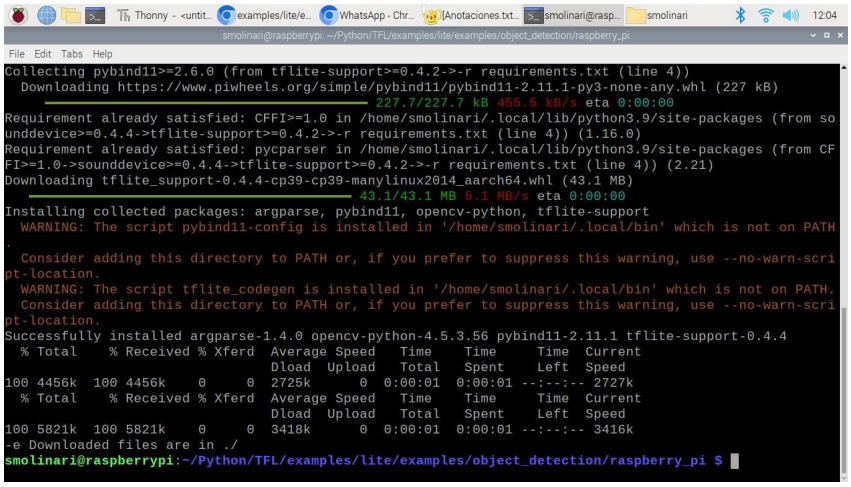


Figura 29. Instalación completa del archivo setup.sh

Cuando se ejecuta el comando **ls** en la dirección actual se pueden observar varias carpetas y archivos nuevos como se muestra en la figura 30, de estos se extraen: **efficientdet_lite0.tflite** el cual es un modelo de detección de objetos pre entrenado con algunas clases comunes y **utils.py** un archivo un archivo de Python que se lo importara en el código para realizar la detección de objetos, los cambiamos de directorio con los comandos **mv efficientdet_lite0.tflite ~/Python** y **mv utils.py ~/Python** respectivamente

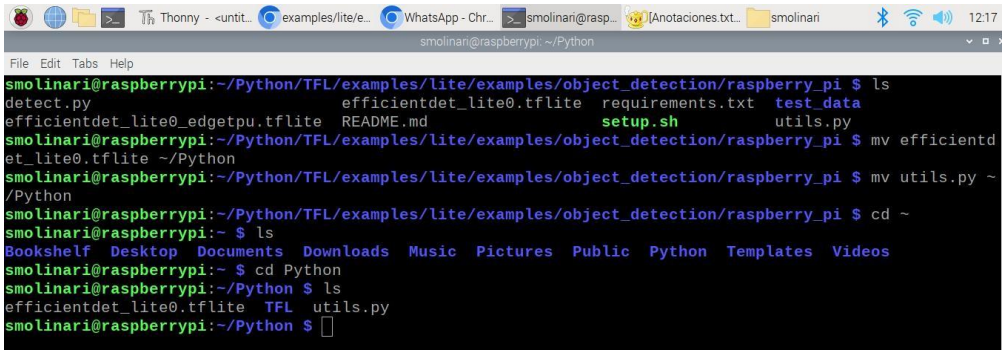


Figura 30. Extracción de archivos necesarios

Al momento de correr el archivo de prueba detect.py aparece un error como se muestra en la figura 31, el cual tiene que ver con las versiones de 2 dependencias de la librería **Tensorflow** las cuales son incompatibles con el resto, la solución para este error fue encontrada en una discusión del foro de Raspberry Pi (fórums Raspberry Pi, 2023).

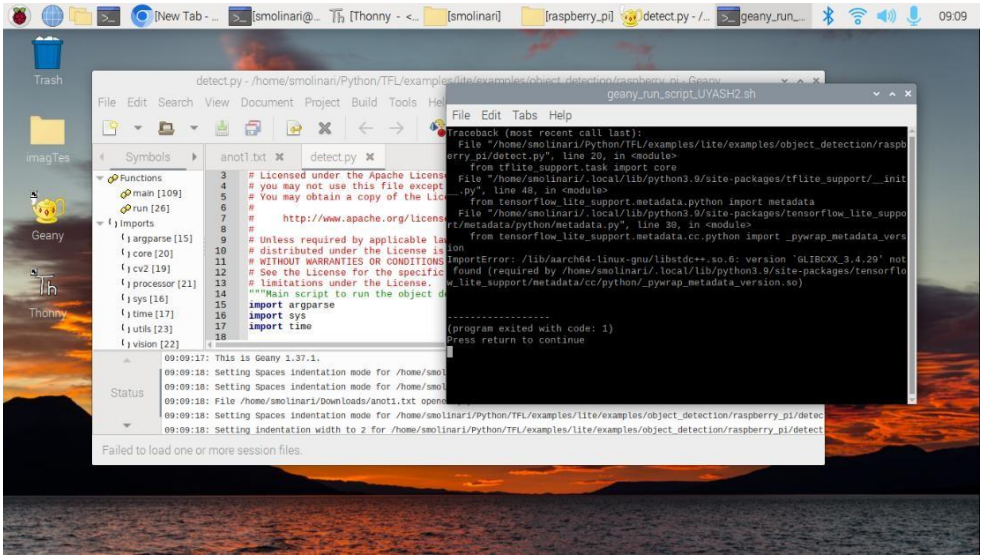


Figura 31 Error al correr detect.py

Para solucionar esta incompatibilidad se instaló la versión 0.4.2 de la dependencia **tflite-support** y la versión 2.11.0 de la dependencia **tflite-runtime**, como se muestra en la figura 32.

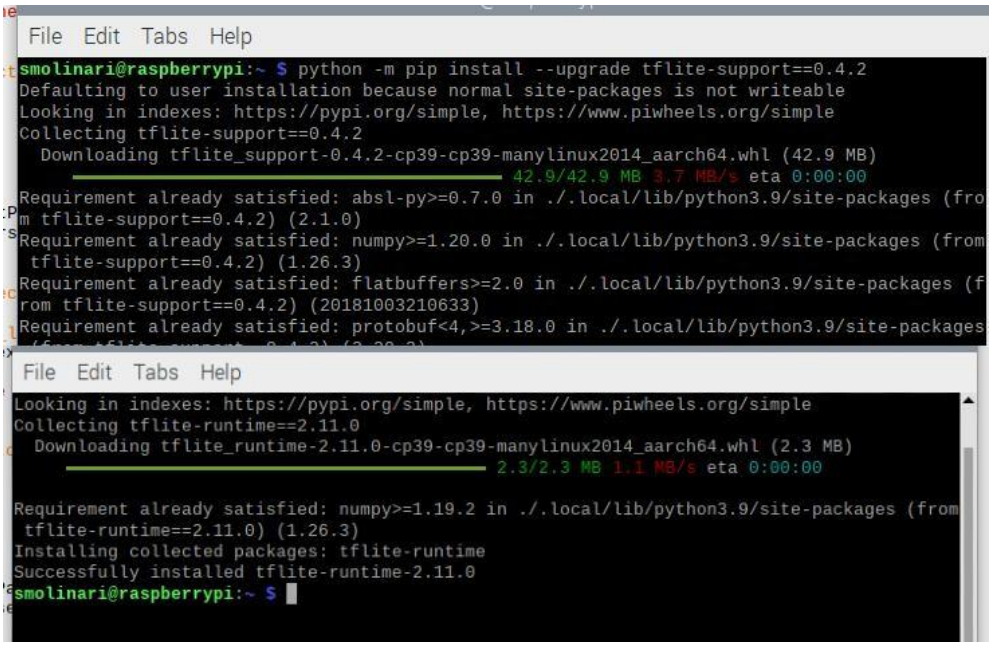


Figura 32. Instalación de las versiones compatibles de las dependencias tflite-support y tflite-runtime

Ya hecho estas correcciones se procede a verificar que funcione el archivo detect.py el cual tiene un código de prueba que corre el modelo **effientdet_lite0.tflite** como se muestra en la figura 33, que a diferencia del modelo de prueba basado en **YOLOv8** que no pasa de 1 **fps**, este se está ejecutando con 5.1 **fps**.

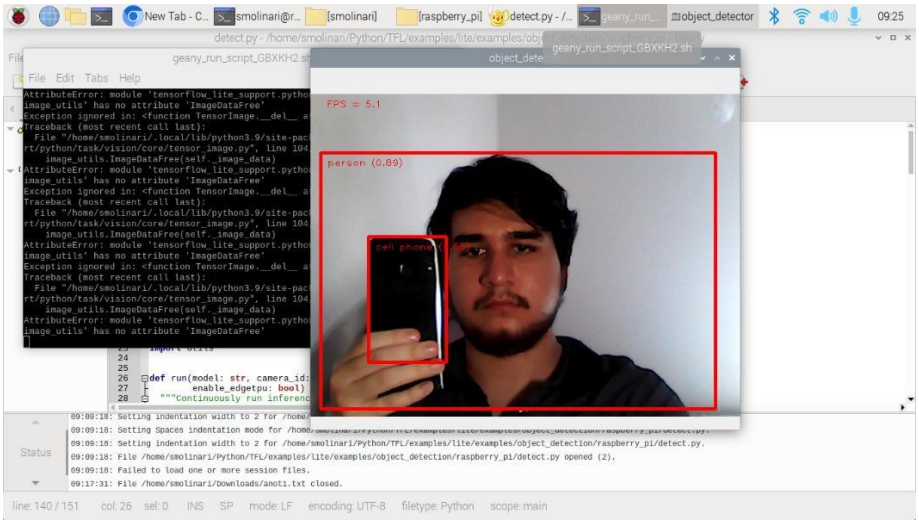


Figura 33. Modelo de detección de objetos effientdet_lite0.tflite

De este archivo extraemos las líneas de código exclusivamente necesarias para el modelo como se muestra en la figura 34, donde primero se importa la librería **OpenCV** para acceder a la cámara web y luego se importan las librerías necesarias para correr modelos de tflite en este caso se está importando los módulos **core**, **processor** y **visión** de la sub-librería **task** de la librería **tflite_support**.

Se indica con que modelo se va a trabajar, en este caso es el modelo de prueba **efficientdet_lite0.tflite** se indica el número de threads o hilos que usara el modelo en este caso 4. Se asignan variables a las diferentes funciones que ajustan los parámetros del modelo:

- **base_options** los parámetros son el nombre del modelo, si usara **coral** el cual es un acelerador de procesamiento para dispositivos embebidos en este caso, no, así que se le asigna false, y el número de hilos ya asignados previamente.

- **detection_options** la cual sus parámetros indican el número de detecciones y el nivel de confianza para realizar una detección.

- **options** la cual sus parámetros son las 2 variables anteriores.

- **detector** la cual su parámetro es la variable **options**.

Luego de asignar los parámetros iniciales se inicia la cámara y se crea un ciclo **while** en el cual almacena la imagen capturada por la función **cam.read()** dentro de la variable **im**.

Para procesar la imagen capturada y asignarle las detecciones primero se la debe convertir de **BGR** a **RGB** ya que la librería de **TensorFlow** trabaja con ese modelo de color, después con la nueva imagen **RGB** se la pasa por la variable imTensor la cual tiene asignada la función que realiza la detección, luego esta variable **imTensor** pasa por la variable **detections** que tiene asignada la función que dibuja la detección, por último se asigna a la variable **image** la función **visualize** la cual muestra el tensor dibujado(**detections**) en las coordenadas donde se encuentra

la detección de la imagen contenida por la variable **im.** Para terminar, se muestra la imagen con la detección en una ventana con la función **cv.imshow**.

Para salir del ciclo **while** se presiona la tecla **ESC**

```python
import cv2 as cv
from tflite_support.task import core
from tflite_support.task import processor
from tflite_support.task import vision
import utils

# Modelo de tensorflow lite
model='efficientdet_lite0.tflite'
num_threads=4

# Ajustes del Modelo
base_options=core.BaseOptions(file_name=model,use_coral=False, num_threads=num_threads)
detection_options=processor.DetectionOptions(max_results=4, score_threshold=.5)
options=vision.ObjectDetectorOptions(base_options=base_options,detection_options=detection_options)
detector=vision.ObjectDetector.create_from_options(options)

#abrir camara
cam = cv.VideoCapture(0)

while True:
    ret, im = cam.read()
    # Inicializar variable de clase
    cigarrillo_detectado = False

    # Procesamiento de la imagen
    imRGB=cv.cvtColor(im,cv.COLOR_BGR2RGB)
    imTensor=vision.TensorImage.create_from_array(imRGB)
    detections=detector.detect(imTensor)
    image=utils.visualize(im, detections)

    #mostrar webcam
    cv.imshow('ObjectDetection', image)
    key = cv.waitKey(1)
    if key == 27:
        break
cv.destroyAllWindows()
```

Figura 34. Código para correr un modelo de tflite con arquitectura efficient_det0

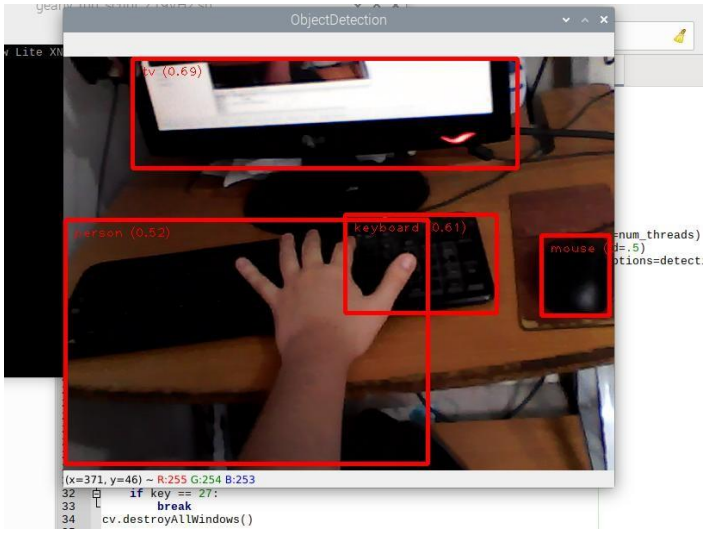


Figura 35. Modelo efficientdet_lite0.tflite en el nuevo código simplificado

4.4.2. Data Set

Para hacer el **data set** que se necesita para entrenar un modelo de **detección de objetos** se debe contar con 2 componentes: 1) la imagen que contiene el escenario y el objeto y 2) la anotación del objeto en algún formato que acepte el método con el que se va a trabajar como se muestra en la figura 36, el escenario o fondo le ayuda a saber el contexto en el que se encuentra al modelo para localizar el objeto por lo que se utilizan todos los elementos de la imagen y no solo el objeto anotado. En este caso se usó las anotaciones en formato **Pascal VOC (Visual Object Classes)** que tiene una estructura XML.

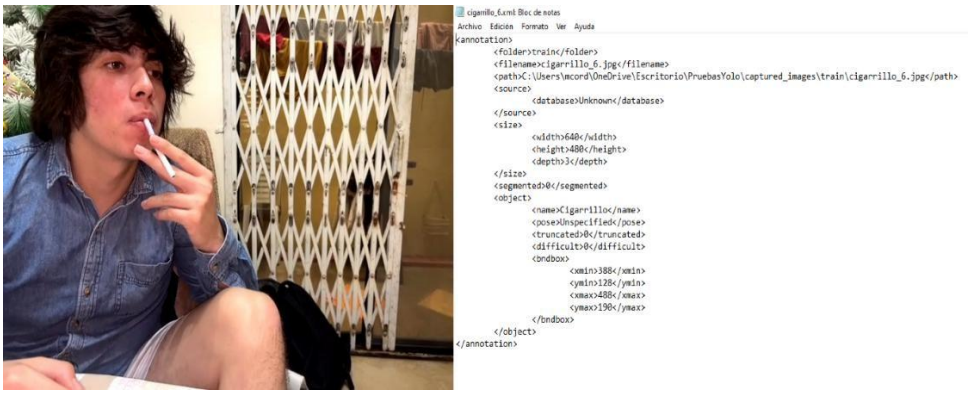


Figure 36. Imágenes y anotaciones

Para realizar las anotaciones se requiere de una herramienta externa que puede ser instalada en Python para ello utilizamos el **IDE PyCharm** para instalar **LabelImg** en forma de librería y realizar las anotaciones de las imágenes del **data set** como se muestra en la figura 37**.**

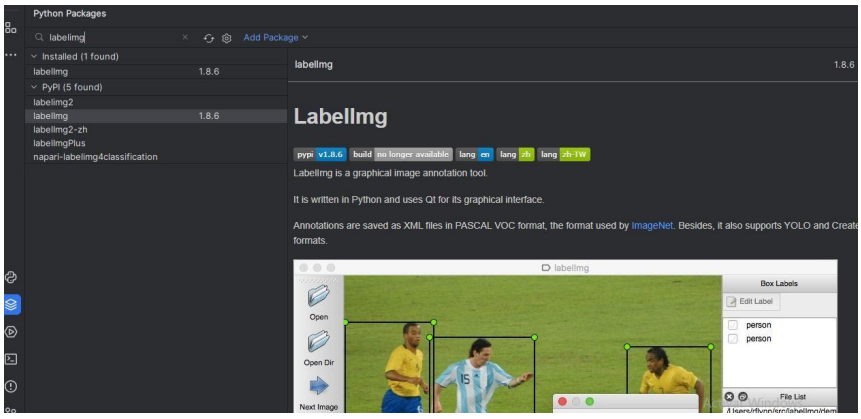


Figura 37. Instalación de la librería LabelImg

Una vez dentro del programa se escogen el directorio de entrada y salida, esto es para iterar sobre las imágenes dentro de una carpeta con un botón de **next** y guardarla una vez se halla realizado la **anotacion**, para realizar la anotación se escoge el formato **Pascal VOC**, se da clic en **Create RecBox** y si selecciona el objeto a anotar en este caso se selecciona el cigarrillo, esto crea una clase que se la llamara Cigarrillo. Para este caso adicionalmente se creará también una clase llamada Otros en la que se anotaran diferentes objetos que se podrían confundir con un cigarrillo como se muestra en la figura 38. Esto debido a que el cigarrillo al ser un objeto con pocas características que lo hagan resaltar puede ser confundido con otros objetos que se sostengan en la mano, como plumas, marcadores, lápices, etc.

