Detección automática, clasificación y reconocimiento de escorpiones mediante técnicas de Aprendizaje Profundo

Tesis de doctorado

Francisco Luis Giambelluca

Presentada ante la Facultad de Ingeniería de la

Universidad Nacional de La Plata

como requisito para la obtención del grado académico de

DOCTOR EN INGENIERÍA

Dirección de Tesis: *Dr. Ing. Marcelo A. Cappelletti*

La Plata, 15 de julio de 2022

Agradecimiento

A toda mi familia y amigos quienes me han brindado afecto y apoyo incondicional y hecho la persona que soy hoy.

A la Universidad Nacional de La Plata, que me proporcionó una beca doctoral, la cual me permitió desarrollar esta tesis con mayor dedicación y mejor equipo.

A los docentes de postgrado y de grado, quienes me proporcionaron los conocimientos necesarios para desarrollar esta tesis.

A los integrantes docentes y no docentes del área CeTAD, Grupo de Control Aplicado (GCA), LEICI–CONICET–UNLP por su hospitalidad y apoyo para la realización de este trabajo.

A los integrantes del laboratorio de aracnología del CEPAVE por su apoyo en la obtención de imágenes utilizadas como base de datos para entrenamiento.

Dedicatoria

A mis padres y hermana, quienes siempre me han apoyado y acompañado durante esta y todas las etapas de mi vida.

A mis amigos incondicionales, aquellos que siempre han estado y que siempre estarán.

Resumen

La detección e identificación temprana de los escorpiones es esencial debido a la peligrosidad de estos arácnidos que ponen en riesgo la salud de la población, en particular, de los sectores más vulnerables al veneno de un escorpión, como son las personas hipertensas, cardíacas o diabéticas, pero también los niños y los ancianos. A su vez, la detección y clasificación de escorpiones puede ser útil con fines de investigación biológica para estudiar las diferentes variedades de géneros y especies. En este trabajo, con el propósito de brindar herramientas de prevención alternativas, se desarrollaron novedosos sistemas automáticos y en tiempo real para detectar y clasificar escorpiones, utilizando heurísticas de visión artificial y Aprendizaje Profundo, basados en las características de la forma y la propiedad de fluorescencia de los escorpiones cuando son expuesto a luz ultravioleta. En particular, se han investigado las tres especies de escorpiones que se encuentran en la ciudad de La Plata: *Bothriurus bonariensis* (sin importancia sanitaria), *Tityus carrilloi* y *Tityus confluens* (ambas de importancia sanitaria). Durante este trabajo se llevaron a cabo comparaciones entre diferentes modelos basados en Aprendizaje Profundo utilizados para detectar e identificar escorpiones, ya sea por género peligroso o no peligroso, como para determinar su especie dentro de un mismo género. Los resultados satisfactorios obtenidos indican que los sistemas desarrollados pueden, de forma temprana, precisa, no invasiva y segura, detectar y clasificar escorpiones, incluso dentro de un ambiente no controlado, es decir, cuando el escorpión se encuentra cerca de otros objetos que podrían dificultar su detección. Los sistemas de detección y clasificación desarrollados en este trabajo se implementaron como una aplicación móvil, con la ventaja de la portabilidad y la facilidad de acceso a la población, que puede ser utilizada como una herramienta de prevención eficaz para minimizar las picaduras de escorpiones y ayudar a reducir el daño que pueden ocasionar a las poblaciones expuestas a estos arácnidos. Además, estos sistemas son fácilmente escalables a otros géneros y especies de escorpiones para ampliar la región donde se puedan utilizar estas aplicaciones.

Abstract

The early detection and identification of scorpions is essential due to the high hazard of these arachnids that put the health of the population at risk, in particular, of the most vulnerable sectors to the venom of a scorpion, such as hypertensive, cardiac, or diabetic people, but also children and the elderly. Additionally, scorpion detection and classification can be helpful for biological research purposes to study the different varieties of genera and species. In this work, with the purpose of providing alternative prevention tools, novel automatic and real-time systems were developed to detect and classify scorpions, using computer vision and Deep Learning techniques, based on the shape features and the fluorescent property of the scorpions when are exposed to ultraviolet light. In particular, the three species of scorpions found in La Plata city: *Bothriurus bonariensis* (of no sanitary importance), *Tityus carrilloi*, and *Tityus confluens* (both of sanitary importance) have been researched. During this work, comparisons were carried out between different Deep Learning-based models used to detect and identify scorpions, either by dangerous or non-dangerous genus, as well as to determine the species within the same genus. The satisfactory results obtained indicate that the systems developed can, in an early, accurate, non-invasive, and safe way, detect and classify scorpions, even within an uncontrolled environment, that is, when the scorpion is close to other objects that could make detection difficult. The detection and classification systems developed in this work were implemented as a mobile application, with the advantage of the portability and readily available to the population, which can be used as a effective prevention tool to minimize scorpion stings and to help reduce the harm they can cause to populations exposed to these arachnids. Also, these systems are easily scalable to other genera and species of scorpions to extend the region where these applications can be used.

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1: Introducción

1.1. Aspectos generales y aportes de la Tesis

Los escorpiones son invertebrados artrópodos, o sea tienen sus patas articuladas, y dentro de este grupo pertenecen a los arácnidos, juntamente con arañas, pseudoescorpiones, ácaros, solífugos, opiliones y amblipigidos. Son principalmente animales nocturnos que suelen evitar la luz solar. Durante el día se pueden encontrar resguardados en ambientes naturales, bajo rocas o en el interior de agujeros, aunque algunas especies también pueden ser domiciliarias, las cuales permanecen escondidas durante el día y deambulan por la noche.

Todas las especies de escorpiones tienen la capacidad de inocular veneno, algunos de ellos incluso con la posibilidad de matar a un humano. En Argentina se registran entre 7000 y 8000 picaduras de escorpiones al año, según datos del Ministerio de Salud de la Nación. Asimismo, se estima que el 10% de los casos han requerido tratamiento y la mortalidad se sitúa entre dos y ocho personas al año. El aumento de las picaduras suele producirse en épocas cálidas del año debido a la mayor actividad de los escorpiones. Los sectores más vulnerables de la población al veneno de un escorpión son las personas hipertensas, cardíacas o diabéticas, pero también los niños y los ancianos, aunque no padezcan enfermedades.

Dentro de los diferentes géneros de escorpiones que se encuentran en Argentina, solo un género (*Tityus*) es considerado de importancia sanitaria, el cual tiene varias especies muy peligrosas, como *Tityus confluens, Tityus carrilloi¸ Tityus bahiensis* y *Tityus serrulatus*. Otras especies de este género pueden causar un gran dolor con sus picaduras, aunque no registran incidentes mortales [1].

Las especies *Tityus carrilloi* y *Tityus confluens* suelen ser domiciliarias, lo cual hace que el contacto con las personas aumente en forma notable. Las grandes ciudades, como Buenos Aires, Rosario, Córdoba, entre otras, sufren a diario este problema, incluyendo al Gran Buenos Aires. En el caso de la especie *Tityus carrilloi* se tiene como agravante, que las poblaciones al sur de la Provincia de Santa Fe, son partenogenéticas. En otras palabras, las hembras producen hembras sin necesidad de machos, lo que las hacen muy eficientes a la hora de colonizar nuevas áreas.

Y en el caso particular de la ciudad de La Plata, la presencia de escorpiones representa un problema sanitario cada vez más importante con focos específicos registrados en distintos puntos de la ciudad que se mantienen activos durante todo el año. Específicamente, en el partido de La Plata, pueden encontrarse dos géneros de escorpiones: *Tityus* y *Bothriurus*. El primero es de importancia sanitaria y se ubica principalmente en áreas urbanas, mientras que el segundo no es peligroso para los humanos y se encuentra mayormente en áreas rurales y periurbanas. Si bien ambas especies tienen muchas similitudes, también tienen algunas pequeñas diferencias,

especialmente en lo que respecta a su morfología. En general, es difícil para el sistema de visión humano reconocer en poco tiempo estas diferentes características morfológicas.

El Centro de Estudios Parasitológicos y de Vectores (CEPAVE) dependiente de la Universidad Nacional de La Plata (UNLP) y el CONICET, ha visto incrementadas las consultas sobre estos escorpiones desde el año 2005 al presente en forma exponencial, y muchas de las consultas han sido provenientes de organismos oficiales de la Provincia de Buenos Aires, como Ministerios de Infraestructuras, Jefatura de Policía, Jardines Maternales, Jardines de Infantes, Escuelas Primarias y Secundarias, como así también de casas particulares. Así lo atestiguan los diarios locales, los cuales han incrementado la cantidad de notas referentes a la aparición e incluso picadura de estos arácnidos. La Fig. 1 muestra una imagen de un ejemplar que fue motivo de un artículo en un diario local de la ciudad de La Plata [2]. Ha llegado a ser tan alta la tasa de hallazgos que entes oficiales han tenido que brindar información sobre las prevenciones que se deben tomar en cuenta, un ejemplo es la UNLP en el año 2019 [3]. También lo atestiguan artículos científicos, tanto a nivel local como nacional [4], [5].

Fig. 1 Artículo de diario El Día de La Plata

En consecuencia, hoy en día, uno de los principales desafíos para la comunidad científica relacionada con estos temas, es el de mejorar la eficiencia en la detección y reconocimiento de estos arácnidos altamente peligrosos.

Asimismo, resulta muy importante el acceso al reconocimiento de los diferentes tipos de escorpiones por parte de las personas ajenas a la materia. Esta información, proporcionada de forma certera y rápida, puede salvar vidas de haber ocurrido una picadura por parte de una especie peligrosa para el ser humano. Tomando como caso de estudio el partido de La Plata, la dualidad de géneros peligrosos y no peligrosos

genera la necesidad de contar con un sistema rápido y preciso que pueda no solamente detectar, sino también identificar cuál es el espécimen encontrado y evaluar su peligrosidad.

Un desarrollo previo ha servido como punto de partida en la búsqueda de los objetivos planteados en esta Tesis [6].

Uno de los aportes originales que se alcanzaron con el desarrollo de la Tesis Doctoral es el de contribuir a la seguridad sanitaria de una problemática local o regional, a partir de la generación de herramientas que permiten detectar escorpiones y reconocer su peligrosidad, de manera temprana, lo cual es esencial para minimizar las picaduras de estos arácnidos en ámbitos sensibles, como son los Jardines Maternales, Jardines de Infantes, Escuelas Primarias y Secundarias, como así también en casas particulares.

Por otro lado, la Tesis ha proporcionado un aporte tecnológico a través del diseño e implementación de un novedoso sistema autónomo para la detección, clasificación y reconocimiento en tiempo real de escorpiones, utilizando heurísticas de Aprendizaje Profundo, considerando las características de forma y color (detección de fluorescencia para detección y o confirmación), para ambientes controlados y no controlados, permitiendo diferenciar a aquellos escorpiones peligrosos para las personas, de aquellos que son inofensivos, dadas las propiedades de su veneno. Se ha confeccionado además una aplicación para teléfonos inteligentes para otorgarle portabilidad al sistema anteriormente mencionado. Esta aplicación puede ser utilizada como una herramienta de ayuda tanto para servicios de emergencia o bien con fines de investigación biológica. El sistema desarrollado es fácilmente escalable a otros géneros y especies de escorpiones, lo cual posibilita ampliar la región donde podrá ser utilizada esta aplicación. Si bien en la literatura se han propuesto diferentes métodos de detección de escorpiones, de acuerdo con nuestro conocimiento, hasta el momento, no se han reportado heurísticas de Aprendizaje Profundo para detectar e identificar estos arácnidos.

Finalmente, otro de los aportes originales de la Tesis, ha sido la implementación de las heurísticas de Aprendizaje Profundo más eficientes hasta el momento para el sistema desarrollado, aplicadas al procesamiento digital de imágenes en tiempo real. Para ello se realizaron estudios comparativos entre los modelos de Aprendizaje Profundo más relevantes hoy en día, tanto para la detección como para la clasificación y reconocimiento de escorpiones. Dichas heurísticas fueron comparadas utilizando diferentes criterios tales como: exactitud, precisión, velocidad de convergencia, complejidad de implementación, etc.

1.2. Objetivos

Los objetivos generales de la Tesis son:

- Generar conocimiento para la detección, clasificación y reconocimiento de escorpiones, mediante heurísticas de Aprendizaje Profundo, con el fin de encontrar y proponer nuevas soluciones o mejoras a las existentes.

- Desarrollar e implementar sistemas automáticos y en tiempo real para la detección, clasificación y reconocimiento de escorpiones, de una manera temprana, precisa, no invasiva y segura, bajo diferente iluminación y en distintos ambientes, con fines de prevención y control sanitario.

- Desarrollar e implementar un sistema de confirmación de detección por forma de escorpiones mediante la detección del color de su fluorescencia.

- Implementar los sistemas entrenados en una plataforma portátil como ser una aplicación para teléfono inteligente.

1.3. Organización de la Tesis

El presente trabajo está organizado en siete capítulos.

En el siguiente capítulo se introducen las nociones generales sobre los escorpiones, donde se presentan conceptos biológicos fundamentales de estos arácnidos, así como su distribución a nivel mundial, en América Latina y en el partido de La Plata. Se describe además en el Capítulo 2, la propiedad de fluorescencia que distingue a los escorpiones de otros artrópodos, que les permite brillar en la oscuridad cuando son iluminados con luz ultravioleta. El Capítulo 2 finaliza con una revisión de los métodos de detección de escorpiones previamente presentados en la literatura.

En el Capítulo 3 se describen los fundamentos teóricos del Aprendizaje Automático y del Aprendizaje Profundo, poniendo énfasis en las Redes Neuronales Artificiales. Se presentan además los conceptos generales para la segmentación para la detección de objetos y clasificación de imágenes, conjuntamente con las heurísticas denominadas Aprendizaje por Transferencia (*Transfer Learning*) y Aumento de Datos (*Data Augmentation*). Las métricas utilizadas para la evaluación de los modelos bajo estudio son presentadas al final del Capítulo 3.

El Capítulo 4 presenta las herramientas tecnológicas utilizadas para la implementación de los diferentes modelos analizados, desde las librerías utilizadas, pasando por la etapa de preparación de las bases de datos y la etapa de entrenamiento. Asimismo, se describen las herramientas para la implementación de los modelos óptimos en aplicaciones de teléfonos móviles, así como los dispositivos que conforman el sistema analógico de detección.

Los Capítulos 5 y 6 muestran el desarrollo propio y los resultados obtenidos para la obtención de los objetivos propuestos en la Tesis, respectivamente.

En particular, en el Capítulo 5 se presentan los sistemas desarrollados de detección y clasificación de escorpiones. Por un lado, para el sistema de detección del objeto escorpión, se han implementado y comparado cuatro métodos diferentes: con el clasificador en cascada (Haar), con el modelo YOLO (You Only Look Once), con el modelo MobileNet, y la detección mediante la propiedad de fluorescencia. En este último caso, con respecto a la detección mediante fluorescencia se implementaron dos sistemas diferentes, uno basado en la detección directa, y el otro como sistema de confirmación o validación cuando un escorpión ha sido detectado por alguno de los métodos anteriores. Y, por otro lado, para el sistema de clasificación de escorpiones (ya sea clasificación por género peligroso o para determinar su especie dentro de un mismo género), se han implementado y comparado tres modelos diferentes: LBPH, VGG16, y MobileNet. A su vez, durante este Capítulo se presenta la implementación de dos aplicaciones para teléfonos inteligentes desarrolladas con MobileNet para la detección y clasificación de escorpiones. Sobre el final del Capítulo 5 se presentan dos subsecciones en las que se muestra la implementación del sistema analógico de detección, y la instalación de algunos de los sistemas de detección desarrollados en un edificio público de la ciudad de La Plata.

Por su parte, el Capítulo 6 presenta los resultados de los ensayos realizados con los diferentes métodos presentados en el Capítulo 5. En primer lugar, se describen los resultados obtenidos con los sistemas de detección del objeto escorpión basados en el clasificador en cascada, en los modelos YOLO (v3 y v4) y MobileNetV2, y en la detección por fluorescencia. Y, en segundo lugar, se presentan los resultados obtenidos con los sistemas de clasificación de escorpiones basados en los modelos LBPH, VGG16 (con y sin uso de Data Augmentation) y MobileNet.

Finalmente, el Capítulo 7 expone las conclusiones de la Tesis y las líneas de trabajo a futuro.

1.4. Lugar de desarrollo de la Tesis

El trabajo se realizó con el asesoramiento y colaboración de dos centros afines a las temáticas abordadas durante el mismo.

La tesis se desarrolló en el área CeTAD del Grupo de Control Aplicado (GCA) perteneciente al Instituto de Investigaciones en Electrónica, Control y Procesamiento de Señales (LEICI – CONICET – FI, UNLP), el cual proporcionó el asesoramiento de los conceptos de electrónica necesarios para el desarrollo. El LEICI es un Instituto de investigaciones científicas dependiente del Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) y de la Facultad de Ingeniería (FI), Universidad Nacional de La Plata (UNLP). El área CeTAD realiza tareas de investigación en electrónica e

informática, impulsando la formación de recursos humanos en áreas tales como: co-diseño hardware-software, microelectrónica y computación en paralelo.

Por otro lado, se tuvo la colaboración muy importante del Laboratorio de Aracnología del Centro de Estudios Parasitológicos y de Vectores (CEPAVE – CONICET – FCNyM, UNLP), al cual asistí, en calidad de pasante, para realizar los ensayos necesarios, y así verificar el correcto funcionamiento de la alarma ante la presencia de los escorpiones. El CEPAVE es un Centro de investigaciones científicas dependiente del CONICET y de la Facultad de Ciencias Naturales y Museo (FCNyM), UNLP. El Centro tiene como objetivo realizar investigación científica sobre la biología y ecología de parásitos, parasitoides, depredadores y patógenos de invertebrados y vertebrados de importancia sanitaria y económica, así como de plagas agrícolas y arácnidos.

Capítulo 2: Nociones generales sobre los escorpiones

2.1. Conceptos biológicos

Los escorpiones son invertebrados del Filum artrópoda que se agrupan en la Clase Arácnida (Fig. 2), la cual incluye a las arañas, opiliones, ácaros, solífugos, pseudoescorpiones y escorpiones [7], [8], más otros grupos menos conocidos por el común de la gente.

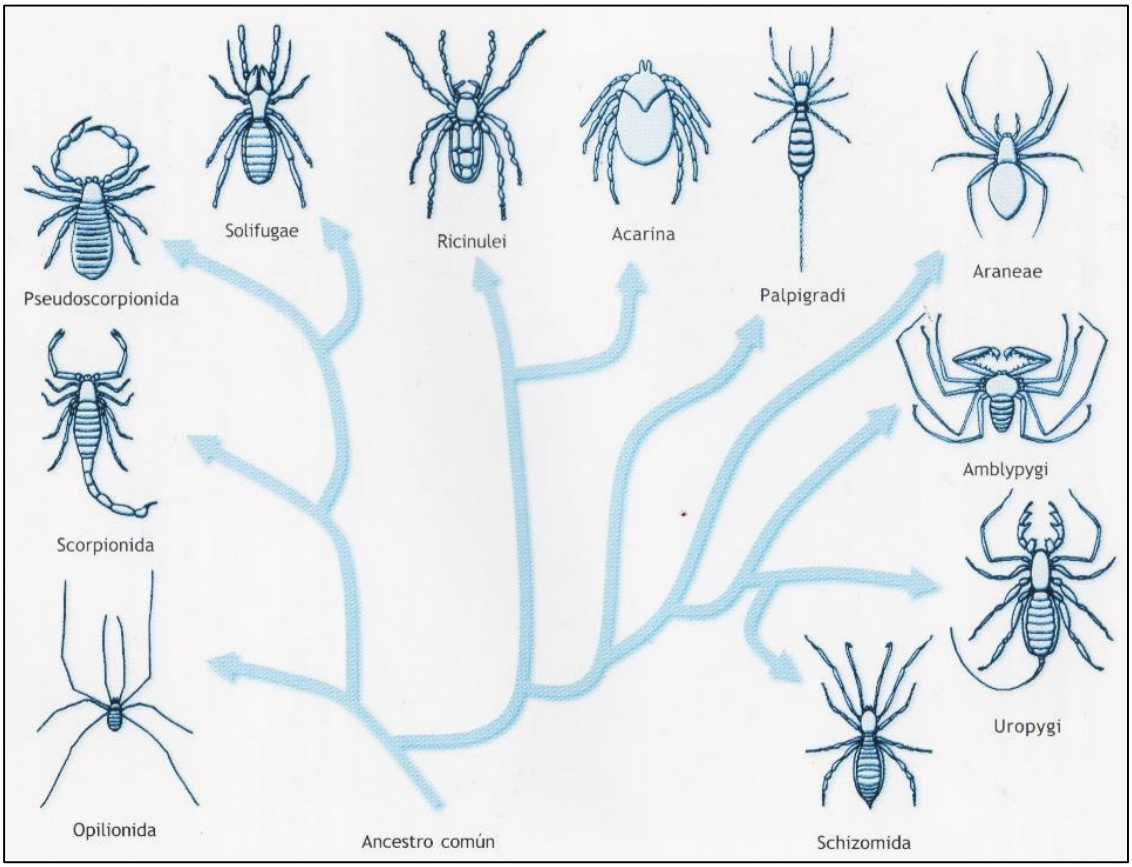


Fig. 2 Relación entre los órdenes de arácnidos.
Fuente: *E. Blanco y G. Salas, Arácnidos guía de campo*

Los artrópodos tienen en común un exoesqueleto quitinoso que actúa de sustento. Poseen apéndices, los cuales tienen funciones diversas, como mandíbulas, palpos, pero lo que le da nombre al grupo son los apéndices que constituyen las patas, ya que éstas están articuladas, de allí el origen griego de la palabra, árthron (articulación) y podia (piernas). En la Fig. 3 se puede observar un esquema que facilita la determinación de los diferentes tipos de artrópodos.

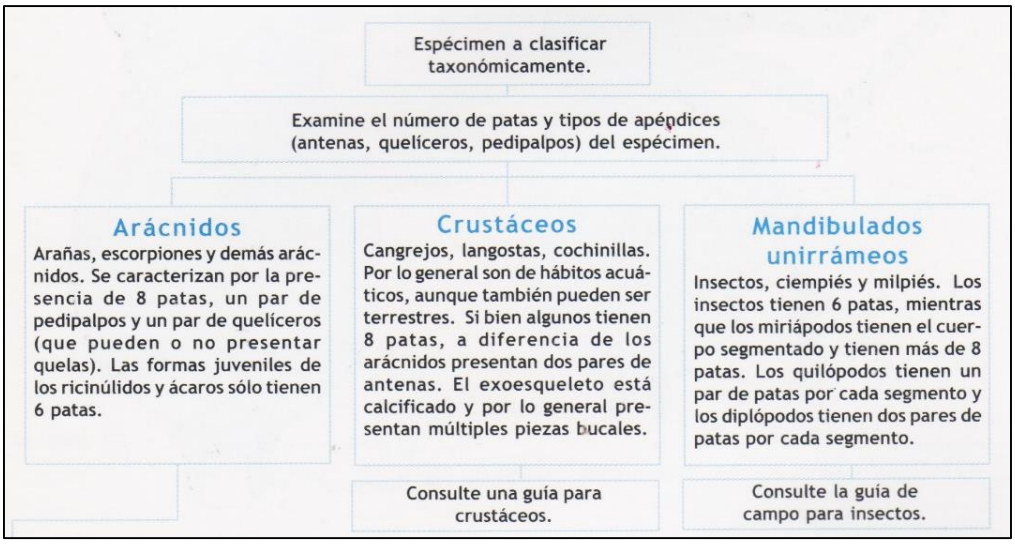


Fig. 3 Esquema de determinación de artrópodos.
Fuente: *E. Blanco y G. Salas, Arácnidos guía de campo*

Por otra parte, la clase Arácnida tiene como carácter en común el tener ocho patas articuladas. Y dentro de esta clase, los escorpiones tienen en común un cuerpo dividido en una zona anterior donde se encuentran la parte cefálica y toráxica (cefalotórax), que contiene los ojos, ocelos, se articulan las pinzas y las mandíbulas; un cuerpo donde se insertan las patas; y finalmente una cola (metasoma) compuesta de cinco segmentos seguida del telson que posee un aguijón (Fig. 4). Es precisamente en esta porción final donde se encuentran un par de glándulas del veneno y el aparato picador.

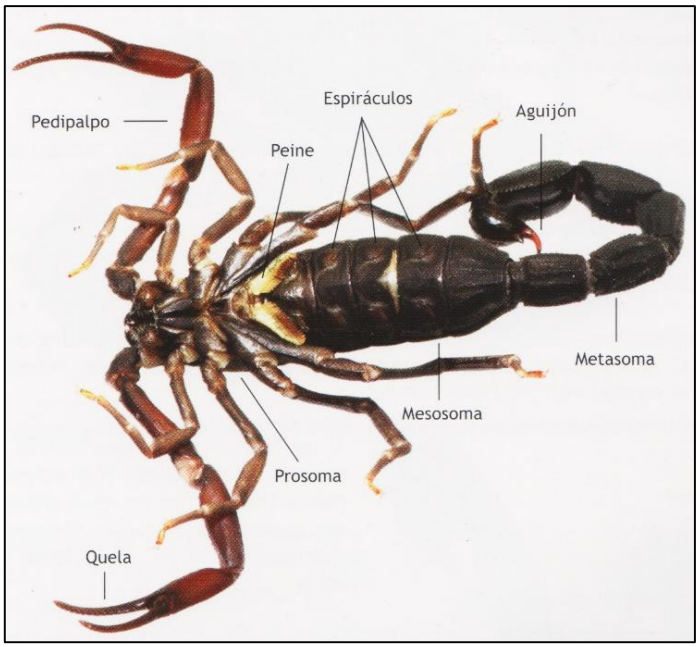


Fig. 4 Vista ventral de un escorpión.
Fuente: *E. Blanco y G. Salas, Arácnidos guía de campo*

Gracias a este veneno, que utilizan para paralizar a sus presas y posiblemente ayudar a la predigestión es que se los conoce a los escorpiones como un grupo peligroso, si bien son pocas las especies en el mundo que pueden causar un accidente de gravedad al ser humano.

A nivel mundial algunas de las especies peligrosas pertenecen a la familia Buthidae, de los géneros *Androctonus, Buthus, Leiurus, Mesobuthus* y *Parabuthus*, en África y Medio Oriente, *Centruroides* principalmente en México y sur de EE.UU. Y *Tityus* en América del Sur.

Para poder visualizar la distribución de estos géneros, se realizó en esta tesis, un mapeo mundial de los mismos, con el propósito de tomar conocimiento de aquellos sitios en dónde se deben concentrar esfuerzos en medidas de prevención y concientización para impedir posibles picaduras que pueden llegar a ser mortales en el peor de los casos.

Para ello se recurrió a "The Global Biodiversity Information Facility" (GBIF) [9] para adquirir los metadatos correspondiente a las familias y géneros previamente mencionados.

El sistema GBIF es una red internacional e infraestructura de investigación financiada por los gobiernos del mundo, y cuyo objetivo es proporcionar a cualquier persona, en cualquier lugar, acceso abierto a datos sobre todo tipo de vida en la Tierra.

El software utilizado para el mapeo fue el QGIS [10], el cual es un Sistema de Información Geográfica de software libre, que fue implementado en esta tesis no solo para el mapeo de los metadatos utilizados, sino también para poder visualizar la densidad geográfica de dicha información y así poder sacar mejores conclusiones que nos lleven a realizar un plan de prevención más acorde a la necesidad mundial. Su entorno de desarrollo se basa en Python, disponiendo de la consola para utilizar de manera complementaria.

Las muestras analizadas corresponden a diferentes bases de datos alojadas en el mencionado repositorio [11]–[19], con una distribución de muestras que se puede observar en la Tabla 1. De dichas muestras se obtuvo la distribución mostrada en la Fig. 5.

Nombre de base de datos	Cantidad de muestras
Parabuthus	1454
Mesobuthus	558
Leirus	176
Buthus	1184
Androctonus	845
Tityus serrulatus	71
Leirus quinquestriatus	141
Centruroides suffusus	403
Centruroides sculpturatus Ewing	272
Androctonus crassicauda	127
Tityus Koch, 1836	2605

Tabla 1 Información de base de datos.

