



UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO DEL USO
DE MASCARILLAS COMO MEDIDA DE PRECAUCIÓN CONTRA EL
COVID19 USANDO DEEP LEARNING

MONTESDEOCA ORDOÑEZ ERIK DANIEL
INGENIERO DE SISTEMAS

MACHALA
2020



UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO
DEL USO DE MASCARILLAS COMO MEDIDA DE PRECAUCIÓN
CONTRA EL COVID19 USANDO DEEP LEARNING

MONTESDEOCA ORDOÑEZ ERIK DANIEL
INGENIERO DE SISTEMAS

MACHALA
2020



UTMACH

FACULTAD DE INGENIERÍA CIVIL

CARRERA DE INGENIERÍA DE SISTEMAS

TRABAJO TITULACIÓN
PROPUESTAS TECNOLÓGICAS

IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO DEL USO DE
MASCARILLAS COMO MEDIDA DE PRECAUCIÓN CONTRA EL COVID19
USANDO DEEP LEARNING

MONTESDEOCA ORDOÑEZ ERIK DANIEL
INGENIERO DE SISTEMAS

RIVAS ASANZA WILMER BRAULIO

MACHALA, 18 DE DICIEMBRE DE 2020

MACHALA
2020

TESIS ERICK MONTESDEOCA

INFORME DE ORIGINALIDAD

6%

INDICE DE SIMILITUD

5%

FUENTES DE
INTERNET

0%

PUBLICACIONES

5%

TRABAJOS DEL
ESTUDIANTE

FUENTES PRIMARIAS

1

Submitted to Universidad Técnica de Machala

Trabajo del estudiante

2%

2

repositorio.utmachala.edu.ec

Fuente de Internet

1%

3

Submitted to Universidad de la Rioja

Trabajo del estudiante

1%

4

dehesa.unex.es

Fuente de Internet

1%

5

Submitted to Universidad San Francisco de
Quito

Trabajo del estudiante

<1%

6

www.slideshare.net

Fuente de Internet

<1%

7

"8th European Medical and Biological
Engineering Conference", Springer Science and
Business Media LLC, 2021

Publicación

<1%

8

Julio Ernesto Suárez Páez. "Arquitectura de

CLÁUSULA DE CESIÓN DE DERECHO DE PUBLICACIÓN EN EL REPOSITORIO DIGITAL INSTITUCIONAL

El que suscribe, MONTESDEOCA ORDOÑEZ ERIK DANIEL, en calidad de autor del siguiente trabajo escrito titulado IMPLEMENTACIÓN DE UN SISTEMA DE RECONOCIMIENTO DEL USO DE MASCARILLAS COMO MEDIDA DE PRECAUCIÓN CONTRA EL COVID19 USANDO DEEP LEARNING, otorga a la Universidad Técnica de Machala, de forma gratuita y no exclusiva, los derechos de reproducción, distribución y comunicación pública de la obra, que constituye un trabajo de autoría propia, sobre la cual tiene potestad para otorgar los derechos contenidos en esta licencia.

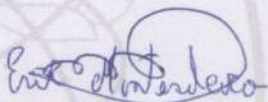
El autor declara que el contenido que se publicará es de carácter académico y se enmarca en las disposiciones definidas por la Universidad Técnica de Machala.

Se autoriza a transformar la obra, únicamente cuando sea necesario, y a realizar las adaptaciones pertinentes para permitir su preservación, distribución y publicación en el Repositorio Digital Institucional de la Universidad Técnica de Machala.

El autor como garante de la autoría de la obra y en relación a la misma, declara que la universidad se encuentra libre de todo tipo de responsabilidad sobre el contenido de la obra y que asume la responsabilidad frente a cualquier reclamo o demanda por parte de terceros de manera exclusiva.

Aceptando esta licencia, se cede a la Universidad Técnica de Machala el derecho exclusivo de archivar, reproducir, convertir, comunicar y/o distribuir la obra mundialmente en formato electrónico y digital a través de su Repositorio Digital Institucional, siempre y cuando no se lo haga para obtener beneficio económico.

Machala, 18 de diciembre de 2020



MONTESDEOCA ORDOÑEZ ERIK DANIEL
0705833028



DEDICATORIA

Dedico de todo corazón este trabajo de titulación a mis padres por haberme educado, apoyado en los momentos difíciles y forjado como la persona que soy hoy en día, muchos de mis logros se los debo a ustedes y este es otro logro que lo he logrado culminar gracias a ustedes.

Erik Daniel Montesdeoca Ordoñez

AGRADECIMIENTO

En mi primer lugar le doy gracias Dios por la vida y salud que nos ha brindado a mis padres y mi persona para poder terminar esta etapa de estudio.

A mis padres por haber estado en los momentos difíciles de mi carrera de estudio, por el apoyo que siempre me han brindado y la motivación para seguir adelante.

También estoy muy agradecido con el tutor Ing. Wilmer Rivas por la amistad, apoyo y orientación brindada, con todos los docentes de la carrera que aportaron sus enseñanzas cada día durante el ciclo de mi carrera.

Erik Daniel Montesdeoca Ordoñez

RESUMEN

El nivel de contagio de COVID 19 en el mundo ha alcanzado un numero critico por no acatar las correctas normas de bioseguridad, Ecuador a pesar de ser un país pequeño posee una cifra grande y para evitar que el número de contagios se incremente el gobierno junto al ministerio de salud han indicado varios medios de bioseguridad que los habitantes deben cumplir.

Utilizar mascarillas KN95 es uno de los métodos de bioseguridad sugerido por el ministerio de salud y para poder llevar un control del uso de mascarillas se propuso desarrollar un sistema que realice la detección de personas que están utilizando y personas que no están utilizando mascarillas para así poder llevar un control más estricto.

La inteligencia artificial ha permitido que el ser humano sea capaz de crear y entrenar redes neuronales capaces de pensar por sí misma y realizar la toma de decisiones, en el caso de este proyecto se utilizó redes neuronales convolucionales que realizan la función de detectar que persona está utilizando mascarilla y quien no, con la tecnología e infinidad de librerías que nos ofrece Python, OpenCV y Tensorflow se construyó y entrenó la red neuronal.

Para construir la red neuronal se construyó una dataset con una cantidad de 1500 fotografías a las que se las clasifico en dos grupos: el primer grupo de fotografías de personas utilizando mascarillas (with mask) y el segundo grupo de fotografías de personas que no están utilizando mascarilla (without mask).

Las fotografías para la creación de la dataset se las obtuvo de la siguiente manera, una cantidad fueron tomadas mediante un smartphone teniendo en cuenta que la fotografía sea de calidad alta, que la iluminación sea alta y que la persona no tenga accesorios que cubran su rostro y el resto de imágenes se las obtuvo de la internet.

En la actualidad existen diversas técnicas para realizar reconocimiento de objetos, para este trabajo se utilizó MobileNetV2 que es una mejora de MobileNetV1 la cual se especializa en el arte del reconocimiento, clasificación, detección de objetos y la segmentación semántica. Una vez creada y entrenada

la red neuronal con la dataset creada anteriormente se procedió a realizar pruebas de la red neuronal.

Se entreno 4 redes neuronales con diferentes parámetros a los cuales se las evaluó y aplicando métricas se seleccionó la red neuronal más eficiente.

Para la implementación de la red neuronal se desarrolló una página web en la que el usuario puede observar e interactuar con la red neuronal. El sistema web nos permite ingresar como entrada una imagen, un video, streaming en tiempo real y como salida nos muestra un cuadro de color verde alrededor del rostro de la persona que está usando mascarilla y un cuadro de color rojo alrededor del rostro de la persona que no está usando mascarilla con un sonido de alarma alertando que no está cumpliendo con las medidas de bioseguridad.

Se realizo pruebas en tiempo real con diferentes mascarillas con la cual permitió ver el alcance de la red neuronal entrenada.

Palabras clave: detección de objetos, reconocimiento de mascarilla, python, covid-19, mobilenetv2

ABSTRACT

The level of contagion of COVID 19 in the world has reached a critical number for not complying with the correct biosecurity regulations, Ecuador despite being a small country has a large number and to prevent the number of infections from increasing the government together with the The Ministry of Health has indicated various means of biosecurity that the inhabitants must comply with.

Using KN95 masks is one of the biosafety methods suggested by the Ministry of Health and in order to control the use of masks, it was proposed to develop a system to detect people who are using and people who are not using masks to thus to be able to keep a stricter control.

Artificial intelligence has allowed the human being to be able to create and train neural networks capable of thinking for themselves and making decisions, in the case of this project convolutional neural networks were used that perform the function of detecting which person is Using a mask and who doesn't, with the

technology and countless libraries offered by python, OpenCV and Tensorflow, the neural network was built and trained.

To build the neural network, a dataset was built with a quantity of 1500 photographs, which were classified into two groups: the first group of photographs of people using masks (with mask) and the second group of photographs of people who are not using mask (without mask).

The photographs for the creation of the dataset were obtained as follows, a quantity was taken using a smartphone taking into account that the photograph is of high quality, that the lighting is high and that the person does not have accessories that cover their face and the rest of the images were obtained from the internet.

Currently there are various techniques to perform object recognition, for this work MobileNetV2 was used, which is an improvement of MobileNetV1 which specializes in the art of recognition, classification, detection of objects and semantic segmentation. Once the neural network was created and trained with the previously created dataset, the neural network was tested.

4 neural networks were trained with different parameters to which they were evaluated and applying metrics, the most efficient neural network was selected.

For the implementation of the neural network, a web page was developed in which the user can observe and interact with the neural network. The web system allows us to input an image, a video, streaming in real time and as output it shows us a green box around the face of the person who is wearing a mask and a red box around the person's face that he is not wearing a mask with an alarm sound alerting that he is not complying with biosafety measures.

Real-time tests were carried out with different masks with which it was possible to see the scope of the trained neural network.

Keywords: object detection, mask recognition, python, covid19, mobilenetv2

ÍNDICE DE CONTENIDO

DEDICATORIA	1
AGRADECIMIENTO	2
RESUMEN	3
ABSTRACT	4
INTRODUCCIÓN	11
1. CAPÍTULO I. DIAGNÓSTICO DE NECESIDADES Y REQUERIMIENTOS	
12	
1.1. Ámbito de Aplicación: descripción del contexto y hechos de interés ..	12
1.2. Establecimiento de requerimientos	13
1.3. Justificación de requerimiento a satisfacer	14
2. CAPÍTULO II. DESARROLLO DEL PROTOTIPO	15
2.1. Definición del prototipo tecnológico	15
2.2. Fundamentación teórica del prototipo.	16
2.2.1. Entorno de Trabajo	17
2.2.2. Técnicas de Tratamiento de Imágenes	19
2.2.3. Comparación de Algoritmos.....	24
2.2.4. Construcción de la Red Neuronal	25
2.3. Metodología	25
2.4. Objetivos del prototipo	26
2.4.1. Objetivo General.....	26
2.4.2. Objetivo Específicos	26
2.5. Diseño del prototipo	27
2.5.1. Requisitos.....	27
2.5.2. Elaboración de la Dataset.....	27
2.5.3. Diseño y entrenamiento de la red	28

2.5.4. Algoritmo de Reconocimiento	30
2.5.5. Desarrollo de la interfaz web	31
2.6. Ejecución y/o ensamblaje del prototipo	34
3. EVALUACIÓN DEL PROTOTIPO	42
3.1. Plan de evaluación	42
3.1.1. Prueba de Entrenamiento y Validación de la Red Neuronal	42
3.2. Resultado de evaluación	43
3.1.2. Resultado de la Prueba de Entrenamiento y Validación	43
3.2. Conclusiones	47
3.3. Recomendaciones	47
4. BIBLIOGRAFIA.....	48
5. ANEXOS.....	52

ÍNDICE DE FIGURAS

Figura 1: Arquitectura de entrenamiento Red Neuronal	15
Figura 2: Arquitectura Detección Mascarilla	16
Figura 3: Mapa mental de la Fundamentación Teórica del Prototipo	16
Figura 4: Estructura CPU	18
Figura 5: Estructura GPU	19
Figura 6: Regiones con funciones CNN	20
Figura 7: MobilteNetV1	21
Figura 8: MobileNetV2	22
Figura 9: Regiones con funciones Fast R-CNN	23
Figura 10: Regiones con funciones Faster R-CNN	24
Figura 11: Comparación de la velocidad del tiempo de prueba de los algoritmos de detección de objetos	24
Figura 12: Estructura de dataset	28
Figura 13: Renombrado de imágenes con Flexxi	28
Figura 14: Diseño de Red Neuronal	29
Figura 15: Entrenamiento de la red neuronal	29
Figura 16: Estructura de la interfaz web	30
Figura 17: Archivos que realizaran la detección en tiempo real	30
Figura 18: Estructura de la base de datos	31
Figura 19: Vista Inicio	31
Figura 20: Vista Streaming	32
Figura 21: Vista Subir video	32
Figura 22: Vista Video	33
Figura 23: Vista Subir imagen	33
Figura 24: Vista Imagen	34
Figura 25: Ejecución de sstreaming.py	35
Figura 26: Ejecución svideo.py	35
Figura 27: Ejecución simagen.py	35
Figura 28: Vista inicia visualizada desde el navegador web	36
Figura 29: Detección en tiempo real, mascarilla A	36
Figura 30: Detección en tiempo real, mascarilla B	37

Figura 31: Detección en tiempo real, sin mascarilla	37
Figura 32: Selección de video para detección en tiempo real.....	38
Figura 33: Subida correcta del video al servidor.....	38
Figura 34: Detección en tiempo real de video subido al servidor.....	39
Figura 35: Selección de imagen	39
Figura 36: Subida correcta de la imagen al servidor	40
Figura 37: Detección en tiempo real de imagen subida al servidor	40

ÍNDICE DE TABLAS

Tabla 1: Parámetros de entrenamiento prueba 1	43
Tabla 2: Parámetros de entrenamiento prueba 2	44
Tabla 3: Parámetros de entrenamiento prueba 3	44
Tabla 4: Parámetros de entrenamiento prueba 4	44
Tabla 5: Resultado de entrenamiento prueba 1	45
Tabla 6: Resultado de entrenamiento prueba 2.....	45
Tabla 7: Resultado de entrenamiento prueba 3.....	45
Tabla 8: Resultado de entrenamiento prueba 4.....	46

INTRODUCCIÓN

Actualmente la detección de objetos ha tenido un gran impacto en las diferentes áreas como por ejemplo en el área de medicina para lograr identificar una enfermedad a tiempo y poder prevenirla, en la robótica, en la programación, videovigilancia, reconocimiento facial entre otras.