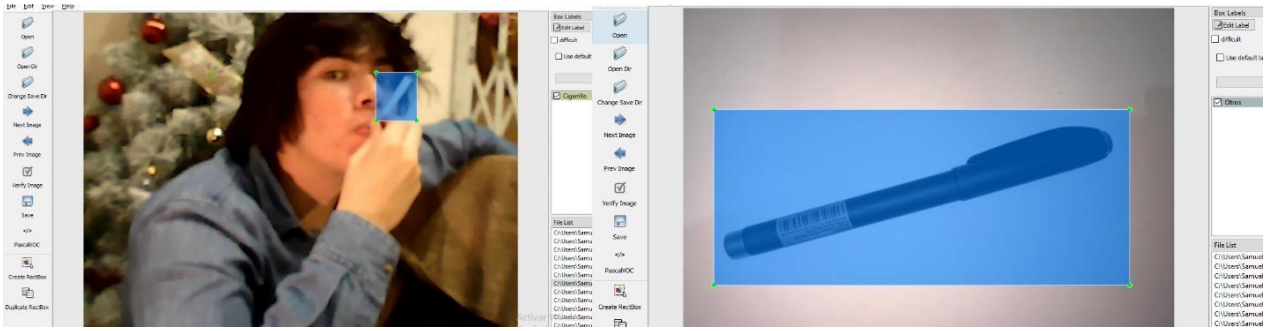


Figura 38. Clase: Cigarrillo y Clase: Otros

Adicional al **data set** creado también se le añadió 2 **data set** ya existentes que se pueden descargar desde la página de **roboflow** en la que tienen una gran variedad de **data sets** creados por la comunidad como se muestra en la figura 39.

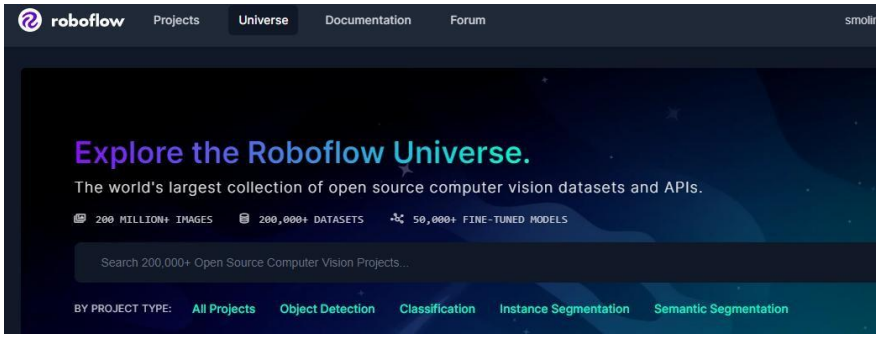


Figura 39. Base de datos de Roboflow (Roboflow universe: Open source computer vision community, s/f)

El entrenamiento se hará tomando 2 carpetas para ello hay que dividir el 70% de nuestras imágenes con sus respectivas anotaciones en una carpeta llamada **train** y el 20% en una carpeta llamada **valid** como se muestra en la figura 40, esto se hace con todos los data set que se vayan a usar.

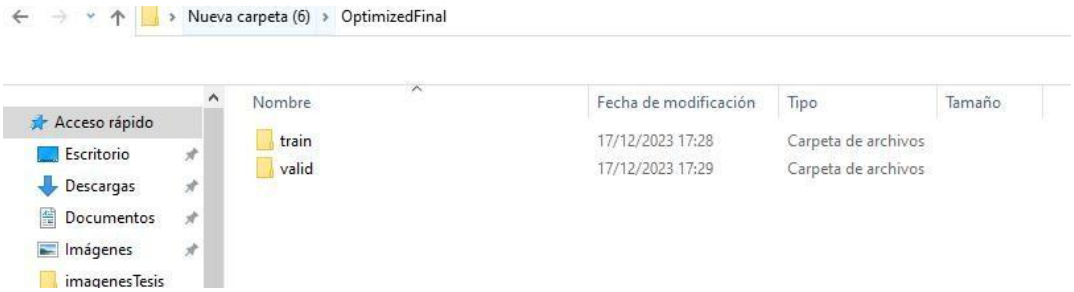


Figura 40. Separar data set en 2 carpetas

Al final se juntan los **data set** en su respectivo orden(juntar las carpetas **train** por un lado y las carpetas **valid** por otro) como se muestra en las figuras 41 a 44. Se comprime el archivo en un .zip y se guarda en una cuenta de Google drive para importarlo en el entrenamiento.

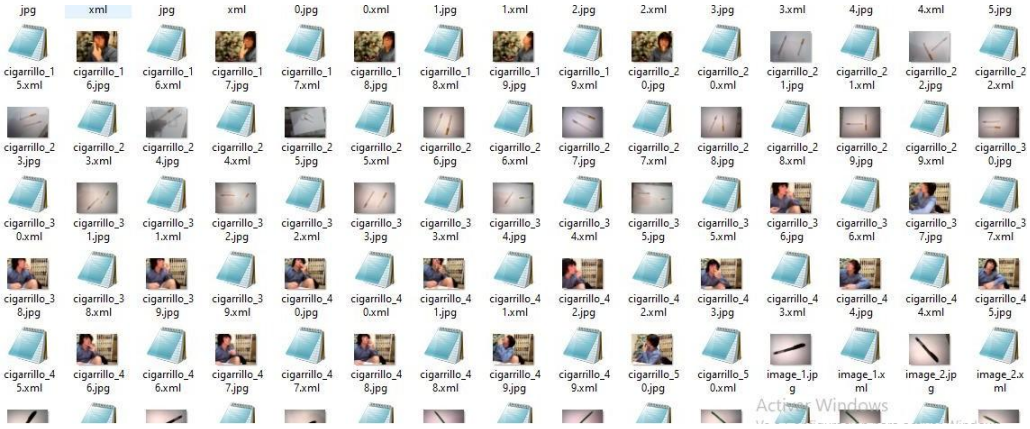


Figura 41. Data set creado con imágenes de gente fumando

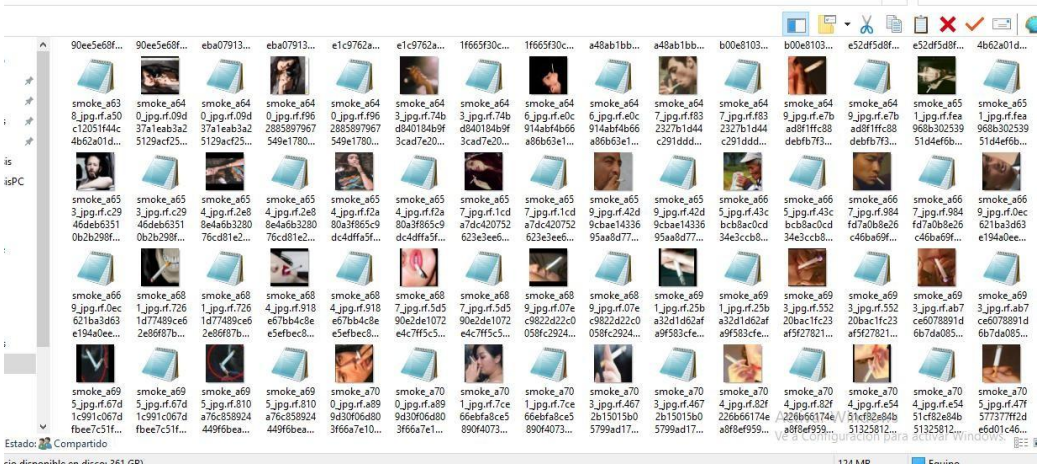


Figura 42. Data set de roboflow con imágenes de gente fumando

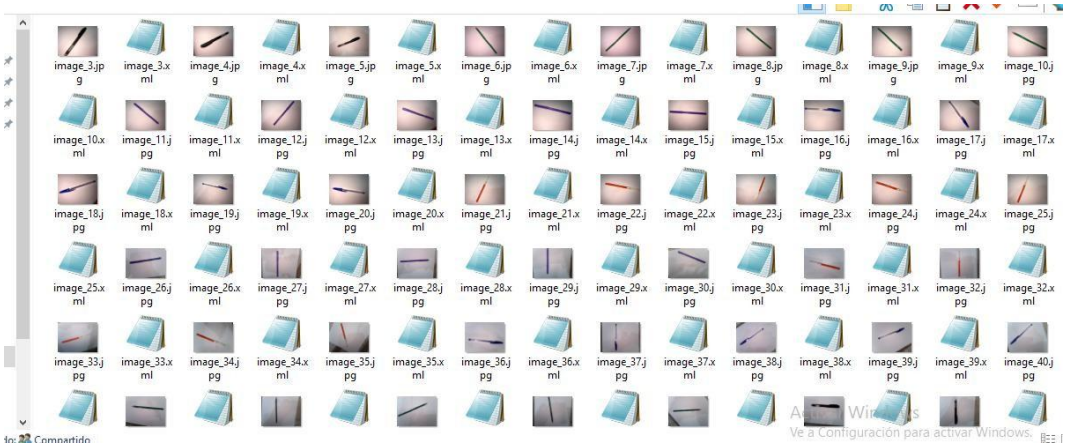


Figura 43. Data set creado con imágenes de objetos similares a un cigarrillo

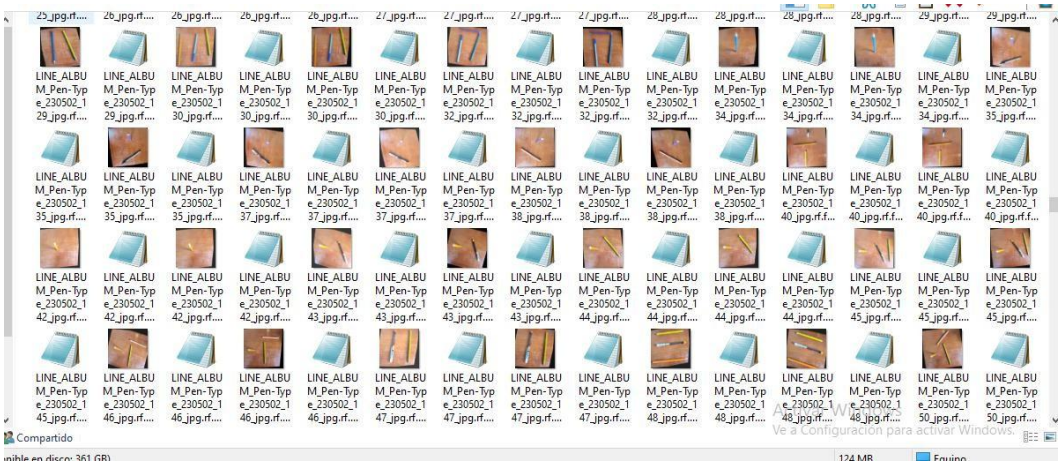


Figura 44. Data set de roboflow con imágenes de objetos similares a un cigarrillo

4.4.3. Entrenar y desplegar modelo customizado para la detección de fumadores

Para entrenar un modelo de detección de objetos con un data set customizado se debe utilizar la herramienta de **TensorFlow** Lite llamada **Model Maker**, en este caso se siguió el ejemplo del GitHub oficial de **TensorFlow Model Maker Object Detection for Android Figurine** que muestra cómo crear un modelo de detección de objetos con la herramienta **Model Maker** este ejemplo cuenta con un Google Colab al cual se puede acceder para cargar un **data set** customizado y empezar el entrenamiento como se muestra en la figura 45. En el momento en el que se desarrolla este proyecto el Google Colab presenta un fallo al descargar las librerías necesarias para correr el creador de modelos, esto debido a que las versiones de las librerías no son compatibles con Python 3.10 el cual es la versión actual de Python en Google Colab.

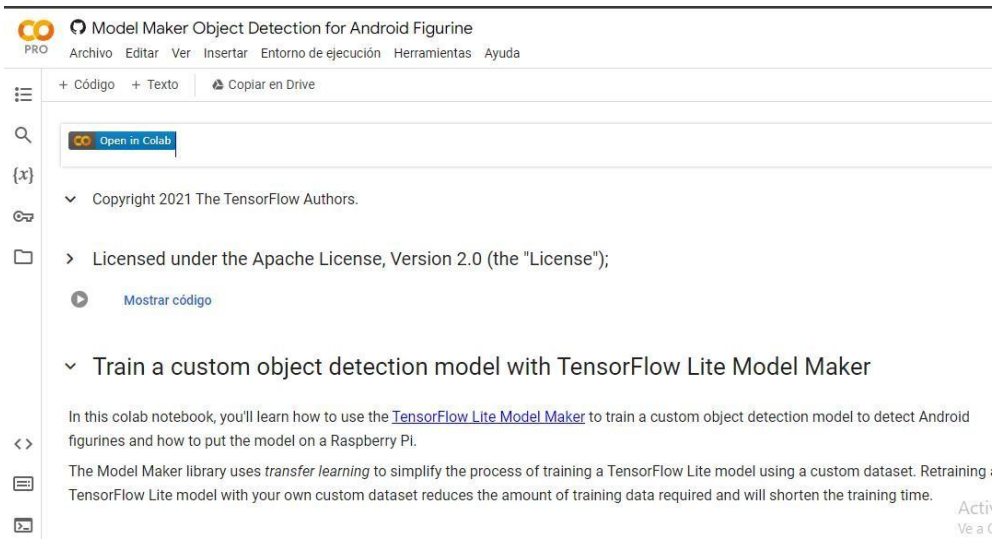


Figura 45. Google Colab oficial del ejemplo de TensorFlow Model Maker Object Detection for Android Figurine (tensorflow, 2023)

Debido a esa incompatibilidad se utilizó una versión modificada de este Google Colab que se encuentra publicado en el foro de **TensorFlow** (TensorFlow Forum, 2023), en la cual se instala el entorno **MiniConda** y dentro de este se instala Python 3.9 el cual si es compatible con las

librerías para crear un modelo usando la herramienta **TensorFlow Lite Model Maker** como se muestra en la figura 46. Para poder instalar las librerías se crea un entorno virtual de Python 3.9.

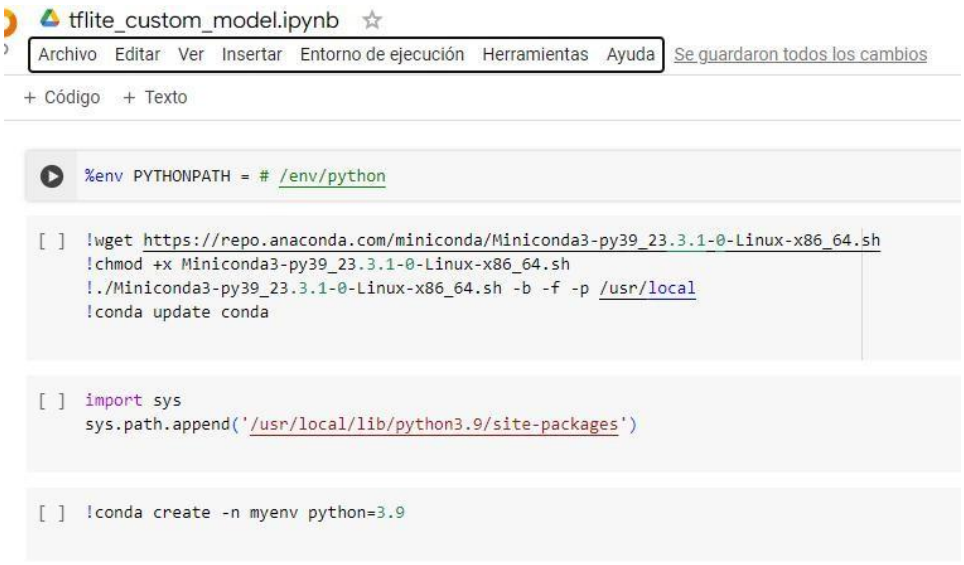


Figura 46. Creación de entorno miniconda e instalación de Python 3.9

Para usar el entorno virtual se lo activa con **activate myvenv**, luego se instalan las librerías con el comando **pip install**, una vez instaladas las librerías se puede adjuntar el data set en este caso se lo descarga del Google drive personal donde se lo tiene en formato .zip por lo que el siguiente paso es descomprimirlo dentro del Google Colab como se muestra en la figura 47.

