UNIVERSIDAD ESTATAL
PENÍNSULA DE SANTA ELENA

FACULTAD DE SISTEMAS Y TELECOMUNICACIONES

CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN

TRABAJO DE TITULACIÓN

Propuesta Tecnológica, previo a la obtención del Título de:

INGENIERO EN TECNOLOGÍAS DE LA INFORMACIÓN

“Implementación de software para la detección de rostros sin mascarillas mediante el entrenamiento de un modelo de inteligencia artificial y visión por computadora en los laboratorios de informática”

AUTOR:
TOMALÁ GUARANDA ANDRÉS DARÍO

PROFESOR TUTOR:
ING. MARJORIE CORONEL, MGTI.

LA LIBERTAD – ECUADOR

2021

AGRADECIMIENTO

A Dios,

A mis padres,

A mis suegros,

A mi familia,

A mi tutora,

A IRIS.

DEDICATORIA

A mi esposa Katherine Alejandro Roca,

Que me incentivó a seguir esta carrera.

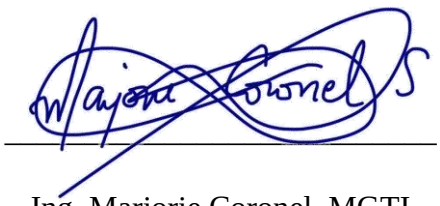
A mis hijas, Kenia y Karol,

Que me impulsan a no rendirme.

APROBACIÓN DEL TUTOR

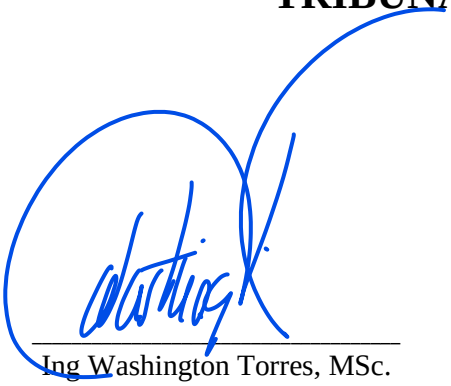
En mi calidad de Tutora del trabajo de titulación denominado: **"Implementación de software para la detección de rostros sin mascarillas mediante el entrenamiento de un modelo de inteligencia artificial y visión por computadora en los laboratorios de informática"**, elaborado por el estudiante **Tomalá Guaranda Andrés Darío**, de la carrera de **Tecnologías de la información** de la Universidad Estatal Península de Santa Elena, me permito declarar que luego de haber orientado, estudiado y revisado, lo apruebo en todas sus partes y autorizo al estudiante para que inicie los trámites legales correspondientes.

La libertad, octubre del 2021

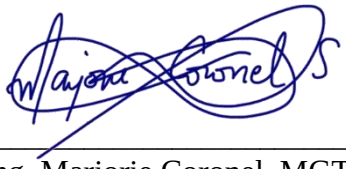


Ing. Marjorie Coronel, MGTI

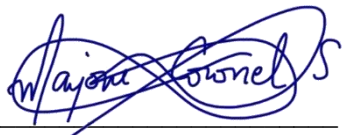
TRIBUNAL DE GRADO



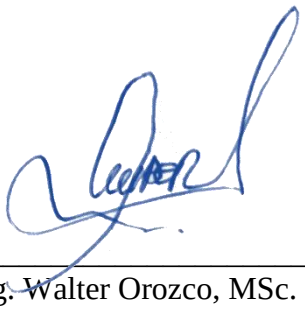
Ing Washington Torres, MSc.
DIRECTOR DE CARRERA



Ing. Marjorie Coronel, MGTI
PROFESOR TUTOR



Ing. Marjorie Coronel, MGTI
PROFESOR GUÍA



Ing. Walter Orozco, MSc.
PROFESOR ESPECIALISTA

RESUMEN

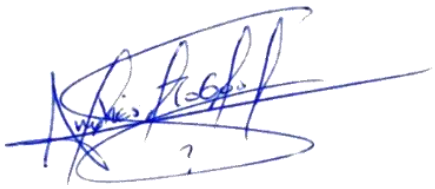
La Universidad Estatal Península de Santa Elena (UPSE) al igual que todos los centros educativos del país, se vieron obligados a cerrar sus puertas debido a la declaratoria de pandemia por aparición del SARS-Cov-2. El Gobierno Central del Ecuador mediante Resolución del Comité de Operaciones de Emergencia decreta el uso de mascarillas con carácter de obligatorio como medida de bioseguridad a fin de contrarrestar el contagio masivo. El control del uso de la mascarilla es una tarea muy importante sobre todo en lugares de concurrencia y por lo general está a cargo del personal de seguridad. En ocasiones resulta difícil detectar la totalidad de los infractores debido al rango de visibilidad del personal a cargo. El propósito de este proyecto es desarrollar un sistema capaz de detectar rostros sin mascarillas mediante la captura de fotogramas de video con OpenCV y el posterior análisis con un modelo de inteligencia artificial entrenado y basado en la arquitectura MobilenetV2. El modelo contiene una red de 128 neuronas de análisis desarrolladas con Tensorflow y Keras y el entrenamiento se realiza con un conjunto de imágenes de rostros con y sin mascarillas. Posteriormente el modelo entrenado es cargado en un script Python en el cual, primero se ejecuta la detección de rostros en cada fotograma mediante ResNet50 y luego la clasificación de rostros sin mascarillas. Para el desarrollo de la aplicación de escritorio que contendrá la ejecución del sistema se utiliza PyQt5. La aplicación posee una vista principal que renderiza el video en tiempo real y tres tipos de reportes estadísticos. Los resultados obtenidos son: un modelo entrenado con nivel de confianza superior al 80%, y, una aplicación para entornos de escritorios Linux que detecta rostros sin mascarillas, enmarca en rojo su posición dentro del fotograma guardando un registro en una base de datos SQLite.

ABSTRACT

The Universidad Estatal Península de Santa Elena (UPSE), like all educational centers in the country, were forced to close their doors due to the declaration of a pandemic due to the appearance of SARS-Cov-2. The Central Government of Ecuador, through a Resolution of the Emergency Operations Committee, decrees the use of masks as mandatory as a biosafety measure in order to counteract the massive contagion. The control of the use of the mask is a very important task, especially in crowded places and is usually in charge of the security personnel. Sometimes it is difficult to detect all the offenders due to the range of visibility of the personnel in charge. The purpose of this project is to develop a system capable of detecting faces without masks by capturing video frames with OpenCV and subsequent analysis with a trained artificial intelligence model based on the MobilenetV2 architecture. The model contains a network of 128 analysis neurons developed with Tensorflow and Keras and the training is carried out with a set of images of faces with and without masks. Subsequently, the trained model is loaded into a Python script in which, first the face detection is executed in each frame by means of ResNet50 and then the face classification without masks. For the development of the desktop application that will contain the execution of the system, PyQt5 is used. The application has a main view that renders the video in real time and three types of statistical reports. The results obtained are: a trained model with a confidence level greater than 80%, and, an application for Linux desktop environments that detects faces without masks, frames its position within the frame in red, saving a record in a SQLite database.

DECLARACIÓN

El contenido del presente Trabajo de Graduación es de mi responsabilidad; el patrimonio intelectual del mismo pertenece a la Universidad Estatal Península de Santa Elena.



Tomalá Guaranda Andrés Darío

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ITEM **PÁGINA**

ÍNDICE DE TABLAS

ITEM **PÁGINA**

LISTA DE ANEXOS

INTRODUCCIÓN

El SARS-Cov-2 es una enfermedad respiratoria infecciosa con sintomatología similar a una gripe común cuyo proceso de incubación tarda catorce días desde el contagio [1]. La transmisión de esta enfermedad se efectúa al tener contacto con fluidos respiratorios de una persona portadora. En Ecuador, el COE Nacional resuelve el uso obligatorio de mascarillas en espacios públicos como medida de prevención para mitigar el contagio comunitario [2]. Las mascarillas deben utilizarse como parte de una estrategia integral de medidas para suprimir la transmisión y salvar vidas; el uso de una mascarilla no basta para proporcionar una protección adecuada contra el SARS-Cov-2 [3].

La Inteligencia Artificial se usa para resolver problemas relacionados al análisis de datos masivos [4]. Las redes neuronales convolucionales son usadas para la clasificación de objetos dentro de una imagen y pueden ser entrenadas con un conjunto de datos apropiados para obtener un nivel de confianza cercano a cien por ciento. En [5] se desarrolló un modelo de detección de mascarillas a partir de la arquitectura de MobilenetV2 en el cuál se clasifica una imagen según las clases *with_mask,* bad*_mask* y *without_mask*. El proceso de formación se lleva a cabo entrenando el conjunto de datos de imágenes a través de métodos de aprendizaje por transferencia y ajuste fino [6].

Este documento propone el desarrollo una aplicación de escritorio para entornos Linux que analice fotogramas capturados con OpenCV, y, detecte rostros sin mascarillas mediante una red neuronal convolucional basada en MobilenetV2 a partir del módulo Keras de Tensorflow. Este modelo de red neuronal será entrenado en Google Colab a partir de un *dataset* con imágenes de rostros con y sin mascarilla. El resultado del entrenamiento es un conjunto compilado de datos capaz de clasificar rostros según las clases with_mask y without_mask.

La estructura de este documento consta de dos capítulos. El primer capítulo abarca la descripción de los antecedentes y la problemática ocurrida a partir de la aparición

del SARS-Cov-2, la metodología usada para desarrollar proyecto de Inteligencia Artificial y las principales características del sistema propuesto.

El segundo capítulo detalla el desarrollo de la solución. La metodología incremental para desarrollo de software indica que el proyecto se realice en cuatro etapas. La primera etapa comprende la implementación de un entorno virtual de trabajo con los componentes apropiados para desarrollo de proyectos de inteligencia artificial. La segunda etapa describe el análisis, diseño del modelo de red neuronal, el flujo de datos y el *dataset* para el entrenamiento. La tercera etapa se refiere al despliegue del modelo entrenado para detecciones de rostros sin mascarillas dentro del módulo de aplicación. La última etapa describe la integración del módulo de aplicación con el módulo de reportes a partir de las detecciones registradas en la base de datos local.

Los resultados obtenidos de la ejecución de este proyecto son: un entorno de trabajo con aceleración por GPU con las herramientas necesarias para proyectos de Inteligencia Artificial, una aplicación de escritorio Linux con un sistema de detección de rostros sin mascarillas y un gestor de reportes estadísticos integrados.

CAPÍTULO I

1. FUNDAMENTACIÓN

1.1. Antecedentes

Una de las principales vías de transmisión del SARS-Cov-2 es a través de las gotas respiratorias que expulsamos al hablar, cantar, toser o estornudar. Aunque todavía se está investigando acerca de este asunto, ya sabemos que el virus también se puede transmitir a través de personas que no presentan síntomas, lo cual significa que algunas personas pueden ser contagiosas sin darse cuenta [7]. Con la propagación del Covid-19, una de las pautas que más recomiendan los especialistas, además del lavado de manos con agua y jabón, es el uso de la mascarilla. Las personas se han concientizado con su uso, sin embargo, en muchos casos no son utilizadas de manera correcta [8].

La Universidad Estatal Península de Santa Elena cuyo campus matriz está ubicado en la ciudad de La Libertad es la única Entidad de Educación Superior de la provincia y por lo tanto la concurrencia de estudiantes ha crecido constantemente desde su inauguración en el año 1998. Sus autoridades principales son la Rectora Margarita Lamas González, Ph.D. y el Vicerrector Ing. Néstor Acosta Lozano, Ph.D. La UPSE cuenta con 6 Facultades y una oferta académica de 18 Carreras que abarcan amplios campos laborales tales como tecnología, biología, educación, civil, industrial entre otros [9].

La UPSE cuenta con dos revistas científicas reconocidas internacionalmente y en las cuales se han desplegado varias publicaciones a partir de la investigación científica promovida por los docentes y en trabajo conjunto con los estudiantes, estas revistas son la Revista Científica y Tecnológica UPSE y la Revista Ciencias Pedagógicas e Innovación [10].

La aparición a mediados de noviembre de 2019 del SARS-Cov-2 cambió el comportamiento de la población a nivel mundial [4]. A medida que surgía más información sobre la interacción de este virus con el cuerpo humano paralelamente se establecían nuevas medidas de bioseguridad para evitar la rápida propagación de

esta contagiosa y mortal enfermedad. Ante la situación los gobiernos locales propusieron restricciones sanitarias tales como el distanciamiento social, lavado de manos, desinfección total y uso obligatorio de mascarillas entre otras [2].

Actualmente controlar el uso adecuado de la mascarilla se ha convertido en una tarea de vital importancia, pero a la vez de mucho esfuerzo por la cantidad de personas que circulan a diario dentro o fuera de un establecimiento (Anexo 1, Anexo 2). Por lo general esta tarea ha sido encargada a los guardias de seguridad o a un grupo de personas que deben estar siempre atentos al comportamiento de las demás personas y obviamente al ser una tarea visual depende del rango de visibilidad que tenga el encargado.

La tecnología ha jugado un papel muy importante en tiempos de pandemia, desde comunicación hasta simulaciones para poder mitigar el impacto que el nuevo coronavirus ha causado a nivel mundial [4]. Las entidades públicas y privadas buscan soluciones que permitan controlar remotamente las actividades que se realizan en sus establecimientos y evitar en gran parte el contacto entre empleados o asistentes. Controlar el comportamiento de las personas frente a las medidas de bioseguridad se volvió una tarea desmesurada por la concurrencia y la falta de personal para ejercer dicha tarea.

En Ecuador la Escuela Superior Politécnica del Litoral en convenio con Tiendas Industriales S.A. fundaron INARI que usa modelos por computadora para controlar al personal mediante video. En INARI, desarrollaron un sistema capaz de estimar el porcentaje de personas que utilizan máscaras mediante el análisis de las imágenes en un vídeo. El sistema detecta si una persona está utilizando una máscara y considera sólo las detecciones más confiables para la estimación [11].

Ante la eminente reactivación de las actividades comerciales en el Ecuador el SIS ECU-911 implementó una herramienta de soporte para el control de la ciudadanía. Distancia2 es el software desplegado en la central de monitoreo en Quito como plan piloto. Es una plataforma que aprovecha la infraestructura de video vigilancia disponible en el país y la combina con inteligencia artificial (IA) para mitigar el riesgo de contagio en sitios donde se registren aglomeraciones de ciudadanos [12].

Distancia2 usa imágenes de las cámaras para medir el distanciamiento, si se detectan concentraciones, se genera una ficha y se remite a las instituciones de control. El software lo desarrolló el BID y el ECU 911 apoyó para integrarlo al sistema de video vigilancia, ejecutar pruebas, ajustes y modificaciones [13]. Dentro de las nuevas actualizaciones se encuentra el modelo para la detección de mascarillas cuyo nivel de confianza está por sobre el 90%.

La afluencia de personas en los espacios públicos y privados promueve un desorden en el comportamiento y el control de las medidas establecidas para evitar la propagación del SARS-Cov-2. El uso de herramientas informáticas se vuelve indispensable para monitorear en tiempo real un gran número de personas, reduciendo considerablemente el contacto entre el encargado del control y los usuarios.

1.2. Descripción del proyecto

La OMS promueve el uso de la mascarilla como uno de los pilares para reducir la propagación del SARS-Cov-2 [3] y ante esta indicación se plantea implementar un sistema de monitoreo autónomo basado en video vigilancia con análisis de datos en tiempo real para ayudar a controlar el uso de la mascarilla dentro del laboratorio de la Facultad de Sistemas y Telecomunicaciones de la Universidad Estatal Península de Santa Elena.

El desarrollo de este proyecto está divido en cuatro etapas [14], la primera etapa incluye la implementación del entorno de trabajo virtual con el fin de no causar conflictos con el sistema operativo. En esta etapa se instalan todas las librerías y componentes para el manejo, análisis y clasificación de imágenes [15], [16] tales como OpenCV, CUDA y Tensorflow, el entorno virtual es creado con Anaconda y Python como lenguaje base. La creación de un entorno virtual permite encapsular los módulos instalados sin generar conflictos con el sistema operativo anfitrión. CUDA y Tensorflow para GPU permiten realizar cálculos matemáticos a mayor velocidad.

En segunda etapa intervienen la herramienta Tensorflow con los módulos de Keras y Mobilnet para realizar el entrenamiento de un modelo sin datos. Mediante un script Python se importan y configuran clases y métodos que pueden ser configurados para obtener mejores índices de precisión y rendimiento. Para el entrenamiento del modelo se usarán 3000 imágenes el 50% de personas con mascarillas y el otro 50% de personas sin mascarillas, adicional a esto tres archivos que contienen las clases o etiquetas, las rutas de las imágenes y el último contiene las coordenadas del objeto a reconocer en este caso la mascarilla.

Todo este proceso de entrenamiento requiere hardware de gran potencia, debido a esto se usará Google Colab con renderizado en GPU para entrenar el modelo sin perder datos. El entrenamiento consiste en guardar en un archivo de pesos con extensión h5, las coordenadas de los puntos más importantes del dataset incluyendo la clase a la que pertenece. Es aquí donde la inteligencia artificial juega un papel fundamental, pues con una red de 128 neuronas de análisis y dos neuronas de decisión logra reconocer patrones y aprender a identificar un objeto dentro de una imagen, este proceso se repite durante 100 épocas para alcanzar una efectividad cercana al 100%.

La tercera etapa se basa en el desarrollo del software de escritorio que contará con un panel de administración que a su vez consta con una barra lateral para la visualización de información relevante como el número de personas con mascarilla y sin mascarilla, también un directorio de las capturas de personas que han incumplido la normativa. Además, cuenta con dos ventanas de monitoreo en video una para el administrador y una adicional para mostrar a los infractores si el administrador lo requiere.

La última etapa comprende el desarrollo de los reportes y para este propósito es necesario la integración con el módulo de aplicación. El módulo de reportes cuenta con tres tipos de gráficos: histogramas, barras y pastel respectivamente que interactúan con variables como el tiempo y detecciones.

La función principal de este sistema es detectar personas sin mascarillas y este proceso se describe de la siguiente forma:

a) La cámara web conectada al sistema emite un video del cual cada fotograma es capturado y enviado individualmente.
b) Los fotogramas capturados mediante visión por computadora son convertidos de RGB a escalas de grises resaltando las características más importantes.
c) Estas características son enviadas al modelo para detectar rostros.
d) Los rostros detectados son analizados y comparados según las clases del modelo, aquí se envían una lista de coordenadas de la ubicación de los rostros.
e) Con el uso de visión por computadora las coordenadas son mapeadas y resaltadas en cada fotograma, luego son renderizadas en video y enmarcar un rectángulo verde si las personas tienen mascarilla y rojo si no tienen.

Todo este análisis se hace en cuestión de milisegundos y depende del hardware en el que se implementa el sistema. Para tener una idea clara la renderización de video por CPU devuelve hasta cinco fotogramas por segundo, esto indica que todo el proceso de detecciones se realiza cinco veces durante un segundo. Con el uso de GPU se puede mostrar hasta 30 fotogramas por segundo, esto ayudará a tener un mejor índice de detecciones en tiempo real.

El sistema analizará cada persona dentro del laboratorio y emitirá una alerta si encuentra a alguna persona sin mascarilla, inmediatamente se guardará un registro en un base datos que incluye la hora y fecha, el número de personas que estuvieron sin mascarilla y adicional se guardará una imagen del infractor dentro de un directorio local.

Para el desarrollo y despliegue de este proyecto se han tomado en cuenta las siguientes especificaciones: procesador de 6ta generación, 8GB de memoria RAM, webcam de 720p o superior con autoenfoque, 3000 archivos para el entrenamiento del modelo, tarjeta gráfica Nvidia GT710.

Este proyecto está dirigido a la línea de investigación Desarrollo de Software de la Carrera de Ingeniería en Tecnologías de la Información. La implementación de un sistema de monitoreo permitirá la detección y correcto uso de la mascarilla dentro

del laboratorio de la Facultad de Sistemas y Telecomunicaciones de la Universidad Estatal Península de Santa Elena reduciendo considerablemente la propagación del nuevo coronavirus. Este proyecto contará con las siguientes características:

- ✓ El sistema está diseñado para realizar la detección de manera autónoma, emitir alertas y guardar registros.
- ✓ El administrador es quien decide el inicio y fin de la video vigilancia.
- ✓ El sistema posee dos pantallas de monitoreo duplicadas para el administrador del sistema y la otra in situ.
- ✓ El sistema analizará cada fotograma del video y mediante un análisis de un modelo entrenado muestra los rostros con mascarillas en cuadros verdes y en rojos los que no tienen mascarillas.
- ✓ El sistema NO realiza reconocimiento facial solo detección de objetos.
- ✓ Muestra datos reales sobre el número de personas con y sin mascarillas.
- ✓ Muestra total de infracciones del día actual y el anterior.
- ✓ Cuenta con un explorador de imágenes enlazado al repositorio de las capturas de los infractores.
- ✓ El menú tiene las opciones de Mostrar Segunda pantalla, Reportes y Salir.
- ✓ El reporte tiene el formato de histograma para visualizar datos históricos, tomando en cuenta variables como el tiempo y el tipo de infractor.
- ✓ Se podrá descargar un informe detallado de la información recolectada.
- ✓ El sistema será desplegado en el computador del encargado del laboratorio.
- ✓ La captura de video se la realiza mediante una webcam con 720p de densidad de píxeles y autoenfoque.
- ✓ El procesamiento del video se realiza tanto en CPU como en GPU, por lo tanto, el hardware estará destinado a esta única tarea.
- ✓ El administrador del sistema podrá descargar datos en formato csv de acuerdo al reporte mostrado.
- ✓ El sistema no permanecerá en ejecución 24/7 sólo en horas de concurrencia de personal.

Esta primera versión está completamente desarrollada en Python para entornos Linux con modelos entrenados para la detección de rostros con una precisión del

80%. El software está desarrollado para entorno de escritorio con compatibilidad para Linux en tecnologías con licencias libres GNU GPL [17], mediante el paradigma de programación MVC (Modelo-Vista-Controlador). La interfaz tiene un diseño intuitivo, las tareas a realizar son claras y la intervención del usuario es mínima. Las herramientas utilizadas para desarrollo y despliegue son:

Python 3.7: Es el lenguaje base para el desarrollo del proyecto. El entrenamiento, ejecución y despliegue del modelo de Inteligencia Artificial está codificado en el lenguaje de programación Python [18].

OpenCV 4.2: Es la librería utilizada para el análisis de fotogramas [19].

PyQt5: Qt es un conjunto de herramientas que provee de librerías para el desarrollo de interfaces de usuario en diferentes plataformas [20].

SQLite3: Es una base de datos ligera que se basa en el manejo de archivos [21].

TensorFlow: Es una biblioteca de herramientas que se usan para el análisis de datos orientada al aprendizaje autónomo [22].

Keras: es una librería de Tensorflow escrita en Python para el desarrollo de inteligencia artificial [23].

Resnet: Son redes residuales que realizan análisis de imágenes en busca de patrones preestablecidos [15].

Mobilenet: Es una librería de aceleración de tratamiento de imágenes [16].

Google Colab: Es un entorno de ejecución de código Python con aceleración de GPU para el análisis de datos [24].

Anaconda: Es un ecosistema de herramientas para ciencia de datos [25].

CUDA: Es un acelerador gráfico que aumenta la velocidad en los cálculos matemáticos, está direccionado al desarrollo de inteligencia artificial [26].

1.3. Objetivos del proyecto

1.3.1. Objetivo general

Implementar un sistema de detección de rostros mediante la captura de video con visión por computadora basado en un modelo de inteligencia artificial entrenado para ayudar con el control del cumplimiento del uso de mascarillas en los laboratorios de FACSISTEL.

1.3.2. Objetivos específicos

1. Implementar un entorno virtual con módulos para visión por computadora y análisis con GPU.
2. Implementar código Python para el entrenamiento del modelo de detección de mascarillas.
3. Compilar un modelo de datos para el análisis de rostros sin mascarillas.
4. Generar reportes estadísticos de las infracciones detectadas en intervalos de tiempo diarios o semanales mediante una interfaz de administración.

1.4. Justificación

El uso de una mascarilla quirúrgica es una de las medidas profilácticas para limitar la propagación de determinadas enfermedades respiratorias, entre ellas la infección por el COVID-19, en las zonas afectadas [27]. Las mascarillas deben utilizarse como parte de una estrategia integral de medidas para suprimir la transmisión y salvar vidas [3].

El brote del SARS-Cov-2 obligó a los gobiernos centrales y locales a regular de forma exhaustiva el uso de mascarillas. El Comité de Operaciones de Emergencias del Ecuador hace un llamado a las autoridades y exhorta a la ciudadanía en general a respetar las medidas establecidas para enfrentar la pandemia [2].

La inteligencia artificial o IA puede entenderse como una disciplina perteneciente a las ciencias de la computación, que plantea modelos computacionales de aprendizaje basado en redes neuronales biológicas humanas, […] Otra especialidad de la IA es el aprendizaje profundo (Deep Learning) para evaluar datos de tipo

imagen, video y audio empleando redes neuronales convolucionales con sus diversas variantes [4].

El aporte que la tecnología está brindando en tiempos de pandemia ha sido de gran ayuda no sólo en el campo medicinal sino también en el gerencial y ocupacional. La inserción de modelos de datos, clasificación de imágenes y predicciones basados en *dataset* con la ayuda de IA ha permitido conocer el comportamiento del virus en el cuerpo humano, además de interacción social y la responsabilidad de la ciudadanía al cumplir con los protocolos de bioseguridad. Una particularidad de las tecnologías asociadas con la IA, es que precisan de algoritmos cada vez más potentes que faciliten el procesamiento con una mayor cantidad de información [4].

Ante la necesidad de un control más efectivo frente al uso correcto de la mascarilla en espacios comunes en especial los laboratorios de FACSISTEL, se implementará un sistema de monitoreo autónomo mediante video vigilancia para detectar posibles infractores que irrespeten las medidas establecidas por las autoridades.

El sistema permitirá al administrador de los laboratorios monitorear en tiempo real el comportamiento de los estudiantes y personal en general dentro del atrio de los laboratorios frente al uso de la mascarilla. Mediante la implementación de un algoritmo de revisión autónomo el sistema podrá detectar rostros sin mascarilla aún sin la presencia de un operador.

El sistema proporcionará un nivel de confianza superior al 90% frente a las detecciones que realice, los datos mantendrán su integridad y confidencialidad dentro de la base de datos para evitar pérdidas de información. Además, podrá generar reportes a partir de variables dinámicas para una mejor visualización de los datos obtenidos.

Es necesario recalcar que, aunque este proyecto será desplegado dentro de los laboratorios de FACSISTEL, podría ser replicado dentro de otros espacios comunes dentro de la Facultad o Campus Universitario a fin de cumplir con las indicaciones del COE Nacional [2].

En esta primera versión de la aplicación se establece el desarrollo para entornos de escritorio con una base de datos local, pero está abierta a la posibilidad de escalar e ingresar datos a una base de datos centralizada para mayor seguridad de la información obtenida.

Los reportes generados a partir de la captura de los datos obtenidos ayudarán al administrador de los laboratorios a la toma de decisiones respecto al comportamiento de los infractores consultando el registro de hora y fecha exactos del hecho, incluso podría identificarlo mediante la captura del rostro que se genera automáticamente.

Este proyecto está alineado al Plan Nacional de Desarrollo "Toda una vida", especificado en los siguientes ejes:

Eje 1.- Derechos para Todos Durante Toda la Vida

Objetivo 1.- Garantizar una vida digna con iguales oportunidades para todas las personas [28].

Política 1.15.- Promover el uso y el disfrute de un hábitat seguro, que permita el acceso equitativo a los espacios públicos con enfoque inclusivo [28].

Eje 2.- Economía al servicio de la sociedad.

Objetivo 5.- Impulsar la productividad y competitividad para el crecimiento económico sostenible de manera redistributiva y solidaria [28].

Política 5.6.- Promover la investigación, la formación, la capacitación, el desarrollo y la transferencia tecnológica, la innovación y el emprendimiento, la protección de la propiedad intelectual, para impulsar el cambio de la matriz productiva mediante la vinculación entre el sector público, productivo y las universidades [28].

1.5. Metodología del proyecto

Debido a que el uso de mascarillas es obligatorio desde la aparición del SARS-Cov-2 [19] y a que la reactivación de las actividades se ha vuelto paulatina dentro de la

Universidad, para este proyecto se aplican estudios exploratorios [29] que permitan recolectar datos relevantes respecto al tema que será un valor agregado a la bibliografía disponible en el desarrollo de este proyecto.

