UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

TRABAJO DE GRADO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

TEMA:

"ASISTENTE VIRTUAL DE ACOMPAÑAMIENTO MEDIANTE REDES NEURONALES APLICADOS EN UN SISTEMA EMBEBIDO QUE VERIFICA LA PRESENCIA DE NIÑOS EN BUSETAS ESCOLARES DURANTE EL TRAYECTO".

AUTOR: FLORES MURILLO RONALD GIOVANNI

DIRECTOR: MSC. VÁSQUEZ AYALA CARLOS ALBERTO

IBARRA - ECUADOR

2022

UNIVERSIDAD TÉCNICA DEL NORTE

BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley Orgánica de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte de manera digital para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1003136882		
APELLIDOS Y NOMBRES:	Flores Murillo Ronald Giovanni		
DIRECCIÓN:	Av. Jorge Guzmán Rueda y Eduardo Garzón Fonseca 105, Pasaje A, Cdla. La Victoria.		
EMAIL:	rgfloresm@utn.edu.ec		
TELÉFONO FIJO:	062 615481	**TELEFONO MÓVIL:**	0997398271

DATOS DE LA OBRA		
TÍTULO:	ASISTENTE VIRTUAL DE ACOMPAÑAMIENTO MEDIANTE REDES NEURONALES APLICADOS EN UN SISTEMA EMBEBIDO QUE VERIFICA LA PRESENCIA DE NIÑOS EN BUSETAS ESCOLARES DURANTE EL TRAYECTO	
AUTOR:	Flores Murillo Ronald Giovanni	
FECHA:	21 de febrero 2022	
PROGRAMA:	■ PREGRADO	☐ POSGRADO
TÍTULO POR EL QUE OPTA:	Ingeniero en Electrónica Y Redes de Comunicación	
DIRECTOR:	Ing. Carlos Alberto Vásquez Ayala, MSc	

2. **CONSTANCIAS**

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de esta y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 21 días del mes de Febrero de 2022.

EL AUTOR:

.....

Ronald Giovanni Flores Murillo

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

MAGISTER CARLOS ALBERTO VÁSQUEZ AYALA, MSC., DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN CERTIFICA:

Que, el presente trabajo de Titulación "ASISTENTE VIRTUAL DE ACOMPAÑAMIENTO MEDIANTE REDES NEURONALES APLICADOS EN UN SISTEMA EMBEBIDO QUE VERIFICA LA PRESENCIA DE NIÑOS EN BUSETAS ESCOLARES DURANTE EL TRAYECTO", fue realizado en su totalidad por el Sr. Ronald Giovanni Flores Murillo, bajo mi supervisión.

Es todo en cuanto puedo certificar en honor a la verdad.

CARLOS ALBERTO VASQUEZ AYALA

Digitally signed by CARLOS ALBERTO VASQUEZ AYALA
Date: 2022.02.21 15:33:06 -05'00'

Ing. Carlos Alberto Vásquez Ayala, MSc.

DIRECTOR DE TESIS

DEDICATORIA

Dedicado,

A mi familia, en especial a mi madre quien supo demostrarme que con esfuerzo y sacrificio los sueños y objetivos son posibles de alcanzar, quien con su sabiduría y experiencia supo guiarme y apoyarme en todo momento.

A mi hermana, quien es un ejemplo para seguir, que con sus palabras de aliento supo motivarme y darme fuerzas durante todo el transcurso de mi vida y de mi carrera universitaria.

A mis amigos que de una u otra manera aportaron para que pueda conseguir este logro, por su apoyo incondicional y desinteresado, que con su presencia hicieron que los días difíciles de universidad sean más llevaderos.

Ronald Flores

AGRADECIMIENTOS

Agradezco,

A Dios por brindarme la oportunidad vivir, a mis padres Juan y Fanny, a mi hermana Vanessa, quienes, con sus consejos, sabiduría, regaños, tiempo, aportaron a mi formación como persona y como profesional.

Al Ingeniero Carlos Vásquez director de mi proyecto de titulación, quien con su colaboración y tiempo supo guiarme para la culminación de este trabajo.

A los Ingenieros Jaime Michelena y Luis Suarez asesores del proyecto de titulación, que gracias a sus conocimientos y experiencia aportaron significativamente en el desarrollo de esta tesis.

A todas las personas que de una u otra manera me apoyaron durante mi carrera universitaria.

Muchas GRACIAS a todos.

Ronald Flores

RESUMEN

En el presente trabajo de titulación se realizará un Asistente Virtual mediante Redes Neuronales para la verificación de presencia en Busetas Escolares que analiza video en tiempo real con el fin de realizar tareas como el reconocimiento facial y conteo de personas, con lo cual se pretende aportar a la seguridad del niño, mientras este se moviliza en el transporte. Al ser un escenario móvil, el uso de sistemas embebidos es ideal, sin embargo, es necesario mencionar que los recursos de procesamiento son limitados, lo que en gran medida limita la implementación unificada de las diferentes técnicas de inteligencia artificial seleccionadas para la identificación facial sobre un gran número de personas y para el conteo de personas. El proyecto se encuentra desarrollado en base a la metodología en cascada con el cual se establecen los parámetros y funciones que se deben cumplir en cada una de sus fases que dará como resultado un modelado eficiente del sistema. Dicho modelamiento se encuentra desarrollado en cuatro fases, la fase 1 se unen la etapa de definición de requerimientos y de análisis, se reúnen todos los requisitos que debe cumplir el software bajo las necesidades de funcionamiento del cliente, la fase 2 es la etapa de diseño la cual traduce los requisitos en una representación de software en donde se enfocan las tareas de estructura de datos, arquitectura del software, representaciones de interfaz y el detalle procedimental o algoritmo, la fase 3 se suman las etapas de implementación y pruebas, donde los requisitos y especificaciones de las fases anteriores se traducen en algoritmos comprensibles por el terminal y así obtener un programa ejecutable, esta etapa se centrará en los procesos lógicos del software, asegurando que todas las sentencias se han comprobado, y en la detección de errores mediante pruebas pertinentes y por último la fase 4 en donde se ejecuta tareas de mantenimiento que conlleva modificaciones o creación de funciones adicionales en el prototipo a lo largo de su vida útil. Esta etapa no ha sido considerada aún en este proyecto.

ABSTRACT

In this degree work, a Virtual Assistant will be carried out through Neural Networks for the verification of presence in School Buses that analyzes video in real time in order to perform tasks such as facial recognition and people counting, with which it is intended to contribute to the safety of the child, while it is mobilized in transport. Being a mobile scenario, the use of embedded systems is ideal, however, it is necessary to mention that the processing resources are limited, which greatly limits the unified implementation of the different artificial intelligence techniques selected for facial identification on a large number of people and for people counting. The project is developed based on the cascade methodology with which the parameters and functions that must be fulfilled in each of its phases are established, which will result in an efficient modeling of the system. This modeling is developed in four phases, phase 1 joins the requirements definition and analysis stage, all the requirements that the software must meet under the client's operating needs are met, phase 2 is the design stage which translates the requirements into a software representation where the tasks of data structure, software architecture, interface representations and the procedural or algorithm detail are focused, phase 3 adds the implementation and testing stages, where the requirements and specifications of the previous phases are translated into algorithms understandable by the terminal and thus obtain an executable program, this stage will focus on the logical processes of the software, ensuring that all sentences have been verified, and on the detection of errors through relevant tests and finally phase 4 where maintenance tasks are carried out that entail modifications or creation of functions. additional ions in the prototype throughout its useful life. This stage has not yet been considered in this project.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

1. CAPITULO I: ANTECEDENTES

1.1. Tema

Asistente Virtual de acompañamiento mediante Redes Neuronales aplicados en un sistema embebido que verifica la presencia de niños en Busetas Escolares durante el trayecto.

1.2. Problema

Según la DINASED, afirma que, en el 2018, la Fiscalía ha delegado a su institución 55 denuncias de intento o secuestro de menores de edad en el país. Según la autoridad, se están investigando si existen bandas dedicadas a raptar a niños y adolescentes por lo que insiste a la ciudadanía que denuncie estos casos. La mayoría de los secuestros a menores de edad se registra cuando van del colegio a la casa, y viceversa, y en lugares públicos, como parques o centros comerciales. (El Comercio, 2016) La mayoría de los casos el plagio a menores tiene fines económicos. En la Policía este tipo de secuestro, conocido como extorsivo, es considerado distinto al secuestro exprés: la principal razón es que el primero implica una mayor planificación y logística (inteligencia previa, lugar de cautiverio, insumos para alimentar a la víctima, etc.). El plagio es un delito propio del crimen organizado, siempre es planificado y se estudia a la víctima, a veces durante meses. El 90% de casos se origina por información proporcionada por amigos cercanos de la familia, trabajadores y hasta guardias de seguridad.

En lo que respecta al servicio de transporte escolar, este constituye un servicio personalizado para estudiantes del Sistema Educativo Nacional que requieren de movilización, desde sus hogares o la parada correspondiente, hasta las instituciones educativas y viceversa, de acuerdo con la necesidad de cada estudiante. (A.N.T., 2014). El servicio de transporte escolar que se preste en las instituciones educativas es opcional para los representantes de los estudiantes, para la contratación del servicio de transporte escolar, el directivo de cada institución educativa y el

representante del comité central de padres de familia, desarrollarán y elaborarán las bases del concurso para la adjudicación del servicio de transporte escolar. (Ministerio de Educación, 2018). En dicho contrato se estipulan todas las cláusulas que los actores deben cumplir, se detalla también, la lista de los beneficiarios con sus respectivas rutas y paradas. El contrato deberá ser entregado a la Dirección Distrital al que corresponda la institución educativa; una copia reposará en la secretaría de la institución educativa; otra copia deberá entregarse a la operadora de transporte escolar contratada; y, la tercera copia se remitirá a al organismo de tránsito competente, de acuerdo con su jurisdicción territorial. (Ministerio de Educación, 2018).

Según el Acuerdo Ministerial Nro. MINEDUC-MINEDUC-2018-00077-A, el traslado de escolares presenta, dentro del conjunto del transporte de viajeros, una especial relevancia dada la singularidad de sus usuarios. La atención que este modo de transporte presta a los estudiantes requiere que se intensifiquen las garantías de su prestación, fundamentalmente en materia de su seguridad. Dicho acuerdo en su Artículo 9, menciona que para garantizar la integridad física y psicológica de las y los estudiantes de Educación Inicial y Educación General Básica Preparatoria y Elemental (niños entre 3 y 8 años), los directivos de las instituciones educativas deberán designar, para cada periodo lectivo, a una persona adulta que acompañe a los estudiantes en cada unidad de transporte escolar durante todo el trayecto desde y hacia el establecimiento educativo. No obstante, en instituciones donde la cantidad de estudiantes es extensa y el servicio de recorridos requiere más busetas, es casi imposible cumplir la normativa ya que la disponibilidad de profesores es limitada y se estaría dejando de lado las diferentes responsabilidades del docente. Sin un acompañante las funciones de este recaen en el chofer quien tiene que: recoger y acompañar a los estudiantes desde la parada hasta el interior del centro escolar, comprobar las subidas y bajadas de los estudiantes en las paradas establecidas de acuerdo con la información facilitada por la

institución educativa, comprobar que sólo utilice el servicio de transporte escolar el estudiante beneficiario del mismo, ayudar a subir y bajar del vehículo de transporte escolar a los estudiantes con movilidad reducida, discapacidad o condición discapacitante. Todo este proceso genera retraso en el recorrido y sin añadir el tiempo que se pierde por situaciones imprevistas en la ruta de cada niño, esto a su vez genera preocupación tanto en los padres de familia que desconocen si su hijo(a) llego a su destino, como en el chofer que tiene que verificar si los niños se han bajado en el lugar indicado.

Este proyecto tiene como finalidad aportar a la seguridad del usuario del transporte escolar, así como también generar confianza en los padres de familia hacia el servicio, para ello se tiene como objetivo el diseño de un sistema de monitoreo con Visión Artificial y alertas en un dispositivo embebido que verifique la presencia del niño durante el trayecto de la buseta escolar desde y hacia el establecimiento educativo y de esta manera aportar a la seguridad del mismo, utilizando metodologías de análisis como procesado de imágenes, reconocimiento de patrones, visión por ordenador y redes neuronales. El interés y preocupación tanto de padres de familia, autoridades de las instituciones educativas, policía nacional y en sí de la sociedad es buscar métodos que garanticen y controlen la seguridad de los usuarios del transporte escolar. Con el avance y demanda de nuevas tecnologías, el incremento del uso de dispositivos inalámbricos, las diferentes técnicas de reconocimiento facial; que se han convertido en una solución eficaz para videovigilancia, la facilidad de adquisición, permiten el desarrollo de aplicaciones que acoplan estas tecnologías para crear sistemas o equipos automáticos que brinden un ambiente de satisfacción o control en una necesidad. En el presente proyecto se propone una solución para aumentar la seguridad de los estudiantes durante el recorrido en el servicio de transporte escolar

usando algunas de las tecnologías mencionadas, los beneficios del software libre, con el fin de brindar bienestar de los usuarios de dicho sistema de transporte.

1.3. Objetivos

1.3.1. Objetivo General

Diseño de un Sistema de acompañamiento virtual con Visión Artificial y alertas en un dispositivo embebido que verifique la presencia del niño durante el trayecto de la buseta escolar y de esta manera aportar a la seguridad del mismo.

1.3.2. Objetivos Específicos

- Determinar mediante revisión literaria que modelos de redes neuronales son adecuadas para la detección de rostros en una secuencia de video.
- Verificar la factibilidad de las técnicas de reconocimiento y/o conteo de personas mediante pruebas de funcionamiento para así establecer el método a aplicar.
- Definir los requerimientos de software y hardware que necesita el prototipo para la etapa de ejecución.
- Diseñar el sistema de acompañamiento virtual y notificaciones por internet que permita verificar la presencia de niños en busetas escolares durante el trayecto.

1.4. Alcance

El presente proyecto estará enfocado en el desarrollo de un sistema de monitoreo y alerta utilizando técnicas para el procesamiento/análisis digital de imágenes aplicadas a la detección y reconocimiento de personas. Para esta propuesta el grupo seleccionado de personas son los niños. Específicamente menores de entre 3 y 8 años. El objetivo es aportar a la seguridad durante el trayecto de la buseta escolar, especialmente el servicio prestado a niños de Educación Inicial y

Educación General Básica Preparatoria y Elemental los cuales están dentro del rango de edad mencionado.

Los recursos bibliográficos son esenciales para la obtención de bases teóricas, es por ello que se realizará una revisión detallada de la literatura referente al tema, y en especial se hará una búsqueda bibliográfica de artículos y estudios científicos similares o que contengan temas relacionados con el objeto de estudio.

Luego de analizar la teoría actual existente y siguiendo el orden de los objetivos específicos planteados, se ejecutará la siguiente metodología: El módulo adquisición de imágenes, corresponderá a la fase 1 del proyecto, en esta etapa se recopilará fotos de los beneficiarios, estas deben ser lo más diferente posible entre sí para lograr una mejor generalización; la cantidad de imágenes que se necesitará por niño se estimará según la teoría revisada y según el modelo de red neuronal. Las imágenes obtenidas se organizarán en carpetas, para posteriormente normalizar las imágenes en cuanto a iluminación, orientación del rostro, tamaño del rostro, eliminación de ruido, etc. Posteriormente se efectuará la extracción de atributos. Todo este procesamiento de imágenes se realiza con el objetivo de generar el formato de archivo adecuado que servirá de datos de entrada a la red neuronal. El proceso para contar a los niños que suben al recorrido mediante visión artificial es similar, en este caso una cámara ubicada estratégicamente captara en video el movimiento de los menores al momento de subirse al transporte, una minicomputadora analizara, procesara, acondicionara y determinara la información necesaria para evaluar el ingreso o salida.

Como se propuso en los objetivos se evaluará la efectividad de estas dos técnicas y se determinara cual es la más adecuada, sin embargo, es muy probable que sea necesario conjugar ambas funcionalidades en un solo sistema. Los parámetros para comprobar serán: si el sistema

detecta la presencia del niño durante el trayecto, si el sistema es capaz de reconocer al niño cuando su rostro no esté completamente visible a las cámaras.

Asumiendo que el asistente virtual necesitará las funciones de reconocimiento facial y conteo de personas, la fase de entrenamiento será la siguiente, en donde se lleva a cabo: la preparación de datos y entrenamiento de la red. La preparación de datos está dada por la selección de las imágenes adaptables al ambiente de prueba y para el módulo de entrenamiento de la red neuronal se analizará las técnicas más adecuadas de procesamiento de imágenes. El conteo de los niños se efectuará de la misma forma siguiendo sus fases correspondientes. Todo estas funcionalidades añaden costo de procesamiento al sistema, especialmente el realizado por el reconocimiento facial mediante redes neuronales; existen métodos que se utilizan para reducir costo de procesamiento, como por ejemplo utilizar base de datos remotas, en donde se realiza el procesamiento de la información para luego ser enviada de vuelta al sistema en la buseta, las bases de datos deben proporcionar robustez y rapidez al momento de procesar las imágenes de todas las iteraciones en el procesamiento; sin embargo esto conlleva un riesgo de seguridad que atenta contra la integridad de los menores, ya que según uno de los artículos del Código de la Niñez y Adolescencia se prohíbe la publicación o exhibición de imágenes que permitan la identificación o individualización de un niño, niña o adolescente. A pesar de esto se puede hacer el trámite correspondiente con la institución y padres de familia para que estos autoricen trabajar con los menores, caso contrario se efectuaran las pruebas con personas adultas.

Es necesario definir los requerimientos de software y hardware, en la actualidad se pueden encontrar varios tipos de ordenadores de placa reducida, por lo cual es necesaria la valoración de características para su implementación dentro del proyecto actual. La cámara es el dispositivo esencial para la adquisición de las imágenes en tiempo real, para lo cual la selección se basa en

características como: Resolución (pixeles), formato, calidad de captura e iluminación, precio. La precisión del sistema depende mucho de la calidad de la imagen que brinde la cámara. Debido al entorno de desarrollo del proyecto se ha definitivo utilizar dos cámaras dentro de la buseta, una ubicada a la altura del espejo retrovisor y la otra en la esquina trasera de la buseta apuntando hacia la puerta; la ubicación variará dependiendo de la necesidad y características del vehículo. Es conveniente aclarar que sistema es un prototipo que no será implementado, no obstante, se efectuará el análisis de la distribución del sistema en la buseta, la alimentación del prototipo, la seguridad que tendrá y los períodos de funcionamiento.

La última etapa consta de la integración de las etapas anteriores en un solo sistema, con el prototipo finalizado se procederá a realizar las pruebas de funcionamiento, la generación de alertas, envío de datos y correcciones del sistema. En la generación de alertas, estas serán enviadas cuando no se reconozca el rostro del niño (A), cuando el número de ocupantes que deberían entrar en el recorrido no sea el correspondiente (B) y cuando se haya desconectado la cámara (C). Las notificaciones serán enviadas mediante una aplicación de mensajería instantánea a los destinatarios correspondientes, por ejemplo, la alerta (A) al padre de familia, (B) y (C) al chofer. El ultimo test será el del módulo de comunicación, la función principal de este módulo es él envió de las alertas y datos, el sistema deberá enviar datos a través de una conexión a internet que será proporcionada por un modem USB. Por último, se determinará si es necesario realizar cambios para mejorar la efectividad del sistema y además generar las respectivas conclusiones y recomendaciones en base al proceso desarrollado.

Para una etapa de implementación del prototipo es de gran importancia la protección de la o las cámaras en el vehículo, debido a que pueden ser manipuladas por los pasajeros. Para este proyecto se pretende trabajar con cámaras de tamaño reducido, para que facilite su instalación,

procurando ocultar todos los cables para evitar daños. La placa embebida, se encargará de alimentar las cámaras. El dispositivo central se alimenta con 5V / 4A DC, por lo que se realizara las conexiones necesarias para alimentar al prototipo mediante la corriente del auto. Dependiendo de los requerimientos del usuario se considerará añadir una batería extra que permita el constante funcionamiento.

1.5. Justificación

La inseguridad que se vive actualmente es crítica, y los grupos más vulnerables siempre van a ser los niños, los medios de comunicación reportan a diario secuestro de niños con fines de trata de personas, los padres de familia han tomado diferentes medidas para salvaguardar la seguridad de cada uno de estos, sin embargo, no ha sido lo suficiente para terminar con este mal. De igual manera las diferentes instituciones educativas han empezado a implementar medidas de seguridad para tratar de disminuir situaciones que atenten contra la integridad de sus estudiantes, sin embargo, estas medidas son ejecutadas dentro de la unidad educativa, dejando como principal responsable de la seguridad de los menores que utilizan el servicio de transporte escolar al chofer de este. (Cevallos, 2018).

Según el Acuerdo Ministerial Nro. MINEDUC-MINEDUC-2018-00077-A, el traslado de escolares presenta, dentro del conjunto del transporte de viajeros, una especial relevancia dada la singularidad de sus usuarios. La atención que este modo de transporte presta a los estudiantes requiere que se intensifiquen las garantías de su prestación, fundamentalmente en materia de su seguridad. Según dicho acuerdo ministerial una persona adulta debe acompañar a los estudiantes en cada unidad de transporte escolar durante todo el trayecto desde y hacia el establecimiento educativo. Este requerimiento implica recursos humanos, los cuales no siempre van a estar

disponibles. Es por ello por lo que se busca otras alternativas que ayuden a garantizar la seguridad durante el trayecto.

Actualmente en el Ecuador, desde el 2012 se ha implementado el proyecto de seguridad vial y ciudadana para el transporte público y comercial denominado “Transporte Seguro”, es una iniciativa del Gobierno Nacional implementado por la Agencia Nacional de Tránsito (ANT) y coordinado con el Sistema Integrado de Seguridad ECU 911. El proyecto está orientado a brindar una inmediata atención a situaciones de emergencia reportadas a la 911 y las generadas por el monitoreo de video vigilancia y alarmas, brindando respuesta inmediata con organismos públicos como: policía nacional, cruz roja, bomberos, fuerzas armadas, con la finalidad de apoyar permanentemente a la seguridad de toda la ciudadanía. Sin embargo, este es un proyecto que solo incluye unidades de transporte públicos como buses y taxis.

La visión artificial es una de las áreas de la computación que tiene infinidad de aplicaciones en el campo de la seguridad y la vigilancia (SUM & MENG, 2015). El poder automatizar ciertas funciones como el reconocimiento, seguimiento, conteo de personas abre un gran campo de posibilidades para detectar situaciones anómalas en secuencias de video tomadas en un entorno concreto. Un sistema informático con estas capacidades puede convertirse en una herramienta eficaz para discriminar los sucesos que merecen ser informados o notificados a las autoridades correspondientes, permitiéndoles tomar decisiones más acertadas y simplificando el proceso de indagación. Así, por ejemplo, se puede tener decenas de cámaras realizando tareas de vigilancia. (García Mata, 2010)

2. CAPITULO II: MARCO TEÓRICO

En el presente capítulo se dará a conocer aspectos sobre la visión por computador para comprender el reconocimiento facial, sus aplicaciones, algoritmos, procedimiento y las técnicas que se pueden utilizar. Con ello, se introduce a un contraste entre redes Neuronales Artificiales y Redes Neuronales Profundas, sus aplicaciones y arquitecturas, detallando las Redes Neuronales Convolucionales que presenten valores de precisión altos y bajo consumo de recursos computacionales.

2.1. Visión por Computadora

La visión artificial es un campo de la Inteligencia Artificial, que proporciona información a las computadoras mediante el análisis y procesamiento de imágenes. Su enfoque principal es obtener información visual de la captura de imágenes tomadas del entorno de trabajo para extraer las características de la imagen, las mismas que son el resultado de las proyecciones en el plano de la imagen de las características físicas del objeto.

2.1.1. Definición.

Según varias fuentes bibliográficas definen a la Visión por Computadora de la siguiente manera:

Según Brownlee, (2019) la visión por computadora (CV), se define como un campo de estudio que busca desarrollar técnicas para ayudar a las computadoras a "ver" y comprender el contenido de imágenes digitales como fotografías y videos.

Por otro lado Dawson-Howe, (2014) define a la visión por computadora como el análisis automático de imágenes y videos por computadoras con el fin de obtener una cierta comprensión del mundo. La visión por computadora se inspira en las capacidades del sistema de visión humana.

Como se muestra en la Figura 1, la visión por computadora es un campo interdisciplinario de la Inteligencia Artificial (AI) que tiene como objetivo brindar a las computadoras y otros dispositivos con capacidades informáticas una comprensión de alto nivel tanto de imágenes digitales como de videos, incluida la funcionalidad para adquirir, procesar y analizar imágenes digitales. Esta es la razón por la cual la CV es, en parte, otra subárea de AI que depende en gran medida del aprendizaje automático y los algoritmos de aprendizaje profundo para crear aplicaciones. Además, la visión por computadora se compone de varias tecnologías que trabajan juntas: gráficos por computadora, matemáticas o incluso física (Fernández Villán, 2019).

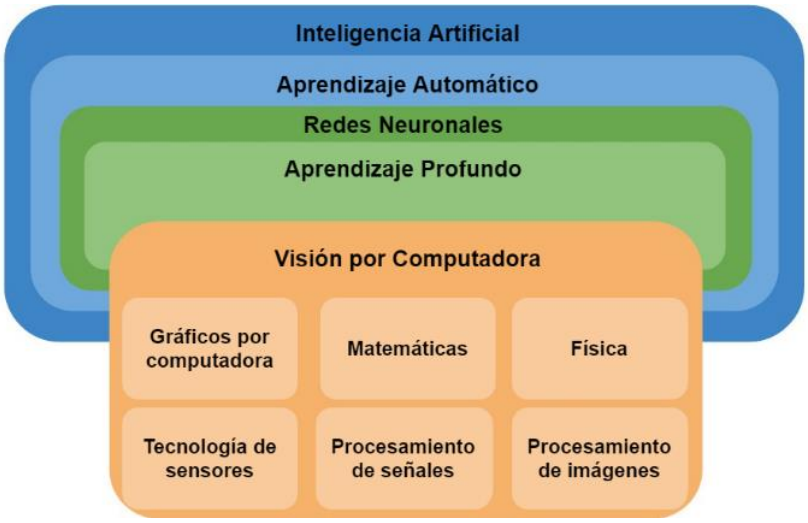


Figura 1. Estructura jerárquica de la AI.

Fuente: Adaptado de (Fernández Villán, 2019)

2.1.2. Percepción de la Visión por Computadora.

Un sistema de AI puede percibir su entorno y tomar acciones basadas en estas percepciones. La visión por computadora se refiere a la parte de percepción visual. Es la ciencia de percibir y comprender el mundo a través de imágenes y videos mediante la construcción de un modelo físico del mundo para que el sistema de AI pueda tomar las medidas adecuadas (Elgendy, 2020).

2.1.2.1. Percepción Visual.

La percepción visual es el acto de observar patrones y objetos a través de la vista o la entrada visual. Por ejemplo, la percepción de la cara es crítica para el funcionamiento social normal, las caras proporcionan información visual clave que se utiliza para discriminar a una persona de otra todos los días. Dado que la percepción facial es evolutivamente relevante en todas las especies (Freiwald et al., 2016). El objetivo no solo es capturar el entorno, si no también, construir sistemas que realmente puedan entender el medio a través de entradas visuales.

2.1.2.2. Sistemas de Visión.

En las últimas décadas, las técnicas tradicionales de procesamiento de imágenes se usaban para considerarse sistemas de visión por computadora. Es completamente diferente el procesamiento de una imagen, a comprender lo que sucede dentro de la misma.

Los sistemas de visión son prácticamente los mismos para humanos y la mayoría de los organismos vivos. Como se aprecia en la Figura 2, consisten en: un sensor o un ojo para capturar la imagen y un cerebro para procesar e interpretar la imagen. Luego, el sistema genera una predicción de los componentes de la imagen en función de los datos que se tiene sobre la información que se extrae de la imagen (Elgendy, 2020).

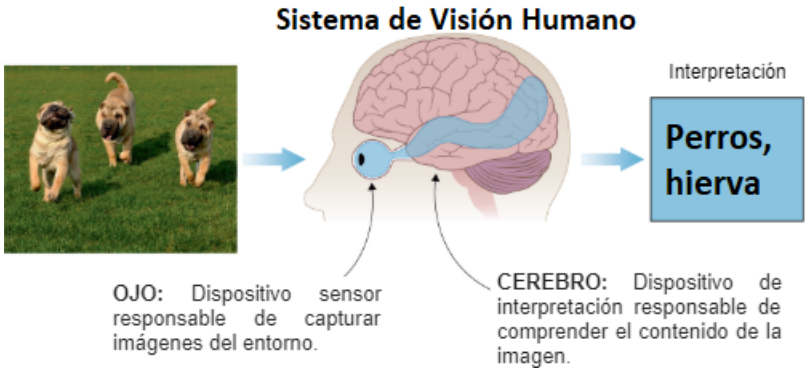


Figura 2. El sistema de visión humana utiliza el ojo y el cerebro para detectar e interpretar una imagen.

Fuente: Adaptado de (Elgendy, 2020)

Con el aumento de la capacidad visual de las máquinas, se ha tratado de imitar el sistema de visión humana, para lo cual se necesita los mismos dos componentes principales: un dispositivo sensor para imitar la función del ojo, y un poderoso algoritmo para imitar la función cerebral al interpretar y clasificar el contenido de la imagen.

2.1.2.3. Dispositivo de detección.

Los sistemas de visión están diseñados para cumplir una tarea específica y un aspecto importante del diseño de un sistema de visión es seleccionar el dispositivo sensor para capturar el entorno de ese ambiente específico. Ya sea una cámara, radar, rayos X, tomografía computarizada o una combinación de más de un dispositivo para proporcionar una escena completa del entorno para cumplir con la tarea en cuestión.

2.1.2.4. Dispositivo de interpretación.

Se encargan de procesar la mayor parte del algoritmo de visión por computadora. El intérprete es el cerebro del sistema de visión. Su función es tomar la imagen de salida del dispositivo de detección y aprender características y patrones para identificar sus objetos. De esta manera, surgieron las Redes Neuronales Artificiales (ANN).

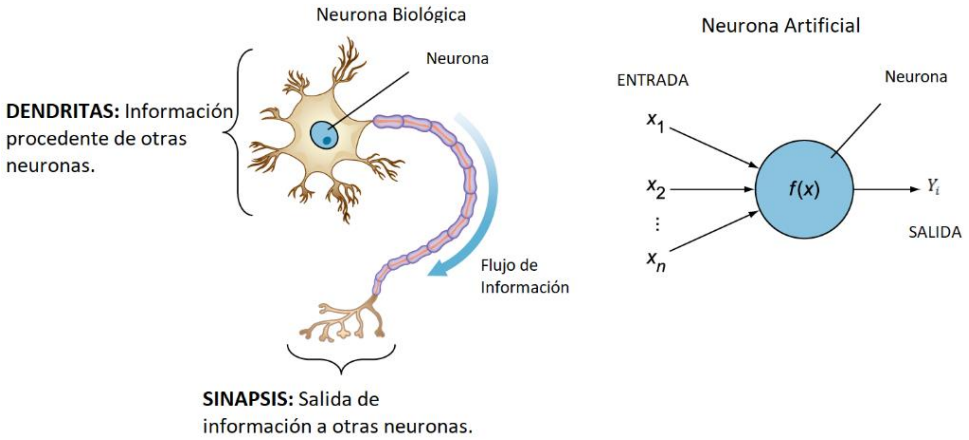


Figura 3. Similitudes entre las neuronas biológicas y los sistemas artificiales.

Fuente: Adaptado de (Elgendy, 2020)

En la Figura 3, se aprecia una analogía entre las neuronas biológicas y los sistemas artificiales. Ambos contienen un elemento de procesamiento principal llamado neurona, con señales de entrada $X_{1,}\ X_{2,}\ \ldots, X_n$ y una salida Y_i.

El comportamiento de aprendizaje de las neuronas biológicas inspiró a los científicos a crear una red de neuronas que están conectadas entre sí. Imitando cómo se procesa la información en el cerebro humano, cada neurona artificial individual disparará una señal a todas las neuronas a las que está conectada cuando se activan suficientes señales de entrada (Elgendy, 2020).

Las neuronas tienen un mecanismo muy simple a nivel individual; pero cuando se tiene millones de estas neuronas apiladas en capas y conectadas entre sí, cada neurona está conectada a miles de otras neuronas, produciendo un comportamiento de aprendizaje.

La construcción de una red neuronal multicapa se denomina aprendizaje profundo. Los métodos de aprendizaje profundo aprenden representaciones a través de una secuencia de transformaciones de los datos a través de capas de neuronas (Bhanu & Kumar, 2017).

2.1.3. Etapas de la Visión por Computadora: Vista General.

Un sistema típico de visión utiliza una secuencia de pasos distintos para procesar y analizar datos de imágenes. Muchas aplicaciones de visión comienzan adquiriendo imágenes y datos, luego procesan esos datos, realizan algunos pasos de análisis y reconocimiento, y finalmente hacen una predicción basada en la información extraída. Por ejemplo:

- Una computadora recibe información visual de un dispositivo de imágenes. Esto generalmente se captura como una imagen o una secuencia de imágenes (video).
- Cada imagen se envía a través de varios pasos de preprocesamiento cuyo propósito es estandarizar cada imagen. Los pasos comunes de preprocesamiento incluyen cambiar

el tamaño de una imagen (estandarización), difuminar, rotar, cambiar su forma o transformar la imagen de un color a otro.

- Se extrae características que permiten definir ciertos objetos, y generalmente son información sobre la forma o el color del objeto. El resultado de este proceso es un vector de características que es una lista de formas únicas que identifican el objeto.

- Finalmente, estas características se introducen en un modelo de clasificación. Este paso analiza el vector de características del paso anterior y predice la clase de la imagen (Elgendy, 2020). La Figura 4 muestra como fluye la imagen a través del proceso de clasificación descrito.

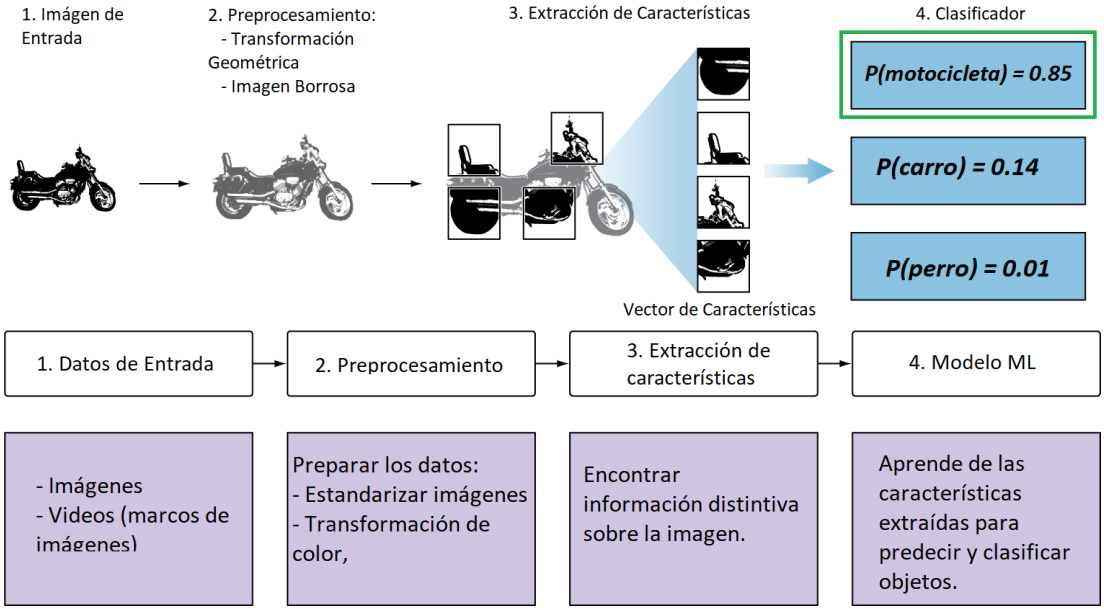


Figura 4. Cadena de etapas de Visión por Computadora.

Fuente: Adaptado de (Elgendy, 2020)

Un sistema de visión efectivo debe ser capaz de ver en cualquier escenario y aun así extraer datos significativos. Las computadoras funcionan bien para problemas estrictamente restringidos, no para abrir problemas ilimitados como la percepción visual (Forsyth & Ponce, 2003).

2.1.4. Fundamentos de la Imagen.

Las imágenes se pueden ver como una vista bidimensional (2D) de un mundo 3D. Una imagen digital es una representación numérica, normalmente binaria, de una imagen 2D como un conjunto finito de valores digitales, que se denominan píxeles. Por lo tanto, el objetivo de la visión por computadora es transformar estos datos 2D en lo siguiente:

- Una nueva representación (una nueva imagen)
- Una decisión (realizar una tarea concreta)
- Un nuevo resultado (clasificación correcta de la imagen)
- Extracción de información útil (detección de objetos) (Fernández Villán, 2019)

La visión por computadora puede abordar problemas comunes (o dificultades) cuando se trata de técnicas de procesamiento de imágenes:

- Imágenes ambiguas porque se ven afectadas por la perspectiva, lo que puede producir cambios en la apariencia visual de la imagen. Por ejemplo, el mismo objeto visto desde diferentes perspectivas puede dar como resultado diferentes imágenes.
- Imágenes comúnmente afectadas por muchos factores, como la iluminación, el clima, los reflejos y los movimientos.
- Los objetos en la imagen también pueden estar ocluidos por otros objetos.

Al desarrollar un proyecto de CV, se debe tener en cuenta todos estos factores. Una buena aproximación es tener muchas imágenes de prueba para validar el algoritmo incorporando algunas dificultades (Fernández Villán, 2019).

2.1.4.1. Visión por Computadora y Procesamiento de Imágenes.

La visión por computadora es diferente del procesamiento de imágenes. El procesamiento de imágenes es el proceso de crear una nueva imagen a partir de una imagen existente, que generalmente simplifica o mejora el contenido de alguna manera. Es un tipo de procesamiento de señal digital y no se preocupa por comprender el contenido de una imagen.

Un sistema de visión por computadora dado puede requerir que el procesamiento de imágenes se aplique a la entrada sin procesar, por ejemplo, procesamiento previo de imágenes.

Como se muestra en la Figura 5, el procesamiento de imágenes incluye los siguientes tres pasos:

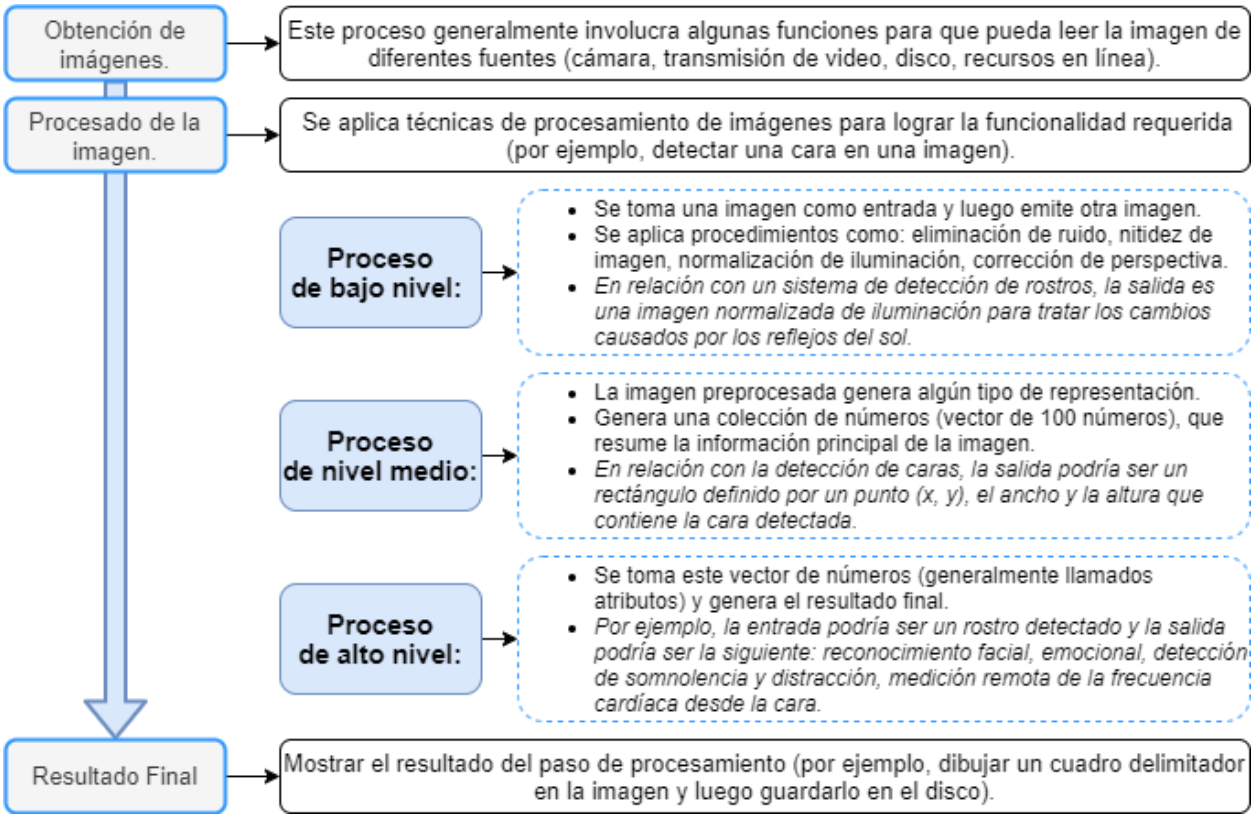


Figura 5. Pasos y descripción del procesado de una imagen.

Fuente: Autoría.

2.1.4.2. Imagen como funciones.

Las imágenes se pueden representar en función de dos variables X e Y que definen un área bidimensional. Las imágenes digitales están hechas de una cuadrícula de píxeles. El Píxel es el bloque de construcción en bruto de una imagen. Cada imagen consta de un conjunto de píxeles en el que sus valores representan la intensidad de la luz que aparece en un lugar determinado de la imagen.

Por ejemplo, la Figura 6, tiene 32 píxeles de ancho por 16 píxeles de alto = 512 píxeles. El eje X comienza de 0 a 31 y el eje Y de 0 a 15, cada píxel contiene un valor que representa la intensidad de la luz en este píxel específico. Los valores de píxel varían de 0 a 255.

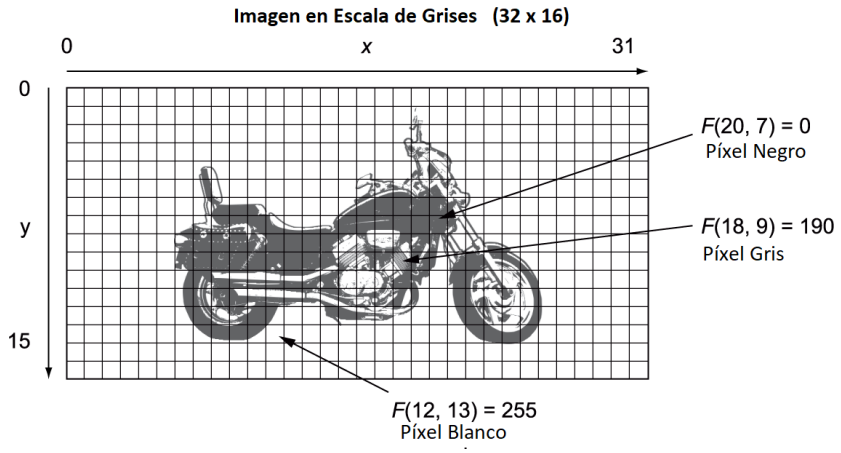


Figura 6. Los valores de píxel representan la intensidad de la luz que aparece en un lugar determinado de la imagen.

Fuente: Adaptado de (Elgendy, 2020)

Dado que el valor de píxel representa la intensidad de la luz, entonces el valor 0 representa píxeles muy oscuros (negro), 255 es muy brillante (blanco) y los valores intermedios representan la intensidad en la escala de grises según la Ecuación 1.

Escala de grises => F (x, y) da la intensidad en la posición (x, y)

Ecuación 1. Función que representa imágenes en la escala de grises.

Fuente: Adaptado de (Elgendy, 2019)

El sistema de coordenadas de la Figura 6, es similar al sistema de coordenadas cartesianas: las imágenes son bidimensionales y se encuentran en el plano xy. El origen $(0, 0)$ está en la parte superior izquierda de la imagen. Para representar un píxel en específico, se utiliza las siguientes notaciones: F como función (x, y), la ubicación del píxel en las coordenadas x, y. Por ejemplo, el píxel ubicado en x = 9 e y = 14 es blanco, se representa con la siguiente función: F (9, 14) =255.

En las imágenes en color, en lugar de representar el valor del píxel con solo un número, el valor se representa con tres números que significan la intensidad de cada color en este píxel. En el sistema RGB (sigla en inglés de *red, green, blue*), por ejemplo, el valor del píxel está representado por tres números: la intensidad del rojo, la intensidad del verde y la intensidad del azul. La Ecuación 2 representa imágenes en color en el sistema RGB:

Imagen en color en RGB => F (x, y) = [rojo (x, y), verde (x, y), azul (x, y)]

Ecuación 2. Función que representa imágenes en color en el sistema RGB

Fuente: Adaptado de (Elgendy, 2019)

Las imágenes pueden ser procesadas como una función de F (x, y) y operarlas matemáticamente para transformarlas en una nueva función de imagen G (x, y). La Tabla 1 muestra ejemplos de transformación de imágenes:

Tabla 1.

Ejemplos de transformación de imágenes.

Solicitud	Transformación
Oscurecer la imagen	G (x, y) = 0.5 * F (x, y)
Iluminar la imagen	G (x, y) = 2 * F (x, y)
Mover un objeto 150 píxeles hacia abajo	G (x, y) = F (x, y + 150)
Eliminar el gris de una imagen para transformarla en solo blanco y negro.	G (x, y) = {0 si F (x, y) <130, 255 de lo contrario}

Fuente: Autoría.

2.1.4.3. Imágenes en Color.

En el sistema RGB estándar, las imágenes en color tienen 3 canales (rojo, verde azul). Es decir, las imágenes en color están representadas por tres matrices: una representa la intensidad del rojo en el píxel, una para el verde y otra para el azul.

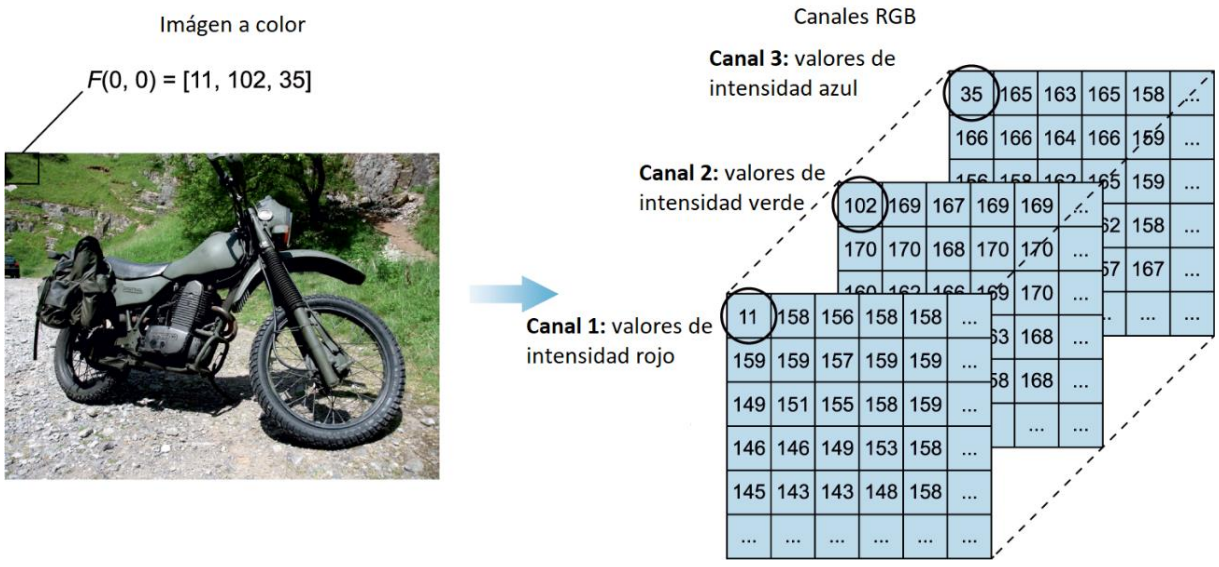


Figura 7. Las imágenes en color están representadas por canales rojos, verdes y azules, y las matrices se pueden utilizar para indicar la intensidad de esos colores**.**

Fuente: Adaptado de (Rosebrock, 2017) .

La Figura 7 se compone de tres canales, rojo, verde y azul. En este caso, se obtendrá 3 matrices apiladas una encima de la otra, una matriz 3D. La dimensionalidad de la imagen en color 700 * 700 será (700, 700, 3). Una primera matriz representa el canal rojo, luego cada elemento de esa matriz representa una intensidad de color rojo en ese píxel, del mismo modo en verde y azul. Cada píxel en la imagen en color tiene tres números (0 a 255) asociados. Estos números representan la intensidad del color rojo, verde y azul en ese píxel en particular.

El píxel (0,0) representa el píxel superior izquierdo y en la Figura 8, este píxel se ve de la siguiente manera:

Figura 8. Una imagen de hierba verde está hecha de tres colores de intensidad variable.

Fuente: Adaptado de (Rosebrock, 2017)

2.2. Biblioteca de Visión Artificial

OpenCV es una biblioteca de programación con capacidades de visión por computadora en tiempo real, es gratuita tanto para uso académico como comercial (licencia BSD). Inicialmente fue desarrollado por Intel, apoyado por Willow Garage y luego por Itseez (Pulli et al., 2012).

OpenCV es compatible con los marcos de aprendizaje profundo TensorFlow, Torch/PyTorch y Caffe. La biblioteca está escrita en C/C ++ optimizado, lo que favorece al procesamiento multinúcleo. Incluye enlaces en Python, Java y MATLAB/OCTAVE (Fernández Villán, 2019).