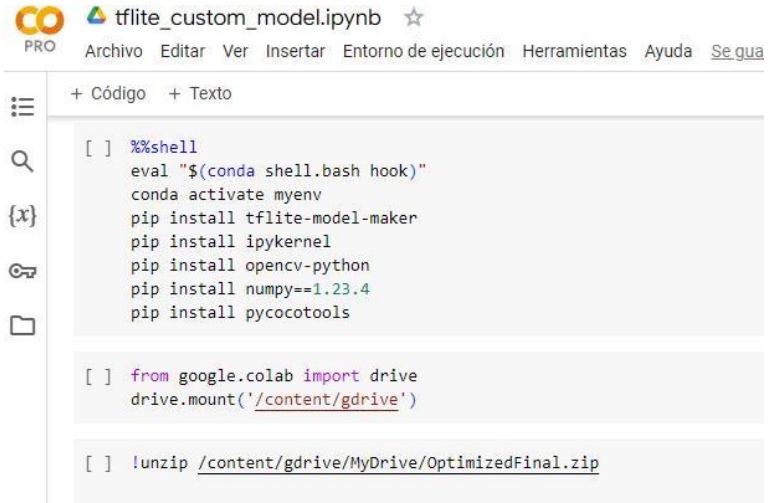


Figura 47. Instalación de librerías y obtener data set de Google drive

Debido a una actualización de Google, el entorno de Colab que se está usando no puede acceder a la optimización de la tarjeta gráfica. Para solucionar este problema se le agrego una celda más de código que instala la librería **nvidia-cuda-toolkit** la cual sirve para establecer la conexión entre el entorno virtual y la tarjeta aceleradora proporcionada por Google Colab como se muestra en la figura 48.

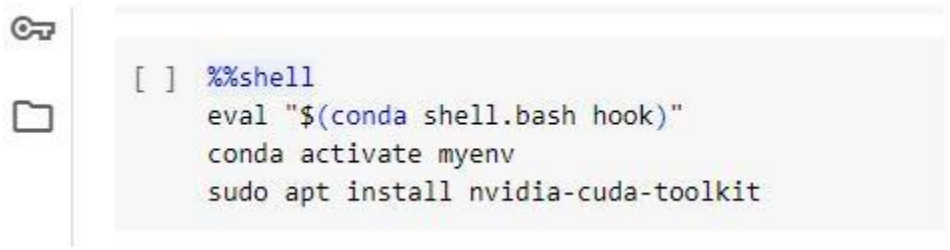


Figura 48. Instalación de la librería nvidia-cuda-toolkit

Por último se procede a entrenar el modelo, para ello hay que descargar un archivo de Python de forma externa dentro del Google Colab este contiene el código necesario para realizar el entrenamiento del modelo de detección de objetos utilizando la arquitectura **EfficientDet_Lite0** así como la ubicación del **data set** que en este caso está dentro de la carpeta "OptimizedFinal" y los parámetros de entrenamiento como se muestra en la figura 49, para **batch_size** se optó por un entrenamiento en **mini-batch**, lo que quiere decir un tamaño de **batch** pequeño a comparación del **data set** para este caso 4, otro parámetro que se puede modificar son el número de **epochs** o épocas, estas se escogen según el rendimiento a base de pruebas con diferentes valores, para este caso se hicieron pruebas con 50, 60 y 100 siendo este último el que mejores resultados dio.

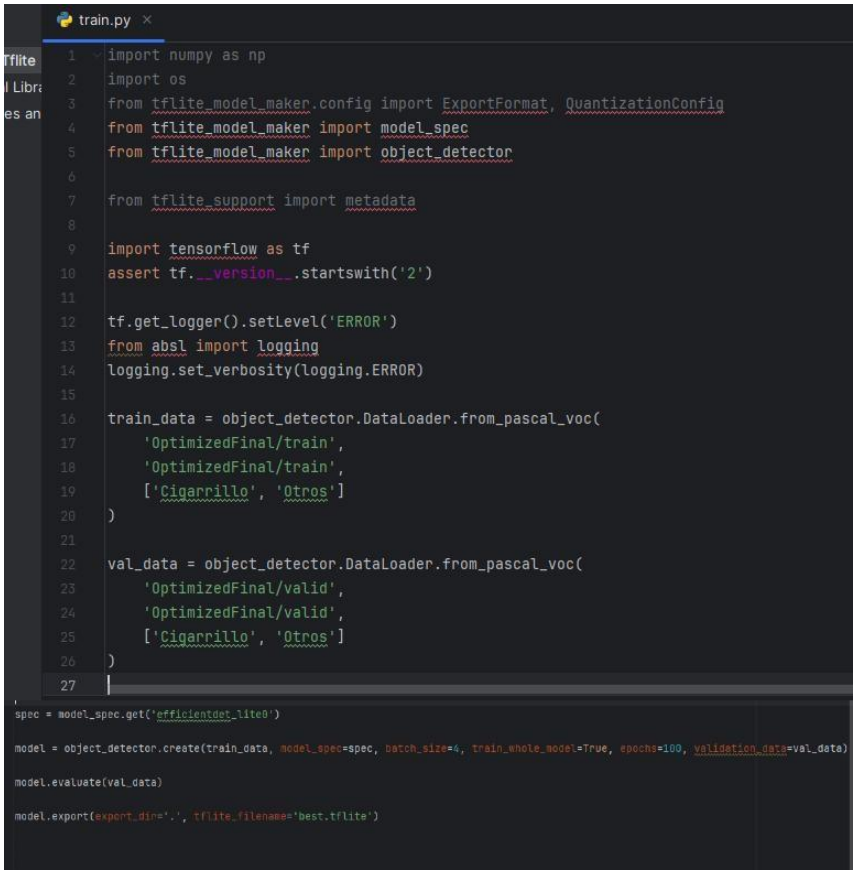


Figura 49. Creación de archivo externo para realizar el entrenamiento

Figura 50. Archivo train.py con los parámetros dentro del Colab

Para empezar el entrenamiento se le da play a cada una de las celdas de código en su respectivo orden como se muestra en la figura 51, y se debe esperar hasta que este termine. Al final un archivo de nombre best.tflite aparecerá en los archivos dentro del Google Colab, este será el modelo entrenado.

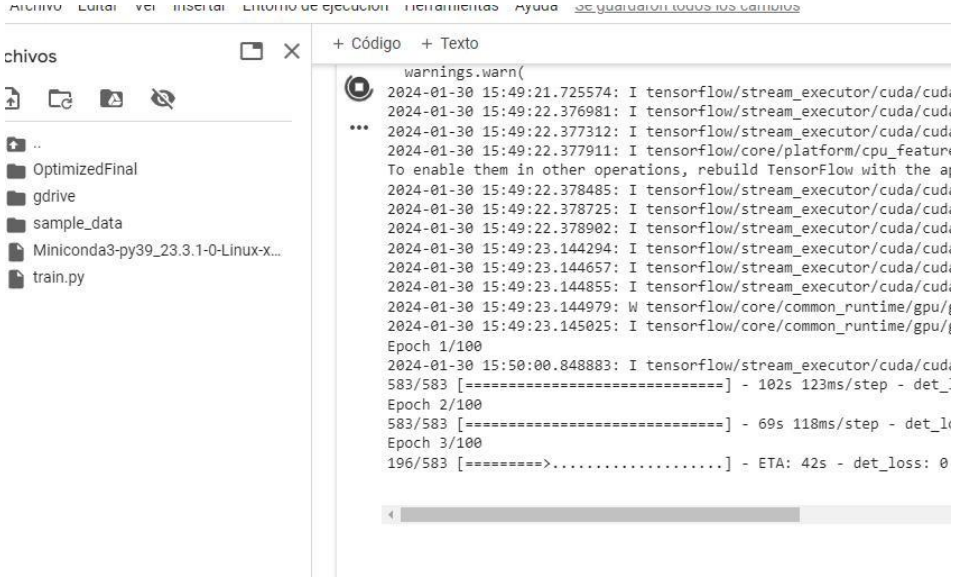


Figura 51. Proceso de entrenamiento del modelo de detección de objetos

4.5. Lógica para realizar la detección de fumadores

Para realizar la función de detectar personas fumando se emplearon los métodos de detección tanto del modelo de detección de objetos customizado hecho con **TensorFlow lite**, como de la librería **Mediapipe** y su función para detectar manos y cara y determinar sus coordenadas dentro de una imagen.

Para comprobar si hay un fumador en la imagen se utilizó la siguiente lógica: si el objeto cigarrilo se encuentra en la imagen entonces se sabrá iterando en los resultados que se almacenan en la variable **detection** dentro de esta tenemos varios elementos que podemos leer entre ellos si la clase: Cigarrillo aparece en la imagen como detección si esto ocurre se le asigna el valor **True** a una variable llamada **cigarrillo_detectado** y se utiliza el mismo proceso si se quiere detectar la

clase: Otros, luego con el comando **if** se asegura si los valores de las distancias calculadas no están vacías ya que sino esto podría hacer que aparezca un error en la próxima línea, entonces entramos a otro **if** que tiene varias condiciones:

1) detecta cuando se cumple que la distancia de los **landmarks** 3 de cara al 8 de la mano sea menor a 125
2) detecta cuando se cumple que la distancia de los **landmarks** 4 y 8 de la mano son menores a 40 **o** que la variable **condition** sea verdadera que indica cuando los **landmarks** 8 y 12 están más arriba que los demás landmarks
3) la variable **cigarrillo_detectado** sea verdadera

cuando se cumplan estas condiciones imprimirá las coordenadas de las variables de cada detección y ejecutara las funciones de salida como se muestra en la figura 52.

optimizado.py

```
210
211     # Comprobar si la clase "Cigarrillo" esta siendo detectada
212     #print(detections)
213     for detection in detections.detections:
214         for category in detection.categories:
215             if category.category_name == 'Cigarrillo':
216                 cigarrillo_detectado = True
217             #if category.category_name == 'Otros':
218                 #print("No es un Cigarrillo")
219     #print(point_3_face)
220     #print(point_4_hand)
221     #print(point_8_hand)
222     #print(distance_3_8)
223     #print(distance_4_8)
224     #print(condition)
225     #print(cigarrillo_detectado)
226     if distance_3_8 is not None and distance_4_8 is not None:
227         if distance_3_8 < 125 and (distance_4_8 < 40 or condition) and cigarrillo_detectado:
228             captura(img_proces)
229             #enviar_correo(image)
230             print(f"distancia de la boca a la punta del dedo indice: {distance_3_8}")
231             print(f"distancia entre las puntas de los dedos pulgar e indice: {distance_4_8}")
232             print(f"los dedos indice y medio estan levantados: {condition}")
233             print(f"Cigarrillo detectado: {cigarrillo_detectado}")
234             #smokerDetected()
235             cerrar_camara(cam)
236             abrir_camara()
237
238     cv.imshow('ObjectDetection+Mediapipe', img_proces)
239     key = cv.waitKey(1)
240     if key == 27:
241         break
242
243 cv.destroyAllWindows()
```

Figura 52. Lógica de detección

4.6. Funciones del prototipo al detectar un fumador

Para ejecutar estas funciones primero se debe importar las librerías requeridas como se indica en la figura 53.

```
import cv2 as cv
import simpleaudio as sa
import mediapipe as mp
import os
import time
import smtplib
from acceso import email_pass
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipa
from email.mime.image import MIMEImage
from tflite_support.task import core
from tflite_support.task import processor
from tflite_support.task import vision
import utils
import PySimpleGUI as sg
```

Figura 53. Librerías utilizadas en el código final

4.6.1. Audio de salida

Para instalar la librería **simpleaudio** se utilizó la herramienta de administración de paquetes del **IDE Thonny Python** como se muestra en la figura 54. En la que se instaló la versión actual de la librería.

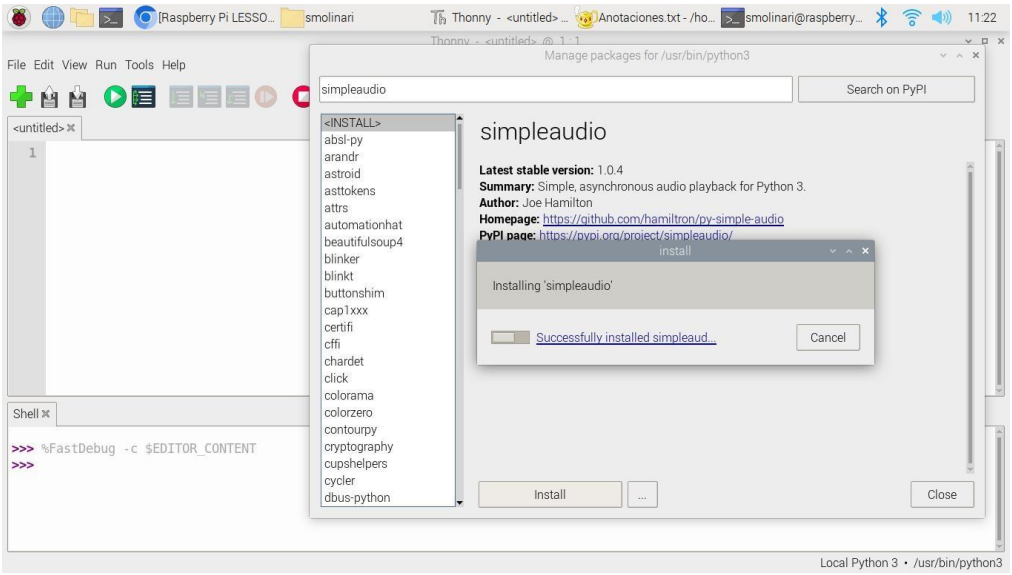


Figura 54. Instalación de la librería simpleaudio

La librería necesita la dependencia **libasaund2-dev** por lo que se la instalara mediante la ventana de comandos como se muestra en la figura 55, con el comando **sudo apt-get install -y python3-dev libasound2-dev**.

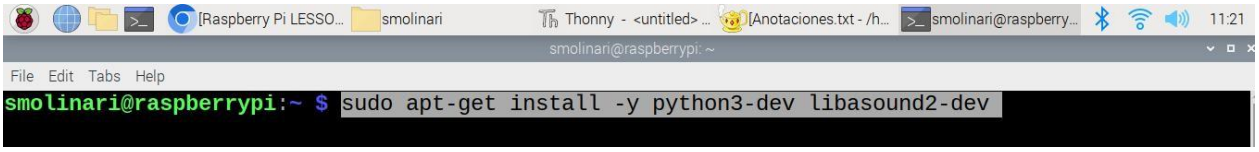


Figura 55. Instalación de dependencia libasound2-dev

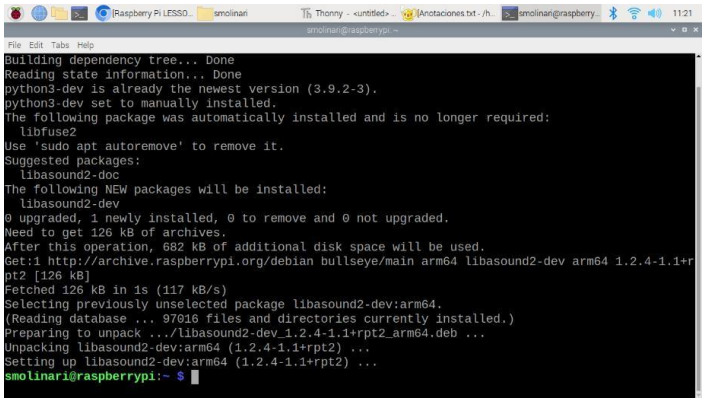


Figura 56. Instalación completa de la dependencia libasound2-dev

Para reproducir un audio de salida utilizando la librería **simpleaudio** se crea una función que cree un objeto con el valor de un audio en formato **wav**, en este caso se está usando un audio generado por computador que dice “les recordamos a los clientes que está prohibido fumar dentro del establecimiento les agradecemos por su comprensión”, luego se corre el audio y se espera hasta que este haya terminado como se muestra en la figura 57. Esto crea un **delay** en el código, pero sirve para que la persona que está cometiendo la infracción tenga tiempo de abandonar el área restringida.

```
# pista de audio de salida
def smokerDetected():
    wave_obj = sa.WaveObject.from_wave_file("fumadorDetectado.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()
```

Figura 57. Función de audio de salida

4.6.2. Almacenar el fotograma de la detección dentro de una carpeta local

Para almacenar el fotograma de la detección en una carpeta local se creó una función la cual utiliza la librería **os** para manejar directorios. El lugar donde se almacenarán será una carpeta llamada “capturas” si no existe se creará automáticamente y se guardara la captura de la imagen con un nombre generado por el año, el día, la hora, el minuto, y el segundo en el que se toma la captura esto se hace con la librería **time** para que no haya nombres repetidos, luego se toma el directorio y el nombre y se guarda la detección con el comando **cv.imwrite** como se muestra en la figura 58.

```
# Capturar localmente una imagen
def captura(im):
    # Crear una carpeta para almacenar capturas si no existe
    folder_name = "capturas"
    if not os.path.exists(folder_name):
        os.makedirs(folder_name)

    # Generar un nombre único para la captura
    capture_name = f"captura_{time.strftime('%Y%m%d%H%M%S')}.png"

    # Ruta completa para guardar la captura en la carpeta
    capture_path = os.path.join(folder_name, capture_name)

    # Guardar la captura en la carpeta especificada
    cv.imwrite(capture_path, im)
```

Figura 58. Función para capturar localmente un fotograma

4.6.3. Enviar el fotograma de la detección por correo electrónico

Para enviar un mensaje por correo electrónico se debe asignar el correo emisor, el receptor, la contraseña del emisor, el tipo de servidor en este caso se enviará desde un Outlook por lo que el servidor es **smtp-mail.outlook.com** y el puerto que para este caso es 587. Luego se configura la estructura del mensaje del correo a enviar, se adjunta el asunto: “detección nueva”, el cuerpo del mensaje: “¡Fumador detectado!” más el nombre de la detección que en este caso se utiliza la librería **time** para asignarle un nombre formado por la fecha y la hora, y se adjunta este texto a la estructura del mensaje. Para la siguiente parte del mensaje se convierte el fotograma de la detección a un valor en bytes con el nombre formado por año, mes, día, hora, minuto y segundo con el formato .png, y se lo adjunta a la estructura del mensaje.