Con el fin de conocer el proceso de detección de uso obligatorio de mascarilla y debido a que las actividades académicas no se han podido reanudar se ha empleado la técnica de observación [29] en sitios de gran afluencia de personas como la Iglesia Matriz y el Registro Civil de Santa Elena.

En la técnica de observación (Anexo 1, Anexo 2) se evidenció como se realiza el proceso en sitios de afluencia y las acciones que se toman respecto a la infracción cometida. Además, se pudo constatar que a pesar de las recomendaciones y el personal destinado a la labor no se pudo detectar el 100% de las infracciones.

Con este proyecto se plantea tener el mayor índice de detecciones durante la jornada dentro de los laboratorios de FACSISTEL, debido a la afluencia de participantes y que sólo existe una persona encargada de los laboratorios, será una tarea muy complicada poder detectar a todos los infractores y mantener un registro de los mismos.

1.6. Beneficiarios del proyecto

La implementación del software tiene como beneficiarios directos a los docentes, personal administrativo y estudiantes de la FACULTAD DE SISTEMAS Y TELECOMUNICACIONES. Los beneficiarios indirectos son todas aquellas personas que ingresan al edificio de los laboratorios de FACSISTEL.

1.7. Variable

La variable del proyecto es el número de infractores detectados en una jornada de trabajo. Esta variable está sujeta a condiciones tales como la iluminación del lugar, el enfoque, ángulo de cobertura y densidad de píxeles de la cámara.

1.8. Análisis de la observación

A pesar de que la observación (Anexo 1, Anexo 2) no se realizó dentro del sitio donde se implementará, los lugares determinados nos dan una idea clara de cómo se realiza el control del uso de mascarillas en sectores concurridos.

Se identificaron falencias en el proceso de detección que dependen del rango de acción del encargado, en ambos casos se evidenció que varios infractores esperaban una distracción del encargado para poder retirarse la mascarilla. Algunas personas obedecían luego del llamado de atención, pero otros eran reincidentes en su falta. Otro punto importante es el reporte de infractores que se genera pues, en el caso de la Iglesia se anota en un cuaderno y el Registro Civil no existe tal reporte.

El cansancio de los controladores del lugar no permitía tener una gran cobertura dentro del establecimiento, por lo tanto, no se lograba tener el total de infracciones reales. Tomando estos parámetros observados es viable implementar un sistema de video vigilancia autónomo que detecte a los infractores en tiempo real, pero debido a la emergencia sanitaria no es posible tener la opinión del encargado de los laboratorios.

1.9. Metodología de desarrollo de software.

El desarrollo del proyecto está adaptado la metodología incremental para desarrollo de software, basado en parámetros importantes como las redes neuronales [14] a fin de aprovechar eficientemente el tiempo disponible para la ejecución del mismo. El desarrollo incremental de la aplicación se presenta en cuatro fases:" Inicio, Elaboración, Construcción y Transición" [30].

En cada incremento se atribuyen hitos que permitirán referenciar el inicio y fin del mismo, de tal forma que se puedan establecer metas alcanzadas. Estos hitos incluyen las etapas de análisis, diseño, desarrollo o codificación y pruebas que han sido adaptadas para el despliegue de Redes Neuronales Artificiales [31]. Asimismo, a través estos hitos se puede determinar si se puede o no continuar con el siguiente incremento.

1.10. Etapas del desarrollo

- ✓ Incremento Uno

  En esta etapa se configura todo el entorno de desarrollo, integrando cada librería y módulo necesario para realizar la detección de objetos.
- ✓ Incremento Dos

En esta segunda etapa se añade un dataset con suficiente información para entrenar el modelo de detección de mascarillas

- ✓ Incremento Tres

  En la tercera etapa se desarrollará la aplicación para entorno de escritorio Linux, se integrará el módulo de detección de mascarillas.

- ✓ Incremento Cuatro

  Se desarrollará el módulo de reportes y se integrará al módulo de aplicación y detección. Se realizarán las pruebas de precisión para la detección de rostros sin mascarillas a fin de entregar un software de calidad.

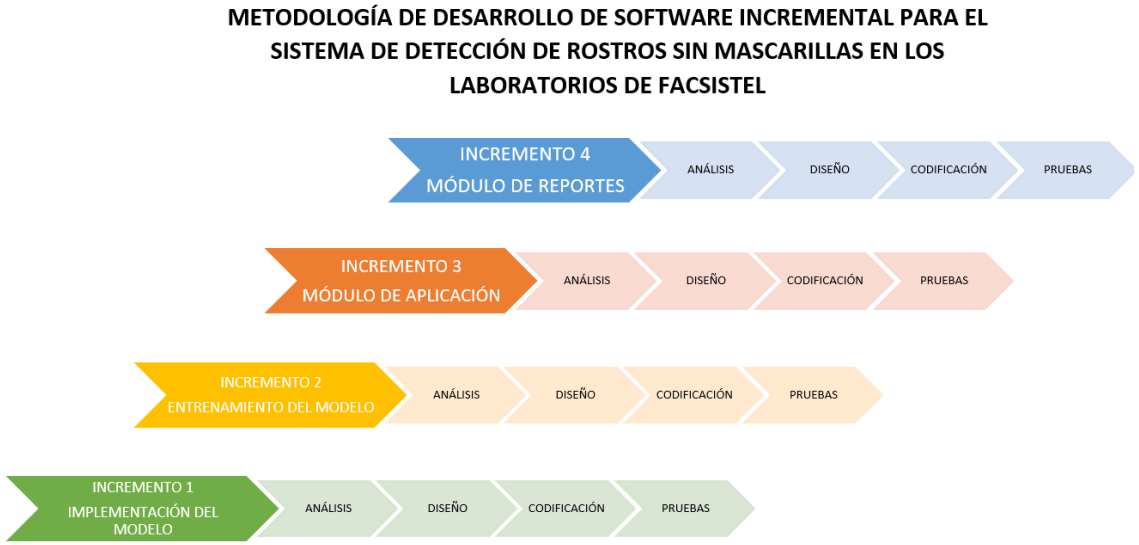


Fig. 1. Gráfico del modelo Incremental del sistema de detección de rostros sin mascarillas

CAPITULO II

2. LA PROPUESTA

2.1. Marco contextual

2.1.1. Universidad Estatal Península de Santa Elena

Universidad Estatal Península de Santa Elena, conocida por su acrónimo UPSE, es una universidad pública localizada en el cantón la libertad de la provincia de Santa Elena en la República del Ecuador, es el primer centro de enseñanza autónomo y la que cuenta con mayor población estudiantil de la zona [32]. La UPSE recibió oficialmente el día miércoles 28 de octubre de 2020, el CERTIFICADO DE ACREDITACIÓN del Consejo de Aseguramiento de la Calidad de la Educación Superior - CACES, luego de cumplir con los parámetros de calidad exigidos por este organismo [33].

Está ubicado en el cantón La Libertad de la Provincia de Santa Elena.

Fig. 2. Ubicación Geográfica de UPSE [34]

Misión

Formar profesionales competentes, comprometidos con la sociedad y el ambiente, sobre la base de una alta calidad académica, la investigación, la adopción y generación de conocimientos científicos y tecnológicos, respetando y promoviendo

nuestra identidad cultural para posibilitar un desarrollo humano integral, participativo, democrático, solidario y de respeto a los derechos humanos, como esencia de los valores de ética pública y combate a las condiciones de pobreza, mediante sus aportes a la producción, tecnología, economía y al desarrollo socio cultural [35].

Visión

Ser la Universidad referente en la zona marino-costera ecuatoriana, por sus competencias académicas de investigación científica y tecnológica y con espíritu innovador y crítico, así como por la responsabilidad social de sus autoridades, estudiantes, profesores/as, investigadores/as, servidores/as y trabajadores/as, en el marco de lo plurinacional y de la interculturalidad, con respeto a la diversidad, con ejercicio de la moral, la solidaridad y la tolerancia, y mediante la convivencia armónica con la naturaleza [35].

2.1.2. Facultad de Sistemas y Telecomunicaciones

La Facultad de Sistemas y Telecomunicaciones (FACSISTEL) es una de las siete Facultades que conforman el Alma Mater [9]. Fue creada el 22 de marzo de 2010 y en la actualidad el decanato está a cargo del Ing. Washington Torres, quien dirige por segunda ocasión y es el quinto Decano en línea de tiempo. La oferta académica de la Facultad de Sistemas y Telecomunicaciones consta de tres carreras: Tecnologías de la Información, Electrónica y Automatización, Telecomunicaciones [36].

Dentro de existen tres unidades de negocio principales, el Centro de Informática Universitario (CIU) que provee asesoría en términos de tecnología no sólo a la Universidad sino también a la comunidad en general. El Club de Emprendimiento Tecnológico (CET) es una incubadora de ideas con carácter colaborativo con el fin de crear oportunidades reales de negocio. Y la Academia CISCO ofrece certificaciones en Redes, Administración en Linux, Introducción a IoT e Introducción a Ciberseguridad [37].

2.1.3. Laboratorio de Informática de FACSISTEL

El Laboratorio de Informática está situado a un costado del edificio del Decanato de FACSISTEL. El edificio cuenta con tres salas de computación (Anexo 3) en los cuales se imparten cátedras referentes al desarrollo de software, análisis de datos entre otros. El aforo nominal es de aproximadamente 120 personas, 40 personas por sala, además, tiene dos baños y una oficina de administración. Tiene dos puertas en condición de entrada y salida, la puerta principal conecta el edificio con la Av. Universitaria. Este edificio no es de uso exclusivo para FACSISTEL, a su vez, es usado para eventos como exposiciones, capacitaciones de otras Facultades e incluso el Colegio UPSE.

2.1.4. Uso de laboratorios

Dentro de la Universidad Península de Santa Elena cada edificio, sala o departamento tiene un reglamento de uso, asimismo de deberes, derechos y obligaciones de quienes los usan. Tomando en cuenta este contexto se cita el REGLAMENTO PARA EL USO DE LABORATORIOS Y TALLERES DE LA UNIVERSIDAD ESTATAL PENÍNSULA DE SANTA ELENA [38].

Art. 8 Los laboratorios y talleres de la UPSE, son espacios que le permiten a las/los estudiantes poner en práctica los conocimientos adquiridos en el aula; además de la experimentación y la exploración necesarias para que éste conozca los límites y posibilidades que el área le ofrece en el marco de un complemento indispensable. Gracias a estas instalaciones las/los estudiantes llevarán a cabo trabajos y proyectos con las especificaciones y la calidad que les serán exigidas en su futuro profesional.

2.1.5. Uso de mascarillas en espacios públicos

Las mascarillas forman parte de una estrategia integral de medidas de bioseguridad para suprimir la transmisión y salvar vidas; el uso de una mascarilla no basta para proporcionar una protección adecuada contra el SARS-Cov.2 [3]. En la resolución emitida en abril de 2020 por parte del Comité de Operaciones de Emergencia Nacional, se dictamina “Disponer a los Gobiernos Autónomos Descentralizados Municipales, dentro del marco de sus competencias, emitan y aprueben una Resolución u Ordenanza Municipal que regule el uso de mascarillas / tapabocas en

espacios públicos" [2]. Hasta la actualidad es de carácter obligatorio el uso de mascarillas sobre todo en espacios públicos y de concurrencia masiva.

2.2. Marco Conceptual

2.2.1. Python 3.7

Python es un lenguaje de programación potente y fácil de aprender. Tiene estructuras de datos de alto nivel eficientes y un simple pero efectivo sistema de programación orientado a objetos [18]. Python es el lenguaje base para el desarrollo e implementación de este proyecto, posee módulos y librerías que facilitan operaciones básicas dentro de la codificación.

2.2.2. OpenCV 4.2

Open Source Computer Vision Library es una biblioteca de software de aprendizaje automático y visión artificial de código abierto. OpenCV fue construido para proporcionar una infraestructura común para aplicaciones de visión computarizada y para acelerar el uso de la percepción de la máquina en los productos comerciales. Al ser un producto con licencia BSD, OpenCV facilita a las empresas el aprovechamiento y la modificación del código [19].

2.2.3. PyQt5

Qt es un conjunto de bibliotecas y herramientas de desarrollo de C++ que incluye abstracciones independientes de la plataforma para interfaces gráficas de usuario, redes, hilos, expresiones regulares, bases de datos SQL, SVG, OpenGL, XML, configuración de usuario y aplicación, servicios de posicionamiento y ubicación, comunicaciones de corto alcance (NFC y Bluetooth), navegación web, animación 3D, gráficos, visualización de datos 3D e interconexión con almacenes de aplicaciones. PyQt5 implementa más de 1000 de estas clases como un conjunto de módulos Python. PyQt5 es compatible con las plataformas Windows, Linux, UNIX, Android, macOS e iOS [20].

2.2.4. SQLite3

Es una biblioteca de lenguaje C que implementa un motor de base de datos SQL pequeño, rápido, autónomo, de alta fiabilidad, con todas las funciones. SQLite es el motor de base de datos más utilizado en el mundo. SQLite está integrado en todos los teléfonos móviles y la mayoría de las computadoras y viene incluido dentro de innumerables otras aplicaciones que la gente utiliza todos los días [21].

2.2.5. TensorFlow

Es una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Cuenta con un ecosistema integral y flexible de herramientas, bibliotecas y recursos de la comunidad que les permite a los investigadores impulsar un aprendizaje automático innovador y, a los desarrolladores, compilar e implementar con facilidad aplicaciones con tecnología de AA [22].

2.2.6. Keras

Es una API de aprendizaje profundo escrita en Python, que se ejecuta sobre la plataforma de aprendizaje automático TensorFlow. Fue desarrollado con un enfoque en permitir la experimentación rápida [23].

2.2.7. ResNet

Las redes residuales son un conjunto de datos creados a partir del entrenamiento de redes neuronales profundas para la clasificación de objetos. Son más fáciles de optimizar y pueden obtener precisión de profundidad considerablemente mayor respecto a otras redes profundas [15].

2.2.8. Mobilenet

Es un conjunto de funciones que mejora el rendimiento de vanguardia de los modelos móviles en múltiples tareas y puntos de referencia, así como en un espectro de diferentes tamaños de modelos [16].

2.2.9. Google Colab

Colaboratory, también llamado "Colab", te permite ejecutar y programar en Python en tu navegador con las siguientes ventajas: no requiere configuración, da acceso gratuito a GPUs, permite compartir contenido fácilmente [24].

2.2.10. Anaconda

Es el lugar de nacimiento de la ciencia de datos de Python. Es una movimiento de científicos de datos, empresas basadas en datos y comunidades de código abierto [25].

2.2.11. CUDA

CUDA es una plataforma de computación paralela y un modelo de programación desarrollado por NVIDIA para computación general en unidades de procesamiento gráfico (GPU). Con CUDA, los desarrolladores pueden acelerar drásticamente las aplicaciones informáticas aprovechando la potencia de las GPU [26].

2.2.12. Entornos de desarrollo

Un entorno de desarrollo integrado (IDE) es un sistema de software para el diseño de aplicaciones que combina herramientas del desarrollador comunes en una sola interfaz gráfica de usuario (GUI). Los IDE permiten que los desarrolladores comiencen a programar aplicaciones nuevas con rapidez, ya que no necesitan establecer ni integrar manualmente varias herramientas como parte del proceso de configuración. [39].

2.2.13. Editores de código

Es un editor de texto que ayuda a escribir el código de software con funciones como el resaltado de la sintaxis con indicaciones visuales, el relleno automático específico del lenguaje y la comprobación de errores a medida que se escribe el código [39].

2.2.14. Visual Studio Code

Visual Studio Code es un editor de código redefinido y optimizado para crear y depurar modernas aplicaciones web y en la nube. Visual Studio Code es gratuito y cuenta con una gran variedad de extensiones que brindan ayudan al desarrollador [40]. Visual Studio Code combina la simplicidad de un editor de código con lo que

los desarrolladores necesitan para su ciclo principal de edición, compilación y depuración. Proporciona soporte completo para la edición, navegación y comprensión de código junto con depuración liviana, un modelo de extensibilidad rico e integración liviana con herramientas existentes [41].

2.2.15. Entornos virtuales

Un entorno cooperativamente aislado de ejecución que permite a los usuarios de Python y a las aplicaciones instalar y actualizar paquetes de distribución de Python sin interferir con el comportamiento de otras aplicaciones de Python en el mismo sistema [42]. Un entorno virtual es un directorio que contiene una colección específica de paquetes que se han instalado. Por ejemplo, puede tener un entorno con NumPy 1.7 y sus dependencias, y otro entorno con NumPy 1.6 para pruebas heredadas. Si cambia un entorno, sus otros entornos no se verán afectados. Puede activar o desactivar entornos fácilmente sin afectar el sistema [43].

2.2.16. Modelo-Vista-Controlador

"El patrón Modelo Vista Controlador, separa la parte gráfica de una aplicación, de los procesos lógicos y de los datos de la misma" [44].

Modelo

Es la representación específica de la información con la que se está operando en el instante de la ejecución de la aplicación, por lo tanto, es el principal moderador del flujo de datos de la aplicación. Cuando se requiere un almacenamiento de datos, por lo general se usa una Base de Datos, pero cuando no lo amerita la información se guarda en fichero [44].

Vista

La vista es la respuesta gráfica de la aplicación la cual interactúa directamente con el usuario [44].

Controlador

El controlador es el encargado de manipular el entorno, variables y objetos dentro del sistema [44].

2.2.17. Detección de objetos con OpenCV

OpenCV contiene varias librerías destinadas al tratamiento de imágenes y redes neuronales. Uno de los objetivos de OpenCV es el de proveer una infraestructura de visión artificial fácil de usar, que ayude a la gente a realizar aplicaciones sofisticadas de visión de una manera mucho más rápida y fácil [45]. Dentro de las funcionalidades principales de OpenCV están:

- ✓ Procesamiento, lectura y creación de imágenes y video.
- ✓ Detección de objetos y Características.
- ✓ Visión artificial estéreo o basada en Geometría Monocular.
- ✓ Fotografía.
- ✓ Aprendizaje profundo (redes neuronales) y clustering.
- ✓ Aceleración por GPU.

2.2.18. Clasificación de imágenes con Redes residuales (ResNet)

ResNet es una red convolucional profunda basada en bloques residuales desarrollada por Microsoft en 2015. Con esta arquitectura se batieron records en clasificación, localización y detección de objetos con una tasa de error de 3.57%, tomando en cuenta que las tareas en inteligencia artificial oscilan entre el 5% y 10%. Se demostró que a partir de cierta profundidad la red deja de aprender. La idea de ResNet es saltar cierta cantidad de capas manteniendo bloques residuales [46].

La arquitectura de la red se basa en reemplazar cada bloque de 2 capas en la red de 34 capas con este bloque de cuello de botella de 3 capas, lo que resulta en una ResNet de 50 capas (Fig. 46 Arquitectura de Resnet50 [15]Anexo 5).

2.2.19. Clasificación de imágenes con Mobilnet

Es una arquitectura de red neuronal específicamente diseñada para entornos móviles y con recursos limitados. Para llevar a cabo esto se utiliza la idea de un bloque utilizado en muchas arquitecturas de redes neuronales: Depthwise Separable Convolutions. Este consiste en reemplazar el operador convolucional por una versión factorizada que lo separa en varias capas [47].

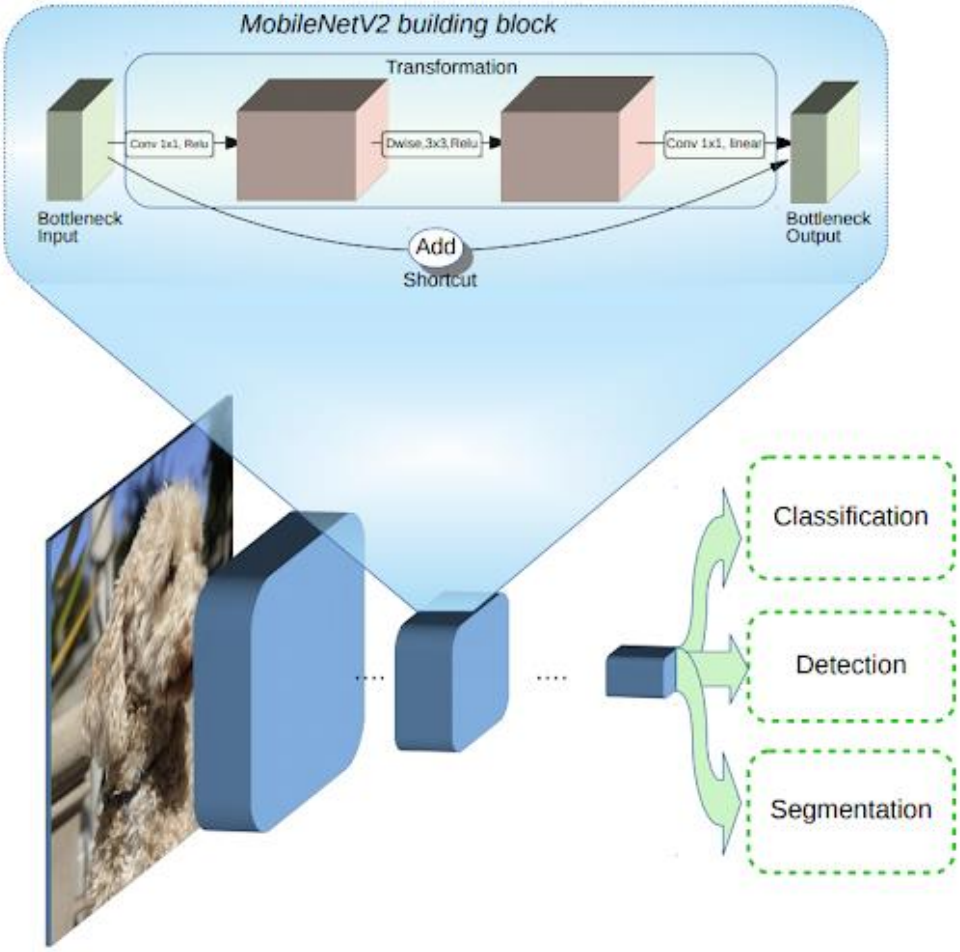


Fig. 3 Arquitectura de MobileNetV2 [48].

MobilenetV2 usa modelos pre entrenados de ResNet50 para la detección y clasificación de objetos según la configuración y clases proporcionadas por la red neuronal artificial. La saturación de los canales (Fig. 4) de cada imagen permite destacar las características de los elementos encontrados, de esta forma la imagen se convierte en un conjunto de datos que puede ser analizada y comparada con los pesos de la red (Fig. 5) para encontrar coincidencias entre los datos de la imagen y los guardados en estos pesos. Finalmente, la comparación devuelve un índice de confianza que permite clasificar los elementos de la imagen según la predicción de la red pre entrenada.

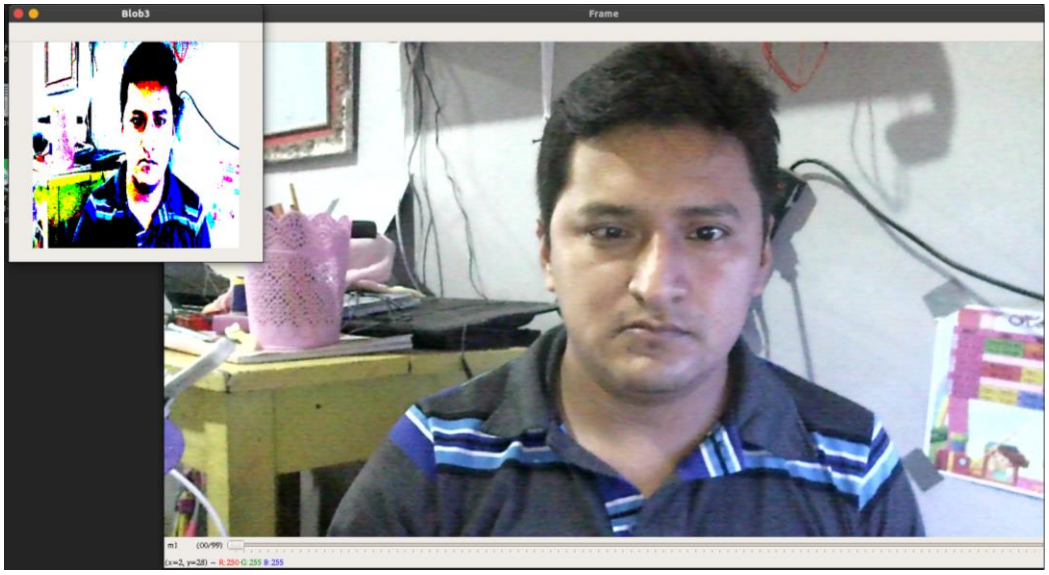


Fig. 4. Saturación de canales de una imagen

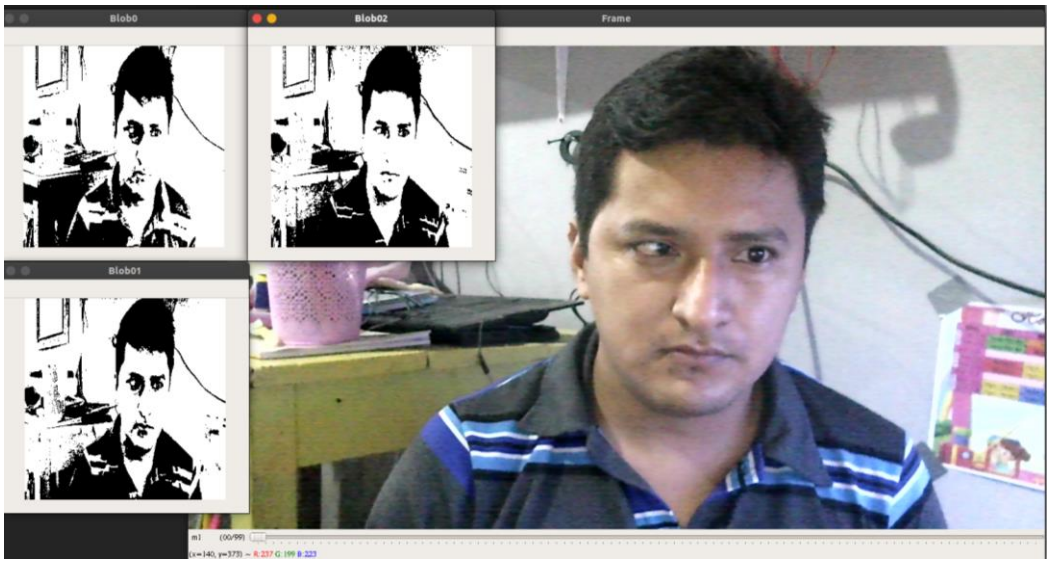


Fig. 5. Extracción de las características de una imagen

2.2.20. Entrenamiento de modelos con Tensorflow Keras

Keras es una API de alto nivel para compilar y entrenar modelos de aprendizaje profundo. tf.keras es la implementación de TensorFlow de esta API [49].

Entrenar el modelo de red neuronal requiere de los siguientes pasos:

1. Entregue los datos de entrenamiento al modelo. En este ejemplo, el set de datos de entrenamiento estan en los arreglos train_images y train_labels.
2. el modelo aprende a asociar imagenes y etiquetas.

3. Se le pide al modelo que haga predicciones sobre un set de datos que se encuentran en el ejemplo, incluido en el arreglo test_images. Se verifica que las predicciones sean iguales a las etiquetas del arreglo test_labels.

Para comenzar a entrenar, se llama al método *fit*, es llamado así porque permite ajustar el modelo a el set de datos de entrenamiento [50].

2.2.21. Arquitecturas de Visión por computador

Dentro de los sistemas de visión por computadora se identifican tres arquitecturas: sistemas de monitoreo continuo, sistemas de monitoreo periódica y sistemas integrales de visión. Según las necesidades del proyecto a realizar se puede escoger una de ellas [51].

Sistema de monitoreo periódico

Los sistemas que realizan una inspección utilizando secuencia de imágenes que no son totalmente continuas, reciben el nombre de sistemas de inspección periódica. Esta arquitectura es apropiada para aplicaciones en las que no se requiere una respuesta en tiempo real, ya que el tiempo de procesamiento y análisis no es crítico [51].

2.2.22. Componentes de un sistema de visón por computador

Las técnicas de visión por computador permiten extraer información del mundo real a partir de una imagen. Una de las etapas críticas es la adquisición de la imagen, parámetros como: la iluminación, la densidad de píxeles de la cámara, son la clave del buen funcionamiento de un sistema de reconocimiento y clasificación [52].

Cámara

Una cámara web posee un lente y un digitalizado con lo cual se obtiene una imagen digital que es enviada al computador para su posterior tratamiento. Las cámaras web ofrecen densidades de píxeles entre 360p y 1080p, así mismo, la calidad de la captura varía según la resolución de la cámara [52].