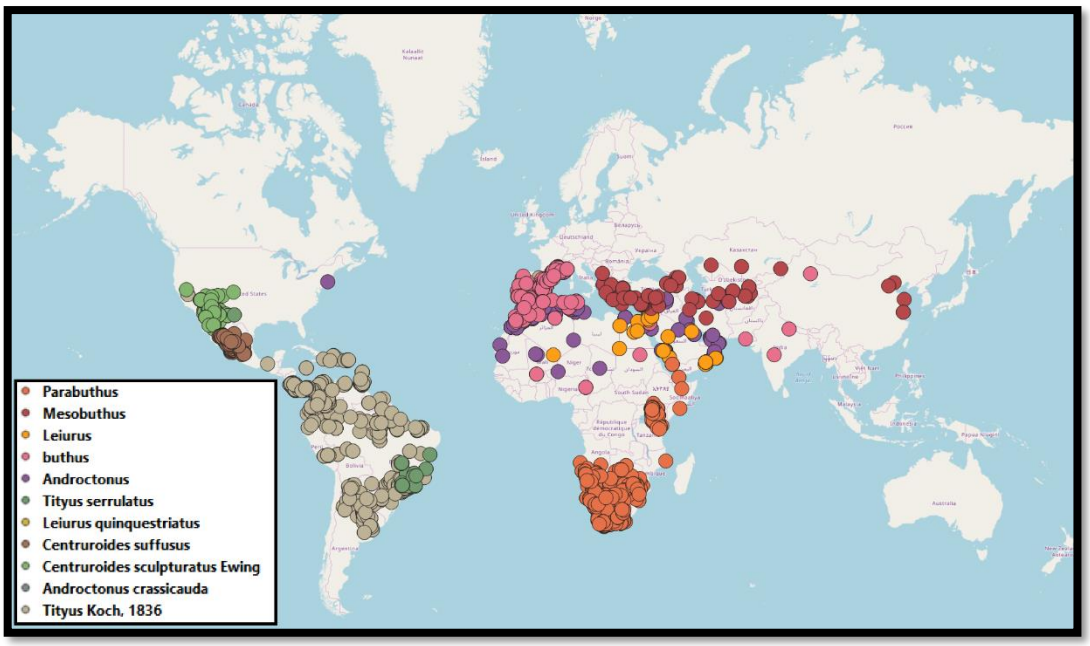


Fig. 5 Distribución mundial de escorpiones según base de datos consultada

Se puede ver que la distribución de estos géneros se circunscribe a las zonas tropical y subtropical. Al igual que con el caso anterior, se procedió a fusionar estos metadatos para poder crear una capa dentro del programa que mostrase el mapa de calor de la presencia de escorpiones de importancia sanitaria a nivel global (Fig. 6).

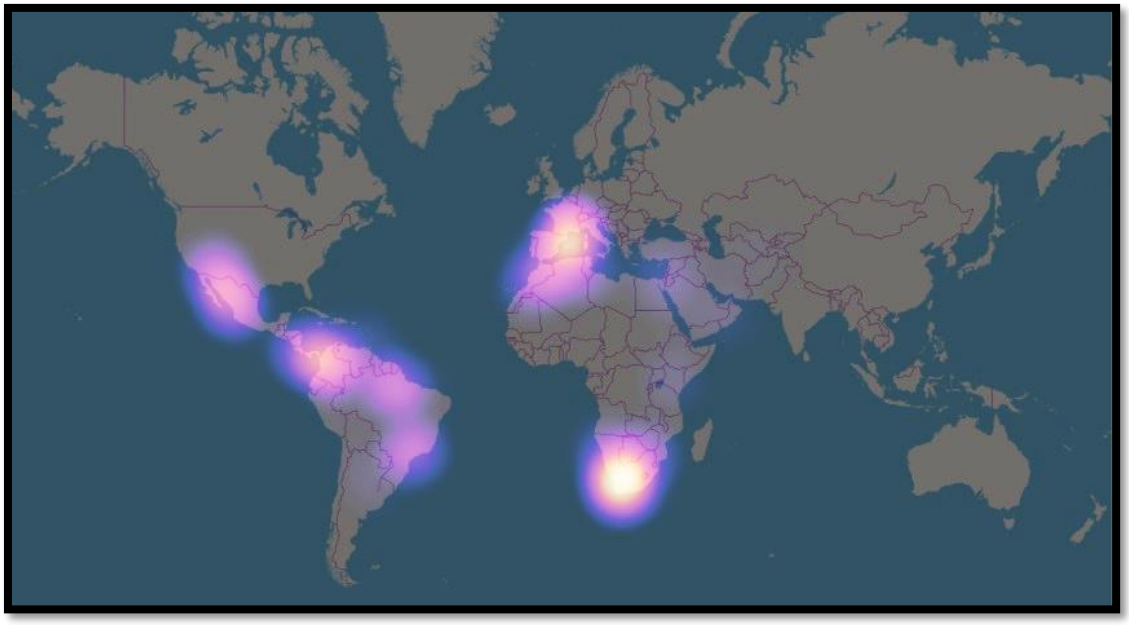


Fig. 6 Mapa de calor mundial de la distribución de escorpiones peligrosos

Habiendo realizado el mapa de calor correspondiente, podemos percibir que se han sumado dos nuevas zonas de riesgo que merecen nuestra mayor atención, como es el caso de Sudáfrica y la zona costera de Europa con el Mediterráneo, cabe destacar que también se denota una zona de cierta relevancia, al igual que la Argentina, que es el Medio Oriente y el norte de África.

Los géneros agregados a nuestra base de datos que componen estas zonas de importancia sanitaria son *Parabuthus* (Sudáfrica) y Buthus (Europa). En los casos de menor densidad de muestras de la base de datos tenemos: *Androctonus* (norte de África) y *Mesobuthus, Leiurus y Androctonus* (Medio Oriente).

Cabe aclarar que esta actividad se desarrolló en el marco de incrementar la capacidad de análisis de metadatos para una aplicación de nuestro interés. Los datos recopilados se basaron en las especies mencionadas y para ello la búsqueda se enfocó únicamente en el repositorio GBIF, por lo que la distribución real de especies de interés sanitario es posiblemente más amplia, dado que no necesariamente todos los países del mundo suben información a este repositorio en particular.

Un claro ejemplo de la ausencia de datos es Australia, país que posee escorpiones, sin embargo no posee especies peligrosas para las personas. Otra fuente de información disponible para consultar acerca de la distribución de las especies de escorpiones en esta y otras regiones del mundo es el sitio “iNaturalist” [20].

Otras opciones que se evaluaron en la tesis fueron, la implementación de complementos (*plugins*) para QGIS que permiten realizar adquisición de datos mediante una interfaz de programación de aplicaciones (*API: Application Programming Interface*) de redes sociales. En base a esto se decidió utilizar el plugin de Flickr [21], que es una

red social de almacenamiento de fotografías usualmente utilizada por fotógrafos profesionales y aficionados, que permite realizar una búsqueda mundial de imágenes georreferenciadas de escorpiones a fin de obtener una distribución geográfica de los mismos. Evaluando de manera exhaustiva, se detectó que la mayoría de las imágenes que los usuarios de esta red social habían etiquetado como escorpiones, correspondían a representaciones de los mismos, ya sea dibujos, tatuajes, o incluso, en el caso más extraño, a un escorpión hecho con una toalla de baño (imagen que llamó la atención por su ubicación en el medio del Océano Pacífico). Posteriormente, se decidió intentar lo mismo con la API de Twitter [22], pero éste solo analiza los mensajes (*tweet*) en tiempo real buscando las palabras claves que elijamos, por lo que no nos provee de una base de datos inicial, demandándonos mucho tiempo de espera para generar una base de datos significativa para analizar.

Es por este tipo de experiencias con redes sociales que se decidió recurrir directamente a un repositorio online que se aboque a la temática, por más que su muestreo pueda estar sesgado a su uso en cada región.

En la Argentina sen encuentran cerca de 60 especies de escorpiones [23] pertenecientes a dos familias, la Bothriuridae con la mayor cantidad de géneros (*Bothriurus, Timogenes, Brachistosternus, Orobothriurus, Vachonia, Phoniocercus* y *Mauryius*), y la familia Buthidae con tres géneros (*Tityus, Zabius* y *Ananteris*). Sin embargo, como se mencionó previamente, solo al género *Tityus* se lo considera de importancia sanitaria en nuestro país

Las picaduras de escorpiones a las personas ocurren principalmente debido a encuentros accidentales con estos arácnidos, dado que solamente atacan en el caso de sentirse provocados o agredidos. En general, las personas son picadas cuando tienen un contacto involuntario con ellos, como por ejemplo, cuando una persona se viste sin conocer la presencia de un escorpión en su prenda, como un zapato o un pantalón, y al entrar violentamente en contacto con el escorpión, éste se siente amenazado y se defiende mediante una picadura inoculando su veneno.

Durante muchos años los escorpiones han sido objeto de estudio, debido a la alta peligrosidad de algunas especies [24]. Razón por la cual la necesidad de la detección e identificación de los mismos.

Hoy en día, muchas poblaciones de escorpiones han encontrado nuevos hábitats proporcionados por los humanos, debido a la modificación del entorno que hace unos cientos de años no existía, como son las grandes ciudades. Especialmente, esto se puede observar muy bien en diferentes ciudades, donde la buena disponibilidad de alimento (insectos y otros arácnidos), el calor y los refugios, las han convertido en un sitio nuevo y apropiado para ser colonizado por algunas especies de escorpiones, los cuales son llamados antropogénicos.

La ciudad de La Plata, Provincia de Buenos Aires, Argentina, fundada en el año 1882, es un buen ejemplo de lo anterior. En el Partido de La Plata se encuentran dos géneros de escorpiones: *Tityus* y *Bothriurus*, los cuales se cree que han llegado a la ciudad a partir

de mediados del siglo XX, por antropocoria. Específicamente, dos especies de *Tityus* (*Tityus carrilloi* y *Tityus confluens*) y una especie de *Bothriurus* (*Bothriurus bonaeriensis*) cohabitan en la ciudad de La Plata. Las dos especies de *Tityus* son de importancia sanitaria y se las puede encontrar en zonas urbanas, por su parte, el *Bothriurus* es un género no peligroso, que se encuentra principalmente en áreas rurales y periurbanas, de baja densidad poblacional.

En las Fig. 7, Fig. 8 y Fig. 9, se pueden observar imágenes de ejemplares de *Tityus carrilloi*, *Tityus confluens* y *Bothriurus bonariensis*, respectivamente. Si bien los géneros *Bothriurus* y *Tityus* tienen muchas similitudes, también tienen algunas diferencias, especialmente en lo que respecta a su morfología, dado que pueden observarse principalmente diferencias en la forma de sus colas y pedipalpos.

Fig. 7 Imagen de un escorpión Tityus carrilloi

Fig. 8 Imagen de un escorpión Tityus confluens

Fig. 9 Imagen de un escorpión Bothriurus bonariensis

Las especies *Tityus confluens y Tityus carrilloi* también se extienden por toda la zona del gran Buenos Aires y CABA (Ciudad Autónoma de Buenos Aires), siendo especies de gran expansión en los últimos años. Es de mencionar que la distribución geográfica de estas especies llega desde el NEA (Noreste Argentino), el centro de Argentina y su máxima distribución actual la alcanzó recientemente en Bahía Blanca [23]. Estas poblaciones tienen machos y hembras hasta Santa Fe, siendo las poblaciones más al sur compuesta por hembras que no necesitan machos para reproducirse, a esta forma reproductiva se la llama partenogenia y todas las crías son hembras.

Con el propósito de corroborar esta información, se buscó el género *Tityus* [11] dentro de la base de datos GBIF. En este caso se encontró una base de datos de hallazgos de escorpiones, principalmente en América (Fig. 10).

Fig. 10 Distribución de escorpiones Tytius en América Latina

Cabe aclarar que, esta base de datos al día de la fecha se encuentra desactualizada, debido a que se han encontrado escorpiones del género *Tityus* en el sur de la Provincia de Buenos Aires, entre otros lugares que no se ven reflejados en el análisis de nuestra base de datos. Así se ve reflejado en estudios a nivel nacional [5], [23].

Para tener un análisis más completo de esta zona del mundo, se buscó el género *Bothriurus* [25] (con 701 muestras), también presente en la Argentina (Fig. 11), pero sin importancia sanitaria.

Fig. 11 Distribución de escorpiones Bothriurus en América Latina

Como se puede observar la distribución de este género se da más al sur y en zonas menos pobladas, es decir que su distribución se da en zonas rurales.

Luego de haber sido relevada la información de interés para Argentina, se decidió aumentar nuestra base de datos e incorporar al género *Centruroides* [26], [27] para completar la presencia de escorpiones de importancia sanitaria en América. Una vez obtenidos dichos metadatos, se procedió a fusionarlos con las otras dos bases de datos y así realizar un mapa de calor de la presencia de escorpiones en América (Fig. 12).

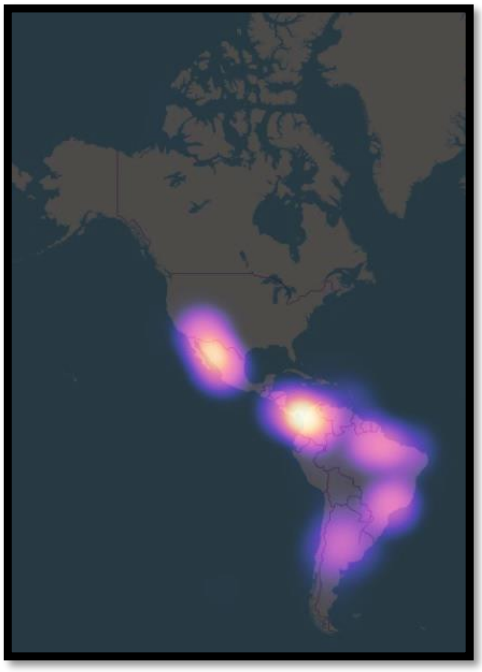


Fig. 12 Mapa de calor de la distribución de escorpiones en América

Queda claro que la mayor concentración de escorpiones se da en las zonas tropicales como ser el caso de Brasil, Panamá, Colombia, México, sur de Estados Unidos, y la Argentina. Se puede ver también que, pese a que todos los países previamente mencionados tienen una considerable presencia de escorpiones, el país que mayores precauciones requiere es el caso de México (el que posee 300.000 casos de picaduras al año).

Volviendo a la situación local, en los últimos años se han incrementado las consultas realizadas en el Laboratorio de Aracnología del CEPAVE (CONICET-UNLP), debido al aumento en la aparición de escorpiones en distintas zonas de la ciudad de La Plata. Las especies de consulta son *Bothriurus bonariensis, Tityus confluens* y *T. carrilloi* [28]. Como ya se mencionó previamente, la primera asociada a zonas rurales- periurbanas y las dos restantes a zonas urbanas, como puede observarse en la Fig. 13. Estas diferentes especies de escorpiones no solo pueden aumentar su población, sino que también pueden conquistar nuevas áreas.

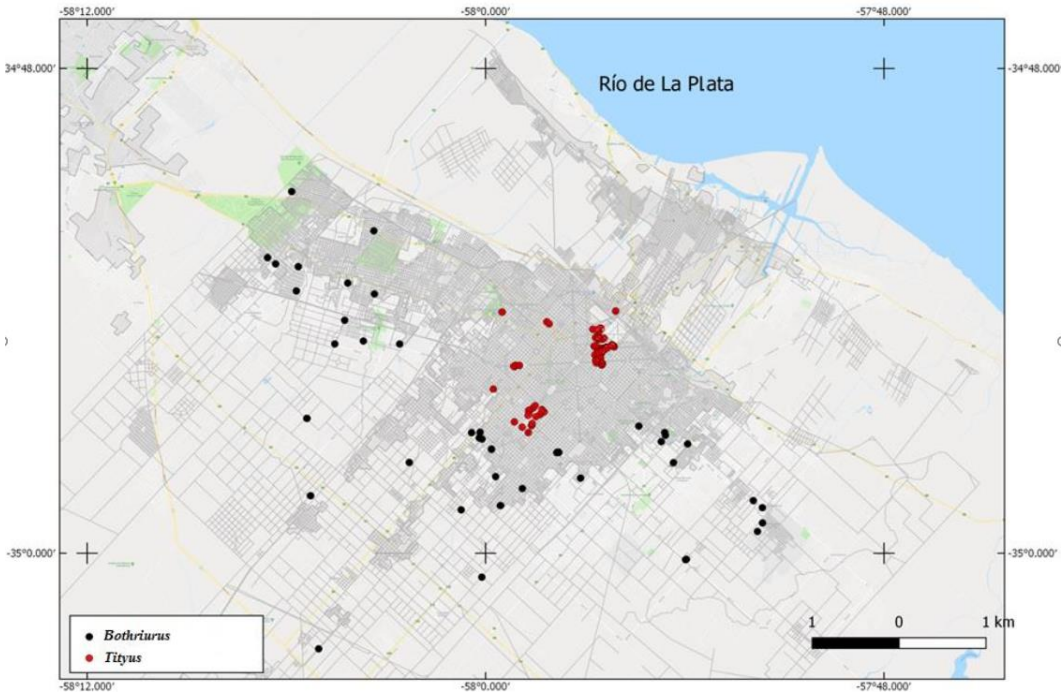


Fig. 13 Distribución de escorpiones en La Plata

2.2. Propiedad de fluorescencia

Una particularidad que caracteriza a los escorpiones es su fluorescencia. En efecto, la cutícula de los estos emite una fluorescencia de color cian (con una longitud de onda entre los 440 y los 490 nanómetros) cuando es iluminada con luz ultravioleta (UV) [29]. Este fenómeno fue descubierto casi simultáneamente en 1954 por el zoólogo italiano

M. Pavan y el zoólogo sudafricano R. F. Lawrence, y revolucionó el estudio de la biología y ecología de los escorpiones gracias a que fue posible localizarlos y observarlos por la noche, de manera relativamente no invasiva, usando lámparas de luz negra [30]–[32]. La Fig. 14 muestra dos imágenes del escorpión *Tityus carrilloi* bajo luz natural (izquierda) y bajo luz UV (derecha), donde puede ser observada la propiedad de la fluorescencia.

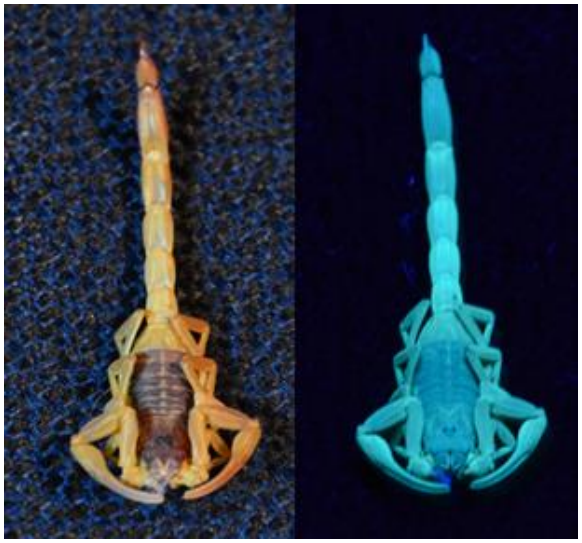


Fig. 14 Escorpión género Tityus con luz natural y UV

La fluorescencia en los escorpiones se debe a la existencia de dos compuestos químicos en la cutícula, conocidos como β-carbolina y 7-hidroxi-4-metilcumarina. Estos dos compuestos se encuentran en la exocutícula hialina, una región de la cutícula que en los escorpiones tiene un espesor de aproximadamente 4 micras. La cutícula es una especie de «piel» que protege a los artrópodos y al estar endurecida, también actúa como armazón que les permite mantener su forma.

La intensidad de la fluorescencia aumenta con la edad del escorpión y la dureza de su cutícula, siendo más brillante en las zonas más duras. Una vez adquirida, la fluorescencia persiste incluso después de la muerte del escorpión [33], [34]. Sin embargo, no se ha podido determinar aún de manera precisa la función de la fluorescencia de los escorpiones bajo luz UV. Al respecto, se han propuesto diferentes teorías, tales como que la fluorescencia puede ayudar a los escorpiones a capturar presas [35], atraer parejas, ahuyentar depredadores y rivales territoriales [33], identificar refugio o decidir cuándo permanecer en sus madrigueras [34], [36]. Además, algunos autores han sugerido que la fluorescencia puede no tener ningún propósito conductual [37], [38].

2.3. Métodos de detección de escorpiones

Los escorpiones son animales de actividad nocturna teniendo un fototropismo negativo. Se encuentran durante el día guarecidos, en ambientes naturales, bajo rocas o dentro de agujeros, pero algunas especies suelen ser domiciliarias, permaneciendo durante el día ocultas en la propia casa y deambulando de noche.

En la literatura, se han propuesto diferentes métodos de detección de escorpiones, sin embargo, hasta el momento, no han sido reportados métodos basados en heurísticas de Aprendizaje Automático para detectar e identificar escorpiones como los que son presentados en esta Tesis. Los métodos más antiguos de detección de escorpiones incluyen el balanceo de rocas, la detección de madrigueras, el desprendimiento de la parte posterior de los árboles y la trampa de caída, los cuales son peligrosos, consumen mucho tiempo y son invasivos [39], [40].

Otros métodos de detección utilizan diferentes características biológicas de los escorpiones. Por un lado, dado que el escorpión, al igual que otros artrópodos, utiliza señales de vibración del sustrato para reconocer y localizar parejas y presas [41], [42], esta información ha sido utilizada para el desarrollo de un sistema de detección de escorpiones que utiliza la técnica de detección de frecuencia de vibración [43]. Sin embargo, cabe aclarar que esta característica no aplica a todos los escorpiones. Ni el género *Tityus*, ni la especie *Bothriurus bonaeriensis* poseen aparato estridulatorio para producir estas señales [44], [45]. Esta es la razón por la que el sistema presentado en esta tesis no solo es novedoso, sino que el único método automático para detectar a las especies involucradas.

Por otro lado, para detectar escorpiones por la noche, se utiliza su característica de fluorescencia previamente mencionada, la cual proporciona una ventaja, dada la actividad nocturna de los mismos. Esta técnica es comúnmente aplicada, mediante el uso de linternas UV, para la recolección de ejemplares para estudios biológicos. Esta metodología es aplicada por el Laboratorio de Aracnología del CEPAVE en sus campañas de búsqueda y recolección.

Como único precedente de un sistema de detección automático de escorpiones, basado en la propiedad de fluorescencia, se encuentra un desarrollo propio [6], el cual consistió en el diseño e implementación de dos sistemas de alarma para la detección de escorpiones, la primera utilizando un sensor analógico de color y la otra utilizando el canal H (Hue) del formato de imagen HSV (Hue Saturation Value) de una imagen procesada morfológicamente. Sin embargo, el uso de luz ultravioleta hace que estos métodos sean eficientes solo de noche o sin luz diurna. Este desarrollo propio previo ha servido como punto de partida en la búsqueda de los objetivos planteados en la presente Tesis.

Capítulo 3: Conceptos generales para la detección y clasificación de objetos

3.1. Inteligencia artificial y Aprendizaje Automático

El concepto de Inteligencia Computacional (IC) se aplica a cualquier heurística que permita a las computadoras imitar la inteligencia humana a través de expresiones lógicas y esquemas abstractos [46]. Estas heurísticas pueden ser utilizadas para modelar, identificar, optimizar, predecir y controlar el comportamiento dinámico de diferentes sistemas reales.

Dentro de la IC se pueden observar cuatro diferentes enfoques [47]. Están aquellos ***sistemas que se comportan como humanos***, dentro de éstos se puede encontrar la Prueba de Turing, la cual establece que la máquina debe poseer las siguientes capacidades: procesamiento del lenguaje natural (posibilidad de comunicarse), representación del conocimiento (posibilidad de almacenar lo que se conoce), razonamiento automático (para accionar en base al conocimiento almacenado y extraer nuevas conclusiones), Aprendizaje Automático (para adaptarse a cambios, detectar y extrapolar patrones), visión computacional (para percibir objetos), y robótica (para manipular y mover objetos). También están aquellos ***sistemas que piensan como humanos***, que tienen capacidades cognitivas de toma de decisiones, resolución de problemas y aprendizaje. Otro enfoque viene dado por aquellos ***sistemas que piensan racionalmente***, es decir, los que encuentran las leyes que rigen el pensamiento, dentro de lo cual se encuentra la lógica. Y, por último, aquellos ***sistemas que se comportan racionalmente***, los cuales actúan con intención de obtener el mejor resultado, o en caso de incertidumbre, obtener el mejor resultado posible, sabiendo que una racionalidad perfecta (hacer siempre lo correcto) no es del todo posible en entornos complejos.

En los sistemas que se comportan como humanos se encuentra el campo del Aprendizaje Automático, también denominado Aprendizaje Máquina (*Machine Learning*), el cual es una heurística cuyo objetivo principal es lograr que las computadoras sean capaces de “aprender” a tomar decisiones sin la necesidad de ser programadas explícitamente [48]. Para ello, el Aprendizaje Automático se centra en el desarrollo de modelos de comportamiento, algunos por medio de heurísticos programables y otros no. El aprendizaje de las computadoras se refiere a la capacidad para identificar patrones en millones de datos y a través de ellos tomar decisiones o hacer una predicción acerca de comportamientos futuros de un ambiente o situación utilizando un análisis estadístico y teorías de probabilidad. De esta manera se permitirá resolver problemas de forma intuitiva y automatizada, sin que el mecanismo de elección se encuentre previamente programado.

Una de las áreas en donde se avanzó notablemente en los últimos años fue en la de segmentación para la detección de objetos y clasificación de imágenes. Esto se debe en su mayor parte al desarrollo de nuevas heurísticas tanto de Aprendizaje Automático (Machine Learning) como de Aprendizaje Profundo (Deep Learning), además de las innovaciones en el manejo de datos a gran escala (Big Data) y el aumento en la capacidad de cómputo mediante el uso de diferentes tecnologías como Computación en la Nube (Cloud Computing) o el uso de GPU (Unidad de Procesamiento Gráfico) para el análisis de información. Este avance puede verse en distintas áreas como: medicina, seguridad, turismo, finanzas, robótica, análisis de suelos, análisis climático, entre otras. Algunos ejemplos de aplicación son: el control de vehículos autónomos, la detección de rostros, detección de matrículas, diagnóstico de enfermedades, realidad aumentada, etc.

El Aprendizaje Automático es un subcampo de la IC en el que se utilizan diferentes algoritmos que sean capaces de aprender de su entorno, a partir de un conjunto de datos que el algoritmo recibe en la etapa de entrenamiento [48], [49]. Las características principales de los algoritmos de Aprendizaje Automático es que son capaces de resolver problemas no lineales; aprender de ejemplos; encontrar correlaciones entre datos diversos; manejar diferentes tipos de datos (numéricos, textuales, imágenes, etc.); tratar con grandes conjuntos de datos y/o conjuntos de datos de alta dimensión; ser tolerantes a fallos, es decir sobrellevar el ruido y datos incompletos o atípicos; realizar predicciones y generalizaciones a altas velocidades; y realizar procesamiento en tiempo real.

Con las heurísticas de Aprendizaje Automático se están consiguiendo resultados que antes no era posible obtener con los métodos tradicionales. Sin embargo, no existe una única heurística óptima para todos los problemas. Cada caso debe analizarse por separado y de acuerdo con los requisitos del problema, se debe aplicar la heurística más adecuada.

Se distinguen tres tipos principales de heurísticas conexionistas de de Aprendizaje Automático: las de aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo. En el primero de ellos, los modelos son entrenados a partir de un conjunto de datos en el que la respuesta correcta es conocida. Es decir, se presenta a la red un conjunto de patrones de entrada junto con la salida esperada, y los pesos se van modificando de manera proporcional al error que se produce entre la salida real de la red y la salida esperada. Por su parte, en el aprendizaje no supervisado, el conjunto de datos empleado en el entrenamiento no contiene la respuesta, de modo que no existe un resultado que reproducir. En este caso los modelos se ajustan a las observaciones dadas y el objetivo que se persigue es el de modelizar la distribución de los datos para conocer más sobre ellos, por ejemplo, para determinar posibles patrones ocultos o para agrupar datos de acuerdo con un cierto criterio. Finalmente, en el aprendizaje por refuerzo, el algoritmo de aprendizaje recibe algún tipo de valoración (recompensa) acerca de la idoneidad de la respuesta dada. El objetivo en este caso es el de extraer qué

acciones deben ser elegidas en los diferentes estados para maximizar la recompensa en base a experiencias pasadas.

Dentro del aprendizaje supervisado, se cuenta con algoritmos de clasificación y de regresión. Los algoritmos de clasificación se usan cuando el resultado es una etiqueta discreta. Esto quiere decir que se utilizan cuando la respuesta se fundamenta en conjunto finito de resultados. Por el contrario, el análisis de regresión es útil para predecir productos que son continuos, esto significa que la respuesta al problema se presenta mediante una cantidad que puede determinarse de manera flexible en función de las entradas del modelo.

El desarrollo de la Tesis se enfocará específicamente sobre algoritmos de aprendizaje supervisado de clasificación, en los cuales, el resultado obtenido es una clase (categoría), entre un número limitado de clases, de acuerdo con el problema específico, por ejemplo: si-no; especie peligrosa-especie no peligrosa; etc. Entre las heurísticas de Aprendizaje Automático que se utilizan en los problemas de clasificación se encuentran: máquinas de vectores de soporte, árboles de decisión, bosques aleatorios y redes neuronales. En este trabajo se hará énfasis en el análisis y utilización de diferentes redes neuronales.

3.2. Redes neuronales artificiales

Las redes neuronales artificiales son sistemas de procesamiento de información inspirados en el funcionamiento y operación de las neuronas de cualquier ente biológico. Están conformadas por un conjunto de elementos simples denominados neuronas artificiales, las cuales procesan información y constituyen las unidades básicas de las redes neuronales artificiales.