Se inicia la conexión con el servidor, se inicia sesión en el correo emisor y se envía el mensaje al receptor tal como se detalla en la figura 59.

```python
# Enviar correo de imagen actual
def enviar_correo(captura):
    # Configurar los detalles del correo electrónico
    sender_email = "smolinari@est.ups.edu.ec"
    receiver_email = "mvelizc@est.ups.edu.ec"
    password = email_pass

    # Configurar el servidor SMTP
    smtp_server = "smtp-mail.outlook.com"
    smtp_port = 587

    # Configurar el mensaje del correo
    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = "Deteccion Nueva"

    # Adjuntar el texto del mensaje
    body = f"¡Fumador Detectado! {time.strftime('Fecha: %d/%m/%Y Hora: %H:%M:%S')}"
    text = MIMEText(body, "plain")
    message.attach(text)

    # Adjuntar la imagen al mensaje
    imgToBytes = cv.imencode('.png', captura)[1].tobytes()
    image = MIMEImage(imgToBytes, name=f"captura_{time.strftime('%Y%m%d%H%M%S')}.png")
    message.attach(image)

    # Iniciar la conexión con el servidor SMTP y enviar el correo
    with smtplib.SMTP(smtp_server, smtp_port) as server:
        server.starttls()
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, message.as_string())
    print("¡Fumador detectado!")
```

Figura 59. Función para enviar el fotograma de la detección por correo electronico

4.7. Interfaz gráfica del prototipo

Para hacer la interfaz gráfica se utilizó la librería PySimpleGUI.

Para instalar la librería **PySimpleGUI** se utilizó la herramienta de administración de paquetes del **IDE Thonny Python** como se muestra en la figura 60.

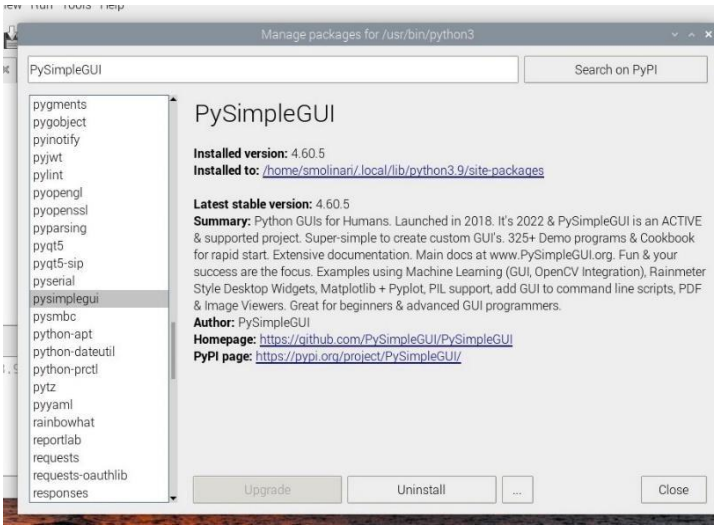


Figura 60. Instalación de PySimpleGUI

La interfaz gráfica se realizó añadiendo botones e inputs de texto para cambiar los tipos de detección que se pueden hacer con el prototipo y modificar sus parámetros los parámetros iniciales de los modelos y sus características, también cuenta con un apartado para visualizar las detecciones como se puede apreciar en las figuras 61 a 65.

```
sg.theme('DarkTanBlue')

boton1 = True
boton2 = True
boton3 = True
boton4 = True
boton5 = True
impresiones = None
```

Figura 61. Asignar valores iniciales a botones y variables

```
window_size = (1185, 675)

layout_l = [[sg.Image(filename='logo_ups_resized.png'), sg.Frame('Presentacion',[[sg.Text('Elaborado por:\nSamuel Molinari, Ac
[sg.Text('Titulo: DISEÑO E IMPLEMENTACION\nDE UN PROTOTIPO PARA LA\nDETECCION DE FUMADORES\nEN AREAS RESTRINGIDAS')]])],
[sg.Image(filename='', key='-IMAGE-')]]

layout_r = [[sg.Frame('Detecciones',
[[sg.TabGroup([[sg.Tab('Tiempo real',[[sg.Text('punto3_Cara:'), sg.Text(key='-OUTPUT1-')],
[sg.Text('punto4_Mano:'), sg.Text(key='-OUTPUT2-')],
[sg.Text('punto8_Mano:'), sg.Text(key='-OUTPUT3-')],
[sg.Text('distancia entre las puntas de los dedos pulgar e indice:'), sg.Text(key='-OUTPUT4-'), sg.T(s=10)],
[sg.Text('distancia de la boca a la punta del dedo indice:'), sg.Text(key='-OUTPUT5-')],
[sg.Text('los dedos indice y medio estan levantados:'), sg.Text(key='-OUTPUT6-')],
[sg.Text('clase Cigarrillo detectada:'), sg.Text(key='-OUTPUT7-')],
[sg.Text('clase Otros detectada:'), sg.Text(key='-OUTPUT8-')]]), sg.Tab('Ultima deteccion', [[sg.Output(key='-Salidas-', s=(55
[sg.Frame('Parametros',
[[sg.Frame('Mediapipe Cara',
[[sg.Text('Modelo deteccion cara (0 or 1)          '), sg.Text(key='-OUT1-'), sg.Text('To'), sg.In(key='-INPUT1-')],
[sg.Text('Confianza de deteccion (0.0, 1.0)    '), sg.Text(key='-OUT2-'), sg.Text('To'), sg.In(key='-INPUT2-')]])],
[sg.Frame('Mediapipe Manos',
[[sg.Text('Modelo deteccion manos (0 or 1)       '), sg.Text(key='-OUT3-'), sg.Text('To'), sg.In(key='-INPUT3-')],
[sg.Text('Numero de manos a detectar             '), sg.Text(key='-OUT4-'), sg.Text('To'), sg.In(key='-INPUT4-')],
[sg.Text('Confianza de deteccion (0.0, 1.0)    '), sg.Text(key='-OUT5-'), sg.Text('To'), sg.In(key='-INPUT5-')],
[sg.Text('Confianza de seguimiento (0.0, 1.0) '), sg.Text(key='-OUT6-'), sg.Text('To'), sg.In(key='-INPUT6-')]])],
[sg.Frame('Tflite modelo custom',
[[sg.Text('Detecciones maximas modelo Tflite   '), sg.Text(key='-OUT7-'), sg.Text('To'), sg.In(key='-INPUT7-')],
[sg.Text('Confianza de deteccion (0.0, 1.0)    '), sg.Text(key='-OUT8-'), sg.Text('To'), sg.In(key='-INPUT8-')]])],
[sg.Push(), sg.Button('Submit'), sg.Push()]])],
[sg.Frame('Opciones de respuesta',
[[sg.Text('Captura local  '), sg.Button('On', size=(3, 1), button_color='white on green', key='-B1-'), sg.VSep(),
sg.Text('Enviar correo'), sg.Button('On', size=(3, 1), button_color='white on green', key='-B2-'), sg.VSep(),
sg.Text('Mediapipe'), sg.Button('On', size=(3, 1), button_color='white on green', key='-B3-')],
[sg.Text('Audio de aviso'), sg.Button('On', size=(3, 1), button_color='white on green', key='-B4-'), sg.VSep(),
sg.Push(), sg.Button('Exit',s=(7,1)), sg.Push(), sg.VSep(),
sg.Text('      tflite'), sg.Button('On', size=(3, 1), button_color='white on green', key='-B5-')]])]]
```

Figura 62. Crear los inputs de texto y botones para cambiar parámetros y funciones

```
    # Procesamiento de la imagen
    ret, im = cam.read()
    if boton3 and boton5:
        imRGB = cv.cvtColor(im,cv.COLOR_BGR2RGB)
        imTensor = vision.TensorImage.create_from_array(imRGB)
        detections = detector.detect(imTensor)
        image = utils.visualize(im, detections)
        img_proces = detect_and_draw(image)
    elif boton3 and not boton5:
        img_proces = detect_and_draw(im)
    elif boton5 and not boton3:
        imRGB = cv.cvtColor(im,cv.COLOR_BGR2RGB)
        imTensor = vision.TensorImage.create_from_array(imRGB)
        detections = detector.detect(imTensor)
        img_proces = utils.visualize(im, detections)
    elif not boton3 and not boton5:
        img_proces = im
```

Figura 63. Acciones según los botones

```
        try:
            max_hands = int(values['-INPUT4-'])
        except ValueError:
                pass
    try:
        min_detection_confidence_hands = float(values['-INPUT5-'])
    except ValueError:
            pass
    try:
        track_hands = float(values['-INPUT6-'])
    except ValueError:
            pass
    try:
        max_tflite = int(values['-INPUT7-'])
    except ValueError:
            pass
    try:
        conf_tflite = float(values['-INPUT8-'])
    except ValueError:
            pass

    mp_face = mp_face_detection.FaceDetection(model_selection=model_selection_face,
              min_detection_confidence=min_detection_confidence_face)

    hands = mp_hands.Hands(model_complexity=model_selection_hands,
            max_num_hands=max_hands,
            min_detection_confidence=min_detection_confidence_hands,
            min_tracking_confidence=track_hands)
```

Figura 64. Cambio de parámetros mediante los inputs de texto

```
elif event == '-B4-':
        boton4 = not boton4

elif event == '-B5-':
        boton5 = not boton5

if ret:
    imgbytes = cv.imencode('.png', img_proces)[1].tobytes()
    window['-IMAGE-'].update(data=imgbytes)
    window['-OUTPUT1-'].update(str(point_3_face))
    window['-OUTPUT2-'].update(str(point_4_hand))
    window['-OUTPUT3-'].update(str(point_8_hand))
    window['-OUTPUT4-'].update(str(distance_4_8)[:5])
    window['-OUTPUT5-'].update(str(distance_3_8)[:5])
    window['-OUTPUT6-'].update(condition)
    window['-OUTPUT7-'].update(cigarrillo_detectado)
    window['-OUTPUT8-'].update(Otros_detectado)
    window['-OUT1-'].update(model_selection_face)
    window['-OUT2-'].update(min_detection_confidence_face)
    window['-OUT3-'].update(model_selection_hands)
    window['-OUT4-'].update(max_hands)
    window['-OUT5-'].update(min_detection_confidence_hands)
    window['-OUT6-'].update(track_hands)
    window['-OUT7-'].update(max_tflite)
    window['-OUT8-'].update(conf_tflite)
    window['-B1-'].update(text='On' if boton1 else 'Off', button_color='white on green' if boton1 else 'white on red')
    window['-B2-'].update(text='On' if boton2 else 'Off', button_color='white on green' if boton2 else 'white on red')
    window['-B3-'].update(text='On' if boton3 else 'Off', button_color='white on green' if boton3 else 'white on red')
    window['-B4-'].update(text='On' if boton4 else 'Off', button_color='white on green' if boton4 else 'white on red')
    window['-B5-'].update(text='On' if boton5 else 'Off', button_color='white on green' if boton5 else 'white on red')
    window['-Salidas-'].update(impresiones)

window.close()
```

Figura 65. Actualización de los valores del interfaz gráfico y visualización de las detecciones

5. RESULTADOS

5.1. Resultados entrenamiento

En la figura 66 se puede apreciar que el entrenamiento culmino y se ha generado un archivo llamado **best.tfilite** este contiene el modelo de TensorFlow Lite customizado con las 2 clases: Cigarrillo y Otros.

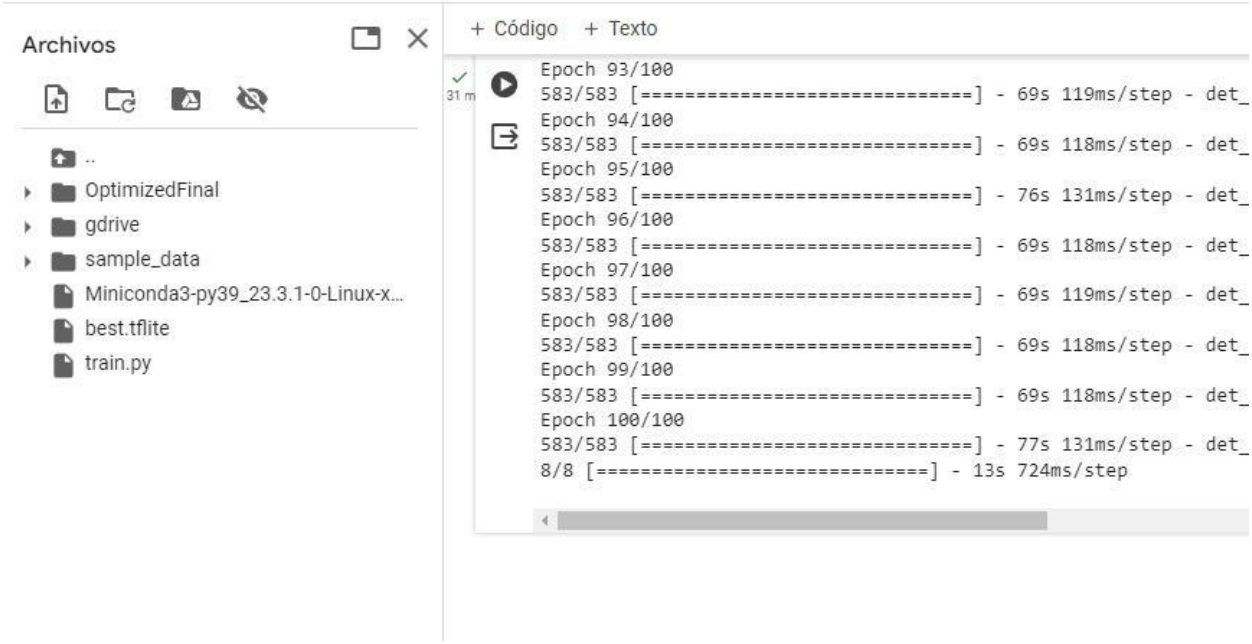


Figura 66. resultados de entrenamiento

Parámetros de las primeras 6 épocas de entrenamiento vs ultimas 6 épocas del entrenamiento que dejan ver la mejoría del modelo en el tiempo.

Epoch	Det Loss	Cls Loss	Box Loss	Reg L2 Loss	Total Loss	Learning Rate
Primeros 5 pasos						
1	1,1755	0,6153	0,0099	0,0639	1,1755	0,0065
2	0,8809	0,4481	0,0074	0,065	0,8809	0,005
3	0,813	0,4096	0,0068	0,0657	0,813	0,005
4	0,7764	0,3954	0,0063	0,0663	0,7764	0,005
5	0,7433	0,3791	0,0059	0,0669	0,7433	0,005
Últimos 5 pasos						
96	0.3322	0.1967	0.0027	0.0753	0.4075	0,15508
97	0.3432	0.2012	0.0028	0.0753	0.4184	0,07968
98	0.3234	0.1932	0.0026	0.0753	0.3986	0,029368
99	0.3266	0.1940	0.0027	0.0753	0.4019	0,0041992
100	0.3421	0.2000	0.0028	0.0753	0.4173	0,0041992

Gradient Norm	Val Det Loss	Val Cls Loss	Val Box Loss	Val Reg L2 Loss	Val Total Loss
Primeros 5 pasos					
5,6883	0,7644	0,3743	0,0065	0,0646	0,7644
5,4785	1,0959	0,6872	0,0069	0,0654	1,0959
5,117	0,7135	0,4285	0,0044	0,066	0,7135
4,8741	0,6576	0,34	0,005	0,0666	0,6576
4,7338	0,6555	0,3257	0,0053	0,0671	0,6555
Últimos 5 pasos					
37703	0.5709	0.3012	0.0054	0.0753	0.6461
38195	0.5836	0.3059	0.0056	0.0753	0.6589
36521	0.5891	0.3076	0.0056	0.0753	0.6643
37391	0.5776	0.3030	0.0055	0.0753	0.6529
38432	0.5964	0.3135	0.0057	0.0753	0.6717

Tabla 1. Parámetros de entrenamiento del modelo de tensorflow lite

Como se puede ver en la tabla 1 entre todos los parámetros podemos destacar el parámetro **loss** el cual indica la perdida general del modelo, en este caso se puede apreciar que este parámetro va disminuyendo lo que significa una mejora del modelo que va minimizando esta pérdida cada época que se entrena.