2.2.1. Módulos de OpenCV.

OpenCV tiene una estructura modular, lo que significa que el paquete incluye varias bibliotecas compartidas o estáticas. Donde cada módulo puede entenderse, en general, como dedicado a un grupo de problemas de visión por computadora. Esta división se puede ver en la Figura 9, donde se muestran los módulos principales:

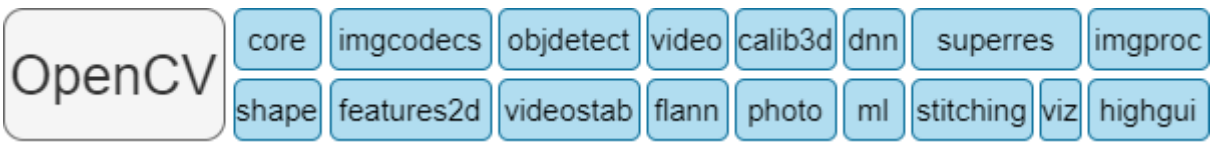


Figura 9. Módulos principales de la Biblioteca OpenCV

Fuente: Adaptado de (Fernández Villán, 2019).

Los diferentes módulos de la biblioteca OpenCV se describen en la Tabla 2:

Tabla 2.

Descripción de los módulos principales de la biblioteca OpenCV.

Módulo	*Descripción*
core:	Funcionalidad principal. Define estructuras de datos básicas y también funciones básicas utilizadas por todos los demás módulos.
imgproc:	Procesamiento de imágenes. Incluye filtrado y transformaciones de imágenes geométricas, conversión de espacio de color e histogramas.
imgcodecs:	Códecs de imágenes. Lectura y escritura de archivos de imagen.
videoio:	E/S de video. Una interfaz para captura de video y códecs de video.
highgui:	GUI de alto nivel. Proporciona una interfaz al usuario para crear y manipular ventanas que puedan visualizar/mostrar imágenes.
video:	Un módulo de análisis de video que incluye sustracción de fondo, estimación de movimiento y algoritmos de seguimiento de objetos.
calib3d:	Calibración de cámara y reconstrucción 3D. Cubre algoritmos básicos de geometría de múltiples vistas, estimación de pose de objeto, etc.
features2d:	Marco de características 2D. Este módulo incluye detectores de características, descriptores y descriptores coincidentes.
objdetect:	Detección de objetos. Detección de objetos e instancias de clases predefinidas (por ejemplo, caras, ojos, personas y automóviles).
dnn:	Módulo de red neuronal profunda (DNN). Contiene API para la creación de nuevas capas, conjunto de capas útiles construidas.
ml:	Aprendizaje automático. Es un conjunto de clases y métodos que se pueden usar para fines de clasificación, regresión y agrupación.

Fuente: Adaptado de (Fernández Villán, 2019).

2.2.2. Lenguajes de Programación.

Un lenguaje de programación es un lenguaje formal que comprende un conjunto de cadenas que producen varios tipos de salida de código de máquina. Los lenguajes de programación son un tipo de lenguaje informático y se utilizan en la programación informática para implementar algoritmos (Argente Villaplana, 2021). A continuación, se describen los lenguajes de programación que se usaran en este proyecto y que son compatibles con OpenCV:

- **Python**: se define como un “lenguaje de programación versátil, multiplataforma y multiparadigma que se destaca por su código legible y limpio”. Python atrae por su sencillez y exactitud en la sintaxis, ya que se trata de un lenguaje como cualquier otro, pero a nivel informático (Coursera, 2021).

- **SQL**: El lenguaje de consultas estructuradas o SQL (Structured Query Language) es un lenguaje de programación estandarizado que se utiliza para administrar bases de datos relacionales y realizar diversas operaciones con los datos que contienen (Sirkin, 2021).

2.3. Aprendizaje Automático

El aprendizaje automático (ML) es un subcampo de la informática en el que las máquinas aprenden a realizar tareas para las que no fueron programadas explícitamente. En resumen, las máquinas observan un patrón e intentan imitarlo de alguna manera que puede ser directa o indirecta (Trask, 2019). A medida que se introducen nuevos datos en estos algoritmos, aprenden y optimizan sus operaciones para mejorar el rendimiento, desarrollando “inteligencia” con el tiempo. Algunos de los algoritmos de Aprendizaje Automático se detallan a continuación:

2.3.1. Aprendizaje Supervisado

En el aprendizaje supervisado, los algoritmos trabajan con datos “etiquetados” (labeled data), intentado encontrar una función que, dadas las variables de entrada (input data), les asigne la etiqueta de salida adecuada. El algoritmo se entrena con un “histórico” de datos y así “aprende” a asignar la etiqueta de salida adecuada a un nuevo valor, es decir, predice el valor de salida (Simeone, 2018).

2.3.2. Aprendizaje no Supervisado

Los algoritmos de aprendizaje no supervisados toman un conjunto de datos que contiene solo entradas y encuentran estructura en los datos, como la agrupación o agrupación de puntos de datos. Por lo tanto, los algoritmos aprenden de los datos de prueba que no han sido etiquetados, clasificados o categorizados. En lugar de responder a la retroalimentación, estos algoritmos identifican elementos comunes en los datos y reaccionan en función de la presencia o ausencia de dichos elementos comunes en cada nueva pieza de datos (Tucker, 2004).

2.3.3. Aprendizaje Semi-supervisado

El aprendizaje semi-supervisado es una clase de tareas y técnicas de aprendizaje automático que también utilizan datos sin etiquetar para el entrenamiento, generalmente una pequeña cantidad de datos etiquetados con una gran cantidad de datos sin etiquetar. Se ha comprobado que los datos no etiquetados, cuando se usan junto con una pequeña cantidad de datos etiquetados, pueden producir una mejora considerable en la precisión del aprendizaje (Abney, 2007).

2.3.4. Aprendizaje por Refuerzo

Aprendizaje por refuerzo es una técnica de aprendizaje automático basada en comentarios en la que un agente aprende a comportarse en un entorno realizando las acciones y viendo los resultados de las acciones. Para cada buena acción, el agente obtiene comentarios positivos y, por cada acción incorrecta, el agente recibe comentarios negativos o penalización. El agente aprende a usar automáticamente comentarios sin ningún dato etiquetado, a diferencia del aprendizaje supervisado. Puesto que no hay datos etiquetados, el agente está obligado a aprender solo por su experiencia (Wiering & Van Otterlo, 2012).

El esquema de la Figura 10, muestra diferentes algoritmos ML, junto con las categorías.

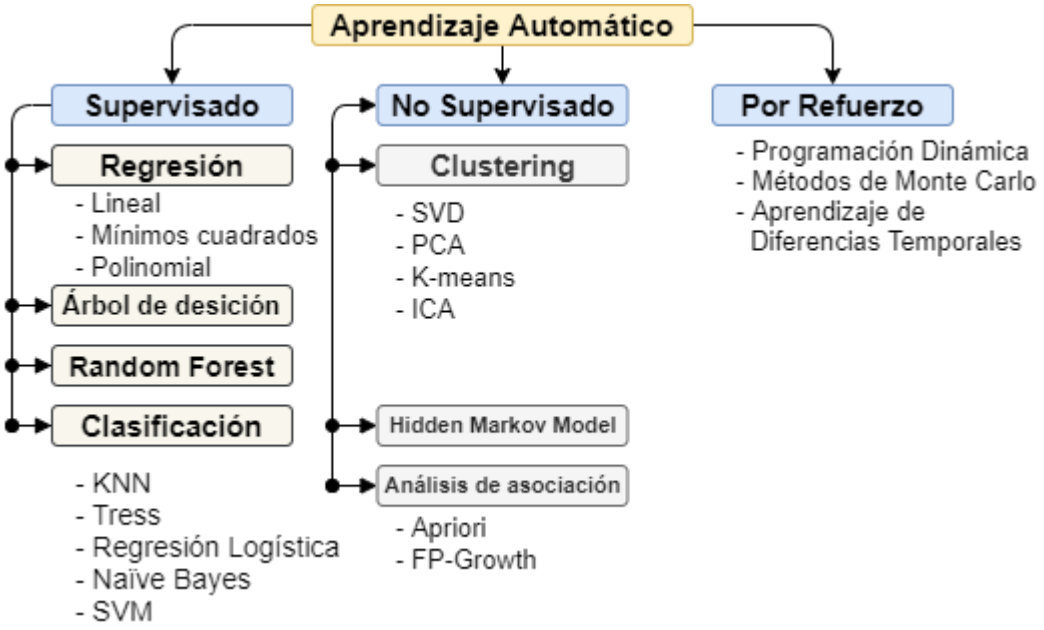


Figura 10. Tipos de algoritmos de aprendizaje automático.

Fuente: Adaptado de (Machine Learning Algorithms - Javatpoint, n.d.)

2.4. Aprendizaje Profundo y Redes Neuronales

El aprendizaje profundo es un subconjunto del aprendizaje automático basado en redes neuronales artificiales (ANN). El proceso de aprendizaje se llama profundo porque la estructura de redes neuronales artificiales se compone de varias capas de entrada, salida y ocultas. Cada capa contiene unidades que transforman los datos de entrada en información que la capa siguiente puede usar para realizar una tarea de predicción determinada. Gracias a esta estructura, una máquina puede aprender a través de su propio procesamiento de datos (Lazzeri, 2020).

2.4.1. El Perceptrón

La red neuronal más simple es el perceptrón, que consiste en una sola neurona. Conceptualmente, el perceptrón funciona de manera similar a una neurona biológica. Una neurona biológica recibe señales eléctricas de sus dendritas, modula las señales eléctricas en varias cantidades, y luego dispara una señal de salida a través de sus sinapsis solo cuando la fuerza total

de las señales de entrada excede un cierto umbral. La salida se alimenta entonces a otra neurona, y así sucesivamente.

Para modelar el fenómeno de la neurona biológica, la neurona artificial realiza dos funciones consecutivas: calcula la suma ponderada de las entradas para representar la fuerza total de las señales de entrada, y aplica una función de paso al resultado para determinar si se debe disparar la salida 1 si excede un cierto umbral o 0 si la señal no excede el umbral (Elgendy, 2020).

2.4.1.1. Pesos de Conexión.

A cada característica de entrada x_i se le asigna su propio peso w_i que refleja su importancia en el proceso de toma de decisiones. Es decir, algunas entradas se hacen más importantes que otras al darles más peso para que tengan un mayor efecto en la salida. Si el peso es alto, amplifica la señal de entrada y si el peso es bajo, disminuye la entrada (Elgendy, 2020).

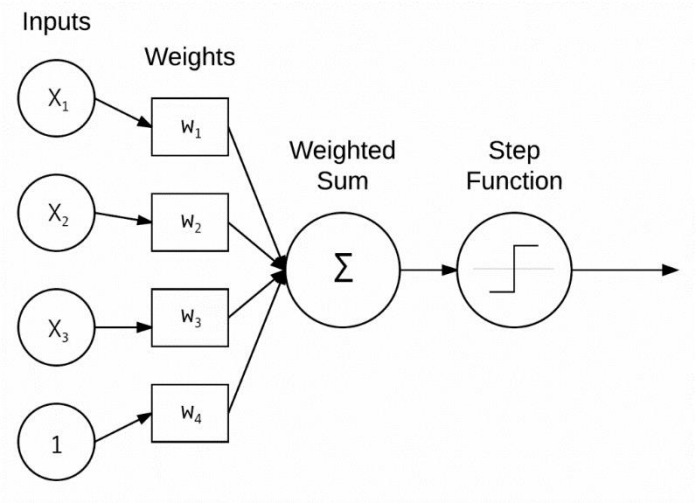


Figura 11. Arquitectura de red Perceptrón simple que acepta una serie de entradas, calcula una suma ponderada y aplica una función de paso para obtener la predicción final.

Fuente: Adaptado de (Vasilev et al., 2019).

En la Figura 11, se puede notar lo siguiente:

- Vector de entrada: el vector de características que se alimenta a la neurona. Por lo general, se denota con una X mayúscula para representar un vector de entradas ($X_1, X_2\ y\ X_3$).
- Vector de pesos: a cada X_1, se le asigna un valor de peso w_i, que representa su importancia. El peso b es un valor especial llamado sesgo de entrada siempre 1.
- Funciones de la neurona: los cálculos realizados dentro de la neurona para modular las señales de entrada - suma ponderada $\sum$ y función de activación por pasos F.
- Salida: la salida se controla mediante el tipo de función de activación que elija para su red. Para la función de paso, la salida es 0 o 1. Otras funciones de activación producen salida de probabilidad o números flotantes. El nodo de salida representa la predicción de perceptrón (Elgendy, 2020; Liljeqvist, 2016; Vasilev et al., 2019).

2.4.1.1.1. Función de Suma Ponderada.

También conocido como combinación lineal. Es la suma de todas las entradas x_i multiplicadas por sus pesos w_i y luego agregadas a un término de sesgo b. La cantidad de variables de entrada se representa con n, mientras que i representa una iteración. Esta función produce una línea recta representada en la Ecuación 3:

$$z = \sum_{i}^{n} xi \cdot wi + b\ (sesgo)$$

$$z = x1 \cdot w1 + x2 \cdot w2 + x3 \cdot w3 + \cdots + xn \cdot wn + b$$

Ecuación 3. Ecuación suma ponderada que produce una línea resta.

Fuente: Adaptado de (Elgendy, 2020).

2.4.1.1.2. Sesgo.

Se define como una intercepción añadida en una ecuación lineal. Es un parámetro adicional en la red neuronal que se utiliza para ajustar la salida junto con la suma ponderada de las entradas a la neurona. Una forma más sencilla de entender el sesgo es a través de una constante de una función lineal como la Ecuación 4:

$$y = mx + b$$

Ecuación 4. Ecuación de la recta.

Fuente: Adaptado de (Elgendy, 2019)

donde b es la intersección en y. Para poder definir una línea lineal, se necesita dos cosas: la pendiente de la línea y un punto en esa línea.

En la Figura 12, el sesgo es el punto en el eje y, con esto es posible mover la línea hacia arriba y hacia abajo en el eje y para ajustar mejor la predicción con los datos. Sin el sesgo (b), la línea siempre debe pasar por el punto de origen $(0, 0)$ y se obtendrá un poder de representación limitado (Vasilev et al., 2019).

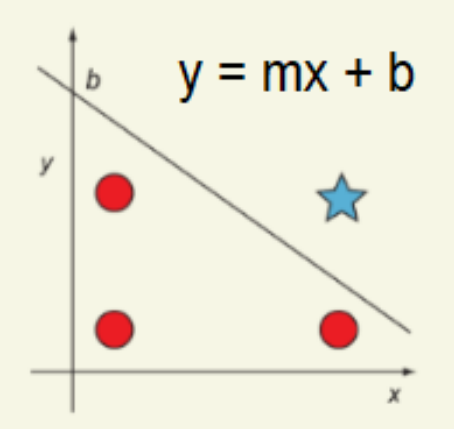


Figura 12. Representación de una función en el hiperplano.

Fuente: Adaptado de (Elgendy, 2019)

2.4.1.1.3. Función de Activación por Pasos.

La función de activación es la encargada de la toma de decisiones del cerebro. La función de activación toma la misma entrada de suma ponderada, Ecuación 3, y activa (dispara) la neurona si la suma ponderada es superior a un cierto umbral.

En la Figura 13, se muestra la función de activación más simple utilizada por el algoritmo perceptrón, "función de paso", esta produce una salida binaria (0 o 1). Básicamente, si la entrada sumada es ≥ 0, entonces se "dispara" (salida = 1). De lo contrario (suma de entrada <0) no se dispara (salida = 0) (Nagyfi, 2018).

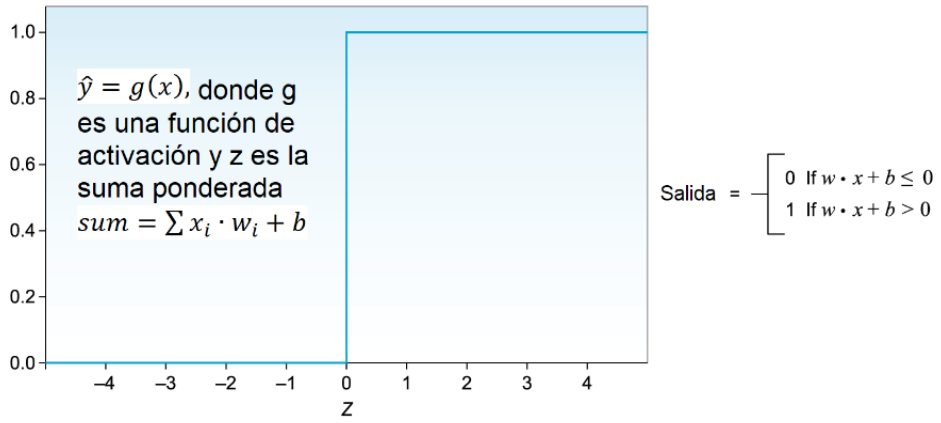


Figura 13. Función de activación para perceptrones (Función de paso, que generara 0 o 1).

Fuente: Adaptado de (Mueller & Massaron, 2016).

2.4.1.2. Proceso de aprendizaje del Perceptrón.

La neurona usa prueba y error para aprender de sus errores. Utiliza los pesos como botones al ajustar sus valores hacia arriba y hacia abajo hasta que la red esté entrenada. La lógica de aprendizaje del perceptrón se describe a continuación (Géron, 2019):

- Se calcula la suma ponderada y aplica la función de activación para hacer una predicción $\hat{y}$. Esto se llama proceso de avance y se representa con la Ecuación 5, donde

n es la cantidad de variables de entrada, x_i denota el vector de entrada para la iteración i, w_i denota el vector de peso para la iteración i y b que es el sesgo.

$$\hat{y} = activacion\,(\sum_{i=1}^{n} x_1 \cdot w_1 + b)$$

Ecuación 5. Suma Ponderada más función de activación.

Fuente: Adaptado de (Elgendy, 2020)

- Luego, en la Ecuación 6, se compara la predicción con la etiqueta correcta para calcular el error de tipo prueba y error, el cual es un método heurístico para obtener conocimiento. Consiste en probar una alternativa y verificar si funciona. Si es así, se tiene una solución. En caso contrario se intenta una alternativa diferente.

$$error = y - \hat{y}$$

Ecuación 6. Cálculo del error.

Fuente: Adaptado de (Elgendy, 2020)

- Se actualiza el peso: si la predicción es demasiado alta, se ajustan los pesos para hacer una predicción más baja la próxima vez y viceversa (Elgendy, 2020)

Este proceso se repite muchas veces, y la neurona continúa actualizando los pesos para mejorar sus predicciones hasta que el paso 2 produce un error muy pequeño cercano a cero. Lo que significa que la predicción de la neurona está muy cerca del valor correcto (Elgendy, 2020; Mueller & Massaron, 2016)

2.4.1.3. Perceptrón Multicapa (MLP)

Un solo perceptrón puede funcionar eficientemente con conjuntos de datos simples que se pueden separar por una línea lineal. Para dividir un conjunto de datos no lineal, se necesita una arquitectura para usar decenas y cientos de neuronas en una red neuronal.

2.4.1.3.1. Arquitectura de Perceptrón Multicapa.

Una arquitectura de red neuronal muy común es apilar las neuronas en capas una encima de otras llamadas capas ocultas. Cada capa tiene *n* números de neuronas. Las capas están conectadas entre sí mediante conexiones de pesos. La Figura 14, detalla la arquitectura de Perceptrón de múltiples capas para el aprendizaje de características. (Géron, 2019).

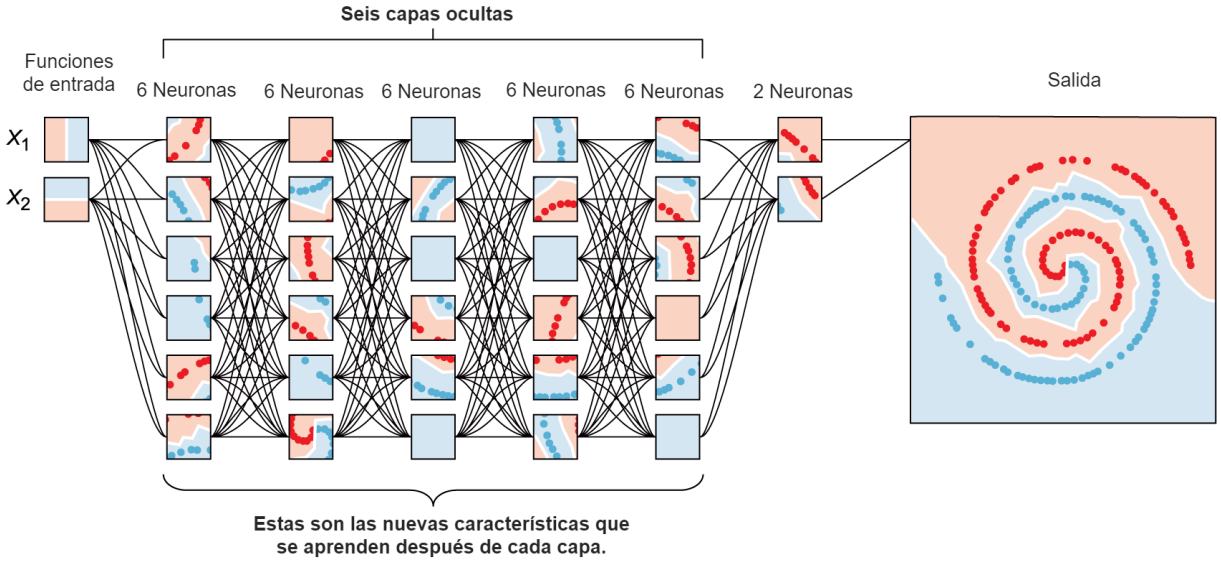


Figura 14. Red secuencial multicapa, donde X_1 y X_2 son las funciones de entrada de la red neuronal.

Fuente: Adaptado de (*A Neural Network Playground*, n.d.).

Los componentes principales de la arquitectura de red neuronal son:

- Capa de entrada: contiene el vector de características

- Capas ocultas: las neuronas se apilan unas encima de otras en capas llamadas capas ocultas. Denominadas así, porque, no se controla la entrada ni la salida de estas.

- Conexiones de peso (bordes): se asignan pesos a cada conexión entre los nodos para reflejar la importancia de este nodo (característica) en la predicción de salida final.

- Capa de salida: Dependiendo de la configuración de la red neuronal, la salida final puede ser una salida de valor real (problema de regresión) o un conjunto de probabilidades (problema de clasificación). Esto está controlado por el tipo de función de activación que se use en las neuronas en la capa de salida (Elgendy, 2020; Vasilev et al., 2019)

2.4.1.3.2. Capas Ocultas.

Las primeras capas detectan patrones simples en el entrenamiento para aprender características de bajo nivel, las siguientes capas detectan patrones dentro de patrones para aprender características y formas más complejas, y así sucesivamente. Por lo general, cuando se tiene 2 o más capas ocultas, se denomina una red neuronal profunda. La regla general es: cuanto más profunda es la red, más se ajustará a los datos de entrenamiento. Lo que no siempre es bueno, porque la red puede ajustarse demasiado a los datos de entrenamiento que no se puede generalizar cuando se le muestra datos nuevos (sobreajuste) y también se vuelve más costoso computacionalmente. Entonces, lo ideal es construir una red que no sea demasiado simple (una neurona) y que no sea demasiado compleja

2.5. Redes Neuronales Convolucionales (CNN)

En el aprendizaje profundo, una red neuronal convolucional (CNN) es una clase de redes neuronales profundas, más comúnmente aplicadas para analizar imágenes visuales. Se utiliza una

técnica especial llamada Convolución, que es un operador matemático que transforma dos funciones en una tercera función que expresa cómo la forma de una es modificada por la otra. En conclusión, el papel de una CNN es reducir las imágenes en un formulario que es más fácil de procesar, sin perder características que son críticas para obtener una buena predicción.

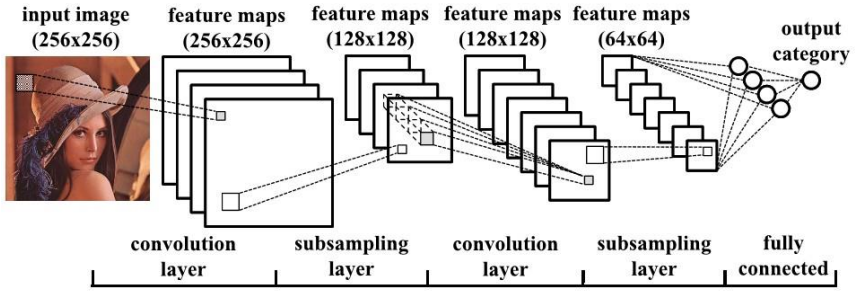


Figura 15. Diferentes capas en una CNN

Fuente: Adaptado de (Parekh, 2019).

Para la Figura 15:

- Primero se alimenta la imagen en bruto a las capas convolucionales.
- La imagen pasa a través de las capas CNN para detectar patrones y extraer características llamadas mapas de características. La salida de este paso se aplana a un vector de las características aprendidas de la imagen.
- El vector de características aplanadas se alimenta a las capas completamente conectadas (FC) para clasificar las características extraídas de la imagen.
- La red neuronal dispara el nodo que representa la predicción correcta de la imagen.

2.5.1.1. Capas Convolucionales (CONV)

La capa convolucional es el bloque de construcción más importante de una CNN. Consiste en un conjunto de filtros (también conocidos como núcleos, kernel o detectores de características),

donde cada filtro se aplica en todas las áreas de los datos de entrada. Un filtro se define por un conjunto de pesos aprensibles (Vasilev et al., 2019).

En matemáticas, la convolución es la operación de dos funciones para producir una tercera función modificada. En el contexto de CNN, la primera función es la imagen de entrada y la segunda función es el filtro CONV. Con algunas operaciones matemáticas se produce una imagen modificada con nuevos valores de píxeles (Elgendy, 2020).

Como se muestra en la Figura 16. Al deslizar un filtro convolucional sobre una imagen de entrada, la red divide la imagen en pequeños fragmentos y procesa esos fragmentos individualmente para ensamblar la imagen modificada que se denomina mapa de características.

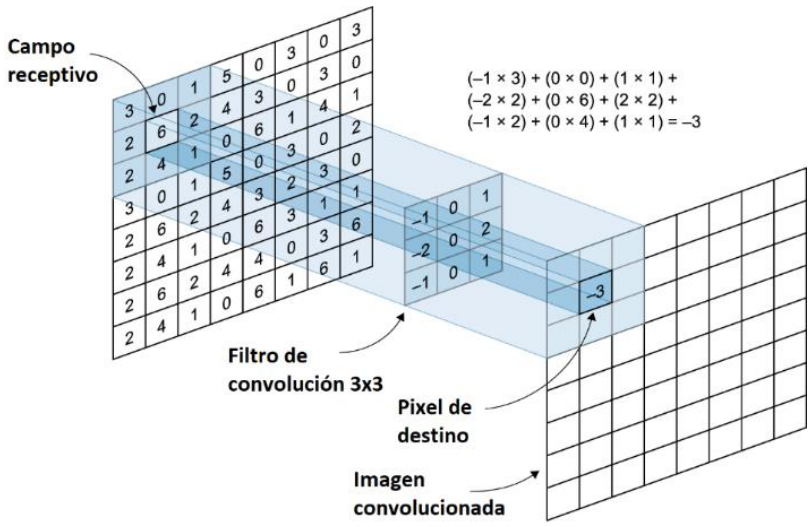


Figura 16. Operación de convolución en una matriz de imagen 8x8x1 con un núcleo de 3x3

Fuente: Adaptado de (Elgendy, 2020).

- La Figura 16 muestra una pequeña matriz de 3x3 en el medio, este es el filtro de convolución (kernels). En las CNN, la matriz de convolución son los pesos. Esto significa que también se inicializan aleatoriamente y la red aprende los valores.

- En la Figura 17 el núcleo se desliza sobre la imagen original píxel por píxel para obtener los valores de la nueva imagen "enrevesada" en la siguiente capa. El área de la imagen que involucra el filtro se llama campo receptivo (Pedrycz & Chen, 2020; Vasilev et al., 2019).

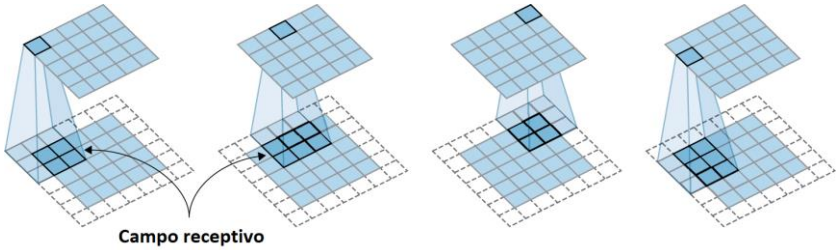


Figura 17. Cálculo de las activaciones de las siguientes posiciones.

Fuente: Adaptado de (Elgendy, 2020).

2.5.1.2. Agrupación de Capas o Submuestreo (POOL)

Similar a la capa convolucional, la capa de agrupación es responsable de reducir el tamaño espacial de la entidad convolucionada. Esto es para disminuir la potencia computacional requerida para procesar los datos al reducir las dimensiones. Hay dos tipos de agrupación: agrupación media (average pooling) y agrupación máxima (max pooling) (Pedrycz & Chen, 2020).

La Agrupación Máxima se encarga de encontrar el valor máximo de un píxel de una parte de la imagen cubierta por el kernel. Max Pooling también actúa como supresor de ruido, descarta las activaciones ruidosas por completo y también realiza la eliminación de ruido junto con la reducción de dimensionalidad. Por otro lado, la Agrupación Promedio devuelve el promedio de todos los valores de la parte de la imagen cubierta por el núcleo. La agrupación promedio simplemente realiza la reducción de la dimensionalidad como un mecanismo de supresión de ruido. Por lo tanto, se puede decir que la agrupación máxima funciona mucho mejor que la agrupación promedio (Mandal, 2021).

2.5.1.3. Capas Completamente Conectadas (FC)

Cuando se pasa la imagen a través del proceso de aprendizaje de características usando las capas CONV + POOL, se extrae todas las características de esta imagen y se las coloca en el largo tubo de características. Con esto se clasifica las imágenes en función de sus características utilizando perceptrones multicapa (MLP).

Los MLP funcionan muy bien en problemas de clasificación, sin embargo, se pierde mucha información valiosa al intentar extraer características de la imagen porque se debe aplanar la imagen antes de alimentarla a la red, mientras que las capas CONV pueden procesar imágenes en bruto (Sereno Rodriguez, 2017). El proceso se ejecuta de la siguiente manera:

- Vector plano de entrada: para alimentar el tubo de características al MLP para su clasificación, se aplana a un vector con las dimensiones (1, n). Por ejemplo, si el tubo de características tiene las dimensiones de 5x5x40, el vector aplanado será = (1, 1000).
- Capa oculta (FC): se agrega una o más capas de FC y cada capa tiene 1 o más neuronas.
- Capa de salida: se utiliza una función de activación. Dependiendo de la aplicación se puede usar diferentes funciones de activación.

2.6. Sistema de Visión Embebido

La miniaturización de los PC y las cámaras pequeñas de alto rendimiento han permitido diseñar sistemas de visión extremadamente compactos para una aplicación específica. Este sistema es lo que se conoce como sistema de visión embebido o embedded.

Los Sistemas Embedded para aplicaciones autónomas de visión por computadora, deben ejecutar como mínimo, una versión reducida de un sistema operativo estándar y bibliotecas estándar para modelos de aprendizaje automático. Si bien se puede programar estas capacidades

en un FPGA[1], resulta más fácil hacer esto con un Microcontrolador (MCU) o Computadora en Módulo (COM) (Peterson, 2020).

2.6.1. Estructura de un Sistema de Visión Embedded

Un sistema de visión embedded se compone de una cámara compacta conectada directamente a la placa de procesamiento por medio de una interfaz GigE (*Gigabit Ethernet*) o USB. Estos componentes se integran en un sistema de mayor tamaño.

En visión artificial, el PC es un elemento adecuado para realizar tareas generales de computación, mientras que las placas de procesamiento de un sistema de visión embedded, están diseñadas para llevar a cabo tareas específicas de una aplicación concreta. Las ventajas principales de este sistema son el tamaño compacto y liviano de sus componentes, su coste reducido y bajo consumo.

La mejor opción dependerá mucho del tipo de aplicación, el presupuesto, los modelos a ejecutar y el número de funciones que realizará el sistema. Múltiples módulos de cámara se pueden incorporar a estos sistemas para capturar imágenes y vídeo, creando un sistema integrado de visión de computadora completo para aplicaciones en AI/ML. De igual manera, el nivel de memoria integrada es crítico en las aplicaciones de visión por ordenador debido a la cantidad de datos que se adquieren en cualquier momento.

La Tabla 3, muestra algunos de los COM utilizados para la visión por computadora en sistemas Embedded con aplicaciones de AI.

[1] FPGA: (del inglés field-programmable gate array), matriz de puertas reprogramable, es un dispositivo que contiene bloques de lógica cuya interconexión y funcionalidad puede ser configurada en el momento, mediante un lenguaje especifico.

Tabla 3.

Opciones de CoM para diferentes aplicaciones de IA.

Plataforma	Ventajas	Desventajas
COM de NVIDIA Jetson	Altamente especializado para ejecutar modelos de TensorFlow integrados. Se utiliza mejor para aplicaciones de procesamiento de imágenes / video.	Más caro que un COM general. Poder excesivo para conjuntos de datos numéricos.
Google Coral Al	Tamaño reducido (Mini PCle / USB / SOM / M.2 Key). Puede integrarse fácilmente con los servicios de Google Cloud.	Utiliza una versión reducida de TensorFlow. Curva de aprendizaje más pronunciada y limitada a aplicaciones de imágenes.
Raspberry Pi Compute 3+	Excelente para la creación de prototipos, tamaño reducido. Puede escalar fácilmente a producción, fácil de agrupar.	Mejor para la clasificación general y la predicción con conjuntos de datos numéricos más pequeños. No es lo mejor para aplicaciones de visión por computadora de video en tiempo real.
Serie Toradex iMX	Conocido entre los fabricantes. Fácil de programar e integrar en una Computadora de placa única (SBC) personalizado.	
COM y SBC de Up-Board.org	Variedad de productos de CPU, GPU y FPGA para aplicaciones de IA y visión por computadora.	Costo similar al de NVIDIA con menos energía.
Inforce 6601 SoM	El potente SOM que se ejecuta en Snapdragon se puede adaptar para aplicaciones integradas de uso general.	Alto costo, no ideal para implementación masiva.

Fuente: Adaptado de (Peterson, 2020).

2.7. Reconocimiento Facial

El reconocimiento facial es un método para identificar o verificar la identidad de un individuo utilizando su rostro. Los sistemas de reconocimiento facial pueden utilizarse para identificar a personas en fotos, vídeos o en tiempo real.

En la Figura 18, se muestra el flujo intuitivo para el reconocimiento de rostros, que incluye la detección y el seguimiento de rostros, la alineación de rostros, la extracción de características y el emparejamiento.

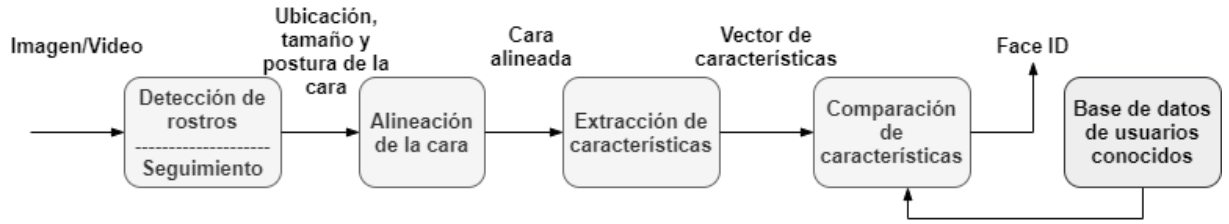


Figura 18. Línea de reconocimiento facial.

Fuente: Adaptado de (Dhingra, 2017).

2.7.1. Reconocimiento Facial en Video.

Reconocer automáticamente rostros en tiempo real a partir del video facilitará, entre otras cosas, el método encubierto de identificación humana utilizando una red existente de cámaras de vigilancia. Dos características distintivas de un video son la disponibilidad de: múltiples cuadros de los mismos sujetos e información temporal (Park, 2009).

- Múltiples cuadros aseguran una variación de poses, lo que permite una selección adecuada de un cuadro de buena calidad (por ejemplo, imagen de la cara de alta calidad en pose casi frontal) para un alto rendimiento de reconocimiento.

- La información temporal en video es considerada como la información incrustada en el movimiento facial dinámico en el video.

La dificultad del reconocimiento facial en video depende de la calidad de las imágenes faciales en términos de pose, variaciones de iluminación, oclusión y resolución. La gran cantidad de cuadros en video también aumenta la carga computacional (Park, 2009).

2.7.2. Detección de rostros

La detección de rostros y el reconocimiento facial son algoritmos claramente diferentes: la detección de rostros indica dónde está una cara (pero no a quién pertenece la cara) mientras que el reconocimiento facial identifica realmente la cara detectada.

Algunos de los enfoques de detección de rostros conocidos se pueden clasificar como: basado en color, basado en plantilla y basado en características, sin embargo, existen diversos enfoques propuestos que utilizan diferentes arquitecturas de aprendizaje profundo. A continuación, se describen las técnicas más populares y con mejores resultados que se han probado, tanto en ordenadores tradicionales, como en placas embebidas con recursos computacionales limitados:

2.7.2.1. Cascadas de Haar de OpenCV

Desarrollado por Paul Viola y Michael Jones 2001, el Marco de Detección de Objetos Viola-Jones puede detectar rápida y precisamente objetos en imágenes y funciona particularmente bien con el rostro humano (Viola & Jones, 2001). Se combina los conceptos de características similares a Haar, imágenes integrales, algoritmo AdaBoost y clasificador de cascada para crear un sistema para la detección de objetos que sea rápido y preciso.

2.7.2.1.1. Características de Haar.

Una característica similar a Haar consta de regiones oscuras y regiones claras, produce un único valor tomando la suma de las intensidades de las regiones de luz y restando eso por la suma de las intensidades de las regiones oscuras. En la Figura 19, se muestran algunas características Haar, las dos primeras son "operaciones de borde", utilizadas para detectar aristas. La tercera es una "entidad de línea", mientras que la cuarta es una "operación de cuatro rectángulos", probablemente utilizada para detectar una línea inclinada.

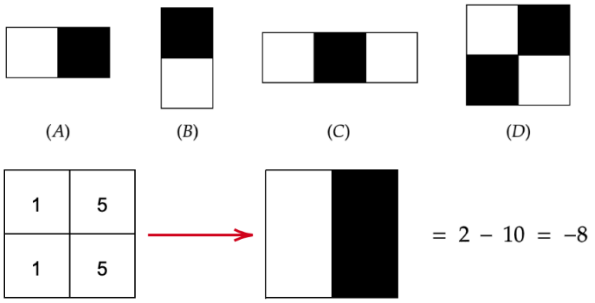


Figura 19. Características similares a Haar (arriba) y cómo calcularlas (abajo).

Fuente: Adaptado de (Lee, 2020)

2.7.2.1.2. Imagen Integral.

Una imagen integral es una representación intermedia de una imagen donde el valor de la ubicación (x, y) en la imagen integral es igual a la suma de los píxeles arriba y a la izquierda (inclusive) de la ubicación (x, y) en la imagen original (Viola & Jones, 2001). Esta representación permite un cálculo rápido de la región rectangular. En la Figura 20 se ilustra este proceso.

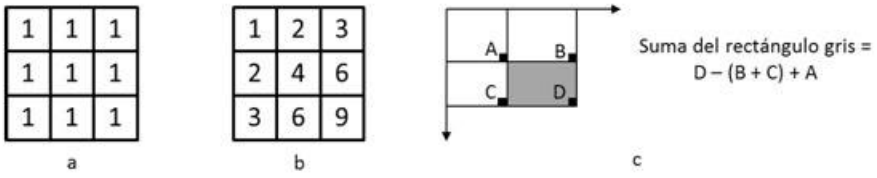


Figura 20. (a) Imagen original, (b) Imagen integral, (c) Valor del rectángulo utilizando la imagen integral.

Fuente: Adaptado de (del Toro Hernández et al., 2012)

2.7.2.1.3. Adaboost.

El algoritmo AdaBoost (Adaptive Boosting) es un algoritmo de aprendizaje automático para seleccionar el mejor subconjunto de funciones entre todas las funciones disponibles. La salida del algoritmo es un clasificador llamado "Clasificador fuerte".

Un clasificador fuerte se compone de combinaciones lineales de "clasificadores débiles" (mejores características). Desde un nivel alto, para encontrar estos clasificadores débiles, el algoritmo se ejecuta para T iteraciones donde T es el número de clasificadores débiles a encontrar

y se lo establece manualmente. En cada iteración, el algoritmo encuentra la tasa de error para todas las características y luego elige el rasgo con la tasa de error más baja para esa iteración (Lee, 2020).

2.7.2.1.4. Clasificador en Cascada.

Cada etapa consiste en un clasificador fuerte producido por AdaBoost. De una etapa a otra, aumenta el número de clasificadores débiles en un clasificador fuerte. Una entrada se evalúa etapa por etapa. Si un clasificador para una etapa específica genera un resultado negativo, la entrada se descarta. Si la salida es positiva, la entrada se reenvía a la siguiente etapa. Este enfoque de varias etapas permite la construcción de clasificadores más simples que pueden utilizarse para rechazar rápidamente la mayoría de las entradas negativas (no faciales) y dedicar más tiempo a las entradas positivas (faciales). En la Figura 21, se ilustra este procedimiento.

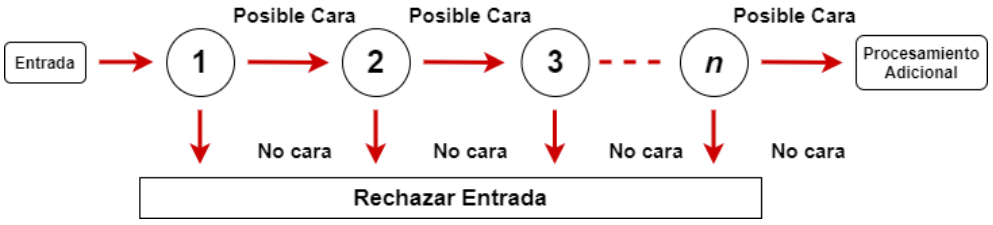


Figura 21. Un diagrama de flujo de clasificadores en cascada. (Fuente de imagen)

Fuente: Adaptado de (Mittal, 2020)

2.7.2.2. Detección facial basada en aprendizaje profundo.

El detector de rostros de aprendizaje profundo de OpenCV se basa en el marco de "Detección de caja múltiple de disparo único" (SSD) con una red base ResNet-10. La red SSD se denomina disparo único, ya que tanto la localización como la clasificación de objetos se realizan dentro de un único avance a través de la red. Además, la red SSD combina múltiples mapas de características con diferentes tamaños para generar predicciones, estas predicciones combinadas con los mapas de características múltiples producen dos salidas, un desplazamiento del cuadro delimitador y una confianza de clase. (Thuis, 2018).

2.7.2.2.1. Arquitectura de SSD.

La red consta de tres partes, una red base, capas SSD y capas de predicción adjuntas a múltiples mapas de características en la red. En la Figura 22, se muestra esta arquitectura.

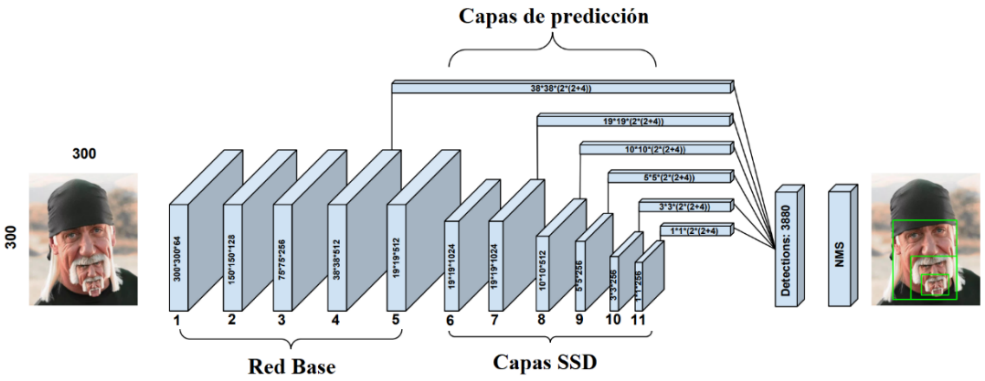


Figura 22. Arquitectura de red SSD.

Fuente: Adaptado de (Thuis, 2018)

La red base consta de convoluciones apiladas con tamaño decreciente. El propósito de esta red base es proporcionar mapas de respuesta que permitan detecciones en diferentes tamaños.

Posteriormente, se agregan capas convolucionales adicionales: conv6 hasta conv11, que se inicializan con una distribución normal truncada. Estas convoluciones adicionales se destacan como capas SSD. De manera similar a la red base, el tamaño decreciente de la función ayuda a generar mapas de respuesta para varios tamaños de objetos. Sin embargo, estas capas tienen campos receptivos más grandes y esto ayuda a detectar caras más grandes.

Las capas de predicción están unidas a la red base convolucional y las capas SSD. Para una capa de características de tamaño $m \times m \times c$, donde m es el tamaño del mapa de características y c es el número de canales. Una capa convolucional se adjunta con un kernel de $3 \times 3 \times r \times$(clases + coordenadas desplazadas), donde r es el número de cuadros delimitadores predeterminados y el número de clases es 2 (cara y fondo). Este núcleo produce una confianza de cara, una confianza

de fondo y un desplazamiento del cuadro delimitador en relación con un cuadro delimitador predeterminado. Todas las capas de predicción se concatenan al final de la red, lo que dará como resultado una única capa de salida con un número fijo de predicciones de cuadro delimitador.

Después de aplicar el modelo de detección de rostros, se obtiene el número de rostros detectados, la ubicación de sus cuadros delimitadores y la puntuación de confianza en esas predicciones (Nagrath et al., 2021).

2.7.2.3. Detección de rostros con dlib (HOG con SVM).

Los Histogramas de gradientes orientados (HOG), se utilizan generalmente en visión por computadora para detectar y reconocer objetos visuales (como rostros). Para este caso, la idea es que la apariencia y forma local de un rostro, pueda ser caracterizado por la manera en que la dirección e intensidad de la iluminación, cambian en el borde o contorno del rostro en una región específica de la imagen. Para saber el cambio en la dirección y la intensidad de iluminación en la región, se hace uso de los vectores gradientes, que describen estos cambios (Cerna et al., 2013).

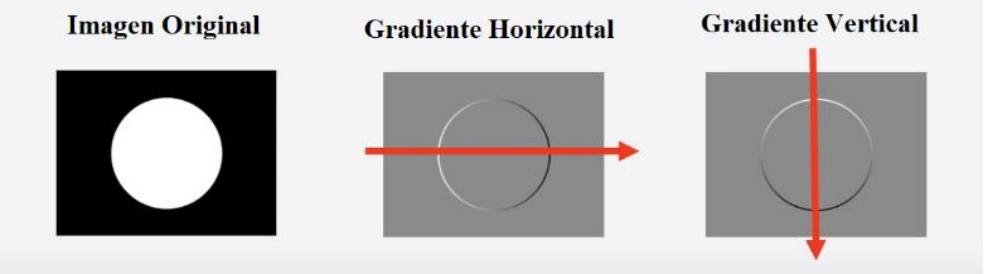


Figura 23. Degradados en la imagen.

Fuente: Adaptado de (Ahmad, 2019).

La Figura 23 muestra un cambio repentino en el valor de píxel, es decir, de un número de píxel inferior negro a un número de píxel más alto blanco. Este cambio repentino en el color se llama degradado y pasar de un tono más oscuro a un tono más claro se llama degradado positivo y viceversa. De izquierda a derecha se genera un degradado horizontal y de arriba a abajo un

degradado vertical (Ahmad, 2019). A continuación, se detalla el proceso para el cálculo de histogramas de degradados orientados:

- Se procesa previamente la imagen, incluido el cambio de tamaño para que tenga una relación de aspecto fija 1:2 (ancho: alto) y la normalización del color.
- Se calcula los gradientes horizontales y verticales de cada píxel mediante el filtrado de la imagen a través de kernels (-1 0 1), así como también su magnitud y dirección. La magnitud es la norma L2 del vector, $g = \sqrt{g_x^2 + g_y^2}$ y la dirección es el arco tangente de la relación entre las derivadas parciales en dos direcciones, $\theta = \arctan(g_y/g_x)$. La magnitud del gradiente aumenta siempre que hay un cambio brusco de intensidad.
- La imagen de degradado elimina gran cantidad de información no esencial (por ejemplo, fondo de color constante), pero resalta contornos (Weng, 2017).
- La imagen se divide en celdas. En cada celda, los valores de magnitud de estas celdas se agrupan y se agregan acumulativamente en un vector (o una matriz) de 9 bins (números) correspondientes a ángulos 0, 20, 40, 60 ... 160.
- A continuación, se desliza un bloque de NxN celdas a través de la imagen. En cada región de bloque, n histogramas de n células se concatenan en un vector unidimensional de n valores y luego se normalizan para tener una unidad de peso.

El vector de características del HOG final es la concatenación de todos los vectores de bloque. Puede introducirse en un clasificador como SVM para aprender tareas de reconocimiento de objetos. En la Figura 24 se puede observar imágenes de las distintas etapas de generación de un vector de características del histograma de gradientes orientados.