Lograr que una computadora aprenda a tomar sus propias dediciones aplicando la inteligencia artificial en conjunto con las redes neuronales es un gran reto. Se propone desarrollar un sistema capaz de detectar personas que están utilizando mascarillas y personas que no están utilizando mascarillas en tiempo real.

Para lograr esto se necesita una gran cantidad de fotografías, entre más fotografías se utilicen para el entrenamiento más efectivo será su aprendizaje y reconocimiento.

El entrenamiento se lo realizara aplicando MobileNetV2 ya que este utiliza un algoritmo para realizar la detección de objetos en tiempo real. Una vez entrenado el algoritmo se construirá una interfaz web para que el usuario pueda realizar pruebas en tiempo real.

Para la prueba en tiempo real se utilizará una cámara ip como fuente de video principal, pero por características del hardware para su demostración se utilizará la cámara web de la laptop, también se tiene la opción de cargar imágenes y videos para realizar pruebas de detección.

El presente documento consta de la siguiente estructura:

Capítulo 1: se describe la problemática, la justificación y se establecen los requerimientos del prototipo.

Capítulo 2: fundamentación teórica del sistema, diseño del prototipo, ejecución y pruebas del prototipo.

Capítulo 3: se detallan los resultados obtenidos de la aplicación, las conclusiones y recomendaciones.

1. CAPÍTULO I. DIAGNÓSTICO DE NECESIDADES Y REQUERIMIENTOS

1.1. Ámbito de Aplicación: descripción del contexto y hechos de interés

En nuestra provincia de El Oro el número de personas contagiadas con COVID-19 aumentan cada día por la falta del control del uso de mascarillas en las personas que transitan y asisten a sus trabajos en lugares públicos, por tal razón se ha propuesto desarrollar un sistema detector de mascarillas aplicando la inteligencia artificial.

El reconocimiento de uso de mascarillas se lo puede utilizar para algunos fines, en este caso lo utilizaremos para llevar un control en las personas; como el reconocimiento es en tiempo real, esto ayudara a prevenir que las personas se contagien del virus COVID-19.

El ser humano a simple vista puede reconocer de inmediato que persona lleva puesta una mascarilla y quien no, en cambio para un sistema es más complejo hacer dicha detección ya que para realizar un reconocimiento preciso requiere una gran cantidad de datos que depende de algunos factores: calidad de la imagen, ángulo de la persona, que persona no debe llevar accesorios que cubra su rostro o pelo, color de la mascarilla, iluminación, clima.

Actualmente existen diversas técnicas para realizar reconocimiento de objetos, para este trabajo se utilizará MobileNetV2 que es una mejora de MobileNetV1 la cual se especializa en el arte del reconocimiento, clasificación, detección de objetos y la segmentación semántica. Una vez creada y entrenada la red neuronal con una base de datos de fotografías creada a partir de diferentes fuentes se procederá a realizar pruebas de la red neuronal.

El alcance de este proyecto es lograr desarrollar un prototipo que realice la detección de personas que no estén utilizando mascarilla.

1.2. Establecimiento de requerimientos

Para realizar el entrenamiento con MobileNetV2 de Google requiere de algunos parámetros con los cuales se irán realizando pruebas hasta llegar al resultado más eficiente.

El proyecto está estructurado de la siguiente manera:

- Recolección de información, en esta primera fase debemos realizar un estudio sobre el entrenamiento de redes neuronales convoluciones y Deep Learning.
- Crear base de datos de imágenes para entrenar la red neuronal, para crear esta base de datos se utilizó un total de 1500 fotografías de excelente calidad obtenidas de internet y tomadas con cámaras de smartphones de personas las cuales están divididas en dos grupos: 750 de personas utilizando mascarilla y 750 personas sin mascarilla.
- Entrenamiento de la red neuronal, se desarrolló un script en Python en el cual se establece las rutas de los dos grupos de imágenes y los parámetros que se requieren: tasa de aprendizaje inicial, numero de épocas, tamaño del lote.
- Desarrollo de la aplicación web, se desarrollará una interfaz amigable con el usuario en el cual se integrará el archivo con los pesos generados para realizar las detecciones.
- Pruebas, una vez finalizada la aplicación web se procederá a realizar las respectivas pruebas del reconocimiento en tiempo real, con imágenes y con videos.

1.3. Justificación de requerimiento a satisfacer

Este proyecto tiene su enfoque en el Dominio: Tecnologías de la Información y la Comunicación y en la línea de investigación: Ciencias de los datos e inteligencia artificial establecido por la Universidad Técnica de Machala (UTMACH).

Desde la aparición de COVID-19 en China y luego de declarar como una pandemia por parte de la OMS, se han puesto en práctica diferentes medidas de bioseguridad como: el distanciamiento social, restricción de viajes, aislamiento, uso de alcohol en gel, uso de mascarillas KN95 para evitar propagar el virus a más personas y así reducir el número de personas contagiadas [1].

En nuestro país Ecuador a pesar de tener una población de aproximadamente de 17.023.000 habitantes se contabilizan 181.104 personas contagiadas por COVID-19 aproximadamente y 13.025 personas fallecidas aproximadamente [2].

La provincia de El Oro cada día suma más casos de personas contagiadas por COVID-19, por tal razón este trabajo de investigación se enfocará en la detección de uso de mascarillas en las personas como medida de bioseguridad para así tratar de controlar y reducir el número de contagios.

Para ello se revisó fuentes bibliográficas sobre detección en tiempo real en revistas y artículos científicos, tras revisar los diferentes métodos y algoritmos de detección de objetos en tiempo real se seleccionó el algoritmo de MobileNetV2 [3] ya que este algoritmo es más rápido y preciso porque utilizar Faster-RCNN [4].

El prototipo a desarrollarse nos permitirá visualizar en tiempo real la detección del uso de mascarillas en personas a través de una cámara ip que por motivos de seguridad propia se colocó una cámara ip en la entrada de la casa y no en un lugar público, la cual enviara la información a la red neuronal que la procesara y nos mostrara en el navegador web el resultado.

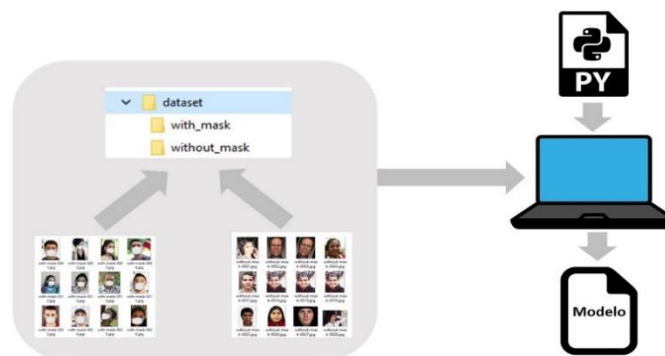
2. CAPÍTULO II. DESARROLLO DEL PROTOTIPO

2.1. Definición del prototipo tecnológico

El Deep Learning (DL) o aprendizaje profundo ha logrado grandes avances recientemente en lo es reconocimiento de imágenes y videos, ya que el Deep Learning permite que los modelos computaciones formados por algunas capas de procesamiento puedan aprender representaciones sobre datos con múltiples niveles de abstracción y mediante ese concepto pueda descubrir representaciones de forma precisa y autónoma en grandes cantidades de datos. MobileNetV2 es una red neuronal convolucional pensada para correr de manera muy eficiente, su construcción y entrenamiento es a partir de una gran cantidad de fotografías con sus respectivos parámetros [5] [6], nuestra red neuronal convolucional consta de 2 capas: la primera es de entrenamiento y la segunda de predicción.

La primera parte del entrenamiento abarca desde la recolección de fotografías hasta obtener el archivo con los pesos. Una vez recolectadas las fotografías creamos una carpeta con el nombre de dataset y dentro crearemos dos carpetas nuevas una con el nombre de with_mask la cual contendrá las 750 fotografías de personas utilizando mascarilla y la otra carpeta con el nombre de without_mask la cual tendrá el resto de fotografías de personas sin usar mascarilla, una vez armada la base de datos con las fotografías a entrenar se procede a crear un script en Python el cual comenzará a entrenar la red neuronal.

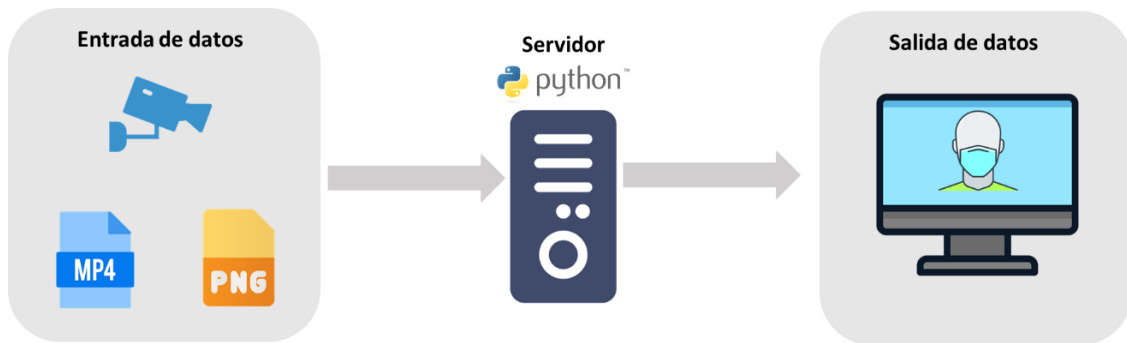
Figura 1: Arquitectura de entrenamiento Red Neuronal



Fuente: Elaboración propia

La siguiente parte lo que concierne a la predicción abarca desde la captura de video en tiempo real (streaming), archivos de videos e imágenes que serán receptados por el programa el cual está conectado a la red neuronal entrenada la que hará la detección en tiempo real y lo mostrará por pantalla.

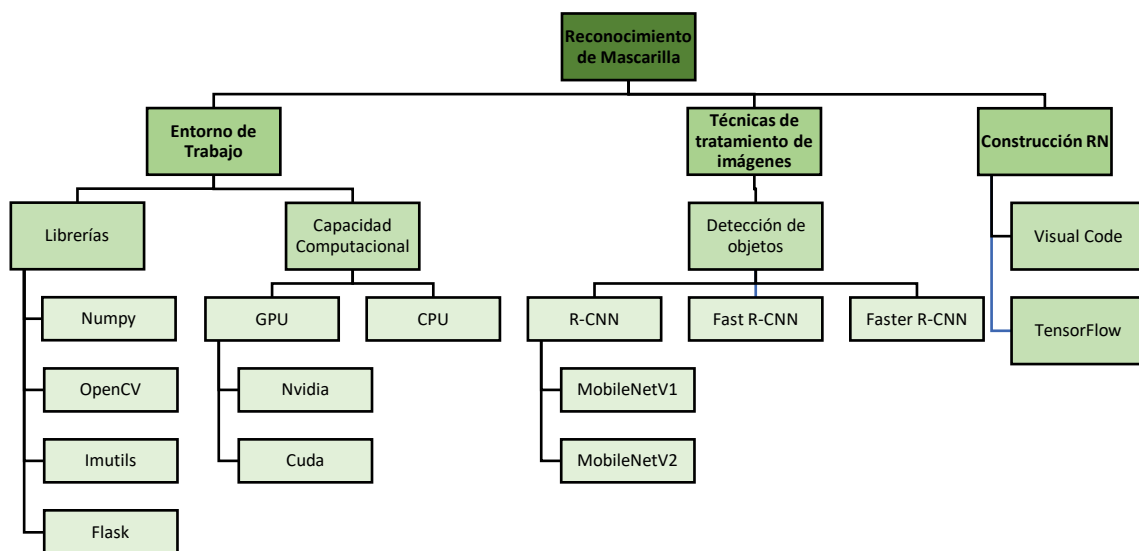
Figura 2: Arquitectura Detección Mascarilla



Fuente: Elaboración propia

2.2. Fundamentación teórica del prototipo.

Figura 3: Mapa mental de la Fundamentación Teórica del Prototipo



Fuente: Elaboración propia

2.2.1. Entorno de Trabajo

2.2.1.1. Librerías

2.2.1.1.1. Numpy

Numpy es una librería que pertenece a la biblioteca de librerías matemáticas la cual se dedica al cálculo sobre vectores y matrices en Python. Esta librería está escrita en lenguaje C la cual es eficiente al momento de trabajar con grandes cantidades de datos. La librería Numpy alcanzado una importancia muy alta que es indispensable para el ámbito de la Ciencia de Datos [7].

2.2.1.1.2. OpenCV

OpenCV es una librería de visión artificial y aprendizaje automático gratuita basada en una licencia de BSD, contiene más de 2500 funciones de procesamiento de video y es de código abierto, también posee una gran importancia para el desarrollo de la Ciencia de Datos [8].

OpenCV es compatible con múltiples sistemas operativos y lenguajes de programación como C++, Python, Java, Matlab [9]. Actualmente sus interfaces están siendo desarrolladas en CUDA y OpenCL.

2.2.1.1.3. Imutils

Imutils es una librería que posee una serie de funciones básicas de procesamiento de imágenes como: cambio de tamaño, traducción, clasificación de contornos, rotación, visualización de imágenes Matplotlib, esqueletización, detección de bordes, que trabaja en conjunto con Python y OpenCV [10].

2.2.1.1.4. Flask

Flask es un microframework de Python que trabaja bajo el patrón de MVC, también utiliza un sistema de plantillas que depende del motor Jinja y del kit de herramientas Werkzeug WSGI, enrutamiento e incluso incluye su propio servidor para ejecutar las aplicaciones. Tiene todo lo necesario para crear una app web básica [11].