Iluminación

Una buena iluminación de la escena permite optimizar el contraste dela imagen, separando el objeto de interés del fondo. Una iluminación inadecuada crea

incertidumbre en la imagen, agrega error al proceso de clasificación, y eleva el tiempo de análisis de la imagen debido al ruido en la escena [52].

2.3. Marco Teórico

2.3.1. Importancia de Uso de Modelos de entrenamiento en la actualidad

El Sars-Cov-2 ha puesto en la palestra la fragilidad de la humanidad ante él, pues ha demostrado ser altamente resistente y peligroso, con un factor de propagación sin precedente alguno a escala global en tiempos modernos. Con base en la dinámica comportamental del Sars-Cov-2, se requieren soluciones prontas para el monitoreo, detección y diagnóstico de las enfermedades generadas por su causa, la Inteligencia Artificial plantea diversas opciones de hardware y software encaminadas para tal fin. En la actualidad la IA mediante el aprendizaje profundo de las redes neuronales artificiales juega un papel importante en la prevención y detección del Sars-Cov-2 con una precisión cercana al 90%. Dicho índice puede incrementarse cuando se entrena el sistema con una mayor cantidad de datos [4].

2.3.2. Detección de Objetos bajo herramientas libres

Paralelamente a los sucesos ocurridos durante la pandemia, se han realizado algunas contribuciones que permiten reducir la propagación del Sars-Cov-2 referentes al área de visión por computadora, generalmente empleando redes neuronales que permitan reconocer patrones y puedan ser aplicados para detección de objetos [5]. Bajo este panorama, ha venido tomando fuerza el desarrollo de software de código abierto, donde la inteligencia colectiva es el engranaje principal para obtener un programa de altas prestaciones, multipropósito en la mayoría de los casos [4].

2.4. Componentes de la propuesta

2.4.1. Módulos del sistema

Módulo de Aplicación

En este módulo se muestra información en tiempo real como: el número de infractores, el número de rostros detectados, el porcentaje de infracciones durante la jornada, una imagen en miniatura de las últimas infracciones, un video en tiempo real donde se enmarcan en rojo los rostros sin mascarillas. En este módulo se guarda un registro en la base de datos.

Módulo de Reportes

Este módulo se realizan las consultas a la base de datos, contiene 3 tipos de reportes según el rango de tiempo, estos son:

- ✓ **Diario:** se muestra información de las detecciones durante la jornada en rangos de una hora dentro de tres gráficos, porcentajes de infractores vs total de personas en un pastel, número de personas y número de infractores en barras y un histograma de las infracciones durante la jornada.
- ✓ **Últimos cinco días:** se muestran cinco histogramas sobrepuestos de las detecciones de las últimas cinco jornadas.
- ✓ **Comparar dos días específicos:** se muestras dos histogramas sobrepuestos de dos jornadas específicas.

2.4.2. Requerimientos funcionales

Código	Requerimiento	Prioridad
RF-01	El sistema permite visualizar un video en tiempo real, clasificar a los infractores y enmarcar dentro del video los rostros sin mascarillas.	Alta
RF-02	El sistema muestra información en tiempo real sobre la cantidad de rostros e infractores detectados.	Alta
RF-03	El sistema guarda el registro de detecciones en la base de datos local que contiene hora, fecha, número de infractores, total de rostros detectados.	Alta
RF-04	El sistema guarda una captura de las infracciones	Alta
RF-05	El sistema muestra una imagen en miniatura de las últimas diez detecciones	Alta
RF-06	El sistema muestra una segunda ventana con el video para monitoreo in situ	Media
RF-07	El sistema muestra reportes estadísticos según los rangos de tiempo: ✓ Diario ✓ Últimos cinco días ✓ Dos días específicos	Alta
RF-08	El control de fechas de reportes será controlado por un datepicker	Alta
RF-09	El calendario debe inhabilitar fechas superiores a la actual según el sistema	Alta
RF-10	El inicio y detención del video estará controlado por botones	Alta
RF-11	El sistema debe descargar datos en formato csv desde la vista de reportes	Alta

Tabla 1. Requerimientos Funcionales

2.4.3. Requerimientos no funcionales

Código	Requerimiento
RNF-01	El sistema debe contar con un modelo de clasificación de rostros sin mascarillas.
RNF-02	La cámara de video debe tener una resolución mínima de 720p
RNF-03	El sistema debe ser compatible para Linux en la distribución Ubuntu 18.04
RNF-04	La base de datos deber ser ligera y mediante archivo autónomo sin gestor.
RNF-05	Integridad, debe guardar información relevante
RNF-06	Seguridad, no tendrá credenciales de inicio de sesión
RNF-07	Usabilidad, las detecciones son autónomas y la intervención del usuario es mínima.
RNF-08	Escalabilidad, el modelo de clasificación puede ser reentrenado para aumentar la eficiencia.

Tabla 2. Requerimientos No Funcionales

2.5. Diseño de la propuesta.

2.5.1. Arquitectura del Sistema.

El proyecto propuesto se basa en la arquitectura Modelo-Vista-Controlador (MVC). El usuario interactúa visualmente con las vistas de la aplicación que son emitidas mediante requerimiento del controlador, este a su vez solicita información al modelo que conectado a la base de datos devuelve los datos requeridos, estos datos son enviados desde el controlador para actualizar las vistas.

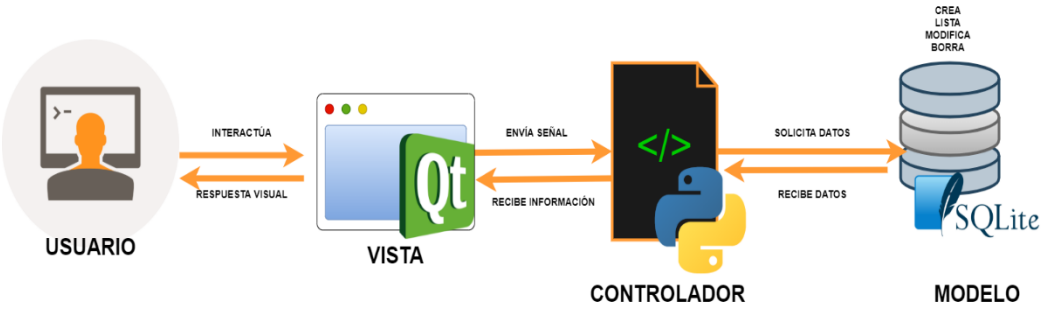


Fig. 6.Arquitectura del sistema

2.5.2. Modelo.

Para el desarrollo de este proyecto se considera el uso de SQLite [21] que es una base de datos ligera y basada en ficheros, no necesita de un gestor externo y su manejo no requiere de configuración adicional como puerto e IP.

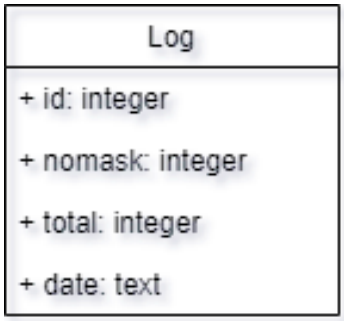


Fig. 7. Modelo de clase

En Fig. 7 se muestran los argumentos necesarios para la clase utilizada para almacenar los datos recolectados a partir del despliegue de la aplicación. Donde:

- id es el identificador del registro, es entero, único, auto incrementable y no puede ser nulo
- nomask es un valor entero que refiere al número de infractores detectados
- total es un valor entero que refiere al número de total de rostros detectados
- date es un valor de tipo texto que refiere a la hora y fecha del registro

Dentro del modelado de la Base de datos sólo se toma una tabla que servirá de bitácora para la aplicación, almacenando cada registro de infracciones detectadas. Para la creación de bases de datos SQLite con Python se hace uso de la librería PyQt5 y su módulo QtSql. Dentro del fichero models/db/db.py se codifican todas las funciones de creación de la conexión, inserción y consulta guardados en el archivo models/db/register.db. Para este fin se crearán objetos a partir de las clases QSqlDatabase que crea una nueva base de datos o la abre si ya existe y QSqlQuery que contiene y ejecuta la transacción.

2.5.3. Vista

La vista recepta las órdenes que el usuario emite respecto a la funcionalidad de sus componentes: botones, listas, tablas, etc. El proyecto contiene cuatro vistas distribuidas de la siguiente forma:

Pantalla Principal

Contiene la función principal, es decir, la renderización del video con las detecciones. Además, muestra información relevante respecto al número de infractores y total de rostros detectados. Asimismo, contiene imágenes en miniatura

de los últimos diez infractores y un menú superior que muestra opciones para una segunda pantalla y reportes.

Fig. 8. Mockup de la pantalla principal de la aplicación.

En la Fig. 8 se muestra la disposición de los elementos dentro de la vista. El desarrollo de la vista se realiza mediante la librería PyQt[20] que es una colección de clases y métodos que generan elementos en pantalla con diseños nativos del Sistema Operativo anfitrión. Estos elementos cuentan con una lista de funcionalidades por defecto, pero que también pueden ser personalizables según la necesidad del usuario. Los principales módulos usados dentro de PyQt son QtWidgets, QtGui, QtCore.

Fig. 9. Diseño final de la pantalla principal de la aplicación.

La Fig. 9 muestra el diseño final luego de la codificación de la vista. Los elementos de la vista están escritos en texto marcado XML dentro del archivo view/dashboard.ui y el comportamiento y funcionalidad del mismo están dentro del archivo control/ui.py.

Reporte Diario

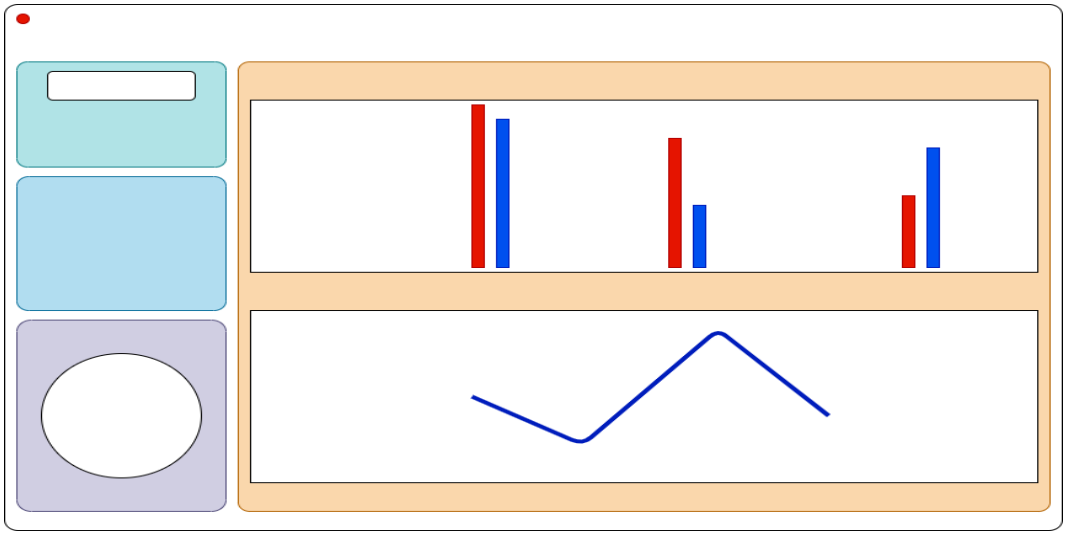


Fig. 10. Mockup de vista de reporte diario.

En la Fig. 10 se muestra el diseño preliminar de la vista de reporte. Esta vista será una ventana de diálogo que mostrará información acumulativa de las detecciones almacenadas en la base de datos.

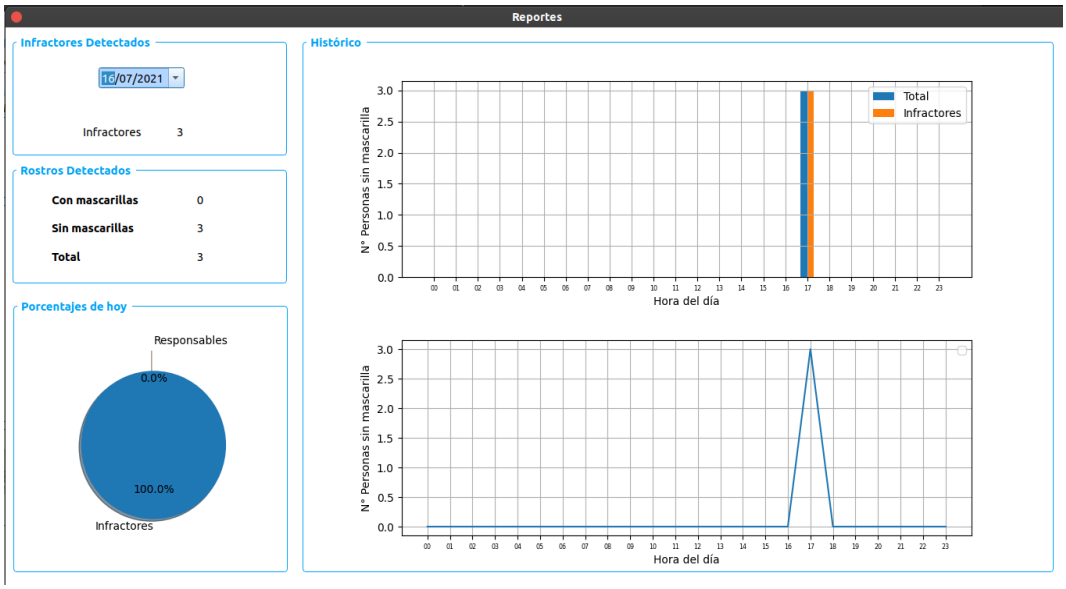


Fig. 11. Diseño final de la vista de reporte diario.

En la Fig. 11 se muestra la disposición final de los elementos en la vista de reporte diario dentro del fichero view/report.ui. Dentro de este cuadro de diálogo se encuentra un elemento interactivo que es el datepicker y cuyo comportamiento es controlado desde el fichero control/report.py.

Reporte Semanal

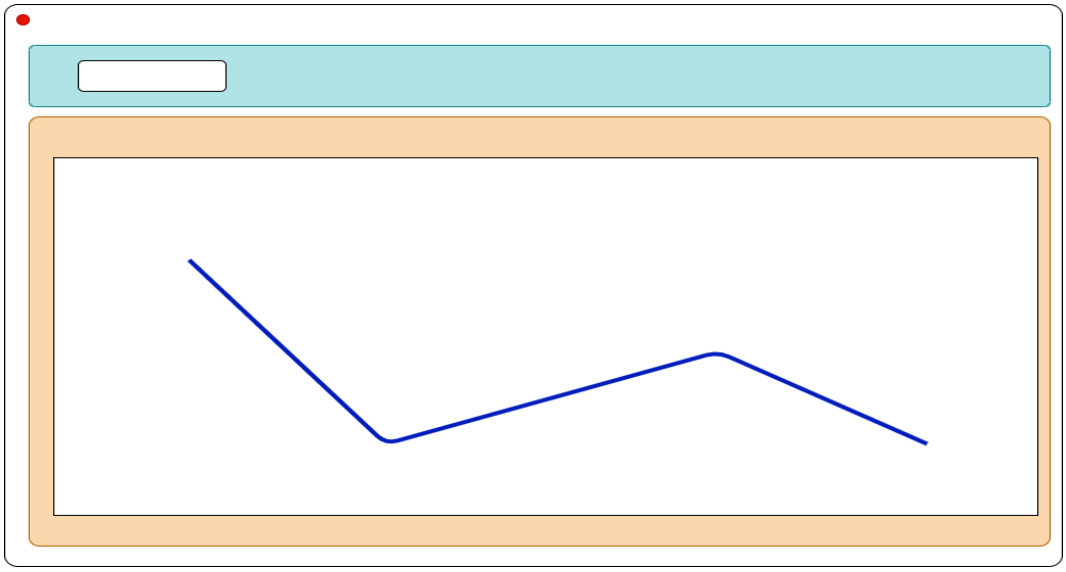


Fig. 12. Mockup de vista de reporte semanal

En Fig. 12 se muestra la disposición de los elementos de la vista de reporte semanal. Esta vista contiene una superposición de gráficos de líneas de los últimos cinco días de acuerdo a la fecha seleccionada en el datepicker.

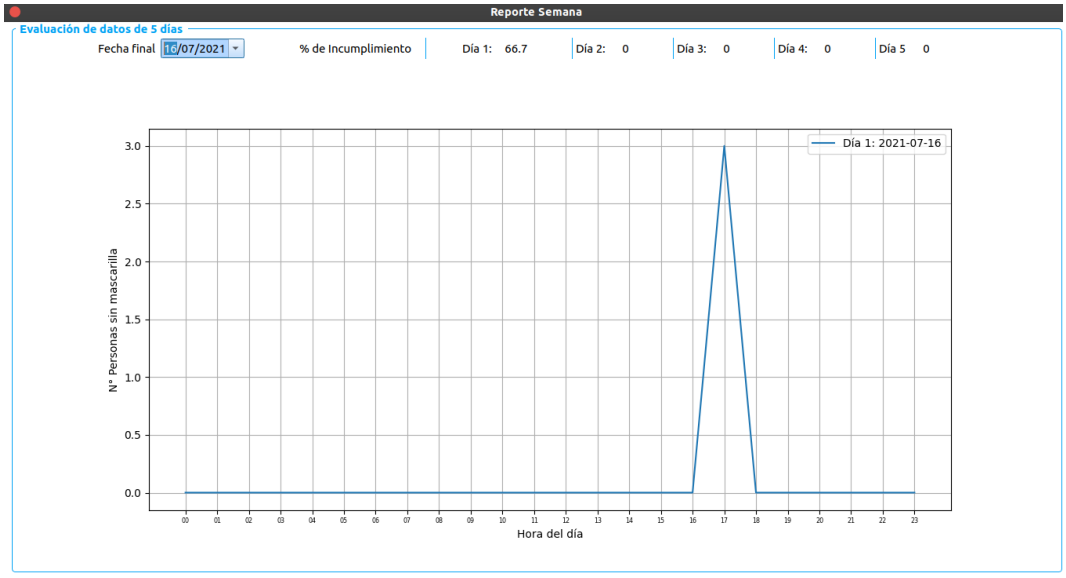


Fig. 13. Diseño final de la vista reporte semanal.

En Fig. 13 se muestra el diseño final de la vista reporte semanal el cual consulta información de un día particular y cuatro días previos según la selección realizada en el calendario. La codificación de esta vista se encuentra en el fichero view/week_report.ui y su comportamiento en el frichero controls/week_report.py.

Reporte de dos días

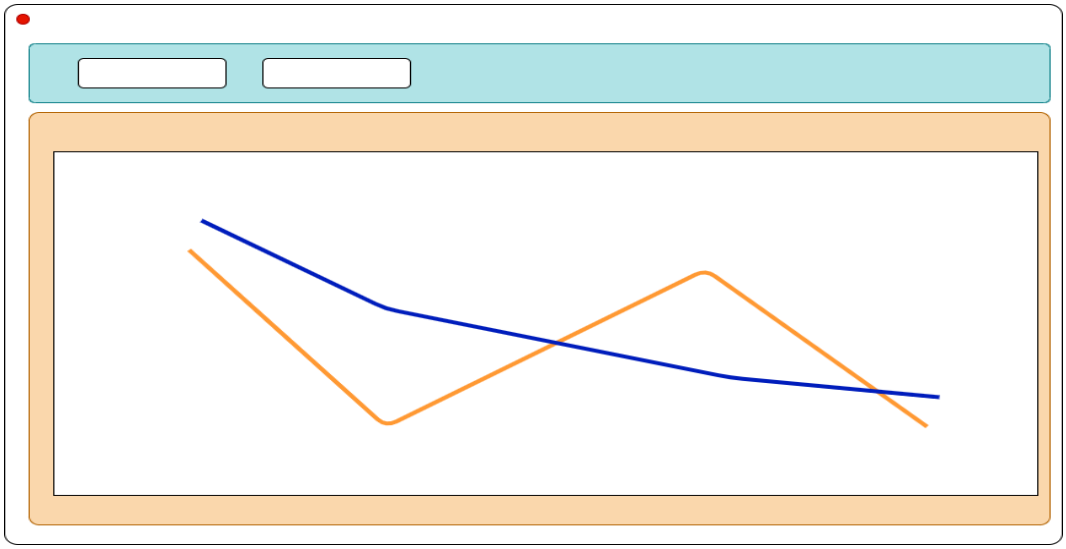


Fig. 14. Mockup de la vista reporte de dos días.

En la Fig. 14 se muestra el diseño preliminar de los elementos dentro de la vista reporte de dos días. En esta vista intervienen dos fechas en particular que no pueden ser las mismas y que al igual que los otros reportes no debe exceder de la fecha actual.

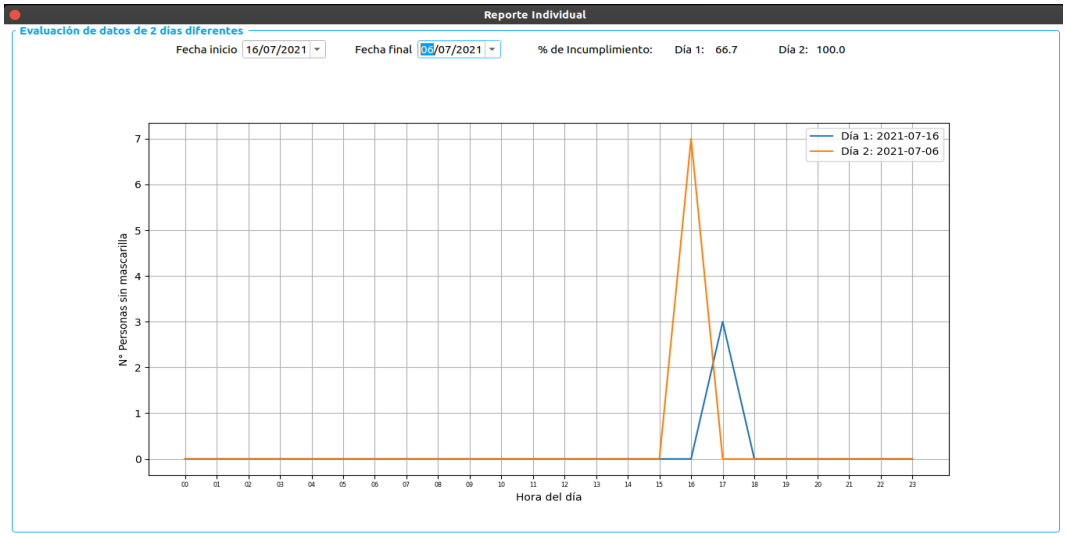


Fig. 15. Diseño final de la vista reporte de dos días.

En Fig. 15 se muestra l disposición final de los elementos dentro de la vista reporte de dos días. El código del diseño se encuentra dentro del fichero view/days_report.ui y la funcionalidad dentro del archivo controls/days_report.py.

2.5.4. Controlador

Dentro del directorio raíz del proyecto se encuentra una carpeta llamada *controls* que contiene los ficheros con toda la programación en Python. Cada archivo con extensión .py contiene el comportamiento, señales, acciones y respuestas de los elementos visuales de la aplicación, así como también los generadores y modificadores del modelo.

Caso de Uso

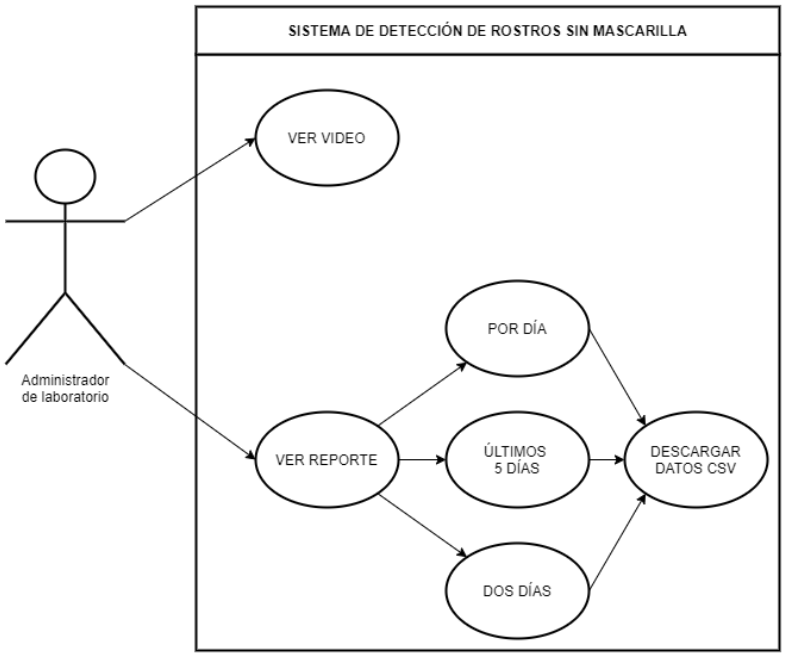


Fig. 16. Caso de Uso General

En Fig. 16 se muestra el Caso de Uso general de este proyecto. Se puede notar que el actor que en este caso es el encargado de los laboratorios tiene pocas interacciones con el sistema, esto se debe a que casi todos los procesos son autónomos y por ende no se necesita de supervisión para la ejecución de los mismos.

Diagrama de Flujo

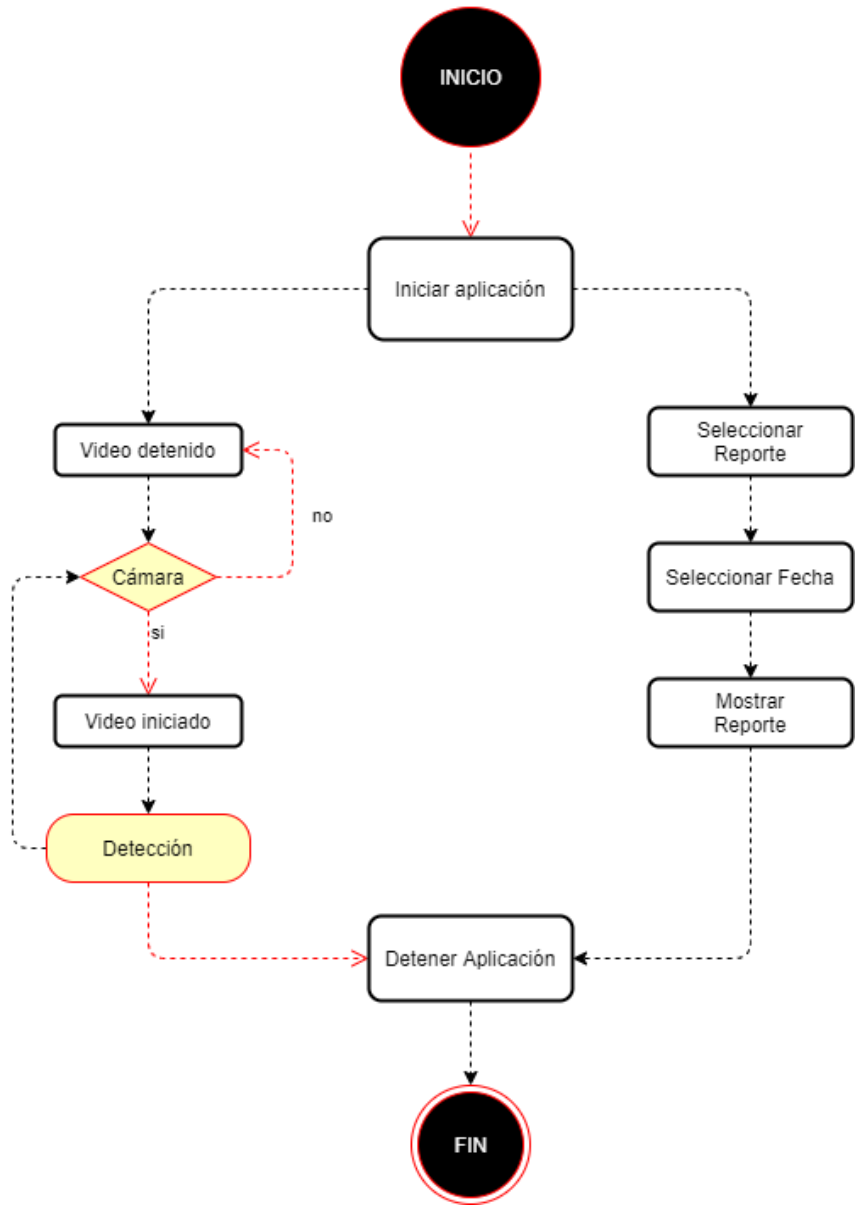


Fig. 17. Diagrama de Flujo del sistema

La figura anterior describe los procesos que se realizan dentro del sistema. Al iniciar la aplicación se ejecutan dos hilos de procesamiento paralelos que lo convierten en un sistema asíncrono. Esto permite que se pueda visualizar cualquiera de los 3 tipos de reportes mientras el video sigue renderizando y el modelo sigue realizando detecciones. La aplicación puede cerrarse en cualquier momento y no se necesita detener los hilos de procesamiento para realizar esta acción.