Las redes neuronales artificiales tienen ramificaciones y un nodo (Fig. 15), de manera similar a las redes neuronales presentes en la naturaleza. Hay ramificaciones de entrada al nodo procedentes de otras neuronas. Esta información se procesará en un nodo, y se generará una información de salida que se trasmitirán por las ramificaciones de salida a otras neuronas.

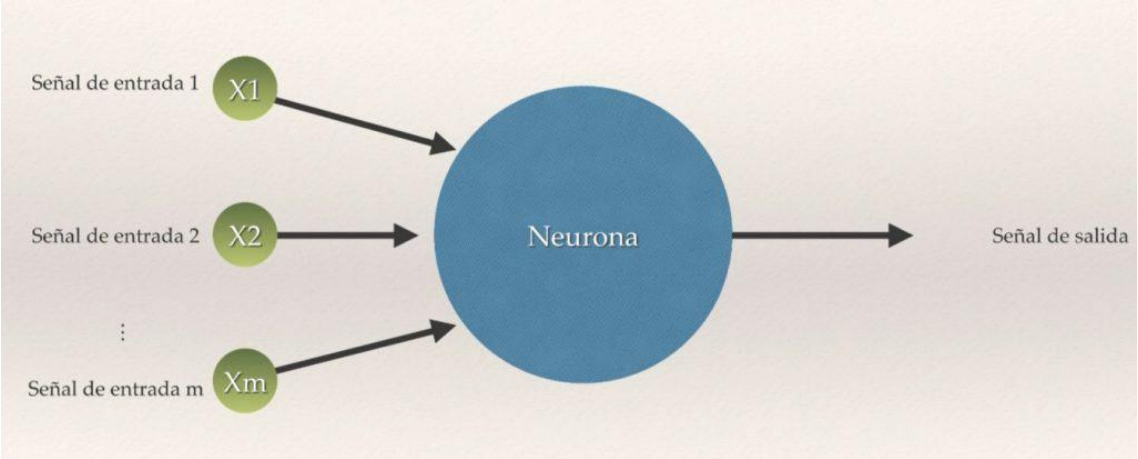


Fig. 15 Neurona artificial.
Fuente: https://www.iartificial.net/

En la Fig. 16 corresponde a un subtipo de red neuronal, la cual es una simplificación que no representa la red neuronal utilizada en esta tesis. Podemos ver que las ramificaciones de salida de algunas neuronas son las ramificaciones de entrada de otras, y así sucesivamente. Pero vemos un par de diferencias entre las capas. Las neuronas de color rojo son información que vamos a dar a las neuronas y las neuronas de color azul son la información de salida de la red neuronal. Dependiendo del número de capas ocultas (amarillas) podemos hablar de una red neuronal simple o profunda.

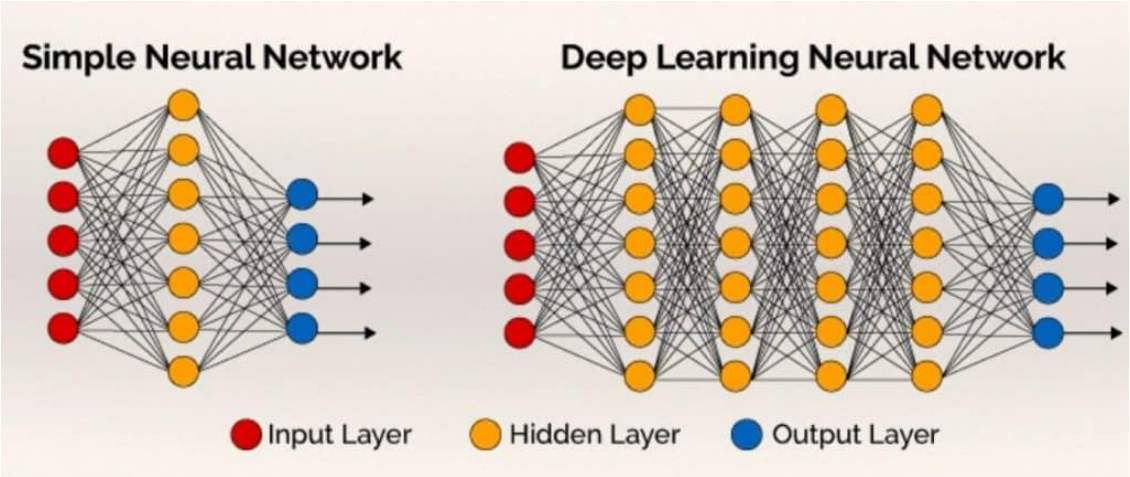


Fig. 16 Red neuronal simple y red neuronal profunda.
Fuente: https://www.iartificial.net/

Los valores de salida pueden ser continuos, como la probabilidad de que una imagen tenga un escorpión en ella. También pueden ser binarios, como por ejemplo si un escorpión es peligroso para las personas o no. O pueden ser una categoría, como el

género y especie de un escorpión. En este contexto, el caso binario es un caso particular donde tenemos dos categorías.

Para poder llevar a cabo el procesado, cada conexión entre neuronas tendrá asignado un valor o **peso**. Este peso equivale a la «fuerza» de la señal que se trasmite por cada sinapsis. El ajuste adecuado de los pesos es fundamental para tener una red neuronal apropiada en búsqueda de la solución del problema bajo estudio.

En el núcleo de la neurona es donde se procesan las señales de entrada y los pesos (Fig. 17). Una de las formas es multiplicar cada señal de entrada por su peso y luego sumar todas las entradas, es decir, hacer una combinación lineal de los pesos de las conexiones y las entradas. Para la transmisión de la información, generada por dicha combinación lineal, a las conexiones de salida se utiliza una función de activación.

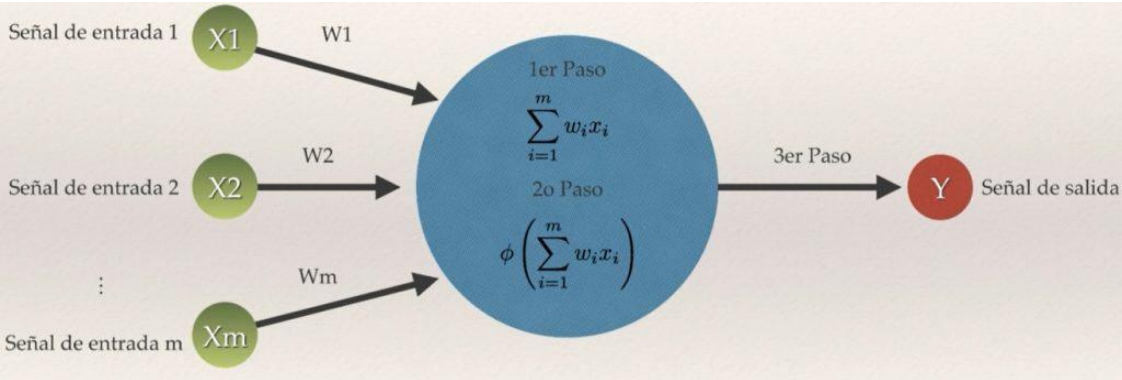


Fig. 17 Procesado de información en una neurona.
Fuente: https://www.iartificial.net/

Puede ser que nos interese transmitir esa información sin modificar, por lo que usaríamos como función de activación a la función identidad. Sin embargo, en general, las funciones de activación se utilizan para dar una «no linealidad» al modelo y que la red sea capaz de resolver problemas más complejos. Si todas las funciones de activación fueran lineales, la red resultante sería equivalente a una red sin capas ocultas.

A continuación, se mencionan las dos familias de funciones de activación utilizadas en esta tesis. Cabe aclarar que existen otras funciones las cuales no serán mencionadas.

-Función escalón (threshold)

Esta función propaga un 0 si el valor de x es negativo, o un 1 si es positivo (Fig. 19). No hay casos intermedios, y sirve para clasificar de forma muy estricta. Es la función más rígida.

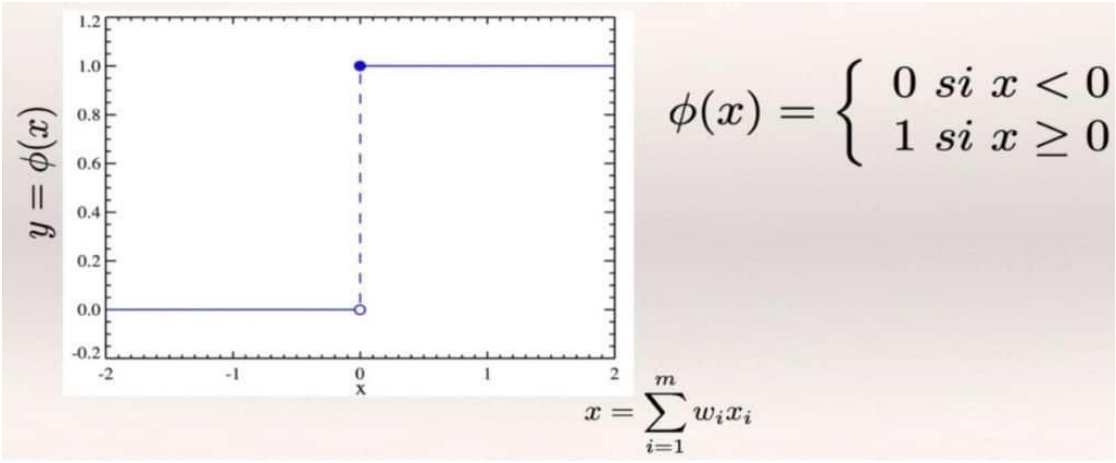


Fig. 18 Función escalón.
Fuente: https://www.iartificial.net/

-Función sigmoide

Como en la función escalón, existe una división entre los valores negativos y positivos de x, pero la función sigmoide (Fig. 19) no es tan estricta, el cambio se hace de manera suave. Se usa en la regresión logística, una de las heurísticas más usadas en Machine Learning. Esta función es muy útil en la capa final de salida al final de la red neuronal, no solo para clasificar con valores categóricos, sino también para intentar predecir las probabilidades de pertenencia a cada categoría, donde sabemos que la probabilidad de un suceso imposible es 0 y la de un suceso seguro es 1.

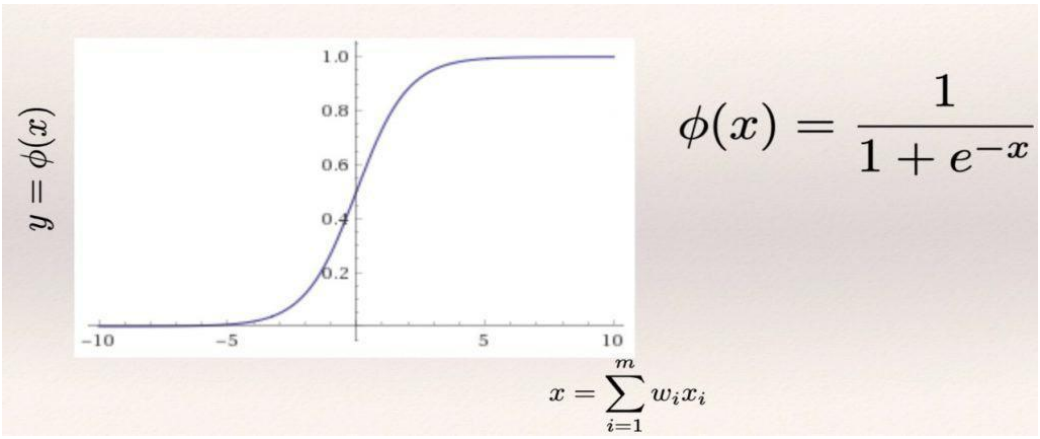


Fig. 19 Función sigmoide.
Fuente: https://www.iartificial.net/

3.3. Aprendizaje Profundo

El Aprendizaje Profundo (*Deep Learning*) es un subcampo del Aprendizaje Automático [50], con el cual se están consiguiendo resultados muy importantes especialmente en aquellas aplicaciones relacionadas con la detección de objetos y la clasificación de imágenes. Mediante el Aprendizaje Profundo, un modelo informático aprende a realizar tareas de detección y clasificación directamente a partir del reconocimiento de imágenes, texto o sonido, y pueden alcanzar niveles de precisión que, en ocasiones, supera el rendimiento humano.

El Aprendizaje Profundo [51], [52] se inspira en el proceso natural de aprendizaje de las neuronas del cerebro humano y emplea arquitecturas de redes neuronales, por lo que, a menudo, los modelos de Aprendizaje Profundo se denominan también Redes Neuronales Profundas. El término "profundo" hace referencia al número de capas ocultas en la red neuronal. Las redes neuronales tradicionales pueden contener una o dos capas ocultas, mientras que las redes profundas pueden tener decenas o incluso cientos de ellas.

Las aplicaciones del Aprendizaje Profundo actualmente son sumamente variadas, como la conducción autónoma, el sector aeroespacial, los dispositivos médicos, la automatización industrial, entre otras.

Si bien las primeras teorías sobre el Aprendizaje Profundo se desarrollaron en la década de los ochenta, existen dos razones principales por las que solo ha empezado a resultar útil recientemente: requiere de grandes cantidades de datos y de una potencia de cálculo significativa. Los modelos de Aprendizaje Profundo se entrenan mediante el uso de extensos conjuntos de datos y arquitecturas de redes neuronales que aprenden directamente a partir de los datos, sin necesidad de una extracción manual de características. Con estos modelos es posible obtener patrones o características simples a partir de entradas complejas.

En años recientes, las heurísticas de Aprendizaje Profundo han realizado aportes importantes en el procesamiento digital de imágenes, colaborando en la extracción de características de la imagen, el relevamiento y manipulación de información, ya sea para reconocimiento automático de patrones o para mejorar el rendimiento de la especialidad conocida como visión artificial o visión por computadora [53], por ejemplo para detectar, reconocer y clasificar objetos [54]–[57], para el reconocimiento facial [58], para analizar texturas [59], detección de cáncer [60], recuperación de imágenes [61], identificación de plantas [62], entre otras aplicaciones [63], [64].

Entre las heurísticas más utilizadas dentro del Aprendizaje Profundo se tienen las Redes Neuronales Profundas (*DNN: Deep Neural Networks*) y las Redes Neuronales Convolucionales (*CNN: Convolutional Neural Networks*). Las DNN tienen una estructura del tipo Perceptrón Multicapa (*MLP: Multi-Layer Perceptron*), compuesta por una capa de entrada en la cual la red recibe estímulos externos (datos de entrenamiento), una capa de salida que ofrece la respuesta de la red, y varias capas intermedias denominadas capas ocultas. Esta heurística, es sumamente versátil, dado que con ella se pueden

procesar textos, imágenes pequeñas y datos numéricos. El problema de esta heurística es que cuanto mayor sea el número de capas ocultas, la cantidad de conexiones entre neuronas aumenta de manera significativa, lo cual requiere contar con equipos de gran potencia de cálculo.

Por su parte, en las CNN, las neuronas de una capa solamente se unen con un subgrupo de ellas, buscando reducir el número de neuronas y el costo computacional. Estas redes convolucionan las características aprendidas con los datos de entrada y emplean capas convolucionales 2D, lo cual hace que esta arquitectura resulte adecuada para procesar datos bidimensionales, como imágenes, por lo que esta heurística es muy utilizada para tareas de clasificación de objetos. Básicamente, las CNN trabajan dividiendo y modelando la información en partes más pequeñas, y combinando esta información en las capas más profundas de la red. Por ejemplo, en el caso del tratamiento de una imagen, las primeras capas tratarían de detectar los bordes de las figuras, las siguientes capas buscarían combinar los patrones de detección de bordes para conseguir formas más simples y además se puede sumar conocimiento a partir de otros patrones como la posición de los objetos o la iluminación. Finalmente, en las últimas capas se podría hacer coincidir la imagen con todos los patrones descubiertos, para conseguir una predicción final considerando la suma de todos ellos. Así es como las CNN consiguen modelar una gran cantidad de datos, dividiendo previamente el problema en partes para conseguir predicciones más sencillas y precisas.

3.4. Visión por Computadora

La Visión por Computador o Visión Artificial es el conjunto de herramientas y métodos que permiten obtener, procesar y analizar imágenes del mundo real con la finalidad de que puedan ser tratadas por una computadora. La Visión Artificial intenta que las computadoras puedan percibir y comprender una imagen o secuencia de imágenes y actuar en consecuencia. Esta comprensión se logra mediante distintos campos como la geometría, la estadística, la física y otras disciplinas.

El objetivo principal de la Visión Artificial es el reconocimiento de patrones complejos en imágenes. Desde luego esta capacidad beneficia a diferentes campos, el más conocido es la robótica, ya que esta depende de la comprensión del mundo real para poder interactuar con él. Sin embargo, este no es el único campo que utiliza estas heurísticas, ejemplos de otros campos son el procesamiento de imágenes médicas [65], [66], los sistemas de seguridad [67], [68], el uso en la industria para detectar fallas en piezas fabricadas (control de calidad), el seguimiento de objetos, entre otros muchos campos los cuales se incrementan día a día [69].

Esto permite automatizar una amplia gama de tareas al aportar a las máquinas la información que necesitan para la toma de decisiones correctas en cada una de las tareas en las que han sido asignadas.

La Visión por Computadora tiene dos ramas fundamentales: la detección de objetos y la clasificación de imágenes, las cuales serán descriptas en las siguientes secciones.

3.5. Detección de objetos

Dentro de la Visión Artificial se encuentra la detección de objetos que, como su nombre lo indica, es la rama que estudia cómo detectar la presencia de objetos pertenecientes a una clase en particular dentro de una imagen (en nuestro caso, escorpiones). Esta detección se puede basar en la apariencia, el tipo, o la singularidad del objeto. Este proceso se compone de dos partes elementales, la extracción de características y la búsqueda del objeto basado en estas características.

Las características de una imagen se obtienen mediante modelos matemáticos que describan el contenido de la misma y, de esta manera, simplifiquen el aprendizaje del modelo. Estas características o descriptores proporcionarán una mayor o peor capacidad del sistema de detectar el objeto buscado en la imagen. Lo más desafiante en la extracción de características es encontrar descriptores y clasificadores invariantes a los cambios que pueda tener un objeto, como su posición, iluminación, u obstrucción parcial en la imagen.

Existen diferentes métodos para el aprendizaje, como la regresión logística, o más avanzados basados en heurísticas de Aprendizaje Automático como las máquinas de vectores de soporte (*SVM: Support Vector Machines*) o el algoritmo de impulso adaptativo (*Adaptive Boosting*, conocido como *AdaBoost*).

Esta última heurística se ha utilizado en diversas aplicaciones, las más conocidas son la detección de rostros y de personas. Una aplicación cada vez más común es su utilización en sistemas de video vigilancia para así poder dar aviso ante la presencia de un intruso (persona) en un lugar de interés.

3.6. Clasificación de imágenes

Otra de las ramas de la Visión por Computadora es la clasificación de imágenes, la cual se encarga de identificar una imagen a partir de las características de una clase en particular. Su principal aplicación es el reconocimiento facial, aplicable a sistemas de seguridad. En esta Tesis será utilizado para seguridad, más precisamente para prevención sanitaria, identificando a escorpiones peligrosos para el ser humano.

En la actualidad se cuenta con varias herramientas para la detección de objetos y la clasificación de imágenes, la mayoría de estos métodos son muy recientes, por lo que aún hoy se siguen mejorando y creando nuevas heurísticas que modifican el liderazgo de ellas.

Aunque existen sistemas muy buenos en sus campos respectivos, aún no existe ningún método que supere a todos los otros en todas sus virtudes. Por ejemplo, hay sistemas con una elevada exactitud (*accuracy*), pero velocidad de procesamiento muy lento, y otros sistemas que sacrifican parte de su exactitud para mejorar su velocidad haciéndolos ideales para el procesamiento en tiempo real, un objetivo buscado en esta tesis.

3.7. Clasificadores en Cascada

La detección de objetos es una tecnología utilizada en el procesamiento digital de imágenes para detectar clases de un objeto en particular. Normalmente implementado para la detección de personas, utiliza las características propias de la clase objeto para lograr la clasificación del mismo.

La detección de objetos mediante clasificadores en cascada basados en características de Haar es un método eficaz de detección de objetos propuesto por Paul Viola y Michael Jones [70], quienes lo implementaron originalmente en la detección de rostros. Es un enfoque basado en el Aprendizaje Automático en el que se entrena una función en cascada a partir de muchas imágenes positivas y negativas. Luego se usa para detectar objetos en otras imágenes.

El Clasificador en cascada se refiere a que debe pasar por diferentes etapas que son aplicadas secuencialmente con un proceso de optimización. Este método se caracteriza por ser más rápido, pero menos preciso que el uso de redes neuronales. Esto se debe a que utiliza menor capacidad de cómputo para funcionar. Otra ventaja es la posibilidad de crear un clasificador con una base de datos relativamente pequeña, todo lo que se tiene que hacer es asignar los pesos respectivos a las características pre establecidas en la estructura Haar.

Haar se caracteriza por extraer buenas características vecinas en una imagen [71], lo cual es de gran utilidad para la detección de objetos ocultos dentro de una imagen, método conocido como COD (Concealed Object Detection) [72], es por ello que es una heurística utilizada en este ámbito, incluso por encima de otras redes neuronales.

En esta tesis se desarrollará la detección de escorpiones. Inicialmente, el algoritmo necesita muchas imágenes positivas (imágenes de escorpiones) e imágenes negativas (imágenes sin ellos) para entrenar al clasificador. Este sistema extraerá las características de los escorpiones. Para ello, se utilizan características de Haar como las que se muestran en la Fig. 20, las cuales son el núcleo convolucional. Cada característica es un valor único obtenido al restar la suma de píxeles debajo del rectángulo blanco con la suma de píxeles debajo del rectángulo negro.

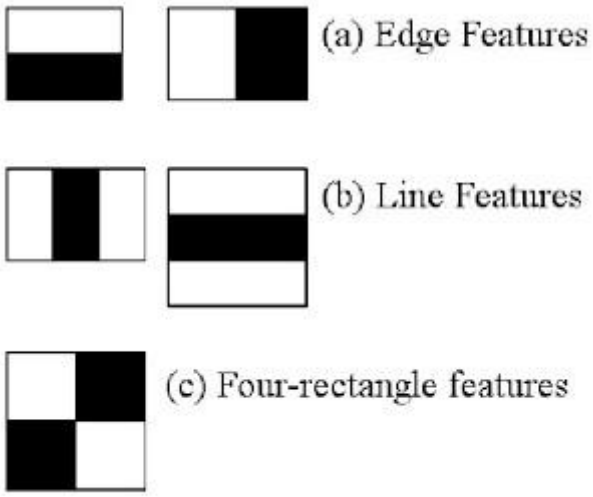


Fig. 20 Núcleos convolucionales de Haar

Cada uno de estos núcleos convolucionales son aplicados en todas las imágenes de entrenamiento. Para cada característica, se encuentra el mejor umbral que clasificará las caras en positivo y negativo. El proceso de entrenamiento selecciona las características con la menor tasa de error, lo que significa que son las características que clasifican con mayor precisión las imágenes de escorpiones.

Inicialmente, a cada imagen se le da el mismo peso. Después de cada clasificación, se aumentan los pesos de las imágenes mal clasificadas. Luego se realiza el mismo proceso, calculando las nuevas tasas de error y los nuevos pesos. El proceso continúa hasta que se alcanza la precisión o la tasa de error requerida, o bien hasta que se encuentra el número requerido de características.

El clasificador final es una suma ponderada de estos clasificadores débiles. Se llama débil porque por sí solo no puede clasificar la imagen, pero junto con otros, forma un clasificador fuerte.

La mayor parte de una imagen no posee al escorpión, y es por haber tomado este detalle en cuenta, que este sistema es meritorio de destacar y utilizar. Lo que caracteriza la velocidad de procesamiento y detección de este sistema es que utiliza un método sencillo para comprobar si una pequeña sección dentro de la imagen (ventana) no es un escorpión. Si no es así, se lo desecha de una sola vez y no se lo vuelve a procesar. En cambio, se concentra en las regiones donde puede haber un escorpión. De esta forma, dedicamos más tiempo a comprobar posibles regiones de interés.

3.8. Transfer Learning

El aprendizaje por transferencia (*Transfer Learning*), es una heurística de Aprendizaje Automático que se enfoca en utilizar el aprendizaje adquirido en el

entrenamiento de un modelo de un sistema dado, y aplicarlo para entrenar otro modelo de un nuevo sistema que resuelve una problemática diferente al modelo original. Este mecanismo tiene el potencial de incrementar la eficiencia del nuevo sistema.

En 1976, Bozinovski y Fulgosi publicaron un artículo que aborda esta temática aplicada al entrenamiento de redes neuronales [73].

Es popular su uso con Aprendizaje Profundo en el que los modelos previamente entrenados se utilizan como punto de partida en la visión por computadora y las tareas de procesamiento del lenguaje natural, dada la gran cantidad de recursos informáticos y de tiempo necesarios para desarrollar modelos de redes neuronales sobre estos problemas y los enormes saltos en las habilidades que proporcionan sobre problemas relacionados.

En el aprendizaje por transferencia, primero entrenamos una red neuronal base en un conjunto de datos y una tarea base, y luego reutilizamos las características aprendidas, o las transferimos, a una segunda red neuronal objetivo, para ser entrenadas en un conjunto de datos y una tarea objetivo. Este proceso tenderá a funcionar si las características son generales, es decir, adecuadas para las tareas base y objetivo, en lugar de específicas para la tarea base.

3.9. Data Augmentation

El aumento de datos (*Data Augmentation*) es una heurística utilizada en el análisis de datos [74] para incrementar de manera significativa la diversidad de información disponible para entrenar modelos. Esta heurística ayuda a reducir los problemas de sobre entrenamiento de un modelo de Aprendizaje Automático [75], [76], dado que permite proporcionar mayor diversidad de imágenes de interés, por lo que el modelo tendrá mayor información para entrenar.

Cuando los datos son imágenes, se agregan copias de ellas con ligeras modificaciones utilizando las siguientes heurísticas de edición digital: Transformación geométrica, volteo, modificación de color, recortado, rotación, inyección de ruido, borrado aleatorio, entre otros.

3.10. Métricas utilizadas

Para validar y conocer mejor el alcance de los modelos entrenados, se utiliza generalmente una matriz de confusión que permite lograr la visualización de la clasificación llevada a cabo por estos modelos. De esta matriz, que se muestra en la Fig. 21, se obtienen las siguientes métricas: Exactitud (*Accuracy*), Precisión, Recall y F1 measure [77], cuyas formas de calcular se muestran a continuación:

$$Accuracy\ [A] = \frac{(TP + TN)}{(TP + TN + FP + FN)}$$

$$Precisión\ [P] = \frac{TP}{(TP + FP)}$$

$$Recall\ [R] = \frac{TP}{(TP + FN)}$$

$$F_{measure}[F1] = 2.\frac{P.R}{(P + R)}$$

Siendo:

$$TP = verdadero\ positivo$$

$$TN = verdadero\ negativo$$

$$FP = falso\ positivo$$

$$FN = falso\ negativo$$

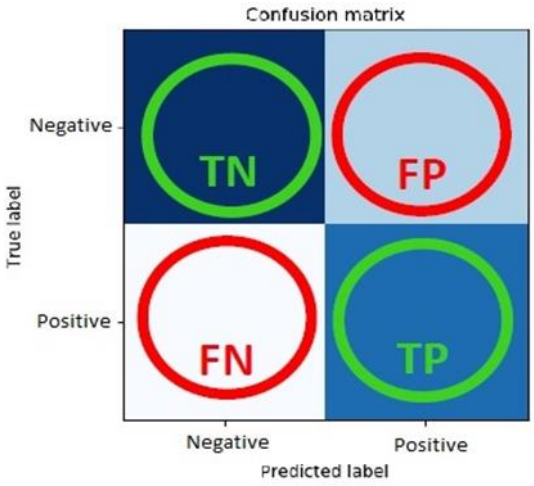


Fig. 21 Matriz de confusión

En general, se consideran como valores verdaderos positivos a la variable de interés, como por ejemplo, la presencia de un escorpión en la imagen de fluorescencia, o así mismo de la identificación de una especie peligrosa.

Otro parámetro que se considera a la hora de evaluar los modelos entrenados es la curva ROC (Receiver Operating Characteristic) [78], la cual representa gráficamente la sensibilidad frente a la especificidad y nos muestra cómo se comporta el modelo clasificador binario al cambiar el umbral de detección. Para dibujarla sólo son necesarios los TP y los FP. La recta TP=FP es equivalente al azar, cuanto mayor sea TP respecto a FP mejor será el modelo entrenado.

Cada resultado de predicción o instancia de la matriz de confusión representa un punto en el espacio ROC. El mejor método posible de predicción se situaría en un punto en la esquina superior izquierda, o coordenada (0,1) del espacio ROC, representando un 100% de sensibilidad (ningún falso negativo) y un 100% también de especificidad (ningún falso positivo).