5.2. Resultados tensorflow

En la figura 67 podemos ver como el modelo está detectando correctamente la clase Cigarrillo en tiempo real.

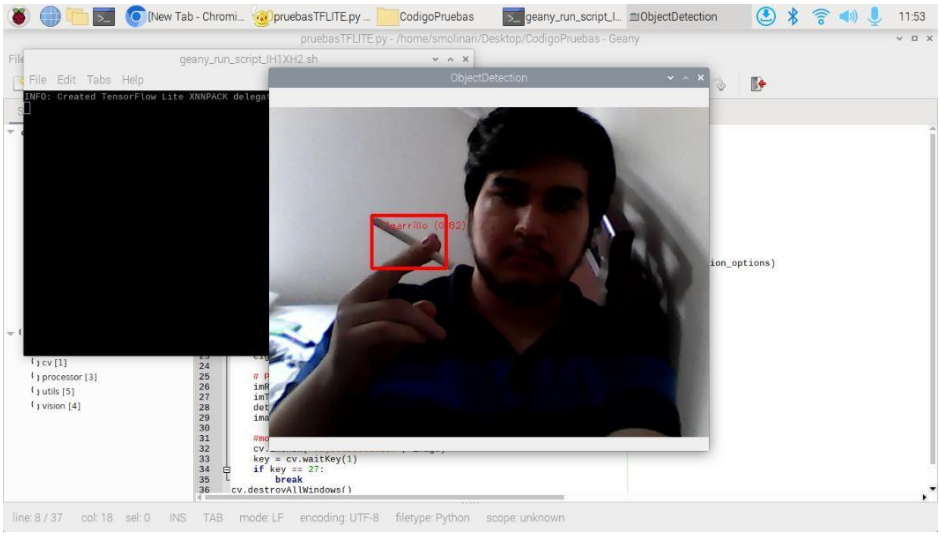


Figura 67. Resultados tensorflow clase Cigarrillo

En la figura 68 podemos ver como el modelo está detectando correctamente la clase Otros en tiempo real.

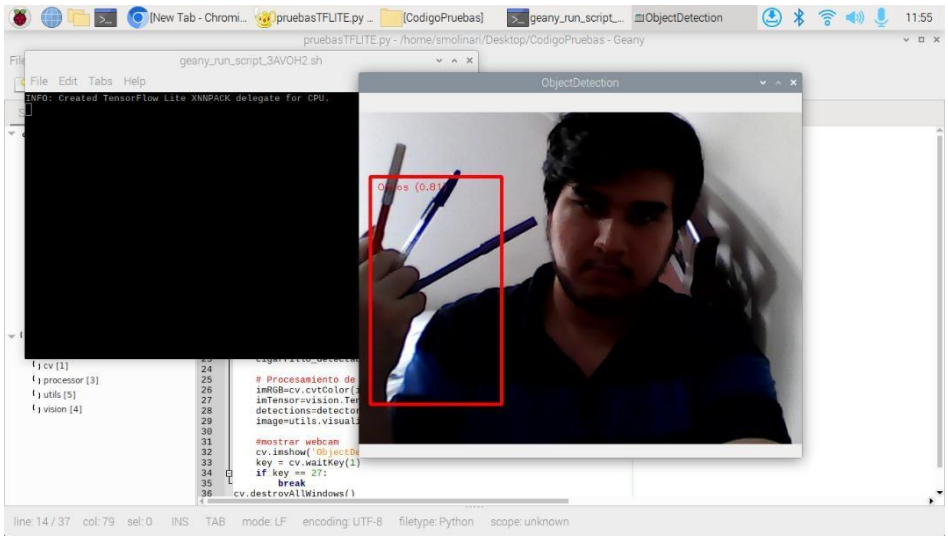


Figura 68. Resultados tensorflow clase Otros

5.3. Resultados Mediapipe

Al ejecutar el código en el que se programó las funciones de mediapipe se puede observar las detecciones de los modelos para el reconocimiento de cara y manos, se dibujan los landmarks que corresponden a los puntos de interés de los modelos y se muestran los valores de los puntos de interés guardados 3 para la cara, 4 y 8 para la mano, también se muestra el estado de la condición que indica si los dedos índice y medio están levantados y los valores de los cálculos de la distancia euclidiana.

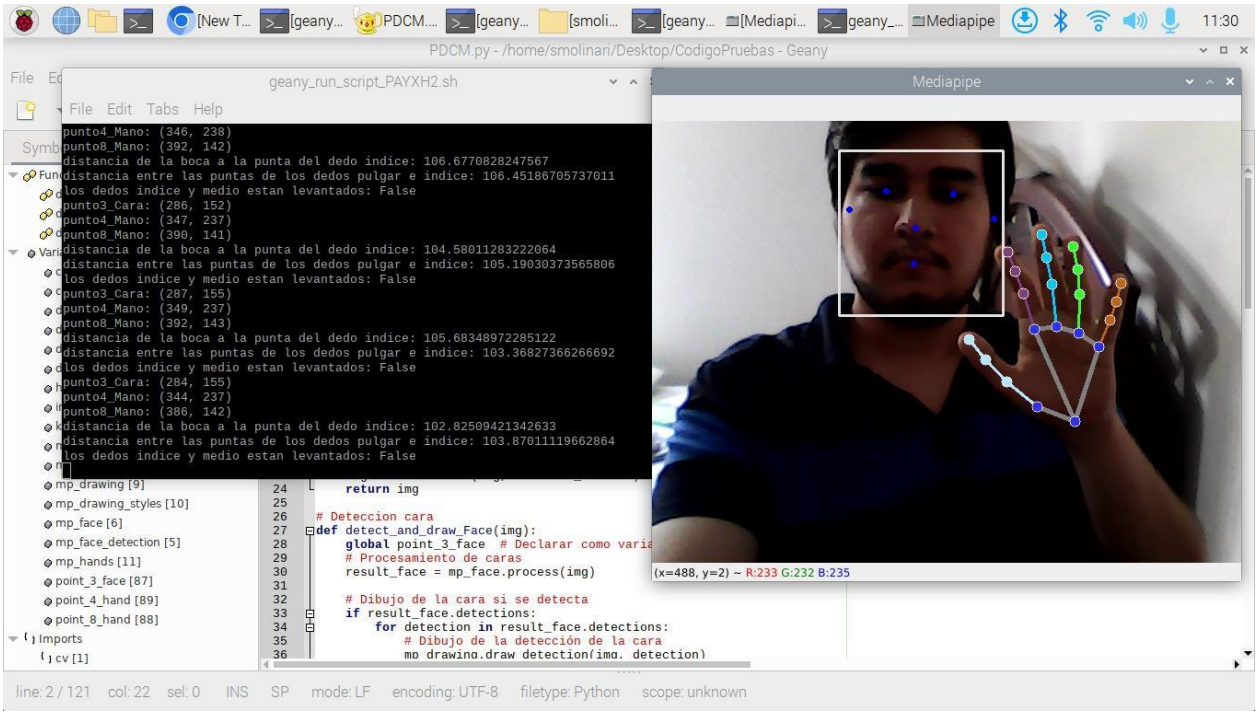


Figura 69. Resultados mediapipe

5.4. Resultados funciones de salida

Como se puede observar en la figura 70 el código tomo una captura del fotograma de la detección y la guardo en una carpeta local.

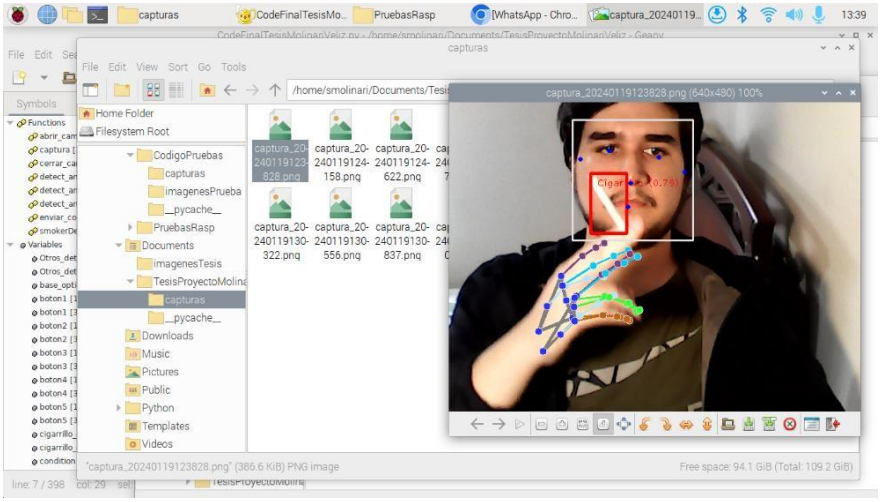


Figura 70. Resultados función para guardar las detecciones en carpeta local

Como se puede apreciar en la figura 71 el código mando por correo electrónico el fotograma de la detección capturado.

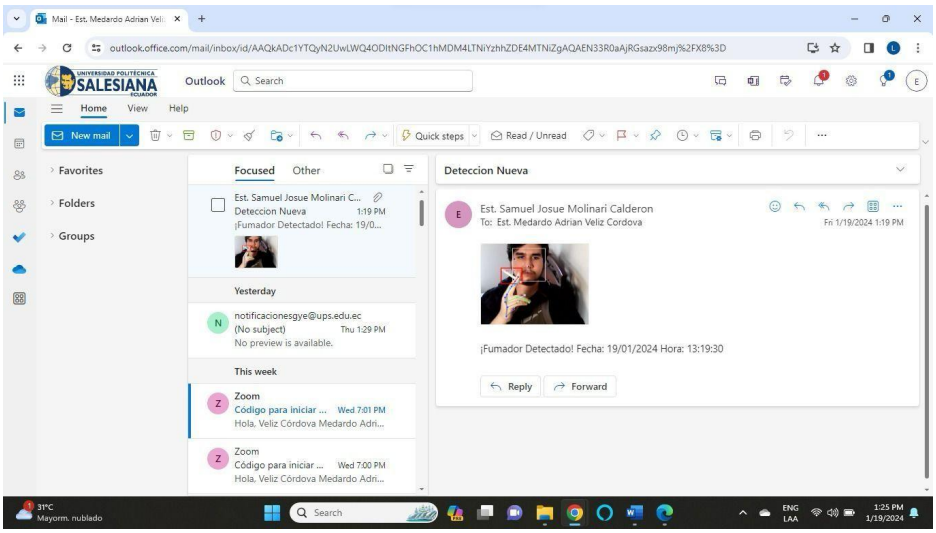


Figura 71. Función para enviar la detección por correo electrónico

5.5. Resultados código final interfaz grafico

En las figuras 72 a 73 se puede observar el interfaz grafico que cuenta con el título del proyecto, los nombres y el logo de la universidad, aparte tiene la capacidad de cambiar los parámetros iniciales de los modelos de detección utilizados.