Diagrama de Flujo del proceso de detección

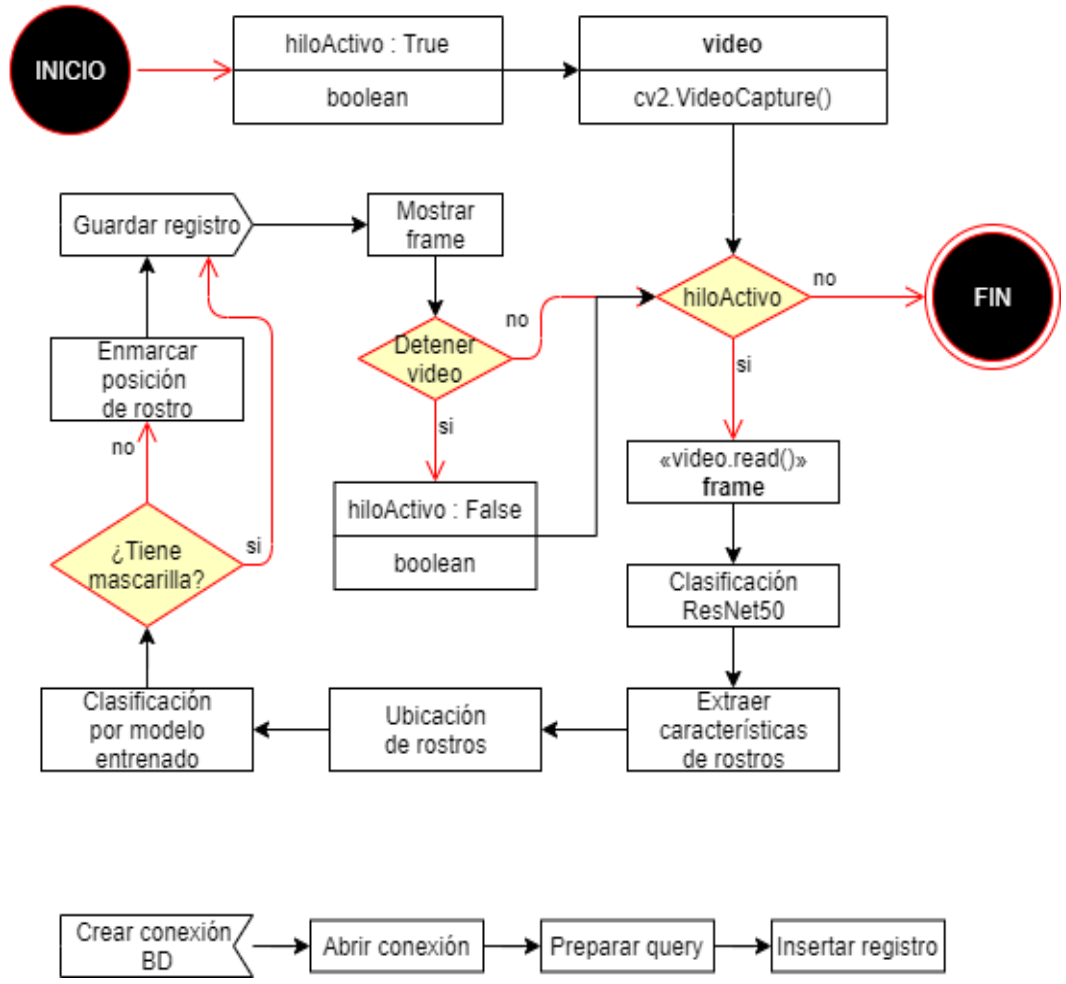


Fig. 18. Diagrama de Flujo de detección.

En Fig. 18 se detalla el flujo del proceso de detección dentro del sistema, donde se puede notar que una vez iniciado se inicializan dos variables, un booleano en verdadero y un objeto de la clase VideoCapture del módulo OpenCV. Estas variables permanecen activas durante todo el proceso y se actualizan con cada iteración. La detección se ejecuta dentro de un ciclo infinito donde hiloActivo toma el papel de bandera para terminar con el proceso. La captura de infractores se realiza mediante la extracción de características relevantes de la imagen, para obtener la ubicación de los rostros y luego ser analizados por el modelo entrenado. Adicionalmente se ejecuta un proceso paralelo para guardar el registro en la base de datos que se activa mediante una señal.

2.6. Estudio de Factibilidad.

Presupuesto el sistema de detección de rostros sin mascarillas				
Componente	**Costo $**	**Cantidad**	**Subtotal**	**Total**
Personal				**$ 4250**
Analista	700	1	1400	
Programador	600	1	1800	
Documentador	350	1	1050	
Hardware				**$ 745**
PC Core i3 6ta Gen. 8RAM Nvidia GT710	700	1	700	
Webcam 1080p	45	1	45	
Software				**$ 300**
Python 3.7	0	1	0	
Open CV 4.2	0	1	0	
PyQt5	300	1	*300	
SQLite3	0	1	0	
Tensorflow	0	1	0	
Google Colab	0	1	0	
Anaconda	0	1	0	
Varios				**$ 195**
Energía eléctrica	10	1	30	
Internet	35	1	105	
Transporte	20	1	60	
TOTAL				**$ 5490**

Tabla 3. Presupuesto

La Tabla 3 muestra el detalle del presupuesto requerido para el desarrollo de este proyecto con un tiempo de ejecución de tres meses (Anexo 12). Según las categorías descritas dentro del presupuesto se analiza lo siguiente:

Personal

El costo de personal se considera en $0 pues todas las etapas de desarrollo serán analizadas, diseñadas y codificadas por el tesista.

Hardware

El costo de hardware es de $745 y corresponden a los equipos utilizados en el desarrollo de la aplicación. Este valor es cubierto por el tesista, por lo tanto, el costo final es de $0. El despliegue del proyecto requiere equipos de similares características que los utilizados para el desarrollo, pero se utilizará un equipo de la oficina de administración de los laboratorios por ende el costo de implementación es $0.

Software

El costo de software es de $300 y este valor es referencial al pago de licencia por PyQt5. Este valor es condicional, pues al ser Licencia GNU GPL se debe cancelar un valor anual o liberar el código fuente en un repositorio público. En este sentido el código de generación de vistas será público, por lo tanto, el costo total de Software es $0.

Varios

El total de gastos por energía eléctrica, transporte e internet es de $195. Este valor es cubierto por el tesista, por ende, el costo por varios es $0.

2.6.1. Factibilidad Técnica

Los requerimientos técnicos para el desarrollo del proyecto se detallan en la siguiente tabla:

REQUERIMIENTOS TÉCNICOS		
CANTIDAD	**DESCRIPCIÓN**	**ESTADO**
1	Procesador Intel i3 6G	Disponible
1	Memoria RAM 8GB	Disponible
1	HDD 500GB	Disponible
1	Tarjeta gráfica Nvidia GT710	Disponible
1	Cámara Web 720p	Disponible

Tabla 4. Requerimientos técnicos

El análisis de la Tabla 4 denota que el proyecto es factible desde el punto de vista técnico debido a que cuenta con disponibilidad de todos los requerimientos.

2.6.2. Factibilidad Operativa

El sistema no requiere de la intervención del usuario final dentro del proceso principal que es la detección de rostros sin mascarillas, el flujo de datos de dicho proceso es autónomo al igual que el registro de la base de datos. Existen micro procesos dentro del sistema que requieren de una intervención mínima del usuario final, como el inicio y fin del video, menú, imágenes en miniatura, selección de fecha para reportes. Por lo tanto, es factible y no se requiere una capacitación exhaustiva para el uso de este sistema, tan solo una inducción al uso del mismo.

2.6.3. Factibilidad Económica

De acuerdo con el análisis del presupuesto, el proyecto "Implementación de software para la detección de rostros sin mascarillas mediante el entrenamiento de un modelo de inteligencia artificial y visión por computadora en los laboratorios de informática" se considera factible tomando en cuenta que el costo total del desarrollo estará a cargo del desarrollador del proyecto.

Costo de Implementación

Los valores detallados en Tabla 3 hacen referencia al costo total de desarrollo, por lo tanto, el costo de instalación tiene otros rubros que serán considerados a partir de la Tabla 4. La descripción del hardware utilizado en el desarrollo es el recomendado para el despliegue por la capacidad de procesamiento que se necesita. La siguiente tabla detalla los requerimientos mínimos, recomendados e idóneos para la implementación de este proyecto, asimismo se muestran los costos de adquisición de los componentes.

REQUERIMIENTOS TÉCNICOS PARA IMPLEMENTACIÓN		
CANTIDAD	**DESCRIPCIÓN**	**COSTO**
REQUERIMIENTOS MÍNIMOS		
1	CPU Tercera Generación: • Procesador Intel i3 3G • Memoria RAM 4 GB • HDD 120 GB • Tarjeta gráfica Nvidia GT710	**$ 350**

	• Cámara Web 720p	
REQUERIMIETOS RECOMENDADOS		
1	CPU Sexta Generación: • Procesador Intel i3 6G • Memoria RAM 8 GB • HDD 500 GB • Tarjeta gráfica Nvidia GT710 • Cámara Web 720p	**$ 650**
REQUERIMIENTOS ÓPTIMOS		
1	CPU Décima Generación: • Procesador Intel i7 10G • Memoria RAM 32 GB • SSD 980 GB • Tarjeta gráfica Nvidia RTX2080Ti • Cámara Web 1080P	**$ 1700**

Tabla 5. Valores de Implementación

2.7. El Modelo de Detección

2.7.1. Entorno virtual de trabajo

Un entorno virtual de trabajo permite encapsular un repositorio de módulos, librerías y dependencias de versiones específicas sin crear conflictos con el sistema operático anfitrión u otros entornos instalados; por ejemplo, un entorno X puede tener como lenguaje de programación Python 3.5 con Tensorflow 1.5 y el entorno Y puede tener Python 3.7 con Tensorflow 2.1, Tensorflow-GPU 2.1 y CUDAToolKit 10 y no existirían errores de dependencias, siempre y cuando se trabaje dentro del entorno correcto.

En este proyecto se crea un entorno virtual con Python 3.7 mediante Anaconda para instalar las dependencias necesarias para el uso de visión por computadora. El sistema operativo anfitrión es Ubuntu 20.04 [53] que es la distribución de Linux más comercial dentro de los entornos de escritorios. Para este propósito es necesario cumplir con los siguientes requerimientos:

a) Comprobar la disponibilidad de GPU Nvidia dentro de la computadora mediante el comando bash ***nvidia-smi***. El resultado es una tabla con la descripción general de la tarjeta de video conectada.

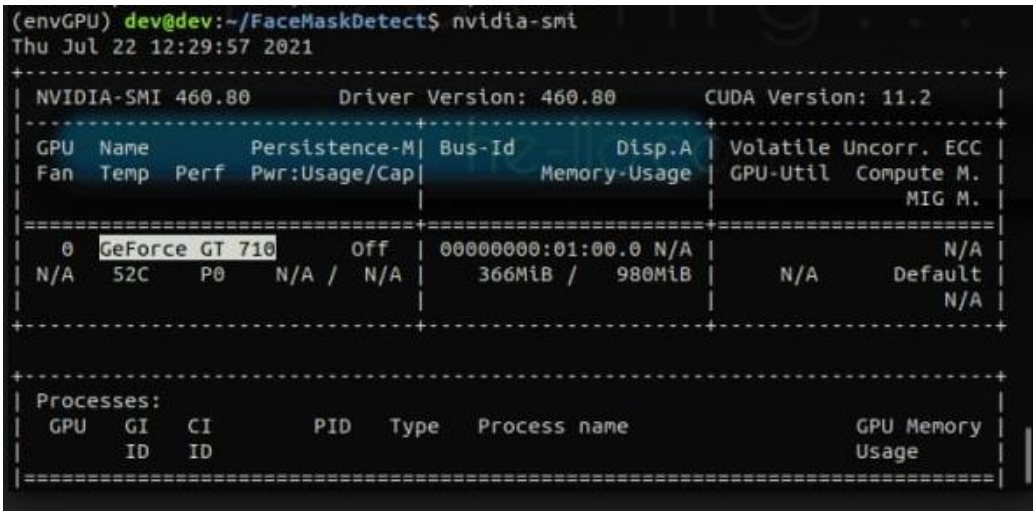


Fig. 19. Descripción de la GPU instalada.

b) Instalar CUDA mediante el comando ***sudo apt install nvidia-cuda-toolkit***.

c) Comprobar la versión de CUDA con el comando ***nvcc –version***.

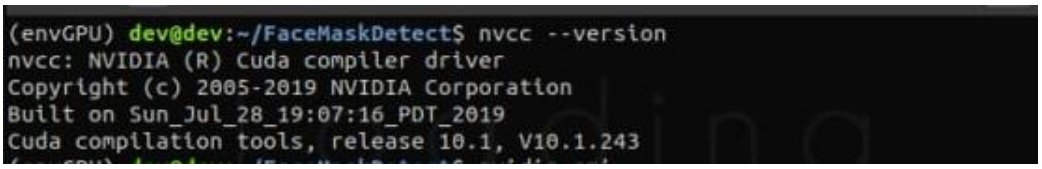


Fig. 20. Versión de CUDA

d) Instalar Anaconda par Ubuntu desde el sitio oficial.

e) Crear un entorno virtual con el comando ***conda create -n envGPU anaconda python=3.7.7***

f) Para trabajar dentro de un entorno es necesario activarlo mediante consola con ***conda activate envGPU.***

g) Instalar dependencias desde conda.
conda install tensorflow-gpu==2.1.0 cudatoolkit=10.1

h) Instalar dependencias desde pip.
pip install tensorflow==2.1.0 keras==2.3.1 numpy opencv-python pyqt5 matplotlib

i) Comprobar la vinculación de la GPU usando el método gpu_device_name del módulo test de Tensorflow. Para este paso se necesita entrar al intérprete de Python desde la consola en importar Tensorflow, como resultado se muestra la información de la GPU instalada.

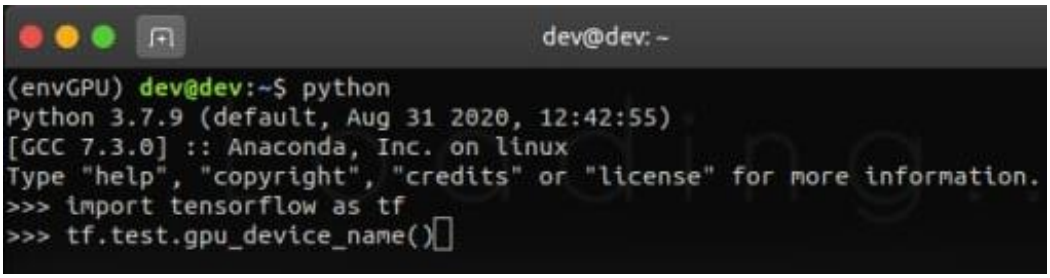


Fig. 21. Comprobación de vinculación de GPU.

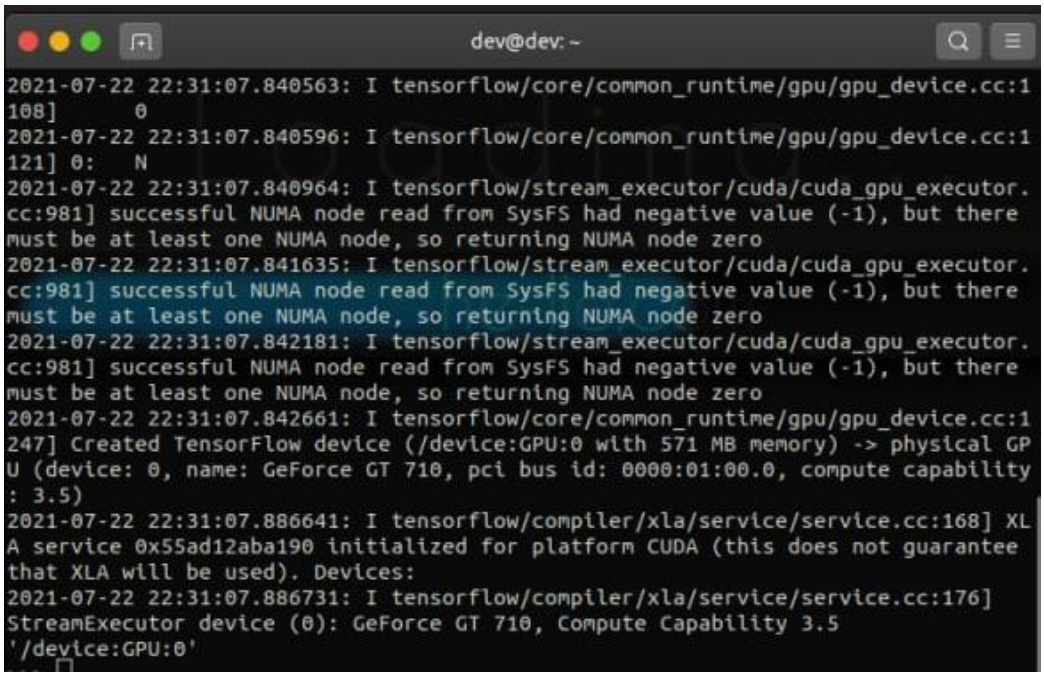


Fig. 22. Información de la vinculación de GPU.

Luego de este procedimiento cada ejecución de un fichero Python dentro de este entorno contará con la aceleración por GPU que provee Nvidia y CUDA.

2.7.2. Análisis del modelo

Para este proyecto se desarrolla un modelo de inteligencia artificial con pesos capaces de identificar un rostro sin mascarilla a partir de clasificación de objetos que provee ResNet50. La implementación de este proyecto se realizará en equipos de escritorio comunes y sin hardware especializado para despliegues de proyectos

con alto nivel de procesamiento, debido a lo expuesto se elige MobilenetV2 como la base de la red neuronal, por su optimización de recursos y su alto índice de confianza.

El entrenamiento de una red neuronal basada en MobilenetV2 requiere de dos componentes principales, un conjunto de datos o *dataset* y los pesos pre entrenados de *ImageNet*. El *dataset* usado en este proyecto corresponde a dos carpetas llamadas with_mask y *without_mask*, cada una contiene 1900 imágenes de rostros con mascarilla y sin mascarilla respectivamente. Los pesos de *ImageNet* [54] son descargados automáticamente durante el entrenamiento, esto gracias a *Tensorflow* y su módulo *Keras*.

La arquitectura prevista para la red neuronal combina las cabeceras de MobilenetV2 con 128 neuronas de análisis adicionales conectadas todas entre sí, y una salida con dos neuronas de decisión una por cada clase creada a partir del *dataset*, es decir una neurona para *with*_mask y otra para *without_mask*. Entre las capas de entrada y salida de la red neuronal se encuentra una capa con característica de 50% de deserción es decir que durante el entrenamiento la mitad de las neuronas serán obviadas aleatoriamente y las restantes deben intervenir para realizar las predicciones sin dañar la arquitectura de la red.

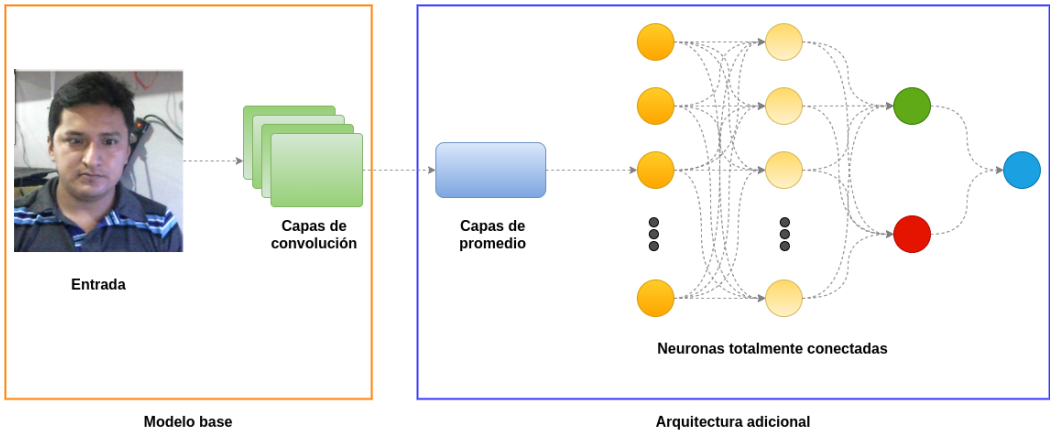


Fig. 23. Arquitectura de la red para detección de mascarillas.

El script de entrenamiento es desarrollado en Python y debido a la gran capacidad de procesamiento se usa Google Colab con procesamiento en GPU, teniendo como resultado un modelo con los pesos del entrenamiento en formato h5 correspondiente

a redes profundas y que puede ser usado junto a OpenCV para la detección en imágenes y videos.

2.7.3. Entrenamiento del modelo

El entrenamiento del modelo está divido en dos etapas: la primera corresponde a la codificación de un script en lenguaje Python que contenga los módulos necesarios para desarrollar la arquitectura de la red, compilar y guardar los pesos obtenidos a partir del análisis del *dataset*, adicionalmente se imprime por consola el proceso del entrenamiento por cada época y finalmente se guarda un fichero de formato h5 y una imagen con un histograma del nivel de confianza obtenido por el modelo. La segunda etapa corresponde a la ejecución del script en Google Colab y la revisión del nivel de confianza.

Codificación del modelo

Se crea un fichero con extensión .py con nombre intuitivo, en este caso es *train_mask_detector.py*. La codificación inicia con la importación de los módulos de Tensorflow, Keras contiene los métodos y clases necesarios para construir y entrenar la red neuronal. Se declaran los valores de tasa de aprendizaje, número de épocas y tamaño de lote y se define la ruta del *dataset*. Se recorre el directorio para examinar cada imagen y convertirla en un conjunto de datos que serán añadidos a una lista de nombre data, de igual forma se llena otra lista de nombre *labels* que contiene la clase correspondiente a cada imagen vectorizada.

Mediante el método *train_test_split* del módulo de *sklearn*[55] es posible muestrear las listas generadas anteriormente, esto con el fin de tener un porcentaje representativo de imágenes de pruebas que serán usadas para evaluar el nivel de confianza del modelo luego de su entrenamiento. Para este caso el 20% de imágenes serán utilizadas para evaluar el modelo. La clase MobileNetV2 del módulo *Keras* de *Tensorflow* es usado para generar la base de la red neuronal, requiere de dos argumentos importantes que son: los pesos de *imagenet* y la entrada de imágenes con dimensión de 224x224 pixeles como se muestra en Fig. 24.

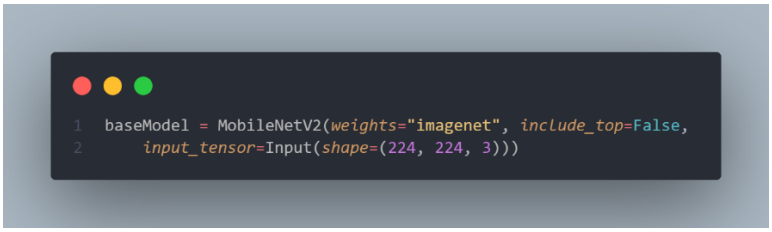
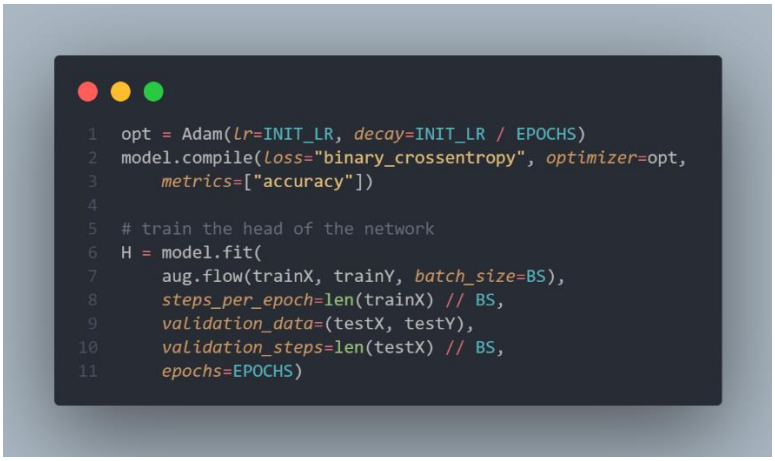


Fig. 24. Código de modelo base en MobilenetV2.

La arquitectura de la red neuronal mostrada en Fig. 23 indica que el modelo base corresponde sólo a la entrada, la salida contiene 128 neuronas interconectadas y 2 neuronas de decisión. Este proceso se realiza mediante el uso de la clase Dense del módulo *Keras*. Adicionalmente se requiere de la clase *Dropout* para inhabilitar aleatoriamente el 50% de las neuronas durante el entrenamiento. Para unir tanto las entradas y salidas en una sola red se usa la clase *Model* como se muestra en la siguiente figura.

Fig. 25. Código de construcción de la red neuronal

El siguiente paso es la compilación del modelo y para esto se definen argumentos que serán parte de toda la configuración previa al entrenamiento. Entre estos argumentos destacan el número de épocas que para este proyecto es de cien y el optimizador de tipo Adam que provee de tasas de aprendizajes variables para cada neurona. Luego de configuración se procede con el entrenamiento que está a cargo del método *fit* que recibe el conjunto de datos de las imágenes que serán iteradas durante las épocas definidas. Este método retorna un vector que contiene todo el historial del entrenamiento y que sirva para generar histogramas a partir del nivel de confianza adquirido en cada época.



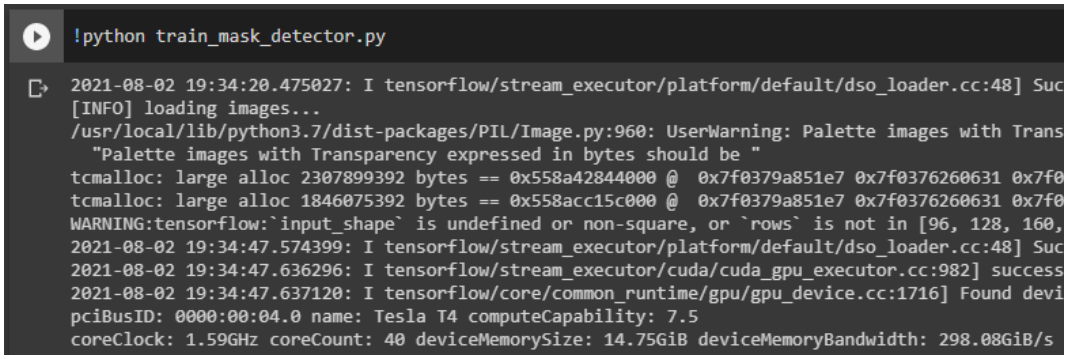
```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
    metrics=["accuracy"])

# train the head of the network
H = model.fit(
    aug.flow(trainX, trainY, batch_size=BS),
    steps_per_epoch=len(trainX) // BS,
    validation_data=(testX, testY),
    validation_steps=len(testX) // BS,
    epochs=EPOCHS)
```

Fig. 26. Código de compilación de la red neuronal

Finalmente, el modelo entrenado se guarda con el método *save* con el formato h5. Para este proyecto se usa *Matplotlib* para generar un reporte del historial del entrenamiento tomando en cuenta los valores de confianza y pérdida de validación.