Capítulo 4: Herramientas utilizadas para la implementación

4.1. Librerías

Durante el desarrollo de esta tesis se utilizaron diferentes librerías, de las cuales podemos mencionar dos referidas al procesamiento digital de imágenes y a la implementación de heurísticas de Aprendizaje Automático: OpenCV y TensorFlow, respectivamente.

OpenCV (Fig. 22) es una librería de código abierto, creada en 1999 por Intel, hoy en día ha llegado a ser la más popular en aplicaciones de visión por computadora. Entre sus aplicaciones posibles se encuentran la detección de movimiento y reconocimiento de objetos.

Fig. 22 Logo OpenCV

Originario en el lenguaje de programación en C, actualmente se encuentra disponible en múltiples lenguajes, el más importante por estos tiempos es Python, el cual proporciona otras librerías de código abierto que se complementan para el procesamiento digital de imágenes y el entrenamiento e implementación de redes neuronales para la comprensión y análisis de las mismas.

Al igual que con los lenguajes, OpenCV tiene soporte para varios sistemas operativos y varias arquitecturas de hardware, y al estar disponible el código fuente se podría compilar para que funcione en otros sistemas operativos aún no contemplados.

Fig. 23 Logo TensorFlow

Por su parte TensorFlow (Fig. 23), al igual que OpenCV, es una librería de código abierto. En este caso, se enfoca en el Aprendizaje Automático. Fue desarrollada por Google ante su necesidad de entrenar redes neuronales para detectar y descifrar

patrones y correlaciones. Aunque inicialmente fue creado por el grupo Google Brain para uso interno de la compañía, fue publicado como código abierto el 9 de noviembre de 2015.

TensorFlow cuenta con una amplia variedad de librerías para el entrenamiento de diferentes modelos de Aprendizaje Automático, Con una amplia comunidad de usuarios a nivel mundial y una gran versatilidad, sumado a la posibilidad de implementar los modelos en múltiples plataformas, la hace una excelente herramienta para el desarrollo e implementación de los modelos presentes en esta tesis. Esto sumado a ser de código abierto, lo hace ideal para realizar el desarrollo de una herramienta que ayude a las personas, sin fines de lucro, dentro del marco de una tesis doctoral de una universidad pública (*Pro Scientiae et Patria*)

Por otro lado, una gran desventaja de utilizar librerías de código abierto resulta ser las incompatibilidades que surgen en la implementación complementaria con otras librerías. Dicha desventaja, que dificulta o imposibilita la implementación de diferentes modelos con las mismas librerías, ha sido solventada con el uso de entornos de desarrollo virtual, los cuales nos permiten instalar diferentes versiones de librerías en entornos paralelos, los cuales utilizaremos para casos específicos de estudio, sin alterar el entorno para un caso en particular ya probado.

Fig. 24 Logo ImageAI

Un ejemplo de una librería utilizada en esta tesis es ImageAI (Fig. 24), la cual es de código abierto en Python para Visión por Computadora que simplifica el entrenamiento e implementación de modelos de inteligencia artificial aplicadas a imágenes [79]. Sus posibles aplicaciones son las siguientes:

- Reconocimiento de imágenes,
- Detección de Objetos,
- Clasificación de imágenes.

ImageAI proporciona clases y funciones muy potentes pero fáciles de usar para realizar la detección y seguimiento de objetos. Permite realizar todo esto con algoritmos de Aprendizaje Profundo como RetinaNet, YOLOv3 y TinyYOLOv3. Con ImageAI se puede ejecutar tareas de detección y analizar videos y transmisiones de video en vivo desde cámaras de dispositivos y cámaras IP.

4.2. Preparación de base de datos

En lo que respecta al entrenamiento de modelos de Aprendizaje Automático, uno de los primeros pasos y más largo de todos es la recolección y acondicionamiento de la base de datos a utilizar. En este caso en particular nuestros datos son imágenes de las cuales se extraen las características buscadas, ya sea para clasificar o detectar objetos.

Nuestra base de datos de imágenes se obtuvo de diferentes fuentes. El Laboratorio de Aracnología del CEPAVE nos proporcionó el acceso a las imágenes que recibe a diario por medio de la aplicación "¿Es Araña o Escorpión?" [80] que han desarrollado con fines de prevención e investigación de la presencia de escorpiones. Dichas imágenes suman 3.269 nuevas imágenes, de las cuales solo 362 son escorpiones. Igualmente, esta base de datos nos ha servido de mucho por la cantidad de imágenes negativas que nos proporcionó, además de que, dichas imágenes, en su mayoría, pertenecen a arácnidos, lo cual mejorará la capacidad del sistema de detectar correctamente a un escorpión y no confundirlo con otro arácnido. Asimismo, este laboratorio nos facilitó acceso su colección de escorpiones, los cuales fueron fotografiados sistemáticamente, esta base de datos, al poseer imágenes de mayor calidad (resolución, iluminación y enfoque) fueron ideales para entrenar los modelos de clasificación, sin dejar de lado imágenes con otras variantes para no generar un sesgo en la base de datos. Otras fuentes que aportaron menor cantidad de imágenes fueron una base de datos pre existente que poseía el CEPAVE y algunas imágenes de falsos positivos obtenidos con un sistema preliminar instalado durante el desarrollo de la tesis en el Ministerio de Infraestructura de la Provincia de Buenos Aires.

Dichas imágenes, dada la naturaleza variada de su origen, tuvieron que ser pre procesadas y seleccionadas acorde a la necesidad del modelo buscado. Nuestro pre procesamiento consistió simplemente en el ajuste de dimensiones según las necesidades del sistema a entrenar (generalmente el ajuste se da a 416x416, de no ser así se menciona oportunamente las dimensione correctas). Los procesamientos siguientes fueron varios, desde la demarcación de escorpiones en imágenes para entrenar la detección de los mismos como objetos, hasta la generación de imágenes con heurísticas de data augmentation para mejorar la calidad de las mismas.

Dado que esta tesis pretende proporcionar una herramienta que ayude a detectar y clasificar los escorpiones presentes a nivel local, las imágenes de escorpiones utilizadas para el entrenamiento fueron todas pertenecientes a especies presentes en la región, por lo que el uso de las aplicaciones desarrolladas solo será funcional en las regiones que cuenten con el mismo tipo de escorpiones. Sin embargo, los modelos pueden ser fácilmente escalables, con la base de datos adecuada, para añadir otras especies y por tanto extender el territorio donde puedan ser utilizados.

En el caso de la demarcación de escorpiones en las imágenes de la base de datos, se utilizó el software LabelImg (Fig. 25), el cual es un software desarrollado para demarcar objetos dentro de imágenes y asignarles sus respectivas etiquetas. Estas etiquetas son almacenadas en archivos xml, en formato "pascal VOC", para posteriormente ser utilizadas para entrenar un sistema de detección de objetos.

Además cuenta con otros formatos para ampliar su compatibilidad ante diferentes heurísticas de entrenamiento.

Fig. 25 Logo LabelImg

LabelImg es una herramienta gratuita y de código abierto, está escrita en Python y utiliza el framework QT para su interfaz gráfica. Para su instalación no hace falta más que ejecutar "pip3 install labelimg" en la consola, y luego ejecutarlo escribiendo "labelimg", o bien se puede descargar desde un repositorio de los desarrolladores [81].

Su uso es sencillo [82], hay que establecer el directorio donde se encuentran las imágenes a etiquetar y el directorio donde se almacenarán las anotaciones generadas. Una vez hecho esto, solo hay que utilizar los archivos generados.

Para el caso de data augmentation, se realizaron diferentes metodologías de aplicación, todas ellas efectivas, y, a fin de cuentas, basadas en las mismas heurísticas. Las técnicas utilizadas fueron las siguientes:

- Flip (Horizontal y vertical)
- Rotación de 90° (completando los 360°)
- Rotación de 45° (hacia derecha e izquierda)
- Saturación ±48%
- Exposición ±25%
- Desenfoque 1.75px
- Ruido 5% de los píxeles

Para entrenar los sistemas de detección de objeto se realizó data augmentation mediante código con las librerías de TensorFlow o, en uno de los casos, la propia librería de entrenamiento contaba con su propio sistema (por lo que solo requería las imágenes de la base de datos sin augmentation). Mientras que para los casos de entrenamiento para clasificación por peligrosidad de escorpión, se decidió utilizar Roboflow (Fig. 26) dada su practicidad y compatibilidad multiplataforma.

Fig. 26 Logo Roboflow

Roboflow es un servicio online que permite subir imágenes, etiquetarlas y exportarlas para crear un modelo de visión por computadora en base a la base de datos de interés de forma rápida. Este sistema ofrece un plan gratuito que permite subir un máximo de 1000 imágenes y exportar un máximo de 5000 imágenes, una vez superados estos límites se puede pagar para conseguir mayor capacidad.

Dado que nuestra base de datos (para clasificación) cumplía con esas restricciones, utilizar este servicio nos permitió ahorrar computo en nuestro equipo, dado que toda operación se realiza en la nube, y al mismo tiempo la exportación de la base de datos incrementada se puede realizar en diferentes formatos, permitiéndonos utilizarla para el entrenamiento de diferentes modelos, con diversas heurísticas de entrenamiento, sin la necesidad de realizar este proceso cada vez que quisiésemos probar otro modelo.

El sistema básicamente consta de poder subir imágenes, las cuales pueden estar etiquetadas previamente con LabelImg o de manera online, y se pueden establecer diferentes parámetros de data augmentation para generar hasta 3 imágenes por cada imagen real subida (en la versión gratuita). Una vez generadas las imágenes se puede seleccionar el formato con el que se la desea exportar a la nueva base de datos aumentada, se ofrece una amplia variedad de formatos adaptados a los diferentes mecanismos y propósitos de entrenamiento. Decidido lo anterior, se puede decidir si descargar las imágenes en cuestión o generar un enlace para poder utilizarla de manera remota, ya sea desde cualquier ordenador, como desde un entorno de desarrollo online.

Roboflow ofrece soporte de exportación tanto para detección de objetos como para clasificación, que es lo que se utilizará durante el desarrollo de esta tesis. Dentro de su página se puede acceder a múltiples tutoriales de uso del mismo con posibles aplicaciones [83].

4.3. Modelos

El siguiente paso, para obtener un modelo de Aprendizaje Automático funcional, es la elección del modelo en sí. Dicha elección se debe basar en las necesidades del proyecto. El objetivo de este trabajo justamente es analizar y comparar diferentes modelos con variadas heurísticas y complejidad para poder determinar cuál es el más idóneo para la tarea de detectar, identificar y clasificar escorpiones.

Durante el transcurso de esta tesis se utilizaron modelos con distintos grados de complejidad, desde los más sencillos que buscan patrones en histogramas hasta los más complejos que utilizan redes neuronales con capas de nivel profundo, entrando en el campo de aprendizaje homónimo.

Los modelos utilizados son cuatro, algunos de los cuales fueron implementados en más de una de sus versiones, LBPH, VGG16, MobileNet, y YOLO. Dichos modelos serán descriptos a continuación.

4.3.1. LBPH

Comúnmente utilizado para el reconocimiento facial, el algoritmo "Local Binary Pattern Histogram" (LBPH) es un sencillo y eficiente operador de texturas, combinado con histogramas de gradiente orientado, los cuales incrementan el rendimiento de algunas bases de datos [84].

Los parámetros que utiliza LBPH son cuatro, el "Radio" utilizado para construir el patrón binario local, y es justamente el radio medido desde el pixel central; el parámetro "Vecinos", que es la cantidad de píxeles que se utilizan para construir el patrón; "X" que es el número de celdas en dirección horizontal; y "Y" para la dirección vertical.

Para realizar el reconocimiento, lo primero que realiza este algoritmo es crear una imagen intermedia que represente a la original de mejor forma, resaltado las características de interés en la misma. Para ello, el algoritmo utiliza el concepto de ventana deslizante, basado en los parámetros "Radio" y "Vecinos".

Una vez obtenida esta imagen binaria con las principales características de la imagen, se utilizan los parámetros "X" e "Y" para dividir la imagen en una cuadrícula. De cada cuadrado de esta cuadrícula se obtiene su respectivo histograma. Luego se concatena cada histograma para obtener uno que represente a toda la imagen. Comparando estos histogramas es cómo el algoritmo logra identificar al objeto en cuestión.

4.3.2. VGG16

Es un modelo de red neuronal convolucional o CNN, por sus siglas en inglés, propuesto por K. Simonyan y A. Zisserman de la Universidad de Oxford [85]. El modelo alcanza una precisión de prueba del 92,7% entre los 5 primeros en ImageNet, que es un conjunto de datos de más de 14 millones de imágenes que pertenecen a 1000 clases.

Realiza la mejora sobre AlexNet al reemplazar filtros de gran tamaño de kernel (11 y 5 en la primera y segunda capa convolucional, respectivamente) con múltiples filtros de tamaño de kernel de 3 × 3 uno tras otro. VGG16 se entrenó durante semanas y utilizaba GPU NVIDIA Titan Black.

En la Fig. 27 se muestra la arquitectura de VGG 16, en el cual se pueden observar cinco bloques convolucionales. Dicha imagen fue obtenida de un blog sobre Transfer Learning [86].

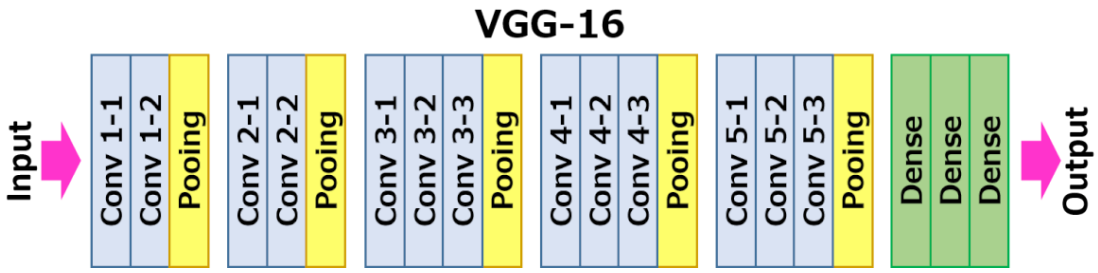


Fig. 27 Capas de VGG 16

4.3.3. Mobile Nets

Es un modelo para su uso en visión por computadora con TensorFlow, optimizado para su implementación en dispositivos móviles [87], [88], puede ser utilizado para clasificación, detección, incrustaciones y segmentación de manera similar a como se usan otros modelos populares a gran escala, como es el caso de Inception que se explicará más adelante.

Diseñado para maximizar la precisión de manera efectiva mientras se tienen en cuenta los recursos restringidos para una aplicación integrada o en el dispositivo. Los MobileNets son modelos pequeños, de baja latencia y bajo consumo de energía parametrizados para satisfacer las limitaciones de recursos de una variedad de casos de uso (Fig. 30).

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Fig. 28 Tabla de capas de arquitectura MobileNet

4.3.4. You Only Look Once (YOLO)

YOLO (Fig. 29) es un sistema de código abierto para la detección de objetos en tiempo real [89]–[91]. Su base de funcionamiento es una red neuronal convolucional para detectar objetos en imágenes. Para ello divide la imagen en regiones prediciendo identificación y probabilidad por cada una. Usa características de toda la imagen para predecir cada cuadro delimitador. También predice todos los cuadros delimitadores en todas las clases para una imagen simultáneamente. Lo que se busca tener es un solo cuadro de identificación por objeto, lo cual se consigue a partir de las probabilidades predichas para cada cuadro, manteniendo el de mayor alta probabilidad.

Fig. 29 Logo YOLO

El modelo se basa en una red neuronal convolucional y fue evaluado en el set de datos para detección de PASCAL VOC. Su funcionamiento se divide en capas, las iniciales de la red se encargan de la extracción de características de la imagen, mientras que las capas de conexión completa predicen la probabilidad de salida y las coordenadas del objeto.

4.4. Entrenamiento

Con el fin de realizar el entrenamiento de los modelos, se recurrió a utilizar un entorno de desarrollo, el cual varió según las necesidades del momento. En una primera instancia la tesis comenzó a desarrollarse en leguaje de programación C, utilizándose el entorno de desarrollo integrado Code::Blocks (Fig. 30). Este entorno de desarrollo es de código abierto, soporta múltiples compiladores y está orientado a C, C++, y Fortran.

Fig. 30 Logo Code::Blocks

Sin embargo, una vez que se avanzó en complejidad dentro de la tesis se llegó a la conclusión de que el lenguaje C tiene importantes limitaciones en lo que respecta al procesamiento digital de imágenes utilizando inteligencia artificial. Es por ello, que luego de realizar un relevamiento de proyectos actuales en la temática, se decidió continuar el desarrollo de este trabajo en Python, teniendo que traducir el código, hasta ese momento realizado en C, al nuevo lenguaje.

Para editar y compilar (*offline*) nuestro código en Python, se utilizó el entorno de desarrollo científico en Python, o por sus siglas en inglés Spyder (Fig. 31), el cual está compuesto por un editor, una consola IPython, un explorador de variables y figuras, un depurador y una consola de ayuda. Perteneciente al grupo de software Anaconda, es un entorno de desarrollo integrado (IDE) gratuito y de código abierto diseñado por científicos, ingenieros y analistas de datos.

Fig. 31 Logo Spyder

Inicialmente creado y desarrollado por Pierre Raybaut en 2009, desde 2012 Spyder ha sido mantenido y mejorado continuamente por un equipo de desarrolladores científicos de Python y la comunidad [92].

Anaconda es una distribución, que al igual que Spyder es libre y abierta, que se caracteriza por trabajar en un entorno de desarrollo independiente al que utiliza el sistema operativo, es decir que los cambios en el entorno solo afectan a Anaconda y sus programas y no a la consola de comandos del sistema operativo que se utiliza.

Los paquetes que se instalan en el entorno lo hacen a partir del sistema de gestión de paquetes Conda, el cual hace sencillo la instalación y actualización de librerías, olvidando el problema de verificar la intercompatibilidad de las librerías entre sí. Si una librería es instalada de forma correcta por el comando Conda significa que no habrá conflictos entre nuestros paquetes instalados, sin embargo, esto no descarta los problemas de incompatibilidad de nuestro código con nuevas librerías instaladas, lo cual ha ocasionado varios inconvenientes durante el desarrollo de esta tesis.

Más avanzados en el desarrollo de esta tesis se comenzó a requerir mayor capacidad de cómputo para procesamientos que podían llegar a tener una duración de días con nuestros equipos totalmente dedicados a dichos cálculos. Ante esta necesidad, se recurrió a Colaboratory (Colab) (Fig. 32), el cual es un entorno gratuito que provee Google para escribir y ejecutar código de Python desde un navegador [93], [94]. Se caracteriza por no requerir configuración previa para su uso, lo que es muy útil teniendo en cuenta que, la creación de entornos de desarrollo diferentes para cada proyecto, de

diferentes características, que querramos ejecutar consume mucho tiempo. Además, provee de acceso gratuito a GPU y facilidad para compartir ya que puede ser enlazado al Google Drive.

Fig. 32 Logo Colaboratory

Los notebooks de Colab permiten combinar código ejecutable y texto enriquecido en un único documento, junto con imágenes, HTML, LaTeX, entre otros. Colab proporciona ejecución en Python 2 y 3, ambos preconfigurados con las bibliotecas esenciales para Aprendizaje Automático e Inteligencia Artificial, como son TensorFlow, Matplotlib y Keras [95].

El entorno de desarrollo tiene un tiempo máximo de 12 horas de ejecución continua, una vez finalizado este periodo todos los datos del entorno de ejecución se pierden y se debe iniciar uno nuevo. Los únicos datos que no se pierden son los que quedan en el notebook, es por ello que, si se está entrenando una red neuronal que vaya a tardar más que este período máximo se debe fraccionar el entrenamiento e ir guardando la información generada, o se puede establecer permisos de almacenamiento directamente en nuestro Google Drive, evitando tener que preocuparnos por la finalización automática de la sesión. Desde luego que existe una versión de acceso premium que extiende este plazo a no más de 24 horas y proporciona GPUs más potentes, lo que reduce el tiempo de cómputo para sistemas paralelizables, pero lamentablemente el acceso a este tipo de entorno está limitado a Estados Unidos y Canadá.

Finalmente, podemos mencionar a Teachable Machine, la cual es una interfaz gráfica de usuario (GUI) basada en la web para crear clasificadores de Aprendizaje Automático personalizados [96]. Este es proporcionado de forma gratuita por Google, y además de proporcionar el entrenamiento del modelo permite exportarlo en múltiples formatos, en nuestro caso TensorFlow cuantificado.

Por otra parte, proporciona algunas instrucciones para poder realizar una implementación exitosa del modelo en el sistema que se lo quiera utilizar.

El sistema de entrenamiento utiliza el 85% de las muestras que le proporcionamos para entrenar y el otro 15% para testear, proporcionándonos el análisis de efectividad del sistema con las métricas y los gráficos correspondientes, tanto de entrenamiento como de testeo.

Esta última herramienta, nos fue de gran utilidad para realizar el entrenamiento con un modelo en particular al cual se lo comparó con modelos que entrenamos con los entornos de desarrollo previamente mencionados.

4.5. Implementación de aplicaciones móviles

Una vez entrenados nuestros modelos, procedimos a implementarlos en aplicaciones de teléfonos móviles. Para ello se recurrió a Android Studio (Fig. 33), el cual es un entorno de desarrollo integrado para la plataforma Android, la cual reemplazó a Eclipse como IDE oficial de este sistema operativo. Aunque su primera versión estable fue lanzada a fines de 2014, sus primeras versiones se encuentran disponibles desde principios de 2013.

Fig. 33 Logo Android Studio

Este IDE permite programar con los lenguajes JAVA, C++, y el preferido de Google, Kotlin. Cuenta con la capacidad de simular cualquier dispositivo, con cualquier versión de Android, para emular el funcionamiento de las aplicaciones desarrolladas.

4.6. Componentes para el sistema analógico de detección

Para el sistema analógico de detección desarrollado, el sensor de color utilizado fue el TSL-257 (Fig. 34) creado por la empresa estadounidense TAOS (Texas Advanced Optoelectronic Solution). A este sensor se lo describe como un convertidor de intensidad lumínica a voltaje de alta sensibilidad y bajo ruido que combina un fotodiodo y un amplificador de transimpedancia en un mismo circuito integrado monolítico CMOS (Fig. 35).

Fig. 34 Sensor TSL-257

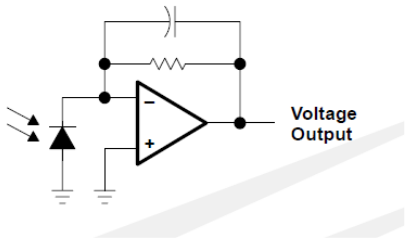


Fig. 35 Esquemático de Sensor TSL-257

Su tensión de alimentación va desde los 2,2 a los 5,5 V con una corriente máxima de 3,5 mA, lo que permite que sea alimentado por la misma fuente que alimenta al microcontrolador que analizará su señal de salida.

Se seleccionó este sensor debido a que presenta una respuesta diferente para diferentes longitudes de onda (Fig. 36) lo que permite la diferenciación de los colores ante una fuente de iluminación estable.

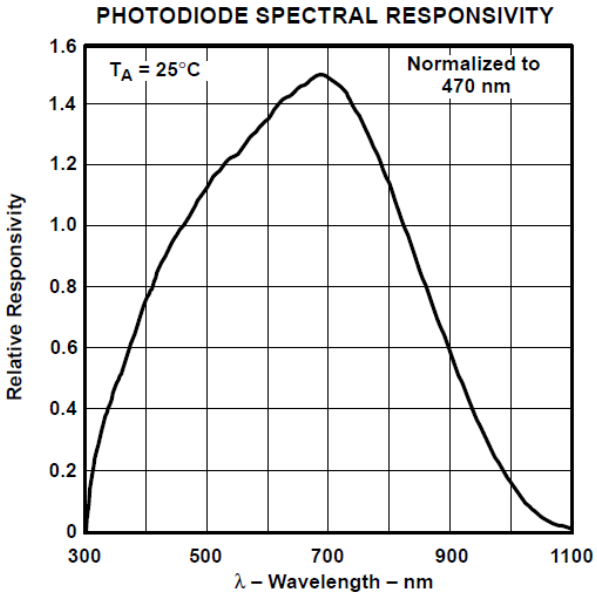


Fig. 36 Respuesta de TSL-257 a diferentes longitudes de onda

Este sensor puede ser considerado como un sensor mono pixel que promedia toda luz que recibe (su respuesta depende del ángulo de incidencia con la que recibe dicha luz (Fig. 37), al igual que lo hace la distancia del objeto que emite la longitud de onda buscada.

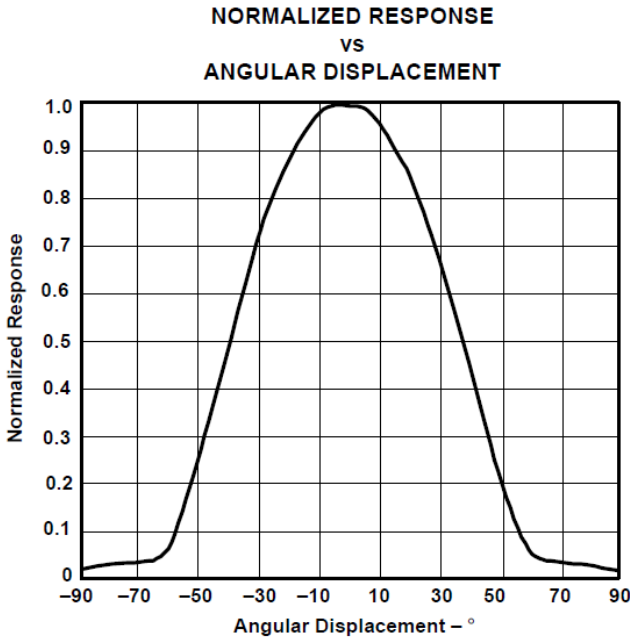


Fig. 37 Respuesta de sensor TSL-257 a ángulo de incidencia de luz

El sensor de distancia utilizado es el HC-SR04 (Fig. 38) y su funcionamiento se base en el envío de un pulso de alta frecuencia, no audible por el ser humano. Este pulso rebota en los objetos cercanos y es reflejado hacia el sensor. La distancia se calcula midiendo el tiempo entre pulsos, conociendo la velocidad del sonido, podemos estimar la distancia del objeto contra cuya superficie impacto el impulso de ultrasonidos. El rango de medición teórico del sensor HC-SR04 es de 2 a 400 cm, con una resolución de 0,3 cm.

Fig. 38 Sensor HC-SR04

Capítulo 5: Desarrollo de la Tesis

5.1. Descripción

En el desarrollo de esta tesis, como se ha mencionado, se realizaron diferentes sistemas que complementados tienen un objetivo común: detectar y clasificar a los escorpiones por su peligrosidad.

Esta sección se dividirá en dos categorías, detección de objetos, encargada de encontrar al escorpión en una imagen; y la clasificación, que los diferencia por género y en un caso por especie, pero más importante informa si es, o no, peligroso para una persona.

El mecanismo de funcionamiento del sistema se puede observar en el diagrama de flujo de la Fig. 39, el cual muestra los sistemas de detección y clasificación, los cuales, aunque pueden ser implementados de forma independiente, como los ensayos realizados, se pueden fusionar en un solo sistema que se encargue de detectar y clasificar al escorpión.

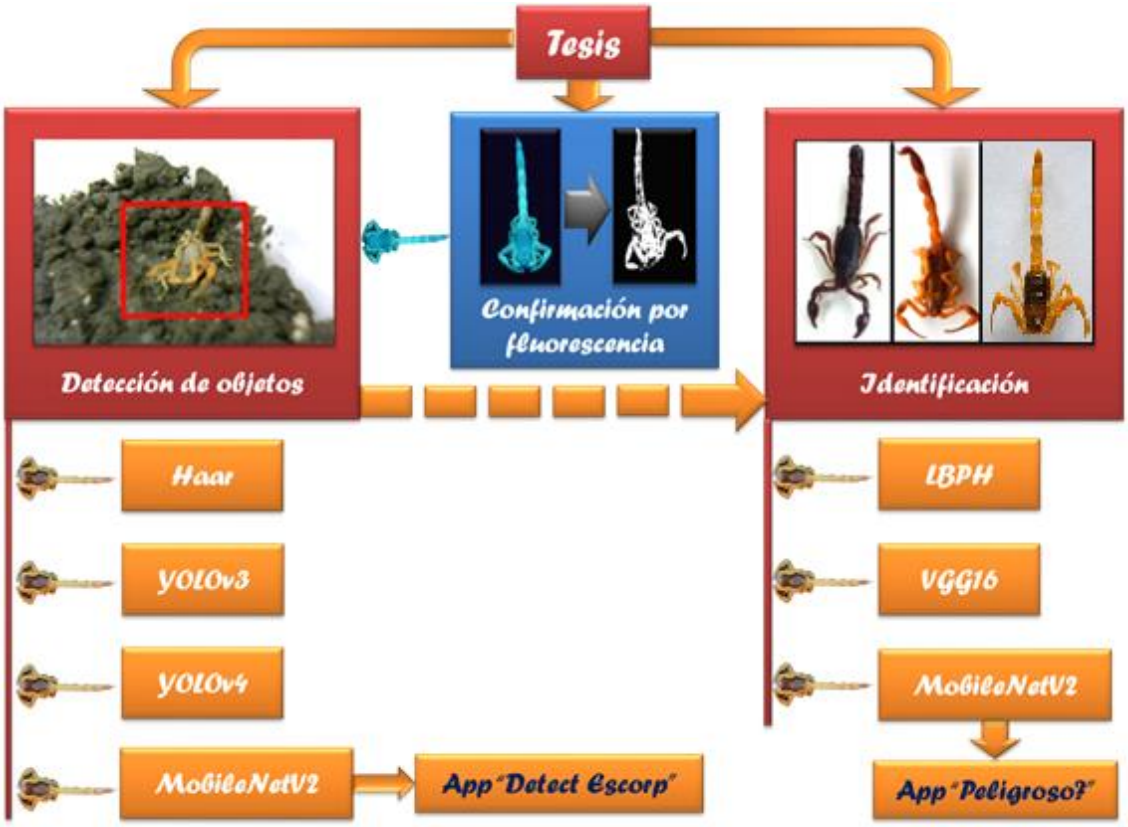


Fig. 39 Diagrama de flujo de Tesis

Como se puede apreciar en el diagrama, la tesis consta del desarrollo de dos grupos principales: Detección del objeto escorpión e identificación de los mismos (sea para su clasificación por peligrosidad o para determinar su especie). El uso de estos dos sistemas puede ser complementario, como se muestra con la flecha punteada que los une, por lo que es un sistema completamente integrado que puede utilizar la detección y luego la identificación de lo detectado.