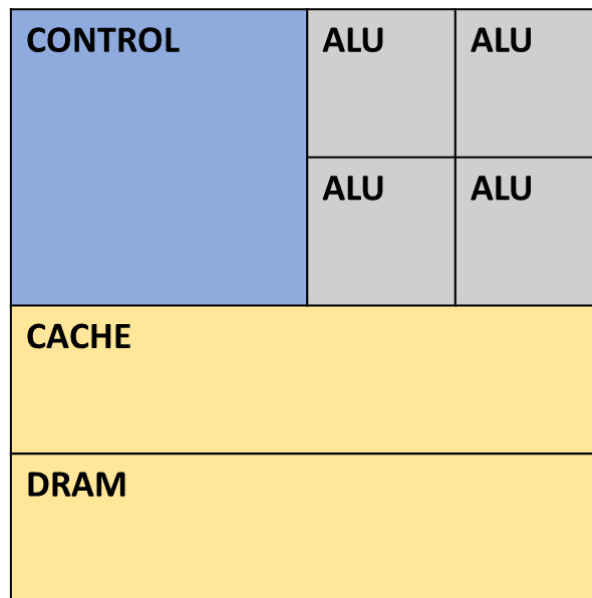
2.2.1.2. Capacidad Computacional

En la actualidad es posible desarrollar y procesar algoritmos poderosos gracias al hardware que ha ido evolucionando con el pasar de los años, ha esto se le conoce como “Capacidad Computacional” o “Poder Computacional”. La CPU y la GPU son indispensables para el desarrollo de estos algoritmos.

2.2.1.2.1. CPU

El CPU (Unidad Central de proceso) es uno de los componentes principales de una computadora la cual se encarga de procesar los datos y realizar cálculos. El CPU cuenta con una ALU (Arithmetic Logic Unit), una DRAM y una Cache [12].

Figura 4: Estructura CPU

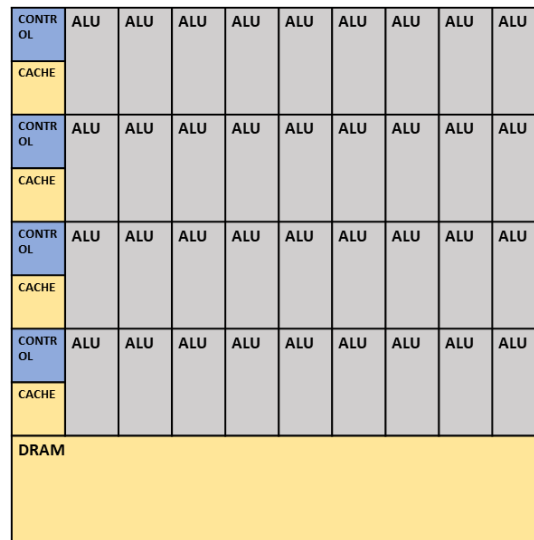


Fuente: Elaboración propia

2.2.1.2.2. GPU

La GPU (Graphics Processing Unit) es uno de los componentes principales de una tarjeta gráfica la cual contiene una serie de millones de transistores que realizaran los cálculos y procesos [12]. A diferencia de la CPU la GPU cuenta con varias ALUs, varios Controles, varios Caches y una DRAM. Su principal uso fue originalmente para procesar imágenes en alta calidad y videojuegos, hoy en día es una herramienta indispensable para el Machine Learning y el Deep Learning [13] [14].

Figura 5: Estructura GPU



Fuente: Elaboración propia

2.2.1.2.3. NVIDIA

Los juegos para computadoras fueron quienes impulsaron al crecimiento de esta empresa NVIDIA dedicada a la fabricación de tarjetas gráficas y actualmente también revoluciono la IA moderna. Gracias a NVIDIA que ha logrado convertir a la GPU en un cerebro para la intersección de la realidad virtual, la inteligencia artificial y la informática de alto rendimiento [15].

2.2.1.2.4. CUDA

CUDA es un modelo de programación desarrollada por la empresa NVIDIA, la cual le permite utilizar la GPU a su máxima capacidad para acelerar rápidamente las aplicaciones desarrolladas.

CUDA es soportada la mayoría de lenguajes de programación populares como MATLAB, C, Python, C++, Fortran. NVIDIA nos ofrece un kit de herramientas para CUDA la cual contiene un compilador, bibliotecas aceleradas por GPU, el tiempo de ejecución de CUDA y las herramientas de desarrollo [16].

2.2.2. Técnicas de Tratamiento de Imágenes

2.2.2.1. Detección de Objetos

Nosotros los seres humanos podemos observar una fotografía e inmediatamente reconocer cualquier objeto que contenga, también observamos algunas características por ejemplo si hay objetos grandes o pequeños, si está claro, oscuro o borrosa; en cambio para que el computador pueda realizar la detección

es necesario crear algoritmos utilizando Machine Learning o Deep Learning que esto implica a desarrollar y entrenar una red convolucional para que pueda detectar algunas características limitas de los objetos [17]. Entre los algoritmos más conocidos tenemos R-CNN, Fast R-CNN y Faster R-CNN [18].

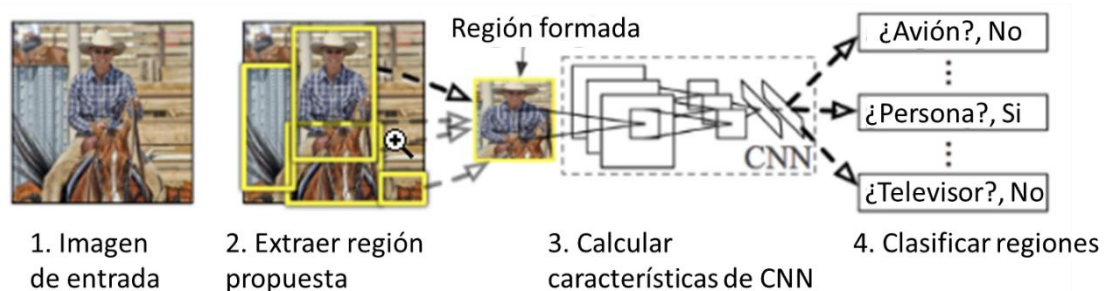
2.2.2.2. R-CNN

Seleccionar varias imágenes por regiones nos puede ocasionar muchos problemas por esta razón Ross Girshick propuso un método en el que se utiliza la búsqueda selectiva para extraer una cantidad limitada de solo 2000 regiones de una imagen a la cual llamo propuesta de región [19]. Las 2000 regiones de la propuesta se generan a partir de un algoritmo de búsqueda selectiva que se detalla a continuación:

Búsqueda Selectiva:

1. Generar subsegmentación inicial, se generan muchas regiones candidatas.
2. Usar un algoritmo codicioso para combinar de forma recursiva las regiones similares en regiones más grandes.
3. Usar las nuevas regiones generadas para producir las propuestas de regiones candidatas finales.

Figura 6: Regiones con funciones CNN



Fuente: Elaboración propia

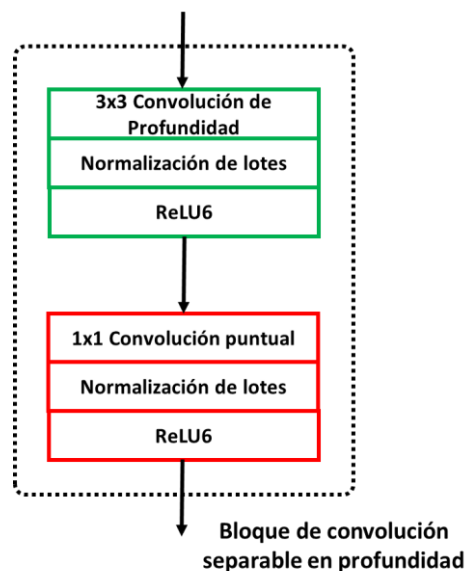
2.2.2.2.1. MobilteNetV1

MobileNetV1 consiste en un tipo de red neuronal convolucional que fue diseñada específicamente para aplicaciones móviles y de visión integrada. Esta se basa en una arquitectura optimizada que usa convoluciones separadas en

profundidad para elaborar redes neuronales profundas y livianas que su latencia es mínima para dispositivos móviles e integrados.

La capa de convolución se divide en dos subtrabajos los cuales son: primero encontramos una capa de convolución en profundidad que se encarga de filtrar en la entrada, seguida de otra capa de convolución de 1x1 que combine los resultados filtrados para generar nuevas características.

Figura 7: MobilteNetV1

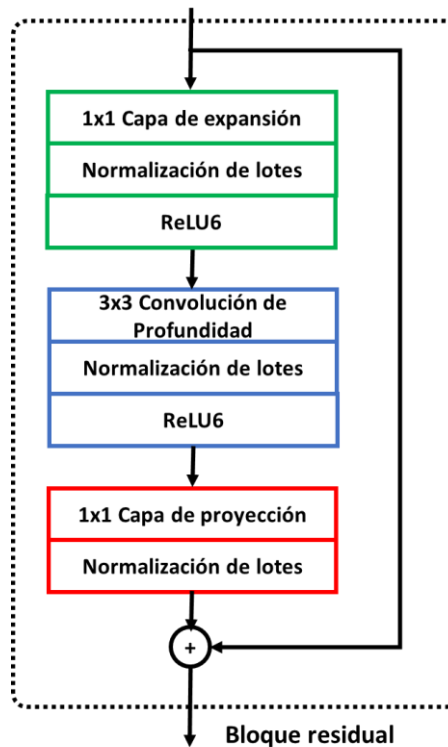


Fuente: [20]

2.2.2.2.2. MobilteNetV2

MobileNetV2 es una mejora de MobileNetV1, este todavía utiliza convoluciones separables en profundidad, pero en cambio su componente usa tres capas convolucionales en el bloque. Los dos últimos bloques son igual que los de su versión anterior, la capa encargada en filtrar las entradas y la capa de convolución puntual. La nueva capa que incorpora es una convolución de 1x1 que tiene como propósito de expandir el número de canales en los datos antes que entren a la convolución de profundidad.

Figura 8: MobileNetV2



Fuente: [20]

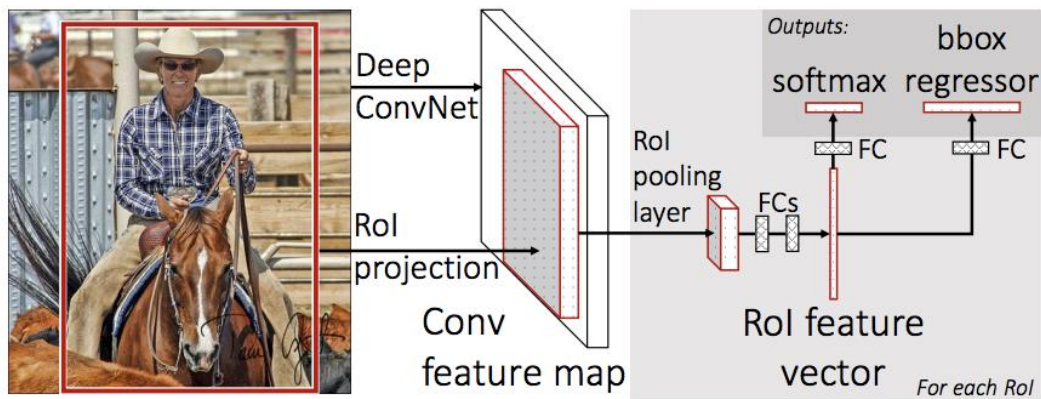
2.2.2.3. Fast R-CNN

El algoritmo Fast R-CNN utiliza un enfoque similar al algoritmo R-CNN, a diferencia del algoritmo anterior que enviaba las propuestas regionales a la CNN este envía la imagen de entrada a la CNN para de esta manera generar un mapa de características convolucional [21].

Una vez obtenido el mapa de características convolucional, se procede a identificar la región de las propuestas y se las deforma en cuadros, mediante el uso de una capa de agrupación de RoI, se las reforma en un tamaño fijo para que se puedan introducir en una capa completamente conectada [22].

La principal razón por la que el algoritmo Fast R-CNN es más rápido que el algoritmo R-CNN es porque no tiene que enviar propuestas de 2000 regiones a la red neuronal convolucional cada vez. En este caso la operación se realiza una vez por imagen y se encarga de generar un mapa de características.

Figura 9: Regiones con funciones Fast R-CNN



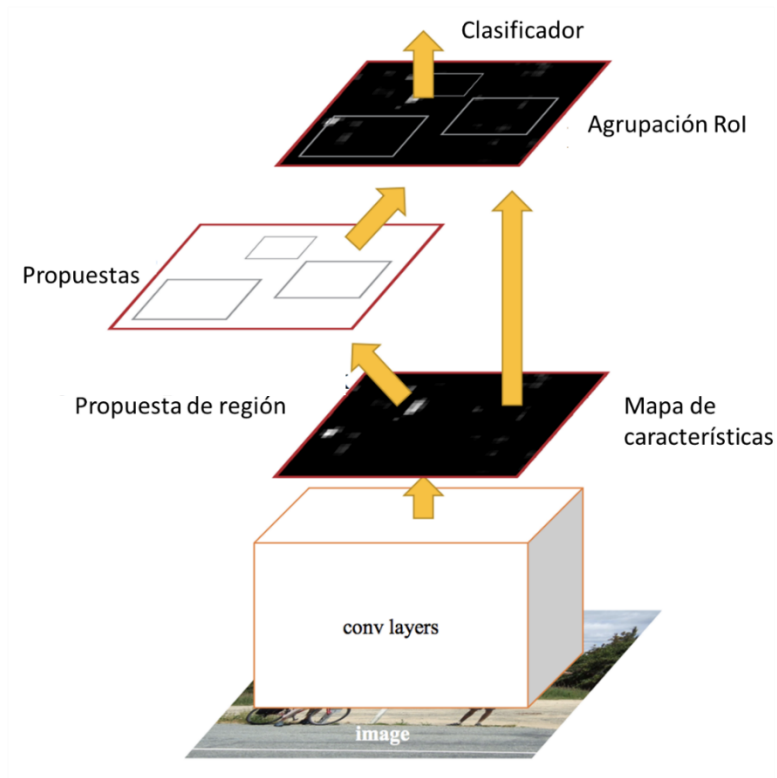
Fuente: [22]

2.2.2.4. Faster R-CNN

Los algoritmos R-CNN y Fast R-CNN utilizan la búsqueda selectiva para encontrar las propuestas de la región. Esta búsqueda selectiva es un proceso lento que afecta al rendimiento de la red. Por esta razón Shaoqing Ren et al creó un nuevo algoritmo para la detección de objetos que elimina el proceso de búsqueda selectiva y permite que la red aprenda las propuestas de la región [23] [24].