Ejecución del entrenamiento

El entrenamiento del modelo se realiza en Google Colab con entorno de procesamiento para GPU esto permite analizar una gran cantidad de datos en menor tiempo en comparación con el equipo de desarrollo. Para este proceso se crea un nuevo entorno dentro de Colab donde se importa el *dataset* y el archivo *train_mask_detector.py*. La ejecución del fichero se realiza con el comando *!python train_mask_detector.py* dentro del directorio raíz como se muestra en la siguiente figura.



```
!python train_mask_detector.py

2021-08-02 19:34:20.475027: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Suc
[INFO] loading images...
/usr/local/lib/python3.7/dist-packages/PIL/Image.py:960: UserWarning: Palette images with Trans
  "Palette images with Transparency expressed in bytes should be "
tcmalloc: large alloc 2307899392 bytes == 0x558a42844000 @  0x7f0379a851e7 0x7f0376260631 0x7f0
tcmalloc: large alloc 1846075392 bytes == 0x558acc15c000 @  0x7f0379a851e7 0x7f0376260631 0x7f0
WARNING:tensorflow:`input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160,
2021-08-02 19:34:47.574399: I tensorflow/stream_executor/platform/default/dso_loader.cc:48] Suc
2021-08-02 19:34:47.636296: I tensorflow/stream_executor/cuda/cuda_gpu_executor.cc:982] success
2021-08-02 19:34:47.637120: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1716] Found devi
pciBusID: 0000:00:04.0 name: Tesla T4 computeCapability: 7.5
coreClock: 1.59GHz coreCount: 40 deviceMemorySize: 14.75GiB deviceMemoryBandwidth: 298.08GiB/s
```

Fig. 27. Ejecución del entrenamiento en Google Colab

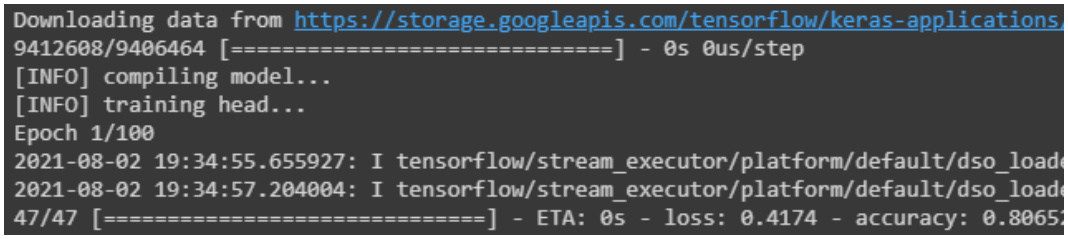


Fig. 28. Valor de confianza en la época uno.

En Fig. 28 se muestra la descarga de los pesos de *ImageNet* y el valor de la confianza el inicio del entrenamiento, es decir en la época uno tiene un valor de 80% y una pérdida de 41%. Esto valores se interpretan como el aprendizaje de la red durante las iteraciones de las épocas, la red aprende a medida que avanza en cada paso del *dataset* [5].

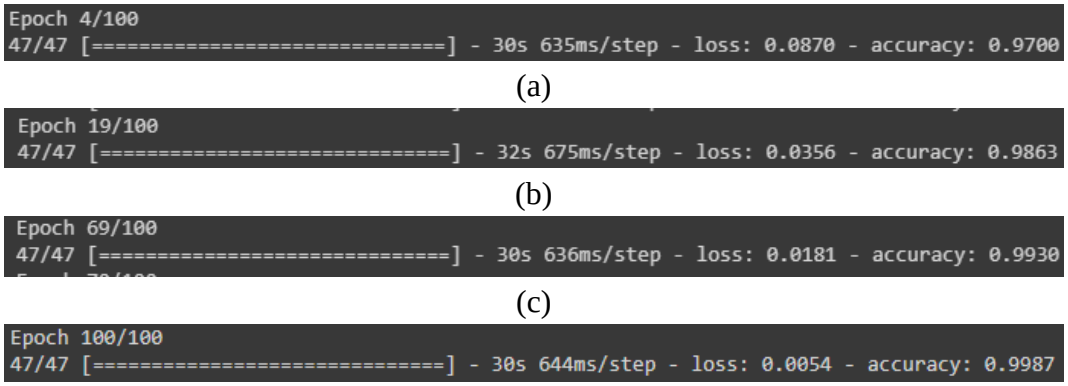


Fig. 29. Valores de confianza del modelo en: (a) época 4; (b) época 19; (c) época 69; (d) época 100.

Una vez finalizado el entrenamiento se muestras los resultados del análisis de la red con los datos de prueba.

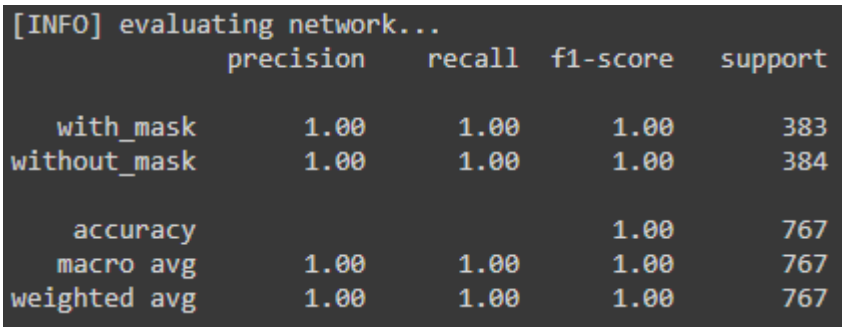


Fig. 30. Datos finales del entrenamiento del modelo.

En Fig. 30 se muestra el porcentaje de confianza obtenido al evaluar 383 imágenes con la clase *with_mask* y 384 con *without_mask*, dando como resultado una precisión del 100%.

2.7.4. Implementación del modelo

Luego del entrenamiento del modelo ya es posible aplicarlo para la detección de rostros sin mascarillas y es en ese orden como se debe programar, primero detectar rostros en una imagen y luego verificar si posee o no mascarillas. La implementación del modelo se lo realiza mediante código Python en un archivo llamado *test.py* estructurado de la siguiente forma:

Importación de métodos

El procesamiento de imágenes depende los métodos *img_to_array* y *preprocess_input* del módulo Keras, el análisis del modelo entrenado está a cargo de *load_model* también perteneciente a Keras. Numpy, OS y OpenCV son las importaciones restantes encargadas de generar arreglos de datos, manipulas el sistema operativo y renderizar video respectivamente.

Cargar modelos

El método dnn correspondiente a OpenCV permite cargar un modelo preentrenado basado en redes profundas, en este caso se usa ResNet para hacer la detección de los rostros y es almacenado en una variable con nombre *faceNet*. La variable *maskNet* contiene la red entrenada cargada mediante el método *load_model*.

Detecciones

El proceso de detección tiene varias tareas como se muestra en Fig. 31 y cuyo funcionamiento es el siguiente: la imagen capturada desde la lectura del video recibe saturación de los canales obteniendo un blob o conjunto de datos con las principales características resaltadas, luego este blob es enviado a la red profunda pre entrenada, capaz de detectar rostros dentro de una imagen devuelve las coordenadas de los mismos. Estas coordenadas permitirán crear regiones de interés (ROI) y almacenarlos en un arreglo para luego ser analizados por el modelo entrenado siendo este último quién retorna el nivel de confianza para cada ROI dentro de la imagen. Finalmente es posible validar la precisión y enmarcar en rojo los rostros sin mascarillas.

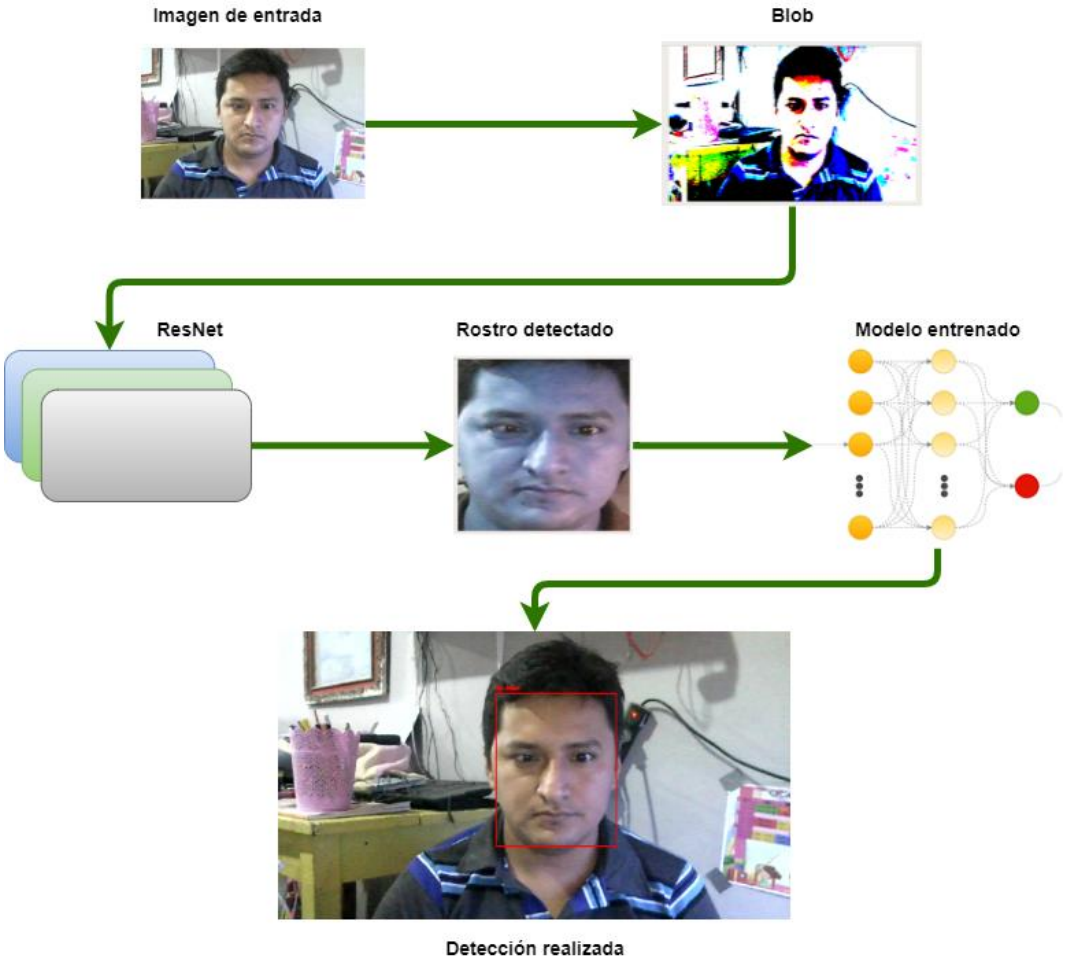


Fig. 31. Proceso de detección

La clase *VideoCapture* del módulo OopenCV permite crear un objeto capaz de capturar los fotogramas de un video mediante la función *read*. Luego se genera el blob mediante la función *blobFromImage* del módulo dnn, el blob entra a faceNet mediante el método *setInput* y devuelve las detecciones de rostros con *forward*. Las detecciones se almacenan en un arreglo de datos que contiene el nivel de confianza del objeto detectado y las coordenadas de ubicación dentro del fotograma. El siguiente paso es iterar el arreglo desestructurando la confianza y la posición del rostro detectado.

La validación del nivel de confianza permite asegurar la precisión al momento de enmarcar una infracción a fin de evitar falsos positivos. La posición permite recortar las regiones de interés y procesarlas con los métodos *img_to_array* y *preprocess_input* para luego ingresarlas al modelo entrenado y recibir las predicciones. El arreglo de predicciones contiene valores de confianza para las dos

clases y mediante una validación del mayor valor se obtiene la detección final. En el caso de ser infractor se enmarca en un rectángulo en rojo con la función *rectangle* de OpenCV utilizando las coordenadas obtenidas anteriormente. Si hubiese varios infractores basta con iterar la lista de predicciones junto a la lista de coordenadas para enmarcar a los rostros correspondientes.

2.8. La aplicación de escritorio

2.8.1. Personalización de vistas con PyQt

PyQt5 es una distribución de Qt para lenguaje Python que crea interfaces gráficas dinámicas que toma el estilo base del sistema operativo anfitrión para mostrar las vistas. Posee dos formas de renderizar interfaces, la primera es mediante el uso de QtDesigner que posee el método *drag and drop* de elementos para diseñar interfaces. Es suficiente arrastrar y posicionar un elemento, cambiarle el nombre y modificar propiedades a fin de dejarlo a gusto propio. La interfaz se guarda en un fichero de extensión .ui cuyo contenido es en formato XML. Para renderizar la vista es necesario importar el módulo *uic* dentro de un archivo Python y cargar el fichero .ui mediante la función *loadUi*.

La segunda forma es crear toda la interfaz a partir de la importación de las clases de PyQt5 y la codificación de sus propiedades dentro de un fichero .py. Este método tiene como ventaja el control total de la funcionalidad de la interfaz, pero la desventaja es la cantidad de líneas de código necesarias para lograrlo. Para este proyecto se usa QtDesigner para diseñar cinco vistas mencionadas en la sección Vista de este documento. En la sección actual se explica la programación de los controladores para cada vista y su conexión con el inicializador de la aplicación.

La estructura del directorio raíz del proyecto se basa en tres carpetas contenedoras de los archivos de modelo, controladores y vistas. Además, al mismo nivel de las carpetas se encuentra el archivo de inicio de la aplicación llamado main.py. Este fichero incluye la importación de la clase QApplication que crea una instancia dentro del sistema operativo en la que se alojará la aplicación, desde la carpeta control se importa la clase *MainWindow* del fichero ui.py que muestra la interfaz mediante el método *show*.

```python
from PyQt5.QtWidgets import QApplication
from controls.ui import MainWindow
from models.db.db import PrepareDatabase
import sys

def main():
    app = QApplication(sys.argv)
    app.setStyle("fusion")
    w = MainWindow()
    if PrepareDatabase():
        w.statusBar().showMessage(
            'Conectado | Presione Iniciar para comenzar con las detecciones'
            )
    w.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

Fig. 32. Función de inicio de la aplicación

Todo el comportamiento de la vista principal se encuentra dentro de *MainWindow*, separada en funciones para estructurar de mejor manera el código Python. La información que se muestra dinámicamente en los elementos *QLabel* son insertados mediante el método *setText* que permite insertar una cadena de caracteres. En esta vista se renderiza el video de las detecciones analizadas por el modelo entrenado desde el fichero *detect.py* y emitido mediante señales. *QMenu* y *QAction* generan un menú desplegable que contiene las opciones para visualizar los tres tipos de reportes, una pantalla duplicada del video y la opción de salir de la aplicación. Estas acciones desencadenan la generación de vistas en diálogo que se superponen a la vista principal, pero sin detener el proceso de detección.

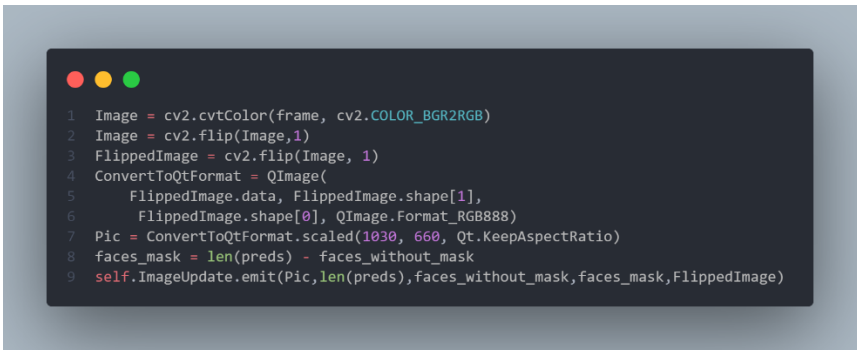
Los reportes heredan de la clase *QDialog* y cada tipo contiene un controlador y una vista vinculadas mediante *loadUi*. Asimismo, como se trata de una visualización de datos se establece conexión con el archivo db.py dentro de la ruta */model/db/*. En este archivo se encuentran todas las funciones necesarias para conectar con *SQLite* y extraer registros según se requiera. El enlace se realiza mediante la función *addDatabase* y se argumenta *QSQLITE* para establecer el tipo de base de datos. La función open establece la conexión y el método prepare de la clase *QSqlQuery* permite ejecutar transacciones, los valores retornan dentro de un vector o *None* si no hay registros disponibles.

```
def CreateConn():
    db = QSqlDatabase.addDatabase('QSQLITE')
    db.setDatabaseName('models/db/register.db')
    return db

def PrepareDatabase():
    db = CreateConn()
    if db.open():
        q = QSqlQuery()
        if (q.prepare(
            """create table if not exists log(
                id integer primary key autoincrement not null,
                nomask integer, total integer, date text
                )"""
            )):
            if (q.exec()):
                db.close()
                return True
```

Fig. 33. Función de creación de base de datos SQLite.

2.8.2. Señales

Las señales son objetos descendientes de *pyqtSignal* que permiten activar un método sin interrumpir la ejecución del código primario. Asimismo, son capaces de enviar argumentos de cualquier tipo si el método conectado lo requiere. El controlador de la ventana principal genera un objeto de la clase Worker1 que hereda de *QThread* para el realizar el renderizado del video que se encuentra desde el fichero *detect.py*. OpenCV captura cada fotograma desde la apertura de la cámara y realiza una superposición de cuadros mediante un bucle infinito. Tomando en cuenta este argumento, la función run perteneciente a *QThread* encapsula el bucle y permite ejecutarlo sin detener el funcionamiento de los demás controladores.



```
Image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
Image = cv2.flip(Image,1)
FlippedImage = cv2.flip(Image, 1)
ConvertToQtFormat = QImage(
    FlippedImage.data, FlippedImage.shape[1],
     FlippedImage.shape[0], QImage.Format_RGB888)
Pic = ConvertToQtFormat.scaled(1030, 660, Qt.KeepAspectRatio)
faces_mask = len(preds) - faces_without_mask
self.ImageUpdate.emit(Pic,len(preds),faces_without_mask,faces_mask,FlippedImage)
```

Fig. 34. Función que emite la señal desde Worker1

La función de detección que se encuentra dentro de un *while* contiene una señal que mediante la función *emit* envía información desde *Worker1* hasta *MainWindow* en cada iteración.

```
def ImageUpdateSlot(self, Image, t,s,c,i):
    self.cam_label.setPixmap(QPixmap.fromImage(Image))
    self.img = cv2.cvtColor(i,cv2.COLOR_BGR2RGB)
    self.detect_person.setText(str(t))
    self.no_mask_person.setText(str(s))
    self.mask_person.setText(str(c))
    self.play = True
```

Fig. 35. Función que recibe la señal en MainWindow

La función *connect* de la señal enlaza al método *ImageUpdateSlot* de la clase *MainWindow*, que recibe la imagen y la inserta en el *label*. En cada iteración del *while* activará el método y la imagen se actualizará automáticamente.

```
if faces_without_mask > 0:
    if len(preds) > dt:
        dt = len(preds)
    if faces_without_mask > ni:
        dt = len(preds)
        ni = faces_without_mask
        img_temp = FlippedImage
        send = True
        name = str(datetime.now())
    flag = 0
else:
    flag += 1