Dentro de lo que respecta a la detección del objeto escorpión, se pueden apreciar, cuatro métodos implementados: el de detección por fluorescencia, el clasificador en cascada (Haar), YOLO (v3 y v4), y MobileNetV2, el cual posibilitó la implementación de una aplicación (App) de celular. Con respecto a la detección mediante fluorescencia, se han planteado dos posibilidades para su implementación, una es la detección directa (en caso de un lugar que siempre esté iluminado con UV y en ausencia de luz natural) y la otra es su utilización para la confirmación de la detección de alguno de los otros métodos de detección usando Machine Learning o Deep Learning.

Por otro lado. se encuentran los métodos empleados para la clasificación de los escorpiones, los cuales presentan tres diferentes modelos, el sistema con LBPH, el sistema con VGG16, y el sistema con MobileNetV2 que nos proporcionó la posibilidad de crear la App “Peligroso?”.

5.2. Sistemas de detección de escorpiones

En esta sección se presentará cada uno de los sistemas de detección del objeto escorpión desarrollados, utilizando las diferentes metodologías mencionadas previamente: clasificador en cascada (Haar), YOLO, detección por fluorescencia y MobileNetV2.

5.2.1. Clasificador en cascada

5.2.1.1. Adquisición de imágenes para generar base de datos.

Con el objetivo de conseguir la cantidad suficiente de imágenes se requirió realizar un software que realice la captura de las mismas mediante los cuadros (*frames*) tomados con una webcam. Para ello se recurrió a las librerías de OpenCV, más precisamente a las funciones cvCaptureFromCAM() y cvSaveImage(), para ir almacenando las imágenes con un nombre numerado. Al programa se le puede configurar la cantidad de imágenes que va a capturar y se lo puede detener en cualquier momento. Además, se puede modificar el tiempo de captura entre imágenes, lo que permite obtener imágenes con mayor cantidad de variaciones entre sí, esto dependerá del nivel de actividad del escorpión o del objeto que querramos detectar.

5.2.1.2. Comparación entre versiones de haartraining de OpenCV

La librería de OpenCV proporciona programas que se utilizan para entrenar clasificadores para sistemas de detección de rostros, llamado HaarTraining, de manera que podemos crear nuestros propios clasificadores de objetos utilizando este programa. Sin embargo, su utilización no nos permite tener conocimiento específico sobre su funcionamiento. Esto último no representa un problema dado que el objetivo de esta tesis es la implementación de un sistema físico que posibilite la detección de escorpiones, no el estudio exhaustivo del funcionamiento de una librería.

Durante el desarrollo de OpenCV se han producidos muchos cambios en las diferentes versiones, una de las cuales, para nuestro pesar, es el formato de generación de clasificadores para la detección de objetos. Básicamente, se están migrando las librerías nuevas de OpenCV del leguaje C al C++, siendo diferente el formato que manejan las funciones que utilizan los clasificadores generados por dichos programas, aunque las extensiones de los mismos son iguales (.xml). Un ejemplo de ello se puede apreciar en la Fig. 40 obtenida del procesamiento de la misma base de datos de imágenes con dos diferentes versiones del HaarTraining.

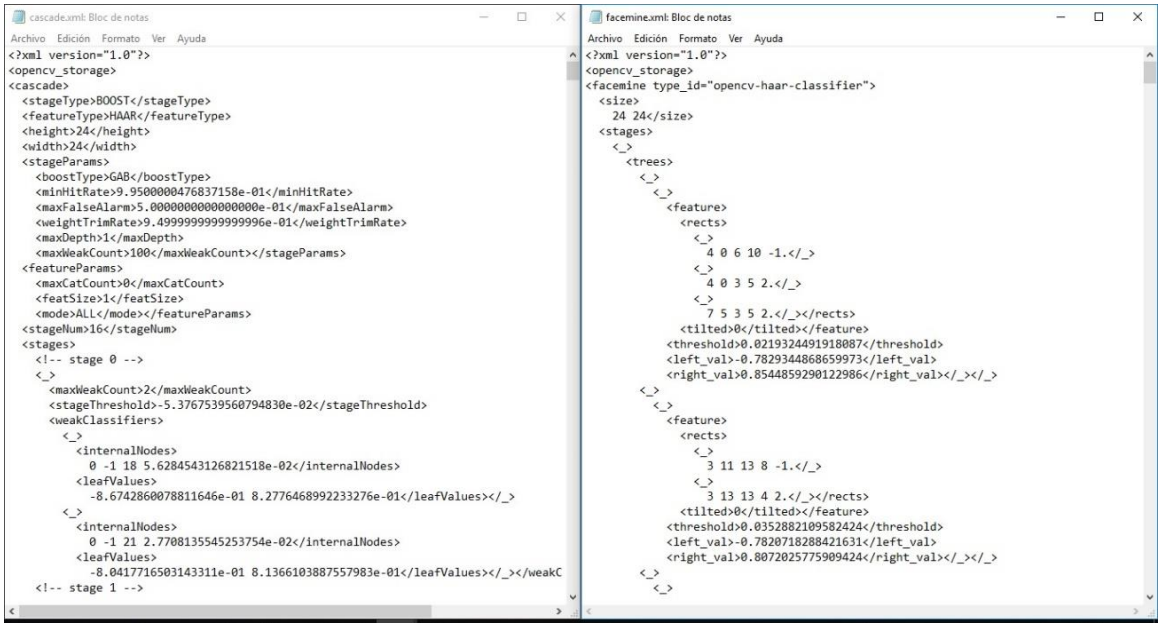


Fig. 40 Comparación de versiones de archivos XML

El repositorio oficial de OpenCV ya no cuenta con la versión original del HaarTraining, sin embargo, puede conseguirse mediante el repositorio GitHub [97]. Dicho enlace cuenta no solo con los programas necesarios para crear el clasificador, sino que, además, cuenta con un instructivo de uso que ha sido de gran utilidad.

5.2.1.3. Desarrollo de sistemas ejemplo con la versión haartraining compatible para las librerías de OpenCV en C

El primer y más utilizado ejemplo de detección de objetos es la detección de rostros. En primera instancia se utilizó una base de datos con imágenes de rostros de hombres y mujeres, lo que se realizó entonces, a modo de prueba del sistema, fue la implementación de un sistema que pudiese discriminar un rostro masculino de uno femenino, o viceversa.

Para poder utilizar el programa mencionado, se deben realizar una serie de pasos obligatorios para el correcto funcionamiento del mismo. Una vez que tenemos una base de datos, con imágenes con el objeto (positivas) y en ausencia del mismo (negativas), realizamos los siguientes pasos:

Lo primero que se debe realizar es una lista, en archivo de texto, con el nombre de todas las imágenes positivas por un lado y negativas por otro. Para no tener que escribir todos estos nombres manualmente se creó un programa cuya única función sea listar todos los elementos que se encuentran en su directorio. De esta forma, colocando imágenes positivas y negativas en carpetas diferentes, se crea un listado para cada tipo de imagen. A continuación, se muestra el archivo para realizar el listado, el cual se guarda en formato ".bat".

```
dir /a /b /-p /o:gen >C:\Users\Administrador\Desktop\nombre.info
start notepad C:\Users\Administrador\Desktop\nombre.info
```

Como se puede observar el listado lo almacena en el escritorio a nombre de nombre.info. Luego este archivo debe ser copiado al directorio donde se encuentran las imágenes que lista para funcionar correctamente. Esto se puede modificar para los diferentes casos, además de diferentes computadoras con nombres de usuario diferentes.

Para el caso de las imágenes positivas, se puede realizar este listado de otra forma que consiste en la demarcación de la zona de interés en las mismas. Dicha información se almacena en el mismo archivo de listado de imágenes, en la misma línea que la dirección del archivo. Cada una de las direcciones cuenta con la ubicación, en píxeles, del extremo superior izquierdo e inferior derecho del rectángulo que encierra la zona de interés.

Para el desarrollo de este segundo paso, se cuenta con otro programa llamado objectmarket.exe, el cual tiene una serie de requisitos para su funcionamiento, debido a que no se pueden agregar comandos para su utilización, cómo la dirección de los archivos. El programa debe ubicarse en una carpeta raíz junto con dos archivos (cv.dll y highgui.dll); las imágenes deben guardarse en una subcarpeta llamada "rawdata" en

formato “.bmp”. Una vez ejecutado el programa se debe seleccionar, manteniendo apretado el botón izquierdo del mouse, la zona de interés, presionar “Space” para guardar dato y “Enter” para pasar a siguiente imagen. Si se desea continuar en otro momento se presiona “Escape” para salir. Para no perder lo ya realizado se debe hacer una copia del archivo “info.txt” que genera el programa con la información que luego utilizaremos en nuestro “Haar-Training”.

Una situación que puede ocurrir es que tengamos nuestras imágenes en otro formato que no sea “.bmp” lo cual imposibilita el uso del programa antes mencionado. Para solucionar este inconveniente, sin recurrir al cambio manual de todas las extensiones de cada archivo, se puede crear otro archivo “.bat” que realice este procedimiento.

```
ren *.formato-entrada *.formato-salida
```

El tercer paso que se debe seguir es la creación del archivo vector basado en las imágenes positivas. Para ello se requiere de la utilización del programa createsamples.exe de las librerías OpenCV. Dicho programa utiliza el listado de imágenes positivas para crear el archivo vector de muestras que se requiere para la implementación del clasificador en cascada. Para poder utilizarlo se necesita utilizar el “Símbolo del sistema” para poder introducir los comandos que deben acompañar a la ejecución de dicho programa, los cuales se pueden apreciar en la Tabla 2.

Comando	**Valor**	**Descripción**
-infor	info.txt	Dirección de archivo con información de imágenes positivas.
-vec	vector.vec	Dirección de archivo vector de salida del programa.
-num	2000	Número de cantidad de imágenes positivas.
-w	20	Ancho de los objetos
-h	20	Largo de los objetos

Tabla 2 Comandos de ejecución de "createsamples.exe"

Para que funcione el programa debe contar en la misma carpeta con los siguientes archivos: cv097.dll, cxcore097.dll, highgui097.dll, y libguide40.dll.

El cuarto paso consiste en realizar el Haar-Training, para ello se utiliza el programa homónimo “haartraining.exe” el cual requiere del archivo vector, generado en el paso anterior, y el listado de imágenes negativas generada en el primer paso. Al igual que en

el caso anterior, debemos recurrir al “Símbolo del sistema” para la ejecución con los parámetros mostrados en Tabla 3.

Comando	Valor	Descripción
-data	cascades	Dirección de almacenamiento de las etapas del clasificador
-vec	vector.vec	Dirección de archivo vector de imágenes positivas
-bg	negativos.txt	Dirección de listado de imágenes negativas
-npos	2000	Número de imágenes positivas
-nneg	2000	Número de imágenes negativas
-nstages	15	Número de etapas de entrenamiento
-mem	1024	Cantidad de memoria asignada en MB
-mode	ALL	Característica del tipo de entrenamiento (conjunto completo de funciones verticales y rotadas 45 grados)
-w y -h	20	Dimensiones (deben coincidir con las utilizadas en createsamples.exe)
-nonsym		Se usa en caso de que no haya simetría horizontal en el objeto a detectar

Tabla 3 Comandos de ejecución "haartraining.exe"

Una vez terminado el entrenamiento realizado con el último programa, se debe crear un archivo que albergue toda esta información para poder utilizarla en la detección, el paso cinco. Dicho archivo es de extensión “.xml” y contiene toda la información que el programa anterior depositó en la subcarpeta “cascades”. Para realizar esto, se debe crear una carpeta principal que contenga el programa “haarconv.exe”, y que contenga la carpeta cascade dentro de ésta. El comando es el siguiente:

Haarconv cascade nombre-clasificador.xml 20 20

donde se coloca el nombre de la carpeta de las etapas del clasificador, el nombre del archivo “.xml” que se va a generar, y dimensiones del objeto (iguales al createsamples y el haartraining).

Finalmente se realizó el entrenamiento del clasificador utilizando imágenes de un mismo rostro para que pueda diferenciarlo del entorno. Para ello se consiguieron 3369 fotos de dicho rostro, capturadas mediante la webcam de la computadora durante un día de trabajo, y como imágenes negativas para el entrenamiento se utilizaron 3135 fotos de rostros diversos, obtenidos de una base de datos de acceso público en internet.

Para dicho entrenamiento se utilizaron los siguientes parámetros:

-nstages 15 -mem 1024 -mode ALL -w 20 -h 20 -nonsym -nsplits 5

El resultado fue una detección positiva del rostro requerido (Fig. 41) y el rechazo de detección de otros.

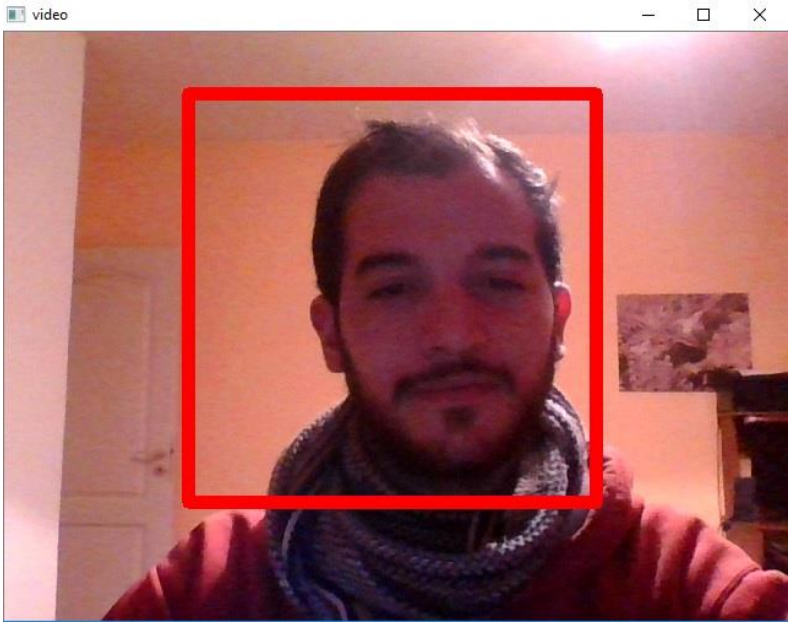


Fig. 41 Detección de mi rostro

5.2.1.4. Creación de clasificador en cascada para escorpiones.

Una vez probado con éxito la detección de rostros, se procedió a repetir el análisis para la detección de escorpiones. Para dicha implementación se contó con la colaboración del CEPAVE, el cual proporcionó el acceso, bajo supervisión, a un escorpión (*Tityus confluens*). Este escorpión fue utilizado para obtener imágenes del mismo en diferentes posiciones y actitudes. Se realizaron varios ensayos y entrenamientos hasta encontrar un sistema que funcionara de manera acorde.

En una primera instancia se utilizaron 1.054 fotos del escorpión mencionado y 502 imágenes sin escorpión. Este primer entrenamiento de cascada se realizó con una supervisión relativa debido a que la única información que recibió fue la discriminación de imágenes positivas y negativas, pero no la ubicación del escorpión dentro de la imagen positiva. Este clasificador no llegó a ser probado con otros escorpiones ya que demostraba un mal funcionamiento en ausencia de los mismos, demarcando múltiples objetos en una oficina como si fueran escorpiones.

Analizando detalladamente las imágenes que se habían utilizado para el clasificador se llegó a la conclusión de que estas imágenes no poseían la calidad necesaria para tener una imagen nítida de los escorpiones, además de ser una cantidad insuficiente. Es por ello, que se realizó una nueva captura de imágenes las cuales fueron seleccionadas con un criterio más estricto para el correcto entrenamiento. Además, en este caso se cuenta con la función (objectmarker), la cual nos permite seleccionar dentro de las imágenes el lugar donde se encuentra el escorpión permitiendo un entrenamiento más supervisado.

Para este segundo ensayo se utilizaron 1.300 imágenes de escorpión y 502 imágenes negativas (sin escorpión). Se realizaron distintas generaciones de

clasificadores con diferentes cantidades de etapas de iteración, siendo a partir de treinta la cantidad ideal para poder realizar una detección acorde. El resultado obtenido con este clasificador fue muy satisfactorio tanto para la detección del escorpión como la ausencia del mismo. A continuación, se implementó el clasificador en el Laboratorio de Aracnología del CEPAVE con un escorpión en diversos ambientes, obteniendo así una detección del mismo (Fig. 42), con aislados casos de falsa detección.

Fig. 42 Primer detección de escorpión con Haar

Luego de obtenido este clasificador se trabajó para mejorarlo, para ello se realizó la incorporación de nuevas imágenes ajenas a nuestro sistema de captura, proporcionadas por personal del laboratorio, las cuales fueron capturadas mediante el uso de una lupa. Dichas imágenes poseían un excesivo nivel de resolución, que fue reducido con fines de no realizar procesamiento demás, y contaban con mucha nitidez, lo que las hace ideales candidatas para la generación de este clasificador.

Una vez incorporadas estas imágenes al entrenamiento del clasificador en cascada, se procedió a ensayarlo en presencia del escorpión. Este nuevo sistema demostró un correcto funcionamiento (Fig. 43), sobre todo al no presentar falsas detecciones, sin embargo, el sistema carece de la sensibilidad esperada para realizar la detección, debido a que el escorpión es detectado con cierta dificultad.

Fig. 43 Detección con Haar

A su vez, este sistema presenta un problema muy importante, dado que también ha detectado otros arácnidos. Este problema se debe, principalmente, a que la base de datos de imágenes negativas que se utilizó originalmente para este clasificador en cascada eran imágenes de escenarios sin ningún animal en ellas. Para solventar dicha falencia se procedió a realizar la incorporación de imágenes de otros animales, principalmente arácnidos, los cuales pueden presentar mayor similitud con un escorpión y es fundamental diferenciarlos.

La implementación del entrenamiento de este sistema se realizó mediante la utilización de los ejecutables "objectmarker.exe", "createsamples.exe", y "haartraining.exe". Estos archivos se encuentran dentro del directorio de instalación (Disco:\opencv\apps), cabe aclarar que, en algunos casos, solo se encuentran los códigos de los mismos, por lo que se requiere de su compilación para la utilización.

La organización básicamente consiste en crear directorios para cada variable de entrada que requieren los programas en sí. Una carpeta para las imágenes positivas, y otra para las negativas, las cuales deberán contener una carpeta interna con las imágenes y un archivo de texto con el direccionamiento de todas las imágenes dentro de la carpeta junto con la cantidad y coordenadas de ubicación de los escorpiones. Habrá otra carpeta, que en este caso llamaremos "vector" la cual tendrá el resultado de ejecutar "createsamples.exe" a partir de las imágenes positivas. Finalmente se encuentra la carpeta que contendrá el resultado de cada capa de entrenamiento.

Toda la información necesaria para el entrenamiento se encuentra en la Tabla 4, la cual muestra los parámetros para establecer las direcciones de los archivos y los valores de atributos necesarios.

Nombre	Función
-data	Almacenamiento de entrenamiento
-vec	Archivo vector creado de positivos
-bg	Archivo direccionamiento negativos
-npos	Cantidad de positivos
-nneg	Cantidad de negativos
-nstages	Cantidad de etapas de entrenamiento
-mem	Asignación de memoria
-w / –h	Dimensiones de plantilla entrenamiento (24x24)
-bt LB	Tipo de clasificador (AdaBoost)

Tabla 4 Descripción de comandos para entrenamiento de Haar

Desde luego estos no son los únicos parámetros con los que se puede modificar el entrenamiento, hay más opciones que mantienen valores por defecto mientras no se los mencione [98]. A fines de la necesidad de este proyecto las mencionadas son suficientes para una detección acorde.

Una vez finalizado el entrenamiento, que en nuestro caso requirió 13 etapas para no llegar al sobre-entrenamiento, se debió proceder a la confección del archivo final que será utilizado para la detección. Al igual que en los pasos anteriores, OpenCV nos proporciona el programa que realiza esta tarea “haarconv.exe” generando nuestro archivo que tendrá extensión “xml”.

La implementación de la detección (Fig. 44) se realizó mediante la incorporación de las librerías para dicho propósito de OpenCV (“cv2” en el caso de Python) [99]. Entre las funciones utilizadas se destaca “CascadeClassifier()” que define un tipo de elemento que permite cargar el archivo seleccionado y realizar la detección en una imagen con su método “.detectMultiScale()”.

Fig. 44 Detección tiempo real con Haar

5.2.2. YOLOv3

Otro método utilizado en esta tesis para la detección de escorpiones ha sido el YOLOv3. Este sistema, cada vez más utilizado por su velocidad y eficiencia frente a otros sistemas similares, utiliza las librerías Keras de Tensorflow [100], [101] para su funcionamiento.

Pese a haber utilizado la misma base de datos con las imágenes positivas del clasificador en cascada, no fue posible utilizar el archivo de direccionamiento y demarcación de las imágenes con escorpiones. Esto se debe a que, como se dijo previamente, el algoritmo de entrenamiento utiliza otro formato de archivo para poder entrenarse. Es por ello que se recurrió al software "LabelImg" para poder generar estos archivos de la manera más rápida y práctica posible [81].

Lo más importante a destacar dentro del direccionamiento es que las imágenes se ubican dentro de un directorio "train/images" mientras que las anotaciones referentes a la ubicación de las imágenes, junto a la de los escorpiones dentro de ella, se encuentran en "train/annotations".

Para el entrenamiento con esta librería, se debe crear un objeto con la función "DetectionModelTrainer()" y luego ir configurándolo con sus métodos, de los cuales cabe destacar "setTrainConfig()" que contiene las variables más relevantes (Tabla 5).

Nombre	Función
object_names_array	Lista de objetos
batch_size	Cantidad de imágenes de trabajo por iteración
num_experiments	Cantidad de etapas de entrenamiento
train_from_pretrained_model	Archivo pre-entrenado si lo hubiese

Tabla 5 Variables del método "setTrainConfig()"

En una primera instancia, se intentó entrenar este sistema de detección mediante un entorno de desarrollo local, utilizando el IDE Spyder, lo cual presentó una serie de inconvenientes que vale la pena mencionar. El primero de ellos fue la incompatibilidad de versiones de las diferentes librerías empleadas, debido a que de acuerdo con la documentación de ImageAI, se debía instalar la versión 1.13.1 de TensorFlow, la cual actualmente está obsoleta, pese a que TensorFlow sigue proporcionando acceso a esta versión. Esto ocasionó múltiples inconvenientes a la hora de compilar, por lo que se instaló la versión más reciente de TensorFlow a la fecha del ensayo (2.2.0) y se volvió a probar el entrenamiento, volviendo a surgir incompatibilidades, esta vez dentro de la librería ImageAI.

Lamentablemente esto se tuvo que solucionar a prueba y error, probando todas las versiones que pudiesen satisfacer a todas las partes del código. Finalmente se consiguió saldar estas incompatibilidades con la versión 1.5 de TensorFlow y la versión 2.3.1 de Keras. Estas modificaciones podrían haber causado un verdadero caos de incompatibilidades con otras librerías adicionales, afortunadamente todas estas instalaciones se realizaron utilizando el comando “Conda” de Anaconda, el cual se encarga de instalar y modificar todas las versiones de librerías que sean necesarias para cumplir con la librería a instalar, a su vez, se utilizó un entorno de desarrollo paralelo al principal (“Environment”) para no alterar el normal funcionamiento con la compilación de otros proyectos que requieran otras librerías.

Una vez configurados todos los parámetros, se ejecuta el método “trainModel()” para realizar el entrenamiento. Dada la naturaleza de tiempo necesario para completar el entrenamiento que presenta el equipo utilizado, primero se realizaron entrenamientos cortos con pocas imágenes a fin de verificar el correcto funcionamiento del sistema de entrenamiento y de detección.

El primer ensayo se realizó con solo 30 imágenes y una sola época de entrenamiento, lo cual, como era de esperar no fue suficiente para realizar una detección, pero si para verificar la ausencia de errores de compilación. Aún buscando la primera detección del sistema, se incrementó la base de datos a 480 imágenes, logrando así la primera detección exitosa (Fig. 45), que aunque no se adapte bien a la forma del

escorpión, claramente detecta su presencia con una probabilidad más que aceptable (60,753% que se ve en la citada figura) para su poco entrenamiento.

Fig. 45 Detección con YOLOv3

Por más que se haya conseguido esta detección, el trabajo de entrenamiento recién comenzaba dado que, hasta ese momento esta fue la única imagen que presentó una detección de un conjunto de más de diez imágenes obtenidas de diferentes fuentes. Es por ello que se entrenaron más épocas, continuando con el entrenamiento original. Cada época tomaba cerca de 8 horas en entrenar, lo que hacía no solo muy tedioso el entrenamiento, sino que inutilizaba la computadora durante este período, ya que el consumo de recursos era muy elevado, a tal punto que se debió agregar una base de refrigeración especial preventiva para ayudar a refrigerar la computadora, la cual estaba sobre exigiendo su ventilador.

La computadora utilizada para este entrenamiento fue una Notebook con un procesador Intel Core i5, con placa de video integrada Intel HD Graphics 620 y 24Gb de RAM, 8 de los cuales con dedicación exclusiva a la placa de video. Sin embargo, estas características no lograron un entrenamiento rápido o velocidad aceptable, sino que el sistema demostró ser extremadamente lento en el entrenamiento. Analizando el consumo de recursos durante el entrenamiento se descubrió que la GPU de la computadora no estaba siendo utilizada durante el mismo.

Las librerías CUDA son las encargadas de la paralelización de los sistemas de entrenamiento automático, en particular de administrar el uso de la GPU. Normalmente, con el comando “conda install tensorflow-gpu” se lo instala como librería complementaria, sin embargo, realizar esta operación ocasionaba que no se pudiese compilar ningún código que tuviese TensorFlow. Analizando la documentación de CUDA [102], se llegó a la conclusión de que CUDA no es compatible con las placas de video integradas, por lo que todos los entrenamientos que se estuvieron realizando fueron implementados con la CPU sin paralelización.

Para intentar reducir los tiempos de procesamiento se recurrió a otra computadora con 16Gb de RAM, un procesador AMD A4, y una placa de video NVIDEA con 2Gb de memoria de video integrada. En este equipo no hubo inconvenientes en el uso de la librería mencionada, pero al intentar realizar el entrenamiento los dos gigabytes de memoria de video resultaron insuficiente. Como último recurso se dedicó RAM de manera exclusiva a la memoria de video, pero CUDA no reconocía a esta memoria y seguía saturando toda la memoria disponible en la placa.

Es por todo lo ante dicho que, debido a la elevada necesidad de procesamiento, requerida para este entrenamiento, se debe utilizar una computadora con GPU con memoria dedicada mínima de 4Gb. En nuestro caso, como alternativa, se utilizó "Google Colaboratory" o Colab [93], configurando el uso de la GPU. Este hardware proporcionado por Google nos permitió entrenar 16 veces más rápido, sin tener que reducir la base de datos como antes. Un detalle de interés es que la CPU de Colab no tiene una capacidad de cómputo más elevada que nuestro equipo empleado, es por ello que se pone más restricciones al uso de GPU que al de CPU.

En principio la GPU está destinada para actividades de interacción en tiempo real, por lo que aparecen avisos advirtiendo que está siendo utilizada y que, de no necesitarla, se sugiere liberar este recurso para otro usuario que la necesite. A diferencia de una consola Python normal, Colab no activa automáticamente su GPU por la necesidad del usuario, y no importa que uno instale las librerías de paralelización necesarias, cuando ejecute un programa el sistema seguirá funcionando como lo indique la sesión de máquina virtual que se inició. Para modificar el tipo de hardware de la máquina virtual se debe modificar su configuración accediendo a "Entorno de ejecución -> Cambiar tipo de entorno de ejecución" y allí seleccionar el acelerador de hardware que necesitamos, que por defecto se encuentra en ninguno, pudiendo elegir entre GPU y TPU.