De igual manera que en Fast R-CNN se proporciona una imagen como entrada en una red convolucional que nos proporciona un mapa de características convolucionales y en lugar de utilizar el algoritmo de búsqueda selectiva usa una red separada para predecir las propuestas de región. Estas propuestas de región se reforman aplicando una capa de agrupación de RoI que más tarde se utiliza para clasificar la imagen dentro de la región propuesta y predecir los valores de desplazamiento [25].

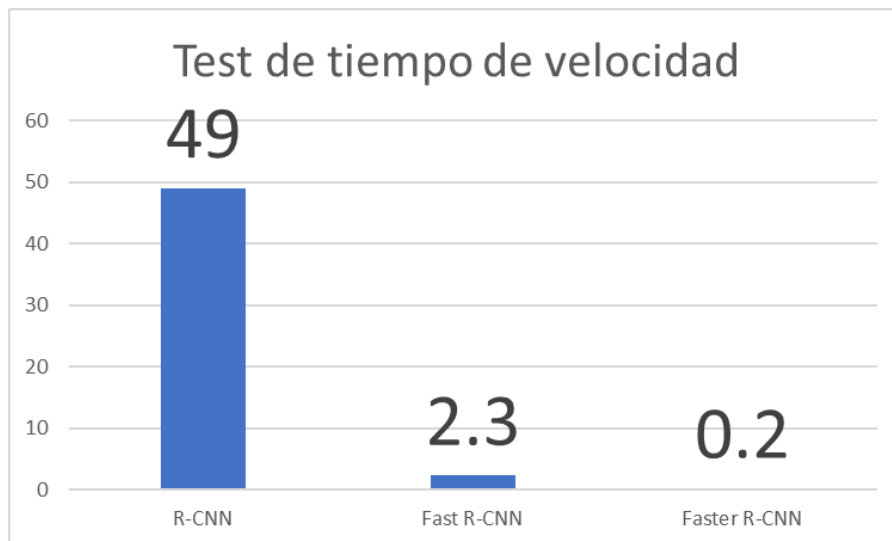
Figura 10: Regiones con funciones Faster R-CNN



Fuente: [24]

2.2.3. Comparación de Algoritmos

Figura 11: Comparación de la velocidad del tiempo de prueba de los algoritmos de detección de objetos



Fuente: [26]

En el grafico se puede observar que el algoritmo de Faster R-CNN es el más rápido que el algoritmo R-CNN y el algoritmo Fast R-CNN, por lo que es preciso para realizar detección de objetos en tiempo real.

2.2.4. Construcción de la Red Neuronal

2.2.4.1. Visual Code Studio

Visual Code Studio es un editor de código y archivos planos desarrollado por Microsoft, es ligero pero muy potente, además funciona en la mayoría de los sistemas operativos como: Windows, MacOS y todas las distribuciones de Linux. Además, incorpora soporte la mayoría de los lenguajes de programación: C++, Java, C#, Python, Go, PHP [27].

2.2.4.2. Tensorflow

Tensorflow es muy conocido por ser una plataforma open source lanzado por Google para el aprendizaje profundo [28]. Posee un ecosistema muy grande en bibliotecas, herramientas y recursos necesarios para impulsar a los investigadores a realizar aprendizajes automáticos innovadores [29].

Las herramientas y librerías que nos brinda Tensorflow nos ayudara a compilar y entrenar modelos con facilidad, nos permite realizar los entrenamientos tanto localmente, como en la nube.

2.3. Metodología

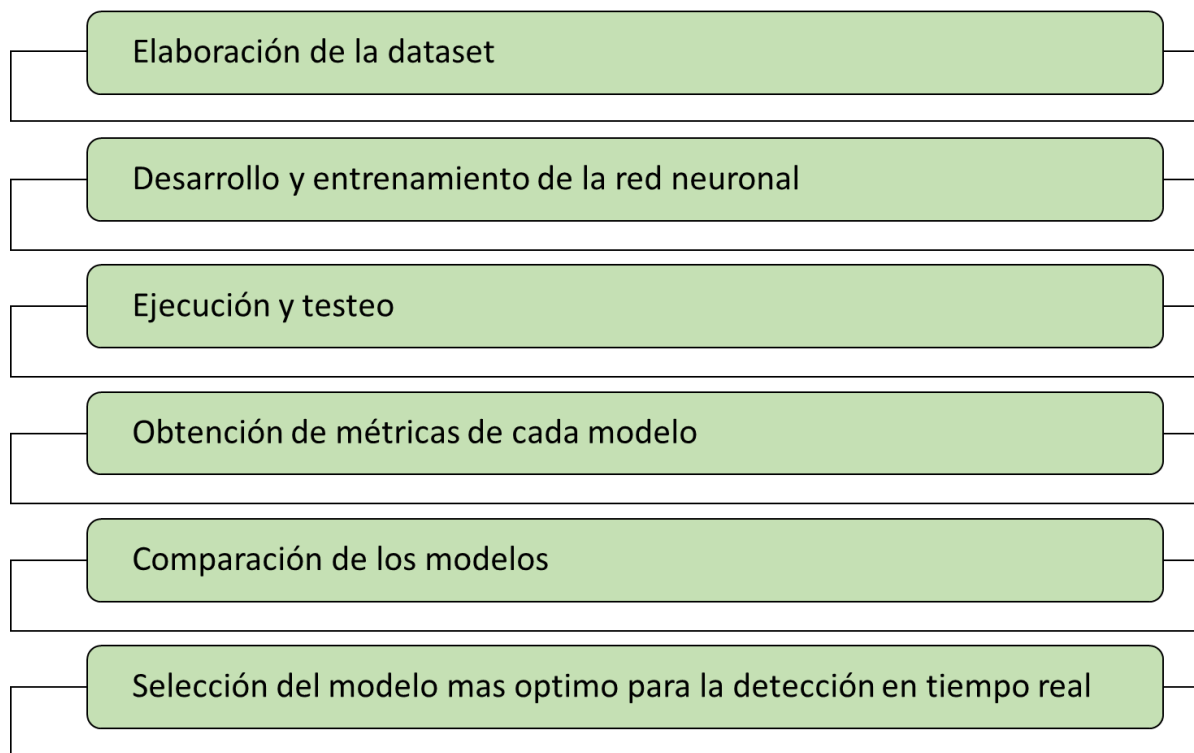
Tras un análisis y estudio exploratorio se identificó varias técnicas de detección de objetos que usan diferentes tipos de herramientas a partir de una base de datos de imágenes. El objetivo principal es entrenar un modelo con diferentes parámetros utilizando técnicas de aprendizaje profundo mediante redes neuronales convolucionales.

Primero se procedió a crear la dataset con imágenes obtenidas de internet y por medio de un smartphone a la cual se le aplico aprendizaje profundo utilizando la herramienta de Python, Keras, Imutils, Numpy y Tensorflow.

Luego se realizó pruebas de ejecución y testeo para obtener métricas y así poder comparar los modelos entrenados.

Finalmente, una vez comparados los modelos se selecciona el modelo más óptimo para utilizarlo en la detección en tiempo real.

Figura 12: Metodología



Fuente: Elaboración propia

2.4. Objetivos del prototipo

2.4.1. Objetivo General

Desarrollar un sistema para el reconocimiento del uso de mascarillas aplicando algoritmos de Deep Learning.

2.4.2. Objetivo Específicos

- Revisar e investigar en revistas científicas los aspectos relacionados con la detección de objetos mediante Deep Learning.
- Construir una dataset para realizar el entrenaamiento de la red neuronal convolucional.
- Entrenar una red neuronal convolucional con la dataset elaborada, para la detección de uso de mascarilla en tiempo real.
- Realizar pruebas de detección en tiempo real de la red neuronal convolucional entrenada.

- Construir una interfaz web amigable para interactuar con la red neuronal convolucional entrenada y realizar la detección en tiempo real.

2.5. Diseño del prototipo

El diseño del prototipo consta de dos partes: la primera consiste en el entrenamiento y prueba de la red neuronal convolucional y la segunda parte de elaborar una interfaz para recibir la entrada y salida de datos que pasaran por la red neuronal.

2.5.1. Requisitos

A continuación, se detalla los recursos a utilizar:

Características del Hardware:

- Marca: Lenovo
- Procesador: Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz 2.90GHz
- Memoria: 12 RAM
- Sistema Operativo: Windows 10 Pro

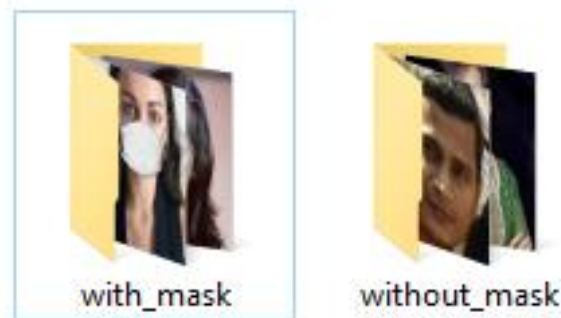
Software:

- Python
- OpenCV
- Numpy
- Flask
- Imutils
- Tensorflow
- Visual Code Studio

2.5.2. Elaboración de la Dataset

Para la elaboración de la dataset se reunió un total de 1500 fotografías las cuales están divididas en dos grupos 750 de imágenes con rostros de personas (fotografías obtenidas mediante un smartphone e internet) y 750 imágenes con rostros de personas utilizando mascarilla (fotografías obtenidas mediante un smartphone e internet).

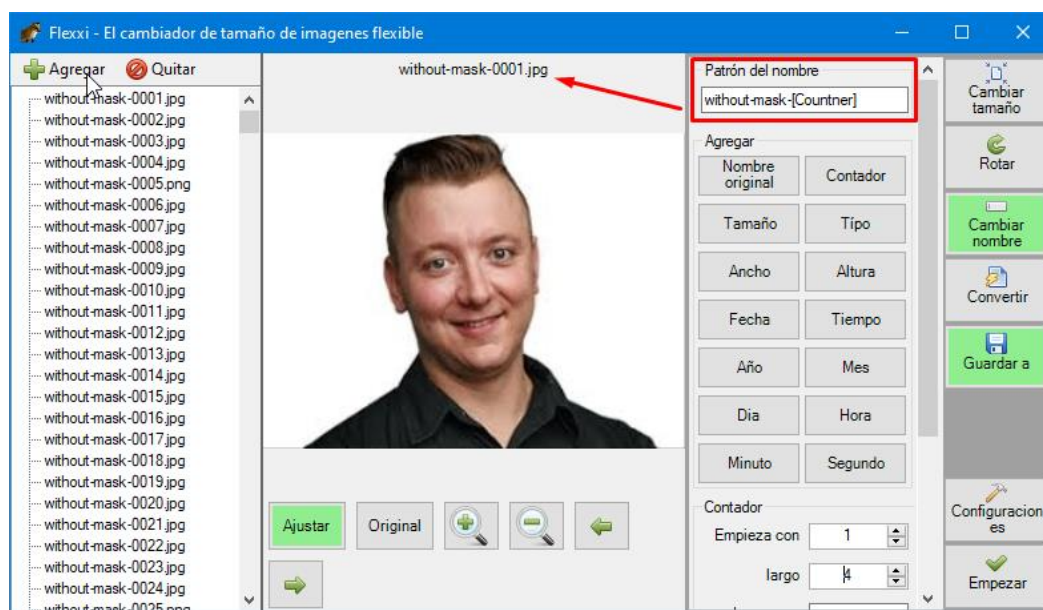
Figura 13: Estructura de dataset



Fuente: Elaboración propia

Una vez reunidas las fotografías se procedió a renombrarlas con la herramienta libre Flexxi de la siguiente manera para las imágenes que no están utilizando mascarilla without-mask-0001 y para las que están con mascarilla with-mask-0001.

Figura 14: Renombrado de imágenes con Flexxi



Fuente: Elaboración propia

2.5.3. Diseño y entrenamiento de la red

Para el diseño de la red se utilizó las funciones de MobileNetV2 y se desarrolló el código en el editor de Visual Code Studio.

Figura 15: Diseño de Red Neuronal

```

27 print(os.getcwd())
28 # constructor de argumentos
29 dataset_path=os.getcwd()+"//dataset"
30 model_path=os.getcwd()+"//model//mask_model"
31 plot_path=os.getcwd()+"//plot"
32
33
34 # inicializar la tasa de aprendizaje inicial, el número de épocas para
35 # entrenar y el tamaño del lote
36 #INIT_LR = 1e-4 #10000
37 INIT_LR = 1e-1
38 #EPOCHS = 20
39 EPOCHS = 5
40 #BS = 32
41 BS = 32
42
43 # cargamos nuestro directorio de conjunto de datos, luego inicialice
44 # la lista de datos (es decir, imágenes) e imágenes de clase
45 print("[INFO] loading images...")
46 imagePath = list(paths.list_images(dataset_path))
47 imagePath = [imagePath.replace("\\", "/", -1) for imagePath in imagePath]
48 data = []
49 labels = []
50
51 # recorrer las rutas de la imagen
52 for imagePath in imagePath:
53     # extraer la etiqueta de clase del nombre del archivo
54     label = imagePath.split("/")[-2]
55
56     # cargar la imagen de entrada (224x224) y preprocesarla
57     image = load_img(imagePath, target_size=(224, 224))
58     image = img_to_array(image)
59     image = preprocess_input(image)

```

Fuente: Elaboración propia

Una vez diseñada la red neuronal se procedió a realizar el entrenamiento, para ello abrimos una terminal y escribimos el comando:

`python train_mask_detector.py --dataset dataset`

Figura 16: Entrenamiento de la red neuronal

```

2020-12-02 13:22:36.445105: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this ma
y take a while): 26 of 38
2020-12-02 13:22:40.681298: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
37/37 [=====] - 249s 7s/step - loss: 0.0729 - accuracy: 0.9752 - val_loss: 0.0431 - val_accurac
y: 0.9896
Epoch 19/20
2020-12-02 13:26:45.668445: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this ma
y take a while): 26 of 38
2020-12-02 13:26:50.000911: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
37/37 [=====] - 247s 7s/step - loss: 0.0619 - accuracy: 0.9760 - val_loss: 0.0324 - val_accurac
y: 0.9931
Epoch 20/20
2020-12-02 13:30:52.148144: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this ma
y take a while): 23 of 38
2020-12-02 13:30:57.930359: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
37/37 [=====] - 249s 7s/step - loss: 0.0515 - accuracy: 0.9846 - val_loss: 0.0462 - val_accurac
y: 0.9896
[INFO] evaluating network...
      precision    recall  f1-score   support

with_mask         0.98         1.00         0.99         150
without_mask      1.00         0.98         0.99         150

   accuracy          0.99
  macro avg          0.99
weighted avg          0.99
[INFO] saving mask detector model... path: C:\wamp64\www\titulacion\python\model\mask_model.h5
(base) C:\wamp64\www\titulacion\python>

```

Fuente: Elaboración propia

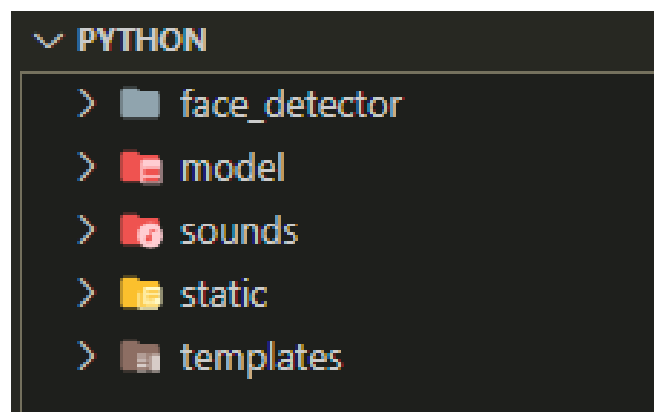
Este proceso se lo realizo 4 veces con diferentes parámetros para luego realizar pruebas.