if flag == 30 and send:
    self.list_persons.emit(dt,ni,img_temp,name)
    dt = 0
    ni = 0
    flag = 0
    send = False
```

Fig. 36. Función que emite señal para guardar registro.

Desde Worker1 se emite otra señal que activa la función de guardado del registro, pero es necesario cumplir varias condiciones como se muestra en Fig. 36. El fragmento de código anterior permite guardar datos durante el bucle, actualiza los

valores anteriores si los actuales son mayores. Esto asegura que la señal emitida contiene la mayor cantidad de infractores detectados en un lapso de tiempo.

2.8.3. Despliegue

El despliegue de este proyecto se realizó en un equipo de escritorio con las características recomendadas en la Tabla 4, siguiendo estos pasos:

- ✓ Se ejecutó en una consola el comando

  git clone https://github.com/andatogua/FaceMaskDetect.git

  para clonar el repositorio

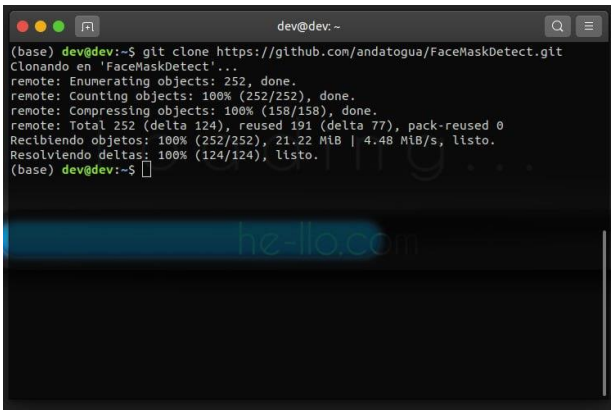


Fig. 37. Clonación del repositorio de Github.

- ✓ Se ingresó al directorio raíz del repositorio
- ✓ Se ejecutó en consola el comando *pwd*, luego se copió la ruta que se mostró

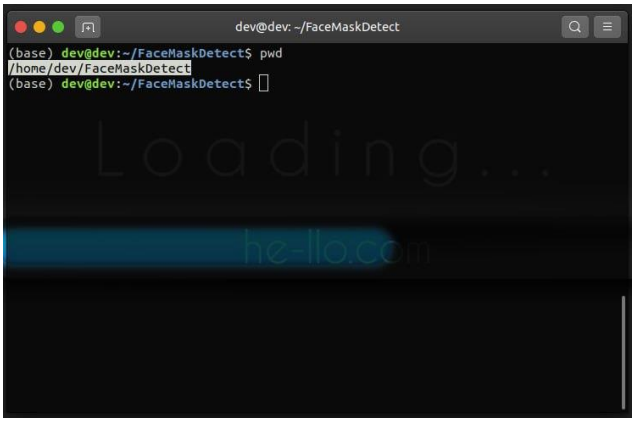


Fig. 38. Directorio raíz del repositorio.

- ✓ Dentro del directorio personal de Ubuntu, se editó el archivo *.bashrc* añadiendo las siguientes líneas al final del archivo

  conda activate nombredelentorno

export PATH="/ruta/copiada/desde/consola:$PATH"

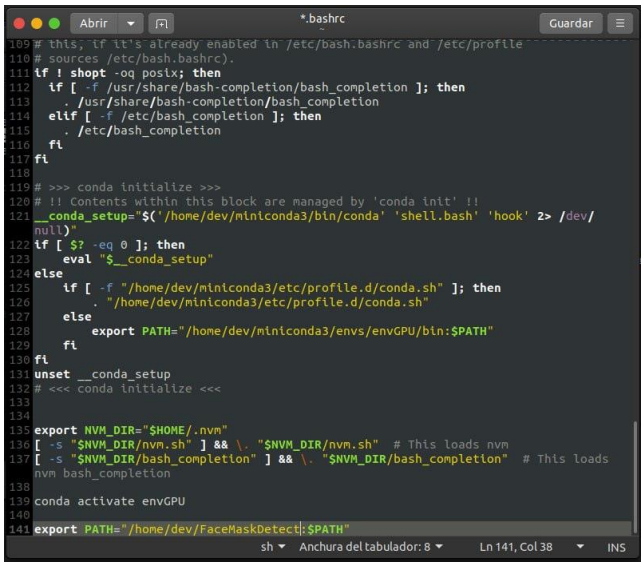


Fig. 39. Modificación del archivo .bashrc.

- ✓ Se editó el archivo ***maskdetect*** del repositorio clonado y se insertó la ruta copiada en la segunda línea luego de cd

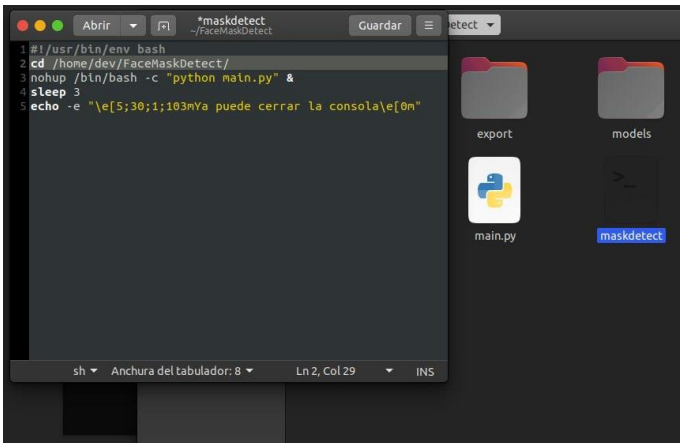


Fig. 40. Modificación del archivo maskdetect.

- ✓ Se reinició la consola y el entorno virtual se activó automáticamente

Fig. 41. Activación del entorno.

✓ Para iniciar el sistema ejecuté ***maskdetect*** y luego pude cerrar la consola de Ubuntu.

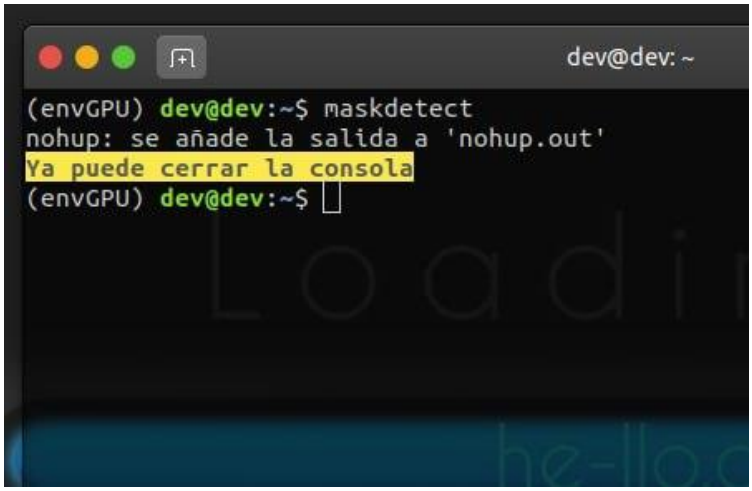


Fig. 42. Ejecución de la aplicación.

Fig. 43. Aplicación Iniciada.

2.9. Resultados.

2.9.1. Pruebas.

En este proyecto se realizaron con el propósito de conocer el porcentaje de confianza óptimo para las detecciones de rostros de ResNet50, luego se hicieron pruebas para determinar la distancia máxima permitida para realizar detección con

un mínimo porcentaje de error, finalmente se ejecutaron pruebas de detección de rostros sin mascarillas para establecer el porcentaje de precisión del modelo ejecutado desde la consola y posteriormente desde la aplicación.

Para todas las pruebas se utilizaron los siguientes elementos según Sección 2.2.22:

Cámara: Webcam con autoenfoque, 720p de densidad de píxeles.

Iluminación: Natural

Prueba de índice de confianza de ResNet50

N° Prueba	Rostros reales	Nivel de confianza					
		50%		**40%**		**30%**	
		Detectados	**Precisión**	**Detectados**	**Precisión**	**Detectados**	**Precisión**
1	5	3	60%	4	80%	5	100%
2	2	2	100%	2	100%	2	100%
3	6	4	66%	4	66%	6	100%
4	1	0	0%	1	100%	1	100%
5	11	8	72%	9	81%	10	91%
6	15	10	66%	11	73%	13	86%
7	8	5	62%	7	87%	8	100%
8	4	4	100%	4	100%	4	100%
Promedio		**65,75%**		**85,87%**		**97,12%**	

Tabla 6. Prueba de índice de confianza

La tabla anterior muestra el detalle de las pruebas realizadas a ocho fotogramas bajo índices de confianza de 50%, 40% y 30%, obteniendo una precisión de 65,75%, 85,87% y 97,12% respectivamente. Se declara que el índice idóneo para detección de rostros en ResNet50 es 30% (Anexo 7).

Prueba de distancia máxima de detección ResNet50

N° Prueba	Rostros reales	Distancia desde el observador					
		2 m		**3 m**		**4 m**	
		Detectados	**Precisión**	**Detectados**	**Precisión**	**Detectados**	**Precisión**
1	5	5	100%	5	100%	2	40%
2	2	2	100%	1	50%	1	50%
3	6	6	100%	4	66%	4	66%
4	1	1	100%	1	100%	0	0%
5	11	10	91%	9	81%	5	45%
6	15	12	90%	11	73%	4	26%
7	8	8	100%	7	87%	4	50%
8	4	4	100%	4	100%	1	25%
Promedio		**97,62 %**		**82,12%**		**37,75%**	

Tabla 7. Prueba de distancia máxima de detección.

Concluida la prueba se determinó que la distancia máxima requerida para realizar detecciones con un alto índice de precisión es tres metros, según los datos detallados en la tabla anterior. El promedio de precisión obtenido es de 82,12% que está dentro del rango permitido para evitar detecciones erróneas (Anexo 8).

A partir de este punto las pruebas se realizaron bajo las siguientes condiciones:

Cámara: Webcam con autoenfoque, 720p de densidad de píxeles.

Iluminación: Natural

Nivel de confianza: 30%

Distancia máxima de detección: 3 metros.

Detección de rostros con ResNet50

N° Prueba	N° rostros reales	N° rostros detectados	Índice de precisión
1	1	1	100%
2	3	3	100%
3	5	5	100%
4	2	2	100%
5	5	4	80%
6	4	4	100%
7	6	5	83%
8	3	2	66%
9	4	4	100%
10	6	6	100%
		Promedio	**92.9%**

Tabla 8. Prueba de detección de rostros

La tabla anterior detalla el índice de confianza obtenido en 10 iteraciones, cada una con un número de personas distinto. Las detecciones de rostros se realizaron desde la consola de Python con el fin de conocer la efectividad de ResNet50 (Anexo 9) obteniendo un 92.9% de precisión .

Detección de rostros sin mascarillas con modelo entrenado

N° Prueba	N° rostros reales	N° rostros detectados	Índice de precisión
1	1	1	100%
2	3	3	100%
3	5	4	80%
4	2	2	100%
5	5	4	80%
6	4	4	100%
7	6	5	83%
8	3	2	66%
9	4	4	75%
10	6	6	100%
		Promedio	**88.4%**

Tabla 9. Prueba de detección de rostros sin mascarillas.

El índice de confianza obtenido de la prueba de detección de rostros sin mascarillas es 88.4%. Esta prueba fue ejecutada desde la consola de Python para comprobar la efectividad del modelo entrenado (Anexo 10).

Detección de rostros con modelo entrenado desde la aplicación

N° Prueba	N° rostros reales	N° rostros detectados	Índice de precisión
1	1	1	100%
2	3	3	100%
3	5	4	80%
4	2	2	100%
5	5	4	80%
6	4	4	100%
7	6	6	100%
8	3	2	66%
9	4	3	75%
10	6	4	66%
		Promedio	**86.7%**

Tabla 10. Prueba de detección de rostros sin mascarillas desde la aplicación.

En Tabla 10 se muestra el índice obtenido en cada una de las 10 pruebas realizadas desde la aplicación. El promedio final es 86.7% de confianza en la detección de rostros sin mascarillas (Anexo 11).

2.9.2. Resultados Finales

Los resultados de la ejecución de este proyecto son:

1. El entorno virtual de trabajo creado por Anaconda, en que se instalaron correctamente las dependencias necesarias para el análisis matemático y aceleración por GPU. La integración entre CUDA y Tensorflow permitió la detección de la tarjeta gráfica instalada en el equipo como lo muestra la Fig. 22. Asimismo, la instalación de los módulos Keras, OpenCV, Numpy Matplotlib y PyQt5 se realizó satisfactoriamente.
2. El script Python de nombre *train_mask_detector.py* con la codificación de la estructura, compilación y entrenamiento de la red neuronal. Este archivo se encuentra disponible en un repositorio público de Github [56], consta de dos parámetros: el número de épocas de entrenamiento y el tamaño de lote que son de tipo entero. La ejecución de este script se realiza de la siguiente forma:

python train_mask_detector.py –e 20 –b 32

3. El modelo para detección de rostros sin mascarillas compilado y entrenado en Google Colab. En Fig. 30 se muestran los valores de confianza obtenidos a partir del entrenamiento del modelo durante 100 épocas, con una índice de certeza del 100% en la validación con imágenes de prueba. Tensorboard recolectó los registros del entrenamiento y mostró la curva de aprendizaje del modelo como se evidencia en la siguiente figura.

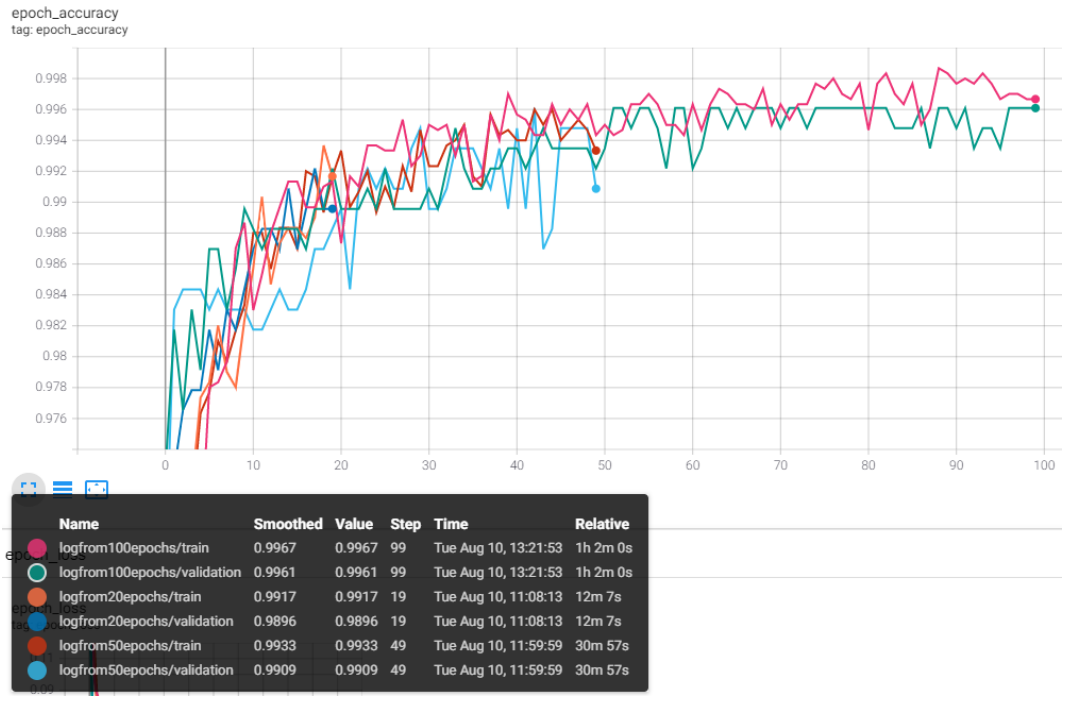


Fig. 44. Curva de aprendizaje del modelo en 100 épocas.

4. La aplicación desarrollada en PyQt5 que contiene el modelo de detección para análisis de rostros sin mascarilla se instaló correctamente en Ubuntu como se muestra en Fig. 43. Posee 3 tipos de reportes estadísticos como se menciona en la sección Vista que muestran histogramas renderizados con Matplotlib a partir de los datos almacenados en una base de datos SQLite.
5. El sistema de detección de rostros sin mascarillas obtuvo un valor de precisión superior a 80%, de acuerdo a las pruebas realizadas al modelo entrenado bajo condiciones óptimas: confianza de 30% y distancia máxima de tres metros.

Conclusiones

- ✓ Debido a que este proyecto de visón por computador tiene su base en los sistemas de monitoreo periódico se concluye que la efectividad del mismo depende los recursos tecnológicos disponibles y las condiciones físicas del entorno. Por lo tanto, es imprescindible aclarar que la implementación del sistema de detección de rostros sin mascarilla deje ejecutarse bajo las recomendaciones establecidas en este documento.

- ✓ La implementación de un entorno virtual de trabajo con integración de CUDA permitió la aceleración del análisis de cada fotograma evitando así el retardo en la renderización del video sin saturar los hilos de procesamiento.

- ✓ El script desarrollado en lenguaje Python junto al módulo Keras de Tensorflow estableció variables incidentes en el análisis del dataset, entrenamiento y compilación del modelo.

- ✓ Los valores de precisión del modelo entrenado varían de acuerdo a la cantidad de épocas destinadas al entrenamiento, adicionalmente se comprobó que la curva de aprendizaje de la red neuronal crece constantemente y se acerca a 100% entre 50 y 100 épocas.

- ✓ La visualización de reportes estadísticos permitió comparar el índice de infracciones registradas en una jornada de trabajo, ayudando así al administrador del sistema a tomar decisiones frente al comportamiento de los infractores en el laboratorio de FACSISTEL.

- ✓ El sistema de detección de rostros sin mascarilla obtuvo una precisión superior al 80% durante las pruebas realizadas, además se comprobó que este valor está sujeto a condiciones como: densidad de píxeles de la cámara, distancia máxima del observador y la iluminación.

Recomendaciones

- ✓ El despliegue de este proyecto debe ejecutarse bajo los requisitos físicos y digitales descritos en este documento. De ser necesario el aumento de la precisión se recomienda potenciar el sistema con: Unidades de procesamiento de gráficos con mayor capacidad de cómputo, una cámara con mayor densidad de píxeles y mayor iluminación en el sitio de ubicación de la cámara.

- ✓ El desarrollo y despliegue de proyecto basados en Inteligencia Artificial requieren grandes cantidades de procesamiento, por lo tanto, es necesario contar con dispositivos compatibles a fin de no saturar los procesos principales del sistema operativo.

- ✓ MobileNetV2 es una arquitectura desarrollada para analizar conjuntos de datos pequeños en equipos con una carga de procesamiento moderada, por esta razón si se desea incrementar el tamaño del dataset será necesario aumentar el número de neuronas de la red y añadir más etiquetas de clases dentro del código.

- ✓ Emplear otras técnicas de detección de rostros con mayor índice de confianza que permita el ingreso de datos más precisos hacia el modelo entrenado.

- ✓ Ejecutar la aplicación durante toda la jornada académica a fin de almacenar datos relevantes que se mostrarán en los reportes estadísticos, y, que ayudarán a la toma de decisiones frente al comportamiento de los infractores que ingresen a los laboratorios.

- ✓ Implementar este proyecto en otros sitios de recurrencia dentro de la Universidad con el fin de controlar en mayor medida el cumplimiento del uso de la mascarilla.

Bibliografía

[1] «COVID-19: Preguntas frecuentes». https://www.unicef.org/es/coronavirus/lo-que-los-padres-deben-saber (accedido ago. 12, 2021).

[2] «Resoluciones COE Nacional 06 de abril 2020 – Servicio Nacional de Gestión de Riesgos y Emergencias». https://www.gestionderiesgos.gob.ec/resoluciones-coe-nacional-06-de-abril-2020/ (accedido jun. 16, 2021).

[3] «Cuándo y cómo usar mascarilla». https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/advice-for-public/when-and-how-to-use-masks (accedido jun. 16, 2021).

[4] J. E. M. Díaz, «Inteligencia Artificial y Big Data como soluciones frente al COVID-19», *Revista de Bioética y Derecho*, vol. 0, n.º 50, Art. n.º 50, jul. 2020, doi: 10.1344/rbd2020.50.31643.

[5] H. A. Carrera, S. S. Maita, y P. H. Lascano, «Modelo para detectar el uso correcto de mascarillas en tiempo real utilizando redes neuronales convolucionales», *Revista de Investigación en Tecnologías de la Información*, vol. 9, n.º 17, Art. n.º 17, ene. 2021.