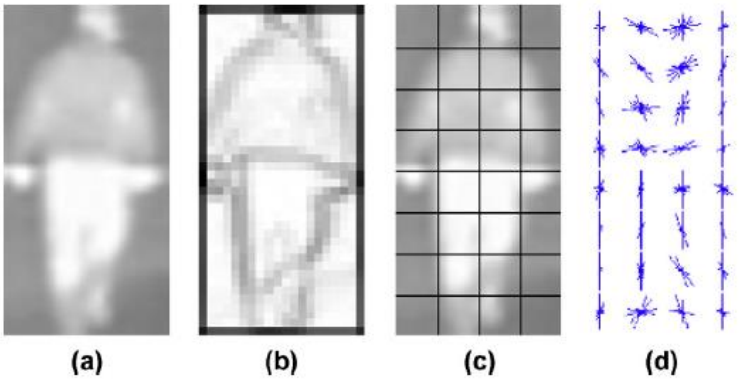


Figura 24. (a) Imagen original (20x40 píxeles), (b) imagen de gradiente, (c) imagen dividida en celdas de 5x5 píxeles, resultando en 4x8 celdas, (d) descriptor HOG resultante en cada celda.

Fuente: Adaptado de (Cerna et al., 2013).

2.7.2.3.1.1. Máquina de vectores de apoyo

La máquina de vectores de apoyo (SVM) es un modelo de aprendizaje automático supervisado que se utiliza para problemas de clasificación de dos grupos. Después de dar a un modelo SVM un conjunto de datos de entrenamiento etiquetados para cada categoría, son capaces de categorizar nuevos datos de prueba.

La SVM clasifica los datos basándose en el plano que maximiza el margen. El límite de decisión de la SVM es recto. La SVM es un algoritmo realmente bueno para la clasificación de imágenes. Los resultados experimentales muestran que las SVM logran una precisión de búsqueda significativamente mayor que los esquemas tradicionales de refinamiento de consultas después de sólo tres o cuatro rondas de retroalimentación de relevancia.

2.7.2.4. MTCNN (Multi-task Cascaded Convolutional Networks).

En MTCNN, la detección de rostros y la alineación de rostros se realizan conjuntamente, en una forma de entrenamiento de tareas múltiples. Esto permite que el modelo detecte mejor las caras que inicialmente no están alineadas.

Este modelo tiene tres redes convolucionales (P-Net, R-Net y O-Net), y es capaz de superar a muchos puntos de referencia de detección de rostros mientras conserva el rendimiento en tiempo real. En la Figura 25 se describe las arquitecturas de cada red, donde “MP” significa agrupación máxima y “Conv” significa convolución (Zhang et al., 2016).

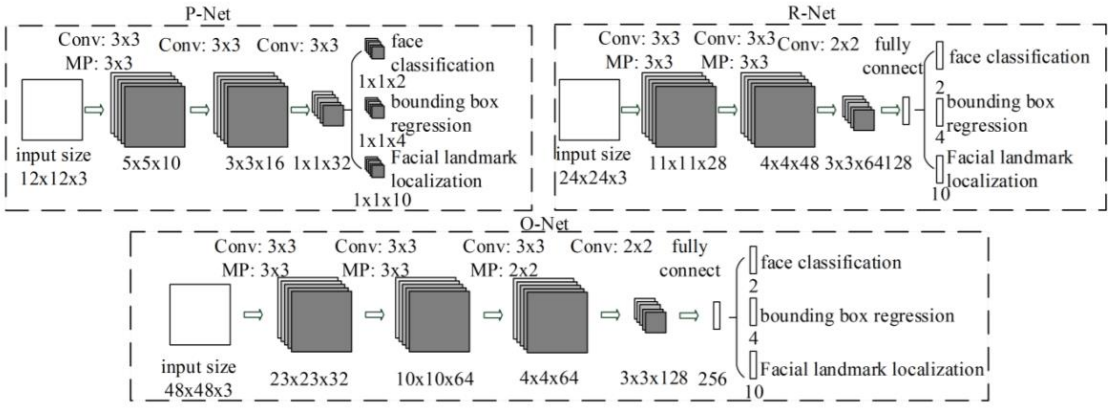


Figura 25. Estructura MTCNN.

Fuente: Adaptado de (Zhang et al., 2016).

En la entrada del sistema se ingresa una imagen que se redimensiona en diferentes escalas, dando lugar a una pirámide de imágenes como se muestra en la Figura 26.

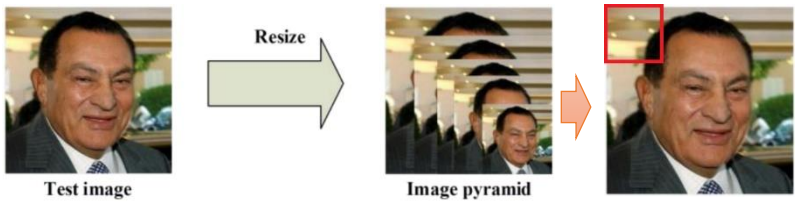


Figura 26. Pirámide de imagen.

Fuente: Adaptado de (Zhang, Li, & Qiao, 2016).

Las imágenes se procesan en la primera etapa de la red llamada P-Net, en la cual, para cada imagen escalada, un núcleo de 12x12 recorre la imagen, buscando una cara. A continuación, el cuadrado rojo representa el núcleo, que se mueve a través de la imagen, buscando una cara. Los datos obtenidos a la salida de la P-Net son los que alimentarán la segunda etapa R-Net. Finalmente,

la etapa nombrada O-Net utiliza como entrada los datos de salida de la R-Net. Las tres etapas mencionadas anteriormente se detallarán a continuación (Zhang et al., 2016).

- **Etapa 1:** P-Net (*Proposal Network*), retorna los recuadros con una probabilidad de contener un rostro dentro del umbral ($0{,}65 \leq \text{umbral} \leq 1$) establecido en la red. Además, se presentan los puntos correspondientes a los vértices del cuadro que delimita la ventana clasificada como rostro y se aplica supresión no máxima (NMS) para minimizar el número de cuadros sobrepuestos en un mismo rostro.

- **Etapa 2:** R-Net (*Refine Network*) adquiere los datos de la P-Net para disminuir la cantidad de recuadros clasificados erróneamente mediante regresión de cuadro delimitador y NMS. Las muestras para el entrenamiento de esta red se obtienen de la misma manera que en la P-Net, incluyendo sus falsos positivos, a través de un procedimiento denominado *hard sample mining* para mejorar la precisión del modelo.

- **Etapa 3:** O-Net (*Output Network*), permite detallar el rostro mediante sus puntos faciales; añadiendo los 5 puntos principales de un rostro a los resultados obtenidos en las etapas anteriores. Las muestras son obtenidas de la misma manera que en la R-Net.

2.7.2.4.1. Entrenamiento.

Se ejecutan tres tareas para entrenar los detectores CNN: clasificación rostro/no rostro, regresión de cuadro delimitador y localización de puntos de referencia faciales.

- **Clasificación de la cara:** el objetivo de aprendizaje se formula como un problema de clasificación de dos clases. Para cada muestra x_i, se usa la pérdida de entropía cruzada (*Cross Entropy*), dada por la Ecuación 7, donde p_i es la probabilidad producida por la

red que indica que la muestra x_i es una cara. La notación $y_i^{det} \in \{0,1\}$ denota la etiqueta de verdad fundamental (*ground thruth*).

$$L_i^{det} = -(y_i^{det} \log(p_i) + (1 - y_i^{det})(1 - \log(p_i)))$$

Ecuación 7. Ecuación Pérdida de entropía cruzada.

Fuente: Adaptado de (Zhang et al., 2016).

- **Regresión del cuadro delimitador:** Para cada ventana candidata, se predice el desplazamiento entre este y la verdad del terreno más cercana (es decir, los cuadros delimitadores de la parte superior izquierda, alto y ancho). El objetivo de aprendizaje se formula como un problema de regresión, y se emplea la pérdida euclidiana para cada muestra x_i, tal y como se muestra en la Ecuación 8, donde $\hat{y}_i^{box}$ es el objetivo de regresión obtenido de la red, y y_i^{box} es la coordenada de la verdad del terreno. Hay cuatro coordenadas, que incluyen la parte superior izquierda, la altura y el ancho, y por lo tanto $y_i^{box} \in \mathbb{R}^4$.

$$L_i^{box} = \left\| \hat{y}_i^{box} - y_i^{box} \right\|_2^2$$

Ecuación 8. Pérdida euclidiana para cada muestra xi.

Fuente: Adaptado de (Zhang et al., 2016).

- **Localización de puntos faciales:** la detección de puntos faciales se formula como un problema de regresión y se minimiza la pérdida euclidiana, véase Ecuación 9, donde $\hat{y}_i^{landmark}$ son las coordenadas del punto facial obtenidas de la red, y $y_i^{landmark}$ es la coordenada de la verdad fundamental para la muestra i-ésima. Hay cinco puntos de referencia faciales, que incluyen: el ojo izquierdo, el ojo derecho, la nariz, la esquina de la boca izquierda y la esquina de la boca derecha, y así $y_i^{landmark} \in \mathbb{R}^{10}$.

$$L_i^{landmark} = \left\| \hat{y}_i^{landmark} - y_i^{landmark} \right\|_2^2$$

Ecuación 9. Reducción de la pérdida euclidiana

Fuente: Adaptado de (Zhang et al., 2016).

Finalmente, hay que mencionar que todo el proceso de aprendizaje se realiza empleando el algoritmo de optimización de descenso de gradiente estocástico (SGD) (Zhang et al., 2016).

2.7.3. Métodos Reconocimiento Facial

El reconocimiento facial es una técnica de reconocimiento utilizada para detectar rostros de individuos cuyas imágenes se guardan en el conjunto de datos. A pesar de que otros métodos de identificación pueden ser más precisos, el reconocimiento facial siempre ha sido un foco importante de investigación debido a su naturaleza no intromisión y porque es el método fácil de identificación personal de las personas.

2.7.3.1. Eigenfaces

El algoritmo Eigenfaces utiliza el análisis de componentes principales para construir una representación de baja dimensión de las imágenes de la cara. Este proceso implica la recopilación de un conjunto de datos de N imágenes de caras de cada persona que se quiere identificar.

Dado este conjunto de datos de imágenes de rostros, que se supone que tiene el mismo ancho, alto e idealmente, con los ojos y estructuras faciales alineados en las mismas coordenadas (x, y), se aplica una descomposición de valores propios del conjunto de datos, manteniendo los vectores propios con los valores propios correspondientes más grandes. Dados estos vectores propios, una cara puede ser representada como una combinación lineal (Sirovich & Kirby, n.d.). En la Figura 27, se puede apreciar este proceso.

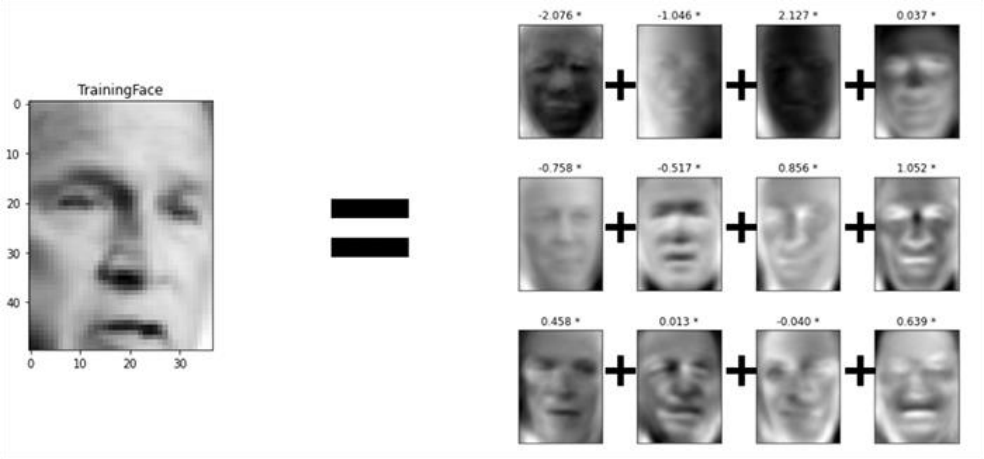


Figura 27. El algoritmo Eigenfaces utilizado para el reconocimiento facial.

Fuente: Adaptado de (Turk & Pentland, 1991)

La identificación de rostros puede realizarse calculando la distancia euclidiana entre las representaciones de las caras propias y tratando la identificación de rostros como un problema de clasificación k-Nearest Neighbor; sin embargo, se suele aplicar algoritmos de aprendizaje automático más avanzados a las representaciones de las caras propias.

2.7.3.2. LBPs para reconocimiento facial

El algoritmo LBP es conocido como un buen descriptor de texturas a nivel local, utilizado para el reconocimiento de patrones (Domínguez Pavón, 2017). Además, se ha determinado que cuando LBP se combina con histogramas de descriptores de gradientes orientados (HOG), mejora considerablemente el rendimiento de detección en algunos conjuntos de datos.

LBP etiqueta cada píxel de la imagen de acuerdo con los valores de sus píxeles vecinos. Para esto, se define un grado de vecindad y se les da el valor de 1 o 0 a estos píxeles según su nivel de intensidad sea mayor o menor que el valor del píxel central. A continuación, se recorren los vecinos y se genera una etiqueta binaria para el píxel central. Este proceso se repite sucesivamente para todos los píxeles de la imagen (Ahonen et al., 2004). La Figura 28, ilustra este proceso.

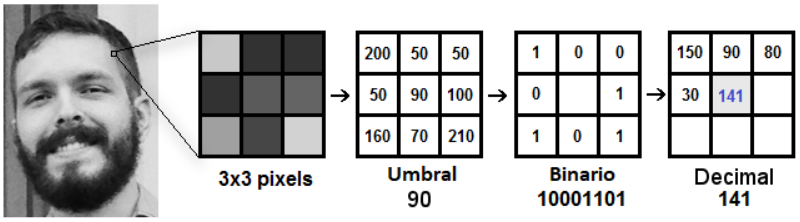


Figura 28. Procesamiento de una imagen con LBP.

Fuente: Adaptado de (Salton do Prado, 2017).

En un enfoque para el reconocimiento facial, se trabaja con la imagen del rostro dividida en regiones (sub-imágenes) como se muestra en la Figura 29. Con el operador LBP se codifica cada píxel de la sub-imagen y se recogen en un histograma regional. Posteriormente, se concatenan todos los histogramas regionales en un solo histograma global, para obtener una representación de la cara (Ahonen et al., 2004).

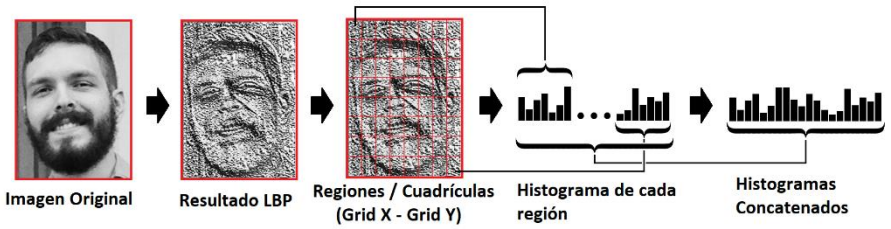


Figura 29. Extracción de los histogramas de cada región.

Fuente: Adaptado de (Salton do Prado, 2017)

2.7.3.3. Reconocimiento facial basado en el aprendizaje profundo

El aprendizaje profundo ha afectado a casi todos los aspectos y subcampos de la informática. El reconocimiento facial no es diferente. Las arquitecturas de redes neuronales especializadas y las técnicas de entrenamiento, incluidas las redes siamesas, los tripletes de imágenes y la pérdida de tripletes, permiten obtener resultados precisos a la hora de aplicar reconocimiento facial.

OpenFace es un proyecto de reconocimiento facial con redes neuronales a partir de las técnicas utilizadas en los sistemas DeepFace (Taigman et al., 2014) de Facebook y FaceNet (Schroff et al., 2015) de Google .

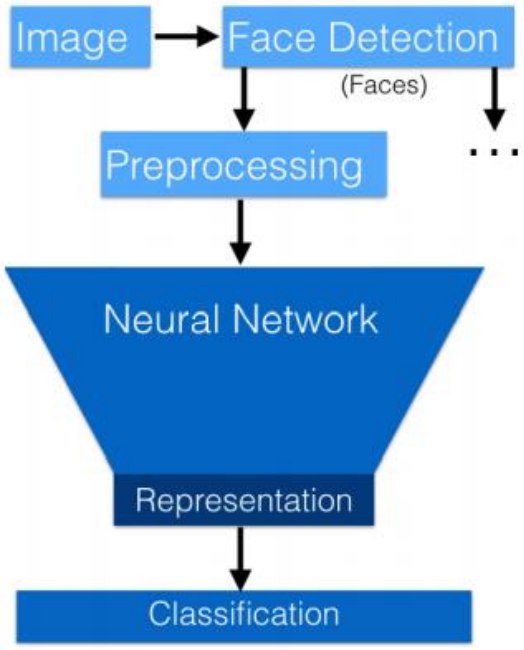


Figura 30. Flujo lógico para el reconocimiento facial con una red neuronal.

Fuente: Adaptado de (Amos et al., 2016).

En la Figura 30, se muestra el flujo lógico para el reconocimiento facial con redes neuronales. Una vez que se detecta un rostro, los sistemas preprocesan cada rostro de la imagen para crear una entrada normalizada y de tamaño fijo a la red neuronal. Las imágenes preprocesadas son de dimensiones demasiado altas para que un clasificador las tome como entrada. La red neuronal se utiliza como extractor de características para producir una representación de baja dimensión que caracteriza el rostro de una persona. Una representación de baja dimensión es clave para que pueda usarse de manera eficiente en clasificadores o técnicas de agrupamiento (Amos et al., 2016). En las siguientes secciones se detallan las técnicas utilizadas en el proyecto OpenFace.

2.7.3.3.1. DeepFace.

Es un sistema de reconocimiento facial de aprendizaje profundo creado por un grupo de investigación en Facebook. Se utiliza varias bases de datos existentes para mejorar los algoritmos y producir una salida normalizada. Sin embargo, estos modelos son insuficientes para producir un reconocimiento facial efectivo en todos los casos. DeepFace utiliza detectores de puntos fiduciales basados en bases de datos existentes para dirigir la alineación de caras. La alineación facial comienza con una alineación 2D, y luego continúa con la alineación 3D y la frontalización. Es decir, el proceso de DeepFace es de dos pasos. En primer lugar, corrige los ángulos de una imagen para que la cara en la foto esté mirando hacia adelante. Para lograr esto, utiliza un modelo 3D de una cara. Luego, el aprendizaje profundo produce una descripción numérica de la cara. Si DeepFace presenta una descripción lo suficientemente similar para dos imágenes, asume que estas dos imágenes comparten una cara (Taigman et al., 2014).

2.7.3.3.2. FaceNet.

A menudo, las aplicaciones de reconocimiento facial buscan una representación deseable de baja dimensión que se generalice bien a las caras nuevas en las que no se entrenó la red neuronal.

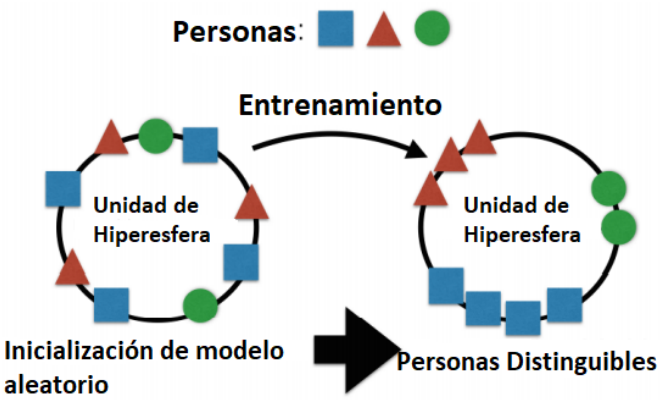


Figura 31. Ilustración del procedimiento de entrenamiento de pérdida de tripletes de FaceNet.

Fuente: Adaptado de (Schroff et al., 2015).

La Figura 31 ilustra cómo el procedimiento de entrenamiento de FaceNet aprende a agrupar representaciones faciales de la misma persona. La unidad de hiperesfera es una esfera de alta dimensión, de modo que cada punto tiene una distancia de 1 desde el origen. Restringir la incrustación a la hiperesfera de la unidad proporciona una estructura a un espacio que de otro modo no estaría delimitado. La innovación de FaceNet proviene de cuatro factores distintos: (a) la pérdida de tripletes, (b) su procedimiento de selección de tripletes, (c) entrenamiento con 100 millones a 200 millones de imágenes etiquetadas y (d) experimentación a gran escala para encontrar una arquitectura de red (Amos et al., 2016).

2.8. Metodología de Desarrollo

Una metodología para la gestión de un proyecto es esencialmente un conjunto de principios rectores y procesos para el cumplimiento de este. La elección de metodología define cómo trabaja y como se comunica.

2.8.1. Metodología en Cascada.

El modelo de cascada es una metodología de gestión de proyectos que tiene al menos cinco a siete fases que siguen en estricto orden lineal, donde una fase no puede comenzar hasta que se haya completado la fase anterior (Murugaiyan, 2012)

El método Waterfall se divide en etapas discretas. Se comienza recopilando y analizando los requisitos, diseñando la solución (y su enfoque), implementando la solución y solucionando problemas, si los hay. Gráficamente, se representa en la Figura 32:

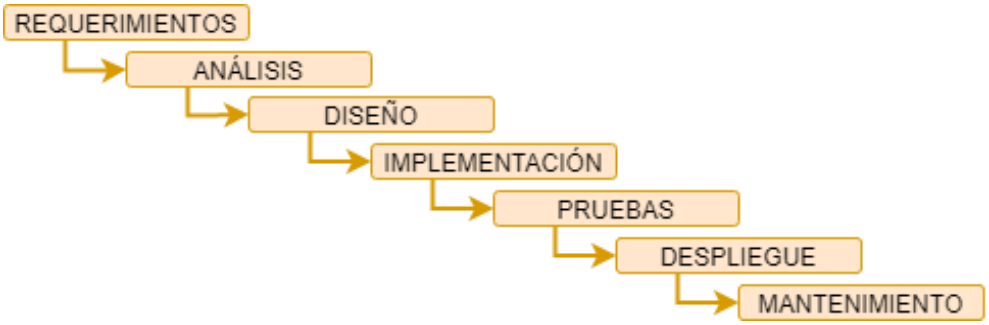


Figura 32. Diseño del modelo en Cascada.

Fuente: Adaptado de (Kumar Pal, 2019).

Los nombres específicos de las fases varían, pero fueron definidos originalmente por Royce de la siguiente manera (Kumar Pal, 2019).

- Requisitos: El aspecto clave de la cascada es que todos los requisitos del cliente se reúnen al principio del proyecto, permitiendo que cada otra fase sea planificada sin más participación del cliente hasta que el producto se haya completado.

- Diseño: La fase de diseño se divide mejor en subfases de diseño lógico y diseño físico. La subfase de diseño lógico es cuando las posibles soluciones se disponen de ideas y se teorizan. La subfase de diseño físico es cuando esas ideas teóricas y esquemas se convierten en especificaciones concretas.

- Implementación: La fase de implementación es cuando los desarrolladores asimilan los requisitos y especificaciones de las fases anteriores y producen código real.

- Verificación: Esta fase es cuando el cliente revisa el producto para asegurarse de que cumple con los requisitos establecidos al principio del proyecto. Esto se hace liberando un producto completado al cliente.

- Mantenimiento: El cliente está utilizando regularmente el producto durante la fase de mantenimiento, descubriendo errores, características inadecuadas y otros errores que se produjeron durante la producción. El equipo de producción aplica estas correcciones según sea necesario hasta que se satisfaga al cliente (Kumar Pal, 2019).

2.8.2. Modelo en V.

Desarrollado por Alan Davis, el modelo en V permite un trabajo secuencial en fases estrechamente enlazadas para el desarrollo de cualquier proyecto con su debida retroalimentación y documentación adecuada. Este modelo es una variación del modelo en cascada y es útil para proyectos que necesitan alta confiabilidad. La Figura 33 muestra el esquema del modelo en V:

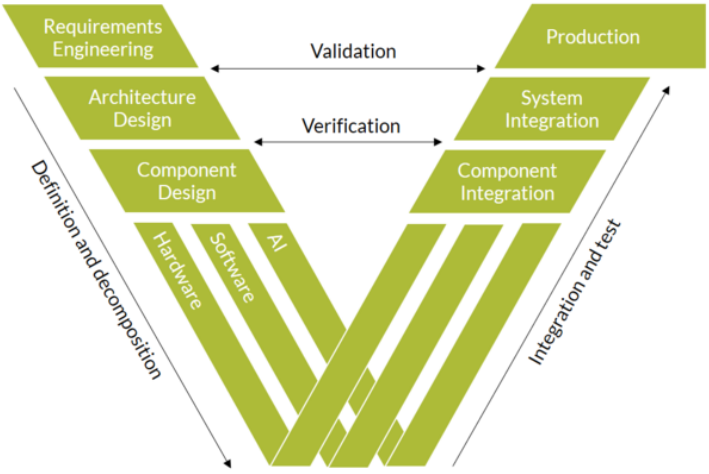


Figura 33. Etapas del modelo en V

Fuente: Adaptado de (*Verification and Validation - Www.Aiotframework.Org*, 2021)

A continuación, se detalla las funciones de cada nivel.

- Nivel 1: Orientado al cliente. Se compone del análisis de requisitos y especificaciones.
- Nivel 2: Se dedica a las características funcionales del sistema propuesto.
- Nivel 3: Define los componentes de hardware y software del sistema final.

- Nivel 4: Es la fase de implementación, en la que se desarrollan los elementos unitarios. o módulos del programa.

2.8.3. Modelo de Prototipado

La metodología de prototipos se define como un modelo de desarrollo de software en el que se construye, prueba y luego se vuelve a trabajar un prototipo cuando es necesario hasta que se logra un prototipo aceptable. También crea una base para producir el sistema final.

Es un método iterativo, de prueba y de error que tienen lugar entre el desarrollador y el cliente. En la Figura 34 se describe el esquema del modelo de prototipado.

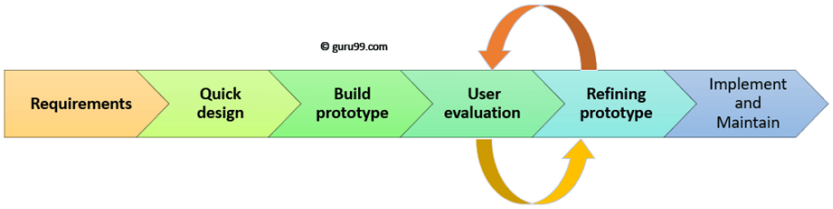


Figura 34. Fases del modelo de prototipo

Fuente: Adaptado de https://www.guru99.com/images/1/051719_0618_Prototyping1.png

Las funciones de cada etapa del modelo se describen a continuación:

- Paso 1: Recopilación y análisis de requisitos: Se definen detalladamente las necesidades del sistema. Durante el proceso, los usuarios del sistema son entrevistados para saber cuál es su expectativa del sistema.

- Paso 2: Diseño rápido: En esta etapa, se crea un diseño simple del sistema.

- Paso 3: Construir un prototipo: En esta fase, un prototipo real se diseña sobre la base de la información recopilada a partir del diseño rápido.

- Paso 4: Evaluación inicial del usuario: El sistema propuesto se presenta al cliente para una evaluación inicial. Ayuda a averiguar la fuerza y debilidad del modelo de trabajo. Las sugerencias se recopilan del cliente y se proporcionan al desarrollador.

- Paso 5: Refinar prototipo: Si el usuario no está satisfecho con el prototipo actual, debe refinar el prototipo de acuerdo con los comentarios y sugerencias del usuario.

- Paso 6: Implementar el producto y mantener: Una vez que el sistema final se desarrolla sobre la base del prototipo final, se prueba a fondo y se implementa en producción. El sistema se somete a un mantenimiento rutinario para minimizar el tiempo de inactividad y evitar errores a gran escala.

3. CAPITULO III: DISEÑO DE LA PROPUESTA

En el presente capítulo se indica el desarrollo de la propuesta del proyecto utilizando la metodología de desarrollo de software basado en el "Modelo en Cascada". Dicho modelo sugiere un enfoque secuencial, lo que permite estructurar y definir los requerimientos de cada una de las etapas del proyecto.

3.1. Metodología

El modelo que se utiliza para el desarrollo de este proyecto es el denominado Modelo en Cascada, Figura 35, el cual cumple en gran manera con el proceso de creación del proyecto, análisis, diseño, codificación, integración, pruebas y mantenimiento del software. De esta manera, se facilita la gestión del proyecto estableciendo etapas para el diseño del software y llevando a cabo los objetivos fase por fase, lo cual, es perfectamente aplicable a proyectos de pequeña escala. A continuación, se especifica a detalle cómo se desarrollará cada una de las fases.

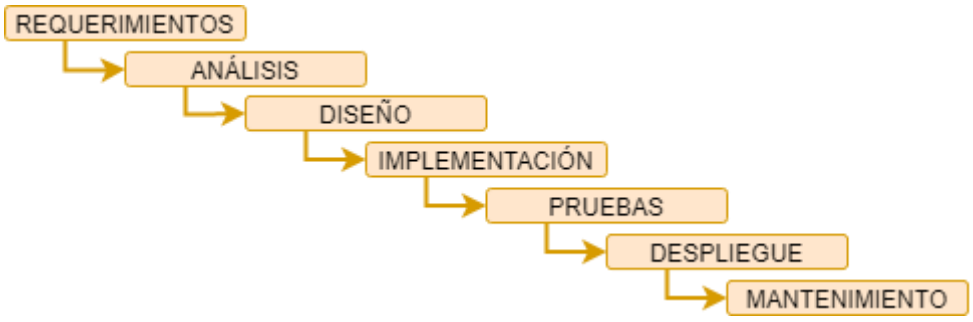


Figura 35. Modelo Lineal en Cascada.

Fuente: Adaptado de (Source Wikipedia, 2013)

- **Fase 1: Requerimientos y Análisis. -** En esta fase se une la etapa de definición de requerimientos y la etapa de análisis, se reúnen todos los requisitos que debe cumplir el software bajo las necesidades de funcionamiento del cliente.

- **Fase 2: Diseño. -** En la etapa de diseño se traducen los requisitos en una representación de software en donde se enfocan las tareas de estructura de datos, arquitectura del software, representaciones de interfaz y el detalle procedimental o algoritmo.

- **Fase 3: Implementación y Testeo. -** En esta fase se suman las etapas de implementación y pruebas, donde los requisitos y especificaciones de las fases anteriores se traducen en algoritmos comprensibles por el terminal y así obtener un programa ejecutable, esta etapa se centrará en los procesos lógicos del software, asegurando que todas las sentencias se han comprobado, y en la detección de errores mediante pruebas pertinentes.

- **Fase 4: Mantenimiento. –** Modificaciones o creación de funciones adicionales en el prototipo a lo largo de su vida útil. Etapa no considerada en este proyecto.

Debido al enfoque del proyecto dentro del campo de la Inteligencia Artificial, el impacto debe orientarse a selección de un método eficiente que logre solventar el objetivo del proyecto a través de la revisión literaria existente, mas no ofrecer una solución definitiva comercial.

3.2. Fase1: Requerimiento y Análisis

El proceso permitirá establecer los requerimientos específicos que debe tener el sistema, además esto permitirá determinar los requisitos de software y hardware. Esta sección presenta los lineamientos principales que rigen el desarrollo del sistema prototipo de reconocimiento facial; los aspectos a cubrir son los siguientes: propósito, beneficiarios y objetivos del sistema.

3.2.1. Propósito y ámbito del sistema.

Se propone desarrollar un prototipo con sistemas embebidos que actúe como Asistente Virtual de acompañamiento mediante el cual se verifique la presencia de personas que hacen uso de un Transporte Escolar a lo largo del trayecto entre Hogar - Institución Educativa y viceversa.

El sistema permitirá diferenciar los usuarios conocidos de los desconocidos dentro del recorrido, haciendo uso de técnicas de reconocimiento facial, además, para confirmar el resultado del proceso anterior se efectuará un conteo de personas que van en el transporte y la generación de alertas, todo esto con fines de seguridad hacia los ocupantes.

El proyecto en general no pretende desacreditar los procedimientos del transporte escolar ni su actuar a la hora de realizar su trabajo, más bien, intenta colaborar en la labor diaria, haciendo uso de recursos tecnológicos asequibles y poco invasivos. El objetivo fundamental en el que se sostiene gran parte del proyecto es el diseño de una solución conveniente al problema del reconocimiento facial en entornos no controlados y con recursos tecnológicos limitados, con el uso de técnicas de vanguardia en el área de la Inteligencia Artificial y la Visión por Computador.

3.2.2. Requerimientos del sistema

Se establecerán bajo el estándar IEEE 29148:2011 basándose en los Stakeholder o atributos. Son requerimientos que determinan el adecuado funcionamiento del sistema y a partir de ellos obtendrá los requisitos necesarios para desarrollar el prototipo. Estos requerimientos se conforman en tres ámbitos que son los siguientes: Requerimientos de Usuario, Requerimientos de Sistema y Requerimientos de Hardware.

3.2.2.1. Beneficiarios.

Corresponde a los beneficiarios directos e indirectos. Los beneficiarios directos constituyen los estudiantes de las Instituciones Educativas que hacen uso de un recorrido escolar y los beneficiarios indirectos comprenden a los Padres de Familia de los estudiantes, conductores de los transportes escolares, autoridades y docentes de las Instituciones. A continuación, se detallan las características de cada tipo de usuario del sistema:

- Los usuarios directos son los sujetos de prueba en la tarea de reconocimiento facial, los cuales se caracterizan por ser los usuarios frecuentes del servicio de transporte escolar.
- Los usuarios indirectos del sistema los conformaran los sujetos interesados y encargados de la seguridad de los estudiantes. Padres de Familia, Profesores, Autoridades, Conductores.

La Tabla 4 muestra los Stakeholders presentes en el sistema.

Tabla 4.

Lista de Stakeholders del sistema.

	Lista de Stakeholders
1.	Estudiantes o Usuarios del Transporte Escolar
2.	Autoridades y docentes de las Instituciones Educativas
3.	MsC. Carlos Vásquez (Director del trabajo de titulación)
4.	MsC. Jaime Michilena (Asesor del trabajo de titulación)
5.	MsC. Luis Suárez (Asesor del trabajo de titulación)
6.	Flores Ronald (Desarrollador del proyecto)

Fuente: Adaptado de (ISO/IEC/IEEE, 2018)

3.2.2.2. Nomenclatura de los Requerimientos.

A continuación, se indican la nomenclatura y consideraciones del estándar para el diseño de tablas que contengan la información relevante del proyecto, el propósito de la elaboración de

tablas es presentar la información pertinente al proyecto de una manera concisa y clara, para posteriormente realizar la selección de hardware y software a emplear.

La Tabla 5 muestra los acrónimos empleados para referirse de forma abreviada a cada requerimiento analizado.

Tabla 5.

Definición de acrónimos.

Acrónimo	Descripción
StSR	Requerimientos de Stakeholders
SySR	Requerimientos del Sistema
SRSH	Requerimientos de Arquitectura

Fuente: Adaptado de (ISO/IEC/IEEE, 2018).

El estándar permite asignar prioridades a cada uno de los requerimientos, estas son determinadas mediante criterios como: la importancia para el usuario, el rendimiento del sistema y el riesgo para el diseño y la implementación del proyecto.

La Tabla 6 describe la prioridad que se le asigna a cada requerimiento del sistema.

Tabla 6.

Prioridad de los Requerimientos del sistema.

Prioridad	Descripción
Alta	Es un requerimiento crítico que debe incluirse durante el desarrollo del sistema. Si no se implementa puede afectar la funcionalidad.
Media	El no incluir este tipo de requerimiento puede afectar la decisión final del sistema, sin embargo, se puede omitir este requerimiento en condiciones de fuerza mayor.
Baja	Si no se incluye este requerimiento no se espera un impacto significante en la decisión final del sistema.

Fuente: Adaptado de (ISO/IEC/IEEE, 2018).

3.2.2.3. Requerimientos de Stakeholders.

Se considera stakeholder a todo grupo o individuo que tiene un interés directo en el resultado alcanzado por el desarrollo del proyecto. La definición de los requerimientos del stakeholder (StSR) tiene como finalidad identificar los requisitos de los interesados para el sistema.

En la Tabla 7 se analizan específicamente los requerimientos operacionales y de usuario que tienen que ver con la interacción directa de los usuarios con el sistema.

Tabla 7.

Requerimientos de Stakeholders.

StSR					
#	REQUERIMIENTOS	PRIORIDAD			RELACION
		Alta	Media	Baja	
REQUERIMIENTOS OPERACIONALES					
StSR1	El prototipo debe permitir ser implementado dentro un vehículo.	☒			
StSR2	El sistema debe presentar bajo consumo de energía.			☒	
StSR3	El sistema debe ser lo más compacto posible.			☒	
StSR4	Adquisición de datos de entrenamiento (imágenes del rostro) para la construcción de un clasificador de aprendizaje automático.	☒			
StSR5	Se debe contar con una base de datos de rostros conocidos, ordenada de manera correcta en carpetas	☒			
StSR6	El sistema debe tener acceso a la red para el envío de notificaciones.	☒			
REQUERIMIENTO DE USUARIOS					
StSR7	La detección y reconocimiento deben ejecutarse con un mínimo de retardo.	☒			
StSR8	El sistema debe funcionar constantemente o en intervalos.		☒		
StSR9	Los usuarios directos no podrán manipular las opciones del sistema.		☒		
StSR10	Los usuarios directos deberán permanecer en su respectivo asiento durante todo el recorrido.		☒		
StSR11	Se debe capturar fotografías de varios ángulos o poses posibles de cada individuo	☒			
StSR12	Las notificaciones de alerta deben ser los más instantáneas posible.			☒	
StSR13	La base de datos debe ser privada y solo el administrador tendrá acceso	☒			
StSR14	Sistema debe contar con alimentación de respaldo.		☒		

Fuente: Adaptado de (ISO/IEC/IEEE, 2018).

Los requerimientos presentados son parte de las necesidades planteadas por los usuarios en la Observación Directa y Encuestas, brindando una idea general del producto final en cuanto al entorno operativo y la relación entre el usuario y sistema.

Como demandas más relevantes están que de ser un sistema compacto pero que brinde alta precisión y presente información verídica en tiempo real. Las encuestas y los resultados se detallan en el Anexo 1 y Anexo 2, respectivamente.

3.2.2.4. Requerimientos del Sistema.

Los requerimientos del sistema (SySR) tienen que ver con las funciones que va a realizar el sistema y todo lo que requiere para su funcionamiento. Entre estos se va a analizar requerimientos de uso, performance, interfaces, modos y estados y requerimientos físicos que guardan una estrecha relación con los requisitos de Stakeholders.

A continuación, se describen en la Tabla 8 los requerimientos iniciales del sistema (SySR).

Tabla 8.
Requerimientos Iniciales del Sistema.

SySR					
REQUERIMIENTOS INICIALES DEL SISTEMA					
#	REQUERIMIENTOS	PRIORIDAD			RELACION
		Alta	Media	Baja	
REQUERIMIENTO DE USO					
SySR1	El sistema deberá ejecutar la detección y reconocimiento de rostros en tiempo real.	☒			
SySR2	El registro de identidad debe guardarse en tiempo real.		☒		
SySR3	Reconocimiento facial para la identificación y registro de personas conocidas.	☒			
SySR4	Reconocimiento facial para la identificación y registro de personas desconocidas.		☒		
REQUERIMIENTO DE INTERFAZ					
SySR5	Pines de entrada y salida de información del sistema embebido.			☒	
SySR6	El sistema debe interactuar con una o varias cámaras conectadas mediante USB o CSI.	☒			
SySR7	El sistema deberá permitir la comunicación y el acceso remoto.		☒		
SySR8	El sistema electrónico (Sistema embebido, los sensores, la cámara y la alimentación eléctrica) deberán interactuar entre sí.	☒			
REQUERIMIENTO DE PERFORMANCE					
SySR9	El sistema debe interactuar con una GPU (Unidad de Procesamiento Gráfico).		☒		
SySR10	Detección y reconocimiento facial en un entorno controlado.	☒			
SySR11	Obtención de video en tiempo real de las cámaras del sistema.	☒			
SySR12	Taza de transferencia de datos alta.		☒		
REQUERIMIENTO DE MODOS/ESTADOS					
SySR13	El sistema debe permitir un modo activo y un modo standby.		☒		
REQUERIMIENTO FISICOS					
SySR14	El sistema debe estar situado correctamente en un lugar donde no interfiera con las actividades de las personas y esté conectado a la red.	☒			
SySR15	Las cámaras del sistema deben estar colocada en un lugar estratégico donde se enfoque directamente al ingreso de las personas y donde aspectos como la oclusión, luminosidad, y ruido en la imagen sean reducidos.		☒		
SySR16	Sensor en la puerta de entrada que detecte cuando esté abierta o cerrada			☒	

Fuente: Adaptado de (ISO/IEC/IEEE, 2018).

3.2.2.5. Requerimientos de Arquitectura.

En los requerimientos de arquitectura (SRSH) se definen los requerimientos de hardware, software y el sistema eléctrico. A continuación, las directrices descritas en los requerimientos de Arquitectura se muestran en la Tabla 9 y son necesarias para la selección del hardware y software a emplear en el proyecto.

Tabla 9.

Requerimientos de Arquitectura.

SRSH					
REQUERIMIENTOS DE ARQUITECTURA					
#	REQUERIMIENTOS	PRIORIDAD			RELACION
		Alta	Media	Baja	
REQUERIMIENTO LÓGICOS					
SRSH1	Se requiere un sistema operativo compatible con una placa embebida.		☒		SRSH21
SRSH2	Se requiere de un sistema operativo que ejecute con rapidez los hilos de procesamiento del sistema.		☒		
SRSH3	Se requiere compatibilidad de software con bibliotecas de aprendizaje automático y profundo.	☒			
REQUERIMIENTO DE DISEÑO					
SRSH4	Las cámaras deben ser instaladas lo más cerca al objetivo, pero asegurando que no sean manipuladas.		☒		
SRSH5	Las cámaras deben enfocar la mayor cantidad de rostros desde un mismo ángulo.	☒			
SRSH6	Debe diseñarse un soporte para las cámaras.			☒	
SRSH7	El conteo de personas debe realizarse en otra placa embebida para reducir el procesamiento de la placa principal.		☒		
SRSH8	Los cables de las cámaras deben ser extendidos para conectarse a la placa reducida		☒		
SRSH9	Los dispositivos servidor que alojan la solución deben estar situados en un lugar seguro y con ventilación suficiente.		☒		
REQUERIMIENTO DE HARDWARE					
SRSH10	Se requiere de un sistema embebido que permita el entrenamiento de modelos de aprendizaje automático y profundo.	☒			
SRSH11	El sistema requiere procesamiento grafico (GPU) para el procesamiento de Imágenes.	☒			
SRSH12	El sistema requiere tecnología de procesamiento de cómputo paralelo CUDA.	☒			
SRSH13	El sistema embebido debe ser compatible con Sistemas Operativos de código abierto.	☒			
SRSH14	El sistema embebido requiere más de 2 puertos USB para cámaras y demás complementos.	☒			

SRSH15	Sistema embebido con entradas y salidas análogas y/o digitales para manejo de sensores.			☒
SRSH16	Se requiere un sistema embebido asequible.		☒	
SRSH17	El sistema embebido debe permitir la compatibilidad de sensores y accesorios de otros fabricantes.		☒	
SRSH18	Se requiere cámaras compatibles con las placas embebidas.	☒		
SRSH19	Se requiere de cámaras de alta resolución para la efectividad de las tareas de identificación facial (1080p o 720p).		☒	
SRSH20	Se debe considerar que las cámaras soporte aplicaciones de reconocimiento facial.	☒		
SRSH21	Se requiere una fuente de poder mayor a 20 Watts para el abastecimiento de los dispositivos embebidos.		☒	
SRSH22	UPS compatible con los dispositivos embebidos.		☒	
	REQUERIMIENTO DE SOFTWARE			
SRSH23	Se requiere de un sistema operativo y lenguaje de programación de código abierto.	☒		
SRSH24	Se requiere compatibilidad con la librería OpenCV y la cámara.	☒		
SRSH25	Se requiere software que permita ejecutar el código de visión artificial desde un sistema embebido en tiempo real	☒		
SRSH26	Se requiere software que permita usar de manera dinámica los recursos de la GPU en la capacitación de modelos de aprendizaje automático y profundo.	☒		
SRSH27	Compatibilidad de software con las bibliotecas para aprendizaje automático.	☒		
SRSH28	Se requiere de un software o kit de herramientas de diseño de interfaces de usuario (GUI) para la visualización de resultados.		☒	
SRSH29	Se requiere compatibilidad de software con tecnología de cómputo paralelo CUDA.	☒		
SRSH30	Se requiere de un software de base de datos no relacional que permita obtener un bajo impacto o costo en el rendimiento del sistema.		☒	
	REQUERIMIENTO ELÉCTRICOS			
SRSH31	Regulador de voltaje de 12V a 5V	☒		

Fuente: Adaptado de (ISO/IEC/IEEE, 2018).

3.2.3. Descripción General del Sistema.

Un asistente virtual es un agente de software que ayuda a usuarios de sistemas computacionales, automatizando y realizando tareas con la mínima interacción hombre-máquina.

Es necesario mencionar que los recursos de procesamiento son limitados ya que se plantea usar un sistema embebido, lo que en gran medida limita la implementación unificada de las diferentes técnicas de inteligencia artificial seleccionadas para la identificación facial sobre un gran número de personas en tiempo real.

El sistema propuesto constará de un par de cámaras USB y del tipo compatibles con CSI (Camera Serial Interface); interfaz disponible en los sistemas embebidos. Dichas cámaras deben ser de tamaño reducido e inaccesibles para el usuario. La función principal del prototipo es detectar los rostros de los usuarios y reconocerlos por lo que la ubicación de las cámaras será en puntos estratégicos. El sistema embebido será capaz de soportar la carga computacional del procesamiento de las imágenes adquiridas por los dispositivos de video, además, el sistema tendrá que enviar notificaciones de alerta, por lo que es necesario una tarjeta de red y acceso a la red.

La información adquirida y generada por el prototipo tendrá que ser almacenada, por lo tanto, es indispensable la creación de una base de datos, los datos de entrada serán las capturas de los rostros de los usuarios y servirán para el entrenamiento del sistema, este procedimiento será el primero en realizarse, y la creación de una interfaz de usuario facilitará este proceso. Posteriormente esta información será procesada en un dispositivo a parte para tratar de reducir la carga computacional.

Con el sistema completo en funcionamiento, el usuario subirá al vehículo y su rostro será reconocido al inicio del trayecto, esta etapa será complementada con el conteo de las personas que

se subieron al transporte. Para asegurar la presencia del usuario durante el trayecto el sistema repetirá este proceso cada vez que la puerta de la buseta se habrá y se cierre.

Por último, el asistente virtual busca apoyar a la seguridad de los usuarios, por ende, las notificaciones se efectuarán cuando el sistema no detecte la presencia de un usuario por un tiempo prolongado o cuando el resultado del conteo de estos no sea el indicado.

3.2.4. Selección de Hardware y Software.

Para la selección de los componentes de hardware y software se realiza una tabla comparativa de especificaciones según los atributos de los requerimientos de Stakeholders, Sistemas y de Arquitectura, se evalúa un componente y mediante dicha tabla se obtiene una valoración de los atributos correspondientes (StRS, SySR, SRSH) y al final se elige al componente de mayor puntuación. Definiendo que un valor de puntuación de “1” cumple con el requerimiento y de una puntuación de “0” cuando no cumple con el requerimiento.

3.2.4.1. Hardware.

La selección del Hardware se realiza de acuerdo con los requerimientos de hardware establecidos en la Tabla 9 sobre Requerimientos de Arquitectura. Específicamente se seleccionará el sistema embebido, la placa secundaria para disminuir el procesamiento de la principal y las cámaras a emplearse para la visión artificial.

3.2.4.1.1. Sistema Embebido.

Para la elección del sistema embebido se seleccionó 5 opciones, las cuales se adaptaban a las necesidades del proyecto. El modo de valoración se realizó de la siguiente manera: si cumple con el requerimiento con un valor de 1 y si no cumple con un valor de 0. La Tabla 10 muestra la valoración de cada requerimiento para la elección del sistema embebido.

Tabla 10.

Elección del Sistema Embebido.

HARDWARE	REQUERIMIENTOS								V. Total
	SRSH10	SRSH11	SRSH12	SRSH13	SRSH14	SRSH15	SRSH16	SRSH17	
Intel NUC AI development kit	1	1	0	0	1	0	0	0	**3**
Jetson Nano Developer Kit	1	1	1	1	1	1	1	1	**8**
Jetson Xavier NX Developer Kit	1	1	1	1	1	1	0	1	**7**
Rock Pi 4C	1	1	0	1	1	1	1	1	**7**
Raspberry Pi 4	0	0	0	1	1	1	1	1	**5**
1 Cumple **0 No cumple**									

Elección: El hardware idóneo según los requerimientos especificados en la tabla de requerimientos de arquitectura es el Jetson Nano Developer Kit**.** Es una unidad compacta con capacidad de procesamiento considerable, su tamaño facilita la instalación en el interior de un vehículo, posee varios puertos USB y un CSi para accesorios y cámaras. Aunque las otras versiones de Jetson poseen características de procesamiento mayores el Jetson Nano es más asequible en cuanto al precio.

Por otro lado, se mencionó la selección de una placa auxiliar que ayudara a repartir el procesamiento del sistema, para este caso se seleccionó la Raspberry Pi 4.

Fuente: Adaptado de (ISO/IEC/IEEE, 2018).

A continuación, en la Tabla 11 se muestran las características técnicas principales de las dos placas seleccionadas.

Tabla 11.

Especificaciones técnicas del Jetson Nano Developer Kit y placa Raspberry Pi 4B.