Sin embargo, cabe aclarar que el uso gratuito de esta herramienta tiene una limitación de uso, por lo que, de querer realizar un entrenamiento de mayor tiempo, se deberá fraccionar el tiempo diario de uso a doce horas. La versión Pro de este entorno de desarrollo en línea solo se encuentra disponible para Estados Unidos y Canadá por el momento.

Con esta nueva unidad de cómputo a nuestra disposición, ahora si se utilizó la base de datos, previamente mencionada, en su totalidad, produciendo un tiempo de entrenamiento razonable para la cantidad de imágenes proporcionadas y la efectividad que se quiere conseguir.

Dado que la máquina virtual de Colab no conserva los cambios en los archivos que se almacenan en él, hubo que asegurarse conservar los archivos de entrenamiento una vez finalizado el entrenamiento de la sesión. En una primera instancia se cargaban y descargaban los archivos cada vez que se utilizaba, pero esto demandaba tiempo y ancho de banda, por lo que se utilizó la cuenta de Google Drive asociada a la sesión para almacenar y editar los archivos desde una carpeta permanente. Para dar permiso de lectura y escritura a Colab se debe hacer una solicitud de autorización con inicio de

sesión y código de seguridad generado para garantizar la autenticidad del proceso. El código para dicho procedimiento se muestra a continuación.

```
from google.colab import drive
drive.mount('/content/drive')
```

Finalmente, se realizó el entrenamiento con 100 épocas y un batch size de cuatro en el transcurso de diez días de procesamiento, obteniéndose así un sistema suficientemente entrenado para realizar su propósito con la eficiencia esperada. Para ello se debió crear una variable objeto con la función “DetectionModelTrainer()” para poder utilizar sus métodos y entrenar el modelo como se muestra a continuación:

```
from imageai.Detection.Custom import DetectionModelTrainer

trainer = DetectionModelTrainer()
trainer.setModelTypeAsYOLOv3()
trainer.setDataDirectory(data_directory="Dataset")

trainer.setTrainConfig(object_names_array=["Scorpion"],
batch_size=4, num_experiments=100,
train_from_pretrained_model="pretrained_Yolov3.h5")

trainer.trainModel()
```

Se puede ver el nombre del directorio de la base de datos (Dataset), el nombre del objeto a entrenar (Scorpion o Escorpión) y nuestro modelo pre entrenado de YOLO para iniciar el entrenamiento aprovechando las ventajas previamente mencionadas de Transfer Learning.

El progreso del error de entrenamiento y validación es mostrado en la Fig. 46. Se puede observar como el error de entrenamiento junto al de validación decrecen de manera considerable en las primeras 10 épocas del entrenamiento. Se podría plantear entonces la siguiente pregunta: ¿Por qué seguir con el entrenamiento luego de las 10 épocas? La principal razón se debe a los falsos positivos (FP) que se presentaban en los ensayos de testeo realizados, los cuales disminuyen gradualmente a medida que el sistema se fue entrenando, por lo que, seguir entrenando para disminuir aún más el error de entrenamiento terminó siendo una idea acertada. Desde luego este hecho podría haber generado sobre entrenamiento causando problemas graves de detección, igualmente se puede apreciar que éste no es el caso ya que ambas curvas (train y validation) se mantuvieron en una meseta ligeramente decreciente durante las últimas 90 épocas.

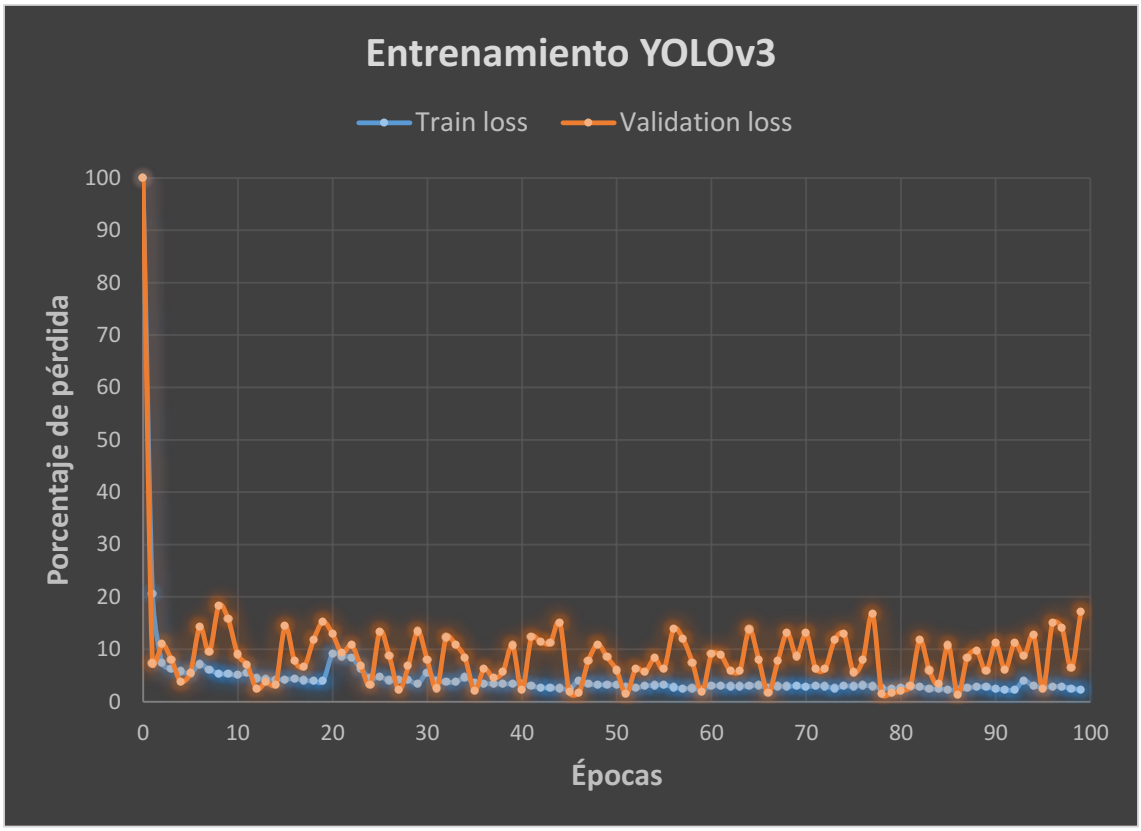


Fig. 46 Progreso de error durante entrenamiento YOLOv3

Debido a que el entrenamiento se desarrolló en el transcurso de diez días en turnos de 12 horas, cada vez que se finalizaba una jornada se ponía en evaluación el modelo entrenado, para ver si realmente se estaba mejorando su desempeño. Para ello se utilizaron las mismas imágenes elegidas en una primera instancia. La Fig. 47 muestra el progreso de la detección de un escorpión a medida que se fue entrenando al sistema. En ella se puede apreciar de manera clara, como la detección comienza de una manera general y a medida que se lo va entrenando se va ajustando al objeto escorpión, incluso, se puede apreciar cómo va aumentando levemente su certeza en el porcentaje de probabilidad de detección que se muestra.

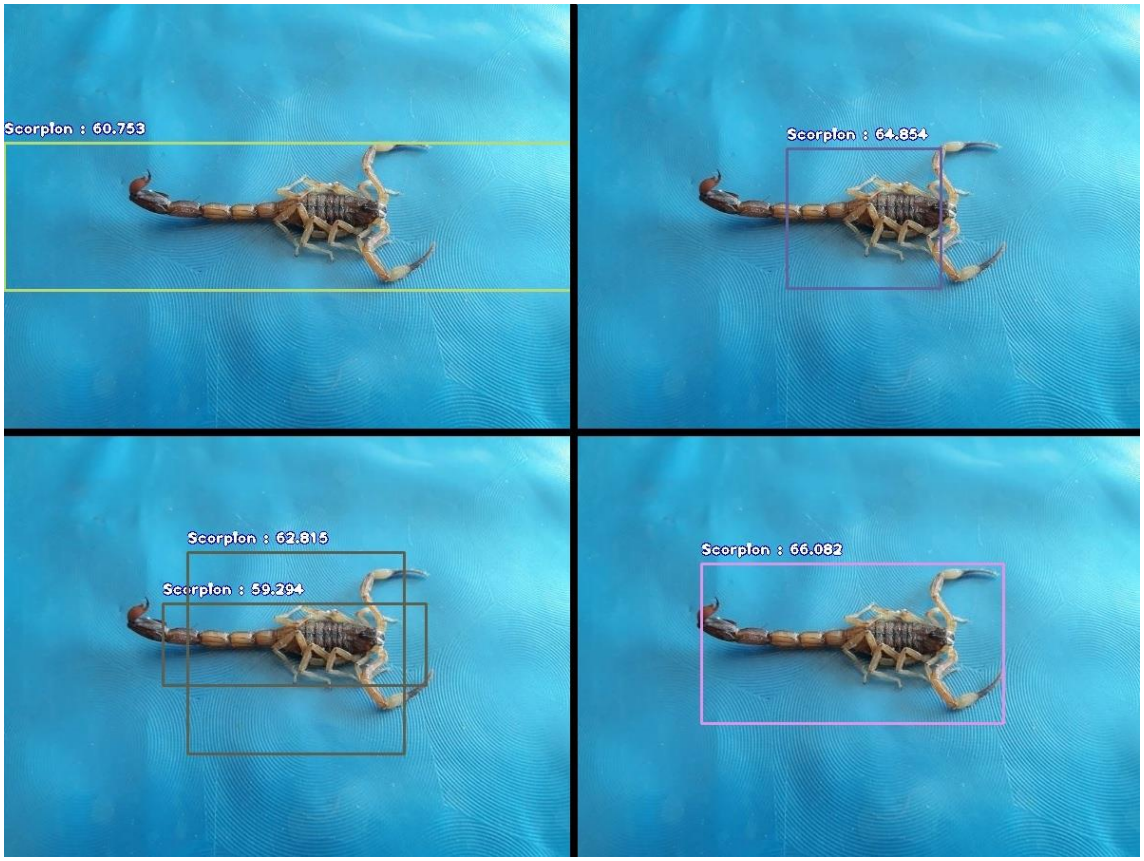
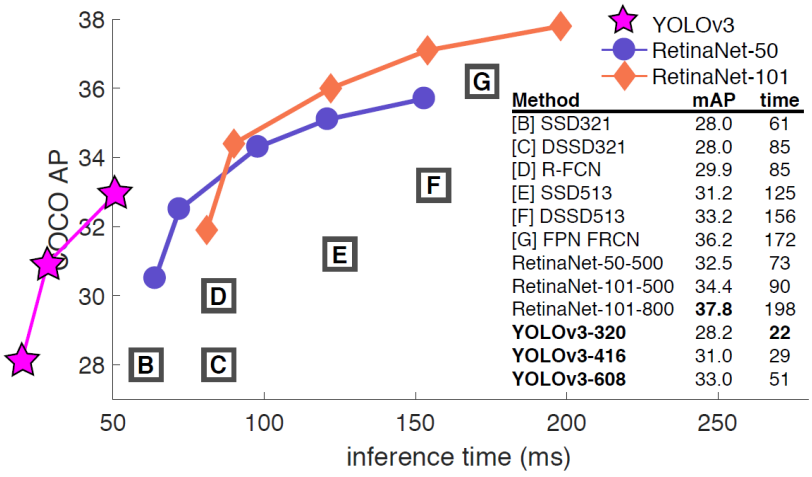


Fig. 47 Progreso de detección durante el entrenamiento dentro de la misma imagen

Entrenado el sistema, se puede utilizar el modelo entrenado directamente con la misma librería de entrenamiento, posibilitando su ejecución en tiempo real (Fig. 48). Esta detección pone de manifiesto el buen desempeño del modelo entrenado ya que esta imagen cuenta con muchos factores que pueden dificultar la detección, como lo es la diferencia de fondo, el reflejo en el vidrio de la derecha, o la rugosidad y heterogeneidad de la tierra de la izquierda. A modo de comparación, esta misma imagen no presenta ninguna detección con el modelo de clasificador en cascada, el cual presentó muchas dificultades en la detección con el fondo de alto contraste que proporciona la tierra.

Fig. 48 Detección tiempo real con YOLOv3

ImageAI permite utilizar múltiples funciones tanto para el entrenamiento como para la detección de objetos, proporcionando, además, una amplia variedad de tipos de modelos diferentes a parte del utilizado YOLOv3. Entre los modelos propuestos se encuentran RetinaNet, ResNet, DenseNet, entre otros. La razón por la que se eligió YOLOv3 en lugar de uno de ellos no fue la exactitud que pudiésemos obtener, sino la velocidad de detección, dado que, para nuestro sistema, la capacidad de respuesta en tiempo real es crucial para evitar accidentes de riesgo sanitario. En la Fig. 49 [103], [104] se puede observar la comparación entre YOLOv3 y otros modelos entre los que se encuentran RetinaNet-50 y RetinaNet-101, mostrando claramente cómo se destaca el primero en lo que refiere a velocidad de detección.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Fig. 49 Comparación de YOLOv3 con otros modelos

Volviendo a la implementación, para poder realizar la detección con esta librería hay que definir, dentro del código, un objeto al que llamaremos “detector” al cual se le

cargaron todos los métodos necesarios para la detección mediante las funciones de ImageAI y los archivos del modelo (.h5) y su configuración (.json) creados durante el entrenamiento. El código para tal fin se muestra a continuación.

```
from imageai.Detection.Custom import CustomObjectDetection

detector = CustomObjectDetection()
detector.setModelTypeAsYOLOv3()
detector.setModelPath("detection.h5")
detector.setJsonPath("detection_config.json")
detector.loadModel()
```

Para la detección en sí, se proveen dos métodos capaces de realizarla, "detectObjectsFromVideo()" y "detectObjectsFromImage()", de los cuales el primero nos sirve para hacer detección a partir de un video, que usaremos para realizar ensayos de eficiencia y comparación con el clasificador en cascada analizando los mismos frames; y el segundo que utilizaremos para la detección en tiempo real, dado que vamos realizando la detección cuadro por cuadro. Este tipo de detección es de gran utilidad dado que podemos ir trabajando con los cuadros de interés. Un ejemplo de esto es el caso de la confirmación de detección por fluorescencia, en el cual requeriremos analizar la parte de la imagen dónde se realizó la detección del escorpión a fin de encontrar su fluorescencia, siempre que estén dadas las condiciones para ello.

La detección en tiempo real mediante este sistema requirió del desarrollo de un programa capaz de capturar la imagen de una cámara y procesar lo devuelto por la detección de la función de la librería antes mencionada. Respecto a esta función se pueden observar los principales parámetros utilizados en la Tabla 6.

Parámetro	**Descripción**	**Valor asignado**
input_type / output_type	Tipo de entrada	Array
input_image	Nombre de variable que contiene a la imagen a procesar	frame
Mínimum_percentaje_probability	Valor mínimo de probabilidad para dar la detección	70

Tabla 6 Parámetros de función de detección con YOLOv3

Esta función nos devuelve dos variables, una es la imagen y la otra que posee información de la o las detecciones. Esta última, a la que llamaremos "detections", nos será de utilidad para marcar los detalles de interés en la imagen en cuestión como se muestra a continuación.

```
for detection in detections:
    print(detection["name"], " : ",
detection["percentage_probability"], " : ", detection["box_points"])
```

5.2.3. YOLOv4

Para el entrenamiento y testeo se recurrió al repositorio “Scaled-YOLOv4” [105], el cual proporciona las librerías y funciones necesarias.

Dicho entrenamiento se realizó con un tamaño de imagen de 416x416 píxeles, un batch size de 16, y 600 épocas. El hardware necesario fue proporcionado por el entorno web de desarrollo de libre acceso “Google Colaboratory” [93], configurando el uso de la GPU, el cual nos proporcionó un “Tesla T4” con 15Gb de memoria dedicada.

El procedimiento demoró 7 horas y 54 minutos en concluir, por lo que la limitación de 12 horas de uso de GPU de Google no afectó el entrenamiento.

Este sistema, cada vez más utilizado por su velocidad y eficiencia frente a otros sistemas similares, utiliza las librerías “models”, “torch”, “yaml”, “utils”, entre otras.

El progreso de la Precisión y el Recall durante el entrenamiento es mostrado en la Fig. 50. Se puede observar su amesetamiento cerca del 70%, lo que garantiza un entrenamiento completo con la base de datos proporcionada, sin llegar al sobre entrenamiento, con los problemas que esto conlleva.

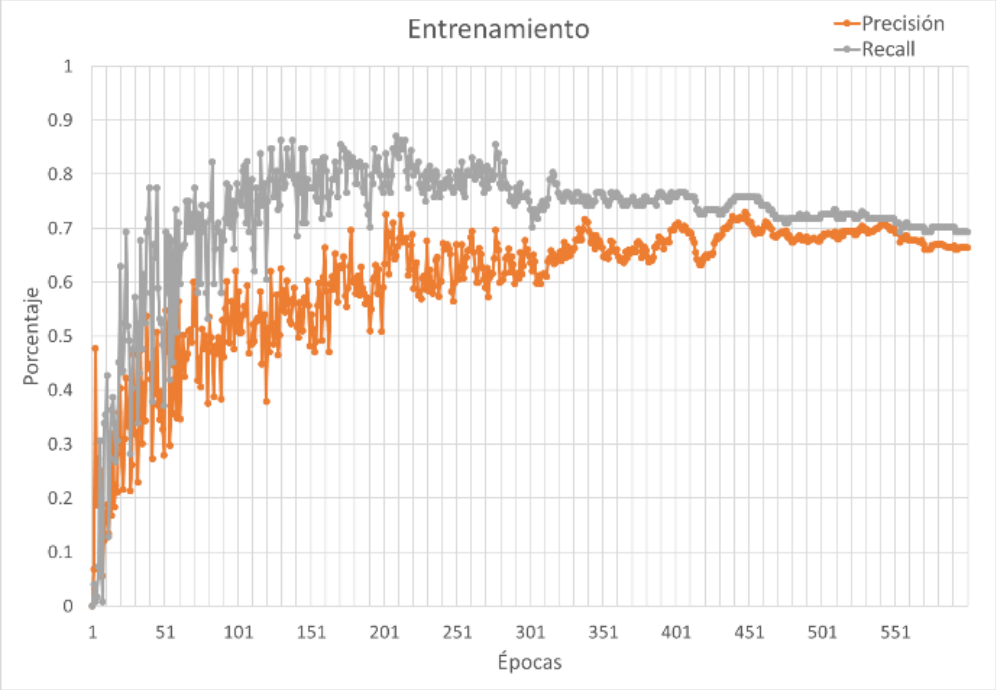


Fig. 50 Evolución de Precisión y Recall durante el entrenamiento YOLOv4

Se puede observar además que el Recall siempre está por sobre la precisión, lo cual es ideal para nuestro sistema que demanda una baja tasa de falsos negativos.

En la Fig. 51 se puede observar una detección de testeo con su respectiva etiqueta, implementado con el sistema entrenado.

Fig. 51 Detección de testeo con YOLOv4

5.2.4. Sistema de confirmación de detección por fluorescencia

5.2.4.1. Detección de fluorescencia

En un inicio se desarrolló un sistema capaz de detectar el rango de valores que representa el color específico de la fluorescencia de los escorpiones ante la presencia de luz ultravioleta mediante el procesamiento morfológico de imágenes digitales. Para ello se utilizaron las librerías de OpenCV en el lenguaje de programación C [106]–[108]. Una representación del funcionamiento se muestra en la Fig. 52.

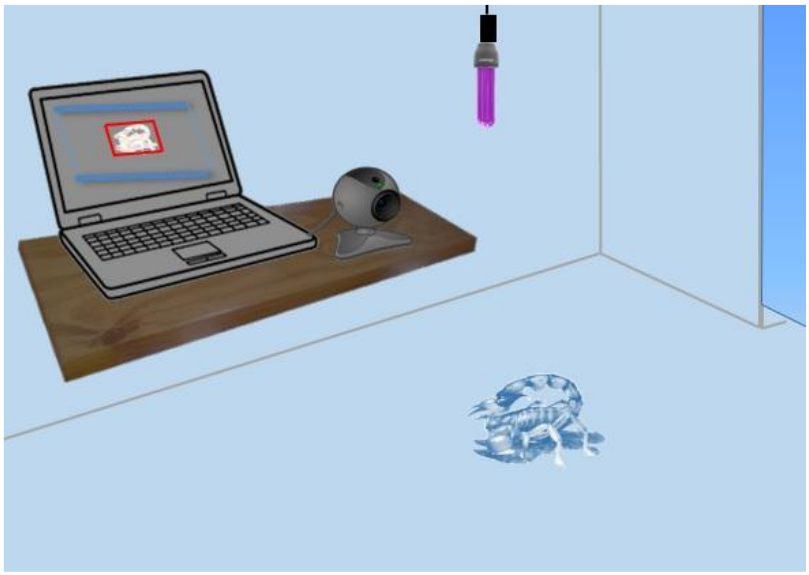


Fig. 52 Representación de sistema de detección de escorpiones por su fluorescencia

Para realizar dicho sistema se utilizó el formato de imagen HSV (Hue Saturation Value, Matiz Saturación Valor) el cual discrimina el color, con un barrido a través de todos los píxeles de la imagen con el fin de encontrar el color deseado. Esto se consiguió mediante un doble ciclo for() para recorrer la imagen a lo alto y ancho píxel a píxel. Luego, mediante una sentencia if(), que compara cada píxel al valor deseado, se realizó la detección, cambiando una variable en caso que la misma sea afirmativa [109].

Para obtener el color a buscar, se creó un programa que toma una captura de la Webcam e imprime el color del píxel que se selecciona con el mouse. Utilizando este software se procedió a realizar un muestreo de múltiples puntos del escorpión, a fin de obtener suficientes muestras para definir un intervalo de valores a utilizar como referencia durante la detección, como se puede apreciar en Fig. 53.

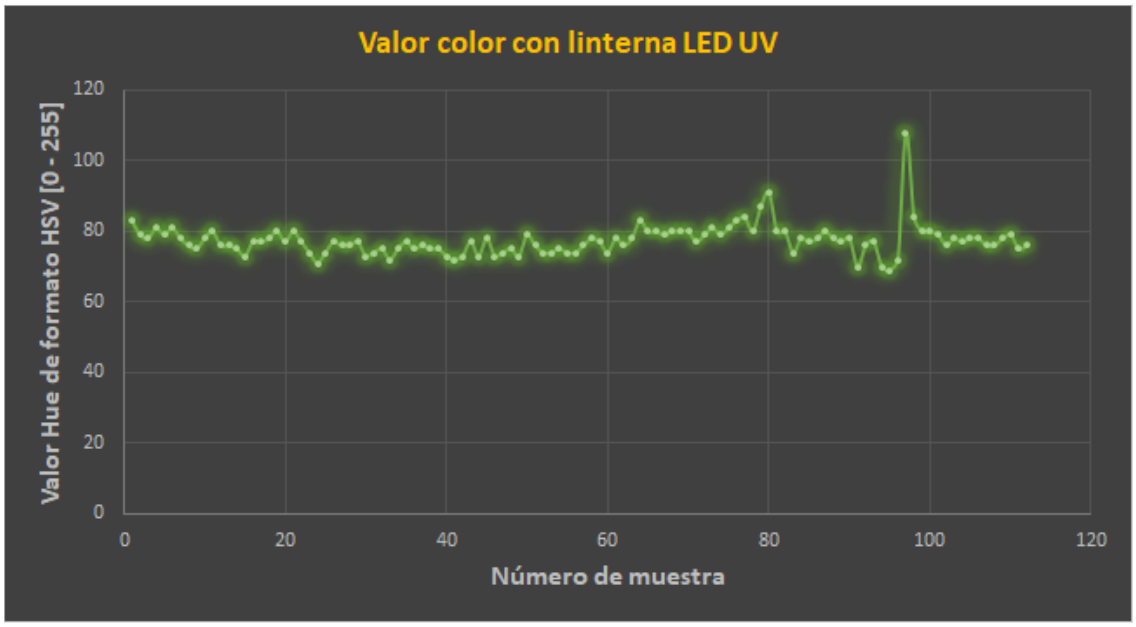


Fig. 53 Valores obtenidos del color del escorpión con fluorescencia

De los datos se obtuvo un valor medio de 77,375 y un desvío estándar de 4,466, por lo que los valores del color deben ir desde 73 hasta 82. Una vez obtenidos estos resultados se procedió a realizar ensayos con el fin de verificar la detección.

Un inconveniente detectado con esta metodología es la presencia de ruido en la imagen por lo que la detección se daba en cualquier situación, hasta en plena oscuridad. Para solucionar este inconveniente se debió reducir el ruido de las capturas obtenidas. Se procedió entonces a realizar un promedio entre dos capturas seguidas, con el fin de que el ruido se reduzca. Dicho procedimiento funcionó dado que ahora la cámara puede permanecer en oscuridad total sin tener una detección errónea.

Con el fin de optimizar el funcionamiento del programa, se introdujo en el mismo un sistema de detección de movimiento por procesamiento digital de imágenes. Esta detección se obtuvo haciendo la diferencia entre dos imágenes y comprobando que esta no supere un umbral mínimo, en caso de superarse se detecta movimiento [110]. La diferencia entre imágenes se realizó en blanco y negro con el fin de minimizar el procesamiento. Las imágenes por diferenciar han sido previamente procesadas para eliminar el ruido, es decir que para realizar este procedimiento se requirieron cuatro capturas (dos por cada imagen sin ruido).

La optimización del programa radica en que la idea es solo procesar en formato HSV aquellos pixeles que hayan tenido movimiento. Para ello es necesario conocer el área de píxeles que se han movido, esto se logra creando una imagen blanco y negro en la cual los pixeles blancos han superado el umbral mínimo requerido para detectar movimiento y los negros los que no se han movido. Utilizando la función cvFindContourns() se obtienen los contornos de estas áreas de movimiento, luego se establece un área mínima para no detectar puntos solitarios y se utiliza la función

cvBoudingRect() para establecer un rectángulo mínimo que encierra esa área de movimiento y la función cvRectangleR() para dibujar este rectángulo en pantalla. Este rectángulo se realizó para tener los límites del objeto animado y poder analizar en ese rectángulo solo la presencia del color deseado.

Ante la presencia de falsas detecciones con la metodología anterior, debido principalmente a la presencia del color buscado en pixeles aislados dentro del movimiento, producto del ruido de la imagen y la inestabilidad del algoritmo, se procedió a realizar un procesamiento de las imágenes similar a la detección de movimiento, pero esta vez, para la detección del color de la fluorescencia. Para ello, se creó una nueva imagen en la cual los pixeles blancos corresponden al color de la fluorescencia buscada y los negros a los restantes colores, tal y como se observa en la Fig. 54.

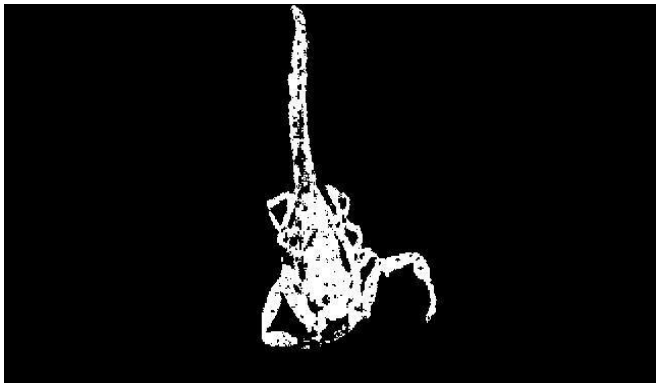


Fig. 54 Imagen de escorpión solo con color de fluorescencia

Utilizando las mismas funciones que para la detección de movimiento se encontraron las áreas correspondientes a este color dentro de la imagen blanco y negro de la fluorescencia generada previamente. Del resultado de dicha búsqueda de contornos, se filtraron las áreas más pequeñas para no detectar puntos solitarios (reducción de ruido), al igual que con la detección de movimiento y se procedió a realizar la detección con esta nueva metodología.

Una vez finalizados los ajustes de este sistema, se procedió a realizar ensayos en presencia de otros especímenes que puedan estar normalmente en el mismo ambiente que un escorpión. El primer ensayo de este tipo se realizó con un bicho bolita (*Porcellio laevis*), dicho ensayo demostró un inconveniente que se ilustra a continuación con una imagen del color buscado como el caso anterior.

Fig. 55 Imagen de un escorpión (izquierda) y de un bicho bolita (derecha).

Como se observa en la Fig. 55, el escorpión presenta un color más uniforme mientras que el bicho bolita presenta el color en su contorno, por lo que esta presencia del color se da por el reflejo de su superficie húmeda.