[6] A. Ansor, R. Ritzkal, y Y. Afrianto, «Mask Detection Using Framework Tensorflow and Pre-Trained CNN Model Based on Raspberry Pi», *Jurnal Mantik*, vol. 4, n.º 3, Art. n.º 3, nov. 2020, doi: 10.35335/mantik.Vol4.2020.946.pp1539-1545.

[7] «La COVID-19 y las mascarillas: consejos para las familias». https://www.unicef.org/es/coronavirus/covid19-y-mascarillas-consejos-para-familias (accedido jun. 16, 2021).

[8] «Uso correcto de la mascarilla», *Universidad Pedro de Valdivia*. https://www.upv.cl/2020/06/12/uso-correcto-de-la-mascarilla/ (accedido jun. 16, 2021).

[9] «GRADO». https://www.upse.edu.ec/index.php (accedido jun. 26, 2021).

[10] «revistas». https://incyt.upse.edu.ec/index.php/component/sppagebuilder/5-revistas (accedido jun. 26, 2021).

[11] «Home | INARI Research Lab». https://www.inarilab.com/#research (accedido jun. 16, 2021).

[12] «Distancia2», *Código para el Desarrollo*. https://code.iadb.org/es/herramientas/distancia2 (accedido jun. 16, 2021).

[13] «Con un moderno software, cámaras del ECU 911 miden el distanciamiento físico y generan alertas – Servicio Integrado de Seguridad ECU 911». https://www.ecu911.gob.ec/con-un-moderno-software-camaras-del-ecu-911-miden-el-distanciamiento-fisico-y-generan-alertas/ (accedido jun. 16, 2021).

[14] M. G. Pose, «Introducción a las Redes de Neuronas Artificiales», p. 20.

[15] H. Kaiming, Z. Xiangyu, R. Shaoqing, y S. Jian, «Deep Residual Learning for Image Recognition», dic. 2015.

[16] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, y L.-C. Chen, «MobileNetV2: Inverted Residuals and Linear Bottlenecks», mar. 2019.

[17] «Licencias - Proyecto GNU - Free Software Foundation». https://www.gnu.org/licenses/licenses.es.html (accedido jun. 16, 2021).

[18] «Tutorial de Python — documentación de Python - 3.8.10». https://docs.python.org/es/3.8/tutorial/index.html (accedido jun. 16, 2021).
[19] «About», *OpenCV*. https://opencv.org/about/ (accedido jun. 16, 2021).
[20] «Introduction — PyQt v5.15 Reference Guide». https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html (accedido jun. 16, 2021).
[21] «SQLite Home Page». https://www.sqlite.org/index.html (accedido jun. 16, 2021).
[22] «TensorFlow». https://www.tensorflow.org/?hl=es-419 (accedido jun. 16, 2021).
[23] K. Team, «Keras documentation: About Keras». https://keras.io/about/ (accedido jun. 16, 2021).
[24] «Google Colaboratory». https://colab.research.google.com/notebooks/intro.ipynb (accedido jun. 16, 2021).
[25] «Anaconda | The World's Most Popular Data Science Platform», *Anaconda*. https://www.anaconda.com/ (accedido jun. 16, 2021).
[26] «CUDA Zone», *NVIDIA Developer*, jul. 18, 2017. https://developer.nvidia.com/cuda-zone (accedido jul. 06, 2021).
[27] «lineamientos_covid-19__final_09-06-2020_v3_1-2.pdf». Accedido: jun. 16, 2021. [En línea]. Disponible en: https://www.salud.gob.ec/wp-content/uploads/2020/03/lineamientos_covid-19__final_09-06-2020_v3_1-2.pdf
[28] «Plan Nacional de Desarrollo 2017 – 2021 Toda una Vida – Secretaría Técnica Planifica Ecuador». https://www.planificacion.gob.ec/plan-nacional-de-desarrollo-2017-2021-toda-una-vida/ (accedido ene. 14, 2021).
[29] R. Hernández Sampieri, C. Fernández Collado, y P. Baptista Lucio, *Metodología de la investigación*. México: McGraw Hill Interamericana, 2014.
[30] F. ALONSO, L. MARTINEZ, y FCO. J. SEGOVIA, *INTRODUCCION A LA INGENIERIA DEL SOFTWARE*. Delta, 2005.
[31] S. I. Mariño y C. R. Primorac, «Propuesta metodológica para desarrollo de modelos de redes neuronales artificiales supervisadas», *IJERI: International Journal of Educational Research and Innovation*, n.º 6, Art. n.º 6, may 2016.
[32] «Universidades de Ecuador», *Universidades de Ecuador*. http://universidades.com.ec/ (accedido jun. 16, 2021).
[33] «UPSE RECIBE LA MÁXIMA ACREDITACIÓN». https://www.upse.edu.ec/index.php?option=com_content&view=article&id=350:upse-recibe-la-maxima-acreditacion&catid=10&Itemid=178 (accedido jun. 16, 2021).
[34] «Matriz». https://www.upse.edu.ec/index.php?option=com_content&view=article&id=14&Itemid=273 (accedido jul. 14, 2021).
[35] «MISIÓN - VISIÓN». https://www.upse.edu.ec/index.php?option=com_content&view=article&id=12&Itemid=167 (accedido jun. 28, 2021).
[36] Alina, «Decanato». http://facsistel.upse.edu.ec/index.php (accedido jun. 28, 2021).

[37] Alina, «UNIDADES Y CENTROS». http://facsistel.upse.edu.ec/index.php (accedido jun. 28, 2021).
[38] «55 REGLAMENTO PARA EL USO DE LOS LABORATORIOS Y TALLERES DE LA UPSE.pdf». Accedido: jul. 07, 2021. [En línea]. Disponible en: http://www.upse.edu.ec/secretariageneral/images/archivospdfsecretaria/4.REGLAMENTOS/2.%20NORMATIVAS%20ADMINISTRATIVAS/55%20REGLAMENTO%20PARA%20EL%20USO%20DE%20LOS%20LABORATORIOS%20Y%20TALLERES%20DE%20LA%20UPSE.pdf
[39] «El concepto de IDE». https://www.redhat.com/es/topics/middleware/what-is-ide (accedido jul. 05, 2021).
[40] «Visual Studio Code - Code Editing. Redefined». https://code.visualstudio.com/ (accedido jul. 06, 2021).
[41] *microsoft/vscode*. Microsoft, 2021. Accedido: jul. 06, 2021. [En línea]. Disponible en: https://github.com/microsoft/vscode
[42] «Glosario — documentación de Python - 3.9.6». https://docs.python.org/es/3/glossary.html#term-virtual-environment (accedido jul. 06, 2021).
[43] «Conda environments — conda 4.10.2.post1+248741a84 documentation». https://conda.io/projects/conda/en/latest/user-guide/concepts/environments.html (accedido jul. 06, 2021).
[44] C. S., «Cómo mantener el patrón modelo vista controlador en una aplicación orientada a la WEB», *Revista Inventum*, vol. 4, p. 72, jul. 2009, doi: 10.26620/uniminuto.inventum.4.7.2009.72-78.
[45] J. A. P. Sánchez, «SISTEMA DE RECONOCIMIENTO DE OBJETOS REMOVIDOS DE UNA ESCENA, UTILIZANDO VISIÓN POR COMPUTADOR», p. 59, 2011.
[46] M. Díaz-Alejo, «Análisis comparativo de arquitecturas de redes neuronales para la clasificación de imágenes», p. 85.
[47] E. Universitat Politècnica de València, «Universitat Politècnica de València», *ing.agua*, vol. 18, n.º 1, p. ix, sep. 2014, doi: 10.4995/ia.2014.3293.
[48] «MobileNetV2: The Next Generation of On-Device Computer Vision Networks», *Google AI Blog*. http://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html (accedido jul. 14, 2021).
[49] «Primeros pasos: Entrenamiento y predicción con Keras | AI Platform», *Google Cloud*. https://cloud.google.com/ai-platform/docs/getting-started-keras?hl=es-419 (accedido jul. 07, 2021).
[50] «Clasificacion Basica: Predecir una imagen de moda | TensorFlow Core», *TensorFlow*. https://www.tensorflow.org/tutorials/keras/classification?hl=es-419 (accedido jul. 07, 2021).
[51] «08_0313_CS.pdf». Accedido: ago. 17, 2021. [En línea]. Disponible en: http://biblioteca.usac.edu.gt/tesis/08/08_0313_CS.pdf
[52] P. N. C. Prócel, «DISEÑO E IMPLEMENTACIÓN DE UN SISTEMA DE VISIÓN ARTIFICIAL PARA CLASIFICACIÓN DE AL MENOS TRES TIPOS DE FRUTAS.», p. 108.
[53] «Ubuntu PC operating system», *Ubuntu*. https://ubuntu.com/desktop (accedido jul. 28, 2021).

[54] «ImageNet». https://image-net.org/ (accedido ago. 02, 2021).
[55] «sklearn.model_selection.train_test_split — scikit-learn 0.24.2 documentation». https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html#sklearn.model_selection.train_test_split (accedido ago. 03, 2021).
[56] andatogua, *trainmaskdetect*. 2021. Accedido: ago. 12, 2021. [En línea]. Disponible en: https://github.com/andatogua/trainmaskdetect

ANEXOS

Anexo 1. Ficha técnica de observación en Iglesia Matriz de Santa Elena

<table>
<tr><th colspan="4">FICHA DE REGISTRO</th></tr>
<tr><td>Lugar</td><td colspan="3">Iglesia Matriz de Santa Elena</td></tr>
<tr><td>Fecha</td><td colspan="3">03/01/2021</td></tr>
<tr><td>N° personas</td><td colspan="3">458</td></tr>
<tr><td>Proceso</td><td colspan="3">Detección de personas sin mascarillas</td></tr>
<tr><td>Hora Inicio</td><td>6:30</td><td>Hora Fin</td><td>11:30</td></tr>
<tr><td colspan="4">OBSERVACIÓN DIRECTA</td></tr>
<tr><td colspan="4">✓ Entrada y salida de personas se encuentran en diferentes sectores y debidamente rotuladas.
✓ Existen 6 personas destinadas a la detección de uso de mascarillas, 2 a la entrada, 2 a la salida y 2 que hacen ronda dentro de la iglesia.
✓ El ingreso y salida de personas se realiza mediante una fila.
✓ Cuando una persona sin mascarillas es detectada el personal se acerca a decirle una advertencia.
✓ Nadie puede ingresar sin mascarilla y si alguien lo intenta es retirado del sitio por incumplir la normativa.
✓ Existe un registro de detecciones realizadas durante la jornada.</td></tr>
<tr><td>Conclusiones</td><td colspan="3">✓ A pesar de contar con 6 personas para monitorear a los participantes de la misa no es suficiente para detectar el 100% de las infracciones.
✓ Se detectó 57 infracciones.</td></tr>
<tr><td>Responsable</td><td colspan="3">Andrés Tomalá Guaranda</td></tr>
</table>

Anexo 1. Observación Iglesia Matriz de Santa Elena

Anexo 2. Ficha técnica de observación en Registro Civil de Santa Elena

<table>
<tr><th colspan="4">FICHA DE REGISTRO</th></tr>
<tr><td>Lugar</td><td colspan="3">Registro Civil de Santa Elena</td></tr>
<tr><td>Fecha</td><td colspan="3">04/01/2021</td></tr>
<tr><td>N° personas</td><td colspan="3">124</td></tr>
<tr><td>Proceso</td><td colspan="3">Detección de personas sin mascarillas</td></tr>
<tr><td>Hora Inicio</td><td>8:30</td><td>Hora Fin</td><td>12:30</td></tr>
<tr><td colspan="4">OBSERVACIÓN DIRECTA</td></tr>
<tr><td colspan="4">✓ Entrada y salida de personas se encuentran en el mismo sector.
✓ Existen 2 personas destinadas a la detección de uso de mascarillas, 1 a la entrada y salida y otro en el interior.
✓ Solo el ingreso realiza mediante una fila.
✓ Cuando una persona sin mascarillas es detectada el personal se acerca a decirle una advertencia, si no obedece es retirado por el guardia de seguridad.
✓ Nadie puede ingresar sin mascarilla.
✓ No existe un registro de detecciones realizadas durante la jornada.</td></tr>
<tr><td>Conclusiones</td><td colspan="3">✓ El personal es muy limitado para realizar un control efectivo del uso de la mascarilla. En varias ocasiones se infringió la norma inclusive por parte del personal que labora en el sitio.
✓ Se detectó 32 infracciones.</td></tr>
<tr><td>Responsable</td><td colspan="3">Andrés Tomalá Guaranda</td></tr>
</table>

Anexo 2. Observación Registro Civil de Santa Elena

Anexo 3. Ficha técnica de observación en Laboratorios Informática UPSE

<table>
<tr><th colspan="4">FICHA DE REGISTRO</th></tr>
<tr><td>Lugar</td><td colspan="3">Laboratorios de Informática FACSISTEL-UPSE</td></tr>
<tr><td>Fecha</td><td colspan="3">01/07/2021</td></tr>
<tr><td>N° personas</td><td colspan="3">1</td></tr>
<tr><td>Proceso</td><td colspan="3">Análisis del atrio y pasillos del laboratorio</td></tr>
<tr><td>Hora Inicio</td><td>15:30</td><td>Hora Fin</td><td>16:30</td></tr>
<tr><td colspan="4">OBSERVACIÓN DIRECTA</td></tr>
<tr><td colspan="4">✓ El atrio tiene un espacio aproximado de 8 m^2 y conecta la puerta principal del laboratorio con la Av. Universitaria.
✓ El laboratorio tiene 2 puertas y ambas en condición de entrada y salida.
✓ El edificio cuenta con 3 salas de computación, una oficina de administración y 2 baños.
✓ El edificio cuenta con un aforo aproximado de 120 personas, suponiendo la excepción del distanciamiento social.
✓ La administración del laboratorio está a cargo de una sola persona.
✓ Los pasillos tienen un espacio aproximado de 30 m^2.</td></tr>
<tr><td>Conclusiones</td><td colspan="3">✓ Tomando en cuenta las restricciones sanitarias establecidas el aforo permitido sería de 20 personas por sala de computación y una sola persona no es suficiente para controlar el uso de mascarillas en el atrio y pasillos del edificio.
✓ El espacio disponible en los pasillos no es suficiente para permitir aglomeraciones.</td></tr>
<tr><td>Responsable</td><td colspan="3">Andrés Tomalá Guaranda</td></tr>
</table>

Anexo 3. Observación Laboratorios Informática FACSISTEL-UPSE

Anexo 4. Árbol del problema

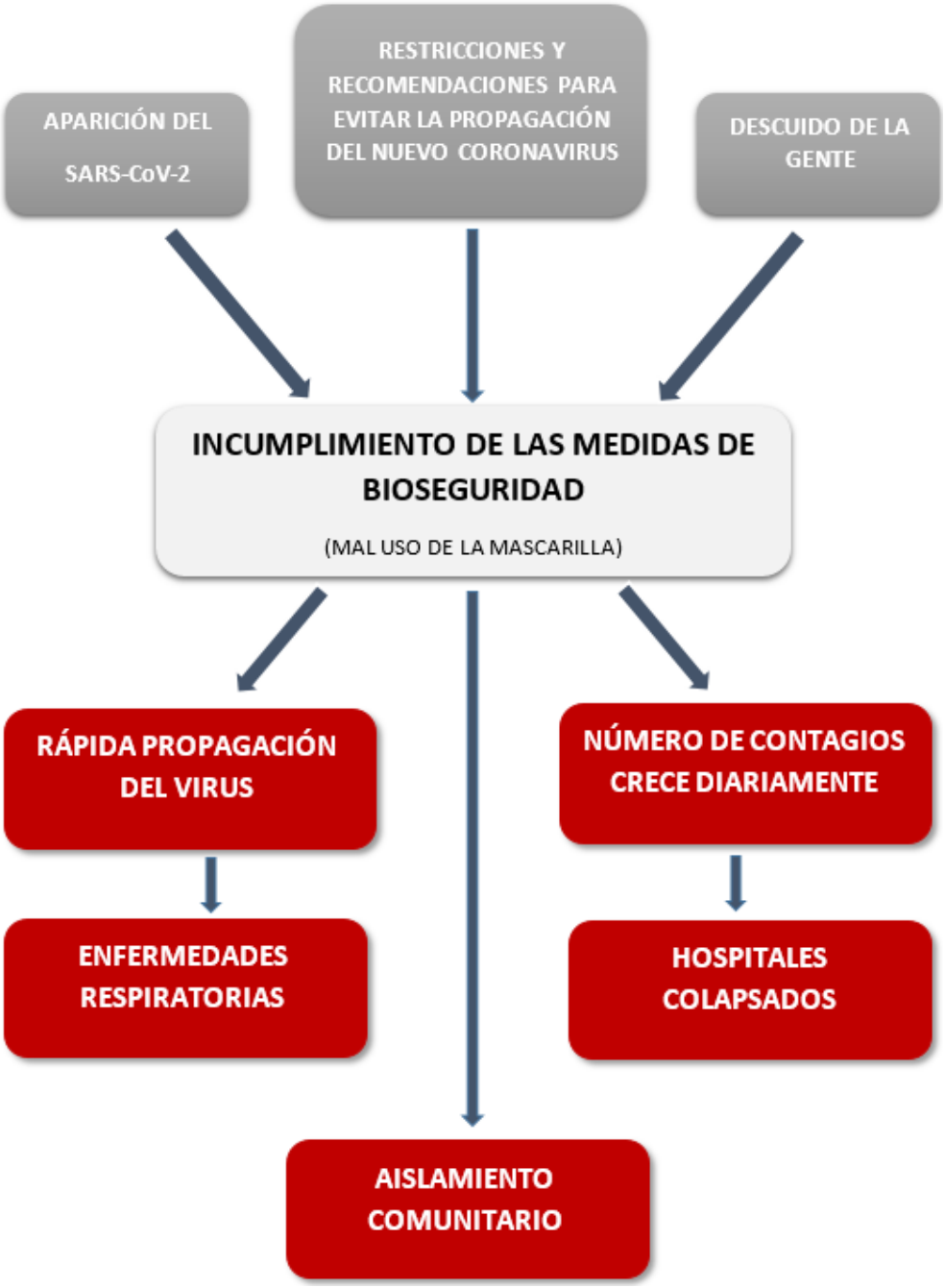


Fig. 45. Árbol del problema del incumplimiento de las medidas de bioseguridad.

Anexo 4. Árbol del problema

Anexo 5. Arquitectura de ResNet50

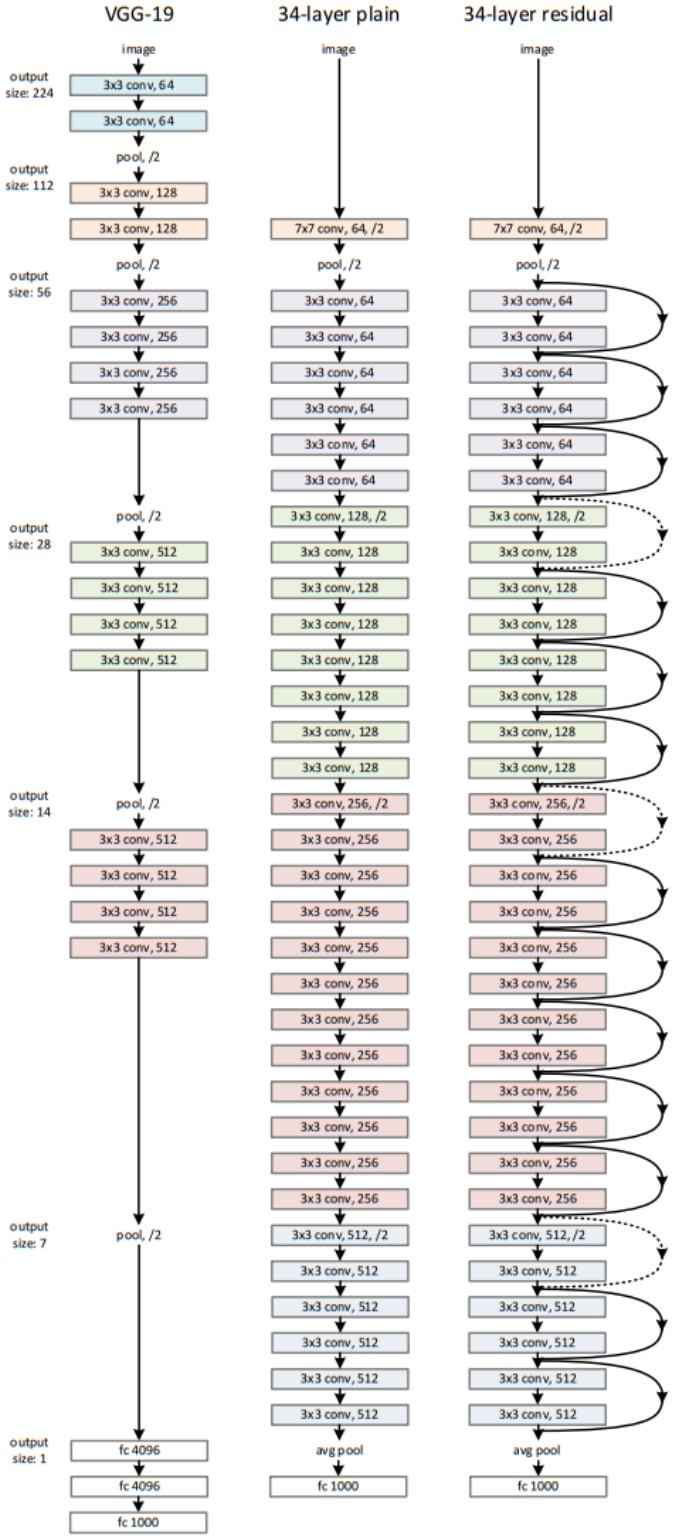
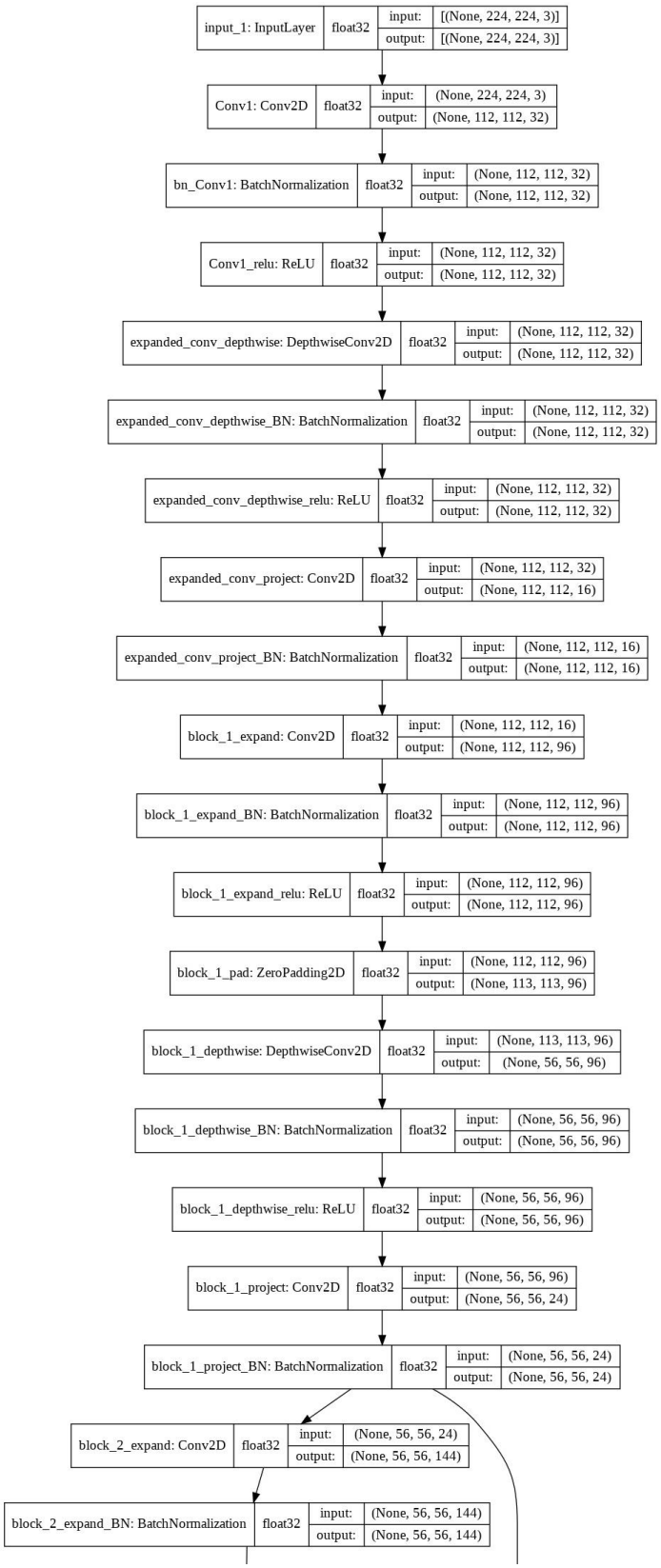


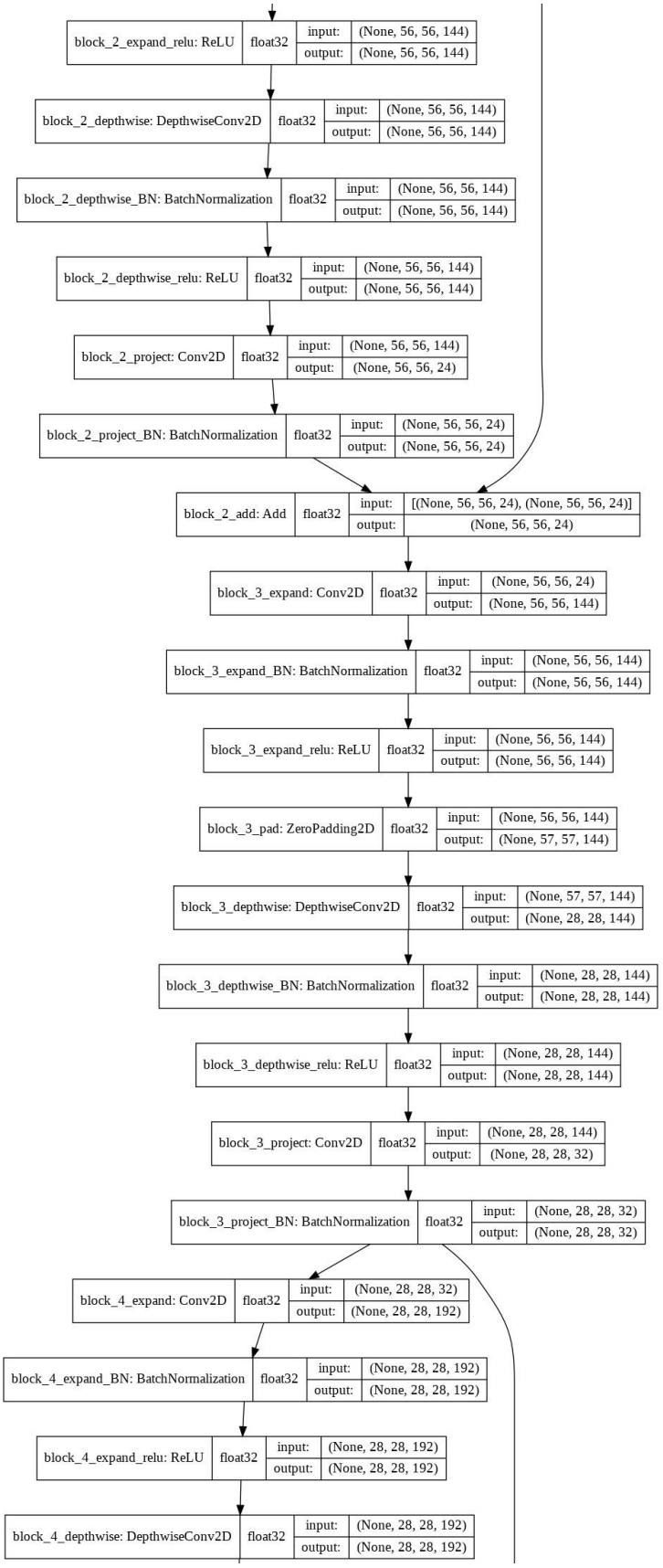
Fig. 46 Arquitectura de Resnet50 [15]

Anexo 5. Arquitectura de ResNet50

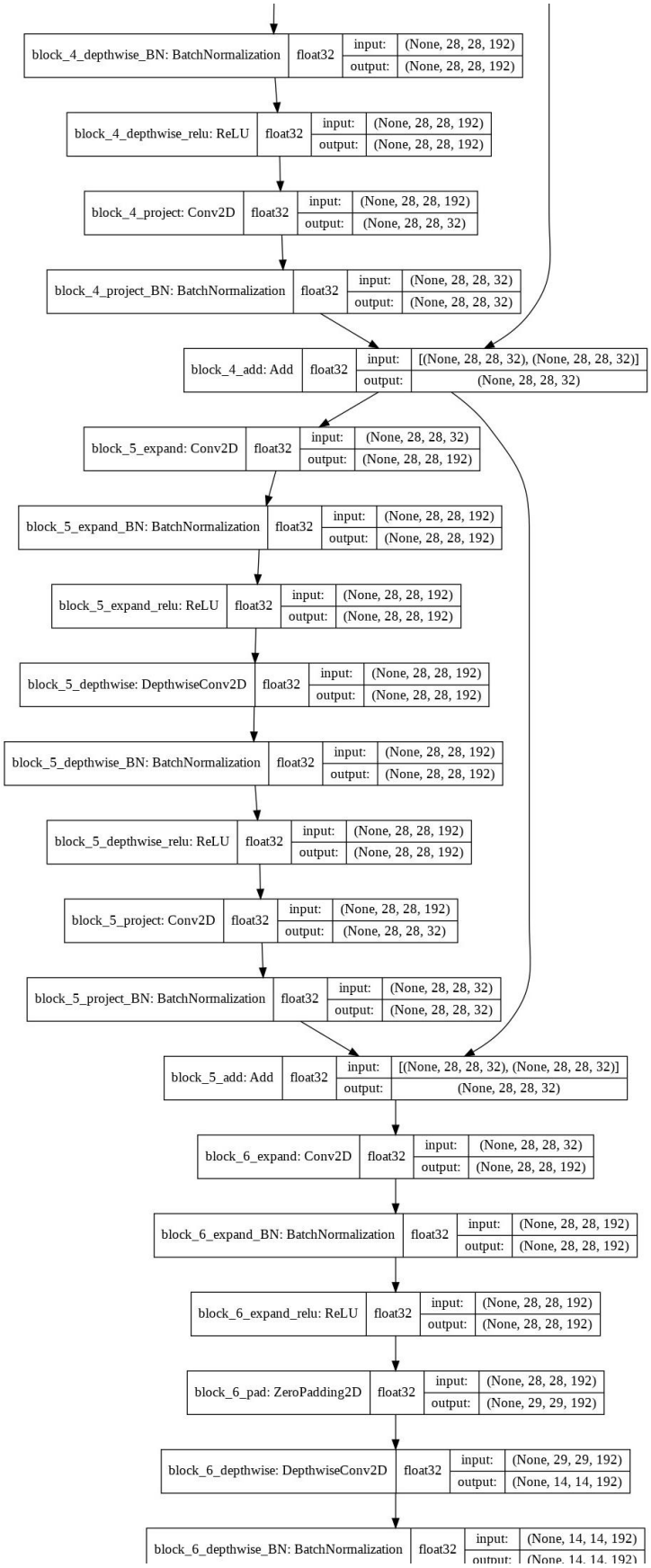
Anexo 6. Arquitectura del modelo de red neuronal



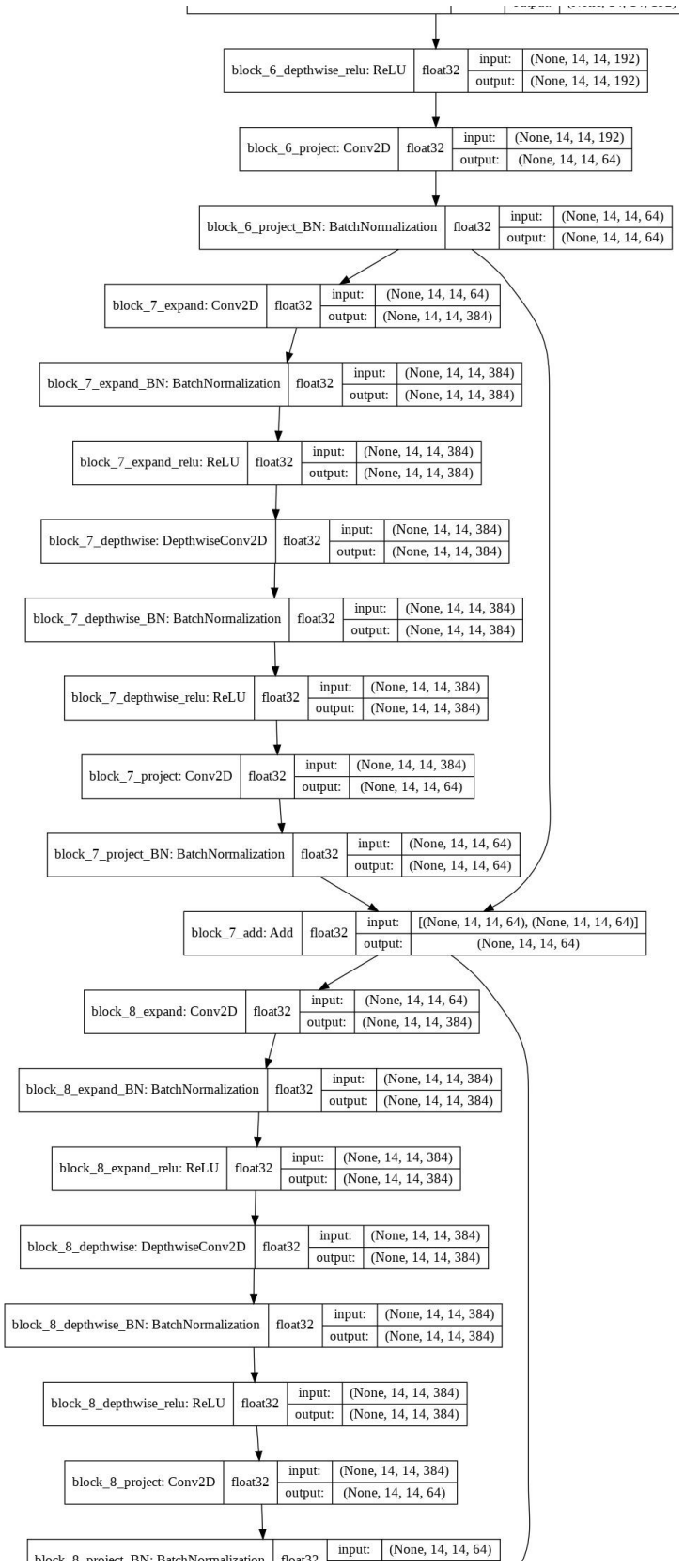
(a)


block_2_expand_relu: ReLU | float32 | input: (None, 56, 56, 144) | output: (None, 56, 56, 144)