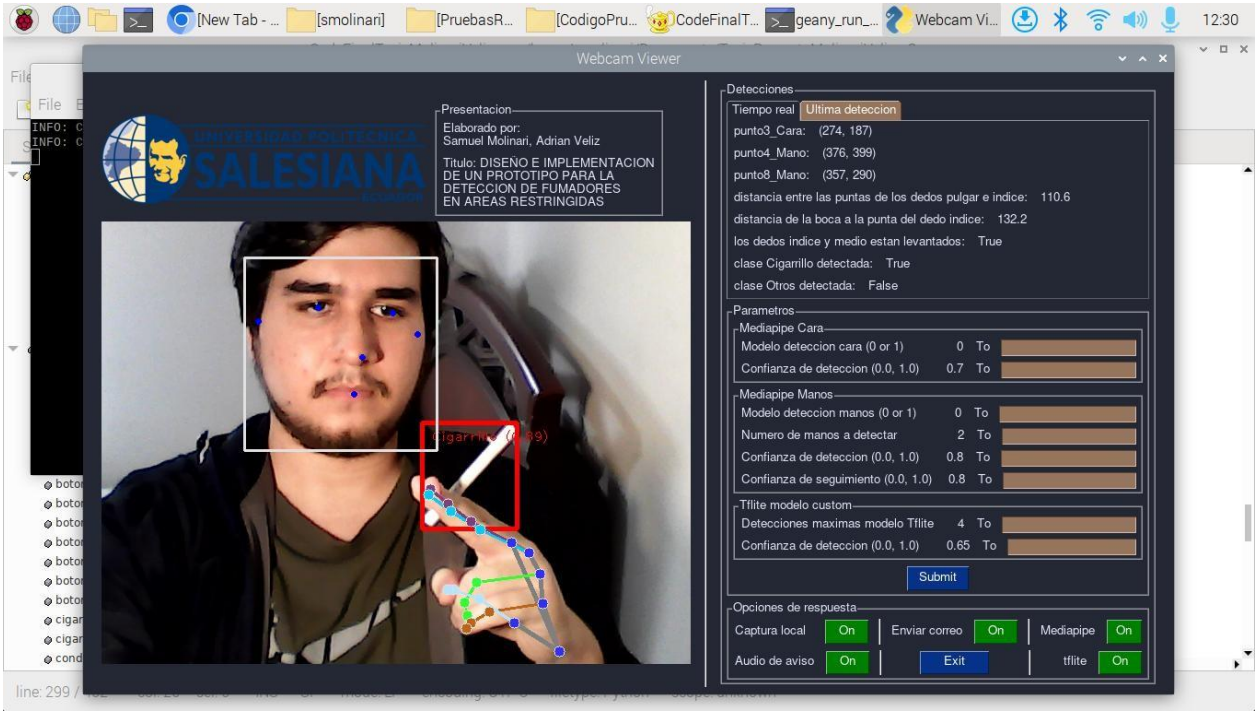


Figura 72. Resultados GUI clase Cigarrillo modelo tflite más mediapipe

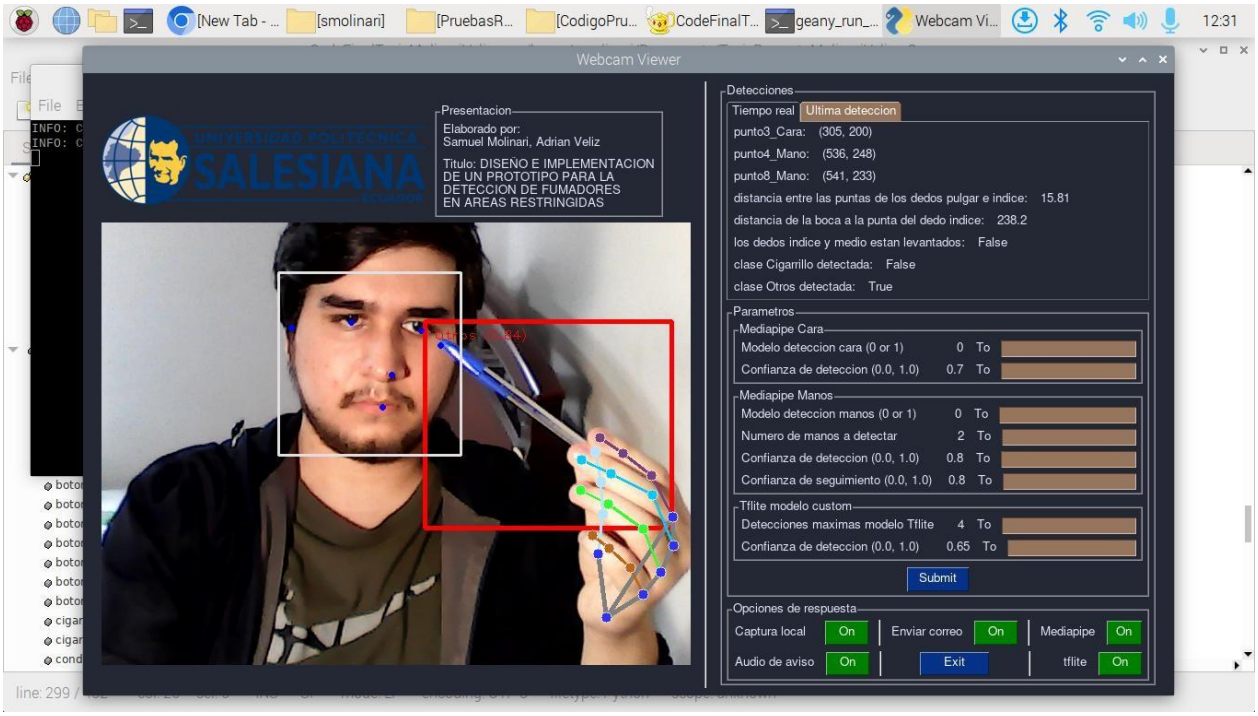
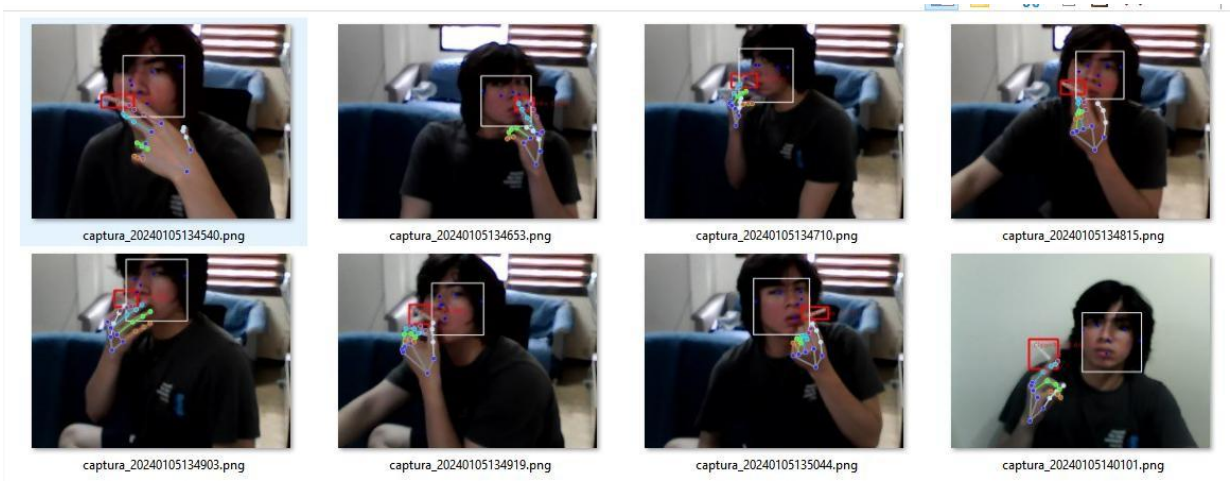
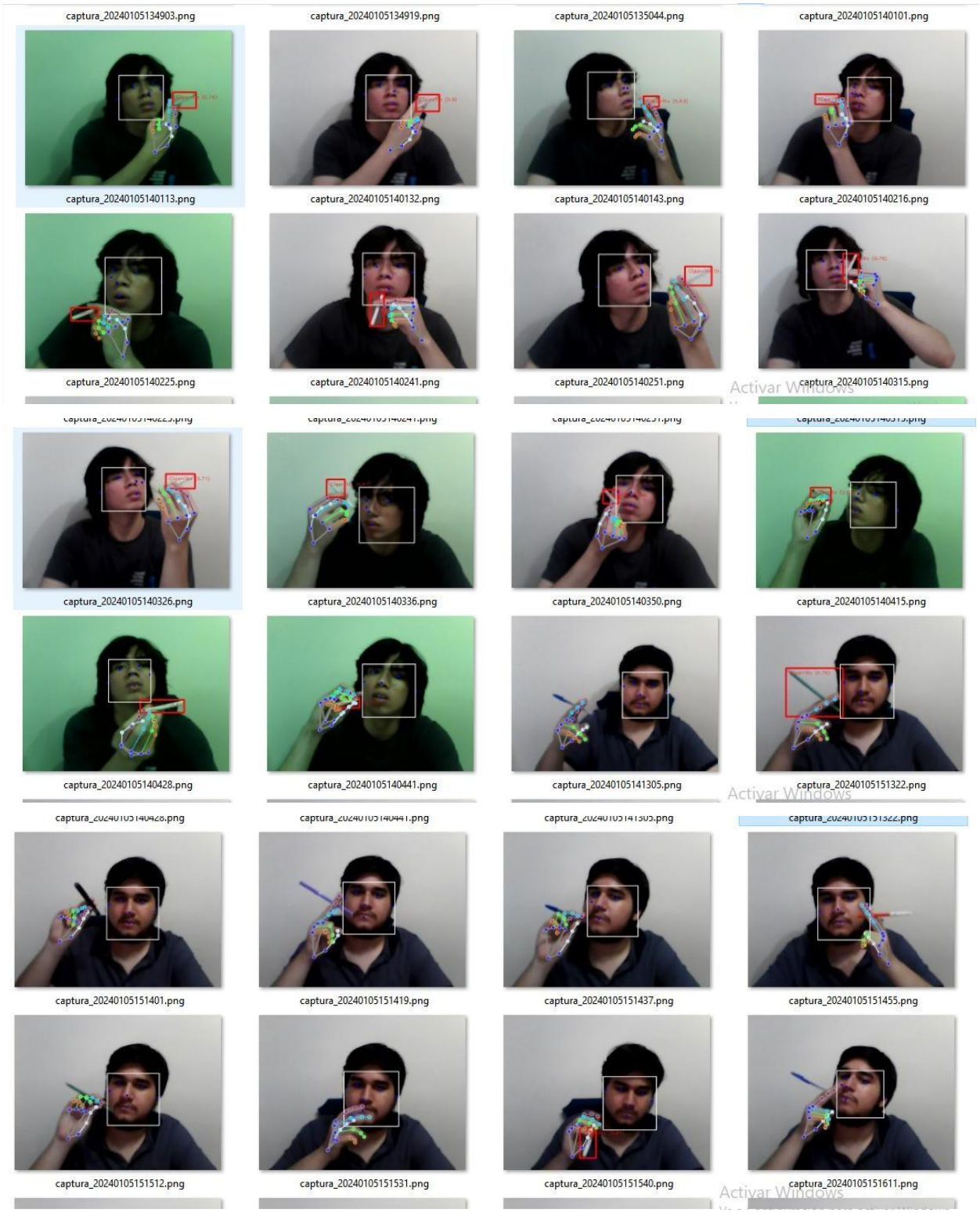


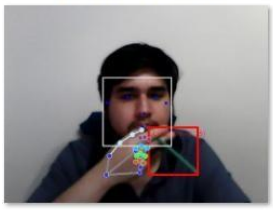
Figura 73. Resultados GUI clase Otros modelos tflite más mediapipe

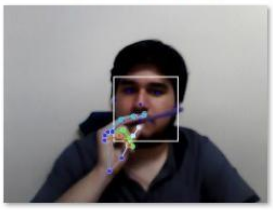
5.6. Análisis de resultados

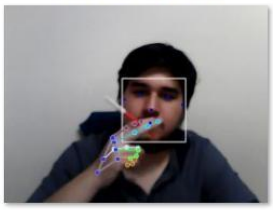
Se tomaron 300 muestras las cuales se dividen en 150 con gente fumando y 150 con gente que no está fumando, pero está sujetando un objeto parecido a un cigarrillo en una pose que haría un fumador. A cada una se le dio un periodo de 10 segundos para realizar la detección si se acababa el tiempo se capturaba la imagen igualmente.

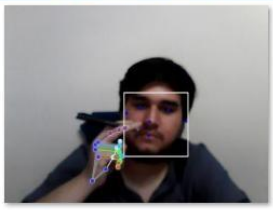


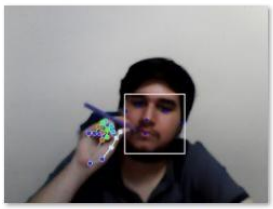

captura_20240105134903.png
captura_20240105134919.png
captura_20240105135044.png
captura_20240105140101.png
captura_20240105140113.png
captura_20240105140132.png
captura_20240105140143.png
captura_20240105140216.png
captura_20240105140225.png
captura_20240105140241.png
captura_20240105140251.png
captura_20240105140315.png
captura_20240105140326.png
captura_20240105140336.png
captura_20240105140350.png
captura_20240105140415.png
captura_20240105140428.png
captura_20240105140441.png
captura_20240105141305.png
captura_20240105151322.png
captura_20240105151401.png
captura_20240105151419.png
captura_20240105151437.png
captura_20240105151455.png
captura_20240105151512.png
captura_20240105151531.png
captura_20240105151540.png
captura_20240105151611.png

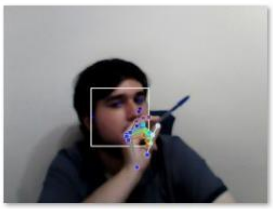

captura_20240105151624.png

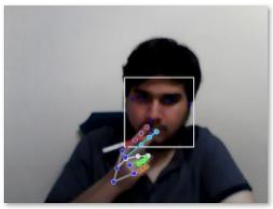

captura_20240105151700.png

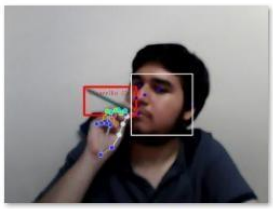

captura_20240105151716.png

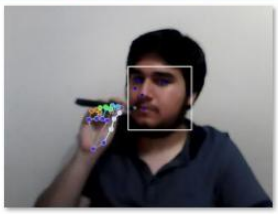

captura_20240105152030.png

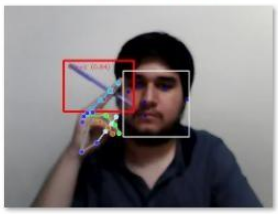

captura_20240105152052.png

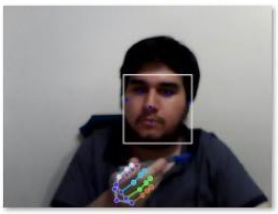

captura_20240105152111.png


captura_20240105152130.png

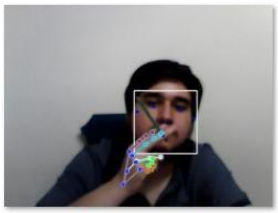

captura_20240105152147.png

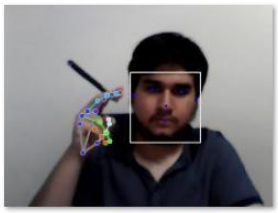

captura_20240105152447.png

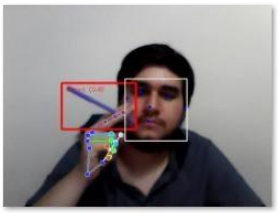

captura_20240105152504.png

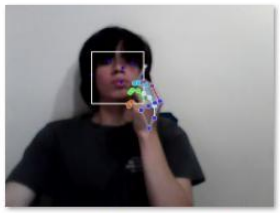

captura_20240105152618.png

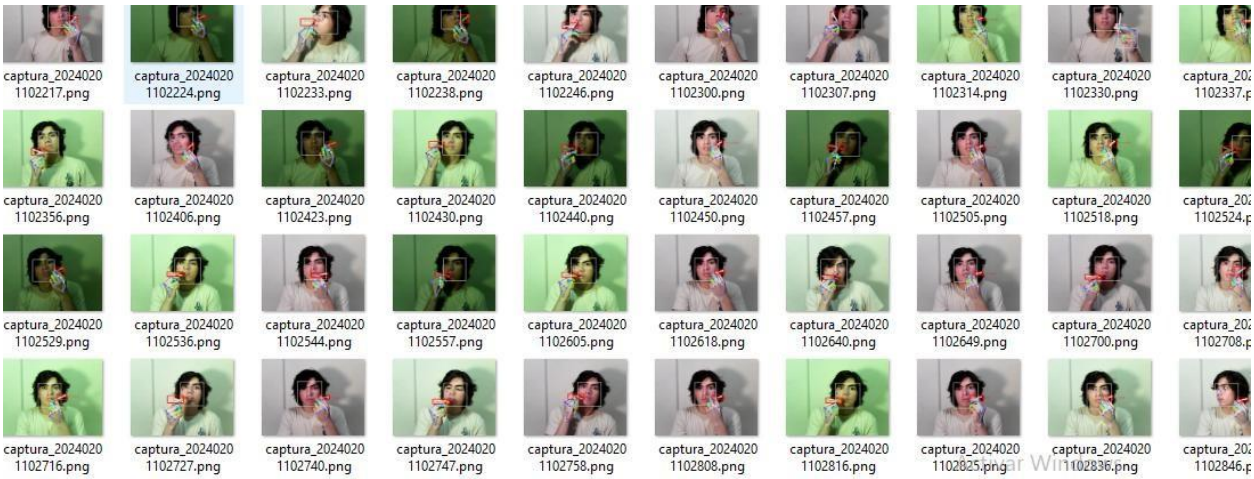

captura_20240105152632.png


captura_20240105152731.png


captura_20240105152752.png


captura_20240105152944.png


captura_20240105153213.png



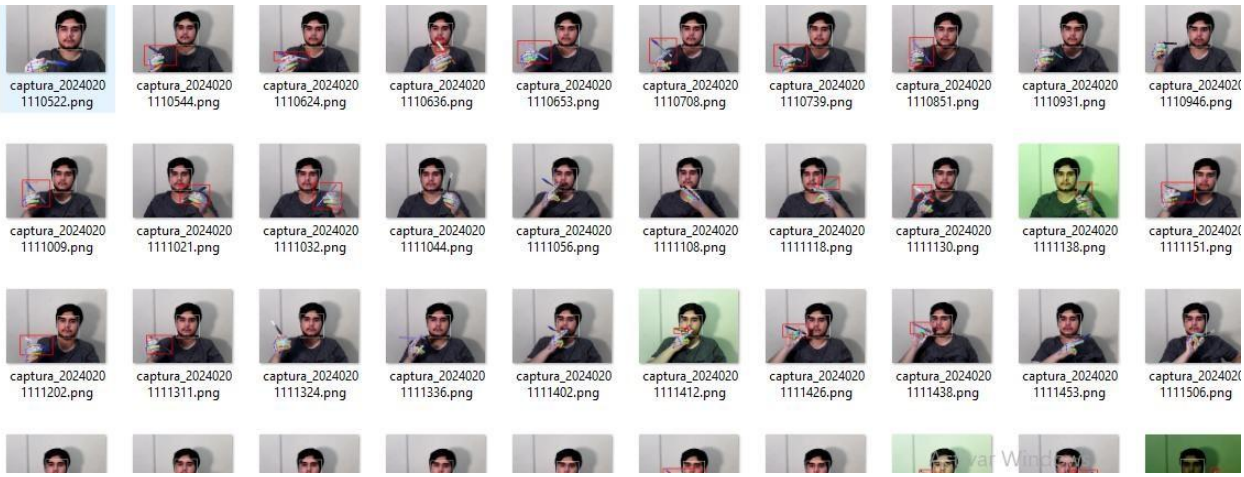


Figura 74. Muestras tomadas después de 10 segundos

La matriz de confusión es una herramienta que ofrece un análisis predictivo de la tarea de clasificación, puede proporcionar una mejor idea del modelo en términos de qué está clasificando correctamente y los errores que está cometiendo. La Figura 75 muestra la matriz de confusión del enfoque propuesto basado en la lógica de detección del prototipo.

los resultados de las muestras se dividen en:

- 144 verdaderos positivos
- 126 verdaderos negativos
- 24 falsos positivos
- 6 falsos negativos

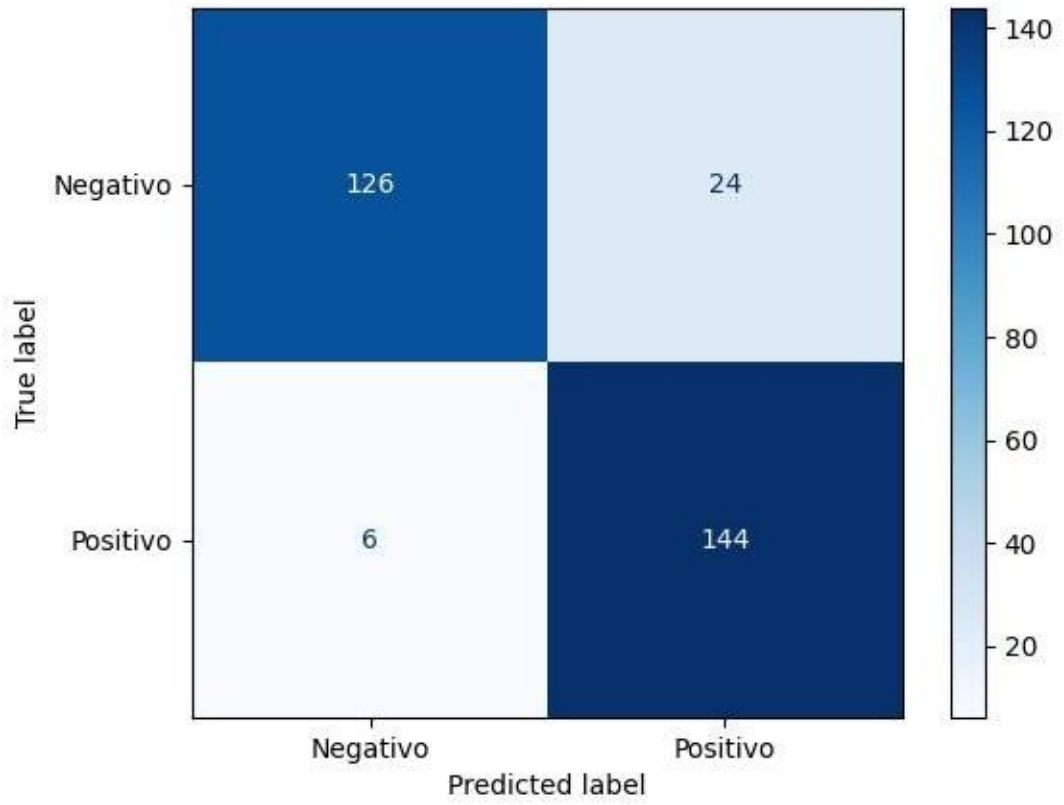


Figura 75. Matriz de confusión

Mediante la matriz de confusión que se ha generado con los resultados de las pruebas del prototipo se pueden calcular diferentes métricas para tener una visión más clara de la eficiencia del modelo (profesorDATA, 2020).