2.5.4. Algoritmo de Reconocimiento

Ahora procedemos a desarrollar el código que cargara nuestro modelo entrenado y realizara la detección en tiempo real, para igual que los pasos anteriores utilizamos Visual Code Studio para programar en python.

Primero creamos la carpeta con la estructura del proyecto que es la siguiente:

Figura 17: Estructura de la interfaz web

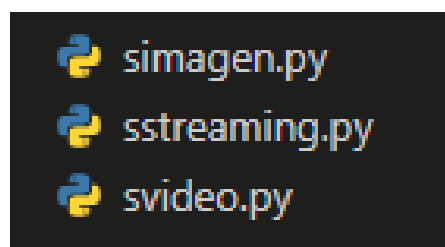


Fuente: Elaboración propia

Creamos una carpeta llamada python y dentro creamos tres carpetas: model que tendrá el modelo entrenado, sounds que tendrá archivos de sonidos, static tendrá los estilos css, imágenes y js, templates tendrá las plantillas html.

Por último, creamos 3 archivos py con los siguientes nombres: sstreaming.py (detección en tiempo real - streaming), svideo.py (detección en tiempo real - video) y simagen.py (detección en tiempo real - imagen).

Figura 18: Archivos que realizaran la detección en tiempo real

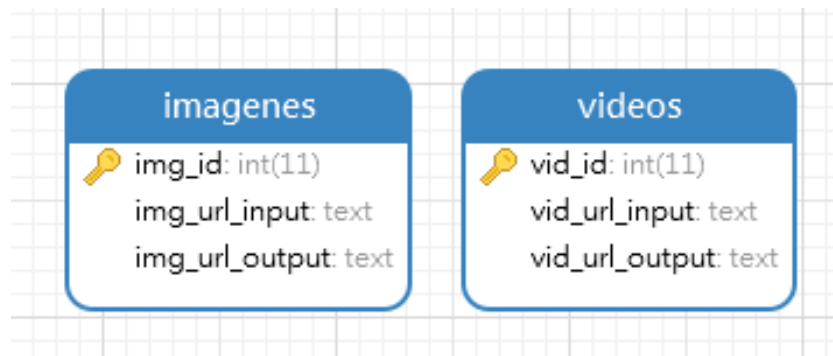


Fuente: Elaboración propia

2.5.5. Desarrollo de la interfaz web

Aprovechando las ventajas que nos ofrece python vamos a usar el microframework Flask para desarrollar las vistas web y un poco de php, también necesitaremos una pequeña base de datos en mysql con dos tablas con tres campos en la cual guardamos el id de la imagen de entrada o video, el nombre de la imagen de entrada o video y el nombre de la imagen de salida.

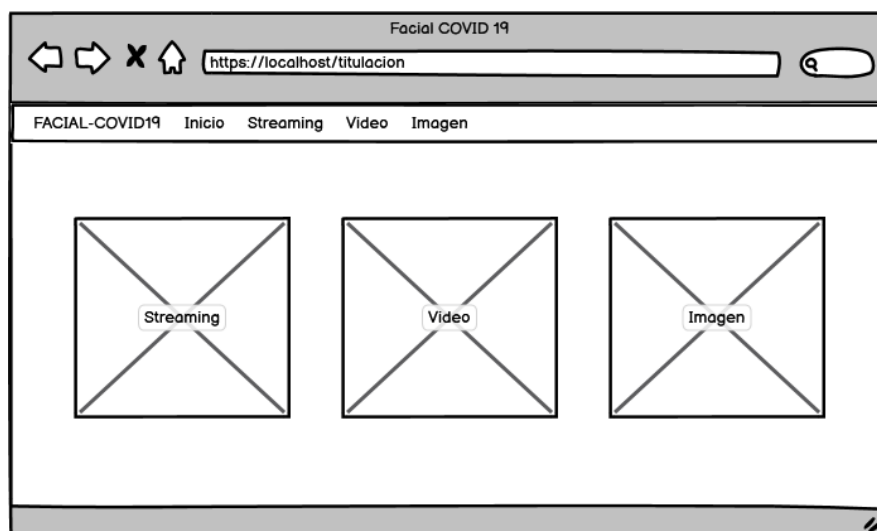
Figura 19: Estructura de la base de datos



Fuente: Elaboración propia

Antes de desarrollar las interfaces maquetamos las vistas en alguna herramienta que nos permita hacer, en este caso se utilizó Balsamiq.

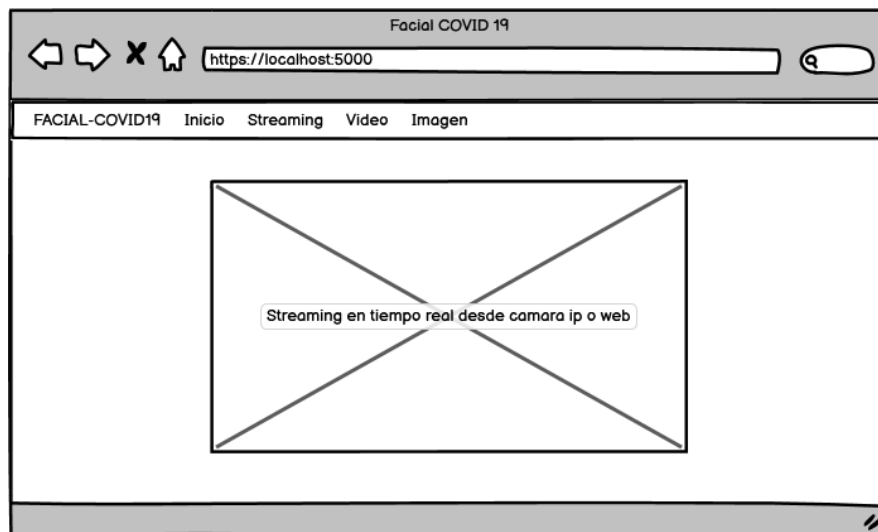
Figura 20: Vista Inicio



Fuente: Elaboración propia

En la vista inicio está formada por una barra de menú y por tres imágenes que funcionan como acceso directo para las vistas de streaming, video e imagen.

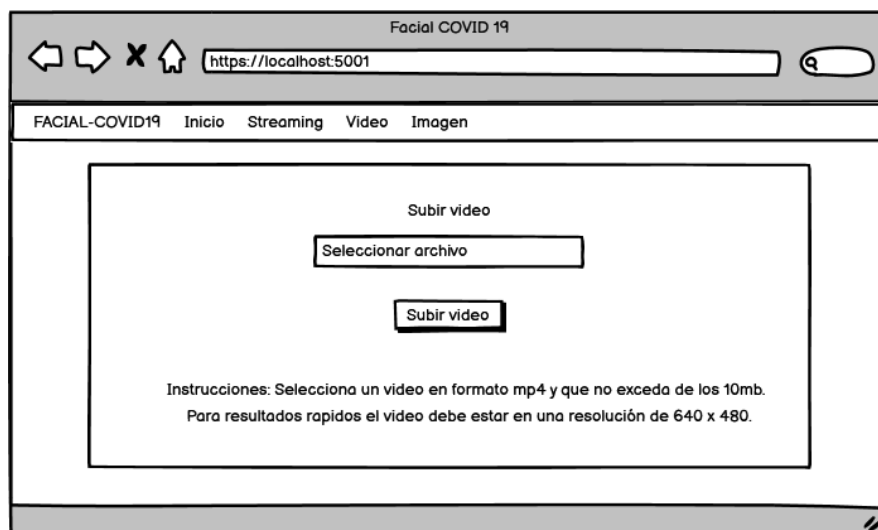
Figura 21: Vista Streaming



Fuente: Elaboración propia

La vista streaming como nos indica el nombre nos permite observar en tiempo real lo que captura por cámara web o cámara ip, que luego es pasado por el modelo entrenado y nos mostrar un cuadrado de color verde sobre el rostro de la persona que este usando mascarilla, en casa de que no tenga mascarilla mostrará un rectángulo de color rojo y se reproducirá un sonido de alerta.

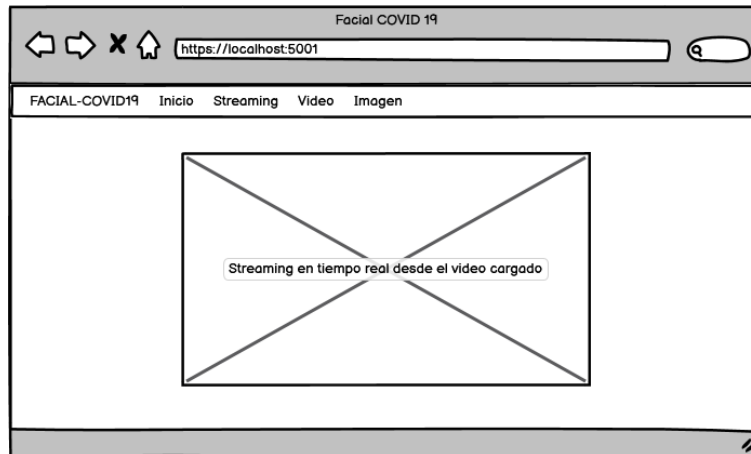
Figura 22: Vista Subir video



Fuente: Elaboración propia

La vista subir video nos muestra un input file en el cual damos clic y nos mostrara una ventana para buscar un video debemos seguir las instrucciones que nos muestra al final de la pantalla, una vez seleccionado el video damos clic en subir video el cual lo subirá a un servidor local y en la base de datos se guardara el nombre del video.

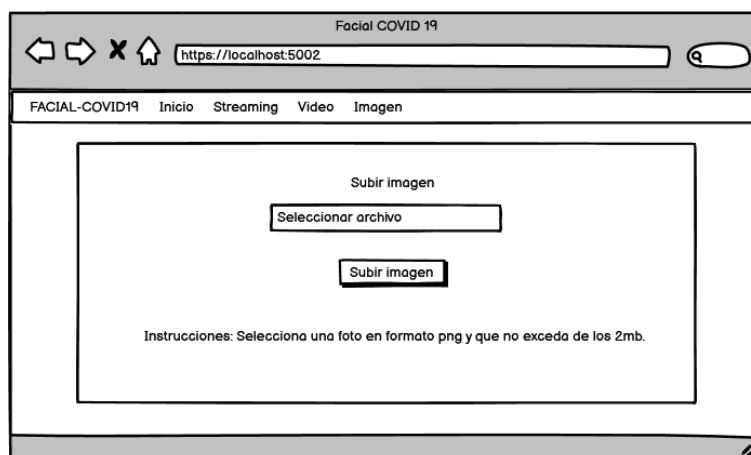
Figura 23: Vista Video



Fuente: Elaboración propia

En la vista video se observa el resultado del video que pasa a través de la red neuronal entrenado el cual nos muestra el ultimo video de la base de datos procesado en tiempo real con un cuadrado verde si la o las personas están utilizando mascarilla y un cuadro de color rojo si la o las personas no están utilizando mascarillas.

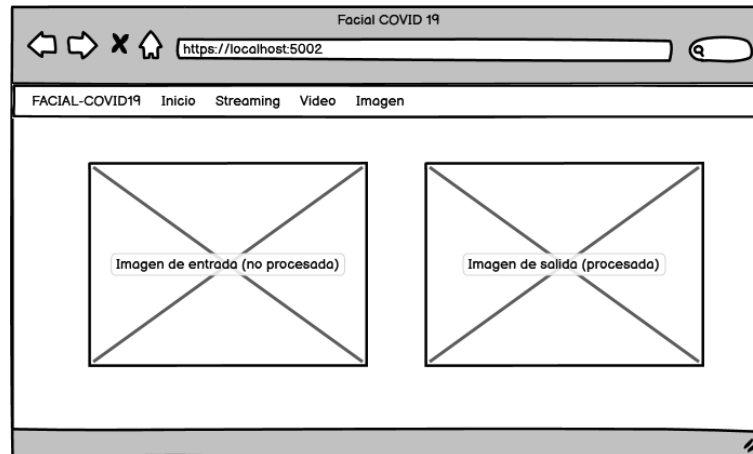
Figura 24: Vista Subir imagen



Fuente: Elaboración propia

La vista subir imagen nos muestra un input file en el cual damos clic y nos mostrara una ventana para buscar una imagen debemos seguir las instrucciones que nos muestra al final de la pantalla, una vez seleccionada la imagen damos clic en subir imagen el cual lo subirá a un servidor local y en la base de datos se guardara el nombre de la imagen.