block_2_depthwise: DepthwiseConv2D | float32 | input: (None, 56, 56, 144) | output: (None, 56, 56, 144)
block_2_depthwise_BN: BatchNormalization | float32 | input: (None, 56, 56, 144) | output: (None, 56, 56, 144)
block_2_depthwise_relu: ReLU | float32 | input: (None, 56, 56, 144) | output: (None, 56, 56, 144)
block_2_project: Conv2D | float32 | input: (None, 56, 56, 144) | output: (None, 56, 56, 24)
block_2_project_BN: BatchNormalization | float32 | input: (None, 56, 56, 24) | output: (None, 56, 56, 24)
block_2_add: Add | float32 | input: [(None, 56, 56, 24), (None, 56, 56, 24)] | output: (None, 56, 56, 24)
block_3_expand: Conv2D | float32 | input: (None, 56, 56, 24) | output: (None, 56, 56, 144)
block_3_expand_BN: BatchNormalization | float32 | input: (None, 56, 56, 144) | output: (None, 56, 56, 144)
block_3_expand_relu: ReLU | float32 | input: (None, 56, 56, 144) | output: (None, 56, 56, 144)
block_3_pad: ZeroPadding2D | float32 | input: (None, 56, 56, 144) | output: (None, 57, 57, 144)
block_3_depthwise: DepthwiseConv2D | float32 | input: (None, 57, 57, 144) | output: (None, 28, 28, 144)
block_3_depthwise_BN: BatchNormalization | float32 | input: (None, 28, 28, 144) | output: (None, 28, 28, 144)
block_3_depthwise_relu: ReLU | float32 | input: (None, 28, 28, 144) | output: (None, 28, 28, 144)
block_3_project: Conv2D | float32 | input: (None, 28, 28, 144) | output: (None, 28, 28, 32)
block_3_project_BN: BatchNormalization | float32 | input: (None, 28, 28, 32) | output: (None, 28, 28, 32)
block_4_expand: Conv2D | float32 | input: (None, 28, 28, 32) | output: (None, 28, 28, 192)
block_4_expand_BN: BatchNormalization | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_4_expand_relu: ReLU | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_4_depthwise: DepthwiseConv2D | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)

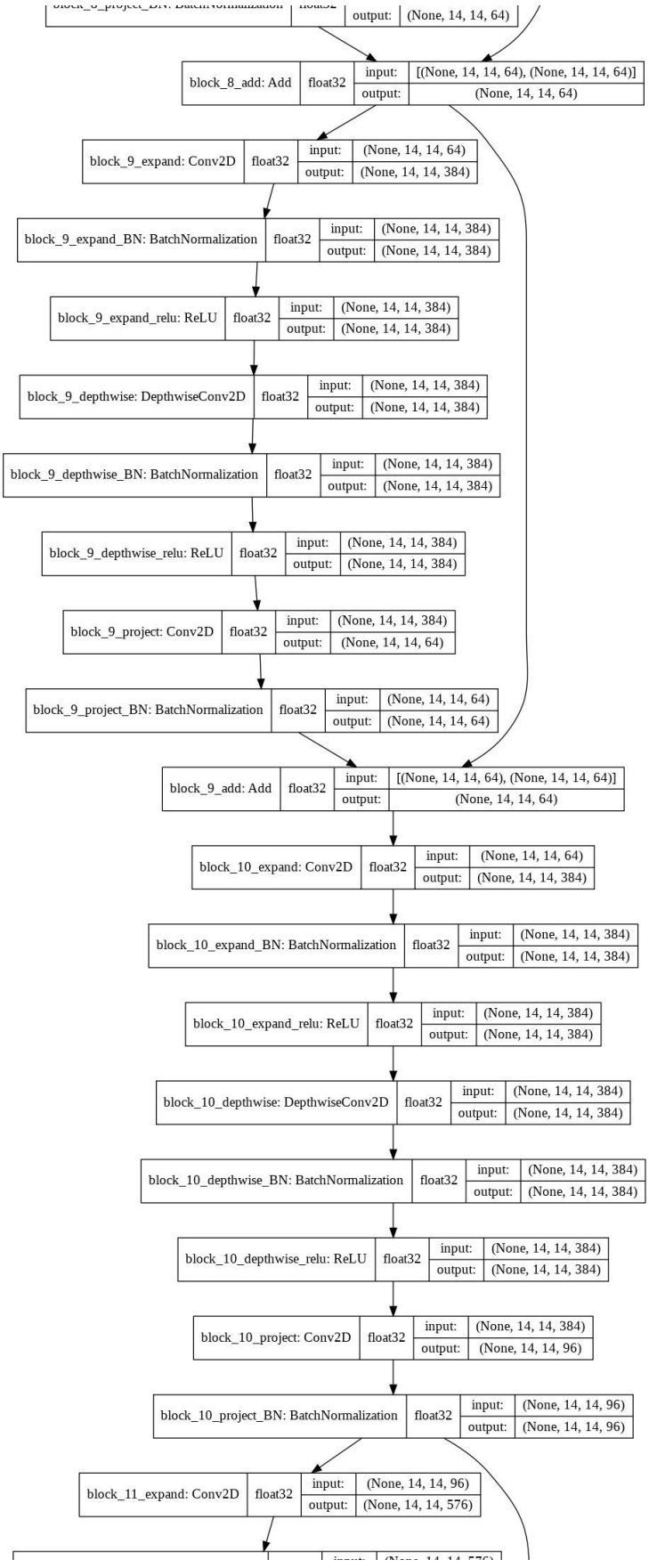
(b)


block_4_depthwise_BN: BatchNormalization | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_4_depthwise_relu: ReLU | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_4_project: Conv2D | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 32)
block_4_project_BN: BatchNormalization | float32 | input: (None, 28, 28, 32) | output: (None, 28, 28, 32)
block_4_add: Add | float32 | input: [(None, 28, 28, 32), (None, 28, 28, 32)] | output: (None, 28, 28, 32)
block_5_expand: Conv2D | float32 | input: (None, 28, 28, 32) | output: (None, 28, 28, 192)
block_5_expand_BN: BatchNormalization | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_5_expand_relu: ReLU | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_5_depthwise: DepthwiseConv2D | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_5_depthwise_BN: BatchNormalization | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_5_depthwise_relu: ReLU | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_5_project: Conv2D | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 32)
block_5_project_BN: BatchNormalization | float32 | input: (None, 28, 28, 32) | output: (None, 28, 28, 32)
block_5_add: Add | float32 | input: [(None, 28, 28, 32), (None, 28, 28, 32)] | output: (None, 28, 28, 32)
block_6_expand: Conv2D | float32 | input: (None, 28, 28, 32) | output: (None, 28, 28, 192)
block_6_expand_BN: BatchNormalization | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_6_expand_relu: ReLU | float32 | input: (None, 28, 28, 192) | output: (None, 28, 28, 192)
block_6_pad: ZeroPadding2D | float32 | input: (None, 28, 28, 192) | output: (None, 29, 29, 192)
block_6_depthwise: DepthwiseConv2D | float32 | input: (None, 29, 29, 192) | output: (None, 14, 14, 192)
block_6_depthwise_BN: BatchNormalization | float32 | input: (None, 14, 14, 192)

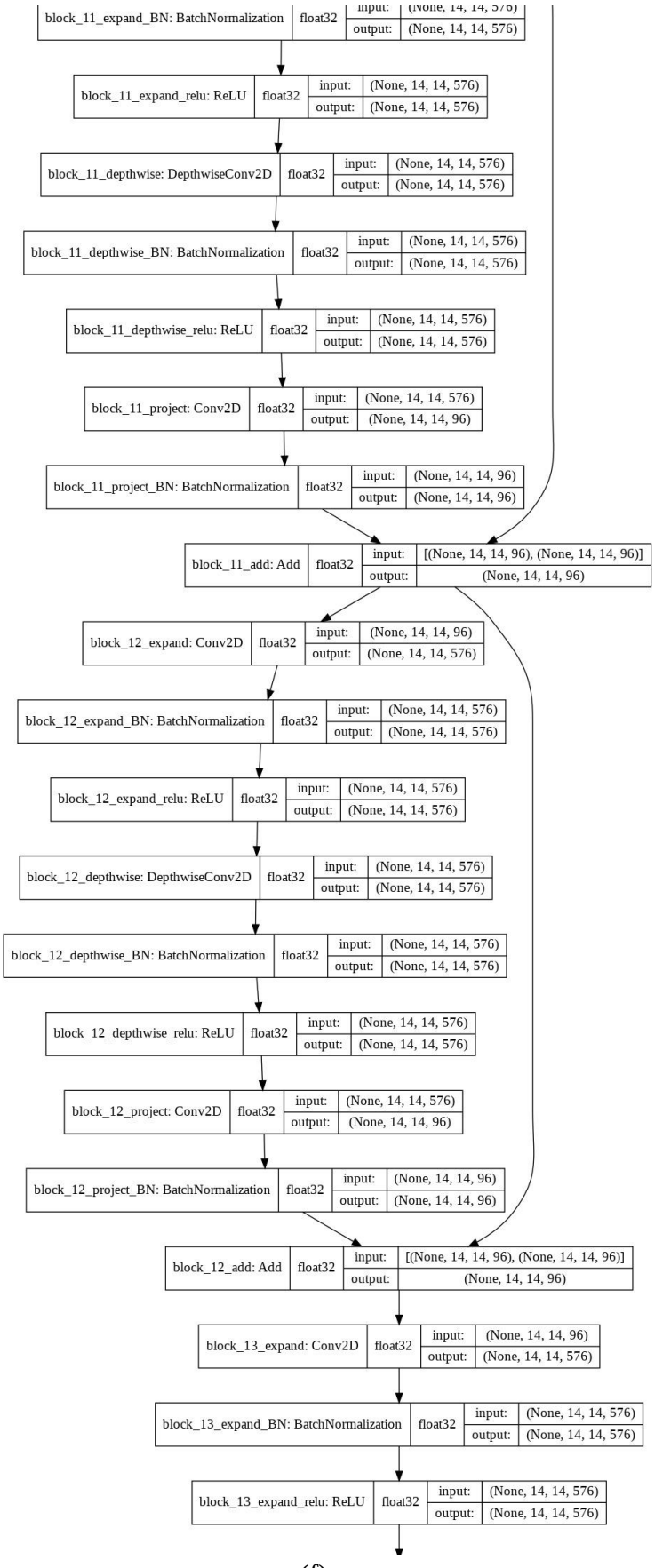
(c)


block_6_depthwise_relu: ReLU | float32 | input: (None, 14, 14, 192) | output: (None, 14, 14, 192)
block_6_project: Conv2D | float32 | input: (None, 14, 14, 192) | output: (None, 14, 14, 64)
block_6_project_BN: BatchNormalization | float32 | input: (None, 14, 14, 64) | output: (None, 14, 14, 64)
block_7_expand: Conv2D | float32 | input: (None, 14, 14, 64) | output: (None, 14, 14, 384)
block_7_expand_BN: BatchNormalization | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_7_expand_relu: ReLU | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_7_depthwise: DepthwiseConv2D | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_7_depthwise_BN: BatchNormalization | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_7_depthwise_relu: ReLU | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_7_project: Conv2D | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 64)
block_7_project_BN: BatchNormalization | float32 | input: (None, 14, 14, 64) | output: (None, 14, 14, 64)
block_7_add: Add | float32 | input: [(None, 14, 14, 64), (None, 14, 14, 64)] | output: (None, 14, 14, 64)
block_8_expand: Conv2D | float32 | input: (None, 14, 14, 64) | output: (None, 14, 14, 384)
block_8_expand_BN: BatchNormalization | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_8_expand_relu: ReLU | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_8_depthwise: DepthwiseConv2D | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_8_depthwise_BN: BatchNormalization | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_8_depthwise_relu: ReLU | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_8_project: Conv2D | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 64)

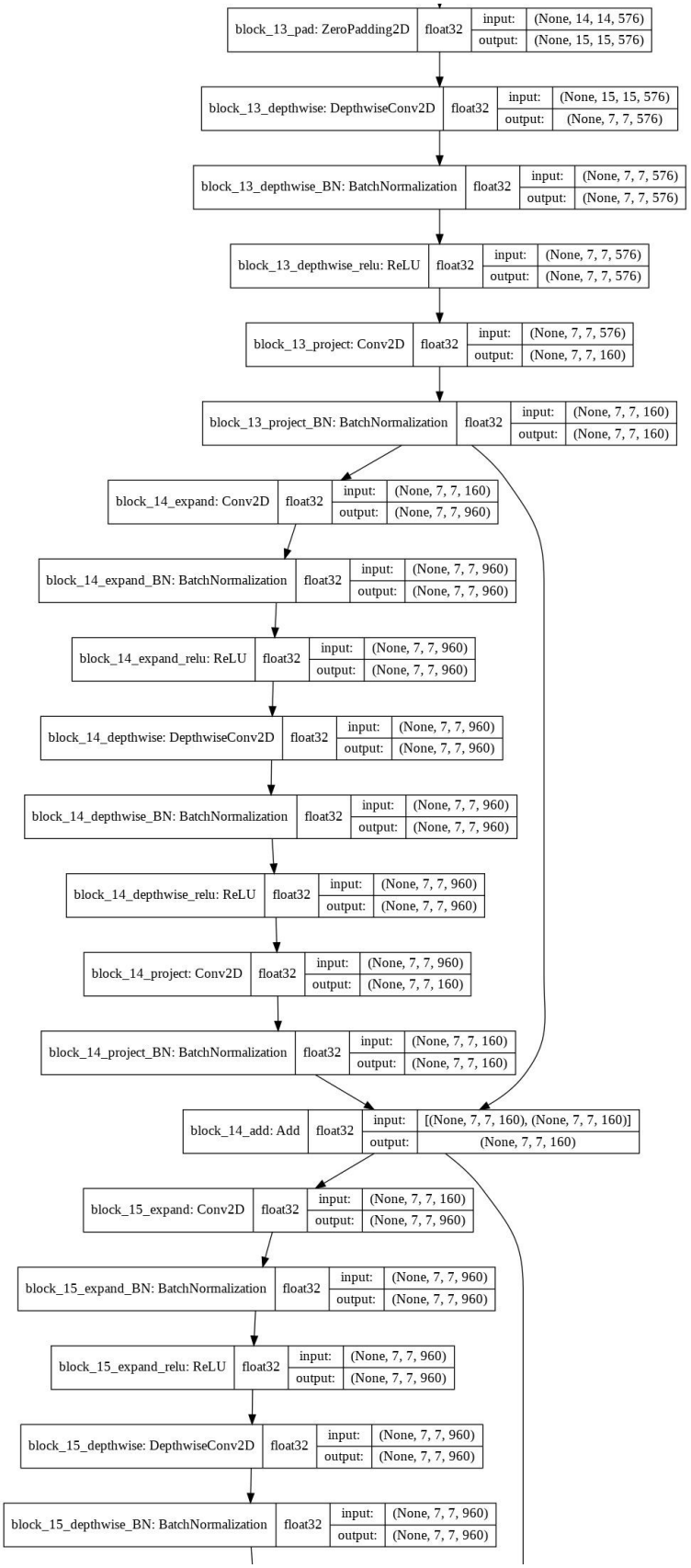
(d)


output: (None, 14, 14, 64)
block_8_add: Add | float32 | input: [(None, 14, 14, 64), (None, 14, 14, 64)] | output: (None, 14, 14, 64)
block_9_expand: Conv2D | float32 | input: (None, 14, 14, 64) | output: (None, 14, 14, 384)
block_9_expand_BN: BatchNormalization | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_9_expand_relu: ReLU | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_9_depthwise: DepthwiseConv2D | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_9_depthwise_BN: BatchNormalization | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_9_depthwise_relu: ReLU | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_9_project: Conv2D | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 64)
block_9_project_BN: BatchNormalization | float32 | input: (None, 14, 14, 64) | output: (None, 14, 14, 64)
block_9_add: Add | float32 | input: [(None, 14, 14, 64), (None, 14, 14, 64)] | output: (None, 14, 14, 64)
block_10_expand: Conv2D | float32 | input: (None, 14, 14, 64) | output: (None, 14, 14, 384)
block_10_expand_BN: BatchNormalization | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_10_expand_relu: ReLU | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_10_depthwise: DepthwiseConv2D | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_10_depthwise_BN: BatchNormalization | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_10_depthwise_relu: ReLU | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 384)
block_10_project: Conv2D | float32 | input: (None, 14, 14, 384) | output: (None, 14, 14, 96)
block_10_project_BN: BatchNormalization | float32 | input: (None, 14, 14, 96) | output: (None, 14, 14, 96)
block_11_expand: Conv2D | float32 | input: (None, 14, 14, 96) | output: (None, 14, 14, 576)

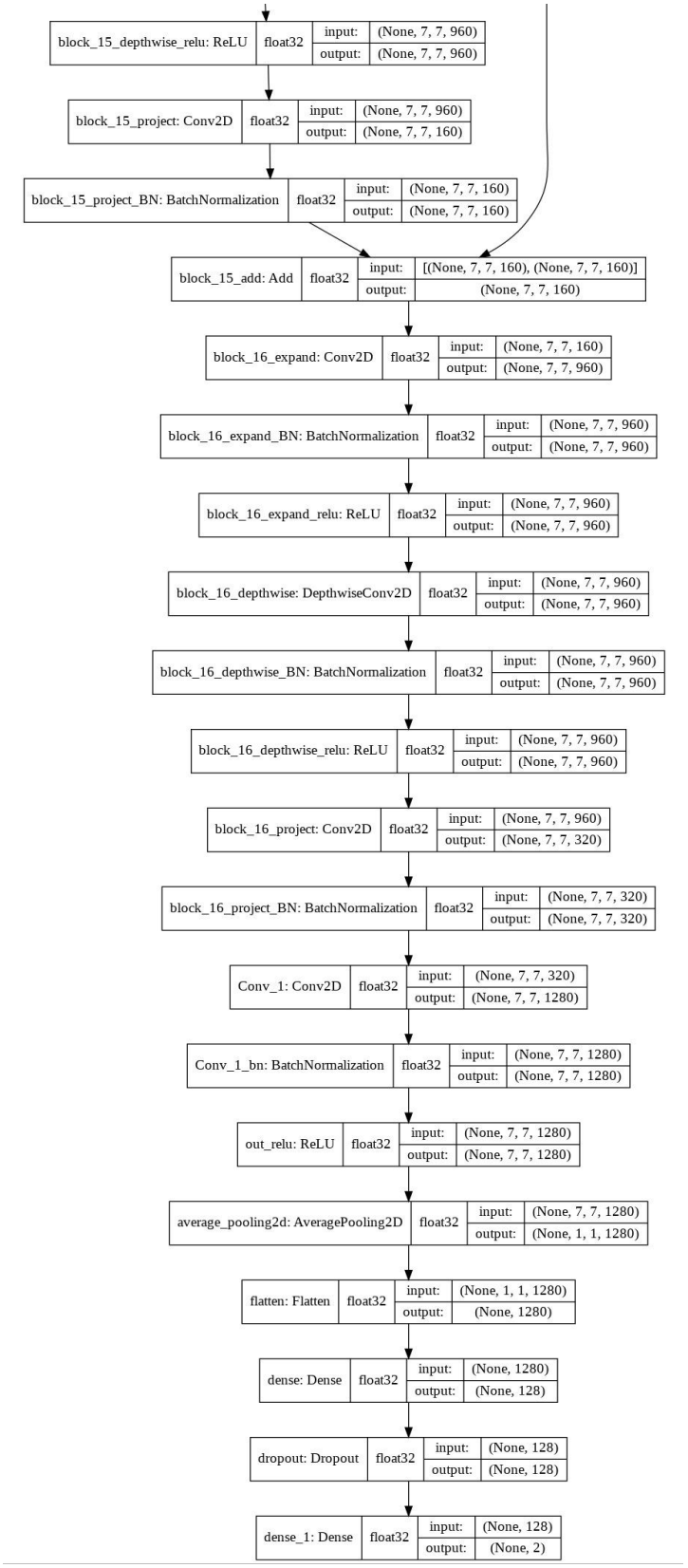
(e)


block_11_expand_BN: BatchNormalization | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_11_expand_relu: ReLU | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_11_depthwise: DepthwiseConv2D | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_11_depthwise_BN: BatchNormalization | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_11_depthwise_relu: ReLU | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_11_project: Conv2D | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 96)
block_11_project_BN: BatchNormalization | float32 | input: (None, 14, 14, 96) | output: (None, 14, 14, 96)
block_11_add: Add | float32 | input: [(None, 14, 14, 96), (None, 14, 14, 96)] | output: (None, 14, 14, 96)
block_12_expand: Conv2D | float32 | input: (None, 14, 14, 96) | output: (None, 14, 14, 576)
block_12_expand_BN: BatchNormalization | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_12_expand_relu: ReLU | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_12_depthwise: DepthwiseConv2D | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_12_depthwise_BN: BatchNormalization | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_12_depthwise_relu: ReLU | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_12_project: Conv2D | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 96)
block_12_project_BN: BatchNormalization | float32 | input: (None, 14, 14, 96) | output: (None, 14, 14, 96)
block_12_add: Add | float32 | input: [(None, 14, 14, 96), (None, 14, 14, 96)] | output: (None, 14, 14, 96)
block_13_expand: Conv2D | float32 | input: (None, 14, 14, 96) | output: (None, 14, 14, 576)
block_13_expand_BN: BatchNormalization | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)
block_13_expand_relu: ReLU | float32 | input: (None, 14, 14, 576) | output: (None, 14, 14, 576)

(f)


block_13_pad: ZeroPadding2D | float32 | input: (None, 14, 14, 576) | output: (None, 15, 15, 576)
block_13_depthwise: DepthwiseConv2D | float32 | input: (None, 15, 15, 576) | output: (None, 7, 7, 576)
block_13_depthwise_BN: BatchNormalization | float32 | input: (None, 7, 7, 576) | output: (None, 7, 7, 576)
block_13_depthwise_relu: ReLU | float32 | input: (None, 7, 7, 576) | output: (None, 7, 7, 576)
block_13_project: Conv2D | float32 | input: (None, 7, 7, 576) | output: (None, 7, 7, 160)
block_13_project_BN: BatchNormalization | float32 | input: (None, 7, 7, 160) | output: (None, 7, 7, 160)
block_14_expand: Conv2D | float32 | input: (None, 7, 7, 160) | output: (None, 7, 7, 960)
block_14_expand_BN: BatchNormalization | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)
block_14_expand_relu: ReLU | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)
block_14_depthwise: DepthwiseConv2D | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)
block_14_depthwise_BN: BatchNormalization | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)
block_14_depthwise_relu: ReLU | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)
block_14_project: Conv2D | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 160)
block_14_project_BN: BatchNormalization | float32 | input: (None, 7, 7, 160) | output: (None, 7, 7, 160)
block_14_add: Add | float32 | input: [(None, 7, 7, 160), (None, 7, 7, 160)] | output: (None, 7, 7, 160)
block_15_expand: Conv2D | float32 | input: (None, 7, 7, 160) | output: (None, 7, 7, 960)
block_15_expand_BN: BatchNormalization | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)
block_15_expand_relu: ReLU | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)
block_15_depthwise: DepthwiseConv2D | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)
block_15_depthwise_BN: BatchNormalization | float32 | input: (None, 7, 7, 960) | output: (None, 7, 7, 960)

(g)



(h)

Fig. 47. Arquitectura del modelo de red neuronal propuesto (a)(b)(c)(d)(e)(f)(g)(h).
Anexo 6. Arquitectura del modelo de red neuronal propuesto.

Anexo 7. Prueba de índice de confianza de ResNet50

Fig. 48. Prueba de índice de confianza de ResNet50.
Anexo 7. Prueba de índice de confianza de ResNet50.

Anexo 8. Prueba de distancia máxima de detección de ResNet50

(a)

(b)

(c)

Fig. 49. Prueba de distancia máxima de detección de resNet50
Anexo 8. Prueba de distancia máxima de detección de resNet50

Anexo 9. Prueba de detección de rostro

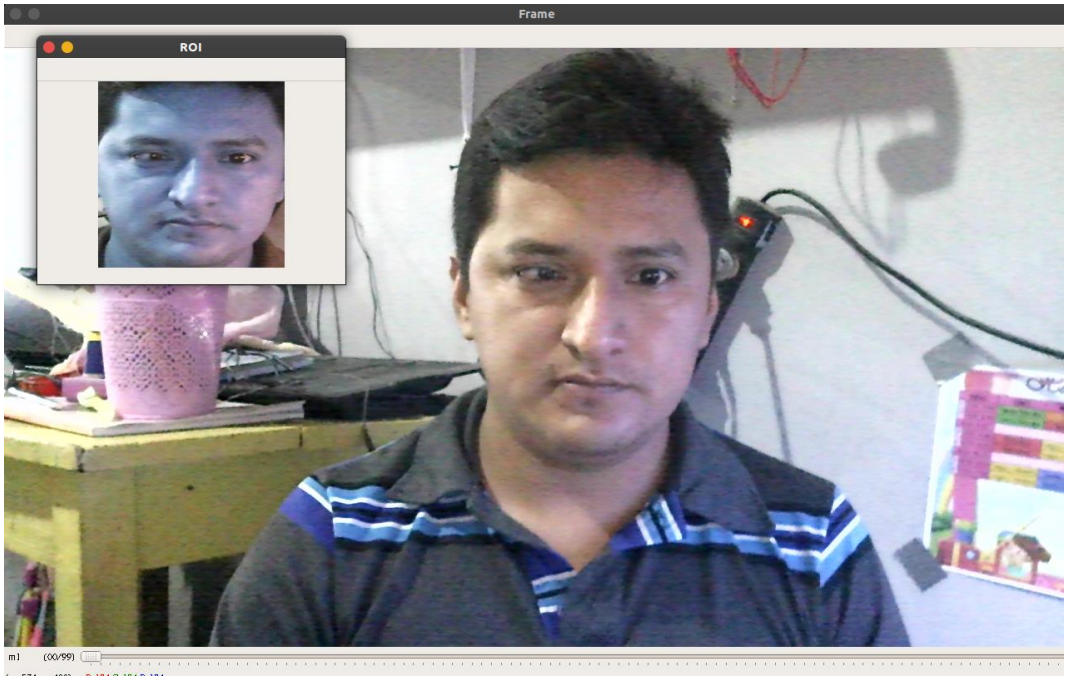


Fig. 50. Prueba de detección de rostro
Anexo 9. Prueba de detección de rostro

Anexo 10. Prueba de detección de rostros sin mascarillas

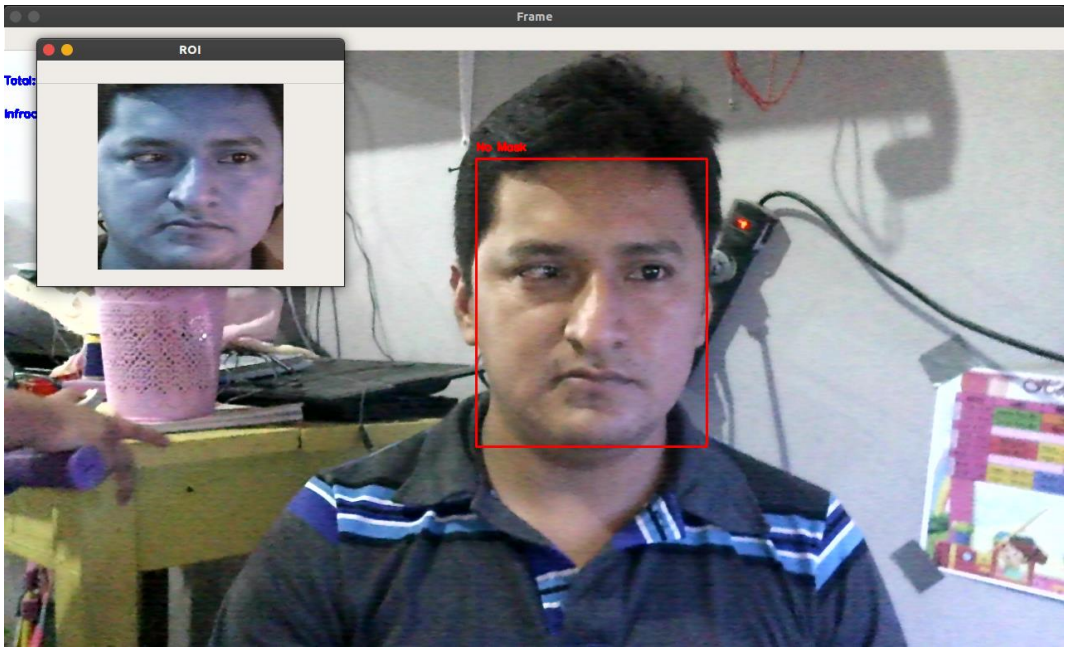
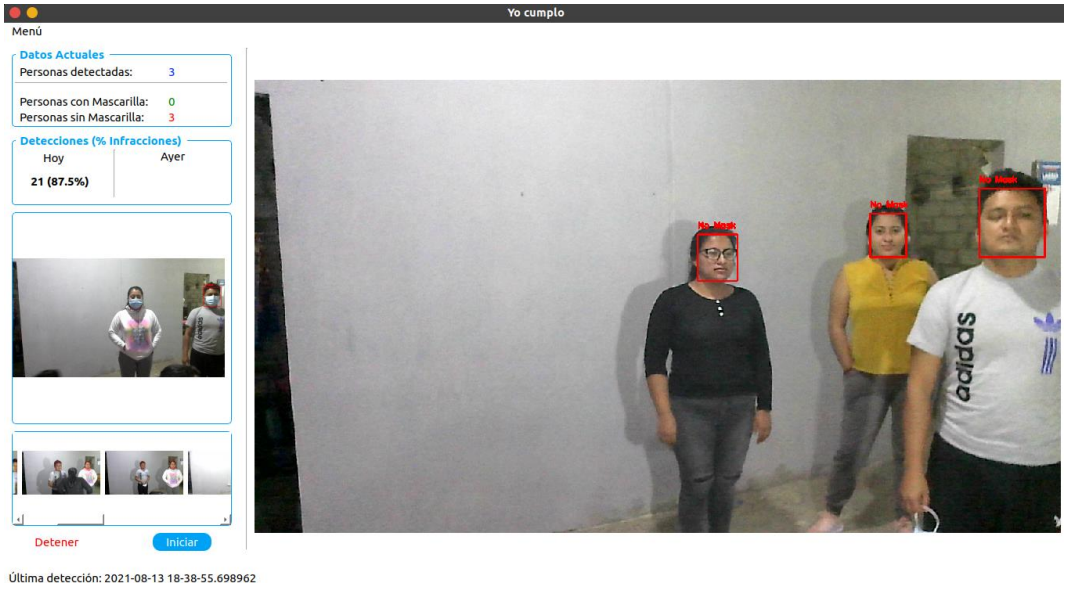
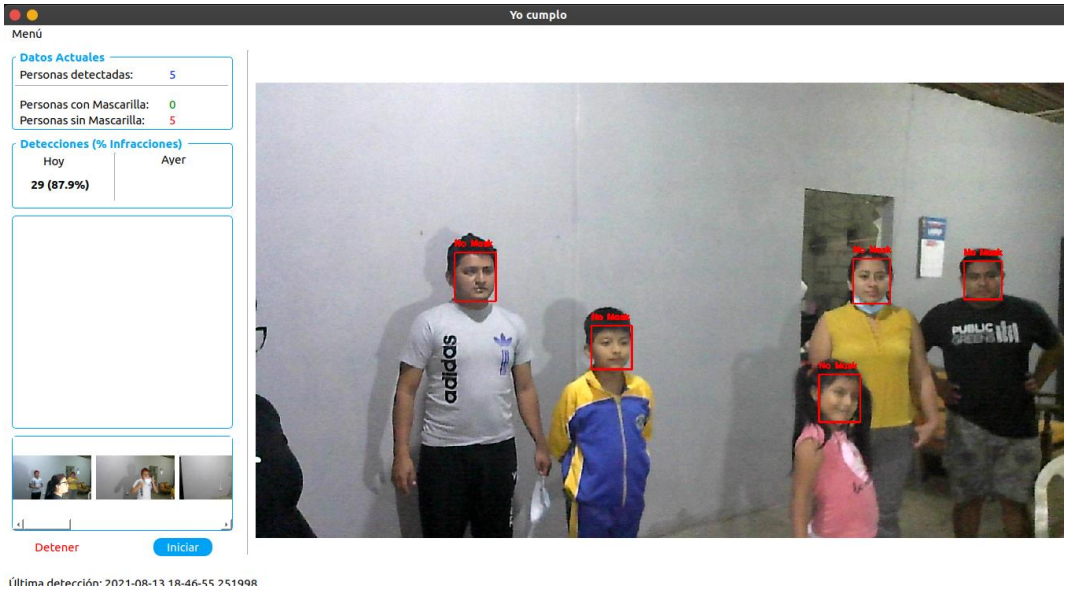


Fig. 51. Prueba de detección de rostros sin mascarillas
Anexo 10. Prueba de detección de rostros sin mascarillas

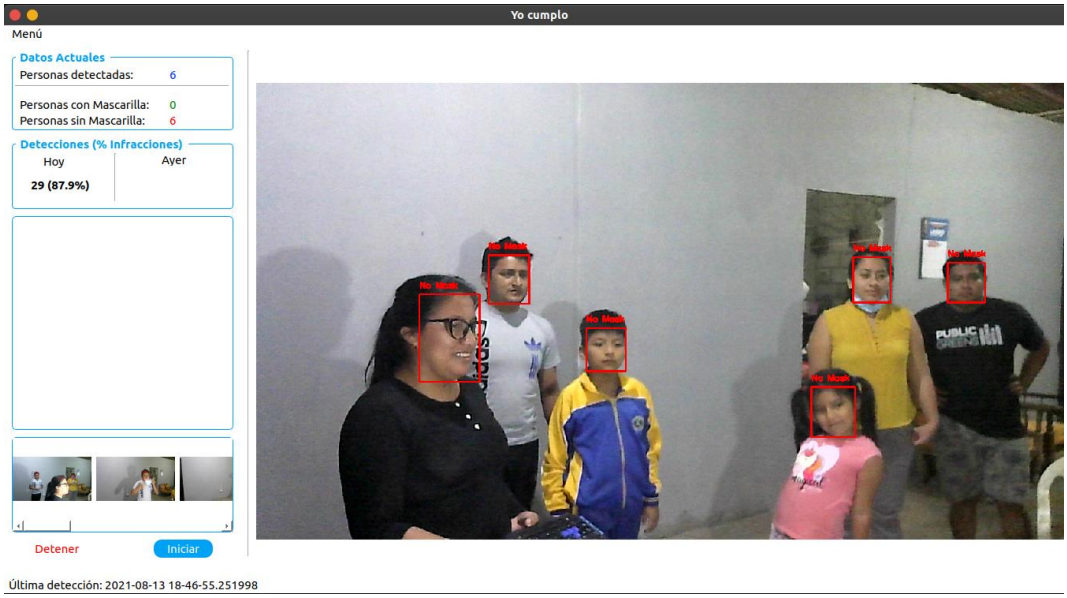
Anexo 11. Prueba de detección de rostros sin mascarillas desde la aplicación



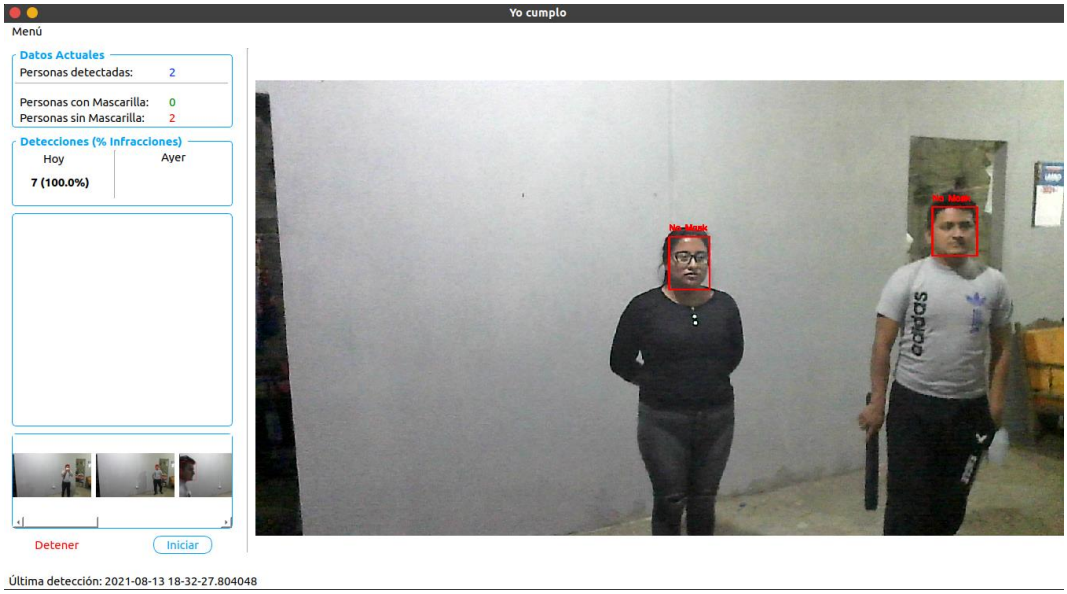
(a)



(b)



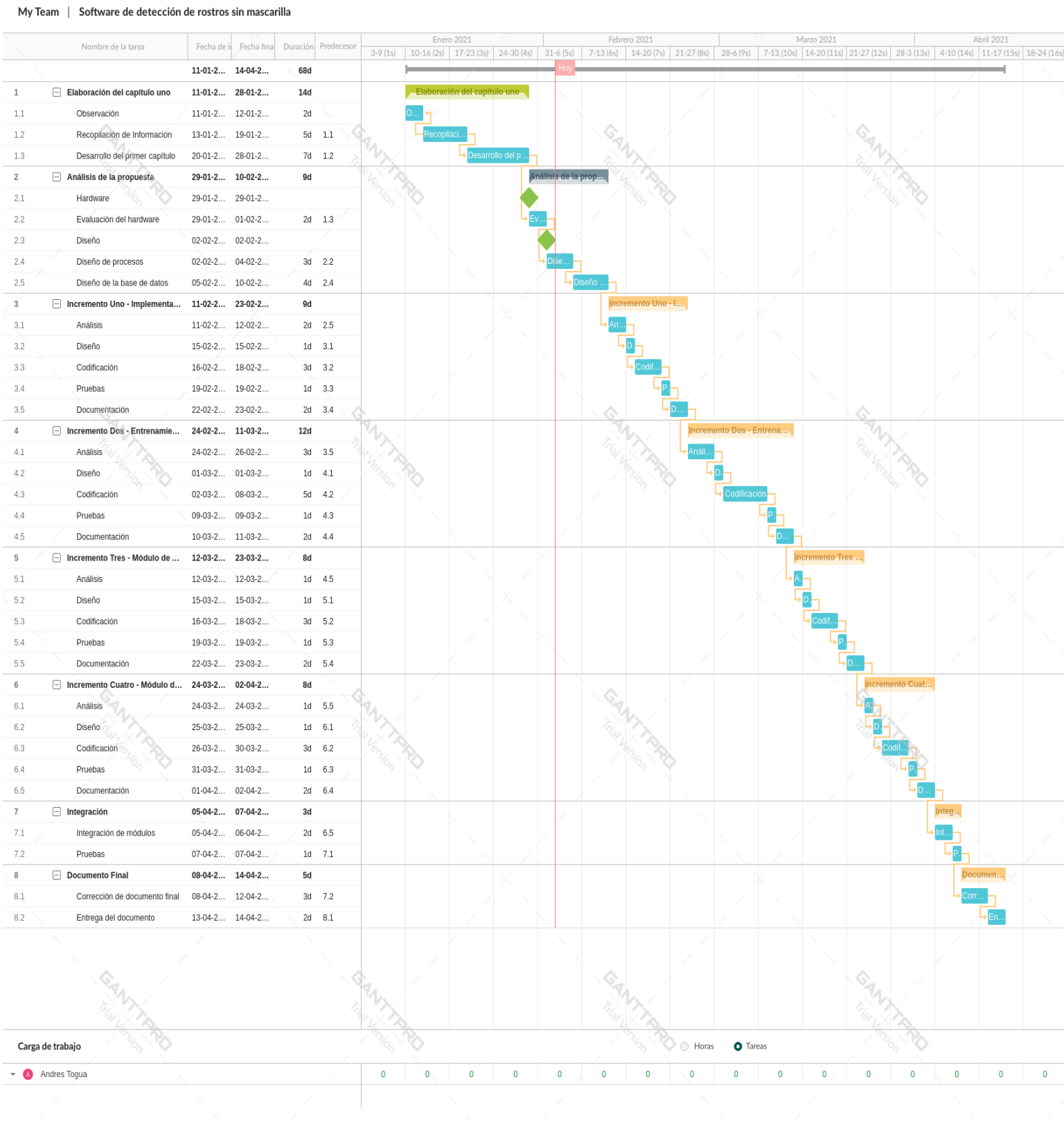
(c)



(d)

Fig. 52. Detección de rostros sin mascarillas desde la aplicación.
Anexo 11. Detección de rostros sin mascarillas desde la aplicación

Anexo 12. Cronograma de trabajo



Anexo 12.Cronograma de trabajo