JETSON NANO DEVELOPER KIT	
ESPECIFICACIONES	**PROPIEDADES**
GPU	128-core Maxwell
UPC	Quad-core ARM A57 @ 1.43 GHz
Memoria	4 GB 64-bit LPDDR4 25.6 GB/s
Almacenamiento	microSD
Codificación de video	4K@ 30 | 4x 1080p @ 30 | 9x 720p @ 30 (H.264/H.265)
Decodificación de video	4K@60 | 2x4K@30 | 8x1080p@30 | 18x720p@ 30 (H.264/H.265)
Cámara	2x MIPI CSI-2 DPHY lanes
Conectividad	Gigabit Ethernet, M.2 Key E
Monitor	HDMI and display port
USB	4x USB 3.0, USB 2.0 Micro-B
Otros	GPIO, I2C, I2S, SPI, UART
Mecánico	69 mm x 45 mm, 260-pin edge connector
PLACA RASPBERRY PI 4B	
Procesador:	Broadcom BCM2711, SoC de cuatro núcleos Cortex-A72 (ARM v8) de 64 bits a 1,5 GHz
Memoria:	LPDDR4 de 2GB, 4GB u 8GB (según el modelo)
Conectividad:	LAN inalámbrica IEEE 802.11b / g / n / ac de 2,4 GHz y 5,0 GHz, Bluetooth 5.0, BLE Gigabit Ethernet, 2 puertos USB 3.0, 2 puertos USB 2.0.
GPIO:	Cabecera GPIO estándar de 40 pines (totalmente compatible con las placas anteriores)
Sonido del video:	2 × puertos micro HDMI (hasta 4Kp60 compatible) Puerto de pantalla MIPI DSI de 2 carriles Puerto de cámara MIPI CSI de 2 carriles Puerto de video compuesto y audio estéreo de 4 polos
Multimedia:	H.265 (decodificación 4Kp60); H.264 (decodificación 1080p60, codificación 1080p30); OpenGL ES, 3.0 graphics
Almacenamiento:	Ranura para tarjeta microSD para cargar el sistema operativo.
Potencia de entrada:	5V CC a través del conector USB-C (mínimo 3A1) 5V CC a través del encabezado GPIO (mínimo 3A1) (PoE): habilitado (requiere PoE HAT separado)
Ambiente:	Temperatura de funcionamiento 0–50ºC

Fuente: Adaptado de (Raspberry Pi Trading Ltd, 2020), (NVIDIA Corporation, 2020)

3.2.4.1.2. Cámara.

Para la elección de la cámara se seleccionó 3 opciones, las cuales se adaptaban a las necesidades del proyecto. La Tabla 12 muestra la elección de la cámara.

Tabla 12.

Elección de Cámara.

HARDWARE	REQUERIMIENTOS			V. Total
	SRSH18	SRSH19	SRSH20	
Cámara IMX219-77	1	1	1	**3**
RPi Camera V2	1	1	0	**2**
LoveRPi	1	1	1	**3**
IMX219 Camera Module	1	1	1	**3**
IYUT HD Webcam 1080P	1	1	0	**2**
1 Cumple **0 No cumple**				

Elección: La cámara seleccionada y que cumple con los requerimientos del sistema es la Cámara IMX219-77 compatible con la placa Jetson Nano, sin embargo, este módulo no es compatible con la placa Raspberry Pi 4, por lo tanto, se seleccionó la cámara LoveRPi que es compatible con aplicaciones de Reconocimiento facial y detección de objetos.

El sistema utilizará dos cámaras una para el reconocimiento y otra para el conteo de personas, sin embargo, en la etapa de pruebas se definirá si es suficiente las dos cámaras o se añadirá una más al sistema, en este caso, se seleccionará una cámara USB ya que las interfaces CSI están ocupadas.

Fuente: Adaptado de (ISO/IEC/IEEE, 2018).

A continuación, en la Tabla 13 y Tabla 14, se muestran las características técnicas principales de las cámaras seleccionadas como primera opción.

Tabla 13.

Especificaciones técnicas de la cámara IP.

ESPECIFICACIONES	PROPIEDADES
Megapíxeles	8 Megapixels
Chip fotosensible	Sony IMX219
Resolución	3280 × 2464
Longitud focal	2.96mm
Campo de visión diagonal (FOV)	77 grados
Tasa máxima de transferencia de imágenes	30 fps for QSXGA
Formatos de salida	8/10bit RGB RAW output
Relación S / N	39db
Distancia del objeto	30CM-∞
Interfaz	15p-1.0 mipi
Fuente de alimentación	3.3V (pin15)
Temperatura de funcionamiento	-20°C to 70°C
Temperatura (imagen estable)	0°C to 50°C
Tamaño de la lente	6.5mm × 6.5mm
Tamaño del módulo	25mm × 24mm ×5.3±0.2mm

Fuente: Adaptado de (Seeed Technology Co.,Ltd., 2020)

Tabla 14.

Especificaciones técnicas de la cámara IP.

ESPECIFICACIONES	PROPIEDADES
Megapíxeles	5 Megapíxeles
Tamaño de píxel	1.4µm x1.4µm
Resolución	2592 × 1944
Filtro IR integrado:	La luz infrarroja es invisible a simple vista, pero provocará cambios de tono en los sensores de la cámara CMOS.
Controles automáticos de imagen:	Ajuste de manera inteligente el balance de blancos; la exposición, la sensibilidad, niveles de negro para optimizarlos para obtener la mejor calidad de imagen. Anti-parpadeo de 50Hz / 60Hz; Corrección de lente y cancelación de píxeles defectuosos.
Instalación	Cables FPC para Raspberry Pi 4, Pi 3, Pi 2, Pi B +. Adaptador de cable PFC para Raspberry Pi Zero, Pi Zero W y Pi 0WH.

Fuente: Adaptado de (Amazon.com, Inc., 2020)

3.2.4.2. Software.

Una vez seleccionado el hardware se procede a la selección del software, el cual se realiza en base a los requerimientos de software establecidos en la Tabla 9 sobre Requerimientos de Arquitectura.

En esta sección se consideran algunas alternativas en cuanto a la codificación del sistema sobre una plataforma de programación adecuada y bien documentada. Debido a que la naturaleza de este proyecto se centra en capacitación e implementación de modelos de aprendizaje automático y profundo casi en su totalidad, es crucial seleccionar con el mayor cuidado posible, ya que, al ser una investigación en constante avance dentro del campo de la inteligencia artificial algunas plataformas conllevan mayor dificultad de implementación que en otras y son compatibles con tecnología de procesamiento paralelo. Además, debe tener compatibilidad con las bibliotecas de visión artificial y OpenCV.

3.2.4.2.1. Software de programación.

Para la selección del software de programación se seleccionaron 4 opciones, las cuales se adaptan parcialmente a los requerimientos del proyecto para su codificación.

La Tabla 15 muestra la valoración de cada requerimiento para la elección del software de programación.

Tabla 15.

Elección del software de programación.

HARDWARE	REQUERIMIENTOS							V. Total
	SRSH23	SRSH24	SRSH25	SRSH26	SRSH27	SRSH28	SRSH29	
Python	1	1	1	1	1	1	1	**7**
C++	1	0	1	1	1	1	1	**6**
JavaScript	1	1	1	0	0	1	0	**5**
Swift	1	0	0	0	0	1	0	**2**
1 Cumple **0 No cumple**								

Elección: En la selección del software idóneo según los requerimientos del software especificados en la tabla de requerimientos de arquitectura la mayor valoración fue obtenida por Python. Este software de programación es altamente ideal para el diseño de aplicaciones de inteligencia artificial, ya que tiene compatibilidad con las más relevantes bibliotecas de aprendizaje automático y profundo (Tensorflow/Scikit-Learn/Keras) utilizadas en este proyecto.

Fuente: Adaptado de (ISO/IEC/IEEE, 2018).

3.3. Fase 2: Diseño

Una vez definidos los componentes y requerimientos necesarios para el sistema, se procede a delimitar las directrices del diseño del producto, tomando en cuenta los criterios analizados en las etapas anteriores, los cuales permitirán el desarrollo y ejecución del Asistente Virtual.

En esta fase, se detalla las funciones del sistema mediante diagrama de bloques y diagrama de flujo, los cuales brindaran una guía ordenada a través de todos los procesos a efectuar en la codificación y posterior ejecución.

3.3.1. Diagrama de Bloques General del Sistema.

Esta sección muestra mediante diagramas de bloques, el diseño general del sistema. El sistema se lo ha dividido en 4 etapas. Cada etapa engloba varios subprocesos afines a la función específica de cada bloque.

En la Figura 36, se muestra la arquitectura del Asistente Virtual a modo de diagrama de bloques general; de esta manera se obtiene una mejor comprensión de las funciones llevadas a cabo en las 4 etapas mencionadas, las cuales se abordarán a detalle en las posteriores secciones.

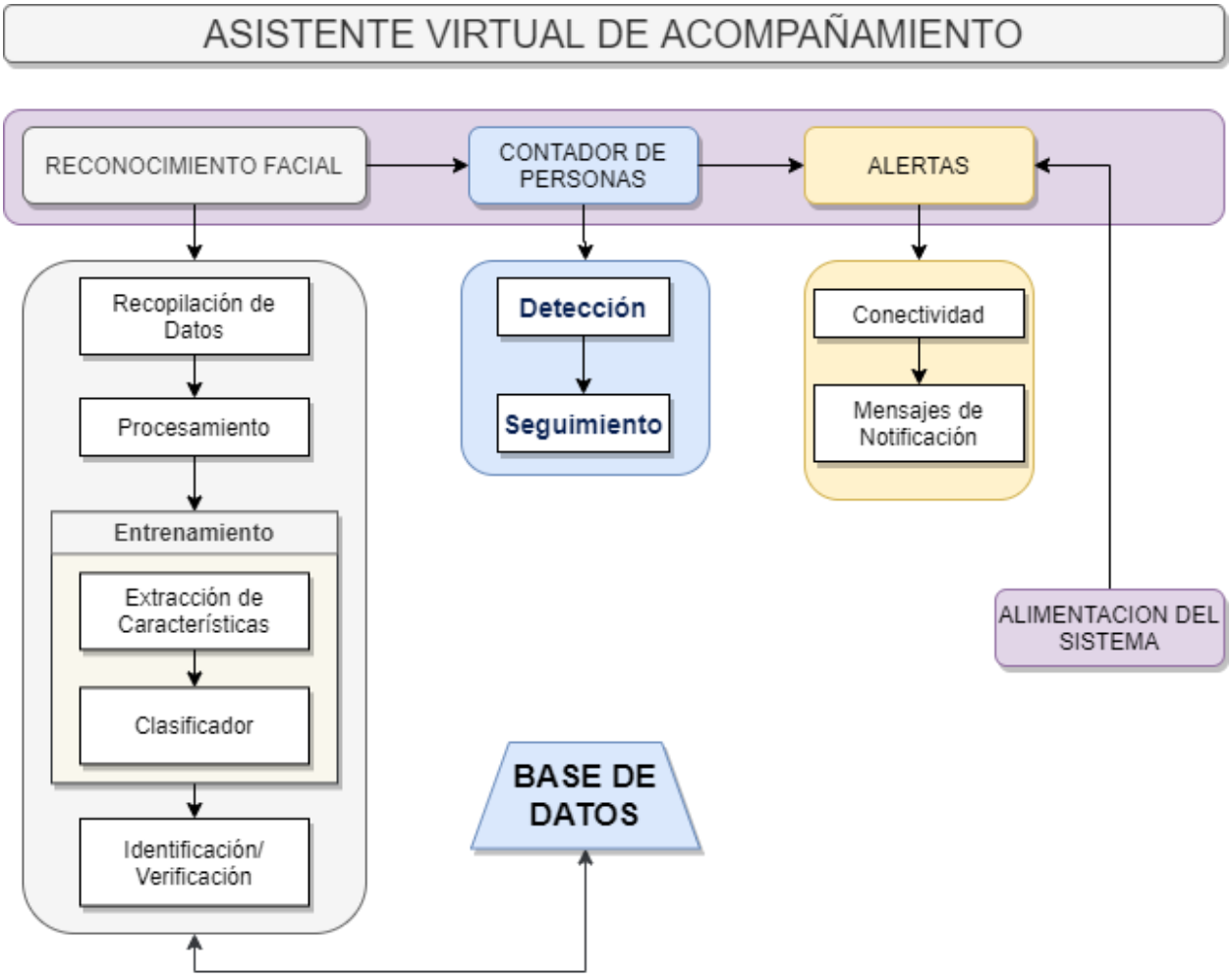


Figura 36. Diagrama de Bloques General del Sistema.

Fuente: Autoría.

3.3.1.1. Diagrama de bloques módulo Reconocimiento Facial.

Se define como la etapa más importante y de más consumo de recursos, está conformada por varios bloques, los cuales serán explicados a continuación. En la Figura 37 se puede observar el diagrama de bloques de la primera etapa.

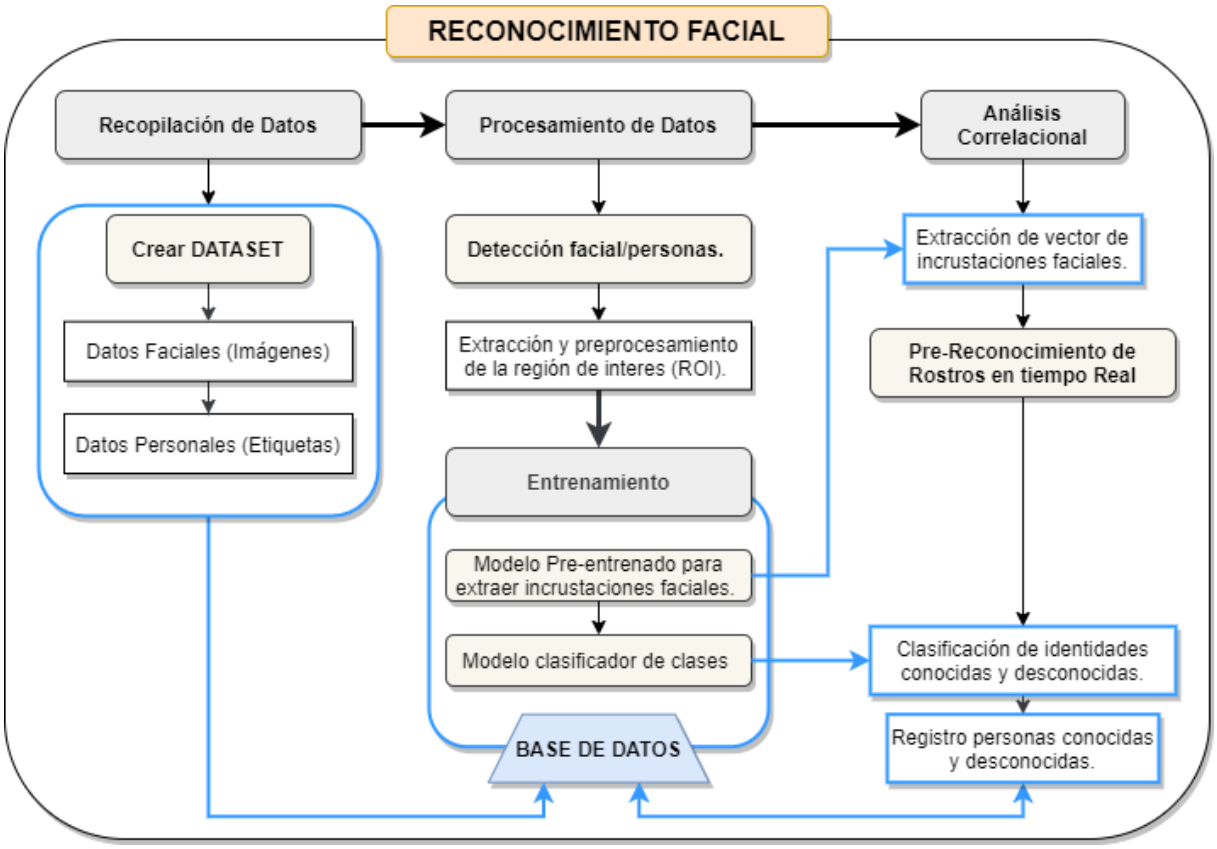


Figura 37. Diagrama de Bloques del Reconocimiento Facial.

Fuente: Autoría.

- **Bloque 1: Recopilación de datos**. - Para lograr la mayor precisión el conjunto de datos debe contener suficientes muestras de variación y una buena representación de la cara. Si el conjunto de datos tiene menos datos para entrenar, se produce un desajuste; por el contrario, si contiene muchos datos, el modelo de entrenamiento se sobreajusta. Se

almacena en la base de datos información personal de cada usuario, información que será usada para generar una etiqueta única de identificación.

- **Bloque 2: Procesamiento de datos**. - Se inicia el tratamiento a las imágenes de entrenamiento mediante la biblioteca OpenCV, el cual procede a realizar un preprocesamiento de las imágenes del conjunto de datos convirtiéndolas de un formato BGR a RGB (convenciones para el orden de los diferentes canales de color) para la precisión del detector facial. Consecuentemente, la alineación del rostro se lleva a cabo por un proceso en el cual se busca recortar la zona de la detección del rostro (ROI) y redimensionar a una imagen.

- **Etapa 3: Entrenamiento. -** Se extrae un vector de incrustaciones faciales de 128 mediciones por cada rostro. Dichos vectores son clasificados en clases mediante un algoritmo de aprendizaje automático supervisado (SVM) tomando como datos de entrenamiento al conjunto de incrustaciones obtenidas de los rostros desde un directorio de imágenes y su etiqueta o identificación de clase correspondiente.

- **Etapa 4: Análisis Correlacional**. - Se correlaciona dicho conjunto de incrustaciones faciales obtenidas en la etapa anterior con el modelo entrenado de aprendizaje automático supervisado. Específicamente, dicha correlación se basa en la selección de la máxima probabilidad de cercanía o similitud del conjunto de incrustaciones faciales obtenidas en tiempo real con el conjunto de incrustaciones previamente entrenadas. El resultado del proceso anterior muestra información de identidad en caso de existir una coincidencia, caso contrario se etiqueta como desconocida. Finalmente, se interpretan

los resultados y se registran en la base de datos de personas conocidas o desconocidas para la generación de un reporte o notificaciones, según sea el caso.

- **Base de datos**. – Finalmente en la base de datos local se almacenará toda la información generada por los subprocesos anteriores.

La representación gráfica de la Figura 38 muestra de mejor manera los aspectos anteriormente mencionados:

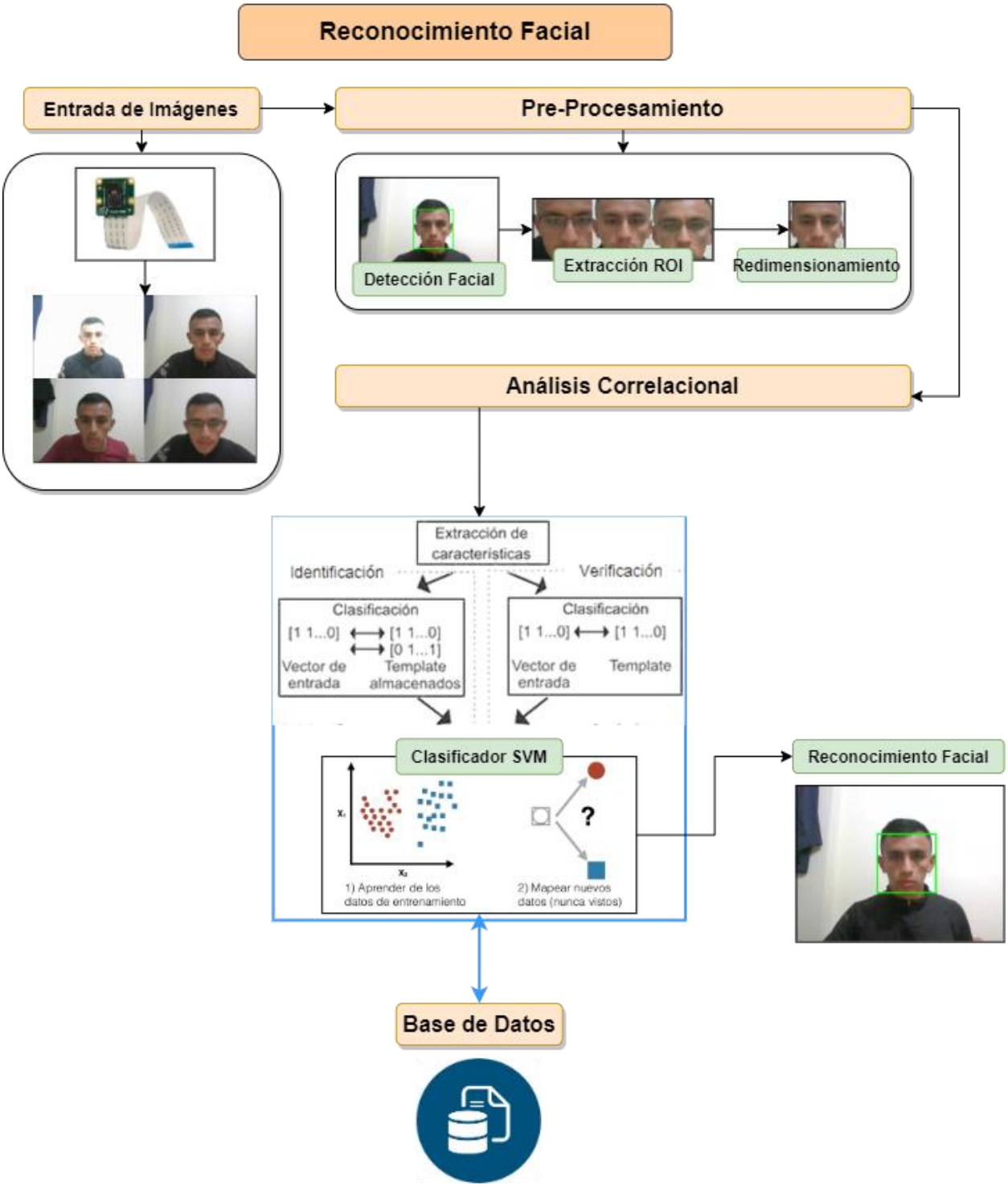


Figura 38. Representación gráfica del diagrama de bloques del Reconocimiento Facial.

Fuente: Autoría.

La representación gráfica de la Figura 39 muestra la distribución de los dispositivos a usarse, y la conexión con la placa base.

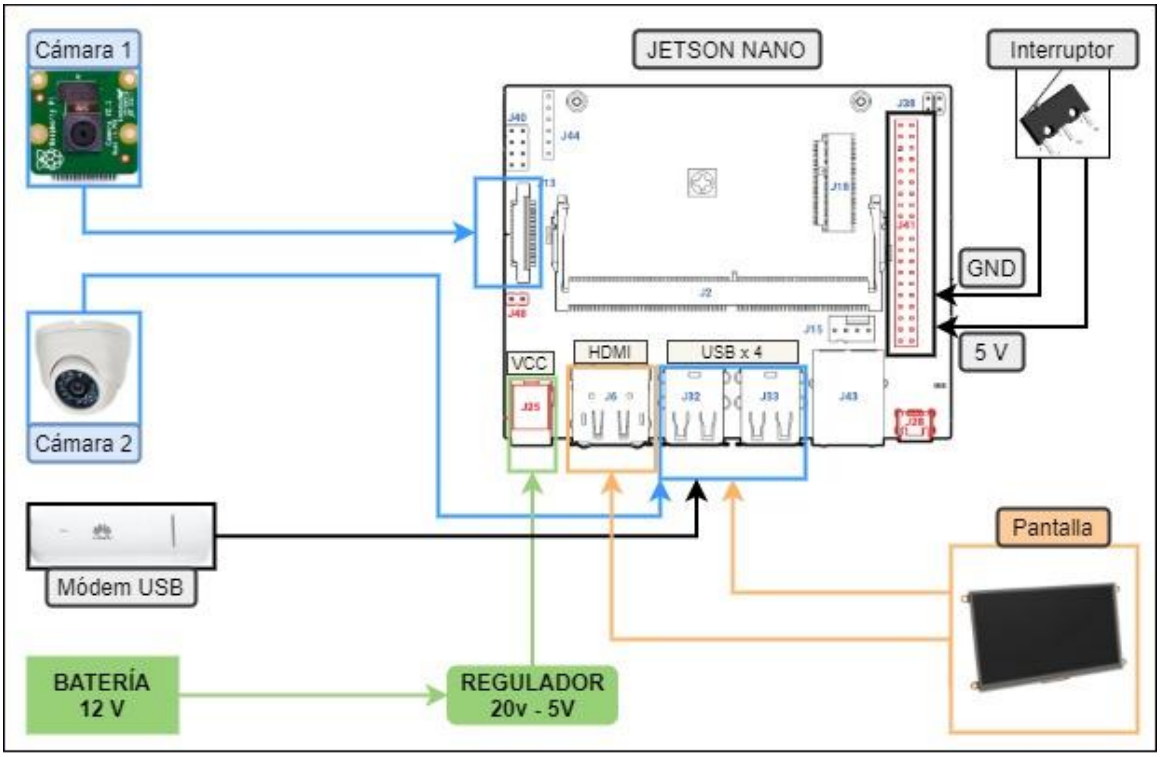


Figura 39. Diagrama de conexiones de la Etapa 1.

Fuente: Autoría.

3.3.1.2. Diagrama de bloques módulo Contador de Personas.

Para implementar un contador de personas se aprovechará las funciones de OpenCV y dlib. Se usará OpenCV para funciones estándar de procesamiento de imágenes/visión por computadora, junto con el detector de objetos de aprendizaje profundo para el recuento de personas. Mientras que dlib se usara para su implementación de filtros de correlación.

Junto con la implementación de seguimiento de objetos de dlib, también se utilizará la función de implementación de seguimiento de centroide.

La Figura 40 muestra la disposición de bloques destinadas a esta etapa.

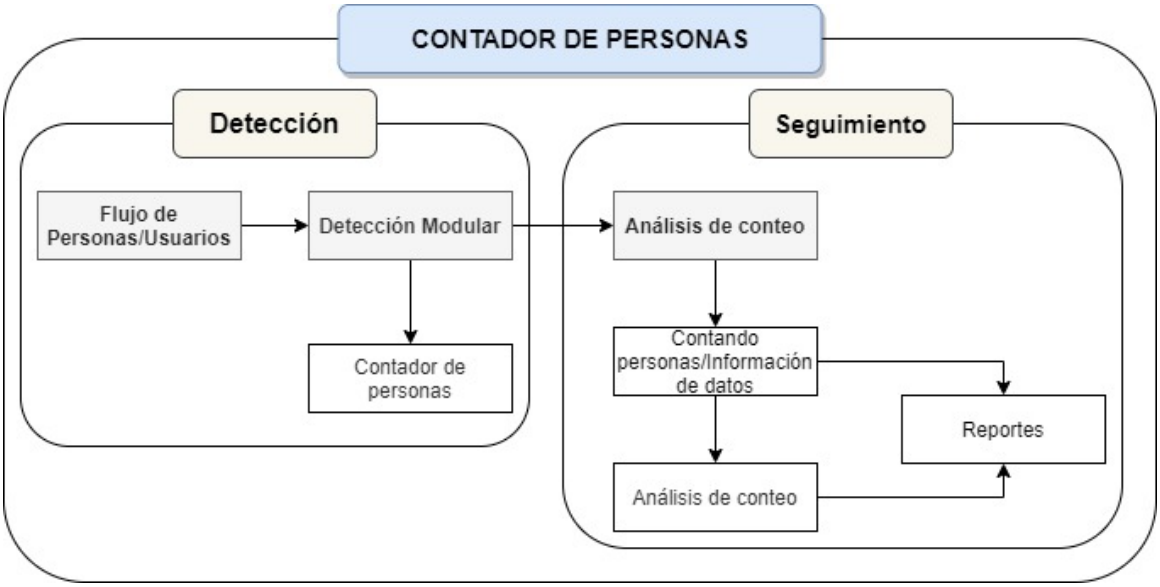


Figura 40. Diagrama de bloques del segundo bloque del Sistema.

Fuente: Autoría.

- **Etapa 1: Detección. -** durante la fase de detección, se ejecuta un rastreador de objetos computacionalmente más costoso para: a) detectar si nuevos objetos han ingresado al ángulo de visión de las cámaras y b) ver si se puede encontrar objetos que se "perdieron" durante la fase de seguimiento. Para cada objeto detectado, se crea o actualiza un rastreador de objetos con las nuevas coordenadas del cuadro delimitador. Dado que el detector de objetos es más caro computacionalmente, solo se ejecuta esta fase una vez cada N fotogramas.

- **Etapa 2: Seguimiento. -** Cuando el sistema no está en la fase de "detección", está en la fase de "seguimiento". Para cada uno de los objetos detectados, se procede a crear un rastreador de objetos para rastrear el objeto a medida que se mueve por el marco. Dicho rastreador de objetos debe ser más rápido y eficiente que el detector de objetos. El proceso de rastreo continua hasta llegar al enésimo cuadro y luego se vuelve a ejecutar el detector de objetos. Luego, todo el proceso se repite.

La representación gráfica de la Figura 41 muestra la distribución de los dispositivos a usarse, y la conexión con la placa base.

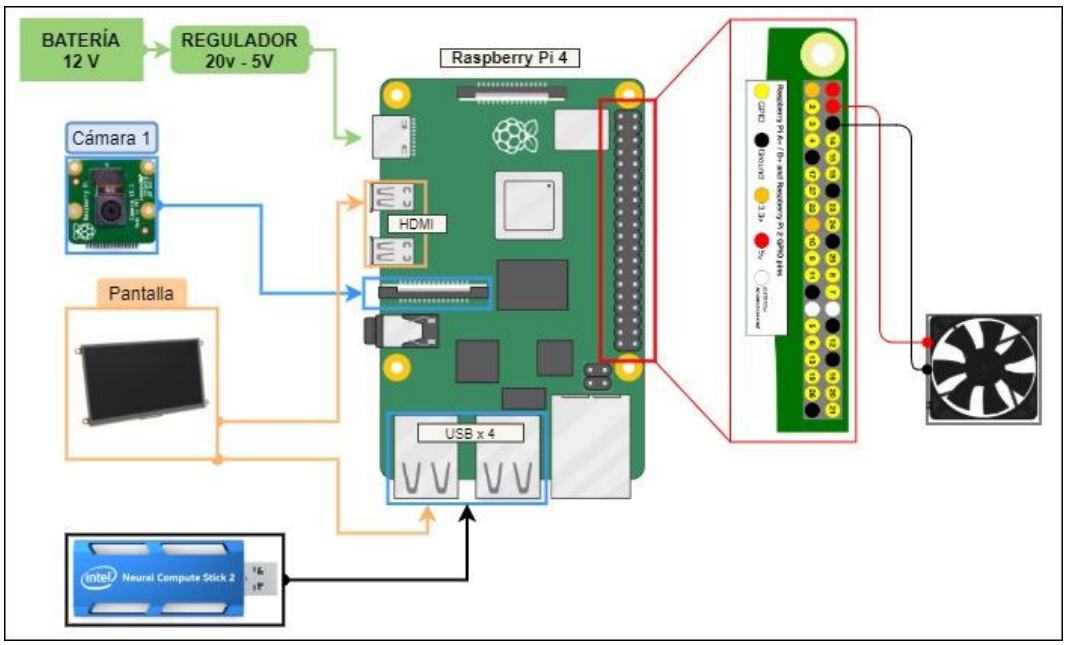


Figura 41. Diagrama de conexiones del Contador de Personas.

Fuente: Autoría.

3.3.1.3. Diagrama de bloques módulo Alertas.

La tercera etapa consta de los mensajes de notificación, es decir las alertas que generara el sistema. Por medio de un modem USB, que utiliza las bandas de frecuencia de las operadoras de telefonía celular, el sistema tendrá acceso a la red para él envió de notificaciones. En la Figura 42 se puede observar el diagrama de bloques de la fase de envió de notificaciones.

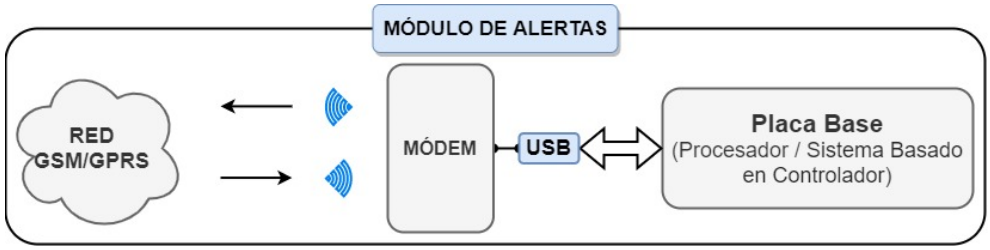


Figura 42. Diagrama de bloques del tercer bloque del Sistema.

Fuente: Autoría.

Estos dispositivos poseen la capacidad de conectarse a redes GSM, GPRS, 3G y 4G con bandas de frecuencia 850, 900, 1800, 1900, 2100 MHz. Su conexión por medio de un puerto USB, y el bajo costo de adquisición lo hace un producto viable para usarlo como parte de la red de transporte de la información registrada.

3.3.1.4. Diagrama de bloques módulo Alimentación del Sistema.

La tercera etapa consta de la alimentación del sistema. Para la alimentación del sistema se procede a conectar el conversor de 12 V CD a 5 V CD a través del encendedor del vehículo, para que el sistema funcione solo cuando el vehículo se encuentre encendido.

En la Figura 43 se puede observar el diagrama de bloques de la fase de envió de notificaciones.

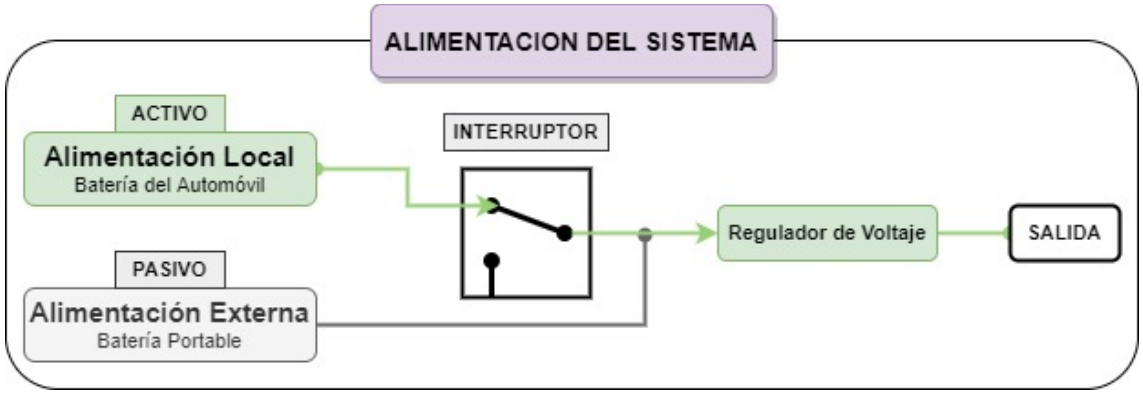


Figura 43. Diagrama de bloques del cuarto bloque del Sistema.

Fuente: Autoría.

El sistema contara con alimentación de respaldo en caso de que la alimentación principal falle. Como se muestra en el diagrama de bloques, la alimentación principal será el circuito conectado con la batería del automóvil, esta genera una tensión de 12 V la cual deberá ser transformada mediante un regulador de voltaje para que a la salida entregue 5 VDC.

Por otro lado, el sistema de alimentación de respaldo entrará en funcionamiento solo si el sistema principal falla o el automóvil se apaga accidentalmente. Para ello el sistema cuenta con una batería portable que suministre la potencia suficiente para el funcionamiento normal del sistema.

En la Figura 44, se muestra el diagrama de conexión de la etapa de alimentación del sistema.

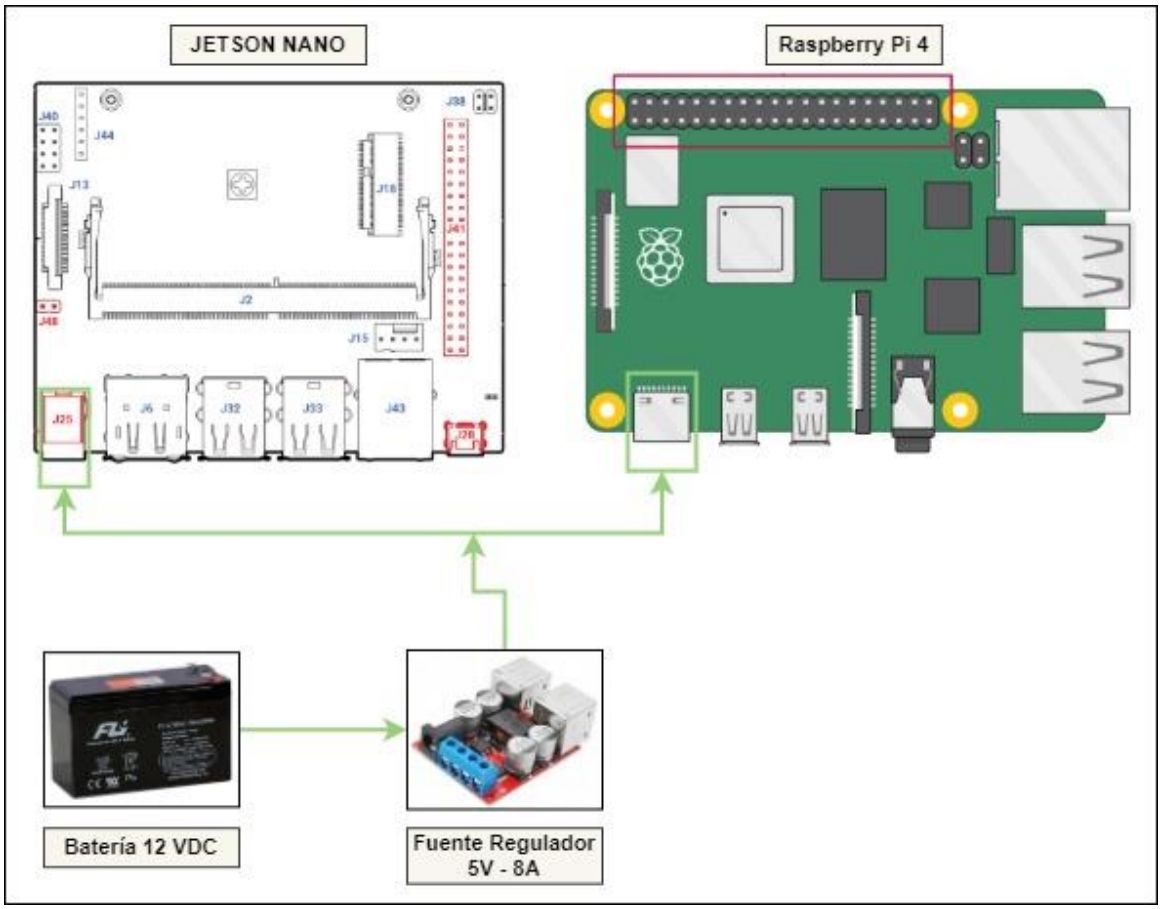


Figura 44. Diagrama de conexiones de la Etapa 4.

Fuente: Autoría.

En la Figura 46, se muestra un diagrama final del Asistente Virtual en donde consta todos los dispositivos y conexiones.

3.3.1.5. Diagrama de bloques Base de Datos.

La base de datos es la encargada de recopilar la información que se genera de los diferentes módulos del sistema. Normalmente, una base de datos está controlada por un sistema de gestión de bases de datos (DBMS).

En la Figura 45 se muestra un diagrama de bloques de la base de datos.

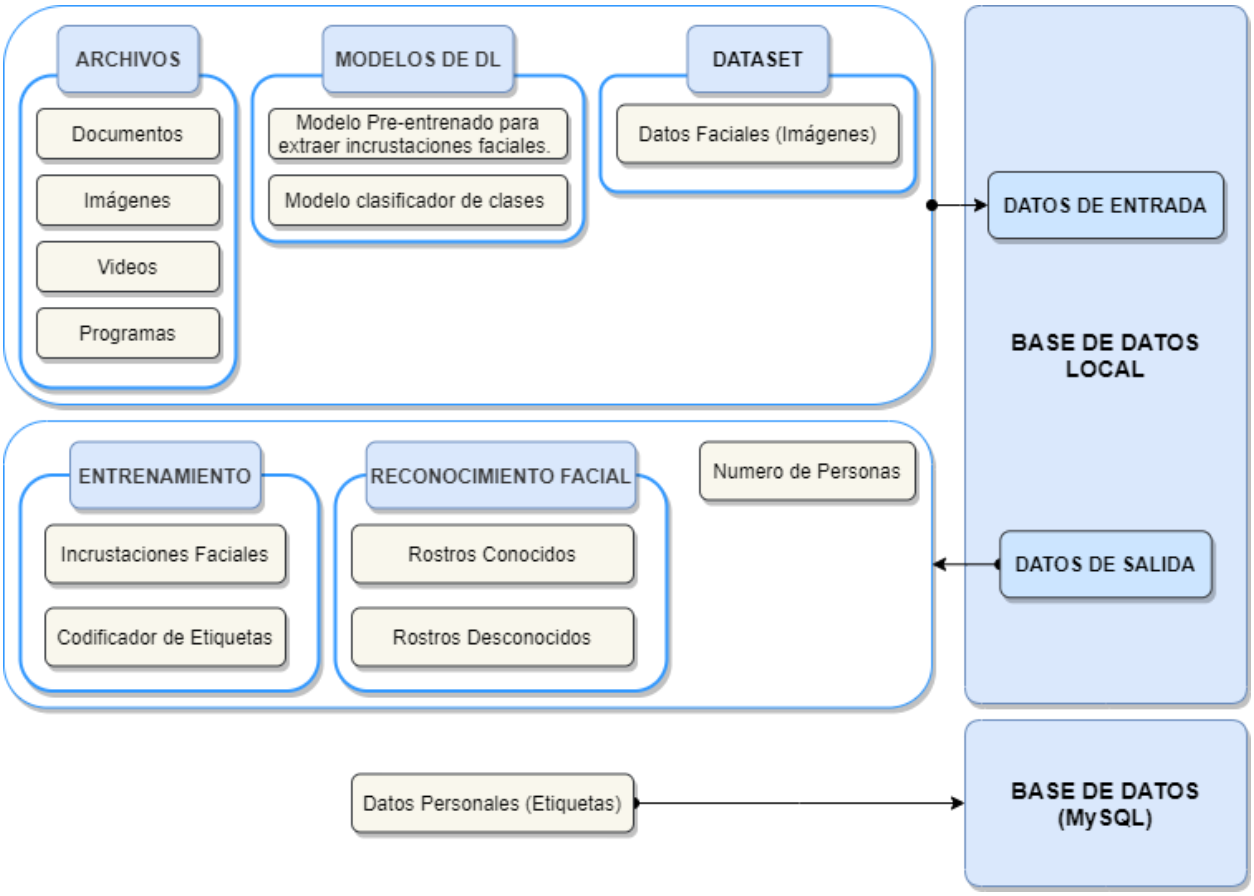


Figura 45. Diagrama de bloques de la Base de Datos.

Fuente: Autoría.

La base de datos se encarga de organizar la información, en este caso se tiene datos de entrada y datos de salida:

- Datos de Entrada: Corresponden a los datos que se almacenan manualmente en la base de datos.

- Datos de Salida: Son el resultado de la ejecución del sistema, estos datos se generan a partir de los módulos que conforman el sistema.

Estos datos se almacenan localmente, es decir, se alojan en carpetas y subcarpetas creadas por el administrador del sistema. Por otro lado, se tiene la base de datos (MySQL) que únicamente se encarga de la recolección de datos personales como: cedula, nombres, apellidos, dirección, institución educativa, edad, genero, nombre del tutor, etc. Dicha base de datos esta anexada a la interfaz de usuario, lo que permite el ingreso de datos de manera ágil y eficiente.

3.3.1.6. Integración del sistema.

Como se describió en las secciones anteriores, la fase de Diseño consta de varios módulos que posteriormente se integraran y funcionaran como un solo sistema.

A continuación, en la Figura 46, se muestra un diagrama con los diferentes módulos del sistema de forma integrada.

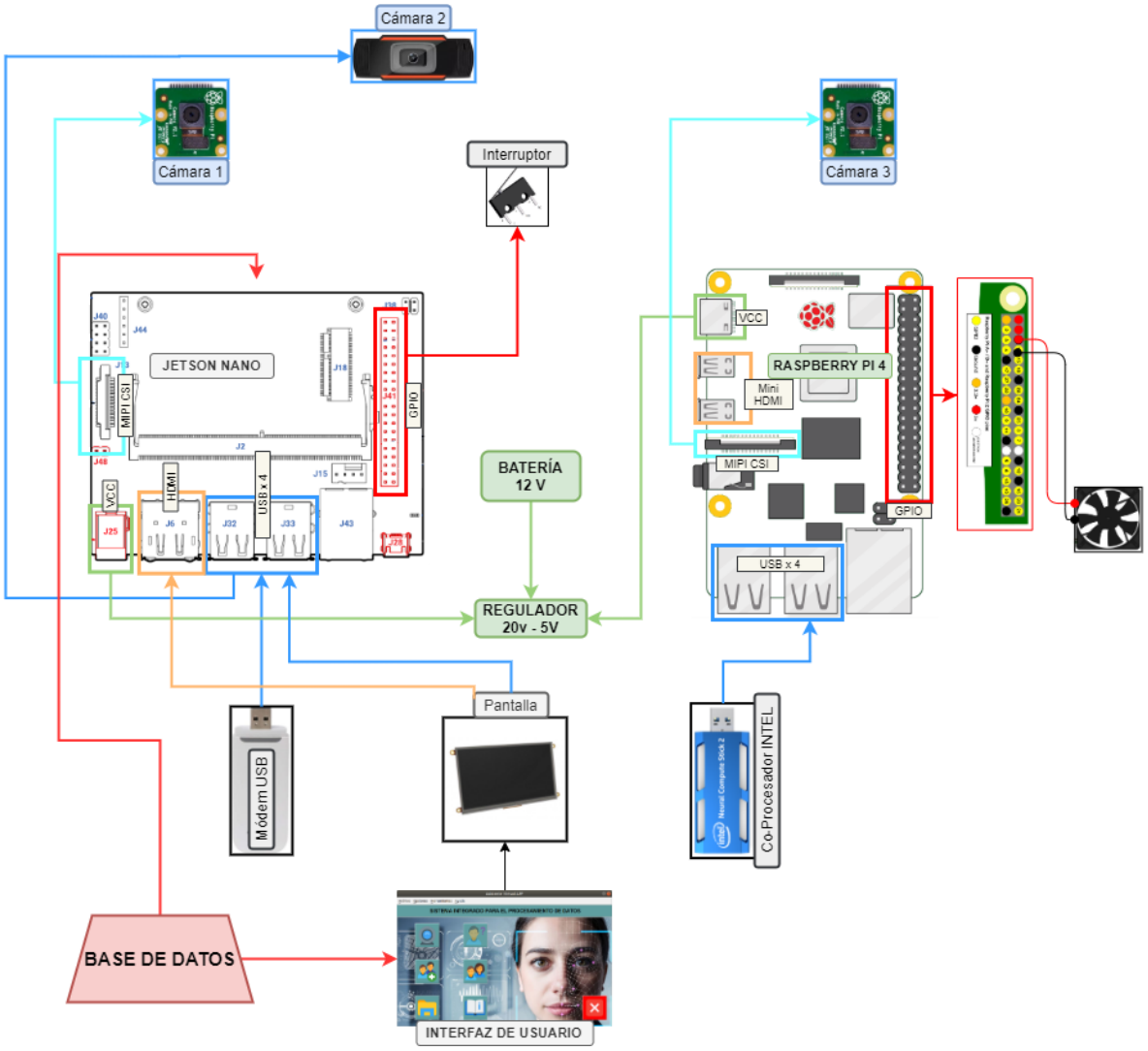


Figura 46. Diagrama Final de las etapas que conforman el proyecto.

Fuente: Autoría.

El módulo de Reconocimiento Facial conlleva el mayor procesamiento; su correcto funcionamiento dependerá de los distintos dispositivos que se integran a este módulo.

Posteriormente se ejecuta el módulo de conteo de personas, dicha etapa está completamente ligada a la anterior, pero cumplen una función diferente. Los dos módulos mencionados representan la mayor parte del procesamiento del sistema completo y además corresponden a la parte esencial del proyecto. El módulo de alertas trabajará en conjunto en el reconocimiento fascial,

se encargará de las notificaciones mediante la red de telefonía existente; se generarán reportes de los usuarios que ingresen o salgan del recorrido. Finalmente, la alimentación del sistema asegurara su funcionamiento adecuado.

3.3.2. Desarrollo del software (Codificación).

En esta sección se procede a desarrollar los módulos que componen el software de programación en las etapas de capacitación de los modelos de visión artificial y la etapa de ejecución en tiempo real. Para el desarrollo de software se utiliza una estación de trabajo Windows y Linux, las cuales se encargarán de compilar toda la codificación realizada a través del lenguaje de programación interprete de Python, donde cada etapa del sistema poseerá un script o archivo ejecutable. Finalmente, se unifica el software de cada etapa en las placas embebidas correspondientes.

3.3.2.1. Reconocimiento Facial (OpenFace)

Esta etapa se destina a la descripción del método de aprendizaje profundo empleado para la posterior extracción de características representativas de cualquier conjunto de datos provisto.

Para el desarrollo de la primera etapa del sistema, se emplea la biblioteca de reconocimiento facial OpenFace, que proporciona una precisión casi humana en el punto de referencia LFW y además dentro del desarrollo de sus características presenta un nuevo punto de referencia de clasificación para escenarios móviles, en los cuales ha realizado experimentos de rendimiento que muestran que el tiempo de ejecución de OpenFace es adecuado para escenarios móviles en comparación con otras técnicas. En la Figura 47 se muestra la estructura del proyecto OpenFace.