Figura 25: Vista Imagen



Fuente: Elaboración propia

La vista imagen nos muestra dos cuadrados, un cuadrado es la imagen de entrada (imagen sin procesar) y el siguiente cuadro es la imagen de salida (imagen procesada) que es la última imagen que se encuentra en la base de datos, de igual forma que en las vistas anteriores nos mostrara un cuadro de color verde si la o las personas están utilizando mascarilla y un cuadro de color rojo si la o las personas no están utilizando mascarilla.

2.6. Ejecución y/o ensamblaje del prototipo

Para poner en marcha el proyecto ejecutamos los tres archivos creados en python: sstreaming.py, svideo.py y simagen.py desde la terminal con el comando python seguido del nombre del archivo py.

Figura 26: Ejecución de sstreaming.py

```
Anaconda Prompt (Anaconda3) - python sstreaming.py

(base) C:\wamp64\www\titulacion\python>python sstreaming.py
[INFO] loading face detector model...
[INFO] loading face mask detector model...
2020-12-03 00:33:15.001238: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2
* Serving Flask app "sstreaming" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
[INFO] loading face detector model...
[INFO] loading face mask detector model...
2020-12-03 00:33:28.120575: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2
* Debugger is active!
* Debugger PIN: 261-622-948
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
```

Fuente: Elaboración propia

Figura 27: Ejecución svideo.py

```
Anaconda Prompt (Anaconda3) - python svideo.py

(base) C:\wamp64\www\titulacion\python>python svideo.py
[INFO] loading face detector model...
[INFO] loading face mask detector model...
2020-12-03 00:35:40.428391: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2
* Serving Flask app "svideo" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
[INFO] loading face detector model...
[INFO] loading face mask detector model...
2020-12-03 00:35:54.135254: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2
* Debugger is active!
* Debugger PIN: 261-622-948
* Running on http://0.0.0.0:5002/ (Press CTRL+C to quit)
```

Fuente: Elaboración propia

Figura 28: Ejecución simagen.py

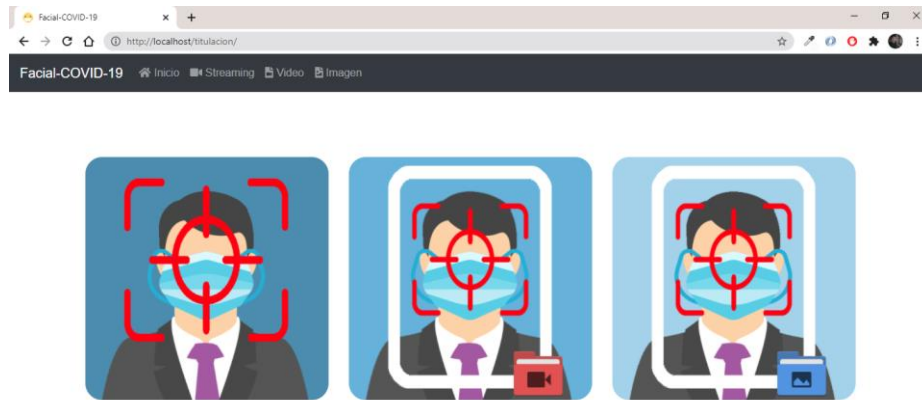
```
Anaconda Prompt (Anaconda3) - python simagen.py

(base) C:\wamp64\www\titulacion\python>python simagen.py
[INFO] loading face detector model...
[INFO] loading face mask detector model...
2020-12-03 00:39:20.211209: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2
* Serving Flask app "simagen" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
[INFO] loading face detector model...
[INFO] loading face mask detector model...
2020-12-03 00:39:34.418779: I tensorflow/core/platform/cpu_feature_guard.cc:142] Your CPU
supports instructions that this TensorFlow binary was not compiled to use: AVX2
* Debugger is active!
* Debugger PIN: 261-622-948
* Running on http://0.0.0.0:5001/ (Press CTRL+C to quit)
```

Fuente: Elaboración propia

Una vez ejecutado los tres archivos py abrimos el navegador web en este caso Google Chrome y en la url escribimos localhost/titulacion y cargara la vista de inicio.

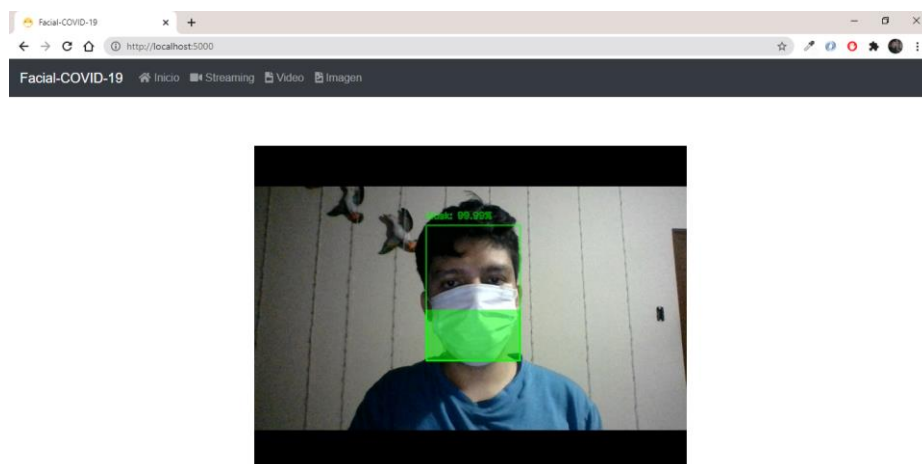
Figura 29: Vista inicia visualizada desde el navegador web



Fuente: Elaboración propia

Ahora damos clic en la primera imagen o en el menú superior en la opción Streaming y empezara a realizar la detección de mascarilla.

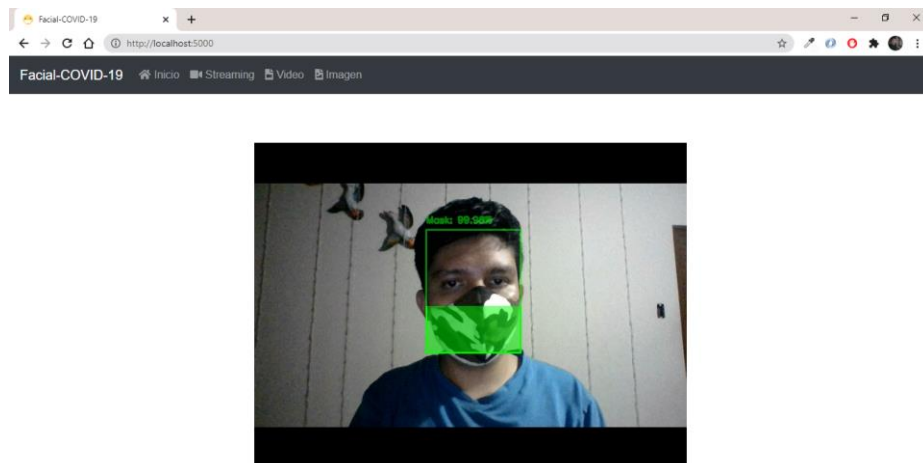
Figura 30: Detección en tiempo real, mascarilla A



Fuente: Elaboración propia:

Se utilizo un tipo de mascarilla a la cual la nombre como Mascarilla A, la detección es exitosa.

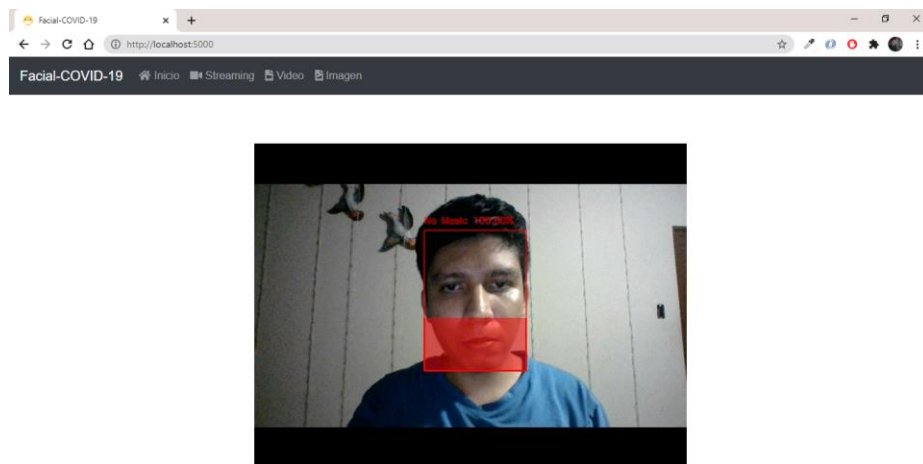
Figura 31: Detección en tiempo real, mascarilla B



Fuente: Elaboración propia

Se utilizo una mascarilla diferente a la del tipo A, la detección es exitosa.

Figura 32: Detección en tiempo real, sin mascarilla

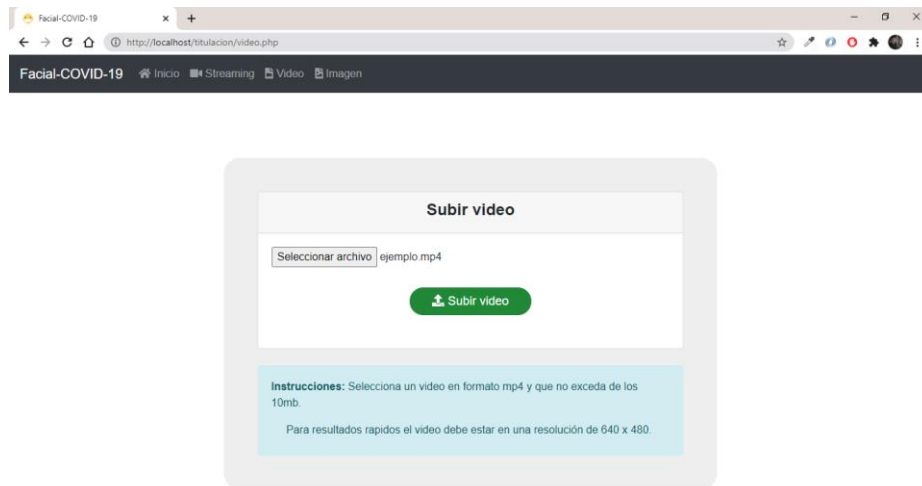


Fuente: Elaboración propia

Aquí no se utilizó ninguna mascarilla y el resultado es tal como se lo explico anteriormente, nos muestra un recuadro de color rojo y un sonido de alarma.

En la vista video nos muestra la opción para subir un video y se lo realiza de acuerdo a las instrucciones que muestra la página.

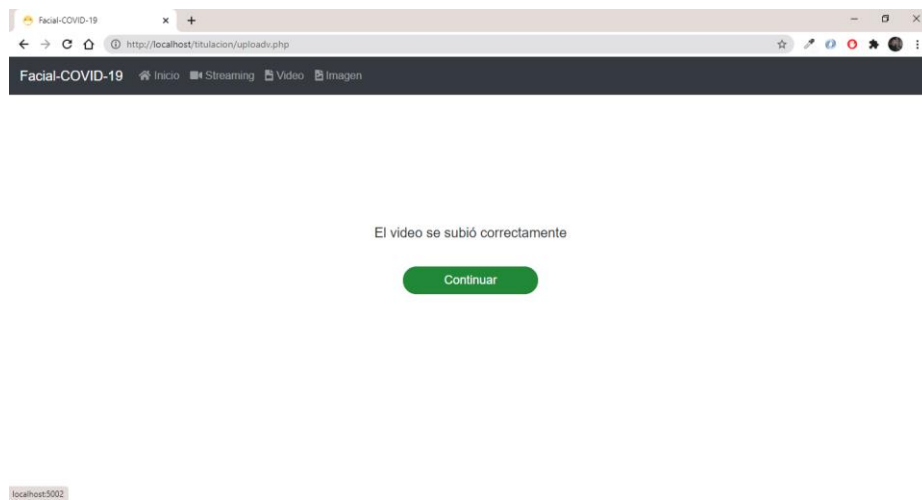
Figura 33: Selección de video para detección en tiempo real



Fuente: Elaboración propia

Si se realizó de acuerdo a las instrucciones nos aparece que el video se subió correctamente y seguido damos clic en continuar.

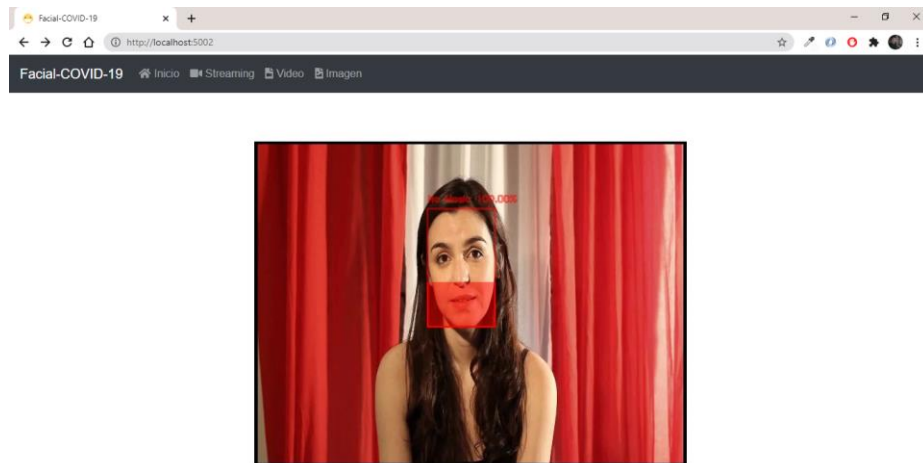
Figura 34: Subida correcta del video al servidor



Fuente: Elaboración propia

Se muestra en el navegador el video procesado en tiempo real que se subió al servidor.