- Precisión

  La precisión es la proporción de predicciones correctas en relación con el total de predicciones. Se calcula como:

$$\frac{+}{+\ \ +\ +} = \frac{144 + 126}{144 + 126 + 24 + 6} = 0.9$$

- Sensibilidad

La sensibilidad mide la capacidad del modelo para identificar correctamente las instancias positivas.

$$\frac{}{+} = \frac{144}{144 + 6} = 0.96$$

- Especificidad

La especificidad mide la capacidad del modelo para identificar correctamente las instancias negativas.

$$\frac{}{+} = \frac{126}{126 + 24} = 0.84$$

Con los resultados de los cálculos de las métricas de la matriz de confusión se puede decir que:

1) El prototipo en general es eficiente teniendo una precisión de 0.9.

2) Es especialmente bueno identificando fumadores con un valor de sensibilidad de 0.96.

3) Tiene potencial de mejora cuando se trata de diferenciar fumadores de los no fumadores con un valor de 0.84 en especificidad, esto teniendo en cuenta que, las pruebas que se realizaron con no fumadores fueron sosteniendo objetos que fácilmente pueden pasar por cigarrillos y en poses que generalmente las tienen gente que está fumando.

CONCLUSIONES

Al finalizar el proyecto de titulación, se lograron cumplir con los objetivos específicos establecidos al inicio de este, dando como resultado los siguientes puntos:

- Se creo el data set con imágenes de gente fumando y se entrenó el modelo de detección de imágenes.

- Se implementó el prototipo utilizando un sistema embebido con Raspberry Pi como la unidad central, una cámara web como entrada y parlantes como salida.

- Se programo el código en Python de acuerdo con la investigación realizada de los métodos de detección de imágenes, modelos de CNN, librerías y documentación necesaria para la elaboración del prototipo.

- Se evaluó al prototipo para medir su efectividad al diferenciar fumadores de no fumadores mediante una matriz de confusión.

- Los resultados obtenidos demuestran que la combinación de la librería **Mediapipe** con el modelo customizado de detección de objetos de **tensorflow lite** ayudaron a hacer del prototipo más eficiente y confiable al momento de realizar las detecciones.

RECOMENDACIONES

Para el desarrollo del prototipo se debió tener en cuenta diversos factores:

- Al crear el data set se debe tener cuidado con las anotaciones de las imágenes que se usan ya que, si estas están mal anotadas, en el caso de la detección de objetos el modelo se entrenará con errores lo que no permitirá realizar la detección deseada.
- Los métodos de entrenamiento generalmente son dados por la librería en la que se basa el modelo en este caso tensorflow tiene una API la cual se utiliza para llevar a cabo el entrenamiento de un modelo customizado en su Google Colab pero por el momento este está desactualizado y no es compatible con sus librerías. Antes de ir de lleno con un modelo hay que realizar las pruebas pertinentes para asegurarse de que la documentación que proveen funcione de manera correcta o halla soluciones que ayuden con el proceso.
- Al momento de realizar el entrenamiento se recomienda utilizar el método de mini-batch, lo que significa utilizar un batch de entrenamiento muy pequeño en comparación del data set que sean en función de 2 , para el numero de épocas lo mas seguro es hacer pruebas con diferentes valores, en general lo más recomendado es usar valores de 60 en adelante.
- Es esencial para el proyecto elegir cuidadosamente el modelo de detección adecuado, considerando las limitaciones de hardware. A pesar de la consideración inicial de YOLOv8, se determinó que era demasiado pesado y no optimizado para la Raspberry Pi. En su lugar, se optó por EfficientDet0, que se incluye como modelo de prueba en tensorflow para Raspberry Pi. Esta elección garantizó resultados óptimos y rendimiento en tiempo real dentro de los recursos disponibles.
- La distancia de detección es un factor importante del prototipo, ya que determina su capacidad de detección. Al ajustar los modelos de mediapipe cambiando su tipo de 0 a 1, se puede mejorar su rendimiento en detecciones a distancias superiores a los 5 metros. Sin embargo, es importante tener en cuenta que el modelo de tensorflow lite puede encontrar limitaciones en cuanto a la distancia de detección debido a las características de la cámara web utilizada. Con una cámara de mayor calidad o una cámara de vigilancia con una iluminación adecuada, estas limitaciones podrían superarse para lograr detecciones a distancias mayores.

REFERENCIAS

Alvear, j. o. (2022). *Ucuenca* . Obtenido de https://www2.ucuenca.edu.ec/component/content/article/233-espanol/investigacion/blog-de-ciencia/1222-machine-learning?Itemid=437

basica, i. (s.f.). Obtenido de https://edu.gcfglobal.org/es/informatica-basica/los-perifericos-para-el-computador/1/

Bazarevsky, e. a. (11 de Julio de 2019). *Cornell University*. Obtenido de https://arxiv.org/abs/1907.05047

Everingham, M. (s.f.). Obtenido de https://homepages.inf.ed.ac.uk/ckiw/postscript/ijcv_voc09.pdf

Factory, G. (2024). *Geek Factory*. Obtenido de https://www.geekfactory.mx/tienda/tarjetas/raspberry-pi/fuente-de-alimentacion-para-raspberry-pi-4-oficial-eliminador/#:~:text=La%20fuente%20de%20alimentaci%C3%B3n%20para,accesorios%20originales%20para%20Raspberry%20Pi.

Fernandez, J. (1 de marzo de 2023). *Google*. Obtenido de https://github.com/google/mediapipe/blob/master/docs/solutions/hands.md

Fernandez, Y. (12 de enero de 2023). *xataka*. Obtenido de https://www.xataka.com/basics/hdd-vs-ssd

forum, T. (2023). Obtenido de https://github.com/tensorflow/tensorflow

FotoAccess. (29 de septiembre de 2022). *amazon*. Obtenido de https://www.amazon.com/-/es/Anbear-Adaptador-adaptador-c%C3%A1maras-soporte/dp/B07RZX9MCS#:~:text=El%20adaptador%20Micro%20HDMI%20macho,8%20pulgadas%20(7.9%20in).

GEANY, I. (2023). *IDE GEANY* . Obtenido de https://www.geany.org/

genius. (s.f.). *Product* . Obtenido de https://genius-me.com/product/?i=facecam-1000x

Google. (s.f.). *google* . Obtenido de https://research.google.com/colaboratory/intl/es/faq.html

IMB. (05 de 03 de 2021). Obtenido de https://www.ibm.com/docs/en/b2badv-communication/1.0.0?topic=concepts-mime-multipart-message-overview

Lite, T. (s.f.). *TensorFlow Lite*. Obtenido de https://www.tensorflow.org/lite/guide?hl=es-419

lugaresi, e. a. (14 de junio de 2019). *Cornell University*. Obtenido de https://arxiv.org/abs/1906.08172

LUNA, L. J. (2021). *Sistemas Embebidos.* Tlaquepaque,Jalisco: el Diario Oficial de la Federación.

M, A. (mayo de 27 de 2016). *Cornell university* . Obtenido de https://arxiv.org/abs/1605.08695

Mantaras, D. (2024). *NetApp*. Obtenido de https://www.bbvaopenmind.com/articulos/el-futuro-de-la-ia-hacia-inteligencias-artificiales-realmente-inteligentes/

Mediapipe. (2023). Obtenido de https://github.com/google/mediapipe/tree/master/docs/solutions

NVIDIA. (2023). *NVIDIA* . Obtenido de https://docs.nvidia.com/

Omes. (11 de febrero de 2020). Obtenido de https://omes-va.com/rgb/amp/

OpenCV. (22 de enero de 2024). *OpenCV*. Obtenido de https://docs.opencv.org/5.x/d1/dfb/intro.html

Perez. (2023).

Perez, j. A. (30 de septiembre de 2020). Obtenido de https://uvadoc.uva.es/bitstream/handle/10324/43277/TFG-G4450.pdf?sequence=1

ProfesorData. (27 de agosto de 2020). Obtenido de profesorDATA.com. https://profesordata.com/2020/08/07/evaluando-los-modelos-de-clasificacion-en-aprendizaje-automatico-la-matriz-de-confusion-claramente-explicada/

Pycharm. (2023). *Pycharm*. Obtenido de https://www.jetbrains.com/es-es/pycharm/

Python. (21 de enero de 2024). *Pythom software foundation*. Obtenido de https://docs.python.org/es/3/tutorial/

Raspberry . (s.f.). *Raspberry PI*. Obtenido de https://www.raspberrypi.com/products/raspberry-pi-zero-w/

raspberry, f. (2023). Obtenido de https://forums.raspberrypi.com/viewtopic.php?t=353534

RoboFlow. (junio de 2023). *Roboflow 100*. Obtenido de https://www.rf100.org/

Rodriguez, D. (9 de mayo de 2022). *Analytcis lane*. Obtenido de https://www.analyticslane.com/2022/05/09/creacion-basicas-de-gui-en-python-con-pysimplegui/

Rus, C. (1 de diciembre de 2020). *xataka*. Obtenido de https://www.xataka.com/accesorios/raspberry-pi-4-tiene-caja-oficial-refrigeracion-ventilador-cuesta-apenas-5-dolares

Tan, M. e. (20 de noviembre de 2019). *Cornell univeristy*. Obtenido de https://arxiv.org/abs/1911.09070

UTE, L. U. (2022). *La Universidad UTE*. Obtenido de https://www.ute.edu.ec/objetivos-de-desarrollo-sostenible/ods-politica-antitabaco/

vazquez, O. (2022). Raspberry Pi . *Con-Ciencia Boletín Científico de la Escuela Preparatoria No. 3*, vol 9, No 18 36 - 40 . Obtenido de file:///C:/Users/mcord/Downloads/9464-Manuscrito-53518-1-10-20220609.pdf

World Health Organization: WHO. (2023). *World Health Organization: WHO, .* Obtenido de https://www.who.int/

Yamashita. (2018). *Convolutional Neural Networks*. Obtenido de https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9

Zelezny. (16 de abril de 2023). *Bouncer*. Obtenido de https://www.usebouncer.com/es/glosario/cual-es-el-servidor-smtp-para-msn-email/#:~:text=SMTP%20(Simple%20Mail%20Transfer%20Protocol,de%20correo%20electr%C3%B3nico%20del%20destinatario.

ANEXOS

CRONOGRAMA

En la tabla 1 se muestra el cronograma que se va a llevar a cabo desde el 23 de octubre al 24 de enero. Primero se desarrollará el código y se lo entrenará con el "data set" esto tendrá una duración de seis semanas, también se va a armar el sistema embebido y la comunicación de los elementos con una duración de tres semanas, en las siguientes dos semanas se va a realizar las pruebas del prototipo y para finalizar se utilizarán dos semanas para hacer el documento de tesis.

No	Tareas realizadas	oct-23				nov-23				dic-23				ene-24
		Semanas												
		1	2	3	4	1	2	3	4	1	2	3	4	1
1	Desarrollo del código y entrenamiento del data set													
2	Armar el sistema embebido y configurar la comunicación de los elementos													
3	Realizar las pruebas del prototipo													
4	Desarrollo del documento de tesis													

Tabla 2: Cronograma

PRESUPUESTO

En la tabla 3 se puede observar los materiales que se van a utilizar entre ellos el Raspberry PI 4 modelo B que será el elemento principal de este prototipo

Materiales	Cantidad	Precio Unitario	Precio Total
Cámara web	1	$ 35,00	$ 35,00
Parlante	1	$ 30,00	$ 30,00
Raspberry PI kit(case, cables, fuente de poder)	1	$ 160,00	$ 160,00
Disco duro externo ssd 120GB	1	$50,00	$50
Total			**$ 275**

Tabla 3: Presupuesto

Código OpenCV

```
#Opencv con webcam
import cv2 as cv

#abrir camara
cam = cv.VideoCapture(0)

while True:
    ret, im = cam.read()

    #mostrar webcam
    cv.imshow('ObjectDetection', im)
    key = cv.waitKey(1)
    if key == 27:
```

```
                    break
cv.destroyAllWindows()
```

Código Mediapipe

```
#codigo pruebas mediapipe en tiempo real
import cv2 as cv
import mediapipe as mp

# Configuración del modelo de detección de caras
mp_face_detection = mp.solutions.face_detection
mp_face = mp_face_detection.FaceDetection(min_detection_confidence=0.8)

# Configuración del modelo de detección de manos
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(max_num_hands=2,
                    min_detection_confidence=0.8, min_tracking_confidence=0.8)

# Proceso principal para deteccion de cara y manos
def detect_and_draw(img):
    # Conversión de BGR a RGB para el procesamiento de Mediapipe
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    detect_and_draw_Face(img)
    detect_and_draw_Hands(img)
    # Conversión de RGB a BGR para mostrar la imagen con OpenCV
```

```
    img = cv.cvtColor(img, cv.COLOR_RGB2BGR)
    return img

# Deteccion cara
def detect_and_draw_Face(img):
    global point_3_face  # Declarar como variable global
    # Procesamiento de caras
    result_face = mp_face.process(img)

    # Dibujo de la cara si se detecta
    if result_face.detections:
        for detection in result_face.detections:
            # Dibujo de la detección de la cara
            mp_drawing.draw_detection(img, detection)

            # Obtener las coordenadas de los landmarks de la cara
            face_landmarks = detection.location_data.relative_keypoints
    return img

# Deteccion manos
def detect_and_draw_Hands(img):
    global point_8_hand, point_4_hand, condition

    # Procesamiento de manos con Mediapipe
    result_hands = hands.process(img)
    if result_hands.multi_hand_landmarks:

        # Dibujo de los puntos para cada mano detectada
        for hand_landmarks in result_hands.multi_hand_landmarks:
            mp_drawing.draw_landmarks(img,
                                      hand_landmarks,
```

```
                    mp_hands.HAND_CONNECTIONS,
                    mp_drawing_styles.get_default_hand_landmarks_style(),
                    mp_drawing_styles.get_default_hand_connections_style()
                    )

    return img


# Inicializar las distancias
distance_3_8 = None
distance_4_8 = None

# Variables para almacenar las coordenadas de los puntos 3 de face_landmark y los puntos 8 y 4
de landmarksHand
point_3_face  =  None
point_8_hand = None
point_4_hand = None
condition = False

cam = cv.VideoCapture(0)

while True:
    ret, im = cam.read()
    # Procesamiento de la imagen
    img_proces = detect_and_draw(im)
    cv.imshow('Mediapipe', img_proces)
    key = cv.waitKey(1)
    if key == 27:
        break
cv.destroyAllWindows()
```

Código TensorFlow Lite

```
#preubas tflite con modelo default en tiempo real
import cv2 as cv
from tflite_support.task import core
from tflite_support.task import processor
from tflite_support.task import vision
import utils

# Modelo de tensorflow lite
model='CigarrillosFinal.tflite'
num_threads=4

# Ajustes del Modelo
base_options=core.BaseOptions(file_name=model,use_coral=False, num_threads=num_threads)
detection_options=processor.DetectionOptions(max_results=4, score_threshold=.5)
options=vision.ObjectDetectorOptions(base_options=base_options,detection_options=detection_
options)
detector=vision.ObjectDetector.create_from_options(options)

#abrir camara
cam = cv.VideoCapture(0)

while True:
        ret, im = cam.read()
        # Inicializar variable de clase
        cigarrillo_detectado = False

        # Procesamiento de la imagen
        imRGB=cv.cvtColor(im,cv.COLOR_BGR2RGB)
        imTensor=vision.TensorImage.create_from_array(imRGB)
```

```
        detections=detector.detect(imTensor)
        image=utils.visualize(im, detections)

        #mostrar webcam
        cv.imshow('ObjectDetection', image)
        key = cv.waitKey(1)
        if key == 27:
                break
cv.destroyAllWindows()
```