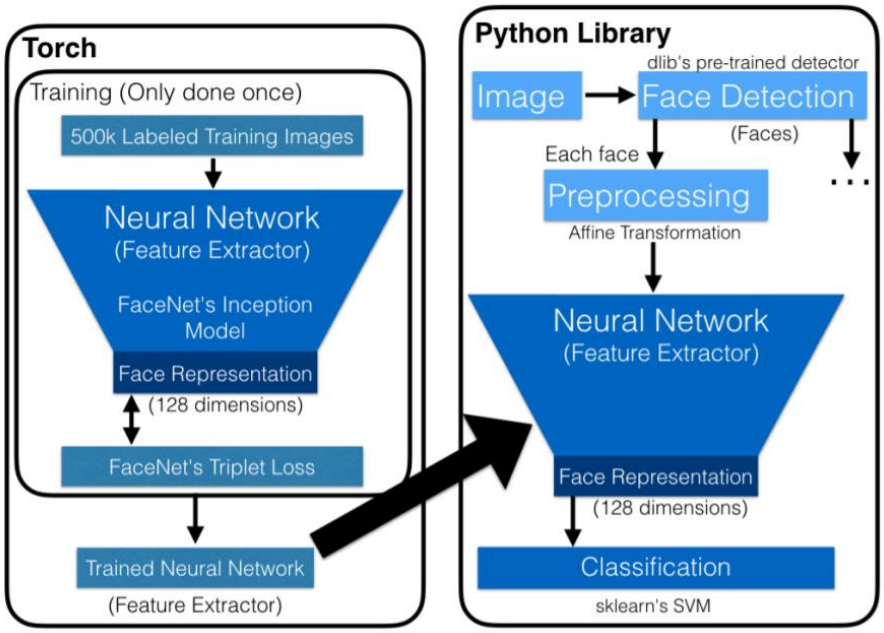


Figura 47. Estructura del proyecto de OpenFace.

Fuente: Adaptado de (Amos, Ludwiczuk, & Satyanarayanan, 2016).

Las porciones de entrenamiento e inferencia de redes neuronales utilizan Torch, Lua y luajit. La biblioteca Python usa numpy para arreglos y operaciones de álgebra lineal, OpenCV para primitivas de visión por computadora y scikit-learn para clasificación. La estructura del proyecto es independiente de la arquitectura de la red neuronal y actualmente usa la arquitectura de FaceNet.

Se emplea el detector facial preentrenado de dlib para una mayor precisión que el detector de OpenCV. OpenFace proporciona el flujo lógico presentado en la Figura 48 para obtener representaciones de caras de baja dimensión para las caras en una imagen.

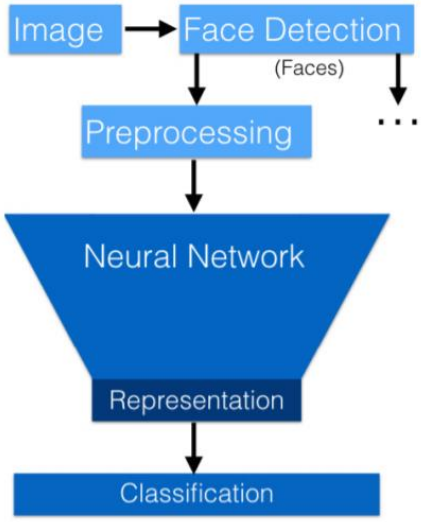


Figura 48. Flujo lógico para el reconocimiento facial con una red neuronal.

Fuente: Adaptado de (Amos, Ludwiczuk, & Satyanarayanan, 2016).

3.3.2.1.1. Preprocesamiento

La parte de detección de rostros devuelve una lista de cuadros delimitadores alrededor de los rostros en una imagen que puede estar en diferentes condiciones de pose e iluminación. Un problema potencial al usar los cuadros delimitadores directamente como entrada en la red neuronal es que las caras podrían mirar en diferentes direcciones o bajo diferentes condiciones de iluminación. Para resolver esto se reduce el tamaño del espacio de entrada normalizando las caras para que los ojos, la nariz y la boca aparezcan en ubicaciones similares en cada imagen. OpenFace utiliza una simple transformación afín 2D para hacer que los ojos y la nariz aparezcan en ubicaciones similares para la entrada de la red neuronal. La Figura 49 ilustra cómo la transformación afín normaliza las caras.

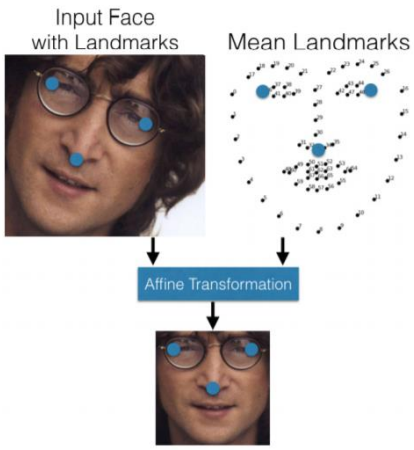


Figura 49. La transformación se basa en los grandes puntos de referencia azules y la imagen final se recorta hasta los límites y se redimensiona a 96 × 96 píxeles.

Fuente: Adaptado de (Amos, Ludwiczuk, & Satyanarayanan, 2016).

Los 68 puntos de referencia se detectan con el detector de puntos de referencia facial de dlib. Dada una cara de entrada, la transformación afín hace que las esquinas de los ojos y la nariz se acerquen a las ubicaciones medias. También cambia el tamaño y recorta la imagen en los bordes de los puntos de referencia para que la imagen de entrada a la red neuronal sea de 96 × 96 píxeles.

3.3.2.1.2. Entrenamiento de la red neuronal

Entrenar la red neuronal requiere una gran cantidad de datos. OpenFace está capacitado con 500k imágenes de la combinación de los dos conjuntos de datos de reconocimiento facial etiquetados más grandes para la investigación, CASIA-WebFace y FaceScrub.

El componente de red neuronal de la Figura 49 asigna una imagen preprocesada (alineada) a una representación de baja dimensión. OpenFace usa una versión modificada de la red nn4 de FaceNet. nn4 se basa en la arquitectura GoogLeNet y la variante nn4.small2 modificada reduce el número de parámetros para el conjunto de datos más pequeño. OpenFace utiliza la pérdida de triplete de FaceNet por lo que la red proporciona una incrustación en la unidad de hiperesfera y la distancia euclidiana representa la similitud.

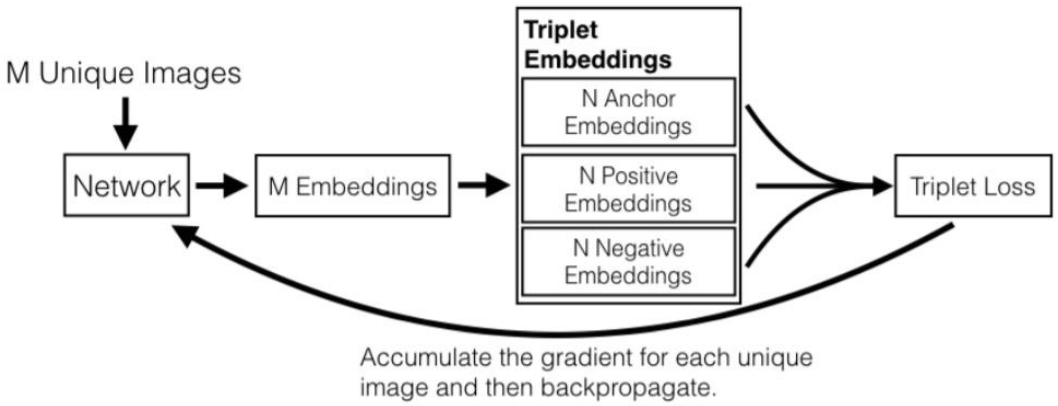


Figura 50. Flujo de entrenamiento de la red de un extremo a otro de OpenFace.

Fuente: Adaptado de (Amos, Ludwiczuk, & Satyanarayanan, 2016).

La figura 50 muestra cómo se entrena la red. Se mapea las imágenes únicas de una sola red en tripletes. El gradiente de la pérdida del triplete se propaga hacia atrás a través del mapeo a las imágenes únicas. En cada mini-lote, se toma muestras como máximo de P imágenes por persona de Q personas en el conjunto de datos y se envía todas las imágenes $M \approx PQ$ a través de la red en una sola pasada hacia adelante en la GPU para obtener M incrustaciones. OpenFace usa $P = 20$ y $Q = 15$. Se toma todos los pares ancla-positivos para obtener $N = Q\binom{P}{2}$ tripletes. Se calcula la pérdida del triplete y se mapea la derivada hasta la imagen original en una pasada de red hacia atrás. Si no se encuentra una imagen negativa dentro del margen α para un par de anclaje positivo dado, no se usa.

3.3.2.1.3. Evaluación

La evaluación estudia la precisión y el rendimiento de OpenFace en comparación con otras técnicas de reconocimiento facial. El conjunto de datos de LFW es un punto de referencia estándar en la investigación de reconocimiento facial y la Sección 3.3.2.1.4 presenta la precisión de OpenFace en el experimento de verificación de LFW. La sección 3.3.2.1.5 presenta un nuevo punto

de referencia de clasificación que utiliza el conjunto de datos LFW para escenarios móviles transitorios. Todos los experimentos utilizaron el modelo OpenFace nn4.small2.v1.

3.3.2.1.4. Verificación LFW

El experimento de verificación LFW predice si los pares de imágenes son de la misma persona. El LFW tiene 13.233 imágenes de 5.750 personas y este experimento proporciona 6.000 pares divididos en diez pliegues.

La precisión en el protocolo restringido se obtiene promediando la precisión de diez experimentos. Los datos se separan en diez pliegues de igual tamaño y cada experimento se entrena en nueve pliegues y calcula la precisión en el pliegue de prueba restante. Los resultados de OpenFace se obtienen calculando la distancia euclidiana al cuadrado en los pares y etiquetando los pares por debajo de un umbral como la misma persona y por encima del umbral como personas diferentes. El mejor umbral en los pliegues de entrenamiento se utiliza como umbral en el pliegue restante. En nueve de cada diez experimentos, el mejor umbral es 0,99. La Figura 51 compara la precisión de OpenFace con otras técnicas. A diferencia de las técnicas modernas basadas en redes neuronales profundas, el resultado de Eigenfaces no utiliza datos externos.

Technique	Accuracy
Human-level (cropped) [KBBN09]	0.9753
Eigenfaces (no outside data) [TP91][3]	0.6002 ± 0.0079
FaceNet [SKP15]	0.9964 ± 0.009
DeepFace-ensemble [TYRW14]	0.9735 ± 0.0025
OpenFace (ours)	0.9292 ± 0.0134

Figura 51. Precisiones de LFW y pares de imágenes de ejemplo.

Fuente: Adaptado de (Amos, Ludwiczuk, & Satyanarayanan, 2016).

El umbral de verificación se puede variar y trazar como una curva ROC (característica operativa del receptor) como se muestra en la Figura 52.

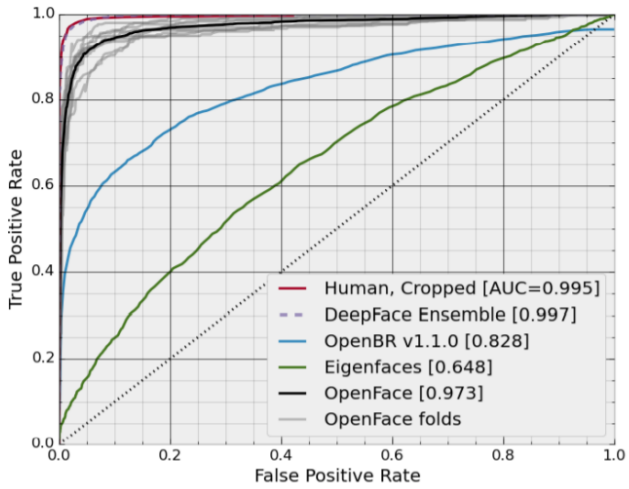


Figura 52. Curva ROC en el punto de referencia LFW con valores de área bajo la curva (AUC).

Fuente: Adaptado de (Amos, Ludwiczuk, & Satyanarayanan, 2016).

La curva ROC muestra las compensaciones entre el TPR (True Positive Rate) y el FPR (False Positive Rate). La curva ROC perfecta tendría un TPR de 1 en todas partes, que es donde se encuentran casi en la actualidad las técnicas de la industria de vanguardia. El área bajo la curva (AUC) es la probabilidad de que el clasificador clasifique las caras elegidas al azar de la misma persona por encima de las caras elegidas al azar de diferentes personas. Cada curva es un promedio de las curvas obtenidas al establecer el umbral de cada pliegue de datos.

Estos resultados muestran que la precisión de OpenFace se acerca a la precisión de las técnicas de aprendizaje profundo de última generación.

3.3.2.1.5. Clasificación LFW

Muchos escenarios que utilizan el reconocimiento facial implican clasificar quién es una persona, no solo si dos caras son iguales. Se mide las precisiones de clasificación en un

subconjunto de imágenes LFW. Además, se presenta resultados que comparan las técnicas de reconocimiento facial de OpenCV (eigenfaces, fisherfaces e histogramas de patrones binarios locales (LBPH)) con las de OpenFace.

La Figura 53 muestra una descripción general de la configuración del experimento. La persona i corresponde a la persona del LFW con más imágenes. Luego, se muestrean 20 imágenes de las primeras N personas. Si no se proporcionan 20 imágenes de una persona, se utilizan todas sus imágenes. A continuación, los datos muestreados se dividen al azar diez veces colocando el 90% de las imágenes en el conjunto de entrenamiento y el 10% de las imágenes en el conjunto de prueba. La semilla de número aleatorio debe inicializarse con el mismo valor para muestrear diferentes experimentos. Se entrena a un clasificador en el conjunto de entrenamiento y la precisión se obtiene al predecir las identidades en el conjunto de prueba.

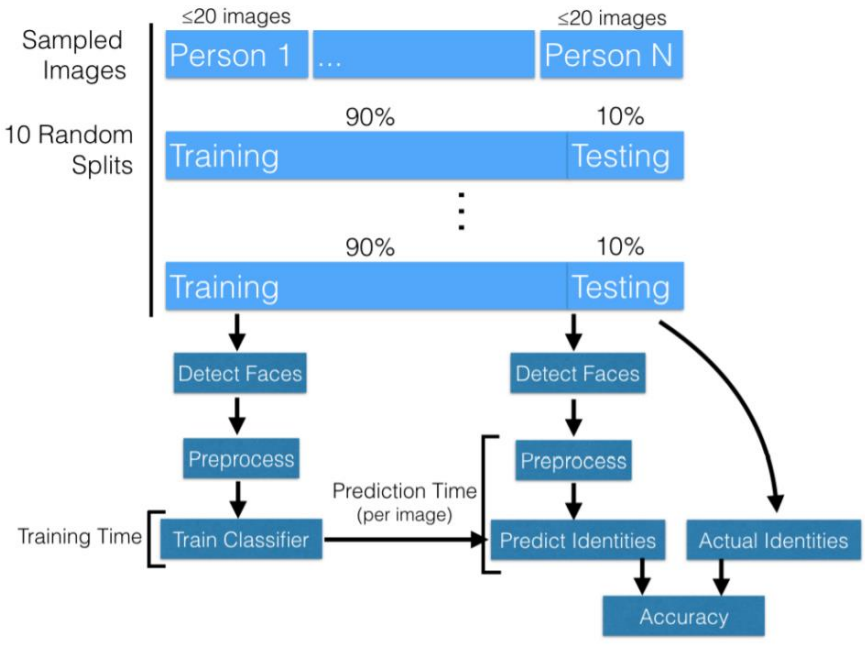


Figura 53. Descripción general de la precisión de la clasificación LFW.

Fuente: Adaptado de (Amos, Ludwiczuk, & Satyanarayanan, 2016).

Medir el rendimiento en tiempo de ejecución de los clasificadores de entrenamiento y predecir a quién pertenecen las nuevas imágenes es una consideración importante para la movilidad. En los dispositivos portátiles, el rendimiento de predecir a quién pertenece un rostro es importante, por lo que no hay un retraso notable cuando un usuario mira a otra persona.

Todas las técnicas de OpenCV tienen la misma interfaz. El preprocesamiento convierte la imagen en escala de grises. El preprocesamiento de OpenFace en este contexto significa la transformación afín para la alineación seguida del pase directo de la red neuronal para la representación de 128 dimensiones. La clasificación de OpenFace utiliza una SVM lineal con una ponderación de regularización de 1.

3.3.2.2. Etapa 1

Para el desarrollo de esta etapa, primero se realiza la detección de rostros, de donde se extrae incrustaciones faciales de 128d para cuantificar una cara mediante el modelo preentrenado descrito en la sección anterior, se entrena una máquina de vectores de soporte (SVM) sobre las incrustaciones y finalmente se reconoce rostros tanto en imágenes como en secuencias de video.

Para construir la canalización de reconocimiento facial OpenCV, se aplica el aprendizaje profundo en dos pasos clave:

- Para aplicar la detección de rostros, que detecta la presencia y ubicación de un rostro en una imagen, pero no lo identifica.
- Para extraer los vectores de características de 128 d (llamados "incrustaciones") que cuantifican cada cara en una imagen.

3.3.2.2.1. Deteccion Facial

El detector de rostros de aprendizaje profundo de OpenCV se basa en el marco de Single Shot Detector (SSD) con una red base ResNet (a diferencia de otras SSD OpenCV que normalmente utilizan MobileNet como red base).

Los detectores de disparo único (SSD) y MobileNets son métodos que se pueden utilizar para la detección de objetos en tiempo real y con considerable rapidez en dispositivos con recursos limitados (incluyendo Raspberry Pi, smartphones, etc.)

3.3.2.2.2. Alineación del Rostros

La alineación de la cara, como sugiere el nombre, es el proceso de: a) identificar la estructura geométrica de las caras y b) intentar obtener una alineación canónica de la cara basada en la traslación, rotación y escala.

Para el proceso de detección de caras la alineación de rostros es opcional, sin embargo, se ha demostrado que la alineación de rostros aumenta la precisión del reconocimiento facial.

3.3.2.2.3. Preprocesamiento

El modelo de aprendizaje profundo de FaceNet calcula una incrustación de 128d que cuantifica la cara en sí. Este proceso se lo realiza incluyendo: los datos de entrada a la red y la función de pérdida de triplete.

Para entrenar un modelo de reconocimiento facial con aprendizaje profundo, cada lote (*Batch*) de datos de entrada incluye tres imágenes: el ancla (*Anchor*), la imagen positiva (*Positive*) y la imagen negativa (*Negative*). La Figura 54 describe este proceso:

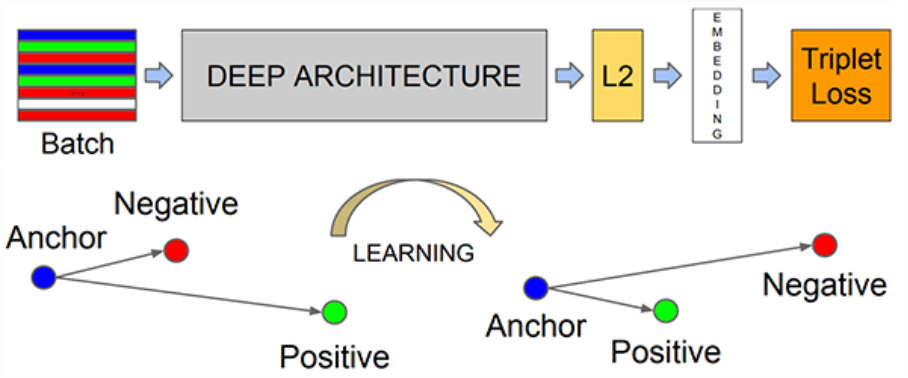


Figura 54. Cómo el modelo de reconocimiento facial de aprendizaje profundo calcula la incrustación facial.

Fuente: Adaptado de (Rosebrock, PyImageSearch, 2018).

El ancla pertenece al rostro actual y tiene identidad A. La segunda imagen es la imagen positiva; esta imagen también contiene un rostro de la persona A. La imagen negativa, por otro lado, no tiene la misma identidad y podría pertenecer a la persona B, C o incluso Z.

La red neuronal calcula las incorporaciones de 128 d para cada cara y luego ajusta los pesos de la red (a través de la función de pérdida de triplete) de modo que: las incrustaciones de 128d del ancla y la imagen positiva se encuentran más juntas, mientras que, al mismo tiempo, se alejan las incrustaciones de la imagen negativa del padre.

De esta manera, la red puede aprender a cuantificar caras y devolver incrustaciones altamente robustas y discriminatorias adecuadas para el reconocimiento facial.

El modelo será capaz de calcular las incrustaciones para cada cara; idealmente, estas incrustaciones de caras serán diferentes, de modo que se puede entrenar un clasificador de aprendizaje automático "estándar" (SVM, clasificador SGD, Random Forest, etc.) en la parte superior de las incrustaciones faciales y, por lo tanto, obtener la tubería de reconocimiento facial OpenCV.

3.3.2.2.4. Estructura de la Etapa 1

La etapa de reconocimiento Facial cuenta con cuatro directorios en la carpeta raíz, como se observa en la Figura 55.

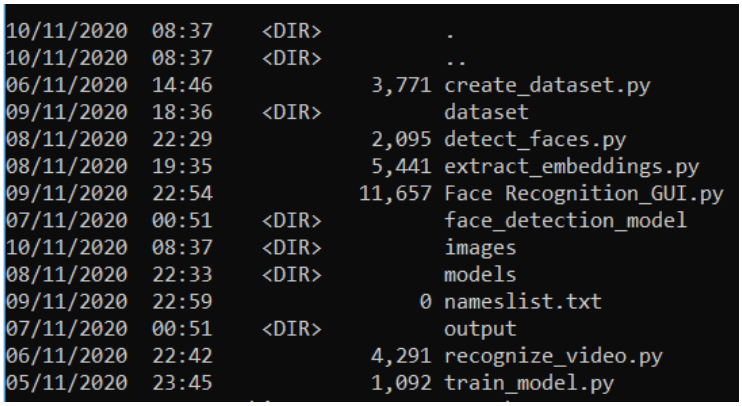


Figura 55. Directorio de los archivos de la etapa 1.

Fuente: Autoría.

Los cuatro directorios se explican a continuación:

- dataset /: Almacenara las imágenes faciales organizadas en subcarpetas por nombre.
- face_detection_models /: Contiene los modelos de aprendizaje profundo de Caffe previamente entrenado proporcionado por OpenCV para detectar rostros y para producir las incrustaciones faciales 128-D.
- images /: contiene imágenes de prueba para verificar el funcionamiento del modelo.
- output /: Guardara los archivos de .pickle de salida.

Los archivos de salida incluyen:

- embeddings.pickle: un archivo de incrustaciones faciales serializado. Las incrustaciones se han calculado para cada rostro en el conjunto de datos y se almacenan en este archivo.

- le.pickle: El codificador de etiquetas. Contiene las etiquetas de nombre de las personas que el modelo puede reconocer.
- recognizer.pickle: El modelo de máquina de vectores de soporte lineal (SVM). Este es un modelo de aprendizaje automático en lugar de un modelo de aprendizaje profundo y es responsable de reconocer caras.

Los archivos en el directorio raíz:

- create_dataset.py: Permite la creación del conjunto de datos para cada usuario.
- extract_embeddings.py: Es responsable de usar un extractor de funciones de aprendizaje profundo para generar un vector 128-D que describe una cara. Todas las caras del conjunto de datos se pasarán a través de la red neuronal para generar incrustaciones.
- train_model.py: El modelo de SVM lineal será entrenado por este script. Detectará rostros, extraerá incorporaciones y se podrá ajustar el modelo SVM a los datos de las incorporaciones.
- recognize_video.py: Describe cómo reconocer quién está en los fotogramas de una transmisión de video.
- Face Recognition_GUI.py: Genera una interfaz de usuario que contiene los archivos anteriores.

3.3.2.2.5. Crear Dataset.

En esta sección se detalla el procedimiento general para la recolección del conjunto de datos personalizado, que básicamente conlleva la creación de una pequeña base de datos, la cual

contiene fotografías de los rostros pertenecientes a los usuarios directos del sistema; para lograr esto se emplea el script “create_dataset.py” (Anexo 3). Este proceso implica la captura y recolección de alrededor de 200 a 300 imágenes de todos los ángulos faciales de cada individuo registrado en el sistema. Cuando se emplean conjuntos de datos más extensos y diversos en el entrenamiento de una arquitectura CNN la precisión del modelo tiende a aumentar significativamente, por lo tanto, el modelo de aprendizaje automático también debe entrenarse con una cantidad promedio ($m \geq 200$) de ejemplos de entrenamiento para lograr una precisión aceptable. Además, es preciso indicar que el registro de información personal tales como: nombres, apellidos y edad es administrada sobre una tabla de la base de datos local. Finalmente, una vez que el conjunto de datos ha sido elaborado, este se encuentra listo para la posterior extracción de incrustaciones faciales con la finalidad de construir un modelo de aprendizaje automático (clasificación) para la identificación facial, el cual se implementará en las siguientes secciones.

La Figura 56, muestra las imágenes recopiladas mediante el script.

Figura 56. Dataset de un usuario almacenada en la base de datos.

Fuente: Autoría.

3.3.2.2.6. Paso 1: Extraer Incrustaciones

En el Anexo 4 se describe el archivo extract_embeddings.py con su respectivo código. A continuación, se detalla algunas secciones del código.

Tabla 16.

Código para cargar el detector facial y el incrustador.

```
23.
24.  # Se carga el detector facial serializado desde el disco
25.  print("[INFO] cargando detector facial ...")
26.  protoPath = os.path.join("face_detection_model/deploy.prototxt")
27.  modelPath =
     os.path.join("face_detection_model/res10_300x300_ssd_iter_140000.caffemodel")
28.  detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)
29.
30.  # Se carga el modelo de incrustación facial serializado desde el disco
31.  print("[INFO] cargando modelo...")
32.  embedder = cv2.dnn.readNetFromTorch("openface_nn4.small2.v1.t7")
33.
```

Fuente: Autoría.

Como se observa en la Tabla 16, se carga el detector facial y el incrustador:

- `detector`: cargado a través de las líneas 26-28. Se utiliza un detector de rostros DL basado en Caffe para localizar rostros en una imagen.
- `embedder`: cargado en la línea 32. Este modelo está basado en Torch y es responsable de extraer incrustaciones faciales mediante la extracción de funciones de aprendizaje profundo.

Se usa las funciones `cv2.dnn` respectivas para cargar los dos modelos separados.

En la Tabla 17 se muestra el código para detectar y localizar rostros:

Tabla 17.

Código para detectar y localizar rostros.

```
34.
35.   # Se construye un BLOB a partir de la imagen
36.       imageBlob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0, (300, 300),
      (104.0, 177.0, 123.0), swapRB=False, crop=False)
37.
38.       detector.setInput(imageBlob)
39.       detections = detector.forward()
40.
41.       if len(detections) > 0:
42.           i = np.argmax(detections[0, 0, :, 2])
43.           confidence = detections[0, 0, i, 2]
44.
45.           if confidence > args["confidence"]:
46.               box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
47.               (startX, startY, endX, endY) = box.astype("int")
48.
49.               face = image[startY:endY, startX:endX]# ROI de la cara
50.               (fH, fW) = face.shape[:2]# obtener las dimensiones del ROI
51.
52.               if fW < 20 or fH < 20:
53.                   continue
```

Fuente: Autoría.

En la línea 36, se construye un BLOB. A partir de ahí se detecta caras en la imagen pasando el `imageBlob` a través de la red de detectores (líneas 38 y 39).

La lista de detecciones contiene probabilidades y coordenadas para localizar caras en una imagen. Suponiendo que se tiene al menos una detección, se ejecuta la sentencia if (línea 41). Si en la imagen solo hay una cara, se extrae la detección con la mayor confianza (`confidence`) y se verifica para asegurar que la confianza cumple con el umbral de probabilidad mínimo utilizado para filtrar detecciones débiles (líneas 42-45). Suponiendo que se alcanzado ese umbral, se extrae el ROI de la cara y se toma/verifica las dimensiones para asegurar de que el ROI de la cara sea lo suficientemente grande (líneas 46-53).

Como se muestra en la Tabla 18, se aprovecha las incrustaciones CNN y se extrae las incrustaciones de caras:

Tabla 18.

Incrustaciones CNN para extraer las incrustaciones de caras.

```
54.
55.                 faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
56.                     (96, 96), (0, 0, 0), swapRB=True, crop=False)
57.                 embedder.setInput(faceBlob)
58.                 vec = embedder.forward()
59.                 # add the name of the person + corresponding face
60.                 # embedding to their respective lists
61.                 knownNames.append(name)
62.                 knownEmbeddings.append(vec.flatten())
63.                 total += 1
```

Fuente: Autoría.

Esta vez se construye otro BLOB a partir del ROI de la cara (no la imagen completa) en las líneas 55 y 56. Posteriormente, el `faceBlob` pasa por el embebedor CNN (Líneas 57 y 58). Esto genera un vector 128-D (`vec`) que describe la cara. Estos datos son aprovechados para reconocer caras nuevas a través del aprendizaje automático. Luego simplemente se agrega el `name` y el `vec` incrustado a `knownNames` y `knownEmbeddings`, respectivamente (líneas 61 y 62).

Se aplica el mismo proceso de recorrer imágenes, detectar rostros y extraer incrustaciones de rostros para todas y cada una de las imágenes del conjunto de datos.

La ejecución del script arrojara la siguiente información y generara el archivo `embeddings.pickle`:

```
[INFO] procesando imagen 292/294
[INFO] procesando imagen 293/294
[INFO] procesando imagen 294/294
[INFO] serializando 36 codificaciones ...
```

Figura 57. Resultado de la extracción de las incrustaciones de caras.

Fuente: Autoría.

En la Figura 57, se puede ver que se ha extraído las incrustaciones de caras para cada una de las imágenes en el conjunto de datos de rostros de entrada.

3.3.2.2.7. Paso 2: Entrenamiento del Clasificador

En este se necesita entrenar un modelo de aprendizaje automático "estándar" como un SVM además de las incorporaciones. El archivo train_model.py del Anexo 5, permite este proceso. Parte del código se presenta en la Tabla 19.

Tabla 19.

Entrenamiento del modelo de aprendizaje automático SVM.

```
# Se cargan  las incrustaciones faciales
print("[INFO] cargando inserciones de caras ...")
data = pickle.loads(open("output/embeddings.pickle", "rb").read())

# Se codifican las etiquetas
print("[INFO] etiquetas de codificación ...")
le = LabelEncoder()
labels = le.fit_transform(data["names"])

# Se entrena el modelo utilizado para aceptar las incrustaciones 128-d de la cara
# para luego ejecutar el reconocimiento facial real
print("[INFO] modelo de entrenamiento ...")
recognizer = SVC(C=1.0, kernel="linear", probability=True)
recognizer.fit(data["embeddings"], labels)

# Se escribe el modelo de reconocimiento facial real en el disco
f = open("output/recognizer.pickle", "wb")
f.write(pickle.dumps(recognizer))
f.close()

# Se escribe el codificador de etiquetas en el disco
f = open("output/le.pickle", "wb")
f.write(pickle.dumps(le))
f.close()
```

Fuente: Autoría.

El módulo Scikit-Learn para aprendizaje automático en Python implementa el clasificador SVM, por lo que en este punto se procede a cargar los datos de entrenamiento que en este caso son los vectores de incrustación facial con sus respectivas etiquetas de clase. Para entrenar el algoritmo se emplea la función "`.fit`" que recibe los datos de entrenamiento mencionados anteriormente y crea un modelo a modo de fichero (`.pickle`) que contiene codificadas las incrustaciones y las etiquetas.

Aquí se carga las incorporaciones del Paso 1 (Extraer Incrustaciones) en la línea 19. En esta secuencia de comandos de entrenamiento del modelo no se genera ninguna incorporación; se usa las incorporaciones generadas y serializadas previamente.

Luego, se inicializa la función `LabelEncoder` scikit-learn y se codifica las etiquetas (`labels`) de nombre (líneas 23 y 24). En la Línea 29 se inicializa el modelo SVM, y en la Línea 30 ajusta (`fit`) el modelo (también conocido como “entrenamiento del modelo”).

Después del entrenamiento, se genera el modelo y el codificador de etiquetas en el disco como archivos pickle.

```
[INFO] cargando inserciones de caras ...
[INFO] etiquetas de codificación ...
[INFO] modelo de entrenamiento ...
```

Figura 58. Entrenamiento del SVN.

Fuente: Autoría.

La Figura 58, muestra que el SVM ha sido entrenado en las incrustaciones y tanto el a) SVM en sí como b) la codificación de la etiqueta se ha escrito en el disco, lo que permite aplicarlos a las imágenes de entrada y al video.

Hay que mencionar que el clasificador SVM dispone de algunas variantes en el tipo de función de kernel que se puede usar, tales como: lineal, sigmoidal, y polinomial. Para fines de este proyecto se escogió el kernel lineal.

3.3.2.2.8. Paso 3: Reconocimiento Facial

El archivo recognize_video.py que se encuentra en el Anexo 6, ejecuta el reconocimiento facial. El proceso empieza cargando los tres modelos generados anteriormente, del disco a la memoria. El código de la Tabla 20, muestra parte del código.

Usaremos `VideoStream` para capturar fotogramas de nuestra cámara y FPS para calcular las estadísticas de fotogramas por segundo.

Tabla 20.

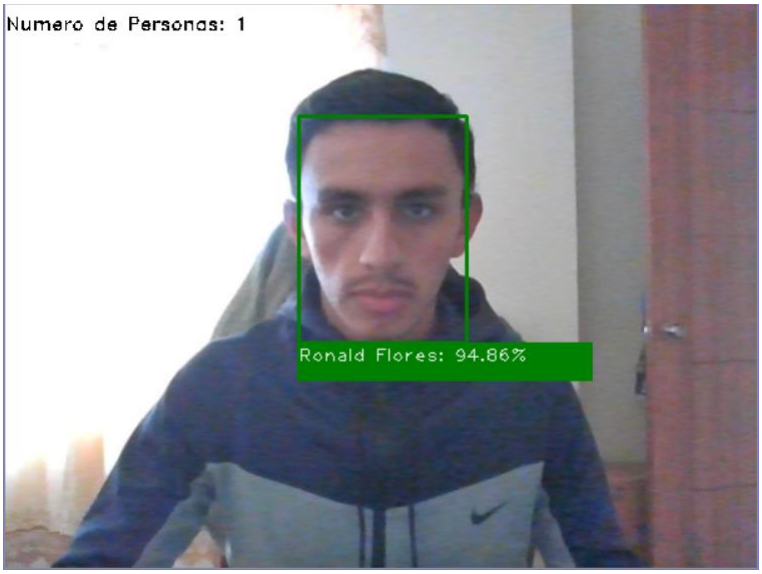
RF en imágenes con rostros.

```
91.         # Genera otro BLOB para el ROI de la cara, luego pasa el BLOB
92.                 # a través del modelo de incrustación facial
93.                 # para obtener la cuantificación 128-d de la cara.
94.                 faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
95.                                     (96, 96), (0, 0, 0), swapRB=True, crop=False)
96.                 embedder.setInput(faceBlob)
97.                 vec = embedder.forward()
98.
99.                 # Realiza clasificación para reconocer el rostro
100.                preds = recognizer.predict_proba(vec)[0]
101.                j = np.argmax(preds)
102.                proba = preds[j]
103.                name = le.classes_[j]
104.
105.                # Dibuja un cuadro delimitador de la cara
106.                # junto con la probabilidad asociada.
107.                text = "{}: {:.2f}%".format(name, proba * 100)
108.                y = startY - 10 if startY - 10 > 10 else startY + 10
109.                cv2.rectangle(frame, (startX, startY), (endX, endY),
110.                                    (0, 0, 255), 2)
111.                cv2.putText(frame, text, (startX, y),
112.                                    cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
113.
114.            # Actualiza el contador de FPS
115.            fps.update()
```

Fuente: Autoría.

Igual que en la extracción de incrustaciones, se construye el `faceBlob` (líneas 94 y 95), para luego calcular las incrustaciones faciales a través del aprendizaje profundo (líneas 96 y 97). A continuación, se realiza la clasificación para reconocer el rostro en donde se reconoce el nombre más probable de la cara mientras se calcula la probabilidad (línea 100-103).

Finalmente dibuja un cuadro delimitador alrededor de la cara y el nombre de la persona más la probabilidad (líneas 107-112). El contador de fps se actualiza en la línea 115. La Figura 59 muestra el resultado de la ejecución del código anteriormente descrito.



(a)

```
[INFO] tiempo transcurrido: 31.02
[INFO] approx. FPS: 15.86
```

(b)

Figura 59. Resultado del Reconocimiento Facial, (a) etiquetas y la probabilidad, (b) información de FPS.

Fuente: Autoría.

3.3.2.3. Etapa 2: Contador de Personas

Existe una diferencia fundamental entre la detección de objetos y el seguimiento de objetos. Cuando se aplica la detección de objetos, se determina en qué parte de una imagen / marco se encuentra un objeto. Un detector de objetos también suele ser más caro desde el punto de vista computacional y, por lo tanto, más lento que un algoritmo de seguimiento de objetos. Algunos ejemplos de algoritmos de detección de objetos incluyen cascadas de Haar, HOG + SVM lineal y

detectores de objetos basados en aprendizaje profundo, como Faster R-CNN, YOLO y detectores de disparo único (SSD).

Un rastreador de objetos, por otro lado, aceptará las coordenadas de entrada (x, y) de dónde está un objeto en una imagen y hará lo siguiente:

- Asignar una identificación única a ese objeto en particular.
- Rastrear el objeto a medida que se mueve alrededor de una secuencia de video, prediciendo la ubicación del nuevo objeto en el siguiente cuadro en función de varios atributos del cuadro (gradiente, flujo óptico, etc.)

Algunos ejemplos de algoritmos de seguimiento de objetos incluyen MedianFlow, MOSSE, GOTURN, filtros de correlación kernalizados y filtros de correlación discriminativos, por nombrar algunos.

3.3.2.3.1. Detección de objetos y seguimiento de objetos

Los rastreadores de objetos de alta precisión combinarán el concepto de detección y seguimiento de objetos en un solo algoritmo, normalmente dividido en dos fases:

- Fase 1 - Detección: durante la fase de detección, se ejecuta el rastreador de objetos computacionalmente más costoso para (1) detectar si nuevos objetos han ingresado al área visual y (2) ver si es posible encontrar objetos que se "perdieron" durante la fase de seguimiento. Para cada objeto detectado, se crea o actualiza un rastreador de objetos con las nuevas coordenadas del cuadro delimitador. Dado que el detector de objetos es más caro computacionalmente, solo se ejecuta esta fase una vez cada N fotogramas.

- Fase 2 - Seguimiento: Para cada uno de los objetos detectados, se crea un rastreador de objetos para rastrear el objeto a medida que se mueve por el marco. Dicho rastreador de objetos debería ser más rápido y eficiente que el detector de objetos. El proceso continúa hasta que se llega al enésimo cuadro y luego se vuelve a ejecutar el detector de objetos. Luego, todo el proceso se repite.

El beneficio de este enfoque híbrido es que se puede aplicar métodos de detección de objetos de alta precisión sin tanta carga computacional.

3.3.2.3.2. Estructura de la Etapa 2

Los archivos generados y necesarios para la ejecución del contador de personas se describen a continuación:

- pyimagesearch /: este módulo contiene el algoritmo de seguimiento del centroide.
- mobilenet_ssd /: contiene los archivos del modelo de aprendizaje profundo de Caffe. Se utilizará un detector de disparo único (SSD) MobileNet.

3.3.2.3.3. Algoritmos de seguimiento de objetos

Para implementar el contador de personas, se usará OpenCV y dlib. OpenCV para funciones estándar de procesamiento de imágenes / visión por computadora, junto con el detector de objetos de aprendizaje profundo para el conteo de personas.

Se usa dlib para la implementación de filtros de correlación. También se podría usar OpenCV; sin embargo, la implementación de seguimiento de objetos dlib permite una implementación mucho más sencilla. Junto con la implementación del seguimiento de objetos de dlib, también se usará la implementación del seguimiento del centroide.

- **Paso 1**: Se acepta un conjunto de cuadros delimitadores y se calcula sus centroides correspondientes (es decir, el centro de los cuadros delimitadores). Los cuadros delimitadores en sí pueden ser proporcionados por: un detector de objetos como HOG + SVM lineal, Faster R- CNN, SSD o un rastreador de objetos (como filtros de correlación). Como se ilustra en la Figura 60, existen dos objetos para rastrear en esta iteración inicial del algoritmo.

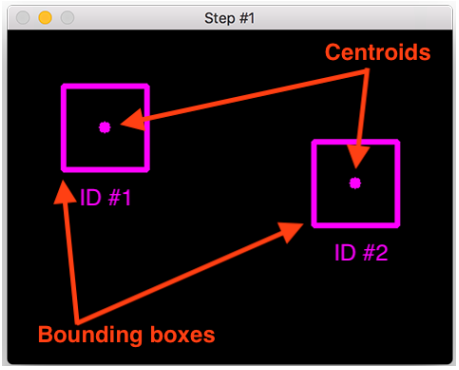


Figura 60. Coordenadas del cuadro delimitador para calcular los centroides.

Fuente: (Rosebrock, PyImageSearch, 2018)

- **Paso 2:** Se calcula la distancia euclidiana entre cualquier centroide nuevo y centroide existente. El algoritmo de seguimiento de centroides supone que los pares de centroides con una distancia euclidiana mínima entre ellos deben tener el mismo ID de objeto. En la Figura 61, se tiene dos centroides existentes (violeta) y tres nuevos centroides (amarillo), lo que implica que se ha detectado un nuevo objeto (ya que hay un nuevo centroide más frente a un centroide antiguo). Las flechas representan entonces el cálculo de las distancias euclidianas entre todos los centroides púrpuras y todos los centroides amarillos.

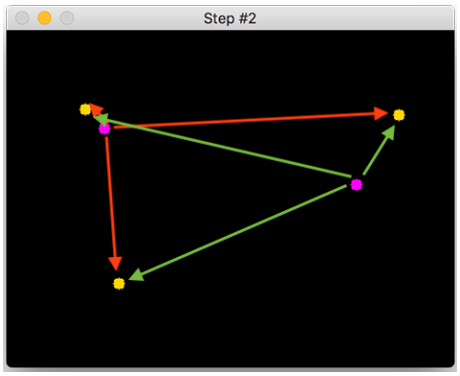


Figura 61. Tres objetos están presentes en esta imagen. Se necesita calcular la distancia euclidiana entre cada par de centroides originales (rojo) y los nuevos centroides (verde).

Fuente: (Rosebrock, PyImageSearch, 2018)

- **Paso 3:** Una vez que se tiene las distancias euclidianas, se intenta asociar los ID de objeto. En la Figura 62 se puede ver que el rastreador de centroides ha optado por asociar centroides que minimizan sus respectivas distancias euclidianas. En donde el punto en la parte inferior izquierda, no se asoció con nada.

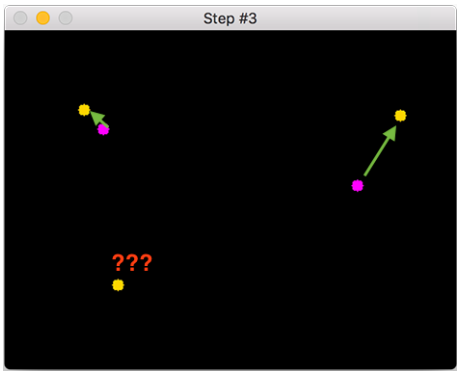


Figura 62. El método de seguimiento de objetos centroide simple tiene objetos asociados con distancias de objeto minimizadas.

Fuente: (Rosebrock, PyImageSearch, 2018)

- **Paso 4:** Registrar nuevos objetos. Como se indica en la Figura 63. Registrarse simplemente significa que se está agregando el nuevo objeto a la lista de objetos

rastreados por: asignarle un nuevo ID de objeto y almacenar el centroide de las coordenadas del cuadro delimitador para el nuevo objeto.

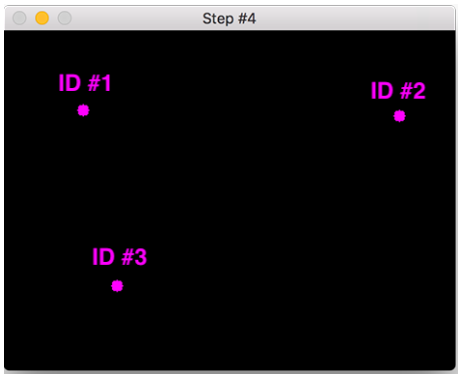


Figura 63. Se tiene un objeto nuevo que no coincide con un objeto existente, por lo que se registra como ID de objeto # 3.

Fuente: (Rosebrock, PyImageSearch, 2018)

- **Paso 5:** En el caso de que un objeto se haya perdido o abandonado el campo de visión, simplemente se cancela el registro del objeto. Para el contador de personas, se elimina las identificaciones de personas cuando no puedan coincidir con ningún objeto de persona existente durante 40 marcos consecutivos.

3.3.2.3.4. Objeto rastreable

Para rastrear y contar un objeto en una transmisión de video, se necesita una forma fácil de almacenar información sobre el objeto en sí, que incluye: (a) es ID de objeto, (b) son centroides anteriores (por lo que se puede calcular fácilmente la dirección en la que se mueve el objeto) y (c) si el objeto ya se ha contado o no. Para lograr todos estos objetivos, se debe definir una instancia de TrackableObject: el archivo trackableobject.py del Anexo 7, permite realizar este proceso: La Tabla 21, muestra el código completo del archivo trackableobject.py.

Tabla 21.

Código usado para definir una Instancia.

```
64.  class TrackableObject:
65.      def __init__(self, objectID, centroid):
66.          # almacenar el ID del objeto, luego inicializar una lista de
     centroides
67.          # usando el centroide actual
68.          self.objectID = objectID
69.          self.centroids = [centroid]
70.
71.          # inicializar un booleano usado para indicar
72.          # si el objeto ya ha sido contado o no
73.          self.counted = False
```

Fuente: Autoría.

El constructor `TrackableObject` acepta un `objectID` + `centroid` y los almacena. La variable de centroides es una lista porque contendrá el historial de ubicación del centroide de un objeto. El constructor también se inicializa contado como Falso, lo que indica que el objeto aún no se ha contado.

El archivo people_counter.py, que realiza la detección y conteo de personas se encuentra en el Anexo 7.

3.3.2.1. Desarrollo de la interfaz de usuario (GUI).

Para desarrollar la interfaz de usuario del sistema, se utiliza la biblioteca Tkinter, que contiene un completo kit de herramientas estándar para desarrollar GUI's en Python. Cada función implementada en el sistema se mostrará de manera unificada a través de la interfaz de usuario. El script "Asistente Virtual APP.py" del Anexo 8, muestra el código completo para la creación de la interfaz de usuario.

En la Figura 64, se puede ver la interfaz de inicio del sistema y el conjunto de opciones disponibles en el sistema. El conjunto de opciones con las que cuenta la interfaz permite el acceso

inmediato a los diferentes procesos del sistema descritos anteriormente. La interfaz permite: Monitoreo de cámaras instaladas, Reconocimiento Facial, Agregar Usuarios a la base de datos, Verificar los usuarios existentes en la base de datos, ingresar al conjunto de datos y acceso a la documentación del proyecto que incluye el manual de usuario. La descripción de cada opción de la interfaz de usuario se encuentra en el Anexo 9.

Figura 64. Interfaz de Usuario - Página de Inicio

Fuente: Autoría.

Con respecto a la base de datos, Python cuenta con librerías destinadas a este proceso, en este caso se hace uso de la librería de MySQL para Python `mysql.connector` y `pymysql` tanto para sistemas operativos Windows y Linux respectivamente. Como se mencionó anteriormente, MySQL es un sistema de administración de bases de datos de código abierto, que comúnmente se

instala como parte de la pila LAMP (Linux, Apache, MySQL, PHP/Python/Perl). Para este proyecto se utiliza un modelo de base de datos relacional. Mediante lenguaje de programación SQL es posible crear tablas de datos para posteriormente integrar con la interfaz de usuario y Python.

En la Figura 65 se observa el proceso de elaboración de una base de datos con sus respectivas tablas y columnas.

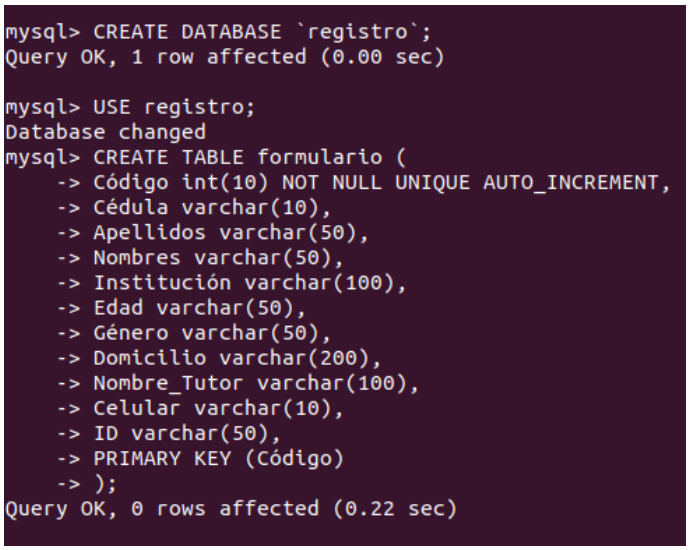


Figura 65. Creación de base de datos mediante lenguaje de programación SQL.

Figura: Autoría.

La base de datos lleva el nombre de registro, dentro de esta se crea una tabla llamada formulario en la cual se almacenarán los datos a las diferentes columnas: Código, Cedula, Apellidos, Nombres, Institución, Edad, Genero, Domicilio, Nombre Tutor, Celular y el ID.

En la Figura 67 se puede apreciar las consultas realizadas para mostrar los datos ingresados mediante la interfaz de Usuario.