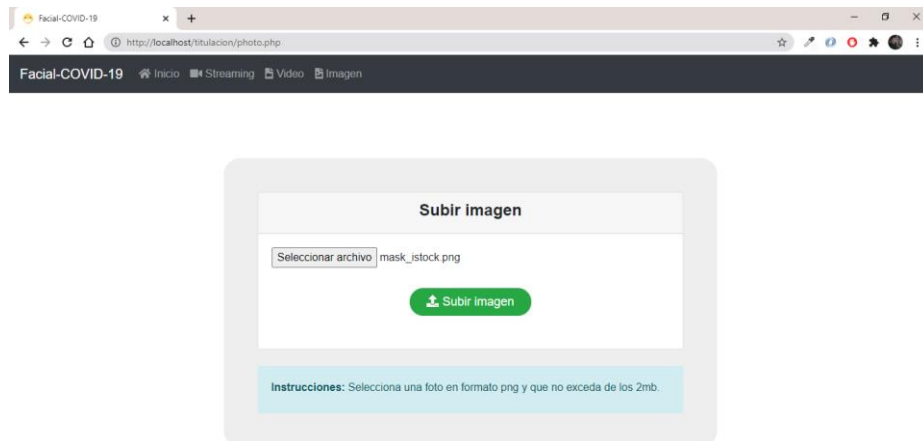
Figura 35: Detección en tiempo real de video subido al servidor



Fuente: Elaboración propia

En la vista imagen nos muestra la opción para subir una imagen y se lo realiza de acuerdo a las instrucciones que muestra la página.

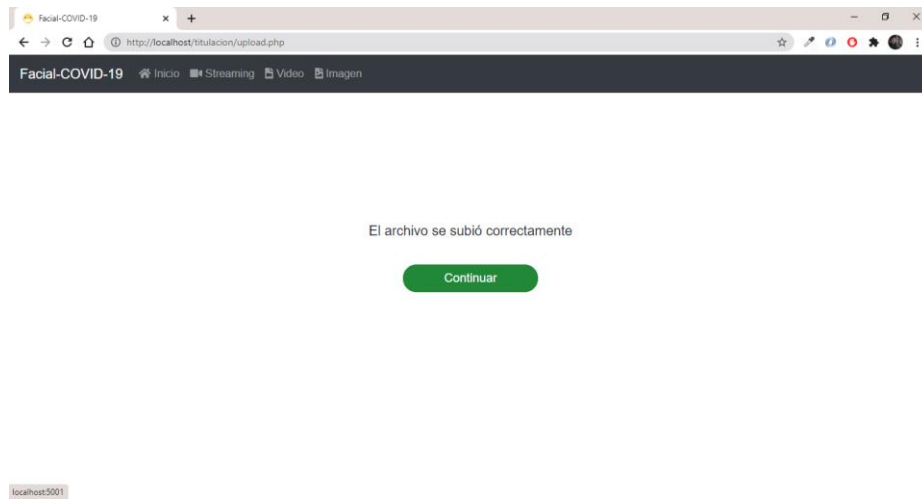
Figura 36: Selección de imagen



Fuente: Elaboración propia

Si se realizó de acuerdo a las instrucciones nos aparece que la imagen se subió correctamente y seguido damos clic en continuar.

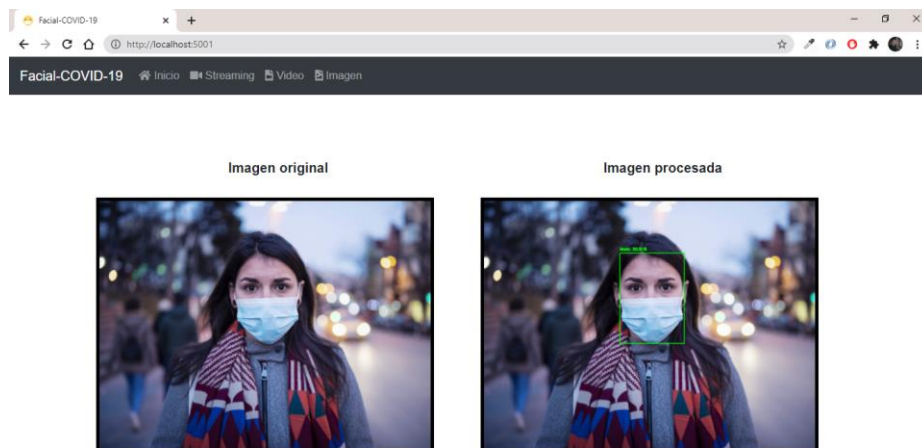
Figura 37: Subida correcta de la imagen al servidor



Fuente: Elaboración propia

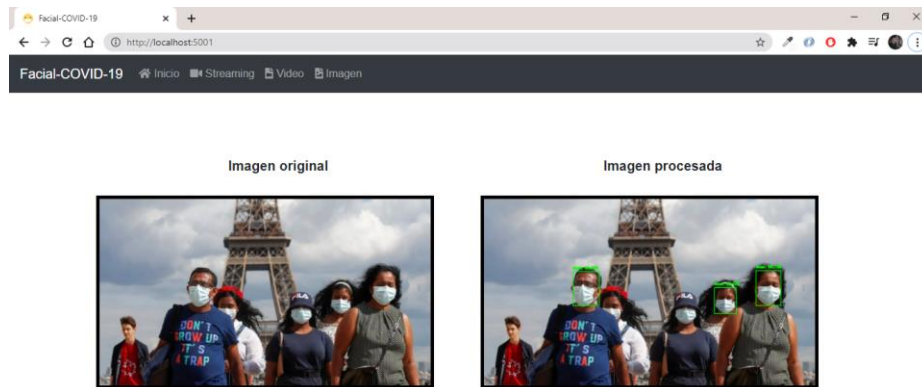
Se muestra en el navegador la imagen de entrada y la imagen de salida procesada en tiempo real.

Figura 38: Detección en tiempo real de imagen subida al servidor



Fuente: Elaboración propia

Figura 39: Detección en tiempo real de imagen subida al servidor



Fuente: Elaboración propia

En la prueba realizada en esta imagen se puede observar que de seis personas solamente se logró realizar la detección a tres personas, esto se debe a que la persona número 1 está muy lejos y no es posible realizar la detección, por otro lado la persona número 3 está siendo obstaculizada por la persona número 2 y tampoco es posible realizar la detección, mientras tanto la persona número 4 tampoco es posible realizar la detección porque está utilizando un sombrero grande y oculta todo su rostro impidiendo que el algoritmo realice la detección.

3. EVALUACIÓN DEL PROTOTIPO

3.1. Plan de evaluación

El plan para evaluar los resultados de la red neuronal convolucional se aplicaron métricas de clasificación y métricas de detección.

3.1.1. Prueba de Entrenamiento y Validación de la Red Neuronal

3.1.1.1. Métricas

El resultado de la salida de la clasificación de una red neuronal es 0 y 1. Los resultados son Verdadero-Positivo es decir la cantidad de clases detectadas de forma correcta, Falso-Negativo es decir total de objetos que la red neuronal convolucional no ha detectado, Falso-Positivo es decir la cantidad total de objetos detectados que no pertenecen a los objetos entrenados en la red y Verdadero-Negativo es decir cuando en el cuadro delimitador no aparece el objeto entrenado y la red lo identifica como verdadero [30].

Tabla 1: Matriz de confusión

		Predicción de la red neuronal	
		Objeto presente	Objeto no presente
Situación real	Objeto presente	VP	FN
	Objeto no presente	FP	VN

Fuente: [31]

Precisión (Precisión)

Esta métrica nos muestra que elementos seleccionados dentro de los cuadros delimitadores son exactos o relevantes en la predicción, para realizar el cálculo de este parámetro se lo realiza dividiendo el número de predicciones VP entre la suma de predicciones VP y las predicciones FP [31].

$$Presicion = \frac{VP}{VP + FP}$$

Sensibilidad (Recall)

Esta métrica visualiza la cantidad de objetos que son seleccionados en los cuadros delimitadores, para realizar el cálculo de este parámetro se lo realiza dividiendo el número de predicciones VP entre la suma de las predicciones VP y las predicciones FN [31].

$$Recall = \frac{VP}{VP + FN}$$

Valor de Referencia (F-Score)

Esta métrica nos da un valor de referencia de que tan bien funciona el sistema, para realizar el cálculo de este parámetro se lo realiza dividiendo la multiplicación de la precisión por la sensibilidad entre la suma de dichos valores [31].

$$F - Score = 2 \times \frac{Presicion \times Recall}{Presicion + Recall}$$

3.2. Resultado de evaluación

3.1.2. Resultado de la Prueba de Entrenamiento y Validación

Resultado de la prueba 1

Tabla 2: Parámetros de entrenamiento prueba 1

Parámetros	Valor
Conjunto de datos	Propio
Imágenes de entrenamiento	1500
Taza de aprendizaje	0.1
Numero de épocas	5
Tamaño de lote	32

Fuente: Elaboración propia

Parámetros utilizados para el entrenamiento de la prueba 1.

Tabla 3: Parámetros de entrenamiento prueba 2

Parámetros	Valor
Conjunto de datos	Propio
Imágenes de entrenamiento	1500
Taza de aprendizaje	0.01
Numero de épocas	10
Tamaño de lote	32

Fuente: Elaboración propia

Parámetros utilizados para el entrenamiento de la prueba 2.

Tabla 4: Parámetros de entrenamiento prueba 3

Parámetros	Valor
Conjunto de datos	Propio
Imágenes de entrenamiento	1500
Taza de aprendizaje	0.001
Numero de épocas	15
Tamaño de lote	32

Fuente: Elaboración propia

Parámetros utilizados para el entrenamiento de la prueba 3.

Tabla 5: Parámetros de entrenamiento prueba 4

Parámetros	Valor
Conjunto de datos	Propio
Imágenes de entrenamiento	1500
Taza de aprendizaje	0.0001
Numero de épocas	20
Tamaño de lote	32

Fuente: Elaboración propia

Parámetros utilizados para el entrenamiento de la prueba 4.

Tabla 6: Resultado de entrenamiento prueba 1

Métricas	Con mascarilla	Sin mascarilla
Precisión (Precision)	0.50	0.00
Sensibilidad (Recall)	1.00	0.00
Valor de referencia (F-Score)	0.67	0.00

Fuente: Elaboración propia

El resultado de presión con mascarilla muestra un 0.50 que no es óptimo y sin mascarilla un 0.00 que tampoco es óptimo, mientras que en la sensibilidad con mascarilla muestra un 1.00 que es óptimo y sin mascarilla 0.00 que no es óptimo, mientras tanto en el valor de referencia con mascarilla muestra un 0.67 que es aceptable y sin mascarilla un 0.00 que no es óptimo.

Tabla 7: Resultado de entrenamiento prueba 2

Métricas	Con mascarilla	Sin mascarilla
Precisión (Precision)	0.98	1.00
Sensibilidad (Recall)	1.00	0.98
Valor de referencia (F-Score)	0.99	0.99

Fuente: Elaboración propia

El resultado de presión con mascarilla muestra un 0.98 que es aceptable y sin mascarilla un 1.00 que es óptimo, mientras que en la sensibilidad con mascarilla muestra un 1.00 que es óptimo y sin mascarilla 0.98 que es aceptable, mientras tanto en el valor de referencia con mascarilla muestra un 0.99 que es aceptable y sin mascarilla un 0.99 que es aceptable.

Tabla 8: Resultado de entrenamiento prueba 3

Métricas	Con mascarilla	Sin mascarilla
Precisión (Precision)	1.00	0.99
Sensibilidad (Recall)	0.99	1.00
Valor de referencia (F-Score)	1.00	1.00

Fuente: Elaboración propia

El resultado de presión con mascarilla muestra un 1.00 que es óptimo y sin mascarilla un 0.99 que es óptimo, mientras que en la sensibilidad con mascarilla muestra un 1.00 que es óptimo y sin mascarilla 1.00 que es óptimo, mientras

tanto en el valor de referencia con mascarilla muestra un 1.00 que es óptimo y sin mascarilla un 1.00 que es óptimo.

Tabla 9: Resultado de entrenamiento prueba 4

Métricas	Con mascarilla	Sin mascarilla
Precisión (Precision)	0.98	1.00
Sensibilidad (Recall)	1.00	0.98
Valor de referencia (F-Score)	0.99	0.99

Fuente: Elaboración propia

El resultado de presión con mascarilla muestra un 0.98 que es aceptable y sin mascarilla un 1.00 que es óptimo, mientras que en la sensibilidad con mascarilla muestra un 1.00 que es óptimo y sin mascarilla 0.98 que es aceptable, mientras tanto en el valor de referencia con mascarilla muestra un 0.99 que es aceptable y sin mascarilla un 0.99 que es aceptable.

Con los datos obtenidos se realizó una matriz para seleccionar el modelo entrenado más óptimo para realizar la detección.

Tabla 10: Tabla de comparación de modelos entrenados

	Precisión		Sensibilidad		Valor de referencia	
	CM	SN	CM	SM	CM	SM
Prueba 1	N	N	O	N	A	N
Prueba 2	A	O	O	A	A	A
Prueba 3	O	A	A	O	O	O
Prueba 4	A	O	O	A	A	A

Fuente: Elaboración propia

Se ha seleccionado el modelo entrenado más resultados óptimos, en este caso es el modelo de la prueba 3.

3.2. Conclusiones

Como resultado del sistema de detección de mascarillas en tiempo aplicando Deep Learning se concluye que:

- Mediante la revisión de revistas y artículos científicos se adquirió el conocimiento sobre redes neuronales convolucionales.
- Se logro construir una dataset con una gran cantidad de fotografías para realizar el entrenamiento de la red neuronal convolucional.
- Aplicando las técnicas investigadas se logró realizar el entrenamiento de cuatro redes neuronales convoluciones.
- Se logro realizar las pruebas de detección en tiempo real con la red neuronal convolucional entrenada.
- La interfaz del sistema se implementó de manera local y se demostró su funcionamiento en tiempo real.

3.3. Recomendaciones

Para la implementación del sistema de detección de mascarillas en tiempo real aplicando Deep Learning se recomienda:

- Realizar una exploración más profunda en las fuentes bibliografías para enriquecer más el conocimiento adquirido.
- Para realizar pruebas más rápidas se recomienda implementar el sistema en un hardware con gran capacidad computacional e incluso una tarjeta gráfica de gama alta.
- Las fotografías seleccionadas para la dataset deben cumplir algunos parámetros como: nitidez, tamaño, entre otros para que cuando se realice el entrenamiento de la red neuronal sea clara.
- Entrenar una red neuronal requiere de gran capacidad computacional, entre más fotografías tenga la dataset más pesado es el trabajo de entrenar.
- Es necesario realizar varias pruebas con diferentes tipos de mascarillas para poder observar aspectos a mejorar a futuro.