Para eliminar este problema se utilizó un método combinado de erosión y dilatación. Esta decisión se debe a que, al ser la fluorescencia del escorpión de color muy uniforme, realizar una dilatación uniría las áreas internas, y realizar una posterior erosión no sería tan nefasto como lo sería para una línea de un reflejo. Posteriormente se haría una dilatación para evitar que queden áreas muy pequeñas que sean descartadas. Para llegar al nivel óptimo de iteraciones de dilataciones y erosiones se procedió a realizar ensayos con la imagen previamente mostrada. Los resultados de los ensayos son mostrados de la Fig. 58 hasta la Fig. 61, incrementando paulatinamente la cantidad de dilataciones y erosiones.

En la Fig. 56 una erosión simple demuestra que si bien ha desaparecido la imagen del bicho bolita, la degradación del escorpión se vuelve importante, lo cual en casos de menor fluorescencia podría desaparecer perdiéndose la posibilidad de detección.

Fig. 56 Una erosión simple.

En la Fig. 57 con la primera dilatación, aunque se resalta el brillo del bicho bolita, consolida la imagen del escorpión haciendo que se unan sus áreas internas, lo que, ante futuras erosiones, dificultará su desaparición. Por otra parte, el bicho bolita ante futuras erosiones se degradará considerablemente dado que su color siguen siendo líneas de su contorno sin un área densa.

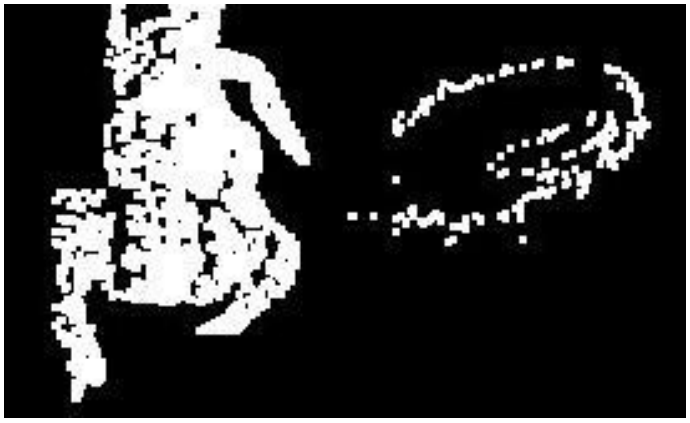


Fig. 57 Una dilatación inicial.

En la Fig. 58, luego de dos dilataciones y seis erosiones el bicho bolita desapareció por completo, dejando al escorpión muy degradado, pero con un área suficiente para poder seguir trabajando. Para compensar la degradación del escorpión se procedió a realizar nuevas dilataciones que le devuelvan algo de su forma y tamaño original.

Fig. 58 Imagen resultado de 2 dilataciones y 6 erosiones.

Fig. 59 Imagen resultado de 2 dilataciones, 6 erosiones y 8 dilataciones.

Finalmente, la Fig. 59 obtenida luego de realizar 2 dilataciones, 6 erosiones y 8 dilataciones, muestra el resultado aceptable que se buscaba, en la cual no se observa al bicho bolita, y el tamaño y densidad del escorpión se pueden apreciar de manera adecuada. Las funciones de OpenCV utilizadas para estos procesos son *cvErode()* y *cvDilate()*.

5.2.4.2. Método de confirmación de detección mediante la fluorescencia de los escorpiones.

Con el fin de poder detectar los escorpiones, se presenta el desarrollo de un sistema de alarma temprana utilizando detección de formas mediante visión por computadora y la propiedad de fluorescencia antes mencionada. Los métodos de detección de formas utilizados para este sistema fueron Haar y YOLOv3, previamente descriptos.

En este trabajo se plantea la utilización de la detección de la fluorescencia de los escorpiones para confirmar la presencia de los mismos ante su detección como objetos (Fig. 60), dada por las heurísticas Haar y YOLO. Es por ello que los ensayos se centraron en determinar con qué seguridad se detecta el color buscado de la fluorescencia, y así garantizar un sistema más seguro de detección evitando falsos positivos de detección del objeto por no hallarse la fluorescencia característica de estos arácnidos. Desde luego esta aplicación es para uso nocturno, o lugares con ausencia de luz natural, dado que no es posible la visualización de la fluorescencia durante el día.

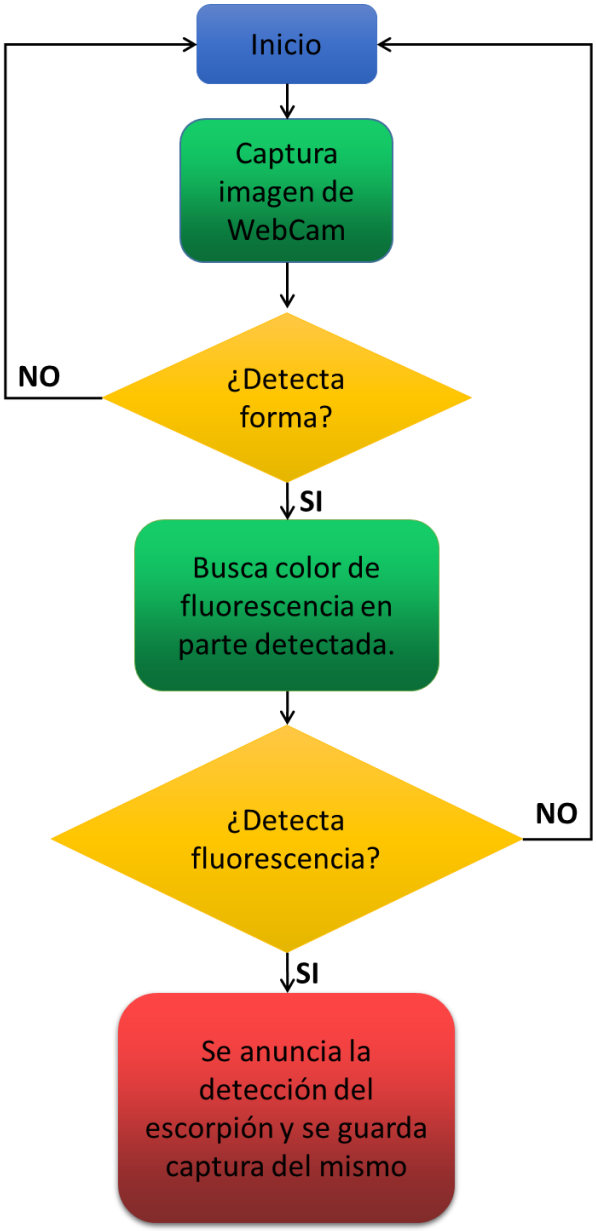


Fig. 60 Diagrama de funcionamiento del sistema de validación por fluorescencia.

Para facilitar la interoperabilidad entre los dos sistemas propuestos, “Haar + UV” y “YOLO + UV”, se realizó una función genérica para la detección de la fluorescencia que devuelve una respuesta booleana indicando la presencia o no del color buscado. Esta se puede observar a continuación.

```
def deteccion_fluorescencia(img):
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    color = hsv[:,:,0]
    count = 0
    for x in color:
        for c in x:
            if(c<82 and c>73):
```

```
                count += 1
    if(count>300):
        flag=1
    else:
        flag=0
    return flag;
```

Se puede observar que en este caso nos basamos en la cantidad de píxeles que poseen el color buscado en la fluorescencia de los escorpiones, esto se da para evitar falsos positivos por presencia de ruido en la imagen. Cabe aclarar que esta sería una versión básica de funcionamiento, si recordamos el funcionamiento de la versión original de detección de fluorescencia el sistema de detección del mismo se basaba en áreas, lo que se buscaba eran grupos de píxeles del color buscado que generasen un área mayor a 200 píxeles. Este mecanismo, aunque más complejo de implementar, mejoraba el rechazo a FP por descartar áreas pequeñas de fluorescencia, usualmente causadas por presencia de ruido en la imagen.

Una vez detectada la fluorescencia, se muestra en imagen una advertencia indicándolo (Fig. 61).

Fig. 61 Detección YOLOv3 con confirmación por fluorescencia

El cálculo del desempeño de la detección de este sistema se realizó mediante el análisis de detecciones de escorpiones iluminados con UV, y de detecciones erróneas inducidas en un ambiente iluminado con luz natural (sin fluorescencia), en cantidades iguales para tener una base de datos equilibrada. Es decir, se obtuvieron la misma cantidad de detecciones de objeto tanto con UV como con luz natural, en cada caso se analizó si hubo, o no, una detección correcta de la fluorescencia.

5.2.5. App "Detecta Escorpión"

5.2.5.1. Descripción

La implementación de la versión "Lite" de TensorFlow se realizó debido a la necesidad de hacer funcionar el sistema de detección en un teléfono inteligente (*smartphone*). Para ello se recurrió al uso de la aplicación de ejemplo proporcionada por TensorFlow, realizando las modificaciones necesarias para poder utilizar el modelo a entrenar.

En una primera instancia, se verificó el correcto funcionamiento del mismo, utilizando el modelo entrenado por defecto. Como se puede ver en la Fig. 62, la App funcionó correctamente, por lo que se procedió a entrenar un nuevo modelo con una base de datos propia. Cabe aclarar que, como la versión de compilación de TensorFlow y otras librerías utilizadas no son iguales a la que se utilizó para el modelo ejemplo, para lograr el correcto funcionamiento de nuestro modelo se debieron realizar modificaciones importantes a nivel de código, sobre todo en el llamado de la librería, llamado a funciones de manera diferente, nombre distintos en nombres de variables, entre otros.

Fig. 62 Demostración de correcto funcionamiento de app ejemplo

Android Studio provee de una interfaz de simulación para probar las apps que se desarrollen. Para utilizarla se requiere la instalación de un paquete del programa llamado "virtualización", dependiendo el procesador que tengamos en nuestra PC. Para

que esta característica del programa pueda ser utilizada, se debe estar seguro de tener habilitada la misma en el BIOS del sistema. Otra verificación que se debe hacer, es que hay antivirus que por defecto bloquean la virtualización, por lo que hay que editar su configuración para que permita la simulación.

Una vez hecho lo anterior tendremos la capacidad de emular cualquier dispositivo Android que se encuentre en el mercado, con la opción de hacerles modificaciones propias, que nos permitan evaluar nuestra app en diferentes consolas.

5.2.5.2. Modelo

Para el entrenamiento de nuestro modelo se utilizó la heurística de transfer learning con el modelo pre entrenado MobileNetv2 con un batch size de 12, 100 iteraciones por época y un total de 400.000 épocas, las cuales demoraron cuatro días en concluir.

Como primera etapa, se debe descargar el modelo base a utilizar, luego se lo debe configurar con nuestras preferencias de entrenamiento del pipeline, para finalmente inicial el entrenamiento con la siguiente sentencia que alberga a todas las variables ya definidas.

```
!python /content/models/research/object_detection/model_main.py \
    --pipeline_config_path={pipeline_fname} \
    --model_dir={model_dir} \
    --alsologtostderr \
    --num_train_steps={num_steps} \
    --num_eval_steps={num_eval_steps}
```

Siendo num_steps=400.000 (épocas) y num_eval_steps=100 (número de iteraciones por época).

El entrenamiento se realizó mediante Colaboratory, tomando un total de 32 horas. En la Fig. 63 se muestra el progreso de la pérdida a medida que el sistema se fue entrenando.

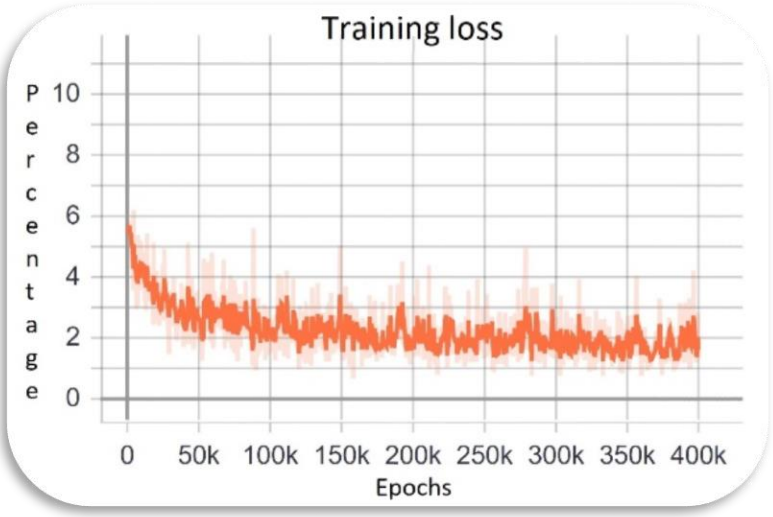


Fig. 63 Entrenamiento de modelo de detección MobileNetV2

Este gráfico fue obtenido mediante el uso de la herramienta TensorBoard de la librería TensorFlow, para ello se requiere abrir un puerto para acceder a la visualización de la interfaz gráfica proporcionada por ésta. Para realizar dicho procedimiento se recurrió a lo siguiente.

```
!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-
amd64.zip
!unzip -o ngrok-stable-linux-amd64.zip
```

```
LOG_DIR = model_dir
get_ipython().system_raw(
    'tensorboard --logdir {} --host 0.0.0.0 --port 6006 &'
    .format(LOG_DIR)
)
```

```
get_ipython().system_raw('./ngrok http 6006 &')
```

```
! curl -s http://localhost:4040/api/tunnels | python3 -c \
    "import sys, json;
print(json.load(sys.stdin)['tunnels'][0]['public_url'])"
```

Esta última genera el enlace que nos abrirá el puerto, permitiéndonos visualizar varias características del modelo, entre las que se encuentran su conformación de capas y sus gráficas de entrenamiento, como son la pérdida antes mostrada.

Este tipo de gráfica se caracteriza por aumentar su valor a medida que la pérdida se reduce. Por lo que se puede observar la curva en dicha gráfica reduce su pendiente cerca de las 400k épocas, por lo que el sistema no mejorará más por ser entrenado durante más tiempo.

5.2.5.3. Implementación

Como base de datos se recurrió a una nueva versión, diferente a la utilizada para YOLOv3, en este caso se seleccionaron las mejores imágenes de la base de datos anterior y se agregaron nuevas fotografías, adquiridas al fotografiar todos los ejemplares de la colección de escorpiones del Laboratorio de Aracnología del CEPAVE, llegando a un total de 612 imágenes de muy buena calidad y nitidez. Además, se utilizó Roboflow para incrementar la base de datos llegando a 1464 imágenes de escorpiones demarcadas [111].

Estas imágenes fueron divididas para su uso en proporción 70:20:10, es decir 70% para entrenamiento, 20% para validación y 10% para testeo. Como el entrenamiento involucra solo a las primeras dos, podemos decir que hay una relación 77:23, lo cual se acerca mucho a "el principio de Pareto" [112] de 80:20.

La exportación de la base de datos para el entrenamiento se realizó en el formato TFRecord, y se generó un enlace para poder utilizarlo directamente desde Colaboratory sin tener que almacenarlo en un equipo propio, por lo que cada vez que vamos a realizar un entrenamiento con dicha base de datos se utiliza dicho enlace para acceder a la base de datos. El comando para la mencionada importación desde el repositorio de Roboflow es el que se muestra a continuación, siendo el "Enlace" la dirección web proporcionada por Roboflow para exportar la base de datos.

```
%cd /content/darknet
!curl -L "Enlace" > roboflow.zip; unzip roboflow.zip; rm roboflow.zip
```

Los valores de edición de imágenes para el incremento de base de datos fueron los siguientes:

- Dar la vuelta (horizontal y vertical)
- Rotación 90° (en ambos sentidos)
- Rotación 45° (en ambos sentidos)
- Ladear 45° (ambos lados)
- Saturación 48% (más y menos)
- Exposición 25% (más y menos)
- Desenfoque (1.75pX)
- Ruido (5%)

Como se mencionó antes, para la implementación del modelo entrenado se utilizó un ejemplo de app proporcionada por TensorFlow, el mismo puede ser compilado con el nuevo modelo mediante el programa "Android Studio", de más está aclarar que solo funcionará en dispositivos con este sistema operativo.

Dado que nuestro sistema entrenado se encuentra en formato TensorFlow, y que nosotros necesitamos la versión Lite del mismo, se debió realizar la correspondiente conversión del modelo. TensorFlow provee de la función necesaria para realizar dicha conversión, la cual se puede visualizar a continuación.

```
!tflite_convert \
  --graph_def_file="/content/models/research/fine_tuned_model/
tflite/tflite_graph.pb" \
  --output_file="/content/models/research/fine_tuned_model/
tflite/final_model.tflite" \
  --input_shapes=1,300,300,3 \
  --input_arrays=normalized_input_image_tensor \
  --output_arrays='TFLite_Detection_PostProcess',
'TFLite_Detection_PostProcess:1',TFLite_Detection_PostProcess:2',
'TFLite_Detection_PostProcess:3' \
  --allow_custom_ops
```

En ella se definen la ubicación del graph del modelo entrenado, la dirección del archivo convertido, las dimensiones de la imagen que recibirá el modelo (300x300), tipo de arreglo de entrada y de salida, y el último argumento que permite operaciones controladas, lo cual evita múltiples problemas con variables del modelo.

Para que el modelo entrenado funcione se debieron realizar modificaciones al código, el más relevante de ellos fue uno que involucra la importación de la librería "java.io.InputStream" dentro del archivo "TFLiteObjectDetectionAPIModel.java", esto se debe a que la librería "…InputStreamReadear" no es compatible en todas sus funciones con nuestra versión de modelo creada. Esto puede ser causado por diferente versión de compilación de ambos proyectos. Como consecuencia de esta modificación de librería, se tuvieron que modificar varias funciones. Una vez que encontramos este problema, las modificaciones necesarias a realizar fueron encontradas en un repositorio de modificación de código para solucionar este problema [113], desarrollado por otro usuario, dentro del repositorio base de TensorFlow [114]. Otra modificación para que el modelo funcione bien en la app, es asegurarse de establecer que no es una app cuantificada, dejando la variable ("*TF_OD_API_IS_QUANTIZED*") con valor "false" en el archivo "DetectorActivity.java".

Habiendo modificado lo antes dicho y entrenado nuestro modelo se procedió a comprobar su correcto funcionamiento, el cual se muestra en la Fig. 64. El desempeño del sistema se mostrará en el Capítulo 6 Resultados.

Fig. 64 Detección de escorpión con app de detección

Se puede ver no solo la correcta demarcación de la detección del escorpión, sino también la precisión con la que asegura su detección (98,96%). Cabe aclarar que el sistema cuenta con una baja tasa de falsos positivos en un ambiente sin presencia de escorpiones. Cuestión que, de ser necesario, se puede solucionar aumentando el entrenamiento o aumentando el porcentaje mínimo para realizar la detección. En un caso extremo se debería modificar la base de datos para corregir este inconveniente.

Por otro lado, la detección del objeto por defecto se realiza cuando su probabilidad supera el 50%, sin embargo, este valor puede ocasionar una gran cantidad de falsos positivos. Considerando esto último, se cambió este valor por 90%, decisión debida a que todas las detecciones correctas de escorpiones se dan por sobre este valor de probabilidad, la mayoría incluso por sobre 95%, y los falsos positivos, aunque no frecuentes, llegan a tener valores entre 60% y 80%. Esta modificación se puede realizar dentro del archivo “DetectorActivity.java”.

Un problema común en la detección de objetos es la detección múltiple del mismo objeto por superposición de más de una “caja de detección” con probabilidad superiores al mínimo (Fig. 65). Para solucionar este problema existe la fusión de detecciones por superposición, para ello se tienen en cuenta el porcentaje de área de imagen que comparten las detecciones en cuestión. Por defecto el valor para fusionar dichas sobre detecciones era de un área mayor al 95%, pero dada la ocurrencia de este inconveniente se decidió reducirlo al 90%, logrando así eliminar el problema en los sucesivos ensayos

que se realizaron posteriormente. Esta modificación se realizó en el archivo “DetectorTest.java”.

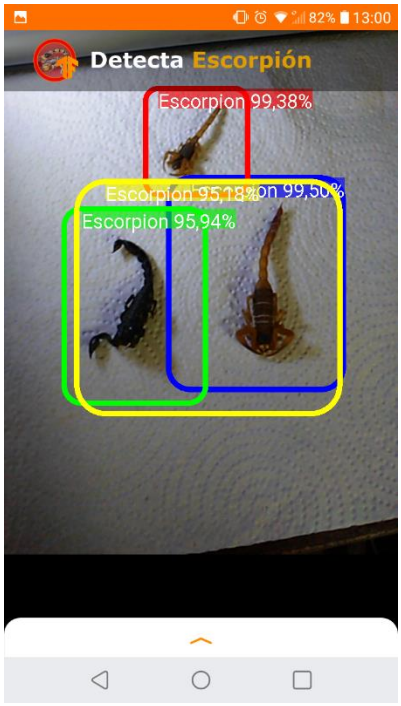


Fig. 65 Múltiples detecciones superpuestas de escorpiones

Una vez comprobado el correcto funcionamiento del modelo entrenado modificamos estéticamente la aplicación. En otras palabras, modificar el ícono, el nombre, cambiar los logos, y la apariencia a fin de hacerlo estéticamente más acorde a la aplicación que le vamos a dar.

En primer lugar, se creó un ícono que represente a la aplicación, la Fig. 66 cumple con esa función ya que cuenta con un escorpión demarcado en rojo por su detección y se encuentra la presencia del logo TensorFlow dado que es la librería utilizada para su implementación y, además, es quién proveyó de la app base para su modificación.

Fig. 66 Ícono de la app de detección

También se creó la barra transparente superior de la interfaz de uso de la app, que puede observarse en la Fig. 67 y está constituida del logo más una descripción del propósito de la app con el mismo formato a la barra original.

Fig. 67 Barra de diseño para app de detección

Para el nombre de la aplicación se eligió una contracción que no ocupase tanto espacio, por lo que se eligió “Det Escorp”. Una vez editadas todas las características de la aplicación, se procedió a su implementación y estudio de desempeño a fin de mejorar el sistema en caso de ser necesario.

5.3. Sistemas de clasificación de escorpiones

A continuación, en esta sección se presentarán los diferentes sistemas de clasificación de escorpiones desarrollados.

5.3.1. Sistema de reconocimiento y clasificación de tres especies de escorpiones.

Se utilizó un conjunto de datos de 132 imágenes de dos géneros de escorpiones (56 *Bothriurus* y 76 *Tityus*) para entrenar y probar tres modelos de Machine Learning para el reconocimiento de imágenes y la clasificación de escorpiones. Estas imágenes fueron obtenidas de la base de datos fotográfica del Laboratorio de Aracnología del CEPAVE. En concreto, se utilizaron 30 imágenes de cada género para el entrenamiento (45% de los datos) y el resto se utilizó con fines de prueba.

Los tres modelos de ML utilizados y presentados en esta subsección, se implementaron en Python. Además, se utilizó un procesador Intel Core i5 de séptima generación, 8 GB de RAM y una tarjeta gráfica Intel HD Graphics 620.

5.3.1.1. Modelo LBPH

Para crear el modelo a entrenar, se implementó el sistema LBPH en OpenCV importando la biblioteca cv2, y usando la función LBPHFaceRecognizer_create (). Se utilizó Python 3.7 y OpenCV 3.0.0.26.

Como primer paso, una vez seleccionados los datos, las imágenes se almacenan en una matriz para ser utilizadas en la fase de entrenamiento, utilizando la función de tren. Luego, todos los datos del modelo entrenado se almacenan en el archivo trainer.yml. Previo a la tarea de predicción, las imágenes se redimensionaron a 150 x 150 píxeles y se transformaron en escala de grises. Finalmente, la función de predicción (imagen) devuelve la confianza de la predicción correspondiente. El tiempo de procesamiento para entrenar el modelo fue del orden de dos minutos.

El algoritmo LBPH es comúnmente utilizado para el reconocimiento de rostros, por lo que, para asegurarnos su correcta implementación, en una primera instancia se lo probó para reconocer los rostros de algunos integrantes de nuestro laboratorio como se puede ver en la Fig. 68.

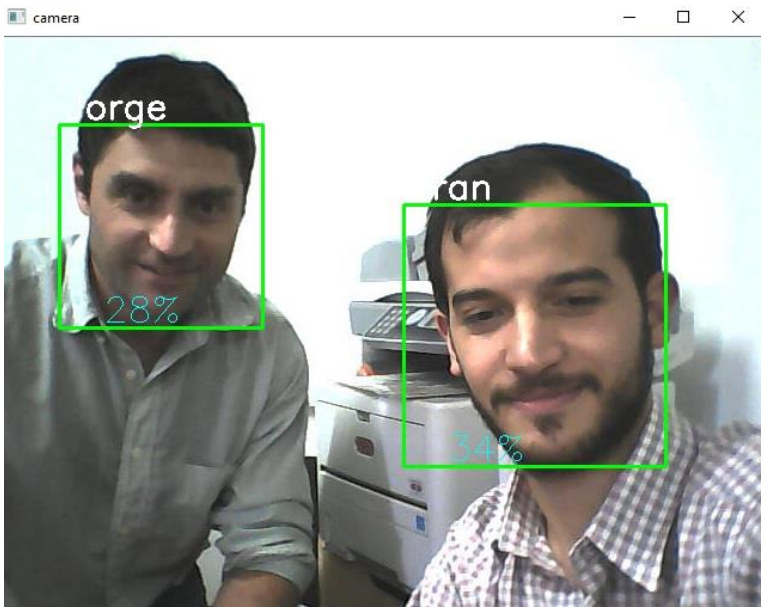


Fig. 68 Detección e identificación de rostros con LBPH

Una vez comprobado el correcto funcionamiento con rostros, se procedió a realizar el entrenamiento con los escorpiones. En la Fig. 69 se puede visualizar la correcta detección y clasificación de dos escorpiones, uno de importancia sanitaria (*Tityus confluens*) y el otro sin importancia sanitaria (*Bothriurus bonaerensis*).

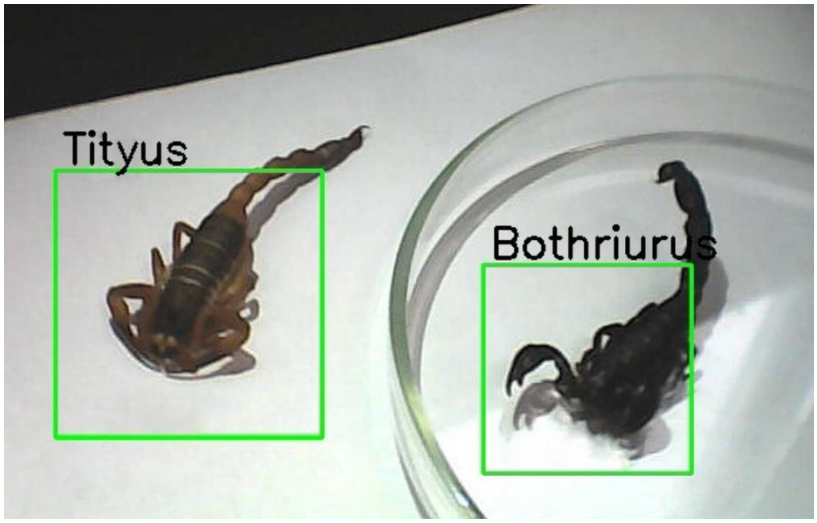


Fig. 69 Detección e identificación de dos escorpiones de diferente género

Se puede observar la correcta detección y clasificación de ambos, por lo que este sistema fue tomado en cuenta para los estudios de efectividad pertinentes.

5.3.1.2. Modelo DNN con Transfer Learning

En los últimos años se han aplicado métodos basados en el Aprendizaje Profundo como alternativas novedosas para el reconocimiento de imágenes [115]. Los algoritmos de Aprendizaje Profundo, como la red neuronal profunda (DNN: *Deep Neural Networks*) o la red neuronal convolucional (CNN: *Convolutional Neural Network*), tienen la capacidad de extraer automáticamente características representativas útiles de los datos de la imagen durante el proceso de entrenamiento. En particular, un DNN es una red neuronal artificial de retroalimentación, con muchas capas ocultas entre sus entradas y salidas.

Es bien sabido que los algoritmos de Aprendizaje Profundo están hambrientos de datos y producen modelos más potentes y precisos a medida que se introducen más datos en la red. Dado que el número disponible de datos de entrada en este trabajo (132 imágenes de escorpiones) es limitado y pequeño, el proceso de entrenamiento de DNN puede resultar en sobreajuste y bajo rendimiento en la fase de prueba. La heurística conocida como aprendizaje por transferencia (TL: *Transfer Learning*) se suele utilizar para mejorar esta limitación y evitar entrenar desde cero [116].

En el enfoque de TL, se comparten los parámetros aprendidos (en particular los pesos) de redes bien entrenadas en un conjunto de datos muy grande. Luego, se modificaron las últimas capas completamente conectadas del modelo preentrenado para adaptar el modelo DNN a la tarea de clasificación del escorpión. Aunque los conjuntos de datos son diferentes, las características de bajo nivel, como los bordes, son similares. Por lo tanto, el método TL puede mejorar el rendimiento y reducir el tiempo

de entrenamiento y el costo de procesamiento, incluso para un pequeño conjunto de datos. El aprendizaje por transferencia se utiliza ampliamente en imágenes médicas con gran eficiencia [117], [118].