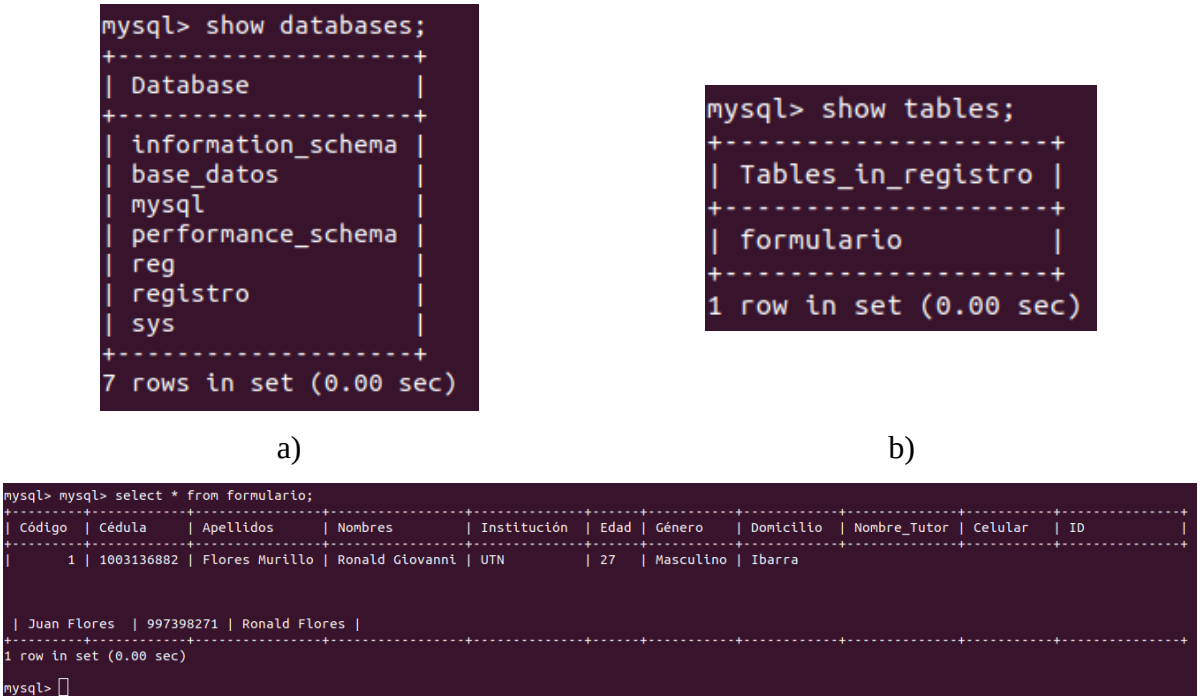


Figura 66. Consultas SQL a) muestra las bases de datos existentes, b) muestra las tablas de la base de datos seleccionada c) muestra la tabla con las respectivas columnas.

Figura: Autoría.

Generalmente SQL permite operaciones básicas de manipulación de datos denominadas operaciones CRUD (Create, Read, Update Delete), es decir, Crear, Leer, Actualizar y Borrar. Con la interfaz gráfica el usuario puede realizar estas operaciones sin necesidad de ingresar líneas de código.

La grafica de la Figura 67, muestra la interfaz en la que interactúa el usuario para realizar operaciones CRUD.

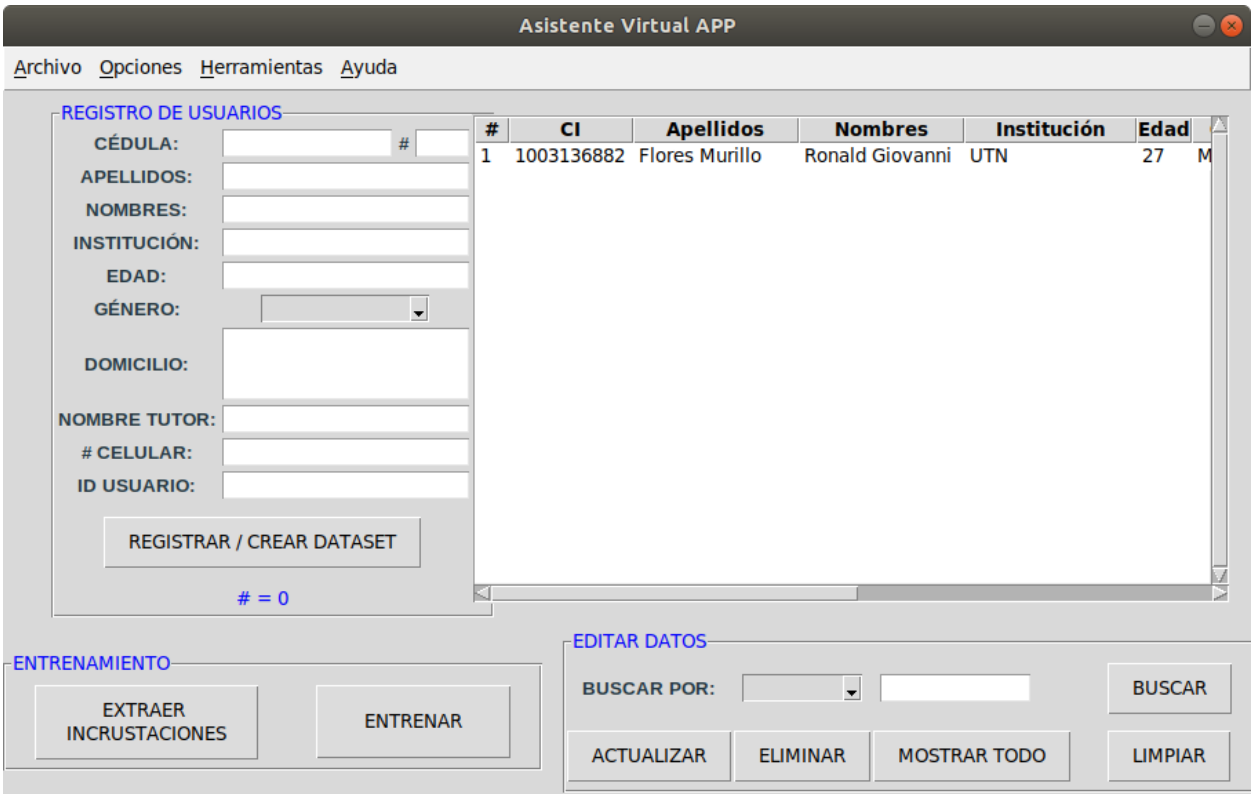


Figura 67. Interfaz gráfica para realizar operaciones CRUD.

Figura: Autoría.

4. CAPITULO IV: ANÁLISIS DE RESULTADOS

En el presente capítulo se indica las pruebas de funcionamiento del software y hardware del Asistente Virtual. Además, se determina la población objetivo del proyecto, con lo cual se realiza las pruebas de desempeño del sistema y finalmente se presentan resultados y/o conclusiones.

4.1. Análisis de población (muestra)

Para determinar la población a la cual se va a realizar las pruebas de funcionamiento se utiliza la Ecuación 10 la cual corresponde a la fórmula propuesta por (López, 2005).

$$n = \frac{N \times K^2 \times p \times q}{(e^2 \times (N-1) + K^2 \times p \times q)}$$

Ecuación 10. Cálculo del tamaño de muestra para una población finita.

Fuente: Adaptado de (López, 2005).

Donde:

$n =$ Tamaño de la muestra.

$N =$ Tamaño de la población.

$K^2 =$ Nivel de confianza.

$e =$ Error porcentual absoluto.

$p =$ Probabilidad a favor.

$q =$ Probabilidad en contra.

Parámetros utilizados:

Para este caso de estudio se tiene un tamaño de población de 12 individuos. Los parámetros considerados para la fórmula de Fisher y Navarro son los siguientes: el valor de N corresponde a 12 usuarios (número de pasajeros en promedio que alcanzan en una buseta de transporte escolar), para el nivel de confianza K se ha tomado un valor del 90% ($K = 1$) esperando que únicamente el 10% de los resultados obtenidos no correspondan a información verídica; tanto para el valor de la probabilidad a favor p como el valor de la probabilidad en contra q se estableció el valor de 0.5 ya que aún no se conoce la efectividad y/o fracaso del sistema en diversas condiciones de iluminación y poses de los individuos. El error porcentual absoluto e con un valor de 0.10 teniendo un porcentaje de respuestas correctas del ±10% de los resultados totales obtenidos. Aplicando la Ecuación 10. Se obtiene el siguiente resultado:

$$n = \frac{12 \times 1^2 \times 0.5 \times 0.5}{(0.1^2 \times (12 - 1) + 1^2 \times 0.5 \times 0.5)} \cong 8$$

Con el resultado obtenido se procede realizar las pruebas a un numero de 8 individuos para posteriormente analizar los resultados. Sin embargo, debido a la situación sanitaria, es difícil reunir a tal cantidad de personas en un solo sitio. Por lo tanto, las pruebas se realizarán a un grupo de personas que se encuentren en un mismo circulo y que no sean vulnerables.

4.2. Fase 3: Implementación y Testeo

Antes de desarrollar las pruebas del sistema, es necesario establecer el lugar conveniente para colocar los elementos del sistema. En la Figura 68, se muestra un esquema de la disposición de los elementos que componen el proyecto; cumpliendo con los requerimientos del sistema, los elementos deben estar ubicados lo más protegidos posibles sin que afecte su propósito, por lo tanto, el componente principal se ubica en la cabina del chofer, justo en el tablero principal, por otro lado

las cámaras de vigilancia se ubican; una apuntando hacia la puerta de entrada del recorrido y la otra direccionada hacia el corredor del vehículo.

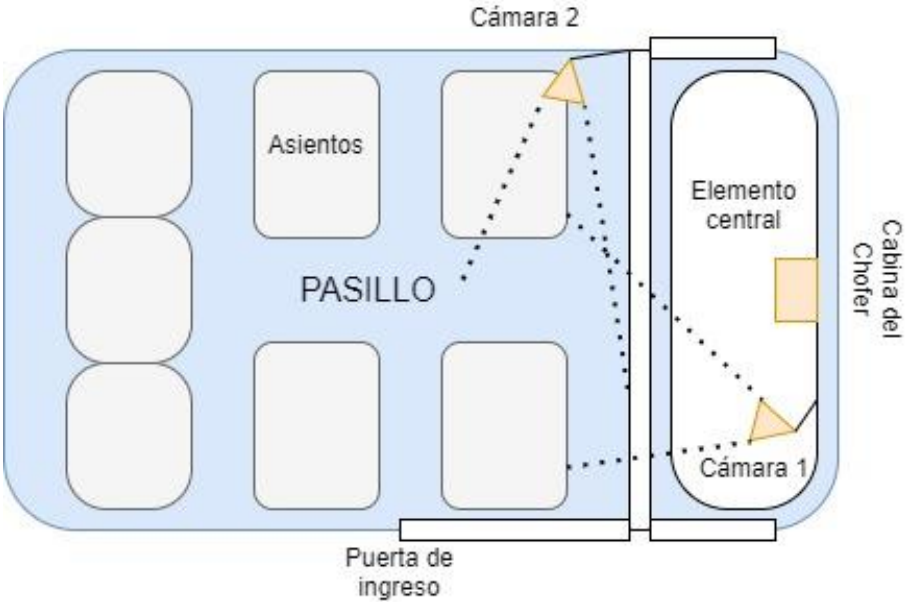


Figura 68. Ubicación del elemento central y las cámaras de vigilancia.

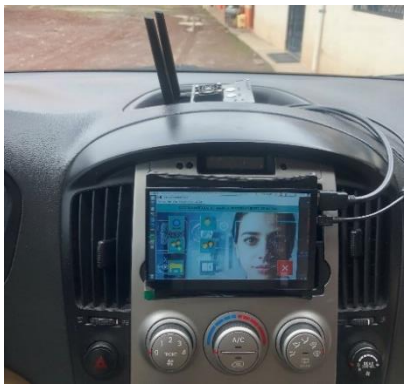
Fuente: Autoría.

El vehículo utilizado para las pruebas de funcionamiento es un Hyundai H1 tipo furgoneta con capacidad para 8 pasajeros. En la Figura 69, se puede apreciar el tipo de vehículo, así como también los elementos del proyecto ya incorporados.

(a)

(b)

(c)



(d)

Figura 69. Implementación de los diferentes elementos que conforman el proyecto. (a) Tipo de Vehículo, (b) Cámara 2, (c) Cámara 1, (d) Elemento central y dispositivos de acceso a la Interfaz de usuario.

Fuente: Autoría.

4.2.1. Funcionamiento General del Sistema.

En este apartado se evidencia el funcionamiento de los diferentes módulos que conforman el sistema. El asistente virtual en su primera etapa empieza adquiriendo los datos de los usuarios para posteriormente procesarlos. Esta etapa se encarga de recolectar fotos faciales para construir un conjunto de datos de entrenamiento. Para este proceso se ha hecho uso de la cámara de un Smartphone. Las características generales del Smartphone se describen en la Tabla 22.

Tabla 22.

Especificaciones técnicas del Smartphone.

ESPECIFICACIONES	XIAOMI POCO M3
Pantalla	6,53” FullHD+, Contraste 1.500:1, Brillo 400 nits, Gorilla Glass 3
Procesador	Qualcomm Snapdragon 662 GPU Adreno 610
RAM / Almacenamiento	4 GB LPDDR4X /
Filtro IR integrado:	48 MP f/1.79, 2 MP macro f/2.4, 2 MP profundidad f/2.4

Fuente: Adaptado de (*Xiaomi POCO M3 , Análisis: Review Con Características, Precio y Especificaciones*, n.d.).

Se captura alrededor de 300 imágenes faciales, las cuales se almacenarán en la base de datos del servidor local, las imágenes se guardarán en carpetas con su respectiva etiqueta. En la Figura 70, se observa la manera y el tipo de imágenes que se captura.

(a) (b) (c) (d)

Figura 70. Captura de imágenes faciales. a) de frente, b) perfil izquierdo, c) perfil derecho, d) rostro hacia arriba.

Fuente: Autoría.

La interfaz de usuario también permite la creación de una base de datos de imágenes por cada usuario. Para empezar la captura de imágenes faciales es necesario llenar el formulario que consta de datos personales del usuario y de su tutor. Con el formulario lleno se registran los datos y se procede a la captura de imágenes. Este proceso se realiza para cada usuario y las capturas se guardan en una carpeta con el ID asignado a cada individuo. En la Figura 71, se aprecia el formulario y la captura de imágenes.

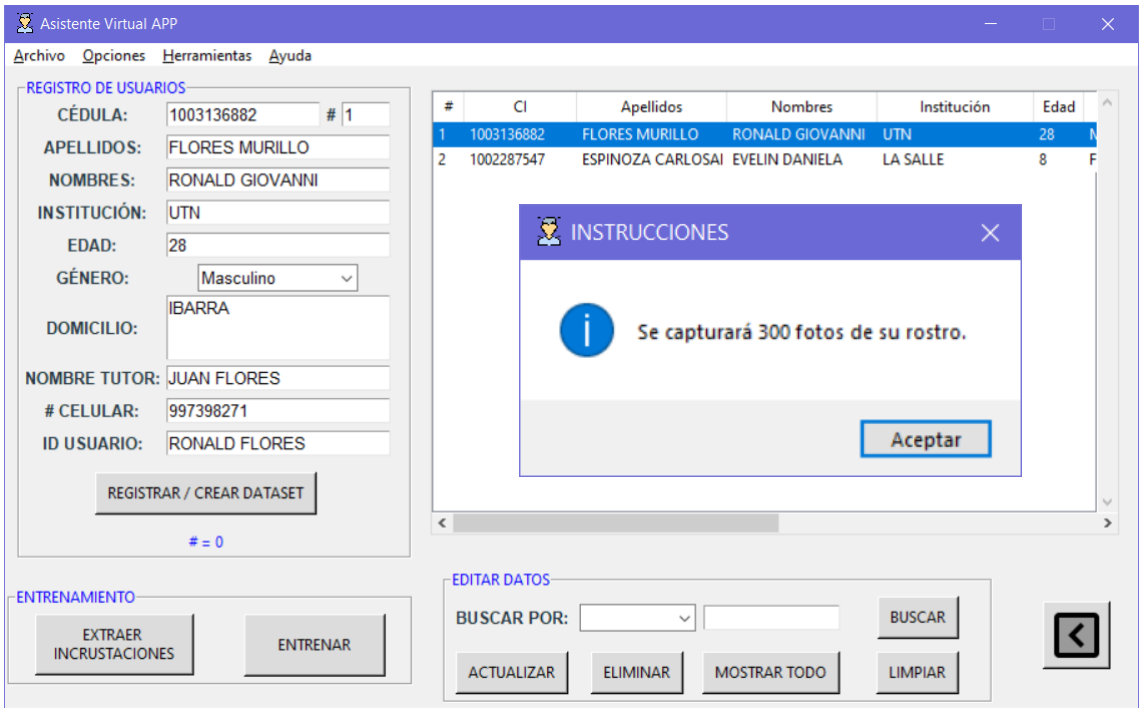


Figura 71. Interfaz de Usuario, ingreso de datos personales para el registro de Usuarios.

Fuente: Autoría.

Para cada imagen de entrada, se aplica la detección de rostros para detectar la ubicación de un rostro en la imagen; se extrae la detección con la mayor confianza, se filtra las detecciones débiles asegurándose de que la "confianza" sea mayor que la confianza mínima, se calcula las coordenadas (x, y) del cuadro delimitador del objeto y se dibuja un cuadro delimitador de la cara junto con la probabilidad asociada. En la Figura 72, se aprecia este procedimiento.

(a) (b)

Figura 72. (a) Detección de rostro en una imagen, 99.99% de confianza, (b) Directorio de imágenes almacenadas.

Fuente: Autoría.

Opcionalmente, se calcula los puntos de referencia faciales, lo que permite preprocesar y alinear el rostro. En la Figura 73, se muestra este proceso.

(a) (b)

Figura 73. (a) Extracción de puntos de referencia faciales, (b) Extracción de ROI.

Fuente: Autoría.

Una vez identificadas y procesadas las caras de cada imagen, el siguiente paso consiste en obtener una transformación numérica que sea capaz de representar los aspectos característicos de cada una. Al vector numérico resultante se le conoce como embedding o encoding. La Figura 74, muestra el resultado de este proceso.

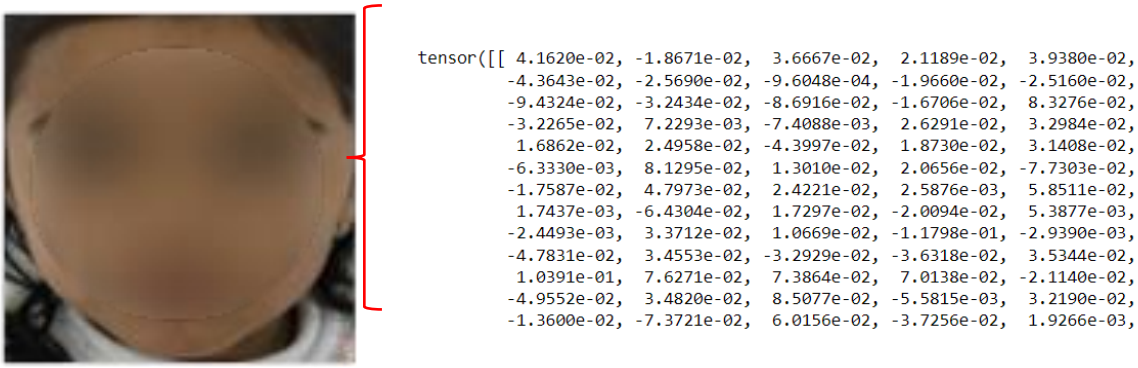


Figura 74. Vector numérico resultante al aplicar un modelo deep learning que genera un embedding de 128 valores.

Fuente: Autoría.

El objetivo de obtener una representación numérica de las caras (embeddings) es poder cuantificar similitudes entre ellas. Dos formas de calcular esta similitud es utilizando la distancia euclídea o la distancia coseno entre embeddings. Cuanto menor es la distancia, mayor la similitud de las caras. El proceso de detección de caras y extracción de incrustaciones de rostros se repite para todas y cada una de las imágenes de conjunto de datos. Todo lo que queda cuando finaliza el ciclo es volcar los datos en el disco.

Se agrega el nombre y los datos incorporados a un diccionario para posteriormente serializarlos en un archivo pickle. En la Figura 75 se puede observar que se ha extraído 1210 incrustaciones de caras, una para cada una de las imágenes (301 por clase) en el conjunto de datos de caras de entrada.

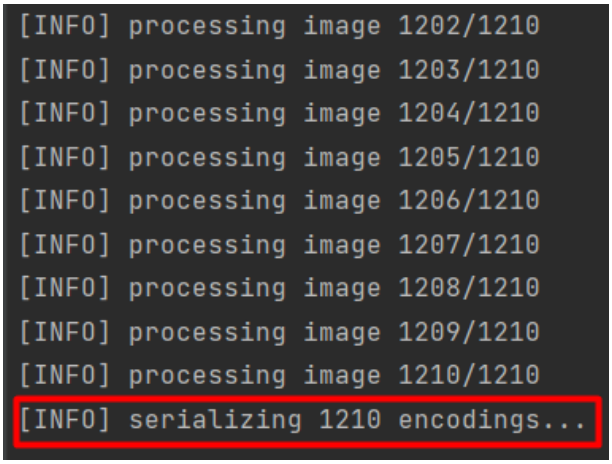
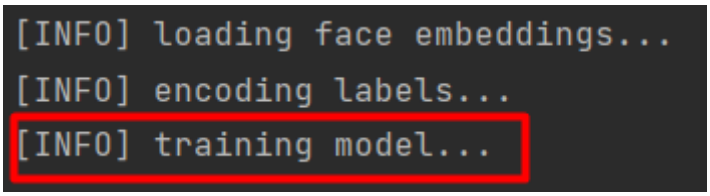


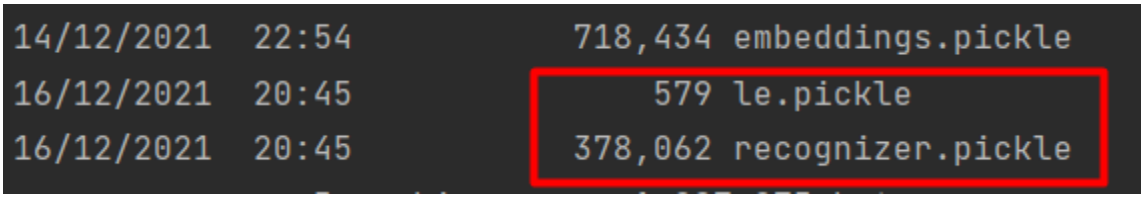
Figura 75. Resultado de la extracción de incrustaciones, 1210 incrustaciones de 301 de cada clase.

Fuente: Autoría.

En este punto, se genera un archivo con extensión pickle que contiene las incrustaciones serializadas, este archivo es usado para entrenar el modelo de aprendizaje automático SVM, además de las incorporaciones. Después de entrenar el modelo, se genera el modelo y el codificador de etiquetas en el disco como archivos pickle.



(a)



(b)

Figura 76. (a) Proceso de entrenamiento del modelo SVM, (b) archivos resultantes del entrenamiento.

Fuente: Autoría.

En la Figura 76, se observa que el SVM ha sido entrenado en las incrustaciones y tanto el SVM en sí, como la codificación de las etiquetas se han escrito en el disco, lo que permite aplicar en las imágenes de entrada y en el video.

El entrenamiento es el último paso previo al reconocimiento facial en imágenes y en video en tiempo real. Para ejecutar este proceso se necesita cargar los tres modelos del disco a la memoria: (a) Detector: Modelo de Caffe DL previamente entrenado para detectar en qué parte de la imagen están las caras, (b) Embedder: Modelo Torch DL previamente entrenado para calcular incrustaciones faciales 128-D y (c) Reconocedor: Modelo de reconocimiento facial Linear SVM.

También se carga el codificador de etiquetas que contiene los nombres de las personas que el modelo puede reconocer. En la Figura 77, se observa el resultado del Reconocimiento facial.

Figura 77. Reconocimiento Facial, con las respectivas etiquetas de cada individuo.

Fuente: Autoría.

4.2.2. Validación y métricas de eficiencia del sistema (Testeo)

Una vez completada la fase de implementación del sistema, se ejecuta la fase de testeo. Para comprobar la eficiencia del proyecto es necesario comparar si el producto final cumple con el rendimiento esperado; para esto se verifica si el resultado proporcionado por el sistema es correcto y válido, y se coteja con las necesidades y requisitos del usuario.

La validación de un sistema inteligente se realiza en torno a diferentes consideraciones; se tienen en cuenta la configuración de ambientes controlados y no controlados. En este caso, los ambientes controlados se caracterizan por cambios mínimos de iluminación y escaso personal; por el contrario, en los ambientes no controlados se presentan diferentes cambios de iluminación, varia la distancia, movimientos bruscos, etc.

Para evaluar el desempeño de clasificación del sistema se utiliza las matrices de confusión, las matrices proporcionan una idea de cómo está clasificando el sistema, a partir de un conteo de aciertos y errores de cada una de las clases. La matriz de confusión de la Figura 78, considera los siguientes índices: verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN).

		Clasificador	
		+	-
Valor real	+	TP	FN
	-	FP	TN

Figura 78. Matriz de confusión

Fuente: Adaptado de (Koldo, 2018)

Donde:

- TP: Son el número de verdaderos positivos, es decir, predicciones correctas para la clase positiva + (rostro).
- TN: Son el número de verdaderos negativos, es decir, predicciones correctas para la clase negativa – (no rostro).
- FP: Son el número de falsos positivos, es decir, una predicción positiva cuando realmente debería ser negativo.
- FN: Son el número de falsos negativos, es decir, una predicción negativa cuando realmente debería ser positiva.

Métricas de eficiencia

Se describen las distintas métricas utilizadas para medir la eficiencia del sistema a partir de la matriz de confusión:

- **Precisión**. - Es la proporción del número total de predicciones que fueron correctamente clasificadas, basándose en los verdaderos positivos y negativos frente a todas las detecciones del sistema. La precisión se calcula mediante la Ecuación 11:

$$Pr = \frac{TP + TN}{Total}$$

Ecuación 11. Formula de la Precisión.

Fuente: Adaptado de

- **Tasa de error. -** Es la proporción del número total de predicciones que fueron incorrectamente clasificadas, basándose en los falsos positivos y negativos frente a todas las detecciones del sistema. La tasa de error se calcula mediante la Ecuación 12:

$$Er = \frac{FP + FN}{Total}$$

Ecuación 12. Fórmula de la tasa de error.

Fuente: Adaptado de (Koldo, 2018)

- **Sensibilidad**. - Tasa de verdaderos positivos obtenidos por el sistema. Proporciona la probabilidad de que, dada una observación realmente positiva, el modelo la clasifique así. La sensibilidad se calcula mediante la Ecuación 13:

$$Rec = \frac{TP}{TP + FN}$$

Ecuación 13. Fórmula de la sensibilidad.

Fuente: Adaptado de (Koldo, 2018)

- **Especificidad**. - La especificidad es el número correcto de detecciones negativas obtenidas por el sistema. Proporciona la probabilidad de que, dada una observación realmente negativa, el modelo la clasifique así. La especificidad se calcula mediante la Ecuación 14:

$$Esp = \frac{TN}{TN + FP}$$

Ecuación 14. Fórmula de la especificidad.

Fuente: Adaptado de (Koldo, 2018)

4.3. Eficiencia del sistema

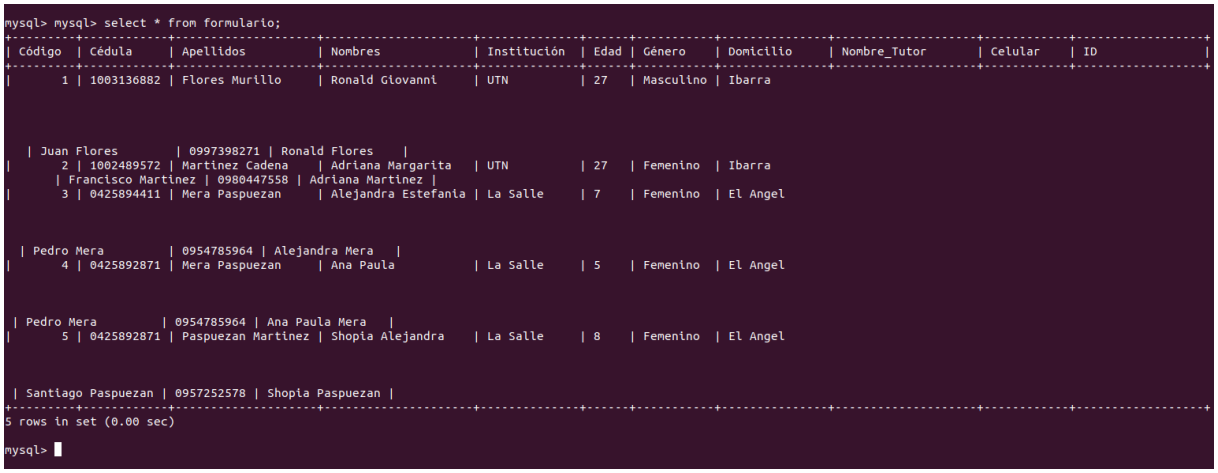
Es preciso recalcar que, la precisión y el rendimiento de OpenFace en comparación con otras técnicas de reconocimiento facial fueron evaluados preliminarmente en las secciones 3.3.2.1.4 y 3.3.2.1.5, donde se lograron resultados de precisión alentadores sobre conjuntos de datos de entrenamiento como LFW. Llegando a la conclusión de que, en un ambiente de pruebas

de laboratorio, los modelos alcanzan un gran desempeño y tasas de éxito aceptables. Sin embargo, las pruebas mencionadas, simplemente revelan un aproximado del desempeño del sistema en condiciones de prueba favorables (proceso offline en entorno controlado), difiriendo de una aplicación de la vida real (proceso online en entorno no controlado). Esta sección analiza el sistema de manera cualitativa y cuantitativa empleando las métricas definidas en la sección 4.2.2.

Para el análisis, se ha establecido un umbral de predicción, que limita personas conocidas y desconocidas en el sistema. El clasificador del sistema ofrece un valor de predicción alto para individuos que se encuentran registrados (conocidos) y una predicción muy baja para individuos no registrados (desconocidos). Los valores del umbral dependerán de la calidad de las imágenes obtenidas en tiempo real en cada situación.

4.3.1. Análisis Cualitativo y Cuantitativo

Se considera dos tipos de ambientes: ambiente controlado y ambiente no controlado. Como punto de partida para ambas situaciones, se recopilo fotos de rostros y la información de cada individuo. La base de datos se conforma con 300 muestras de 5 individuos, debidamente ordenadas en directorios para cada clase (sujeto). Las fotografías se encuentran almacenadas en formato “.jpg” con una resolución 1080p (1920x1080) para el posterior proceso de alineamiento. El registro de la información personal de cada individuo de la base de datos se muestra en la Figura 79.



```
mysql> mysql> select * from formulario;
+---------+------------+--------------------+--------------------+--------------+------+-----------+--------------+--------------------+------------+-----------------+
| Código  | Cédula     | Apellidos          | Nombres            | Institución  | Edad | Género    | Domicilio    | Nombre_Tutor       | Celular    | ID              |
+---------+------------+--------------------+--------------------+--------------+------+-----------+--------------+--------------------+------------+-----------------+
|       1 | 1003136882 | Flores Murillo     | Ronald Giovanni    | UTN          | 27   | Masculino | Ibarra
   | Juan Flores        | 0997398271 | Ronald Flores    |
|       2 | 1002489572 | Martinez Cadena    | Adriana Margarita  | UTN          | 27   | Femenino  | Ibarra
      | Francisco Martinez | 0980447558 | Adriana Martinez |
|       3 | 0425894411 | Mera Paspuezan     | Alejandra Estefania | La Salle    | 7    | Femenino  | El Angel
  | Pedro Mera         | 0954785964 | Alejandra Mera   |
|       4 | 0425892871 | Mera Paspuezan     | Ana Paula          | La Salle     | 5    | Femenino  | El Angel
 | Pedro Mera         | 0954785964 | Ana Paula Mera   |
|       5 | 0425892871 | Paspuezan Martinez | Shopia Alejandra   | La Salle     | 8    | Femenino  | El Angel
 | Santiago Paspuezan | 0957252578 | Shopia Paspuezan |
+---------+------------+--------------------+--------------------+--------------+------+-----------+--------------+--------------------+------------+-----------------+
5 rows in set (0.00 sec)

mysql>
```

Figura 79. Registro de usuarios en la base de datos.

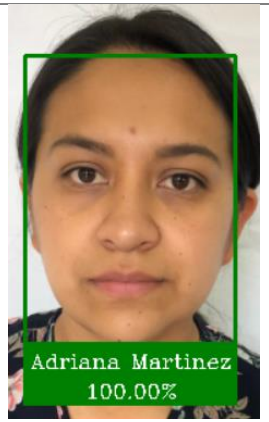
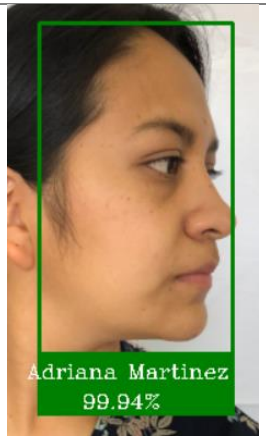
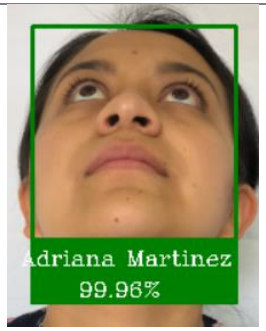
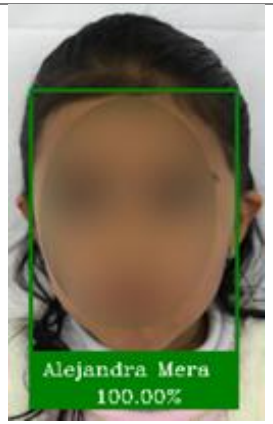
Fuente: Autoría.

4.3.1.1. Pruebas en Ambiente Real.

Para el desarrollo de este tipo de pruebas se ha reunido a 5 individuos en un lugar con óptimas condiciones de iluminación y con leves cambios de postura facial. El umbral de predicción del clasificador se ha establecido en un valor alto de 0.5 para este tipo de prueba. Los resultados obtenidos con el primer grupo de individuos se presentan en la Tabla 23:

Tabla 23.

Verificación Facial Individual

#	Imagen Procesada (SALIDA)	N° rostros detectados	Verificación del rostro en diferente posición			Etiqueta Correcta	Porcentaje de verificación alcanzado
			Izquierdo	Derecho	Frontal		
1		1				OK	99.96 %
2		1				OK	99.89 %

3		1				OK	99.78 %
4		1				OK	100 %
5		1				OK	100 %

Fuente: Autoría.

En la Tabla 23, se muestran los resultados finales alcanzados por el sistema para cada individuo registrado, donde se puede apreciar el cuadro delimitador de la región de interés (ROI) del rostro, el nombre del individuo en la parte inferior; la cantidad de rostros detectados, el porcentaje de detección en diferentes posiciones del rostro, además del porcentaje total alcanzado.

Continuando con la evaluación del sistema, se estableció un grupo de sujetos, donde se puede corroborar el desempeño del sistema con más de un individuo.

- Grupo Nº 1

El primer grupo de personas está compuesto por 2 individuos (clases) (Figura 80).

Figura 80. Grupo Nº1 de pruebas.

Fuente: Autoría.

Para el primer subgrupo se ha considerado las distancias de 1, 3 metros desde la cámara a la posición de los individuos. La Figura 80 muestra la disposición de los individuos; uno ingresando al vehículo escolar y el otro atrás del primer individuo fuera del vehículo.

- Grupo Nº 2

El segundo grupo de personas está compuesto por 3 individuos (clases) (Figura 81).

Figura 81. Grupo Nº2 de pruebas.

Fuente: Autoría.

Para el segundo subgrupo se ha considerado la distancia de 2 metros desde la cámara a la posición de los individuos. La Figura 81 muestra la disposición de los individuos en un sector al aire libre y con un abuena iluminación.

- Grupo Nº 3

El tercer grupo de personas está compuesto por 4 individuos (clases) (Figura 82).

Figura 82. Grupo Nº3 de pruebas.

Fuente: Autoría.

Para el tercer subgrupo se ha considerado las distancias de 1.5, 1.7, 2 y 2.5 metros desde la cámara a la posición de los individuos.

La Figura 82 muestra la disposición de los individuos dentro de la buseta. Como se puede observar la primera fila de asientos está ocupado por dos individuos, de los cuales el de la izquierda de la imagen posee un 99.23 % de coincidencia a diferencia del otro usuario que obtuvo un 42.52% de coincidencia. El porcentaje más bajo corresponde al nivel de iluminación que recibe el sujeto y a la posición del rostro.

El individuo de la fila 2 señala un 64.60 % de coincidencia a 2 m de la cámara, de igual manera este porcentaje se ve afectado por variación de la iluminación, sin embargo, el porcentaje sube por la posición del rostro.

La tercera fila de asientos muestra una persona con un 87.70 % de coincidencia, lo que indica que a 2.5m de la cámara, el sistema es capaz de reconocer un rostro con un alto grado de confiabilidad, pero esto dependerá mucho de la iluminación, calidad de la imagen y posición del rostro.

En cada situación el sistema logra detectar y reconocer de manera correcta en tiempo real, considerando cambios de expresión, pose, y un cambio de iluminación mínimo. Los resultados alcanzados se pueden apreciar en la Tabla 24.

Tabla 24.

Evaluación de resultados en entorno controlado con 5 personas.

CLASIFICADOR (SVM)		GRUPO 1		GRUPO 2		GRUPO 3	
		cara: etiqueta (+)	cara: no etiqueta (-)	cara: etiqueta (+)	cara: no etiqueta (-)	cara: etiqueta (+)	cara: no etiqueta (-)
VALOR REAL	cara: etiqueta (+)	2	0	3	0	4	0
	no cara: no etiqueta (-)	0	0	0	0	0	0

MÉTRICAS	**Valores obtenidos GRUPO 1**	**Valores obtenidos GRUPO 2**	**Valores obtenidos GRUPO 3**
Precisión (Pr)	100 %	100 %	100 %
Tasa de error (Er)	0 %	0 %	0 %
Sensibilidad o Recall (Rec)	100 %	100 %	100 %
Especificidad (Esp)	0 %	0 %	0 %

Fuente: Autoría.

Específicamente, la Tabla 24, muestra la eficiencia del clasificador SVM, cuando se ha entrenado sobre una base de datos de alrededor de 3000 imágenes de rostros. Las métricas calculadas ante cada grupo de individuos, indican un valor alto de verdaderos positivos, en varias de las pruebas se originó un falso negativo obtenido de manera casual, que puede deberse a algunas variantes, tales como: calidad de la óptica, calidad de imagen, aberraciones cromáticas o imperfecciones encontradas en la imagen, ángulo de captura del rostro o un cambio drástico de

apariencia facial, ocasionando que el clasificador interprete de forma errónea las facciones de un individuo. Cabe recalcar que en la mayoría del flujo de video este individuo fue reconocido. No obstante, estos resultados aún demuestran una alta correlación de los datos obtenidos frente a cada usuario (etiquetas correctas), además se demuestra en la Figura 83, que el detector de rostros identifica la zona ROI de forma fiable en todos los grupos.

(a)

(b)

Figura 83. (a) Detección de los puntos de referencia faciales en una imagen, (b) Extracción de la zona de interés ROI.

Fuente: Autoría.

4.4. Discusión de Resultados

El reconocimiento facial puede ser una tarea fácil para los humanos y las máquinas en casos simples como el abordado en la sección 4.3.1.1, donde se ha obtenido un desempeño de un 100%, pero con un grupo de personas muy pequeño, sin embargo, en escenarios desafiantes con un

número considerable de individuos, sin restricciones es una tarea difícil y propensa a errores. A pesar de que los modelos de aprendizaje profundo de última generación como las CNN´s, han logrado un rendimiento de clasificación a nivel humano en algunas tareas, actualmente no parecen ser capaces de tener un excelente desempeño sobre imágenes con bajas resoluciones; tal y como lo demuestra la Figura 84, que muestra un bajo porcentaje entre 42 y 64 % en algunos rostros.

Figura 84. Detecciones con bajo porcentaje.

Fuente: Autoría.

La distancia juega un papel importante al momento del desempeño del sistema, los mejores resultados en cuanto a la precisión se encuentran entre un rango de distancias cortas de 1.5 a 2.5 m, mientras que, en cualquier situación con distancias lejanas de 3.5 o más mts, resulta evidente el bajo desempeño del sistema, obteniendo así los peores resultados. Esto se debe a que, en varias ocasiones no se logra identificar al individuo o se genera una identificación errónea por la resolución promedio de la zona ROI. Otro factor adicional es que la cámara ofrece imágenes con ciertas deformaciones en los bordes, que consecuentemente deforma los rostros que se obtienen en esas zonas (Figura 85).

Figura 85. Falsas detecciones o simplemente el sistema no reconoce al individuo.

Fuente: Autoría.

Debido a los escasos detalles que existen de la forma en que funcionan internamente los sistemas de reconocimiento facial comerciales actuales, este estudio ha considerado la cantidad de identificaciones correctas con individuos aglomerados de forma simultánea en una colección de varios cuadros de video, lo cual ha arrojado porcentajes de precisión alentadores.

Es importante destacar que, las métricas obtenidas representan una aproximación de la efectividad del sistema en un ambiente sin restricciones, en el cual no se han considerado otras secuencias de video, donde la identificación facial sucede en algún/os frame/s de video de forma casual o persistente en algún/os individuo/s; esto sucede en gran proporción a cambios abruptos de pose y bajas resoluciones de la zona ROI.

Por otro lado, la eficacia del sistema depende en gran medida de las imágenes de alta calidad, la Tabla 25 muestra el tamaño de imágenes en pixeles y la precisión alcanzada en cada caso. Como se puede evidenciar, la tasa de precisión aumenta positivamente al elevar el tamaño de la imagen (resoluciones mayores a 80x80 pixeles); además, es preciso mencionar que el entrenamiento con imágenes faciales de menor resolución y el uso de técnicas SR (Super-Resolution) también podría mejorar los resultados obtenidos.

Tabla 25.

Precisión obtenida a diferentes cantidades de pixeles

# Pixeles	Precisión
1600	37.8%
6400	79.5%
14400	84.5%
25600	85.7%
65536	86.4%

Fuente: Adaptado de (Schroff, Kalenichenko, & Philbin, 2015)

4.5. Limitaciones del sistema

De acuerdo con el análisis cualitativo y cuantitativo realizado previamente, se presentan algunas de las conclusiones en base a las limitaciones encontradas en el sistema.

- **Resolución de la cámara**

Como en cualquier proyecto de visión artificial, la óptica juega el rol más importante. Esto se debe a que la imagen que se obtiene a través de los dispositivos ópticos debe poseer una alta calidad para obtener buenos resultados mediante el tratamiento con algoritmos de extracción de características profundas, como el desarrollado en este estudio. La resolución de los rostros capturados a distancias lejanas es demasiada baja, lo que conlleva a que el sistema no logre reconocer al/los individuos en la escena. La Figura 86 muestra las diferencias de calidad entre dos secciones de una imagen obtenida a diferentes distancias.

(a)

(b)

Figura 86. Capturas de rostro realizadas a una distancia (a) lejana, (b) cercana

Fuente: Autoría.

Por lo que se aprecia, en la Figura (86a) contiene a 4 individuos, de los cuales 3 han sido detectados correctamente a una distancia mayor a 1.5 mts, mientras que la Figura (86b) contiene a individuos, de los cuales 3 han sido detectados correctamente a una distancia de 2 mts; en la situación (86a) se denota una evidente pérdida de calidad de la imagen a largas distancias, y aunque

el detector facial aún puede lidiar con la baja calidad/resolución de la imagen, la identificación facial resulta ser efectiva solamente a cortas distancias, tal y como lo corrobora este estudio. Los resultados también pueden atribuirse a: aberraciones cromáticas o imperfecciones encontradas en la imagen.

- **Cambio de pose**

Según los análisis realizados, el cambio de pose no resulta ser un problema para la identificación facial en ambientes controlados, ya que todas las pruebas realizadas en la sección 4.3.1.1 Tabla 23, resultaron exitosas. La situación cambia en un entorno no controlado, debido a que existen muchas variaciones de postura de los individuos. Las pruebas realizadas demostraron efectividad cuando los rostros de los individuos estaban parcialmente de frente, mas no cuando se encontraban completamente de lado a cualquier distancia; esto es evidente, debido a que, la baja resolución de la imagen nuevamente presenta inconvenientes en poses laterales. Las variaciones laterales del rostro afectan el rendimiento de un sistema de reconocimiento facial, puesto que el entrenamiento del modelo de aprendizaje profundo fue alimentado por un conjunto de datos de entrenamiento que presenta imágenes faciales de forma frontal en su gran mayoría.

Para lidiar con el cambio de pose se podría aplicar un proceso de alineación facial completo, similar al propuesto en el estudio de Arcoverde (2014), el cual propone el uso de puntos de referencia de los ojos (landmarks) para lograr una alineación canónica de la zona ROI del rostro basada en la traslación, rotación y escala, de manera que los ojos encuentren una línea horizontal con coordenadas similares en el eje y. El proceso de alineación facial expuesto queda a consideración en un trabajo futuro. La Figura 87 muestra el bajo rendimiento del sistema a largas distancias y con variaciones de pose lateral.

Figura 87. Identificación facial con bajo desempeño

Fuente: Autoría.

4.6. Reporte del Sistema

La aplicación genera un reporte en formato ".xlsx" con las detecciones de personas conocidas, en el que se detalla: cedula, nombres, apellidos, fecha y hora de detección. La Figura 88 muestra el reporte generado por el sistema.

Sheet1

REPORTE DE USUARIOS		
FECHA DE DESCARGA:		17/02/22
CÉDULA	ID	HORA DE DETECCIÓN
1003136882	Ronald Flores	17/2/2022 17:31
1003136883	Ronald Flores	17/2/2022 17:32
1003136884	Ronald Flores	17/2/2022 17:32
1003136885	Ronald Flores	17/2/2022 17:32
1003136886	Ronald Flores	17/2/2022 17:33
1003136887	Ronald Flores	17/2/2022 17:34

Figura 88. Reporte de personas reconocidas por el sistema

Fuente: Autoría.

A continuación, en la Figura 89 se presenta el directorio con las capturas del rostro de los individuos reconocidos por el sistema (conocidos).

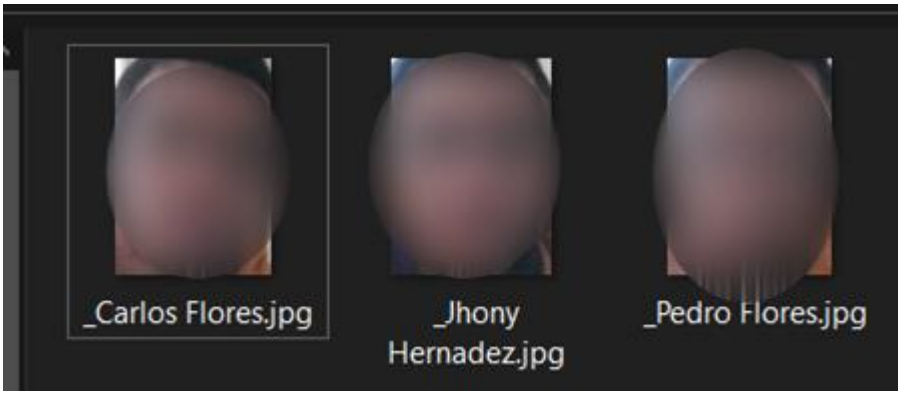


Figura 89. Capturas de fotografías de personas reconocidas por el sistema.

Fuente: Autoría.

Para personas desconocidas se genera un reporte de la misma manera, con los siguientes campos: ID de captura, fecha y hora de detección, además de una captura de rostro del individuo.

5. CAPÍTULO V. Conclusiones y Recomendaciones.

En esta sección se detallan las conclusiones y recomendaciones que se generaron durante el desarrollo del proyecto.

5.1. Conclusiones

El uso de técnicas de Visión Artificial para solventar problemas de reconocimiento facial y detección de personas en tiempo real y durante un lapso, resulta una tarea complicada y aún más cuando se requiere implementar estos procesos en un sistema embebido, sin embargo, existen técnicas simplificadas, específicamente orientadas a aplicaciones móviles que no consumen procesamiento y se ejecutan casi en tiempo real, es por eso que el reconocimiento facial se ha convertido en un área de estudio de gran interés en el campo de investigación de la inteligencia artificial representando una de las tecnologías más disruptivas y con un gran avance económico.

En cuanto al método de detección de rostros, se determinó que las Cascadas Haar de OpenCV son extremadamente rápidas pero propensas a falsos positivos y, en general, menos precisas que los detectores faciales basados en aprendizaje profundo, sin embargo, es una buena opción para dispositivos integrados como el Jetson Nano, por otro lado, la biblioteca dlib, implementa HOG + Linear SVM en una CPU, con resultados poco alentadores debido a la carga computacional, en cuanto a precisión, el detector de rostros MMOD CNN de dlib ofrece una mayor precisión pero con tiempos de ejecución más lento.

El detector facial DNN de OpenCV presenta un buen equilibrio. Como detector de rostros basado en aprendizaje profundo, este método es preciso, y dado que es una red poco profunda con una red troncal SSD, es fácilmente capaz de ejecutarse en tiempo real en una CPU. Además, dado que se puede usar el modelo con el módulo cv2.dnn de OpenCV, implica un aumento de velocidad

mediante el uso de una GPU o utilizando unidades externas de procesamiento de visión como el Movidius NCS o Google Coral para un dispositivo integrado.

El aprendizaje profundo ha impactado en casi todos los aspectos y subcampos de la informática incluyendo el reconocimiento facial. Mediante revisión literario y según el enfoque del proyecto se pudo determinar que, técnicas como LBP y Eigenfaces / Fisherfaces resultan poco precisas para el ámbito de nuestra aplicación; actualmente, las arquitecturas de redes neuronales especializadas y las técnicas de entrenamiento, incluidas las redes siamesas, los trillizos de imágenes y la pérdida de tripletes, permiten obtener una precisión de reconocimiento facial aceptable, para este caso, la perdida de tripletes resulto la mejor opción, ya que de esta manera, la red puede aprender a cuantificar rostros y devolver incrustaciones altamente robustas y discriminatorias adecuadas para el reconocimiento facial.

En el análisis de los requerimientos del sistema se tomó como referencia el estándar ISO/IEC/IEEE 29148, el mismo que contiene directrices para el proceso relacionado a la ingeniería de requisitos, dicho estándar permitió definir cada función que se requiere en el proyecto, las restricciones necesarias, y especificar los requisitos y funciones del sistema, además se pudo determinar los elementos específicos para cada etapa del proyecto así como también los requerimientos de software para un correcto desempeño del sistema.