4. BIBLIOGRAFÍA

- [1] R. Barouki *et al.*, «The COVID-19 pandemic and global environmental change: Emerging research needs», *Environ. Int.*, vol. 146, p. 106272, ene. 2021, doi: 10.1016/j.envint.2020.106272.
- [2] A. M. Ríos, «Número de casos confirmados y muertes causadas por el coronavirus (COVID-19) en Ecuador entre el 1 de marzo y el 18 de noviembre de 2020». <https://es.statista.com/estadisticas/1110063/numero-casos-muertes-covid-19-ecuador>.
- [3] S.-Y. Lu, S.-H. Wang, y Y.-D. Zhang, «A classification method for brain MRI via MobileNet and feedforward network with random weights», *Pattern Recognit. Lett.*, vol. 140, pp. 252-260, dic. 2020, doi: 10.1016/j.patrec.2020.10.017.
- [4] C. Peng, K. Zhao, y B. C. Lovell, «Faster ILOD: Incremental learning for object detectors based on faster RCNN», *Pattern Recognit. Lett.*, vol. 140, pp. 109-115, dic. 2020, doi: 10.1016/j.patrec.2020.09.030.
- [5] V. A. Kuiava, E. L. Kuiava, R. Rodriguez, A. E. Beck, J. P. M. Rodriguez, y E. O. Chielle, «Method of histopathological diagnosis of mammary nodules through deep learning algorithm», *J. Bras. Patol. E Med. Lab.*, vol. 55, n.º 6, 2019, doi: 10.5935/1676-2444.20190055.
- [6] Q. Fang *et al.*, «Detecting non-hardhat-use by a deep learning method from far-field surveillance videos», *Autom. Constr.*, vol. 85, pp. 1-9, ene. 2018, doi: 10.1016/j.autcon.2017.09.018.
- [7] M. Q. Martínez, G. V. Landivar, J. P. Villamar, y L. A. D. Rosario, «DETECCIÓN DE PERSONAL NO AUTORIZADO EN EL DEPARTAMENTO DE TI UTILIZANDO REDES NEURONALES CONVOLUCIONALES EN TIEMPO REAL CON RASPBERRY Pi 3 B+», *J. Sci. Res. Rev. Cienc. E Investig.*, vol. 5, pp. 49-60, jul. 2020.
- [8] S. Sivkov *et al.*, «The algorithm development for operation of a computer vision system via the OpenCV library», *Procedia Comput. Sci.*, vol. 169, pp. 662-667, 2020, doi: 10.1016/j.procs.2020.02.193.

- [9] K. Pulli, A. Baksheev, K. Korniyakov, y V. Eruhimov, «Realtime Computer Vision with OpenCV», 2012, vol. 10, n.º 4, p. 17, doi: 10.1145/2181796.2206309.
- [10] A. K. Jaiswal, P. Tiwari, S. Kumar, D. Gupta, A. Khanna, y J. J. P. C. Rodrigues, «Identifying pneumonia in chest X-rays: A deep learning approach», *Measurement*, vol. 145, pp. 511-518, oct. 2019, doi: 10.1016/j.measurement.2019.05.076.
- [11] Atix Libre, *Flask y Python 3*, 24.^a ed., vol. 1. 2018.
- [12] M. Fulgueira-Camilo, E. Insúa-Suárez, y H. Díaz-Pando, «Implementación del Algoritmo Trace Alignment Empleando Técnicas de Programación Paralela», *Lámpsakos*, n.º 15, pp. 11-21, 2016.
- [13] NVIDIA, «DEEP LEARNING», *DEEP LEARNING*. <https://developer.nvidia.com/deep-learning>.
- [14] A. Lopez-Fernandez, D. Rodriguez-Baena, F. Gomez-Vela, F. Divina, y M. Garcia-Torres, «A multi-GPU biclustering algorithm for binary datasets», *J. Parallel Distrib. Comput.*, vol. 147, pp. 209-219, ene. 2021, doi: 10.1016/j.jpdc.2020.09.009.
- [15] Nvidia, «Nvidia». <https://www.nvidia.com/>.
- [16] F. Yasmany Chávez y D. Gálvez Lio, «Aplicación del modelo de programación CUDA en la simulación de la evolución de secuencias genéticas», *Enfoque UTE*, vol. 8, n.º 2, pp. 78-93, mar. 2017, doi: 10.29019/enfoqueute.v8n2.159.
- [17] T. Jiang, J. L. Gradus, y A. J. Rosellini, «Supervised Machine Learning: A Brief Primer», *Behav. Ther.*, vol. 51, n.º 5, pp. 675-687, sep. 2020, doi: 10.1016/j.beth.2020.05.002.
- [18] S. Ren, K. He, R. Girshick, y J. Sun, «Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, n.º 6, pp. 1137-1149, jun. 2017, doi: 10.1109/TPAMI.2016.2577031.
- [19] V. Sharma y R. N. Mir, «Saliency guided faster-RCNN (SGFr-RCNN) model for object detection and recognition», *J. King Saud Univ. - Comput. Inf. Sci.*, sep. 2019, doi: 10.1016/j.jksuci.2019.09.012.

- [20] Y. Jia *et al.*, «Caffe: Convolutional Architecture for Fast Feature Embedding», en *Proceedings of the ACM International Conference on Multimedia - MM '14*, Orlando, Florida, USA, 2014, pp. 675-678, doi: 10.1145/2647868.2654889.
- [21] S.-H. Wang, V. V. Govindaraj, J. M. Górriz, X. Zhang, y Y.-D. Zhang, «Covid-19 classification by FGCNet with deep feature fusion from graph convolutional network and convolutional neural network», *Inf. Fusion*, vol. 67, pp. 208-229, mar. 2021, doi: 10.1016/j.inffus.2020.10.004.
- [22] K. He, X. Zhang, S. Ren, y J. Sun, «Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition», *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, n.º 9, pp. 1904-1916, sep. 2015, doi: 10.1109/TPAMI.2015.2389824.
- [23] M. A. Galarza Bravo y M. Flores, «Detección de peatones en la noche usando Faster R-CNN e imágenes infrarrojas», *Ingenius*, n.º 20, pp. 48-57, jun. 2018, doi: 10.17163/ings.n20.2018.05.
- [24] D. Chaves, S. Saikia, L. Fernández-Robles, E. Alegre, y M. Trujillo, «Una Revisión Sistemática de Métodos para Localizar Automáticamente Objetos en Imágenes», *Rev. Iberoam. Automática E Informática Ind.*, vol. 15, n.º 3, p. 231, jun. 2018, doi: 10.4995/riai.2018.10229.
- [25] M. J. Flores Calero, C. Conlago, J. Yunda, M. Aldás, y C. Flores, «Implementación de un algoritmo para la detección de señales de tránsito del Ecuador: Pare, Ceda el paso y Velocidad», *Ingenius*, n.º 20, pp. 9-20, jun. 2018, doi: 10.17163/ings.n20.2018.01.
- [26] R. Gandhi, «R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms». <https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e>.
- [27] Microsoft, «Getting Started - Visual Code Studio». <https://code.visualstudio.com/docs>.
- [28] L. Rampasek y A. Goldenberg, «TensorFlow: Biology's Gateway to Deep Learning?», *Cell Syst.*, vol. 2, n.º 1, pp. 12-14, ene. 2016, doi: 10.1016/j.cels.2016.01.009.

- [29] P. Janardhanan, «Project repositories for machine learning with TensorFlow», *Third Int. Conf. Comput. Netw. Commun. CoCoNet19*, vol. 171, pp. 188-196, ene. 2020, doi: 10.1016/j.procs.2020.04.020.
- [30] M. Massiris, C. Delrieux, y J. Á. Fernández, «Detección de equipos de protección personal mediante red neuronal convolucional YOLO», en *Actas de las XXXIX Jornadas de Automática, Badajoz, 5-7 de septiembre de 2018*, mar. 2020, pp. 1022-1029, doi: 10.17979/spudc.9788497497565.1022.
- [31] S. Nofallah *et al.*, «Machine learning techniques for mitoses classification», *Comput. Med. Imaging Graph.*, vol. 87, p. 101832, ene. 2021, doi: 10.1016/j.compmedimag.2020.101832.

5. ANEXOS

Anexo 1: Resultados del entrenamiento de la red neuronal.

```
Anaconda Prompt (Anaconda3)
y: 0.4965
Epoch 4/5
2020-12-02 17:04:33.351940: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1431 Filling up shuffle buffer <this ma
y take a while>: 27 of 38
2020-12-02 17:04:37.588285: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1931 Shuffle buffer filled.
37/37 [=====] - 272s 7s/step - loss: 7.6895 - accuracy: 0.5000 - val_loss: 7.4332 - val_accuac
y: 0.4965
Epoch 5/5
2020-12-02 17:09:04.936123: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1431 Filling up shuffle buffer <this ma
y take a while>: 27 of 38
2020-12-02 17:09:08.941175: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1931 Shuffle buffer filled.
37/37 [=====] - 280s 8s/step - loss: 7.6766 - accuracy: 0.5009 - val_loss: 7.4332 - val_accuac
y: 0.4965
[INFO] evaluating network...
C:\Users\Game\Anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use 'zero_division' parameter to con
trol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
      precision    recall  f1-score   support

with_mask         0.50         1.00         0.67         150
without_mask       0.00         0.00         0.00          150

   accuracy
macro avg   0.25         0.50         0.33         300
weighted avg 0.25         0.50         0.33         300

[INFO] saving mask detector model... path: C:\wamp64\www\titulacion\python\model\mask_model.h5
<base> C:\wamp64\www\titulacion\python>
```

```
Anaconda Prompt (Anaconda3)
2020-12-02 19:19:26.006756: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1431 Filling up shuffle buffer <this ma
y take a while>: 26 of 38
2020-12-02 19:19:30.281689: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1931 Shuffle buffer filled.
37/37 [=====] - 248s 7s/step - loss: 0.0776 - accuracy: 0.9726 - val_loss: 0.0209 - val_accuac
y: 0.9896
Epoch 9/10
2020-12-02 19:23:33.913280: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1431 Filling up shuffle buffer <this ma
y take a while>: 25 of 38
2020-12-02 19:23:39.465109: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1931 Shuffle buffer filled.
37/37 [=====] - 249s 7s/step - loss: 0.0744 - accuracy: 0.9743 - val_loss: 0.0257 - val_accuac
y: 0.9896
Epoch 10/10
2020-12-02 19:27:43.028969: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1431 Filling up shuffle buffer <this ma
y take a while>: 26 of 38
2020-12-02 19:27:47.614146: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1931 Shuffle buffer filled.
37/37 [=====] - 247s 7s/step - loss: 0.0791 - accuracy: 0.9743 - val_loss: 0.0123 - val_accuac
y: 0.9896
[INFO] evaluating network...
      precision    recall  f1-score   support

with_mask         0.98         1.00         0.99         150
without_mask       1.00         0.98         0.99         150

   accuracy
macro avg   0.99         0.99         0.99         300
weighted avg 0.99         0.99         0.99         300

[INFO] saving mask detector model... path: C:\wamp64\www\titulacion\python\model\mask_model.h5
<base> C:\wamp64\www\titulacion\python>
```

```
Anaconda Prompt (Anaconda3)
2020-12-03 01:03:07.608183: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1431 Filling up shuffle buffer <this ma
y take a while>: 26 of 38
2020-12-03 01:03:12.545987: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1931 Shuffle buffer filled.
37/37 [=====] - 276s 7s/step - loss: 0.0303 - accuracy: 0.9872 - val_loss: 0.0072 - val_accuac
y: 0.9965
Epoch 14/15
2020-12-03 01:07:43.908639: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1431 Filling up shuffle buffer <this ma
y take a while>: 22 of 38
2020-12-03 01:07:49.965478: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1931 Shuffle buffer filled.
37/37 [=====] - 274s 7s/step - loss: 0.0525 - accuracy: 0.9846 - val_loss: 0.0415 - val_accuac
y: 0.9826
Epoch 15/15
2020-12-03 01:12:18.599129: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1431 Filling up shuffle buffer <this ma
y take a while>: 26 of 38
2020-12-03 01:12:22.888226: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:1931 Shuffle buffer filled.
37/37 [=====] - 286s 8s/step - loss: 0.0460 - accuracy: 0.9846 - val_loss: 0.0080 - val_accuac
y: 0.9965
[INFO] evaluating network...
      precision    recall  f1-score   support

with_mask         1.00         0.99         1.00         150
without_mask       0.99         1.00         1.00         150

   accuracy
macro avg   1.00         1.00         1.00         300
weighted avg 1.00         1.00         1.00         300

[INFO] saving mask detector model... path: C:\wamp64\www\titulacion\python\model\mask_model.h5
<base> C:\wamp64\www\titulacion\python>
```

```
Anaconda Prompt (Anaconda3)
2020-12-02 13:22:36.445105: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this may take a while): 26 of 38
2020-12-02 13:22:40.681298: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
37/37 [=====] - 249s 7s/step - loss: 0.0729 - accuracy: 0.9752 - val_loss: 0.0431 - val_accuracy: 0.9896
Epoch 19/20
2020-12-02 13:26:45.668445: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this may take a while): 26 of 38
2020-12-02 13:26:50.000911: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
37/37 [=====] - 247s 7s/step - loss: 0.0619 - accuracy: 0.9760 - val_loss: 0.0324 - val_accuracy: 0.9931
Epoch 20/20
2020-12-02 13:30:52.148144: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:143] Filling up shuffle buffer (this may take a while): 23 of 38
2020-12-02 13:30:57.930359: I tensorflow/core/kernels/data/shuffle_dataset_op.cc:193] Shuffle buffer filled.
37/37 [=====] - 249s 7s/step - loss: 0.0515 - accuracy: 0.9846 - val_loss: 0.0462 - val_accuracy: 0.9896
[INFO] evaluating network...
      precision    recall  f1-score   support

with_mask      0.98      1.00      0.99      150
without_mask    1.00      0.98      0.99      150

   accuracy      0.99      0.99      0.99      300
  macro avg      0.99      0.99      0.99      300
weighted avg      0.99      0.99      0.99      300

[INFO] saving mask detector model... path: C:\wamp64\www\titulacion\python\model\mask_model.h5
(base) C:\wamp64\www\titulacion\python>
```