Código Final con GUI

```
#codigo que utiliza el modelo customizado de tflite y el cálculo de las distancias de los
landmarks de mediapipe
#para determinar si hay o no un fumador en la imagen capturada por la webcam en tiempo real
#reproduce un audio de aviso si hay una deteccion
#guarda localmente la captura del frame de la camara donde se realizo la deteccion
#envia la captura del frame de la camara donde se realizo la deteccion por correo electronico
#tiene una interfaz grafica que permite modificar los parametros de las detecciones
import cv2 as cv
import simpleaudio as sa
import mediapipe as mp
import os
import time
import smtplib
from acceso import email_pass
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email.mime.image import MIMEImage
from tflite_support.task import core
from tflite_support.task import processor
```

```
from tflite_support.task import vision
import utils
import PySimpleGUI as sg

model_selection_face = 0
min_detection_confidence_face = 0.7

model_selection_hands = 0
max_hands = 2
min_detection_confidence_hands = 0.8
track_hands = 0.8

# Configuración del modelo de detección de caras
mp_face_detection = mp.solutions.face_detection
mp_face = mp_face_detection.FaceDetection(model_selection=model_selection_face,
                                  min_detection_confidence=min_detection_confidence_face)

# Configuración del modelo de detección de manos
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(model_complexity=model_selection_hands,
                       max_num_hands=max_hands,
                       min_detection_confidence=min_detection_confidence_hands,
                       min_tracking_confidence=track_hands)

#Abrir y cerrar camara
def abrir_camara(webCam='/dev/video0', width=640, height=480, fps=60):
    global cam
    cam = cv.VideoCapture(webCam)
    cam.set(cv.CAP_PROP_FRAME_WIDTH, width)
```

```
    cam.set(cv.CAP_PROP_FRAME_HEIGHT, height)
    cam.set(cv.CAP_PROP_FPS, fps)
    return cam

def cerrar_camara(cam):
    if cam.isOpened():
        cam.release()

# Proceso principal para deteccion de cara y manos
def detect_and_draw(img):
    # Conversión de BGR a RGB para el procesamiento de Mediapipe
    img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
    detect_and_draw_Face(img)
    detect_and_draw_Hands(img)
    # Conversión de RGB a BGR para mostrar la imagen con OpenCV
    img = cv.cvtColor(img, cv.COLOR_RGB2BGR)
    return img

# Deteccion cara
def detect_and_draw_Face(img):
    global point_3_face  # Declarar como variable global
    # Procesamiento de caras
    result_face = mp_face.process(img)

    # Dibujo de la cara si se detecta
    if result_face.detections:
        for detection in result_face.detections:
            # Dibujo de la detección de la cara
            mp_drawing.draw_detection(img, detection)

            # Obtener las coordenadas de los landmarks de la cara
```

```
            face_landmarks = detection.location_data.relative_keypoints

            # Almacenar las coordenadas del punto 3 de face_landmark
            if len(face_landmarks) > 3:
                point_3_face = (int(face_landmarks[3].x * img.shape[1]), int(face_landmarks[3].y *
img.shape[0]))
    return img

# Deteccion manos
def detect_and_draw_Hands(img):
    global point_8_hand, point_4_hand, condition

    # Procesamiento de manos con Mediapipe
    result_hands = hands.process(img)
    if result_hands.multi_hand_landmarks:

        # Dibujo de los puntos para cada mano detectada
        for hand_landmarks in result_hands.multi_hand_landmarks:
            mp_drawing.draw_landmarks(img,
                                      hand_landmarks,
                                      mp_hands.HAND_CONNECTIONS,
                                      mp_drawing_styles.get_default_hand_landmarks_style(),
                                      mp_drawing_styles.get_default_hand_connections_style()
                                      )

            # Obtener las coordenadas de los landmarks
            landmarksHand = hand_landmarks.landmark

            # Almacenar las coordenadas del punto 8 de landmarksHand
            #if len(landmarksHand) > 8:
```

```
            point_8_hand = (int(landmarksHand[8].x * img.shape[1]), int(landmarksHand[8].y *
img.shape[0]))

            # Almacenar las coordenadas del punto 4 de landmarksHand
            #if len(landmarksHand) > 4:
            point_4_hand = (int(landmarksHand[4].x * img.shape[1]), int(landmarksHand[4].y *
img.shape[0]))

            # Condición manos puntos 12 y 8
            condition = all(
                landmarksHand[i].y < landmarksHand[j].y
                for i, j in [(12, 20), (8, 20), (12, 16), (8, 16), (12, 4), (8, 4)])

    return img

# Capturar localmente una imagen
def captura(im):
    # Crear una carpeta para almacenar capturas si no existe
    folder_name = "capturas"
    if not os.path.exists(folder_name):
        os.makedirs(folder_name)

    # Generar un nombre único para la captura
    capture_name = f"captura_{time.strftime('%Y%m%d%H%M%S')}.png"

    # Ruta completa para guardar la captura en la carpeta
    capture_path = os.path.join(folder_name, capture_name)

    # Guardar la captura en la carpeta especificada
    cv.imwrite(capture_path, im)
```

```
# pista de audio de salida
def smokerDetected():
    wave_obj = sa.WaveObject.from_wave_file("fumadorDetectado.wav")
    play_obj = wave_obj.play()
    play_obj.wait_done()

# Enviar correo de imagen actual
def enviar_correo(captura):
    # Configurar los detalles del correo electrónico
    sender_email = "smolinari@est.ups.edu.ec"
    receiver_email = "mvelizc@est.ups.edu.ec"
    password = email_pass

    # Configurar el servidor SMTP
    smtp_server = "smtp-mail.outlook.com"
    smtp_port = 587

    # Configurar el mensaje del correo
    message = MIMEMultipart()
    message["From"] = sender_email
    message["To"] = receiver_email
    message["Subject"] = "Deteccion Nueva"

    # Adjuntar el texto del mensaje
    body = f"¡Fumador Detectado! {time.strftime('Fecha: %d/%m/%Y Hora: %H:%M:%S')}"
    text = MIMEText(body, "plain")
    message.attach(text)

    # Adjuntar la imagen al mensaje
    imgToBytes = cv.imencode('.png', captura)[1].tobytes()
```

```
    image = MIMEImage(imgToBytes,
name=f"captura_{time.strftime('%Y%m%d%H%M%S')}.png")
    message.attach(image)

    # Iniciar la conexión con el servidor SMTP y envíar el correo
    with smtplib.SMTP(smtp_server, smtp_port) as server:
        server.starttls()
        server.login(sender_email, password)
        server.sendmail(sender_email, receiver_email, message.as_string())
    print("¡Fumador detectado!")

# Modelo de tensorflow lite
model='CigarrillosFinal.tflite'
num_threads=4

max_tflite=4
conf_tflite=0.65

# Ajustes del Modelo
base_options=core.BaseOptions(file_name=model,use_coral=False, num_threads=num_threads)
detection_options=processor.DetectionOptions(max_results=max_tflite,
score_threshold=conf_tflite)
options=vision.ObjectDetectorOptions(base_options=base_options,detection_options=detection_
options)
detector=vision.ObjectDetector.create_from_options(options)

sg.theme('DarkTanBlue')

boton1 = True
boton2 = True
boton3 = True
```

```
boton4 = True
boton5 = True
impresiones = None

window_size = (1185, 675)

layout_l = [[sg.Image(filename='logo_ups_resized.png'),
sg.Frame('Presentacion',[[sg.Text('Elaborado por:\nSamuel Molinari, Adrian Veliz')],
[sg.Text('Titulo: DISEÑO E IMPLEMENTACION\nDE UN PROTOTIPO PARA
LA\nDETECCION DE FUMADORES\nEN AREAS RESTRINGIDAS')]])],
[sg.Image(filename='',  key='-IMAGE-')]]

layout_r = [[sg.Frame('Detecciones',
[[sg.TabGroup([[sg.Tab('Tiempo real',[[sg.Text('punto3_Cara:'), sg.Text(key='-OUTPUT1-')],
[sg.Text('punto4_Mano:'), sg.Text(key='-OUTPUT2-')],
[sg.Text('punto8_Mano:'),  sg.Text(key='-OUTPUT3-')],
[sg.Text('distancia entre las puntas de los dedos pulgar e indice:'), sg.Text(key='-OUTPUT4-'),
sg.T(s=10)],
[sg.Text('distancia de la boca a la punta del dedo indice:'), sg.Text(key='-OUTPUT5-')],
[sg.Text('los dedos indice y medio estan levantados:'), sg.Text(key='-OUTPUT6-')],
[sg.Text('clase Cigarrillo detectada:'), sg.Text(key='-OUTPUT7-')],
[sg.Text('clase Otros detectada:'), sg.Text(key='-OUTPUT8-')]]), sg.Tab('Ultima deteccion',
[[sg.Output(key='-Salidas-', s=(55,12))]])]])],
[sg.Frame('Parametros',
[[sg.Frame('Mediapipe Cara',
[[sg.Text('Modelo deteccion cara (0 or 1)          '), sg.Text(key='-OUT1-'), sg.Text('To'),
sg.In(key='-INPUT1-')],
[sg.Text('Confianza de deteccion (0.0, 1.0)    '), sg.Text(key='-OUT2-'), sg.Text('To'),
sg.In(key='-INPUT2-')]])],
[sg.Frame('Mediapipe Manos',
```

```
[[sg.Text('Modelo deteccion manos (0 or 1)       '), sg.Text(key='-OUT3-'), sg.Text('To'),
sg.In(key='-INPUT3-')],
[sg.Text('Numero de manos a detectar             '), sg.Text(key='-OUT4-'), sg.Text('To'),
sg.In(key='-INPUT4-')],
[sg.Text('Confianza de deteccion (0.0, 1.0)    '), sg.Text(key='-OUT5-'), sg.Text('To'),
sg.In(key='-INPUT5-')],
[sg.Text('Confianza de seguimiento (0.0, 1.0) '), sg.Text(key='-OUT6-'), sg.Text('To'),
sg.In(key='-INPUT6-')]])],
[sg.Frame('Tflite modelo custom',
[[sg.Text('Detecciones maximas modelo Tflite  '), sg.Text(key='-OUT7-'), sg.Text('To'),
sg.In(key='-INPUT7-')],
[sg.Text('Confianza de deteccion (0.0, 1.0)    '), sg.Text(key='-OUT8-'), sg.Text('To'),
sg.In(key='-INPUT8-')]])],
[sg.Push(), sg.Button('Submit'), sg.Push()]])],
[sg.Frame('Opciones de respuesta',
[[sg.Text('Captura local  '), sg.Button('On', size=(3, 1), button_color='white on green', key='-B1-
'), sg.VSep(),
sg.Text('Enviar correo'), sg.Button('On', size=(3, 1), button_color='white on green', key='-B2-'),
sg.VSep(),
sg.Text('Mediapipe'), sg.Button('On', size=(3, 1), button_color='white on green', key='-B3-')],
[sg.Text('Audio de aviso'), sg.Button('On', size=(3, 1), button_color='white on green', key='-B4-'),
sg.VSep(),
sg.Push(), sg.Button('Exit',s=(7,1)), sg.Push(), sg.VSep(),
sg.Text('      tflite'), sg.Button('On', size=(3, 1), button_color='white on green', key='-B5-')]])]])]]

layout = [[sg.Col(layout_l), sg.VSeperator(), sg.Col(layout_r)]]

window = sg.Window('Webcam Viewer', layout, size=window_size, resizable=True)
# Abrimos la camara al ejecutar
abrir_camara()
```

```
while cam.isOpened():

    # Inicializar las distancias
    distance_3_8 = None
    distance_4_8 = None

    # Variables para almacenar las coordenadas de los puntos 3 de face_landmark y los puntos 8 y 4 de landmarksHand
    point_3_face  =  None
    point_8_hand = None
    point_4_hand = None
    condition = False

    # Inicializar variable de clases
    cigarrillo_detectado = False
    Otros_detectado = False

    # Procesamiento de la imagen
    ret, im = cam.read()
    if boton3 and boton5:
      imRGB = cv.cvtColor(im,cv.COLOR_BGR2RGB)
      imTensor = vision.TensorImage.create_from_array(imRGB)
      detections = detector.detect(imTensor)
      image = utils.visualize(im, detections)
      img_proces = detect_and_draw(image)
    elif boton3 and not boton5:
      img_proces = detect_and_draw(im)
    elif boton5 and not boton3:
      imRGB = cv.cvtColor(im,cv.COLOR_BGR2RGB)
      imTensor = vision.TensorImage.create_from_array(imRGB)
      detections = detector.detect(imTensor)
```

```
  img_proces = utils.visualize(im, detections)
elif not boton3 and not boton5:
  img_proces = im

# Calcular la distancia si ambos puntos están disponibles
if point_4_hand is not None and point_8_hand is not None:
    distance_4_8 = ((point_4_hand[0] - point_8_hand[0]) ** 2 + (
          point_4_hand[1] - point_8_hand[1]) ** 2) ** 0.5

# Calcular la distancia si ambos puntos están disponibles
if point_3_face is not None and point_8_hand is not None:
    distance_3_8 = ((point_3_face[0] - point_8_hand[0]) ** 2 +
          ( point_3_face[1] - point_8_hand[1]) ** 2) ** 0.5

# Comprobar si la clase "Cigarrillo" esta siendo detectada
#print(detections)
for detection in detections.detections:
    for category in detection.categories:
       if category.category_name == 'Cigarrillo':
          cigarrillo_detectado = True
       if category.category_name == 'Otros':
          Otros_detectado = True
if distance_3_8 is not None and distance_4_8 is not None:
    if distance_3_8 < 125 and (distance_4_8 < 40 or condition) and cigarrillo_detectado:
      if boton1:
        captura(img_proces)
      if boton2:
        enviar_correo(img_proces)
      if boton4:
        smokerDetected()
      impresiones=(f"punto3_Cara: {point_3_face}\n"\
```

```
                f"punto4_Mano: {point_4_hand}\n"\
                f"punto8_Mano: {point_8_hand}\n"\
                f"distancia de la boca a la punta del dedo indice: {str(distance_3_8)[:5]}\n"\
                f"distancia entre las puntas de los dedos pulgar e indice:
{str(distance_4_8)[:5]}\n"\
                f"los dedos indice y medio estan levantados: {condition}\n"\
                f"Cigarrillo detectado: {cigarrillo_detectado}\n"\
                f"Otros detectado: {Otros_detectado}")
        cerrar_camara(cam)
        abrir_camara()

  event, values = window.read(timeout=0.1)
  if event == 'Exit' or event == sg.WIN_CLOSED:
    break
  if event == 'Submit':
    try:
      model_selection_face = int(values['-INPUT1-'])
    except ValueError:
        pass
    try:
      min_detection_confidence_face = float(values['-INPUT2-'])
    except ValueError:
        pass
    if values['-INPUT3-'] == '0' or values['-INPUT3-'] == '1':
      try:
       model_selection_hands = int(values['-INPUT3-'])
      except ValueError:
          pass
    if values['-INPUT4-'] != '0':
      try:
        max_hands = int(values['-INPUT4-'])
```

```
        except ValueError:
            pass
    try:
        min_detection_confidence_hands = float(values['-INPUT5-'])
    except ValueError:
        pass
    try:
        track_hands = float(values['-INPUT6-'])
    except ValueError:
        pass
    try:
        max_tflite = int(values['-INPUT7-'])
    except ValueError:
        pass
    try:
        conf_tflite = float(values['-INPUT8-'])
    except ValueError:
        pass

    mp_face = mp_face_detection.FaceDetection(model_selection=model_selection_face,
            min_detection_confidence=min_detection_confidence_face)

    hands = mp_hands.Hands(model_complexity=model_selection_hands,
            max_num_hands=max_hands,
            min_detection_confidence=min_detection_confidence_hands,
            min_tracking_confidence=track_hands)

    detection_options = processor.DetectionOptions(max_results=max_tflite,
score_threshold=conf_tflite)
  elif event == '-B1-':
      boton1 = not boton1
```

```
elif event == '-B2-':
    boton2 = not boton2

elif event == '-B3-':
    boton3 = not boton3

elif event == '-B4-':
    boton4 = not boton4

elif event == '-B5-':
    boton5 = not boton5

if ret:
    imgbytes = cv.imencode('.png', img_proces)[1].tobytes()
    window['-IMAGE-'].update(data=imgbytes)
    window['-OUTPUT1-'].update(str(point_3_face))
    window['-OUTPUT2-'].update(str(point_4_hand))
    window['-OUTPUT3-'].update(str(point_8_hand))
    window['-OUTPUT4-'].update(str(distance_4_8)[:5])
    window['-OUTPUT5-'].update(str(distance_3_8)[:5])
    window['-OUTPUT6-'].update(condition)
    window['-OUTPUT7-'].update(cigarrillo_detectado)
    window['-OUTPUT8-'].update(Otros_detectado)
    window['-OUT1-'].update(model_selection_face)
    window['-OUT2-'].update(min_detection_confidence_face)
    window['-OUT3-'].update(model_selection_hands)
    window['-OUT4-'].update(max_hands)
    window['-OUT5-'].update(min_detection_confidence_hands)
    window['-OUT6-'].update(track_hands)
    window['-OUT7-'].update(max_tflite)
```

```
        window['-OUT8-'].update(conf_tflite)
        window['-B1-'].update(text='On' if boton1 else 'Off', button_color='white on green' if boton1 else 'white on red')
        window['-B2-'].update(text='On' if boton2 else 'Off', button_color='white on green' if boton2 else 'white on red')
        window['-B3-'].update(text='On' if boton3 else 'Off', button_color='white on green' if boton3 else 'white on red')
        window['-B4-'].update(text='On' if boton4 else 'Off', button_color='white on green' if boton4 else 'white on red')
        window['-B5-'].update(text='On' if boton5 else 'Off', button_color='white on green' if boton5 else 'white on red')
        window['-Salidas-'].update(impresiones)

window.close()
```

Generar una matriz de confusión con valores escalares

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.metrics import ConfusionMatrixDisplay
true_positive = 23
false_positive = 7
true_negative = 18
false_negative = 2

cm = np.array([[true_negative, false_positive],
               [false_negative, true_positive]])

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Negativo", "Positivo"])
disp.plot(cmap='Blues', values_format='d')
plt.show()
```