Con el diseño del Asistente Virtual se logró demostrar la factibilidad/aplicabilidad del reconocimiento facial en aplicaciones de seguridad y verificación, implementadas en sistema embebido utilizando métodos basados en aprendizaje profundo; los resultados del sistema fueron alentadores en cuanto a la verificación de la identidad de cada usuario, además, se pudo evidenciar el aporte significativo al momento de brindar seguridad y monitoreo de los usuarios durante el trayecto.

5.2. Recomendaciones

En la fase de Requerimientos y Análisis es muy importante definir requerimientos específicos que debe tener el sistema en cuanto a software y hardware, ya que esta sección representa los lineamientos principales que rigen el desarrollo del sistema prototipo de reconocimiento facial.

En la etapa de testeo del sistema es recomendable analizar el sitio y ubicación de los elementos que conforman el sistema en la buseta; de esta manera se podrá garantizar el correcto funcionamiento del Asistente Virtual y la integridad de los diferentes dispositivos electrónicos.

Para lograr la mayor precisión el conjunto de datos debe contener suficientes muestras de variación y una buena representación de la cara. Si el conjunto de datos tiene menos datos para entrenar, se produce un desajuste; por el contrario, si contiene muchos datos, el modelo de entrenamiento se sobreajusta.

Antes de comenzar cualquier investigación en el marco de la inteligencia artificial es importante poseer todo el conjunto de herramientas de software y hardware, así como también de un conocimiento general de los principales algoritmos de aprendizaje supervisado y no supervisado.

Al existir muchas arquitecturas de redes neuronales convolucionales es recomendable emplear una arquitectura que no demande de muchos recursos computacionales y posea buenas referencias de éxito en ambientes móviles y con dispositivos embebidos.

Es recomendable el uso de forma experimental de diferentes formatos de imágenes y comprobar los niveles de precisión que conlleva usar cada uno, ya que formatos como JPG, JPG2000, entre otros, presentan perdidas de información al momento de procesar las imágenes.

BIBLIOGRAFÍA

A Neural Network Playground. (n.d.). Retrieved May 1, 2021, from https://playground.tensorflow.org

Abney, S. (2007). Semisupervised learning for computational linguistics. CRC Press.

Ahmad, R. (2019, March 5). A Take on H.O.G Feature Descriptor | by Rehan Ahmad | Analytics Vidhya | Medium. https://medium.com/analytics-vidhya/a-take-on-h-o-g-feature-descriptor-e839ebba1e52

Ahonen, T., Hadid, A., & Pietikäinen, M. (2004). Face recognition with local binary patterns. European Conference on Computer Vision, 469–481.

Amos, B., Ludwiczuk, B., & Satyanarayanan, M. (2016). OpenFace: A general-purpose face recognition library with mobile applications. http://cmusatyalab.github.io/openface/

Argente Villaplana, E. (2021). Lenguajes de programación.

Bhanu, B., & Kumar, A. (2017). Deep learning for biometrics. Springer.

Brownlee, J. (2019). Deep Learning for Computer Vision: Image Classification, Object Detection, and Face Recognition in Python. Machine Learning Mastery.

Cerna, L. R., Cámara-Chávez, G., & Menotti, D. (2013). Face detection: Histogram of oriented gradients and bag of feature method. Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), 1.

Coursera. (2021, June 24). What Is Python Used For? A Beginner's Guide | Coursera. https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

Dawson-Howe, K. (2014). A Practical Introduction to Computer Vision with OpenCV. Wiley.

del Toro Hernández, E., Cabrera Sarmiento, A. J., Sánchez Solano, S., & Cabrera Aldaya, A. (2012). Impacto de la memoria cache en la aceleración de la ejecución de algoritmo de detección de rostros en sistemas empotrados. Ingeniería Electrónica, Automática y Comunicaciones, 33(2), 57–71.

Dhingra, A. (2017). Face Identification and Clustering. ArXiv Preprint ArXiv:1704.08328.

Domínguez Pavón, S. (2017). Reconocimiento facial mediante el Análisis de Componentes Principales (PCA).

Elgendy, M. (2020). Deep Learning for Vision Systems (Simon and). Manning Publications.

Fernández Villán, A. (2019). Mastering OpenCV 4 with Python: A Practical Guide Covering Topics from Image Processing, Augmented Reality to Deep Learning with OpenCV 4 and Python 3.7. Packt Publishing.

Forsyth, D. A., & Ponce, J. (2003). A modern approach. Computer Vision: A Modern Approach, 17, 21–48.

Freiwald, W., Duchaine, B., & Yovel, G. (2016). Face processing systems: from neurons to real-world social perception. Annual Review of Neuroscience, 39, 325–346.

Géron, A. (2019). Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. O'Reilly Media.

Kumar Pal, S. (2019, September 9). Software Engineering | Classical Waterfall Model - GeeksforGeeks. https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/

Lazzeri, F. (2020). Aprendizaje profundo frente a aprendizaje automático - Azure Machine Learning | Microsoft Docs. https://docs.microsoft.com/es-es/azure/machine-learning/concept-deep-learning-vs-machine-learning

Lee, S. (2020, May 22). Understanding Face Detection with the Viola-Jones Object Detection Framework | by Socret Lee | Towards Data Science. https://towardsdatascience.com/understanding-face-detection-with-the-viola-jones-object-detection-framework-c55cc2a9da14

Liljeqvist, I. (2016, March 20). The Essence of Artificial Neural Networks | by Ivan Liljeqvist | Medium. https://medium.com/@ivanliljeqvist/the-essence-of-artificial-neural-networks-5de300c995d6

López, C. P. (2005). Muestreo estadístico: conceptos y problemás resueltos. Pearson Educación. https://books.google.com.ec/books?id=OP7xAAAACAAJ

Machine Learning Algorithms - Javatpoint. (n.d.). Retrieved April 27, 2021, from https://www.javatpoint.com/machine-learning-algorithms

Mandal, M. (2021, May 1). Convolutional Neural Networks (CNN) | by Manav Mandal | Apr, 2021 | Medium. https://manavmandal.medium.com/convolutional-neural-networks-cnn-d88e7f9329eb

Mittal, A. (2020, December 20). Haar Cascades, Explained. A brief introduction into Haar… | by Aditya Mittal | Analytics Vidhya | Medium. https://medium.com/analytics-vidhya/haar-cascades-explained-38210e57970d

Mueller, J. P., & Massaron, L. (2016). Machine learning for dummies. John Wiley & Sons.

Murugaiyan, D. (2012). International Journal of Information Technology and Business Management WATEERFALLVs V-MODEL Vs AGILE: A COMPARATIVE STUDY ON SDLC. 2(1). www.jitbm.com

Nagrath, P., Jain, R., Madan, A., Arora, R., Kataria, P., & Hemanth, J. (2021). SSDMNV2: A real time DNN-based face mask detection system using single shot multibox detector and MobileNetV2. Sustainable Cities and Society, 66, 102692. https://doi.org/10.1016/j.scs.2020.102692

Nagyfi, R. (2018, September 4). The differences between Artificial and Biological Neural Networks | by Richard Nagyfi | Towards Data Science. https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828b7

Parekh, M. (2019, July 16). A Brief Guide to Convolutional Neural Network(CNN) | by Manan Parekh | Nybles | Medium. https://medium.com/nybles/a-brief-guide-to-convolutional-neural-network-cnn-642f47e88ed4

Park, U. (2009). Face Recognition: face in video, age invariance, and facial marks. Citeseer.

Pedrycz, W., & Chen, S.-M. (2020). Deep Learning: Concepts and Architectures. Springer.

Peterson, Z. (2020, January 20). Computer Vision in Embedded Systems and AI Platforms | NWES Blog. https://www.nwengineeringllc.com/article/computer-vision-in-embedded-systems-and-ai-platforms.php

Pulli, K., Baksheev, A., Kornyakov, K., & Eruhimov, V. (2012). Real-time computer vision with OpenCV. Communications of the ACM, 55(6), 61–69.

Rosebrock, A. (2017). Deep Learning for Computer Vision with Python: Starter Bundle. PyImageSearch.

Salton do Prado, K. (2017, November 10). Face Recognition: Understanding LBPH Algorithm | by Kelvin Salton do Prado | Towards Data Science. https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 815–823.

Sereno Rodriguez, P. (2017). Reconocimiento de expresiones faciales mediante el uso de redes neuronales convolucionales. Universitat Politècnica de Catalunya.

Simeone, O. (2018). A very brief introduction to machine learning with applications to communication systems. IEEE Transactions on Cognitive Communications and Networking, 4(4), 648–664.

Sirkin, J. (2021, April). ¿Qué es SQL (Structured Query Language o Lenguaje de consultas estructuradas)? - Definición en WhatIs.com. https://searchdatacenter.techtarget.com/es/definicion/SQL-o-lenguaje-de-consultas-estructuradas

Sirovich, L., & Kirby, M. (n.d.). Low-dimensional procedure for the characterization of human faces (Vol. 4, Issue 3).

Taigman, Y., Yang, M., Ranzato, M., & Wolf, L. (2014). Deepface: Closing the gap to human-level performance in face verification. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1701–1708.

Thuis, C. (2018). SSD-Sface: Single shot multibox detector for small faces.

Trask, A. (2019). Grokking deep learning. Manning Publications Co.

Tucker, A. B. (2004). Computer science handbook. CRC press.

Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. Journal of Cognitive Neuroscience, 3(1), 71–86. https://doi.org/10.1162/jocn.1991.3.1.71

Vasilev, I., Slater, D., Spacagna, G., Roelants, P., & Zocca, V. (2019). Python Deep Learning: Exploring deep learning techniques and neural network architectures with Pytorch, Keras, and TensorFlow. Packt Publishing Ltd.

Verification and Validation - www.aiotframework.org. (2021, April 9). https://aiotframework.org/index.php?title=Verification_and_Validation#Authors_and_Contributors

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, 1, I–I.

Weng, L. (2017, October 29). Object Detection for Dummies Part 1: Gradient Vector, HOG, and SS. https://lilianweng.github.io/lil-log/2017/10/29/object-recognition-for-dummies-part-1.html#image-gradient-vector

Wiering, M., & Van Otterlo, M. (2012). Reinforcement learning (Vol. 12). Springer.

Xiaomi POCO M3 , análisis: review con características, precio y especificaciones. (n.d.). Retrieved December 1, 2021, from https://www.xataka.com/analisis/poco-m3-analisis-caracteristicas-precio-especificaciones

Zhang, K., Zhang, Z., Li, Z., & Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Processing Letters, 23(10), 1499–1503.

ANEXOS

ANEXO 1 (Encuesta a padres de familia)

1. ¿Considera factible el desarrollo de un prototipo con reconocimiento facial, que detecte la presencia de estudiantes (niños entre 3 y 8 años) en el transporte escolar durante todo el recorrido?

a) Si

b) No

2. ¿Considera que implementar un sistema de reconocimiento facial dentro de una buseta escolar ayudara a verificar la presencia e integridad de sus usuarios?

a) Si

b) No

3. En cuanto al tamaño del prototipo. ¿Cuál de las siguientes opciones considera usted que es la adecuada?

a) Debe ser pequeño

b) Debe ser mediano

c) Debe ser lo más compacto posible

d) El tamaño del prototipo me parece indiferente

4. En cuanto al consumo de energía. ¿Cuál de las siguientes opciones considera usted que es la adecuada?

a) Debe consumir poca energía

b) Debe consumir la energía necesaria

c) El consumo de energía me parece indiferente

5. En cuanto al tiempo de funcionamiento del prototipo. ¿Cuál de las siguientes opciones considera usted que es la adecuada?

a) Siempre debe funcionar

b) Debe funcionar solo cuando el recorrido se detenga

c) Debe funcionar solo cuando la puerta del recorrido se abra

6. En cuanto a la alimentación del prototipo. ¿Cuál de las siguientes opciones considera usted que es la adecuada?

a) Con la batería del carro

b) Fuente de poder individual (batería individual)

c) Con batería del carro y batería individual como respaldo

7. Para que el proceso de reconocimiento facial sea eficaz es necesario adquirir imágenes (fotos de todos los ángulos posibles) del rostro de cada usuario. ¿Aprobaría que el desarrollador del proyecto tome y almacene fotos con el rostro de cada usuario?

a) Si

b) No

8. Con el prototipo en funcionamiento, ¿Quiénes considera que deberían ser los responsables del mantenimiento y administración del prototipo?

a) Solo el desarrollador del prototipo

b) El conductor del recorrido

c) Los padres de familia

d) Los profesores de las instituciones educativas

9. En cuanto a las alertas. ¿Con qué frecuencia le gustaría recibir las notificaciones?

a) Solo cuando el usuario llegue a la institución.

b) Solo cuando el usuario regrese al lugar donde se lo recogió.

c) Las dos anteriores

10. ¿Qué tipo de mensajes de alerta considera usted que son los más apropiados?

a) Mensajes de texto con la confirmación de llegada

b) Mensajes con la imagen del usuario

c) Mensajes de texto con la imagen del usuario

11. En cuanto al manejo y acceso a los datos adquiridos y generados del prototipo. ¿Cuál de las siguientes opciones considera usted que es la adecuada?

a) Acceso solamente para el desarrollador del prototipo

b) Acceso solamente para el conductor del recorrido

c) Acceso solamente para los padres de familia

d) Acceso solamente para los profesores de las instituciones educativas

12. ¿Cuáles de las siguientes características considera usted que debe ser la más importante para el desarrollo del prototipo?

a) Seguridad de los datos

b) Precisión en el reconocimiento

c) Envió de mensajes o notificaciones

d) Tiempo de respuesta del sistema

e) Todas la anteriores

13. Considera usted que es importante la generación de reportes periódicos sobre el funcionamiento del proyecto

a) Si

b) No

ANEXO 2 (Resultados y Análisis de la encuesta)

Pregunta 1: En la primera pregunta realizada se obtuvo como resultado que el 100% de los encuestados manifiestan que es factible el desarrollo del proyecto propuesto. La Figura muestra la tabulación de los resultados a la primera pregunta.

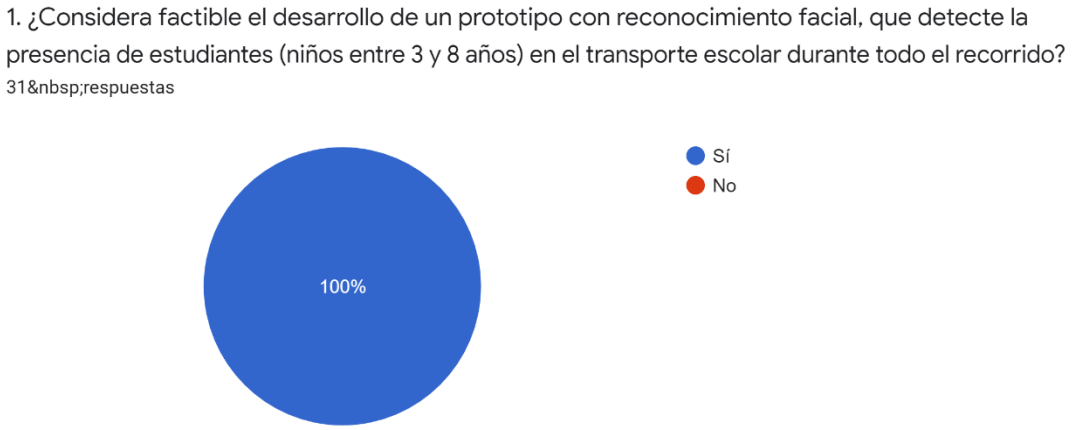


Figura. 1 Resultados de la primera pregunta de la encuesta

Fuente: Autoría

Pregunta 2: Los resultados obtenidos de la segunda pregunta muestran que el 90.3% de los encuestados considera que implementar un sistema de reconocimiento facial dentro de una buseta escolar ayudara a verificar la presencia e integridad de sus usuarios y el 9.7% piensa que no ayudara en nada. La Figura muestra la tabulación de los resultados a la segunda pregunta.

Figura. 2. Resultados de la segunda pregunta de la encuesta

Fuente: Autoría

Pregunta 3: Los resultados obtenidos de la tercera pregunta muestran que, en cuanto al tamaño del prototipo, el 45.2% de los encuestados considera que el tamaño del prototipo debe ser lo más compacto posible. La Figura muestra la tabulación de los resultados a la tercera pregunta.

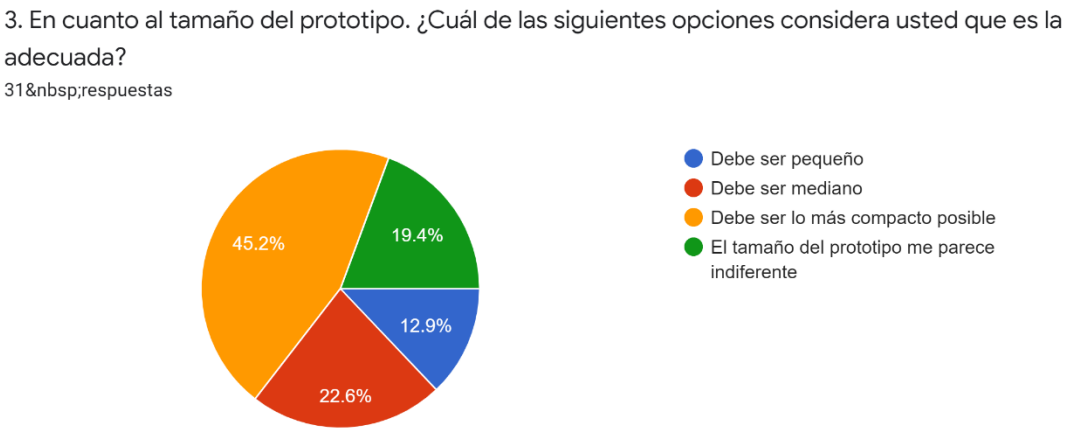


Figura. 3. Resultados de la tercera pregunta de la encuesta

Fuente: Autoría

Pregunta 4: Los resultados obtenidos de la cuarta pregunta muestran que, en cuanto al consumo de energía del prototipo, el 51.6% de los encuestados considera que el prototipo debe consumir la energía necesaria. La Figura muestra la tabulación de los resultados a la cuarta pregunta.

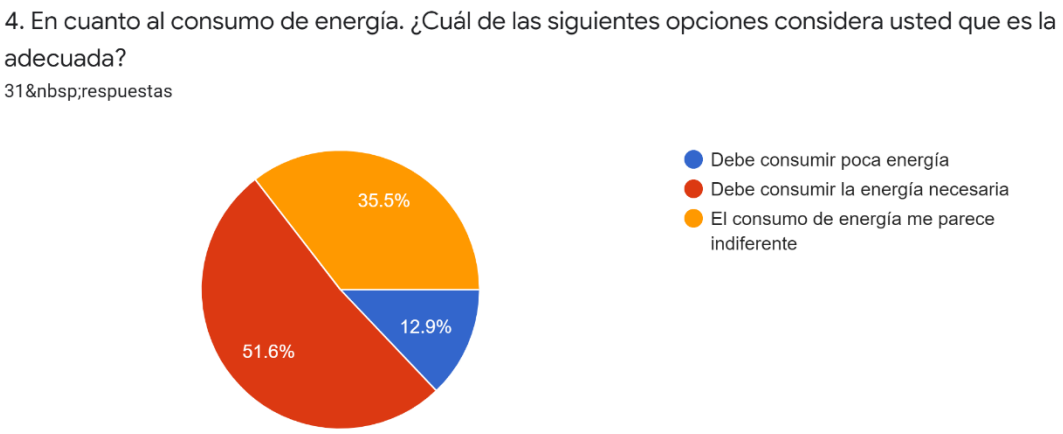


Figura. 4. Resultados de la cuarta pregunta de la encuesta

Fuente: Autoría

Pregunta 5: Los resultados obtenidos de la quinta pregunta muestran que, en cuanto al tiempo de funcionamiento del prototipo, la mayoría considera que el funcionamiento debe ser por intervalos, es decir, cuando la puerta del recorrido se habrá. Este requerimiento variara dependiendo de las pruebas de funcionamiento. La Figura muestra la tabulación de los resultados a la quinta pregunta.

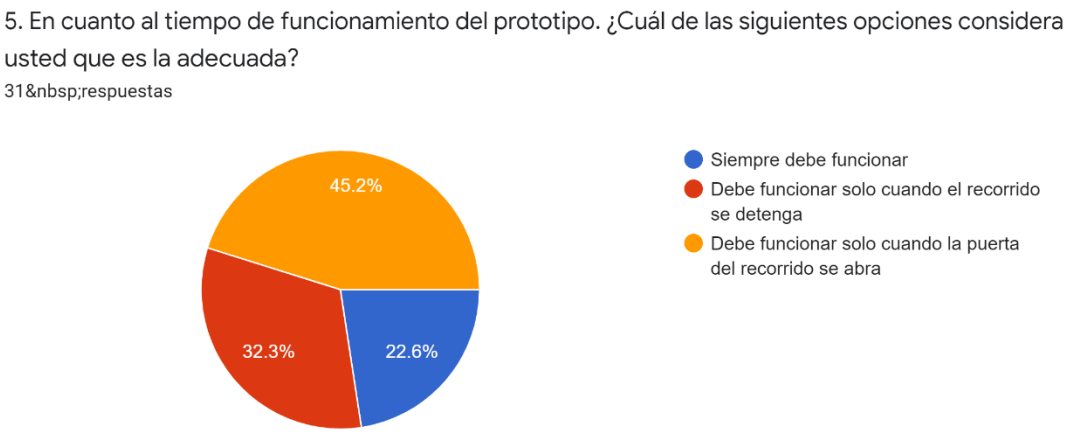


Figura. 5. Resultados de la quinta pregunta de la encuesta

Fuente: Autoría

Pregunta 6: Los resultados obtenidos de la sexta pregunta muestran que, en cuanto a la alimentación del prototipo, una gran mayoría de los encuestados considera que el prototipo debe alimentarse con la batería del carro y tener una batería adicional de respaldo. La Figura muestra la tabulación de los resultados a la sexta pregunta.

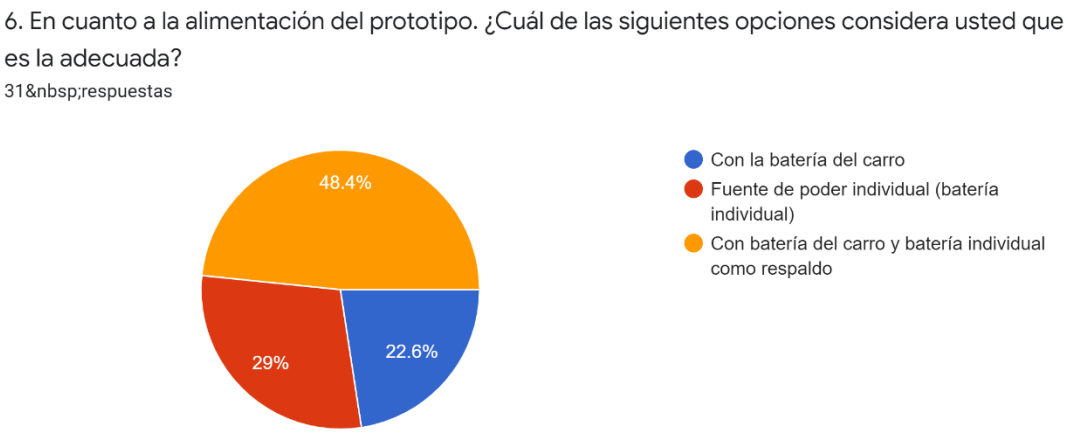


Figura. 6. Resultados de la sexta pregunta de la encuesta

Fuente: Autoría

Pregunta 7: Los resultados obtenidos de la séptima pregunta muestran que, el 100% de los encuestados autorizan que el desarrollador del proyecto tome y almacene fotos con el rostro de cada usuario. La Figura muestra la tabulación de los resultados a la séptima pregunta.

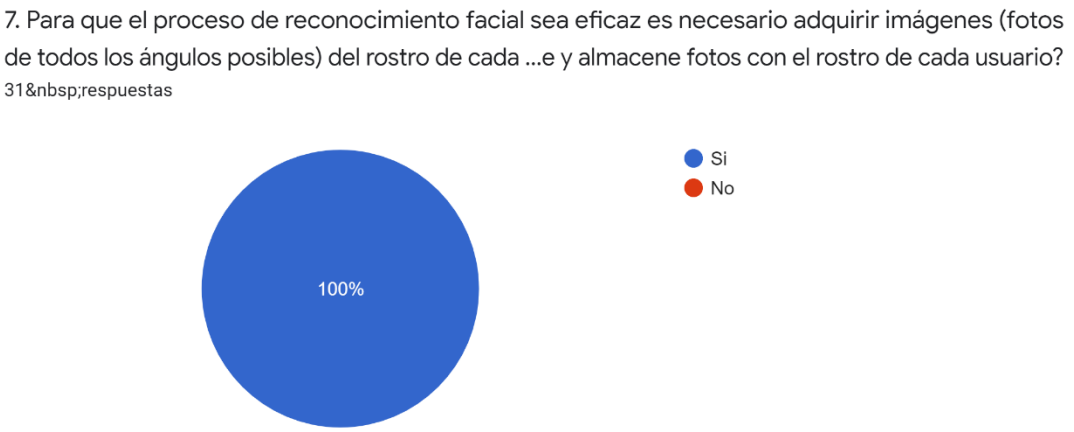


Figura. 7. Resultados de la séptima pregunta de la encuesta

Fuente: Autoría

Pregunta 8: Con los resultados obtenidos de la octava pregunta, se concluye que, solo el desarrollador del prototipo puede administrar y manipular el sistema. El resto de stakeholders no podrán realizar estas acciones a menos que el administrador las autorice. La Figura muestra la tabulación de los resultados a la octava pregunta.

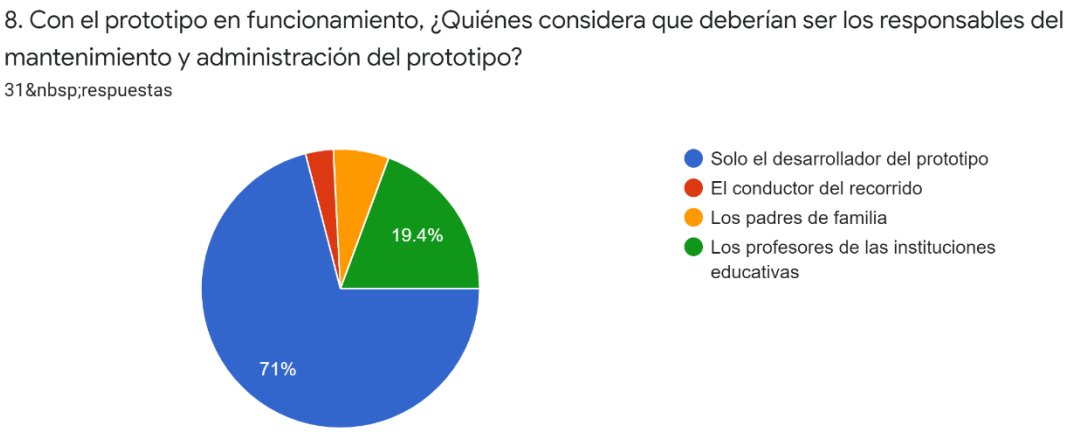


Figura. 8. Resultados de la octava pregunta de la encuesta

Fuente: Autoría

Pregunta 9: Los resultados obtenidos de la novena pregunta muestran que, en cuanto a las alertas que generara el prototipo, la mayoría de los encuestados prefiere que las notificaciones se envíen cuando el usuario llegue a la institución y cuando el usuario regrese al lugar de origen. La Figura muestra la tabulación de los resultados a la novena pregunta.

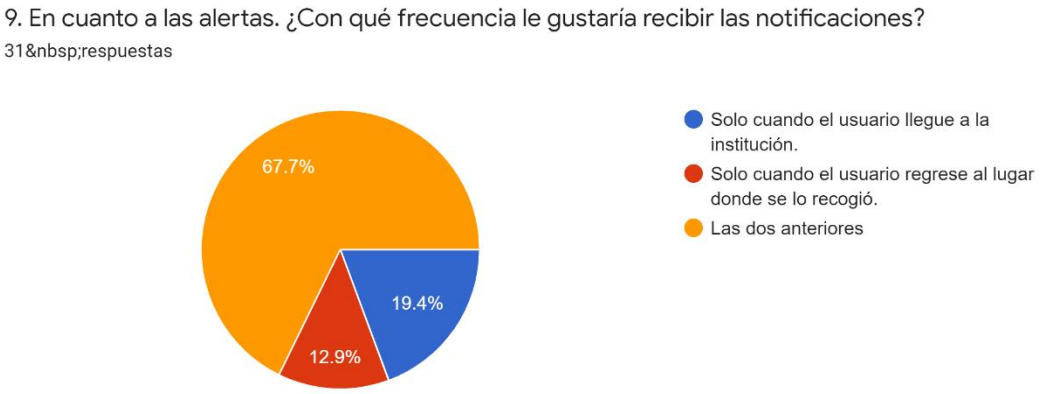


Figura. 9. Resultados de la novena pregunta de la encuesta

Fuente: Autoría

Pregunta 10: Los resultados obtenidos de la décima pregunta muestran que, en cuanto a tipo de mensajes de alerta, la mayoría de los encuestados prefiere que las notificaciones sean mensajes de texto. Este punto se evaluará en la fase de pruebas y se determinará si esta opción es factible o no. La Figura muestra la tabulación de los resultados a la décima pregunta.

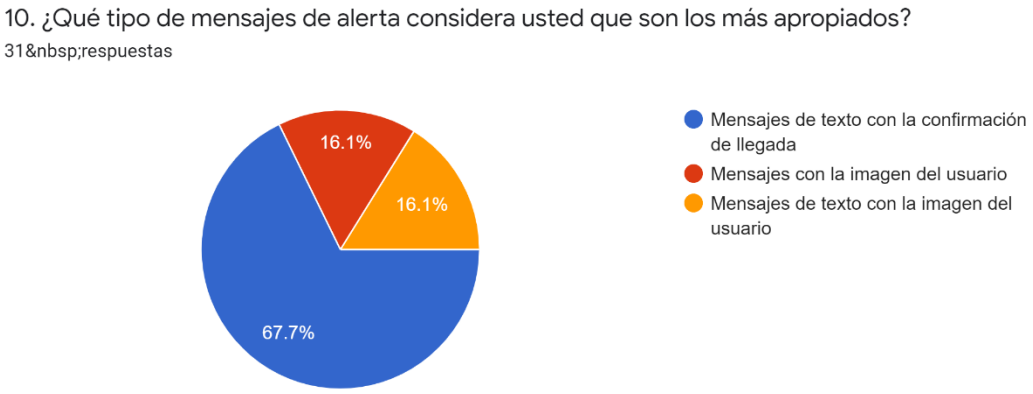


Figura. 10. Resultados de la décima pregunta de la encuesta

Fuente: Autoría

Pregunta 11: Los resultados obtenidos de la décimo primera pregunta muestran que, en cuanto a la seguridad y manejo de los datos obtenidos y de los datos que el prototipo generara, el 67.7% de los encuestados opina que solo el administrador del proyecto tenga acceso a estos datos. La Figura muestra la tabulación de los resultados a la décimo primera pregunta.

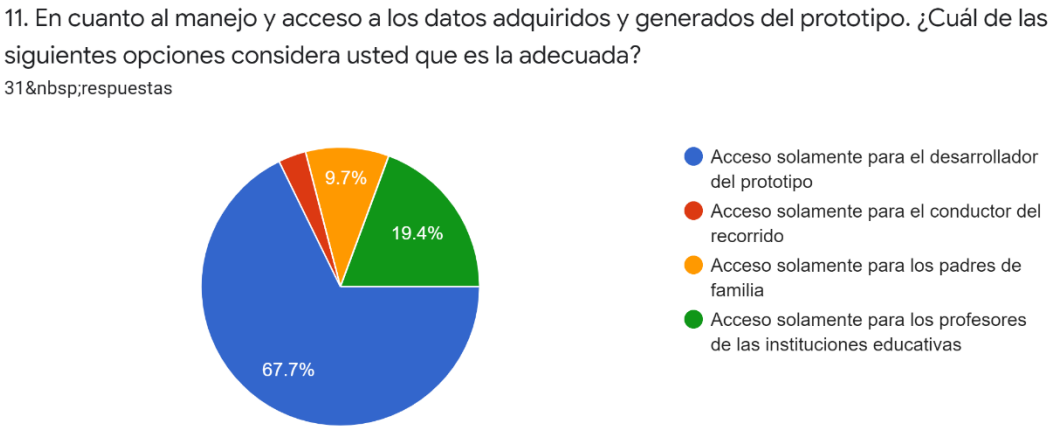


Figura. 11. Resultados de la pregunta #11 de la encuesta

Fuente: Autoría

Pregunta 12: Los resultados obtenidos de la décimo segunda pregunta muestran que, el punto más importante del sistema es la precisión del reconocimiento facial, seguido de la seguridad de los datos, a continuación, el tiempo de respuesta del sistema y por ultimo las notificaciones. La Figura muestra la tabulación de los resultados a la décimo segunda pregunta.

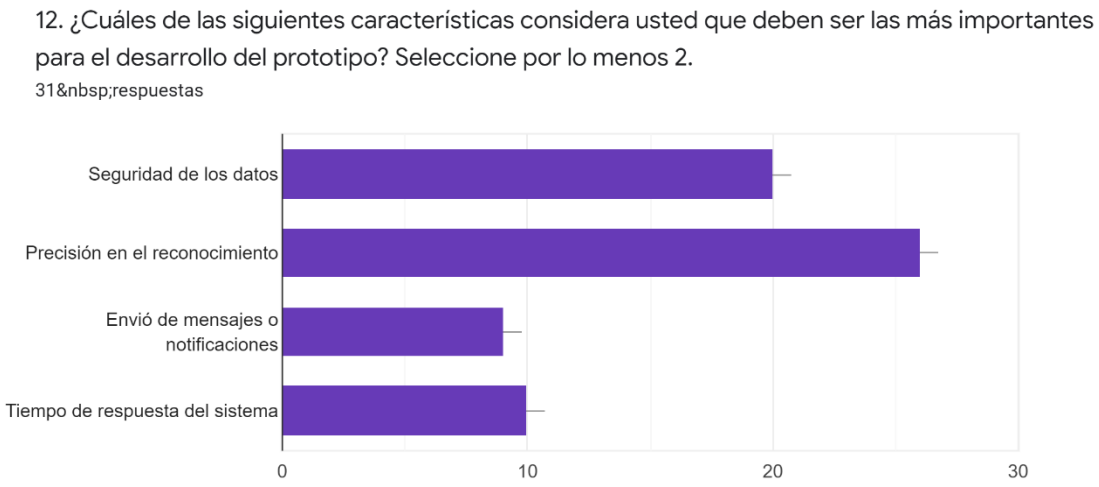


Figura. 12. Resultados de la pregunta #12 de la encuesta

Fuente: Autoría

Pregunta 13: Los resultados obtenidos de la décimo tercera pregunta muestran que, en cuanto a la generación de reportes periódicos, más del 50% de los encuestados considera que la generación de reportes periódicos no es importante para el proyecto. La Figura muestra la tabulación de los resultados a la décimo tercera pregunta.

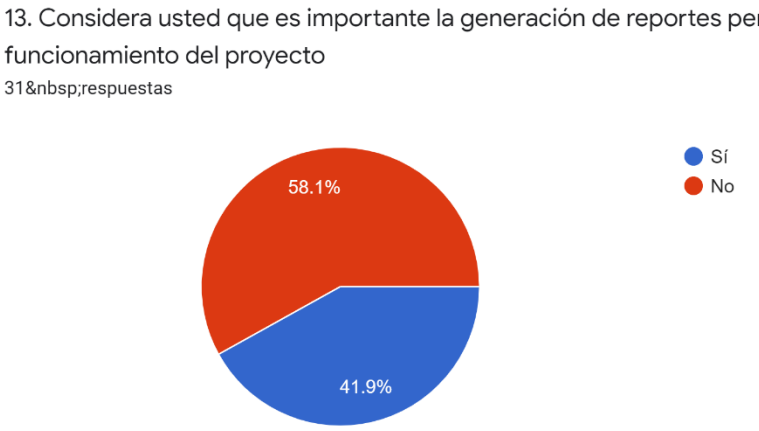


Figura. 13. Resultados de la pregunta #13 de la encuesta

Fuente: Autoría

ANEXO 3 (create_dataset.py**)**

```
# Se importa los paquetes necesarios
from imutils.video import VideoStream
from imutils import paths
import cv2
import os
import time
import imutils
import argparse
import numpy as np

# Se define los argumentos y luego se analiza
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

def start_capture(name):
        path = "./dataset/" + name
        total = 0  # inicializa el contador de Imagenes faciales almacenasdas
en el disco hasta el momento

        # Carga el modelo serializado desde el disco
        print("[INFO] cargando detector facial ...")
        protoPath = os.path.join("face_detection_model/deploy.prototxt")
        modelPath =
os.path.join("face_detection_model/res10_300x300_ssd_iter_140000.caffemodel")
        net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

        try:
            os.makedirs(path)# Crea un directorio con el nombre del usuario
        except:
            print('Directorio creado exitosamente')

        # Inicia la transmisión de video y espera que el sensor de la cámara
se caliente
        print("[INFO] iniciando transmisión de video ...")
        vs = VideoStream(src=0).start()  # Se inicializa la transmisión de
video
        time.sleep(2.0)  # permite que el sensor de la cámara se caliente
(2s)

        # Bucle sobre los fotogramas de video
        while True:
            frame = vs.read()
            orig = frame.copy()
            frame = imutils.resize(frame, width=400)

            # Toma las dimensiones del marco y luego las convierte en un BLOB
            (h, w) = frame.shape[:2]
            blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)), 1.0,
(300, 300), (104.0, 177.0, 123.0))

            # Se pasa el blob a través de la red y luego se obtiene las
detecciones y predicciones
```

```
            net.setInput(blob)
            detections = net.forward()

            # Bucle sobre las detecciones
            for i in range(0, detections.shape[2]):
                # Se extrae la confianza (es decir, la probabilidad) asociada
con la predicción
                confidence = detections[0, 0, i, 2]

                # Se filtra las detecciones débiles asegurándose de que la
"confianza" sea mayor que la confianza mínima
                if confidence < args["confidence"]:
                    continue

                # Se calcula las coordenadas (x, y) del cuadro delimitador
del objeto
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")

                # Dibuja un cuadro delimitador de la cara junto con la
probabilidad asociada
                text = "{:.2f}%".format(confidence * 100)
                y = startY - 10 if startY - 10 > 10 else startY + 10
                cv2.rectangle(frame, (startX, startY), (endX, endY), (0, 0,
255), 2)
                cv2.putText(frame, text, (startX, y),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
                cv2.putText(frame, str(str(total) + " imagenes capturadas"),
(startX, y + 200),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255))

                cv2.imshow("FaceDetection", frame)
                key = cv2.waitKey(1) & 0xFF

            try :
                cv2.imwrite(str(path+"/"+str(total)+name+".jpg"), orig)#
Guarda la imagen original en el directorio
                total += 1

            except :

                pass
            if key == ord("q") or key == 27 or total > 40:
                break
        cv2.destroyAllWindows()
        return total
```

ANEXO 4 (extract_embeddings.py)

```
# Se importa los paquetes necesarios
from imutils import paths
import numpy as np
import argparse
import imutils
import pickle
import cv2
import os

# Se construye el analizador de argumentos y analiza los argumentos
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="minimum probability to filter weak detections")
args = vars(ap.parse_args())

# Se carga el detector facial serializado desde el disco
print("[INFO] cargando detector facial ...")
protoPath = os.path.join("face_detection_model/deploy.prototxt")
modelPath =
os.path.join("face_detection_model/res10_300x300_ssd_iter_140000.caffemodel")
detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# Se carga el modelo de incrustación facial serializado desde el disco
print("[INFO] cargando modelo...")
embedder = cv2.dnn.readNetFromTorch("openface_nn4.small2.v1.t7")

print("[INFO] cuantificando caras ...")
# Ruta de acceso a cada imagen del conjunto de datos
imagePaths = list(paths.list_images("dataset"))

knownEmbeddings = []# Listas de incrustaciones faciales extraídas y
knownNames = []     # Listas de los nombres de personas correspondientes

total = 0 # Número total de caras procesadas

# Este bucle será responsable de extraer las incrustaciones de las caras que
se encuentran en cada imagen:
for (i, imagePath) in enumerate(imagePaths):
    # Se extrae el nombre de la persona de la ruta de la imagen
    print("[INFO] procesando imagen {}/{}".format(i + 1, len(imagePaths)))
    name = imagePath.split(os.path.sep)[-2]# Produce una lista de nombres de
carpeta/archivo (cadenas)

    image = cv2.imread(imagePath)# Se carga la imagen
    image = imutils.resize(image, width=600)# Se cambia el tamaño, ancho de
600 píxeles (mientras mantiene la relación de aspecto)
    (h, w) = image.shape[:2] # Toma las dimensiones de la Imagen

    # Se construye un BLOB a partir de la imagen
    imageBlob = cv2.dnn.blobFromImage(cv2.resize(image, (300, 300)), 1.0,
(300, 300), (104.0, 177.0, 123.0), swapRB=False, crop=False)
    # La función cv2.dnn.blobFromImage devuelve un Blob que es la imagen de
entrada después de la resta media,
    # la normalización y el intercambio de canales.
```

```
    detector.setInput(imageBlob)# Se aplica el detector de rostros basado en
aprendizaje profundo de OpenCV
    detections = detector.forward()#  para localizar rostros en la imagen de
entrada

    # Si se ha encontrado al menos una cara, se ejecuta la sentencia if.
    if len(detections) > 0:
        # Encuentra el cuadro delimitador con la mayor probabilidad
        i = np.argmax(detections[0, 0, :, 2])# Extraemos la detección con la
mayor confianza
        confidence = detections[0, 0, i, 2]

        # Verificamos para asegurarnos de que la confianza cumple con el
umbral de probabilidad mínimo
        # utilizado para filtrar las detecciones débiles
        if confidence > args["confidence"]:
            # Suponiendo que hemos alcanzado ese umbral, extraemos el ROI de
la cara y tomamos/verificamos las
            # dimensiones para asegurarnos de que el ROI de la cara sea
suficientemente grande (líneas 66-75).
            box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])# calcular
las coordenadas (x, y) del cuadro delimitador de la cara
            (startX, startY, endX, endY) = box.astype("int")

            face = image[startY:endY, startX:endX]# extraer el ROI de la cara
y
            (fH, fW) = face.shape[:2]# obtener las dimensiones del ROI
            # Comprueba el ancho y la altura de la cara sean lo
suficientemente grandes
            if fW < 20 or fH < 20:
                continue

            # Genera otro BLOB para el ROI de la cara, luego pasa el BLOB a
través del modelo de
            # incrustación facial para obtener la cuantificación 128-d de la
cara.
            faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255, (96, 96), (0,
0, 0), swapRB=True, crop=False)
            embedder.setInput(faceBlob)#Se procesa el faceBlob por el
embebedor CNN
            vec = embedder.forward()#Esto genera un vector 128-D (vec) que
describe la cara.
            # Estos datos sirven para reconocer caras nuevas a través del
aprendizaje automático.

            # Agrega el nombre de la persona + la cara correspondiente y el
vec incrustado a knownNames y knownEmbeddings, respectivamente
            knownNames.append(name)# agrega el nombre de la persona + rostro
correspondiente
            knownEmbeddings.append(vec.flatten())
            total += 1
# Este proceso de recorrer imágenes, detectar rostros y extraer
# incrustaciones de rostros continúa para todas y cada una de las imágenes
del conjunto de datos.

# Guardar las incrustaciones faciales + nombres en el disco
```

```
print("[INFO] serializando {} codificaciones ...".format(total))
data = {"embeddings": knownEmbeddings, "names": knownNames}
f = open("output\embeddings.pickle", "wb")
f.write(pickle.dumps(data))
f.close()
```

ANEXO 5 (train_model.py)

```
# Importamos los paquetes necesarios
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
import argparse
import pickle

# Se cargan  las incrustaciones faciales
print("[INFO] cargando inserciones de caras ...")
data = pickle.loads(open("output/embeddings.pickle", "rb").read())

# Se codifican las etiquetas
print("[INFO] etiquetas de codificación ...")
le = LabelEncoder()
labels = le.fit_transform(data["names"])

# Se entrena el modelo utilizado para aceptar las incrustaciones 128-d de la
cara
# para luego ejecutar el reconocimiento facial real
print("[INFO] modelo de entrenamiento ...")
recognizer = SVC(C=1.0, kernel="linear", probability=True)
recognizer.fit(data["embeddings"], labels)

# Se escribe el modelo de reconocimiento facial real en el disco
f = open("output/recognizer.pickle", "wb")
f.write(pickle.dumps(recognizer))
f.close()

# Se escribe el codificador de etiquetas en el disco
f = open("output/le.pickle", "wb")
f.write(pickle.dumps(le))
f.close()
```

ANEXO 6 (recognize_video.py)

```
# Se impoprta los paquetes necesarios
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import pickle
import time
import cv2
import os

# Se define los argumentos y luego se los analiza
ap = argparse.ArgumentParser()
ap.add_argument("-c", "--confidence", type=float, default=0.5,
    help="probabilidad mínima de filtrar detecciones débiles")
args = vars(ap.parse_args())

def reconize_face(name):
    # Carga los modelos serializados desde el disco
    print("[INFO] cargando modelos...")
    protoPath = os.path.join("face_detection_model/deploy.prototxt")
    modelPath =
os.path.join("face_detection_model/res10_300x300_ssd_iter_140000.caffemodel")
    detector = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

    # Se carga el modelo de incrustación facial serializado desde el disco
    print("[INFO] cargando reconocimiento facial ...")
    embedder =
cv2.dnn.readNetFromTorch("face_detection_model/openface_nn4.small2.v1.t7")

    # Se carga el modelo de reconocimiento facial real junto con el
codificador de etiquetas
    recognizer = pickle.loads(open("output/recognizer.pickle", "rb").read())
    le = pickle.loads(open("output/le.pickle", "rb").read())

    # Se inicializa la transmisión de video
    print("[INFO] iniciando transmisión de video ...")
    vs = VideoStream(src=0).start()
    time.sleep(2.0)

    # Inicia el estimador de rendimiento de FPS
    fps = FPS().start()

    # Sentencia que recorre los fotogramas de la secuencia del archivo de
video
    while True:
        # Toma el fotograma de la secuencia de video enhebrada
        frame = vs.read()

        # Cambia el tamaño del marco para que tenga un ancho de 600 píxeles
        # (manteniendo la relación de aspecto),
        # y luego toma las dimensiones de la imagen
        frame = imutils.resize(frame, width=600)
        (h, w) = frame.shape[:2]
```

```
        # Se construye un BLOB a partir de la imagen
        imageBlob = cv2.dnn.blobFromImage(
            cv2.resize(frame, (300, 300)), 1.0, (300, 300),
            (104.0, 177.0, 123.0), swapRB=False, crop=False)

        # Se aplica el detector de rostros basado en aprendizaje profundo de
OpenCV,
        # para localizar rostros en la imagen de entrada
        detector.setInput(imageBlob)
        detections = detector.forward()

        # Bucle sobre las detecciones
        for i in range(0, detections.shape[2]):
            # Extrae la confianza (probabilidad) asociada con la predicción
            confidence = detections[0, 0, i, 2]

            # Se filtra las detecciones débiles asegurándose de que la
"confianza"
            # sea mayor que la confianza mínima.
            if confidence > args["confidence"]:
                # Se calcula las coordenadas (x, y) del cuadro delimitador
del objeto
                box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
                (startX, startY, endX, endY) = box.astype("int")

                # Se extrae el ROI de la cara
                face = frame[startY:endY, startX:endX]
                (fH, fW) = face.shape[:2]

                # Comprueba el ancho y la altura de la cara sean
                # lo suficientemente grandes.
                if fW < 20 or fH < 20:
                    continue

                # Genera otro BLOB para el ROI de la cara,
                # luego pasa el BLOB a través del modelo de incrustación
facial
                # para obtener la cuantificación 128-d de la cara.
                faceBlob = cv2.dnn.blobFromImage(face, 1.0 / 255,
                                                 (96, 96), (0, 0, 0),
swapRB=True, crop=False)
                embedder.setInput(faceBlob)
                vec = embedder.forward()

                # Realiza clasificación para reconocer el rostro
                preds = recognizer.predict_proba(vec)[0]
                j = np.argmax(preds)
                proba = preds[j]
                name = le.classes_[j]

                # Dibuja un cuadro delimitador de la cara
                # junto con la probabilidad asociada.
                text = "{}: {:.2f}%".format(name, proba * 100)
                y = startY - 10 if startY - 10 > 10 else startY + 10
                cv2.rectangle(frame, (startX, startY), (endX, endY),
                              (0, 0, 255), 2)
```

```
            cv2.putText(frame, text, (startX, y),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)

    # Actualiza el contador de FPS
    fps.update()

    # Muestra el marco de salida
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # Para salir del bucle, presione q
    if key == ord("q"):
        break

# Para el temporizador y muestra la información de FPS
fps.stop()
print("[INFO] elasped time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

cv2.destroyAllWindows()
vs.stop()
```

ANEXO 7 (people_counter.py)

Archivo trackableobject.py

```
class TrackableObject:
    def __init__(self, objectID, centroid):
        # store the object ID, then initialize a list of centroids
        # using the current centroid
        self.objectID = objectID
        self.centroids = [centroid]

        # initialize a boolean used to indicate if the object has
        # already been counted or not
        self.counted = False
```

Archivo people_counter.py

```
# Se importa los paquetes necesarios
from pyimagesearch.centroidtracker import CentroidTracker
from pyimagesearch.trackableobject import TrackableObject
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2
import os

# Se construye los argumentos y luego se los analiza
ap = argparse.ArgumentParser()
ap.add_argument("-o", "--output", type=str,
    help="path to optional output video file")
ap.add_argument("-c", "--confidence", type=float, default=0.4,
    help="minimum probability to filter weak detections")
ap.add_argument("-s", "--skip-frames", type=int, default=30,
    help="# of skip frames between detections")
args = vars(ap.parse_args())

# Inicializa la lista de etiquetas de clase
# con las que MobileNet SSD fue entrenado para detectar
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"]

# Carga el modelo serializado desde el disco
print("[INFO] cargando modelo ...")
protoPath = os.path.join("mobilenet_ssd/MobileNetSSD_deploy.prototxt")
modelPath = os.path.join("mobilenet_ssd/MobileNetSSD_deploy.caffemodel")
net = cv2.dnn.readNetFromCaffe(protoPath, modelPath)

# Si no se proporciona una ruta de video, toma una referencia a la cámara web
```

```
print("[INFO] iniciando transmisión de video ...")
vs = VideoStream(src=0).start()
time.sleep(2.0)

# Se inicializa el escritor de video (se creará una instancia más adelante si
es necesario)
writer = None

# Inicializa las dimensiones del marco (se configuran
# tan pronto como se lea el primer marco del video)
W = None
H = None

#Crea una instancia del rastreador de centroide,
# luego iniciliza una lista para almacenar cada uno de los rastreadores de
correlación dlib,
# seguido de un diccionario para asignar cada ID de objeto único a un
TrackableObject
ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
trackers = []
trackableObjects = {}

# Inicializa el número total de fotogramas procesados hasta el momento,
# junto con el número total de objetos que se han movido hacia arriba o hacia
abajo.
totalFrames = 0
totalDown = 0
totalUp = 0

# Inicia el estimador de rendimiento de fotogramas por segundo
fps = FPS().start()

# Recorre los fotogramas de la secuencia de video
while True:
    # Toma el siguiente cuadro y lo maneja
    # si se esta leyendo de VideoCapture o VideoStream
    frame = vs.read()
    # Se cambia el tamaño del marco para que tenga un ancho máximo de 500
píxeles
    # (cuantos menos datos, más rápido el procesamiento),
    # luego convierte el marco de BGR a RGB para dlib
    frame = imutils.resize(frame, width=500)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # si las dimensiones del marco están vacías, se configura
    if W is None or H is None:
        (H, W) = frame.shape[:2]

    # Inicializa el escritor, para escribir el video en el disco
    if args["output"] is not None and writer is None:
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 30,
                                 (W, H), True)

    # Inicializa el estado actual junto con la lista de rectángulos
delimitadores devueltos
    # por (1) el detector de objetos o (2) los rastreadores de correlación
```

```
    status = "Esperando"
    rects = []

    # Verifica si se debe ejecutar un método de detección de objetos
    # más costoso desde el punto de vista informático para ayudar a al
rastreador
    if totalFrames % args["skip_frames"] == 0:
        # establece el estado e inicializa el nuevo conjunto de rastreadores
de objetoss
        status = "Detectando"
        trackers = []

        # Convierte el marco en un blob y pasa el blob
        # a través de la red para obtener las detecciones
        blob = cv2.dnn.blobFromImage(frame, 0.007843, (W, H), 127.5)
        net.setInput(blob)
        detections = net.forward()

        # Recorre las detecciones
        for i in np.arange(0, detections.shape[2]):
            # Extrae la confianza (es decir, la probabilidad) asociada con la
predicción
            confidence = detections[0, 0, i, 2]

            # Filtra las detecciones débiles requiriendo un mínimo de
confianza
            if confidence > args["confidence"]:
                # Extrae el índice de la etiqueta de clase de la lista de
detecciones
                idx = int(detections[0, 0, i, 1])

                # Si la etiqueta de la clase no es una persona, se ignora
                if CLASSES[idx] != "person":
                    continue

                # Calcula las coordenadas (x, y) del cuadro delimitador del
objeto
                box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
                (startX, startY, endX, endY) = box.astype("int")

                # Construye un objeto rectángulo dlib a partir de las
coordenadas
                # del cuadro delimitador y luego inicia el rastreador de
correlación dlib
                tracker = dlib.correlation_tracker()
                rect = dlib.rectangle(startX, startY, endX, endY)
                tracker.start_track(rgb, rect)

                # Agrega el rastreador a la lista de rastreadores
                # para que se pueda utilizar durante los marcos de salto
                trackers.append(tracker)

    # De lo contrario, se debe utilizar los * rastreadores * de objetos
    # en lugar de * detectores * de objetos *
    # para obtener un mayor rendimiento de procesamiento de cuadros
    else:
        # Bucle sobre los rastreadores
```

```
        for tracker in trackers:
            # Se establece el estado de nuestro sistema en 'seguimiento'
            # en lugar de 'esperando' o 'detectando'
            status = "Tracking"

            # Actualiza el rastreador y toma la posición actualizada
            tracker.update(rgb)
            pos = tracker.get_position()

            # Desempaqueta el objeto de posición
            startX = int(pos.left())
            startY = int(pos.top())
            endX = int(pos.right())
            endY = int(pos.bottom())

            # Agregua las coordenadas del cuadro delimitador a la lista de
rectángulos
            rects.append((startX, startY, endX, endY))

    # Dibuja una línea horizontal en el centro del marco;
    # una vez que un objeto cruce esta línea,
    # determina si se estaba moviendo 'hacia arriba' o 'hacia abajo'
    cv2.line(frame, (0, H // 2), (W, H // 2), (0, 255, 255), 2)

    # Utiliza el rastreador de centroides para asociar los
    # (1) centroides de objetos antiguos con
    # (2) los centroides de objetos recién calculados
    objects = ct.update(rects)

    # Recorre los objetos rastreados
    for (objectID, centroid) in objects.items():
        # Comprueba si existe un objeto rastreable para el ID de objeto
actual
        to = trackableObjects.get(objectID, None)

        # Si no hay un objeto rastreable existente, se crea uno
        if to is None:
            to = TrackableObject(objectID, centroid)

        # De lo contrario, hay un objeto rastreable,
        # por lo que se puede utilizarlo para determinar la dirección
        else:
            # La diferencia entre la coordenada y del centroide * actual *
            # y la media de los centroides * anteriores *
            # dirá en qué dirección se mueve el objeto (negativo para
'arriba' y positivo para 'abajo')
            y = [c[1] for c in to.centroids]
            direction = centroid[1] - np.mean(y)
            to.centroids.append(centroid)

            # Comprueba si el objeto ha sido contado o no
            if not to.counted:
                # Si la dirección es negativa (lo que indica que el objeto se
está moviendo hacia arriba)
                # Y el centroide está por encima de la línea central, cuenta
el objeto
                if direction < 0 and centroid[1] < H // 2:
```

```
                    totalUp += 1
                    to.counted = True

                # Si la dirección es positiva (indicando que el objeto se
mueve hacia abajo)
                # Y el centroide está debajo de la línea central, cuenta el
objeto
                elif direction > 0 and centroid[1] > H // 2:
                    totalDown += 1
                    to.counted = True

            # Almacena el objeto rastreable en nuestro diccionario
            trackableObjects[objectID] = to

            # Dibuja tanto la ID del objeto como el centroide del objeto en el
marco de salida
            text = "ID {}".format(objectID)
            cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
            cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)

        # Construye una tupla de información que se muestra en el marco
        info = [
            ("Arriba", totalUp),
            ("Abajo", totalDown),
            ("Estado", status),
        ]

        # Recorre las tuplas de información y las dibuja en nuestro marco
        for (i, (k, v)) in enumerate(info):
            text = "{}: {}".format(k, v)
            cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

        # Comprueba si se debe  escribir el marco en el disco
        if writer is not None:
            writer.write(frame)

        # Muestra el marco de salida
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

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

        # Incrementa el número total de fotogramas procesados
        # hasta el momento y luego actualiza el contador de FPS.
        totalFrames += 1
        fps.update()

    # Detiene el temporizador y muestra información de FPS
    fps.stop()
    print("[INFO] tiempo transcurrido: {:.2f}".format(fps.elapsed()))
    print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

    # Verifica si necesita soltar el puntero del escritor de video
```

```
if writer is not None:
    writer.release()

# Si no se usa un archivo de video, detenga la transmisión de video de la
cámara
if not args.get("input", False):
    vs.stop()

# de lo contrario, suelte el puntero del archivo de video
else:
    vs.release()

# close any open windows
cv2.destroyAllWindows()
```

ANEXO 8 (GUI.py)

```
import tkinter as tk
import webbrowser as web
from tkinter import font
from tkinter import messagebox

from monitor_camaras import monitor
from create_dataset import start_capture
from extract_embeddings import generate_embeddings
from train_model import train_classifier
from recognize_video import reconize_face


import PIL
import mysql.connector
import pymysql
from PIL import Image, ImageTk
from imutils.video import VideoStream
import cv2
import os
# ******************** EXTRAS *************
from extras.PreferencesDialog import *
from extras.KeyboardShortcuts import *
# from extras.BasicControls import *
# from extras.AboutDialog import *
# from extras.Tooltip import *
# from extras.AnnotationOverlay import *

# from  NotePage import *
# from  CameraUtils import *
# from extras.BasicControls import *


names = set()

class MainUI(tk.Tk):

    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)

        global names
        '''La función open () devuelve un objeto de archivo,
        que tiene un método read "r" () para leer el contenido del
archivo:'''
        with open("idlist.txt", "r") as f:
            x = f.read()
            z = x.rstrip().split(" ")
            '''El método rstrip() elimina cualquier carácter final
(caracteres al final de una cadena),
            el espacio es el carácter final predeterminado para eliminar.
            El método split () divide una cadena en una lista.'''
            for i in z:
                names.add(i)
```

```
        self.title_font = tk.font.Font(family='Helvetica', size=16,
weight="bold")
        # font = ("Helvetica Neue", 10, "bold")
        self.title("Virtual Assistant APP")
        self.resizable(False, False)
        self.geometry("905x510+25+25")
        self.protocol("WM_DELETE_WINDOW")
        self.active_name = None
        # ToolTip.LoadToolTips()

        # ----------- Íconos para el Menú y Botones ------------------------
        self.iconClose = ImageTk.PhotoImage(PIL.Image.open("Assets/window-
close.png"))
        self.iconPrefs =
ImageTk.PhotoImage(PIL.Image.open("Assets/prefs1_16x16.png"))
        self.iconWeb =
ImageTk.PhotoImage(PIL.Image.open("Assets/web_16x16.png"))
        image = PIL.Image.open('Assets/camera-icon.png')
        self.iconCameraBig = ImageTk.PhotoImage(image.resize((22, 22),
Image.ANTIALIAS))
        self.iconCamera = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('Assets/video-icon-b.png')
        self.iconVideoBig = ImageTk.PhotoImage(image.resize((22, 22),
Image.ANTIALIAS))
        self.iconVideo = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('Assets/icons8-manual-de-usuario-48.png')
        self.iconmanual = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('Assets/icons8-documentos-48.png')
        self.iconreportes = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('Assets/icons8-aceptar-la-base-de-datos-
64.png')
        self.icondb = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('Assets/icons8-documentos-de-producto-48.png')
        self.icondocs = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('Assets/icons8-teclado-48.png')
        self.iconteclado = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))

        image = PIL.Image.open('images/iconos/icons8-cámara-web-48.png')
        self.iconmonitor = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('images/iconos/icons8-añadir-grupo-de-
usuarios-hombre-hombre-96.png')
        self.iconusers = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('images/iconos/icons8-grupo-de-usuarios-
hombre-y-mujer-96.png')
        self.iconveri = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))
        image = PIL.Image.open('images/iconos/icons8-¿por-qué-somos-hombre-
96.png')
```

```
        self.iconrf = ImageTk.PhotoImage(image.resize((16, 16),
Image.ANTIALIAS))

        # ----------------------- Barra de Menús ----------------------------
--------------
        # Menu ARCHIVO
        menubar = tk.Menu(self, foreground='black', background='#F0F0F0',
activebackground='#86ABD9',
                          activeforeground='white')
        filemenu = tk.Menu(menubar, tearoff=0, foreground='black',
background='#F0F0F0', activebackground='#86ABD9',
                           activeforeground='white')
        filemenu.add_command(label="Preferencias...", underline=0,
image=self.iconPrefs, compound='left',
                             command=lambda e=None:
self.SystemPreferences(e))
        filemenu.add_separator()
        filemenu.add_command(label="Salir", underline=0,
image=self.iconClose, compound='left',
                             accelerator='Ctrl+Q', command=lambda e=None:
self.on_closing(e))
        self.DefineAccelerators('c', 'q', lambda e=None: self.on_closing(e))
        menubar.add("cascade", label='Archivo', underline=0, menu=filemenu)

        # Menu VENTANA
        toolsmenu = tk.Menu(menubar, tearoff=0, foreground='black',
background='#F0F0F0',
                            activebackground='#86ABD9',
activeforeground='white')
        toolsmenu.add_command(label="Monitor", underline=0,
accelerator='Ctrl+M',
                              command=lambda e=None: self.show_monitor(e),
image=self.iconmonitor, compound='left')
        self.DefineAccelerators('c', 'm', lambda e=None:
self.show_monitor(e))
        toolsmenu.add_command(label="Registro de Usuarios", underline=12,
                              command=lambda e=None:
self.show_frame("PageOne"), image=self.iconusers, compound='left')
        toolsmenu.add_command(label="Usuarios Registrados", underline=9,
                              command=lambda e=None:
self.show_frame("PageTwo"), image=self.iconveri, compound='left')
        toolsmenu.add_separator()
        toolsmenu.add_command(label="Reconocimiento Facial", underline=15,
command=lambda e=None: self.HelpAbout(e),
                              image=self.iconrf, compound='left')
        menubar.add("cascade", label='Opciones', underline=0, menu=toolsmenu)

        # Menu HERRAMIENTAS
        toolsmenu = tk.Menu(menubar, tearoff=0, foreground='black',
background='#F0F0F0',
                            activebackground='#86ABD9',
activeforeground='white')
        toolsmenu.add_command(label="Manual de Usuario", underline=0,
                              command=lambda e=None:
self.KeyboardShortcuts(e), image=self.iconmanual, compound='left')
        toolsmenu.add_command(label="Reportes", underline=0,
```

```
                                command=lambda e=None: self.JetsonNanoDocs(e),
image=self.iconreportes, compound='left')
        toolsmenu.add_command(label="Comandos Base de Datos", underline=9,
                                command=lambda e=None: self.docs(e),
image=self.icondb, compound='left')
        toolsmenu.add_separator()
        toolsmenu.add_command(label="Documentación", underline=0,
command=lambda e=None: self.quick_help(e),
                                image=self.icondocs, compound='left')
        menubar.add("cascade", label='Herramientas', underline=0,
menu=toolsmenu)

        # Menu AYUDA
        helpmenu = tk.Menu(menubar, tearoff=0, foreground='black',
background='#F0F0F0',
                           activebackground='#86ABD9',
activeforeground='white')
        helpmenu.add_command(label="Atajos de teclado...", underline=10,
                                command=lambda e=None:
self.KeyboardShortcuts(e), image=self.iconteclado, compound='left')
        helpmenu.add_command(label="Documentación de NVIDIA JETSON NANO...",
underline=17,
                                command=lambda e=None: self.JetsonNanoDocs(e),
image=self.iconWeb, compound='left')
        helpmenu.add_command(label="Documentación de OpenFace...",
underline=17,
                                command=lambda e=None: self.OpenFaceDocs(e),
image=self.iconWeb, compound='left')
        helpmenu.add_separator()
        helpmenu.add_command(label="Acerca de...", underline=1,
command=lambda e=None: self.HelpAbout(e))
        menubar.add("cascade", label='Ayuda', underline=1, menu=helpmenu)

        self.config(menu=menubar)

        #
**********************************************CONTENEDOR******************
**
        '''Creando un Contenedor: El contenedor es donde se apilan los marcos
uno encima del otro,
        luego el que queremos que sea visible se elevará por encima de los
demás'''
        container = tk.Frame(self)
        container.grid(sticky="nsew")
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        # Inicializando marcos a una matriz vacía
        self.frames = {}
        # Iterando a través de una tupla que consta de los diferentes diseños
de página
        for F in (StartPage, PageOne):  # Tupla con todas las páginas
posibles
            page_name = F.__name__
            frame = F(parent=container, controller=self)
            '''Inicializando el marco de ese objeto desde la página de
inicio, página1, página2
```

```
            respectivamente con for loop'''
            self.frames[page_name] = frame
            frame.grid(row=0, column=0, sticky="nsew")
        self.show_frame("StartPage")

    # Muestra el marco actual pasado como parámetro
    def show_frame(self, page_name):
        frame = self.frames[page_name]
        frame.tkraise()

    def on_closing(self, event):
        if messagebox.askokcancel("Salir ", " ¿Está seguro?"):
            self.destroy()

    def DefineAccelerators(self, keys, char, callFunc):
        commandDic = {'c': 'Control-', 'a': 'Alt-', 's': 'Shift-', 'f': ''}
        cmd = '<'
        for c in keys.lower():  # Se Necesita manejar teclas de función
también.
            cmd = cmd + commandDic[c]
        cmd1 = cmd
        cmd = cmd + char.upper() + ">"
        self.bind(cmd, callFunc)
        if len(char) == 1:
            cmd1 = cmd1 + char.lower() + ">"
            self.bind(cmd1, callFunc)

    def SystemPreferences(self, event):
        PreferencesDialog(self, title='Preferencias', minwidth=400,
minheight=350, okonly=False)
        # self.ControlMapping.SetControlMapping()

    def KeyboardShortcuts(self, event):
        KeyboardShortcutsDialog(self, title='Keyboard shortcuts',
                                resizable=True, minwidth=400, minheight=300)

# ********************************************** Página de inicio del primer
marco de ventana *********************
class StartPage(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

        font = ("Helvetica Neue", 10, "bold")
        colorfondo = "#ffffff"
        colobtnfont = "#5F9EA0"
        colorfuente = "#000000"
        colorfondo0 = "#5F9EA0"

        labelframe0 = tk.LabelFrame(self, bg=colorfondo0,
highlightbackground=colorfuente)
        labelframe0.place(x=0, y=0, width=905, height=50, )
        labelframe1 = tk.LabelFrame(self, bg=colorfondo,
highlightbackground=colorfuente)
        labelframe1.place(x=0, y=50, width=905, height=510)

        # Fondo de Pantala
```

```
        background_img = Image.open("images/fondos/x2.jpg")
        background_tkimg = ImageTk.PhotoImage(background_img)
        background_label = tk.Label(labelframe1, image=background_tkimg)
        background_label.image = background_tkimg
        background_label.place(x=0, y=0, relwidth=1, relheight=1)
        frame = tk.Frame(self)
        frame.place(anchor="c", relx=.5, rely=.5)

        # -----------------------------------------/////--------------------
----------------------------
        label = tk.Label(labelframe0, text="SISTEMA INTEGRADO PARA EL
PROCESAMIENTO DE DATOS",
                         font=self.controller.title_font,
                         bg=colorfondo0, fg=colorfuente).place(x=135, y=8)
        # -----------------------------------------/////--------------------
----------------------------
        # label0 = tk.Label(labelframe1, text="COMPROBAR CÁMARA", font=font,
fg=colorfuente, bg=colorfondo)
        # label0.grid(row=1, column=1, pady=10, padx=20, ipady=5, ipadx=10,
sticky="ew")
        render2 = PhotoImage(file='images/iconos/icons8-cámara-web-96.png')
        img2 = tk.Button(labelframe1, image=render2, bg=colobtnfont,
command=self.show_monitor)
        img2.image = render2
        img2.grid(row=0, column=0, pady=5, padx=75)
        ToolTip(img2, 1)
        # -----------------------------------------/////--------------------
----------------------------
        # label1 = tk.Label(labelframe1, text="AGREGAR USUARIO", font=font,
fg=colorfuente, bg=colorfondo)
        # label1.grid(row=2, column=1, pady=10, padx=20, ipady=5, ipadx=13,
sticky="ew")
        render3 = PhotoImage(file='images/iconos/icons8-añadir-grupo-de-
usuarios-hombre-hombre-96.png')
        img3 = tk.Button(labelframe1, image=render3, bg=colobtnfont,
                         command=lambda:
self.controller.show_frame("PageOne"))
        img3.image = render3
        img3.grid(row=1, column=0, pady=25, padx=25)
        ToolTip(img3, 2)
        # -----------------------------------------/////--------------------
----------------------------
        # label2 = tk.Label(labelframe1, text="COMPROBAR USUARIO", font=font,
fg=colorfuente, bg=colorfondo)
        # label2.grid(row=3, column=1, pady=10, padx=20, ipady=5, ipadx=4,
sticky="ew")
        render4 = PhotoImage(file='images/iconos/icons8-grupo-de-usuarios-
hombre-y-mujer-96.png')
        img4 = tk.Button(labelframe1, image=render4, bg=colobtnfont,
                         command=lambda:
self.controller.show_frame("PageTwo"))
        img4.image = render4
        img4.grid(row=1, column=1, pady=25, padx=25)
        ToolTip(img4, 3)
        # -----------------------------------------/////--------------------
----------------------------
```

```
        # label3 = tk.Label(labelframe1, text="RECONOCIMIENTO FACIAL",
font=font, fg=colorfuente, bg=colorfondo)
        # label3.grid(row=4, column=1, ipady=5, ipadx=5)
        render5 = PhotoImage(file='images/iconos/icons8-¿por-qué-somos-
hombre-96.png')
        img5 = tk.Button(labelframe1, image=render5, bg=colobtnfont,
state='disable',
                         command=lambda:
self.controller.show_frame("PageTwo"))
        img5.image = render5
        img5.grid(row=0, column=1, pady=25, padx=25)
        ToolTip(img5, 6)
        # ------------------------------------------/////-------------------
-----------------------------
        # label4 = tk.Label(labelframe1, text="ARCHIVOS", fg=colorfuente,
font=font, bg=colorfondo)
        # label4.grid(row=5, column=1, ipady=5, ipadx=5)
        render6 = PhotoImage(file='images/iconos/icons8-windows-explorer-
96.png')
        img6 = tk.Button(labelframe1, image=render6, bg=colobtnfont,
command=self.Proyect_Files)
        img6.image = render6
        img6.grid(row=3, column=0, pady=5, padx=75)
        ToolTip(img6, 4)
        # ------------------------------------------/////-------------------
-----------------------------
        # label5 = tk.Label(labelframe1, text="MANUAL", fg=colorfuente,
font=font, bg=colorfondo)
        # label5.grid(row=6, column=1, ipady=5, ipadx=5)
        # render7 = PhotoImage(file='images/iconos/icons8-manual-de-usuario-
96.png')
        # img7 = tk.Button(labelframe1, image=render7, bg=colobtnfont,
command=self.make_align)
        # img7.image = render7
        # img7.grid(row=3, column=1, pady=5, padx=5)
        # ToolTip(img7, 5)
        # ------------------------------------------/////-------------------
-----------------------------
        btn_salir = PhotoImage(file='images/iconos/icons8-cerrar-ventana-
96.png')
        button6 = tk.Button(labelframe1, image=btn_salir, bg="red",
command=self.on_closing)
        button6.image = btn_salir
        button6.grid(row=3, column=2, pady=25, padx=375)
        ToolTip(button6, 7)
        # ------------------------------------------/////-------------------
-----------------------------

    def show_monitor(self):
        monitor(self)

    def on_closing(self):
        self.controller.destroy()

    def Proyect_Files(self):
        open('dataset')
```

```
    # def make_align(self):
    #     start_align(self)


class PageOne(tk.Frame):
    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        self.controller = controller

        colorfg = "#000000"
        colorbg0 = "#dcdad5"
        colobtnfont = "#bdc2c6"
        colorfuente = "#000000"
        colorbg1 = "#ffffff"
        colorbotregis = "#bdc2c6"

    #     # ----------------------------------------//NOTEBOOK///----------
----------------------------------
    #     frame1 = ttk.Frame(self, padding=(5, 5, 5, 5))
    #     frame1.grid(row=0, column=0, sticky="NSEW")
    #     frame1.rowconfigure(2, weight=1)
    #     frame1.columnconfigure(0, weight=1)
    #
    #     self.AlwaysPreview = False
    #
    #     n = Notebook(frame1, padding=(5, 5, 5, 5))
    #     n.grid(row=1, column=0, rowspan=2, sticky=(N, E, W, S))
    #     n.columnconfigure(0, weight=1)
    #     n.enable_traversal()
    #
    #     # self.BasicControlsFrame = BasicControls(n)
    #
    #     self.UserRegistration = ttk.Frame(n)
    #     self.UserVerification = ttk.Frame(n)
    #     self.FinerControlFrame = ttk.Frame(n)
    #     # self.TimelapseFrame = Timelapse(n,camera)
    #
    #     # n.add(self.BasicControlsFrame, text='Basic', underline=0)
    #     self.user_ima = tk.PhotoImage(file="images/iconos/icons8-grupo-de-
usuarios-hombre-y-mujer-16.png")
    #     n.add(self.UserRegistration, text='Registro de Usuarios',
underline=0, image=self.user_ima, compound=tk.LEFT)
    #     n.add(self.UserVerification, text='Verificacion de Usuarios',
underline=0)
    #     n.add(self.FinerControlFrame, text='Advanced', underline=0)
    #     # n.add(self.TimelapseFrame,text='Time lapse',underline=0)
    #
    #     f1 = ttk.Frame(self.UserRegistration, pad=(10, 10))
    #     f1.grid(row=0, column=0, columnspan=2, padx=5, pady=5)
    #     # ************************************//INICIO
FORMULARIO//**************************************
    #     tk.Label(f1, text="CÉDULA:", font='Helvetica 9 bold').grid(row=1,
column=0, padx=3, pady=2, sticky='W')
    #     self.ci = tk.StringVar()
    #     self.ci = tk.Entry(f1, width=11, textvariable=self.ci,
font='Helvetica 10')
```

```
    #     self.ci.grid(row=1, column=1, pady=2, padx=3, ipadx=2, ipady=2,
sticky='W', columnspan=1)
    #     ToolTip(self.ci, 21)
    #     #
************************************///************************************
********************
    #     tk.Label(f1, text="N°:", font='Helvetica 9 bold').grid(row=1,
column=2, padx=3, pady=2, sticky='W')
    #     self.codigo = tk.StringVar()
    #     self.codigo = tk.Entry(f1, width=3, textvariable=self.codigo,
font='Helvetica 9', justify=tk.CENTER)
    #     self.codigo.grid(row=1, column=3, pady=2, padx=3, ipadx=2, ipady=2,
sticky='W', columnspan=1)
    #     ToolTip(self.codigo, 22)
    #     #
************************************///************************************
**********************
    #     tk.Label(f1, text="APELLIDOS:", font='Helvetica 9
bold').grid(row=2, column=0, padx=3, pady=2, sticky='W')
    #     self.apellido = tk.StringVar()
    #     self.apellido = tk.Entry(f1, width=20, textvariable=self.apellido,
font='Helvetica 10')
    #     self.apellido.grid(row=2, column=1, pady=2, padx=3, ipadx=2,
ipady=2, sticky='W', columnspan=3)
    #     ToolTip(self.apellido, 23)
    #     #
************************************///************************************
************************
    #     tk.Label(f1, text="NOMBRES:", font='Helvetica 9 bold').grid(row=3,
column=0, padx=3, pady=2, sticky='W')
    #     self.nombre = tk.StringVar()
    #     self.nombre = tk.Entry(f1, width=20, textvariable=self.nombre,
font='Helvetica 10')
    #     self.nombre.grid(row=3, column=1, pady=2, padx=3, ipadx=2, ipady=2,
sticky='W', columnspan=3)
    #     ToolTip(self.nombre, 24)
    #     #
************************************///************************************
*************************
    #     tk.Label(f1, text="EDAD:", font='Helvetica 9 bold').grid(row=4,
column=0, padx=3, pady=2, sticky='W')
    #     self.edad = tk.StringVar()
    #     self.edad = tk.Entry(f1, width=20, textvariable=self.edad,
font='Helvetica 10')
    #     self.edad.grid(row=4, column=1, pady=2, padx=3, ipadx=2, ipady=2,
sticky='W', columnspan=3)
    #     ToolTip(self.edad, 26)
    #     # #
************************************///************************************
**********************
    #     tk.Label(f1, text="GÉNERO:", font='Helvetica 9 bold').grid(row=5,
column=0, padx=3, pady=2, sticky='W')
    #     self.genero_var = tk.StringVar()
    #     combo_genero = ttk.Combobox(f1, width=17,
textvariable=self.genero_var, font='Helvetica 10', state="readonly")
    #     combo_genero['values'] = ("MASCULINO", "FEMENINO")
```

```
    #     combo_genero.grid(row=5, column=1, pady=2, padx=3, ipadx=2,
ipady=2, sticky='W', columnspan=3)
    #     ToolTip(combo_genero, 27)
    #     #
************************************///************************************
**************************
    #     tk.Label(f1, text="INSTITUCIÓN:", font='Helvetica 9
bold').grid(row=6, column=0, padx=3, pady=2, sticky='W')
    #     self.insti = tk.StringVar()
    #     self.insti = tk.Entry(f1, width=20, textvariable=self.insti,
font='Helvetica 10')
    #     self.insti.grid(row=6, column=1, pady=2, padx=3, ipadx=2, ipady=2,
sticky='W', columnspan=3)
    #     ToolTip(self.insti, 25)
    #     #
************************************///************************************
****************************
    #     tk.Label(f1, text="DOMICILIO:", font='Helvetica 9
bold').grid(row=7, column=0, padx=3, pady=2, sticky='W')
    #     # self.address_txt = tk.StringVar()
    #     self.address_txt = tk.Text(f1, width=20, height=3, font='Helvetica
10')
    #     self.address_txt.grid(row=7, column=1, pady=2, padx=3, ipadx=2,
ipady=2, sticky='W', columnspan=3)
    #     ToolTip(self.address_txt, 28)
    #     #
************************************///************************************
****************************
    #     self.tutor_name = tk.StringVar()
    #     tk.Label(f1, text="TUTOR:", font='Helvetica 9 bold').grid(row=8,
column=0, padx=3, pady=2, sticky='W')
    #     self.tutor_name = tk.StringVar()
    #     self.tutor_name = tk.Entry(f1, width=20,
textvariable=self.tutor_name, font='Helvetica 10')
    #     self.tutor_name.grid(row=8, column=1, pady=2, padx=3, ipadx=2,
ipady=2, sticky='W', columnspan=3)
    #     ToolTip(self.tutor_name, 29)
    #     # #
************************************///************************************
*************************
    #     self.phone = tk.StringVar()
    #     tk.Label(f1, text="N° CELULAR:", font='Helvetica 9
bold').grid(row=9, column=0, padx=3, pady=2, sticky='W')
    #     self.phone = tk.Entry(f1, width=20, textvariable=self.phone,
font='Helvetica 10')
    #     self.phone.grid(row=9, column=1, pady=2, padx=3, ipadx=2, ipady=2,
sticky='W', columnspan=3)
    #     ToolTip(self.phone, 30)
    #     #
************************************///************************************
**************************
    #     tk.Label(f1, text="ID USUARIO:", font='Helvetica 9
bold').grid(row=10, column=0, padx=3, pady=2, sticky='W')
    #     self.id = tk.StringVar()
    #     self.id = tk.Entry(f1, width=20, textvariable=self.id,
font='Helvetica 10')
```

```
    #     self.id.grid(row=10, column=1, pady=2, padx=3, ipadx=2, ipady=2,
sticky='W', columnspan=3)
    #     ToolTip(self.id, 31)
    #     # ************************************//BOTON DE
REGISTRO//*********************************************
    #     self.regis_button = tk.Button(f1, text="REGISTRAR", font='Helvetica
9 bold', command=self.start_regis)
    #     self.regis_button.grid(row=11, column=0, pady=5, padx=3, ipadx=2,
ipady=2, columnspan=4)
    #     ToolTip(self.regis_button, 32)
    #     self.numimglabel = tk.Label(f1, text="# = 0")
    #     self.numimglabel.grid(row=11, column=3, padx=3, pady=5, sticky="E")
    #     ToolTip(self.numimglabel, 35)
    #     # ------------------------------------------/////------------------
------------------------------
    #     # ------------------------------------------/////------------------
------------------------------
    #     self.regis_button = tk.Button(f1, text="EXTRAER INCRUSTACIONES",
font='Helvetica 9 bold',
    #                                   command=self.start_regis)
    #     self.regis_button.grid(row=12, column=0, pady=5, padx=3, ipadx=2,
ipady=2, columnspan=4)
    #     ToolTip(self.regis_button, 32)
    #     self.numimglabel = tk.Label(f1, text="# = 0")
    #     self.numimglabel.grid(row=12, column=3, padx=3, pady=5, sticky="E")
    #     ToolTip(self.numimglabel, 36)
    #     # ------------------------------------------/////------------------
------------------------------
    #     self.regis_button = tk.Button(f1, text="ENTRENAR MODELO",
font='Helvetica 9 bold',
    #                                   command=self.start_regis)
    #     self.regis_button.grid(row=13, column=0, pady=5, padx=3, ipadx=2,
ipady=2, columnspan=4)
    #     ToolTip(self.regis_button, 32)
    #     self.numimglabel = tk.Label(f1, text="# = 0")
    #     self.numimglabel.grid(row=13, column=3, padx=3, pady=5, sticky="E")
    #     ToolTip(self.numimglabel, 37)
    #
    #     # labelframe5 = tk.LabelFrame(f1, text="ENTRENAMIENTO", fg="blue")
    #     # labelframe5.grid(row=12, column=1, pady=5, padx=5, ipadx=5)
    #     # self.embeddingsbutton = tk.Button(labelframe5,
text="INCRUSTACIONES",
    #     #                                   fg="Black", bg="Gainsboro",
state='disable', command=self.extractembeddings)
    #     # self.embeddingsbutton.grid(row=0, column=0, ipadx=5, ipady=5,
padx=2, pady=5, sticky="w")
    #     # self.trainbutton = tk.Button(labelframe5, text="ENTRENAR",
    #     #                              fg="Black", bg="Gainsboro",
state='disable', command=self.trainmodel)
    #     # self.trainbutton.grid(row=1, column=0, ipadx=5, ipady=5, padx=2,
pady=5, sticky="w")
    #     # ------------------------------------------/////------------------
------------------------------
    #     btn_salir = PhotoImage(file='images/iconos/icons8-izquierda-en-
cuadrado-48.png')
    #     button6 = tk.Button(f1, image=btn_salir, bg=colobtnfont,
command=lambda: self.controller.show_frame("StartPage"))
```

```
    #     button6.image = btn_salir
    #     # button6.grid(row=12, column=2, pady=25, padx=375)
    #     button6.place(x=815, y=395)
    #     ToolTip(self.regis_button, 38)
    #     # ////////////////////////////////////////////****BUSCAR
USUARIOS****/////////////////////////////////
    #     labelframe5 = tk.LabelFrame(f1, text="BUSCAR USUARIO", fg="blue")
    #     labelframe5.grid(row=8, column=4, rowspan=3, pady=2, padx=2,
ipadx=5, sticky='W')
    #     # ************************************////**************
    #     tk.Label(labelframe5, text="BUSCAR POR:",
    #              fg="#263942", font='Helvetica 9 bold').grid(row=0,
column=0, padx=3, pady=2, sticky='W')
    #     self.buscar_por = tk.StringVar()
    #     combo_buscar = ttk.Combobox(labelframe5,
textvariable=self.buscar_por,
    #                                 width=10, font=("Helvetica 9"),
state='readonly')
    #     combo_buscar['values'] = ("Código", "Cédula", "Apellidos")
    #     combo_buscar.grid(row=0, column=1, padx=1, pady=1)
    #     # ----------------------------------------/////-----------------
-------------------------------
    #     self.buscar_txt = tk.StringVar()
    #     self.buscar_txt = tk.Entry(labelframe5,
textvariable=self.buscar_txt,
    #                                width=15, borderwidth=1, bg="White",
font='Helvetica 9')
    #     self.buscar_txt.grid(row=0, column=2, padx=3, pady=2, sticky="w")
    #     # # ----------------------------------------/////----------------
---------------------------------
    #     self.search_button = tk.Button(labelframe5, text="BUSCAR",
fg="Black", bg="Gainsboro",
    #
command=self.buscar_data).grid(row=0, column=3, padx=3, pady=2, sticky='W')
    #     # # ----------------------------------------/////----------------
---------------------------------
    #     self.show_all = tk.Button(labelframe5, text="MOSTRAR TODO",
fg="Black", bg="Gainsboro",
    #                               command=self.obtener_datos).grid(row=0,
column=4, padx=3, pady=2, sticky='W')
    #     # # ----------------------------------------/////----------------
---------------------------------
    #     self.show_all = tk.Button(labelframe5, text="ACTUALIZAR",
fg="Black", bg="Gainsboro",
    #                               command=self.update_data).grid(row=1,
column=4, padx=3, pady=2, sticky='W')
    #     # # ----------------------------------------/////----------------
---------------------------------
    #     self.show_all = tk.Button(labelframe5, text="ELIMINAR", fg="Black",
bg="Gainsboro",
    #                               command=self.delete_data).grid(row=1,
column=2, padx=3, pady=2, sticky='W')
    #     # # ----------------------------------------/////----------------
---------------------------------
    #     self.show_all = tk.Button(labelframe5, text="LIMPIAR", fg="Black",
bg="Gainsboro",
```

```
    #                                 command=self.clear).grid(row=1, column=3,
padx=3, pady=2, sticky='W')
    #     # # ----------------------------------------/////----------------
--------------------------------
    #
    #     # ----------------------------------------/////------------------
------------------------------
    #     Table_Frame = tk.Frame(f1)
    #     Table_Frame.grid(row=1, column=4, padx=3, pady=2, sticky='W',
rowspan=7)
    #     # Table_Frame.place(x=250, y=50, width=560, height=200)
    #
    #     scroll_x = tk.Scrollbar(Table_Frame, orient='horizontal')
    #     scroll_y = tk.Scrollbar(Table_Frame, orient='vertical')
    #     self.Users_table = ttk.Treeview(Table_Frame,
    #                                     columns=("Código", "CI",
"Apellidos", "Nombres", "Institución", "Edad",
    #                                              "Género", "Domicilio",
"NombreT", "Celular", "ID"),
    #                                     xscrollcommand=scroll_x.set,
yscrollcommand=scroll_y.set)
    #     ToolTip(self.Users_table, 34)
    #     scroll_x.pack(side=tk.BOTTOM, fill=tk.X)
    #     scroll_y.pack(side=tk.RIGHT, fill=tk.Y)
    #
    #     scroll_x.config(command=self.Users_table.xview)
    #     scroll_y.config(command=self.Users_table.yview)
    #     self.Users_table.heading("Código", text="#")
    #     self.Users_table.heading("CI", text="CI")
    #     self.Users_table.heading("Apellidos", text="Apellidos")
    #     self.Users_table.heading("Nombres", text="Nombres")
    #     self.Users_table.heading("Institución", text="Institución")
    #     self.Users_table.heading("Edad", text="Edad")
    #     self.Users_table.heading("Género", text="Género")
    #     self.Users_table.heading("Domicilio", text="Domicilio")
    #     self.Users_table.heading("NombreT", text="NombreT")
    #     self.Users_table.heading("Celular", text="Celular")
    #     self.Users_table.heading("ID", text="ID")
    #     self.Users_table['show'] = 'headings'
    #     self.Users_table.column("Código", width=25)
    #     self.Users_table.column("CI", width=90)
    #     self.Users_table.column("Apellidos", width=120)
    #     self.Users_table.column("Nombres", width=120)
    #     self.Users_table.column("Institución", width=125)
    #     self.Users_table.column("Edad", width=40)
    #     self.Users_table.column("Género", width=80)
    #     self.Users_table.column("Domicilio", width=125)
    #     self.Users_table.column("NombreT", width=120)
    #     self.Users_table.column("Celular", width=90)
    #     self.Users_table.column("ID", width=110)
    #
    #     self.Users_table.pack(fill=tk.BOTH, expand=1)
    #     self.Users_table.bind("<ButtonRelease-1>", self.get_cursor)
    #     self.obtener_datos()
    #
    # def obtener_datos(self):
```

```
    #     con = pymysql.connect(host="localhost", user="root", password="",
database="reg")
    #     cur = con.cursor()
    #     cur.execute("select * from formulario")
    #     rows = cur.fetchall()
    #     if len(rows) != 0:
    #         self.Users_table.delete(*self.Users_table.get_children())
    #         for row in rows:
    #             self.Users_table.insert('', tk.END, values=row)
    #         con.commit()
    #     con.close()
    #
    # def clear(self):
    #     self.codigo.delete("0", tk.END)
    #     self.ci.delete("0", tk.END)
    #     self.nombre.delete("0", tk.END)
    #     self.apellido.delete("0", tk.END)
    #     self.insti.delete("0", tk.END)
    #     self.edad.delete("0", tk.END)
    #     self.genero_var.set("")
    #     self.address_txt.delete("1.0", tk.END)
    #     self.tutor_name.delete("0", tk.END)
    #     self.phone.delete("0", tk.END)
    #     self.id.delete("0", tk.END)
    #
    # def get_cursor(self, ev):
    #     cursor_row = self.Users_table.focus()
    #     contents = self.Users_table.item(cursor_row)
    #     row = contents['values']
    #     self.codigo.delete("0", tk.END)
    #     self.codigo.insert(tk.END, row[0])
    #     self.ci.delete("0", tk.END)
    #     self.ci.insert(tk.END, row[1])
    #     self.apellido.delete("0", tk.END)
    #     self.apellido.insert(tk.END, row[2])
    #     self.nombre.delete("0", tk.END)
    #     self.nombre.insert(tk.END, row[3])
    #     self.insti.delete("0", tk.END)
    #     self.insti.insert(tk.END, row[4])
    #     self.edad.delete("0", tk.END)
    #     self.edad.insert(tk.END, row[5])
    #     self.genero_var.set(row[6])
    #     self.address_txt.delete("1.0", tk.END)
    #     self.address_txt.insert(tk.END, row[7])
    #     self.tutor_name.delete("0", tk.END)
    #     self.tutor_name.insert(tk.END, row[8])
    #     self.phone.delete("0", tk.END)
    #     self.phone.insert(tk.END, row[9])
    #     self.id.delete("0", tk.END)
    #     self.id.insert(tk.END, row[10])
    #
    # def update_data(self):
    #     if (self.codigo.get() == "" or self.ci.get() == "" or
self.apellido.get() == "" or self.nombre.get() == ""
    #             or self.insti.get() == "" or self.edad.get() == "" or
self.genero_var.get() == ""
```

```
    #             or self.tutor_name.get() == "" or self.phone.get() == "" or
self.id.get() == ""):
    #         messagebox.showerror("Error", "Seleccione el registro a
actualizar")
    #     else:
    #         con = pymysql.connect(host="localhost", user="root",
password="", database="reg")
    #         cur = con.cursor()
    #         cur.execute("update formulario set
Cédula=%s,Apellidos=%s,Nombres=%s,Institución=%s,"
    #
"Edad=%s,Género=%s,Domicilio=%s,Nombre_Tutor=%s,Celular=%s,ID=%s where
Código=%s", (
    #                           self.ci.get(),
    #                           self.apellido.get(),
    #                           self.nombre.get(),
    #                           self.insti.get(),
    #                           self.edad.get(),
    #                           self.genero_var.get(),
    #                           self.address_txt.get("1.0", tk.END),
    #                           self.tutor_name.get(),
    #                           self.phone.get(),
    #                           self.id.get(),
    #                           self.codigo.get()))
    #         con.commit()
    #         self.obtener_datos()
    #         self.clear()
    #         messagebox.showinfo("Actualizando", "Se actualizó correctamente
el registro")
    #         con.close()
    #
    # def delete_data(self):
    #     if (self.codigo.get() == "" or self.ci.get() == "" or
self.apellido.get() == "" or self.nombre.get() == ""
    #             or self.insti.get() == "" or self.edad.get() == "" or
self.genero_var.get() == ""
    #             or self.tutor_name.get() == "" or self.phone.get() == ""):
    #         messagebox.showerror("Error", "Seleccione el registro a
eliminar")
    #     else:
    #         con = pymysql.connect(host="localhost", user="root",
password="", database="reg")
    #         cur = con.cursor()
    #         cur.execute("delete from formulario where Código=%s",
self.codigo.get())
    #         con.commit()
    #         self.obtener_datos()
    #         self.clear()
    #         messagebox.showinfo("Eliminar", "Se eliminó correctamente el
registro")
    #         con.close()
    #
    # def buscar_data(self):
    #     con = mysql.connector.connect(host="localhost", user="root",
passwd="", database="reg")
    #     cur = con.cursor()
```

```
    #     cur.execute("select * from formulario where " +
str(self.buscar_por.get()) + " LIKE '%" + str(
    #         self.buscar_txt.get()) + "%'")
    #     rows = cur.fetchall()
    #     if len(rows) != 0:
    #         self.Users_table.delete(*self.Users_table.get_children())
    #         for row in rows:
    #             self.Users_table.insert('', tk.END, values=row)
    #         con.commit()
    #     con.close()
    #
    # def capimg(self):
    #     self.numimglabel.config(text=str("# IMÁGENES = 0 "))
    #     messagebox.showinfo("INSTRUCCIONES ", " Se capturará 300 fotos de
su rostro.")
    #     x = start_capture(self.controller.active_name)
    #     # x = start_capture(self.id.insert("0", tk.END))
    #     self.controller.num_of_images = x
    #     self.numimglabel.config(text=str("# IMÁGENES = " + str(x)))
    #     # messagebox.showinfo("Imágenes Capturadas")
    #
    # def extractembeddings(self):
    #     if self.controller.num_of_images < 40:
    #         messagebox.showerror("ERROR ", " No hay suficientes datos,
¡captura al menos 300 imágenes!")
    #         return
    #     generate_embeddings(self.controller.active_name)
    #     messagebox.showinfo("ÉXITO ", " ¡Codificaciones generadas
correctamente!")
    #
    # def trainmodel(self):
    #     if self.controller.num_of_images < 40:
    #         messagebox.showerror("ERROR ", " No hay suficientes datos,
¡captura al menos 300 imágenes!")
    #         return
    #     train_classifier(self.controller.active_name)
    #     messagebox.showinfo("ÉXITO ", " ¡El modelo ha sido entrenado con
éxito!")
    #     self.controller.show_frame("PageThree")
    #
    # def start_regis(self):
    #     con = pymysql.connect(host="localhost", user="root", password="",
database="reg")
    #     cur = con.cursor()
    #     cur.execute("insert into formulario
values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)", (
    #         self.codigo.get(),
    #         self.ci.get(),
    #         self.apellido.get(),
    #         self.nombre.get(),
    #         self.insti.get(),
    #         self.edad.get(),
    #         self.genero_var.get(),
    #         self.address_txt.get("1.0", tk.END),
    #         self.tutor_name.get(),
    #         self.phone.get(),
    #         self.id.get()))
```

```
    #
    #     con.commit()
    #     # self.obtener_id()
    #     self.obtener_datos()
    #     self.clear()
    #     messagebox.showinfo("Adelante...", "Se agregó correctamente el
registro")
    #     self.capimg()
    #     con.close()
    #     # if self.capturebutton['state'] == tk.DISABLED:
    #     #     self.capturebutton['state'] = tk.NORMAL
    #
    # def ImageDenoiseChecked(self):
    #     self.camera.image_denoise = self.ImageDenoise.get()
    #
    # def VideoDenoiseChecked(self):
    #     self.camera.video_denoise = self.VideoDenoise.get()
    #
    # def VideoStabChecked(self):
    #     self.camera.video_stabilization = self.VideoStab.get()
    #
    # def UseVideoPort(self, val):
    #     pass  # self.camera.use_video_port = val
    #
    # def MyLabelFrame(f, txt, row, col, stick='NEWS', py=5, span=1, pad=(5,
5, 5, 5)):
    #     l = ttk.LabelFrame(f, text=txt, padding=pad)
    #     l.grid(row=row, column=col, sticky=stick, columnspan=span, pady=py)
    #     return l
    #
    # def MyRadio(f, txt, varValue, varName, cmd=None, row=0, col=0,
stick='W',
    #             span=1, pad=(5, 5, 5, 5), tip='No Tip number provided'):
    #     # def MyRadio ( f, txt, varValue, varName = None, cmd=None, row=0,
col=0, stick='W',
    #     #               span=1, pad=(5,5,5,5), tip='No Tip number
provided'):
    #     # if varName is None:
    #     #     # Determine type of var from varValue and create one
    #     #     if type(varValue) is int:
    #     #             varName = MyIntVar(varValue)
    #     #     elif type(varValue) is boolean:
    #     #             varName = MyBooleanVar(varValue)
    #     #     elif type(varValue) is str:
    #     #             varName = MyStringVar(varValue)
    #     if cmd is None:
    #         r = ttk.Radiobutton(f, text=txt, value=varValue,
variable=varName,
    #                             padding=pad)
    #     else:
    #         r = ttk.Radiobutton(f, text=txt, value=varValue,
variable=varName,
    #                             command=lambda: cmd(varValue), padding=pad)
    #     r.grid(row=row, column=col, sticky=stick, columnspan=span)
    #     ToolTip(r, msg=tip)
    #     return r  # , varName     # return RadioButton and BooleanVar
    #
```

```
    # '''
    #   params = [ ['text', True or False, row, col, sticky, rowspan, tipNum]
    #                ['text', True or False, row, col, sticky, rowspan,
tipNum] ]
    #   Command = the function to call if pressed, pass True or False
    #   Create two radio buttons, return the first one, and the BooleanVar
    #   associated with the two. If command is not None, then call the
command
    #   with the same value passed in the list
    #
    #   Return, first radio created, BooleanVar created
    # '''




#
# class PageTwo(tk.Frame):
#
#     def __init__(self, parent, controller):
#         tk.Frame.__init__(self, parent)
#         self.controller = controller
#
# class PageThree(tk.Frame):
#
#     def __init__(self, parent, controller):
#         tk.Frame.__init__(self, parent)
#         self.controller = controller

app = MainUI()
app.iconphoto(False, tk.PhotoImage(file='images/iconos/icon.ico'))
app.mainloop()
```

ANEXO 9 (Manual de Usuario)

https://docs.google.com/document/d/1d6n6aJftRkZV2WgQIDleOgmNGj_ACGV9/edit?usp=sharing&ouid=103941741641349986790&rtpof=true&sd=true