En este trabajo se entrenó un modelo de aprendizaje secuencial y denso utilizando un DNN con TL. La red neuronal previamente entrenada utilizada fue VGG16 [85], que se ha importado de la biblioteca de Keras mediante el backend de TensorFlow. La red VGG16 es una arquitectura de 16 capas (convolucional y completamente conectada) construida, basada en el conjunto de datos ImageNet, con el propósito de reconocimiento y clasificación de imágenes. En este caso, se utilizaron imágenes en color redimensionadas (150 x 150 píxeles para cada imagen) para reducir el costo computacional de entrenamiento y evaluar el modelo. La red se entrenó durante 100 épocas, con una tasa de aprendizaje de 0,001 utilizando un optimizador Adam. La Tabla 7 muestra la arquitectura de red optimizada del modelo propuesto.

Layer (type)	Output shape	Param#
Flatten_1 (Flatten)	(None, 8192)	0
Dense_1 (Dense)	(None, 256)	2097408
Dropout_1 (Dropout)	(None, 256)	0
Dense_2 (Dense)	(None, 1)	257
Dense_3 (Dense)	(None, 512)	1024
Activation_1 (Activation)	(None, 512)	0
Dropout_2 (Dropout)	(None, 512)	0
Dense_4 (Dense)	(None, 1)	513
Activation_2 (Activation)	(None, 1)	0

Tabla 7 Arquitectura de DNN con TL

Durante el entrenamiento, la exactitud (*accuracy*) alcanzó un 98%, mientras que para la validación alcanzó y se mantuvo cerca del 70% (Fig. 70). Por lo que se ha realizado un entrenamiento satisfactorio con valores aceptables y sin sobre entrenamiento.

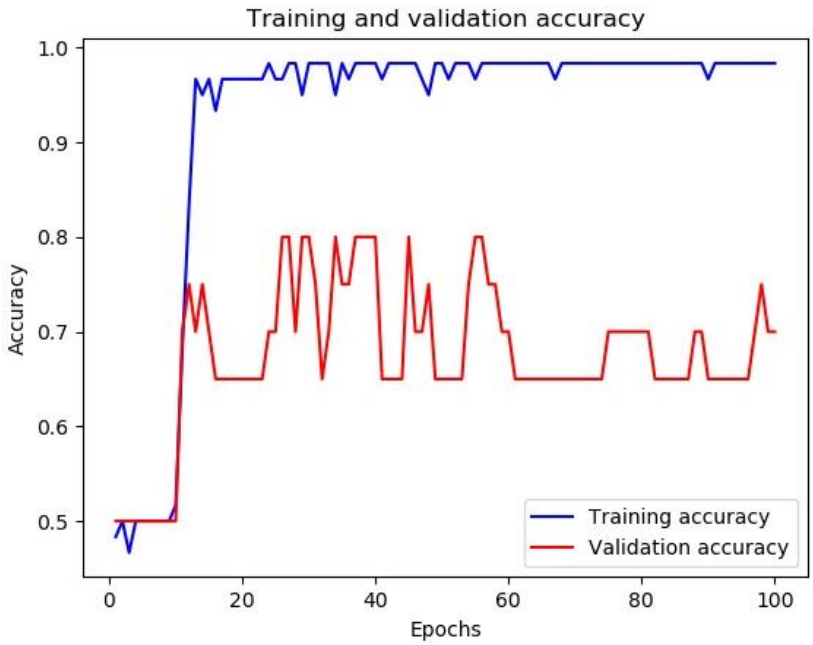


Fig. 70 Gráfica accuracy de entrenamiento y validación con VGG16

Por su parte, el error (*loss*) de entrenamiento se redujo hasta el orden del 20% de manera exponencial como se puede apreciar en la Fig. 71, sin embargo, el error de validación lo hizo hasta el 50%, lo que podría sugerir un problema con el modelo entrenado. De todos modos, al ser probado con las imágenes de testeo demostró ser un modelo con buen desempeño como se verá en el Capítulo 6 Resultados.

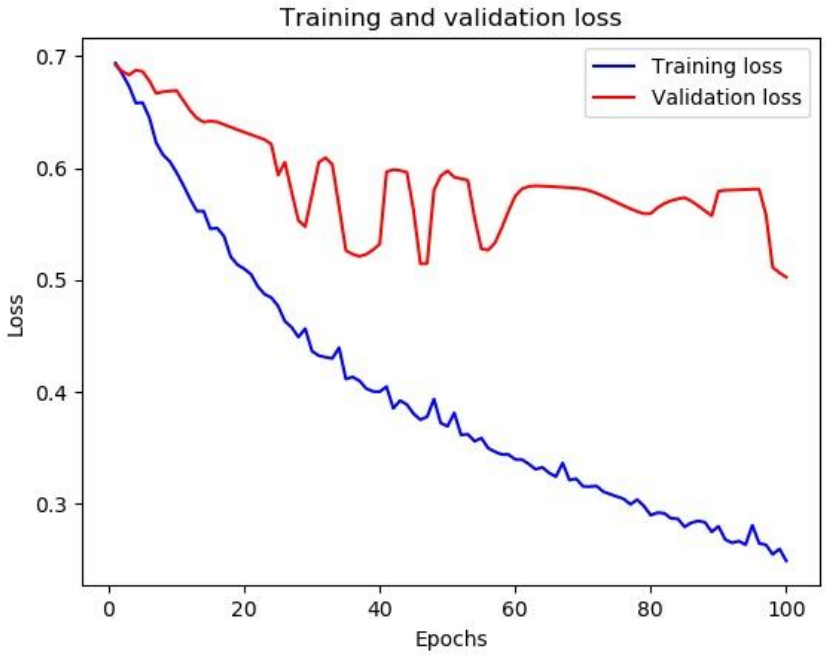


Fig. 71 Gráfica loss de entrenamiento y validación con VGG16

El tiempo de entrenamiento fue de menos de un minuto, mientras que el tiempo de procesamiento para entrenar el modelo sin la red preentrenada VGG16 fue cercano a los 90 minutos. Dado que el problema de reconocimiento puede considerarse como un problema dicotómico, los modelos se entrenaron utilizando el algoritmo de entropía cruzada binaria (BCE: *Binary Cross Entropy*), que proporciona una respuesta booleana a cada observación (presencia de una u otra especie).

5.3.1.3. *Modelo DNN con TL y Data Augmentation*

Recientemente, en el caso de pequeños conjuntos de datos, el proceso conocido como aumento de datos (DA: *Data Augmentation*) es otra estrategia utilizada para mejorar el entrenamiento de DNN. El propósito de este método es aumentar significativamente el número de imágenes recopiladas para el conjunto de datos, con el fin de evitar el sobreajuste y mejorar la generalización del modelo [52], [119], [120].

Por lo tanto, las operaciones de transformación de imágenes, como rotación, corte, traslación, zoom, etc., se pueden aplicar al conjunto de datos original para producir nuevas versiones. En este trabajo se ha aplicado la heurística DA al modelo con la red preentrenada VGG16, y se ha implementado utilizando la clase ImageDataGenerator de Keras, con los parámetros mostrados en la Tabla 8.

Parameter name	Parameter value
rotation_range	50
width_shift_range	0.2
height_shift_range	0.2
shear_range	0.2
horizontal_flip	True
zoom_range	0.3

Tabla 8 Parametros para ImageDataGenerator

Se han realizado transformaciones aleatorias y operaciones de normalización con esta clase durante el entrenamiento. El número y la dimensión de las capas se redujeron para evitar que la red se atasque en los mínimos locales. La Tabla 9 muestra la arquitectura de red optimizada del modelo propuesto. En este caso, la red se entrenó durante 20 épocas, con una tasa de aprendizaje de 0,001 utilizando un optimizador de Adam. Además, el tamaño del lote y el paso por época se establecieron en 30 y 100, respectivamente.

Layer (type)	Output shape	Param#
Flatten_1 (Flatten)	(None, 8192)	0
Dense_1 (Dense)	(None, 8)	65544
Dropout_1 (Dropout)	(None, 8)	0
Dense_2 (Dense)	(None, 1)	0

Tabla 9 Arquitectura del modelo DNN con TL y DA

Durante el entrenamiento, el *accuracy* (Fig. 72) logró mejores resultados y una curva de progreso más suave dada la mayor disponibilidad de imágenes que proporciona este método. El entrenamiento llegó a un valor cercano al 94%, mientras que la validación en la época 8 se estabilizó en un 75%, un poco mejor que el caso sin Data Augmentation.

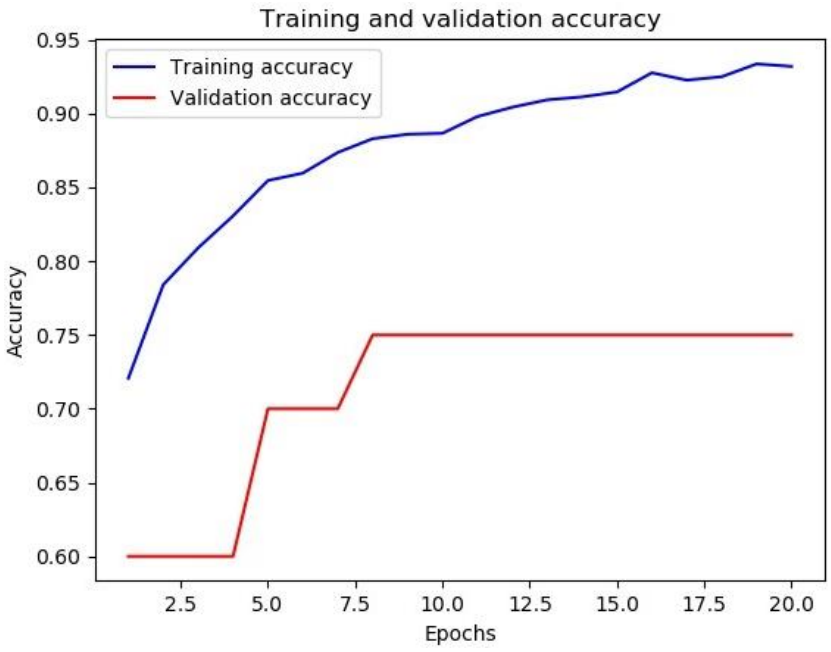


Fig. 72 Gráfica accuracy de entrenamiento y validación con VGG16 con DA

El entrenamiento de este sistema demuestra una leve mejoría en su error tanto de entrenamiento como de validación (Fig. 73). Y al igual que con el caso anterior, aunque pueda plantear dudas el error de validación, el *accuracy* del mismo demuestra estar bien entrenado, además de que, el testeo, con la base de datos para tal fin, demostró una mejora mayor de este sistema respecto al anterior.

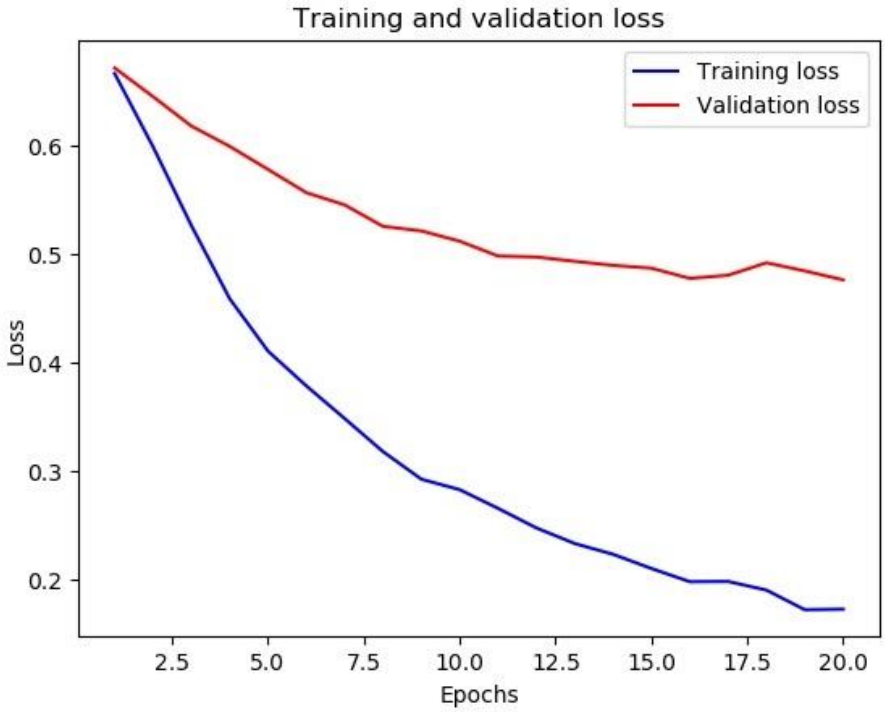


Fig. 73 Gráfica loss de entrenamiento y validación con VGG16 con DA

Para asegurarnos de que el sistema entrenado y sus resultados no se deban al azar, relacionado con las imágenes generadas para el entrenamiento, se realizó este entrenamiento y su testeo treinta veces. Con esto nos aseguramos que la base de datos y el tipo de procedimiento para el entrenamiento es correcto y repetible en el tiempo.

5.3.2. App "Peligroso?"

5.3.2.1. Descripción

Esta aplicación fue desarrollada con el propósito de poder proporcionar una herramienta de prevención de incidentes con escorpiones de importancia sanitaria. Dicha aplicación consta de un clasificador que diferencia tres clases, escorpión peligroso (*Tityus*), escorpión no peligroso (*Bothriurus*), y ausencia de ambos (un ambiente sin presencia cercana del mismo).

5.3.2.2. Modelo

Para el entrenamiento se utilizó el sistema de entrenamiento online "Teachable Machine" proporcionado por Google. Este sistema de entrenamiento, a diferencia del

sistema de clasificación previamente entrenado para clasificar estos escorpiones, se basa en el modelo pre entrenado MobileNet. Para el entrenamiento se utilizaron 105 imágenes de *Tityus*, 113 de *Bothriurus*, y 60 de ausencia de escorpiones. Estas imágenes fueron cargadas a Roboflow y se generó un incremento de base de datos obteniéndose las siguientes cantidades, 315 de *Tityus*, 339 de *Bothriurus*, y 180 de ausencia de escorpiones.

El entrenamiento constó de 200 épocas, con tamaño de lote (batch size) de 512, y una tasa de aprendizaje de 0,001. Durante este entrenamiento se logró la mejora esperada en el *accuracy* (Fig. 74) mostrando una correlación alta entre el entrenamiento y la validación del mismo. También se puede observar el amesetamiento de ambas curvas en un valor alto, superior al 90%, por lo que nuestro sistema no está sub entrenado y dado que no decrecen sus valores tampoco está sobre entrenado.

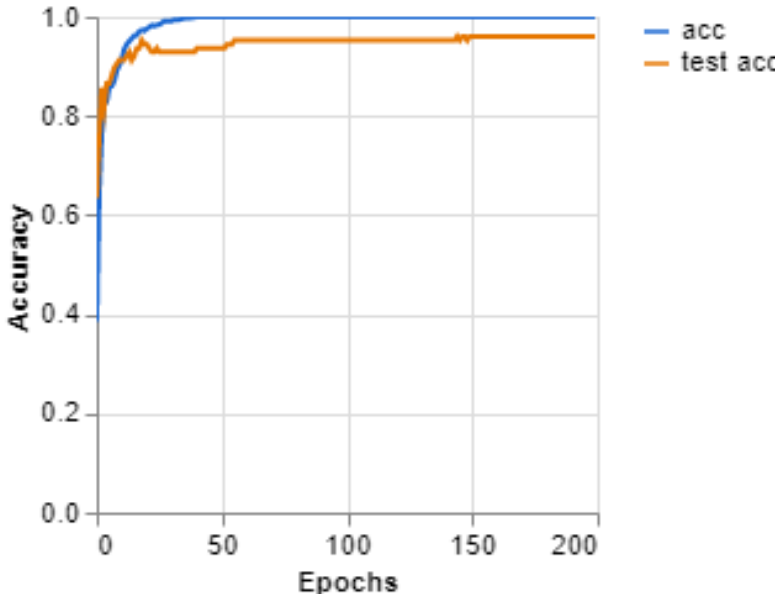


Fig. 74 Gráfica accuracy de entrenamiento y validación con MobileNetV2

En referencia a la pérdida, obtuvimos la disminución esperada durante el entrenamiento (Fig. 75), llegando a valores menores al 20% en caso de la validación y a 0% en caso del entrenamiento. El sistema no ha sido sub entrenado, e incluso se podría notar un inicio de sobre entrenamiento en caso de la validación que se incrementó levemente.

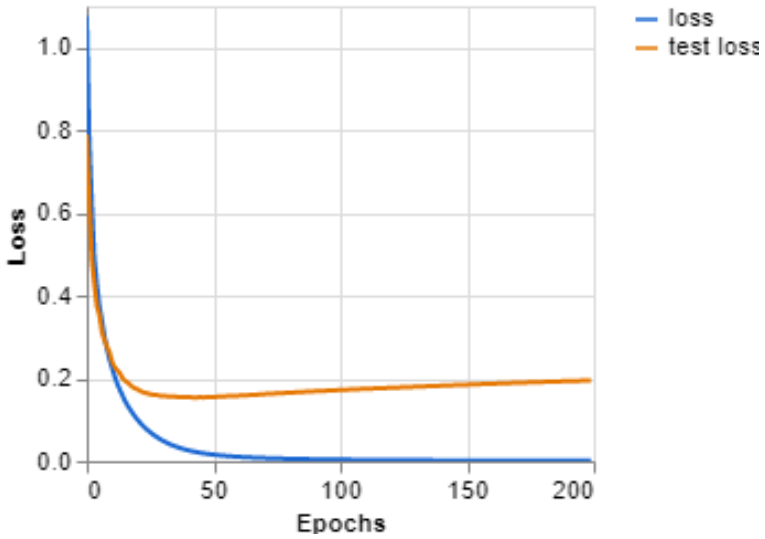


Fig. 75 Gráfica loss de entrenamiento y validación con MobileNetV2

Como se puede observar se logró un entrenamiento con una exactitud y pérdida aceptables para lograr una buena clasificación. En el Capítulo 6 se analizarán los resultados de testeo de este sistema.

5.3.2.3. Implementación

Para hacer posible el funcionamiento del modelo entrenado con Teachable Machine dentro de la aplicación ejemplo, en primer lugar, hay que exportar el modelo en formato TensorFLow Lite cuantificado, luego hay que copiar el modelo dentro de la carpeta “assets” en el directorio del proyecto de la app. Además, se deben copiar los nombres de las clases o labels que, cuando exportamos el proyecto es proporcionado, pero en otro formato al necesario. El proporcionado tiene múltiples características de cada clase, y solo se necesita un archivo que proporcione el nombre de la clase por línea, sin otra información adicional.

Copiados estos dos archivos en el directorio establecido, se procedió a modificar el código de la aplicación a fin de que identifique estos archivos y los reconozca como un modelo a utilizar. A continuación, se muestran las líneas de código necesarias a agregar.

```
@Override
protected String getModelPath() {
  return "converted_tflite_quantized/model.tflite";
}

@Override
protected String getLabelPath() {
  return "converted_tflite_quantized/labels.txt";
}
```

Una vez obtenidos los resultados deseados y comprobado el correcto funcionamiento de la aplicación (Capítulo 6 Resultados), se procedió a personalizarla para que se adapte a nuestras necesidades, tanto estéticas como funcionales.

En cuanto a las modificaciones estéticas se encuentran las modificaciones a íconos y de la barra ilustrativa dentro de la aplicación. En la Fig. 76 se muestra el ícono desarrollado para esta aplicación, se puede ver que consta de un escorpión con el logo de TensorFlow.

Fig. 76 Ícono de la App de clasificación

La barra ilustrativa dentro de la aplicación se encuentra por encima de la visualización de la cámara del celular, y combina el ícono de la app con una oración descriptiva de lo que realiza la aplicación, esta barra se muestra en la Fig. 77.

Fig. 77 Barra de diseño para App de clasificación

En lo que respecta a funcionalidad, la aplicación ejemplo, por defecto tiene un selector de modelo entrenado para utilizar en la clasificación, lo cual en este caso carece de sentido dado que poseemos un solo modelo entrenado, por lo que se lo quitó para evitar inconvenientes en la operatividad del sistema (se estableció dentro del código a nuestro modelo como única opción).

Por último, se buscó un nombre corto y representativo para la aplicación, y se decidió por el nombre "¿Peligroso?", el cual representa de la mejor manera la funcionalidad de esta aplicación que nos clarifica si el escorpión que hemos encontrado es, en efecto, peligroso para el ser humano.

Se obtuvo así un sistema funcional que brinda seguridad sanitaria ante el encuentro con un escorpión. En la Fig. 78 podemos apreciar un ejemplo de clasificación de un escorpión del género *Tityus*.

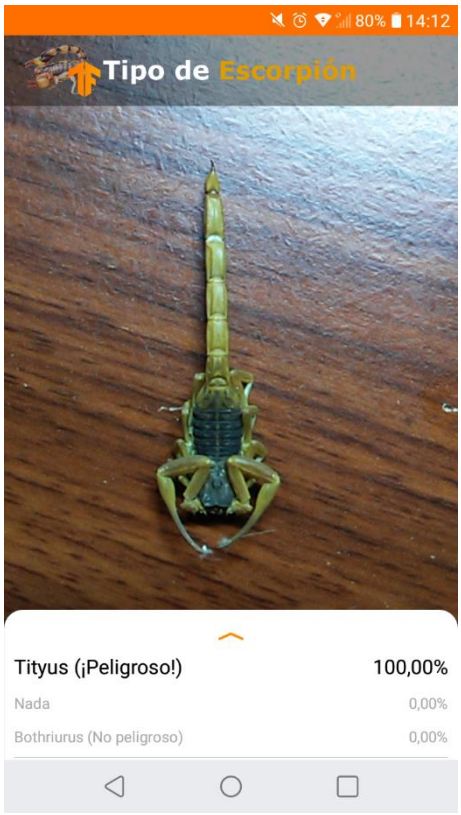


Fig. 78 Clasificación de Tityus con App de clasificación

Se puede observar que la clasificación se muestra en la parte inferior de la pantalla, demarcando claramente la peligrosidad del ejemplar en imagen, con un porcentaje de certeza del 100%. Esto no quiere decir que el sistema tenga dicha certeza, sino que, para esta imagen el sistema dice que su clasificación no puede ser otra más que *Tityus*. Debajo se pueden ver las otras dos opciones del clasificador con sus respectivas probabilidades de corresponder a la imagen analizada, en este caso ambas con 0%. El desempeño del sistema se muestra en el Capítulo 6.

5.4. Sistema analógico de detección de escorpiones

5.4.1. Sistema analógico de detección con sensor de distancia

El sistema analógico de detección de escorpiones previamente desarrollado [1] utiliza la característica fluorescente de los mismos para lograr así su detección, a partir del uso de un sensor de color y un microcontrolador [121]. Sin embargo, este sistema tuvo el inconveniente de detectar numerosos falsos positivos, debido a que las amplitudes de la salida del sensor de color, que abarcan la detección del color buscado,

podían predecir también la presencia de otro color a una distancia diferente frente al sensor.

Con el objetivo de mejorar la eficiencia de este sistema se decidió incorporar al mismo un sensor de distancia por ultrasonido que asegure que la respuesta brindada por el sensor de color se corresponda a un escorpión a una distancia específica.

En la Fig. 79 se puede observar el comportamiento del sensor respecto a su salida en relación con la distancia a la que se encuentra del sensor.

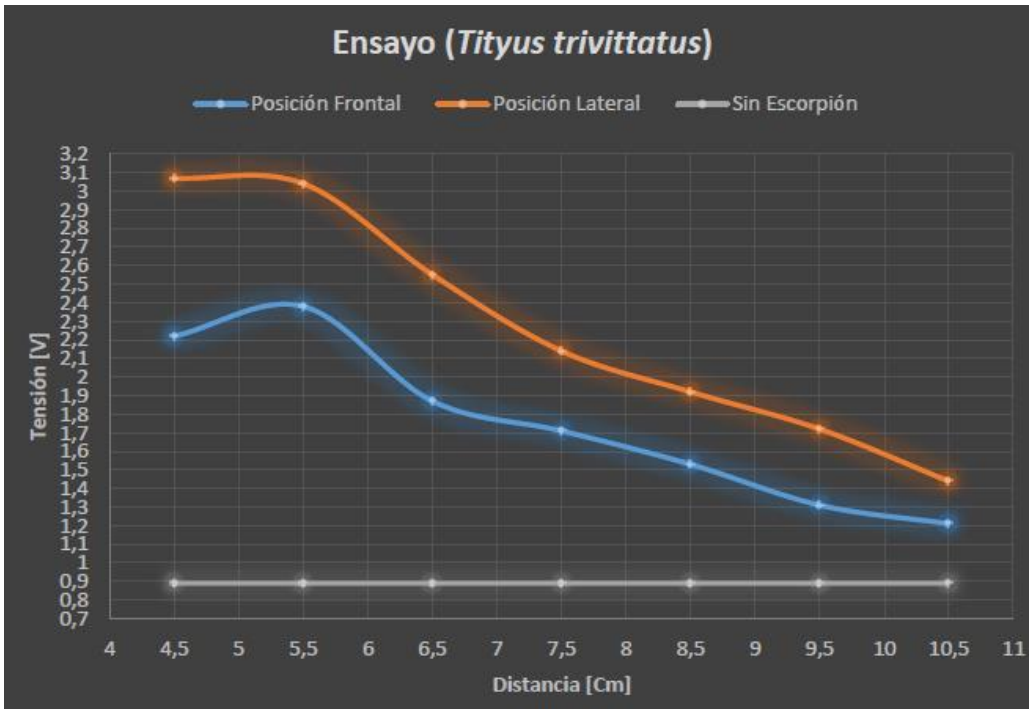


Fig. 79 Valores de tensión del sensor TSL-257 ante escorpión fluoresciendo según la distancia

Para realizar la correcta detección de la fluorescencia de los escorpiones, y no una falsa detección por otro insecto o arácnido cuyo color a una distancia diferente proporcione la misma respuesta en el sensor de color, se utilizó un sistema de doble condición (distancia/color) para validar dicha detección. Para saber qué valores se deben obtener para anunciar la detección se procedió a analizar la Fig. 80 la cual demuestra el rango de valores proporcionados por el sensor de color ante diferentes distancias del escorpión fluoresciendo. Usando un sistema de cotas en la distancia y las tensiones brindadas por ambos sensores (color y distancia) se pudo establecer las condiciones para asegurar la presencia del escorpión, y así mejorar el desempeño del sistema previamente desarrollado.

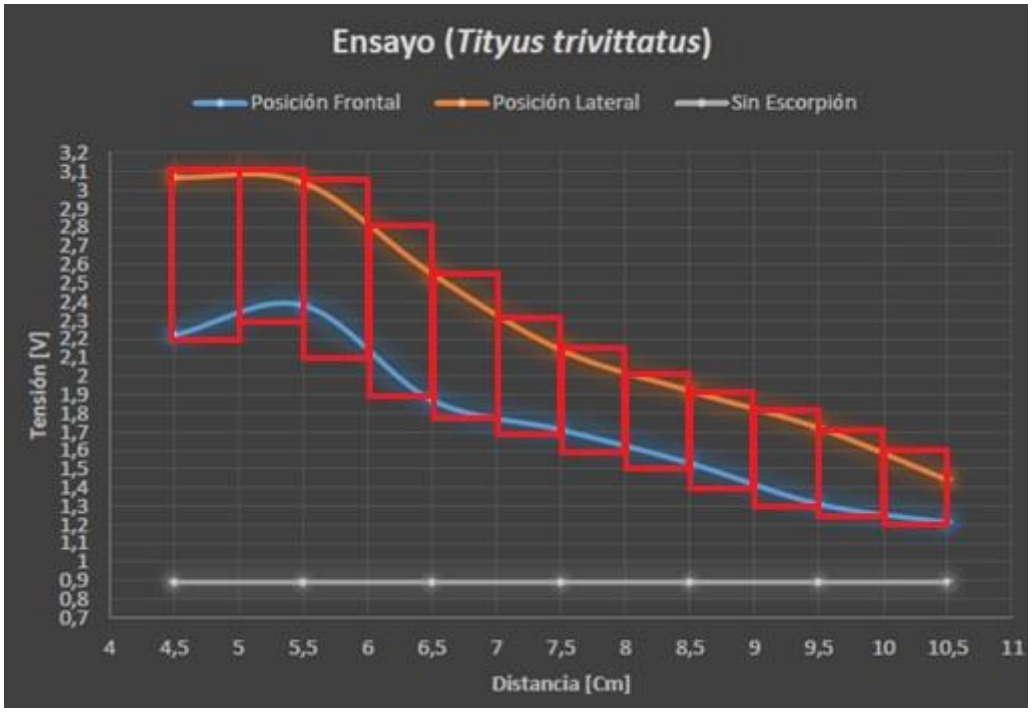


Fig. 80 Representación de las sub secciones de detección posibilitadas por el sensor de distancia

En la Fig. 81 se puede observar el diagrama de flujo del sistema resultante, con la segunda toma de decisión incorporada, relacionada con si el valor de color buscado se encuentra a la distancia que corresponda a un escorpión. En caso de ser así, se confirma la detección, sino el sistema se mantiene a la espera de ésta.

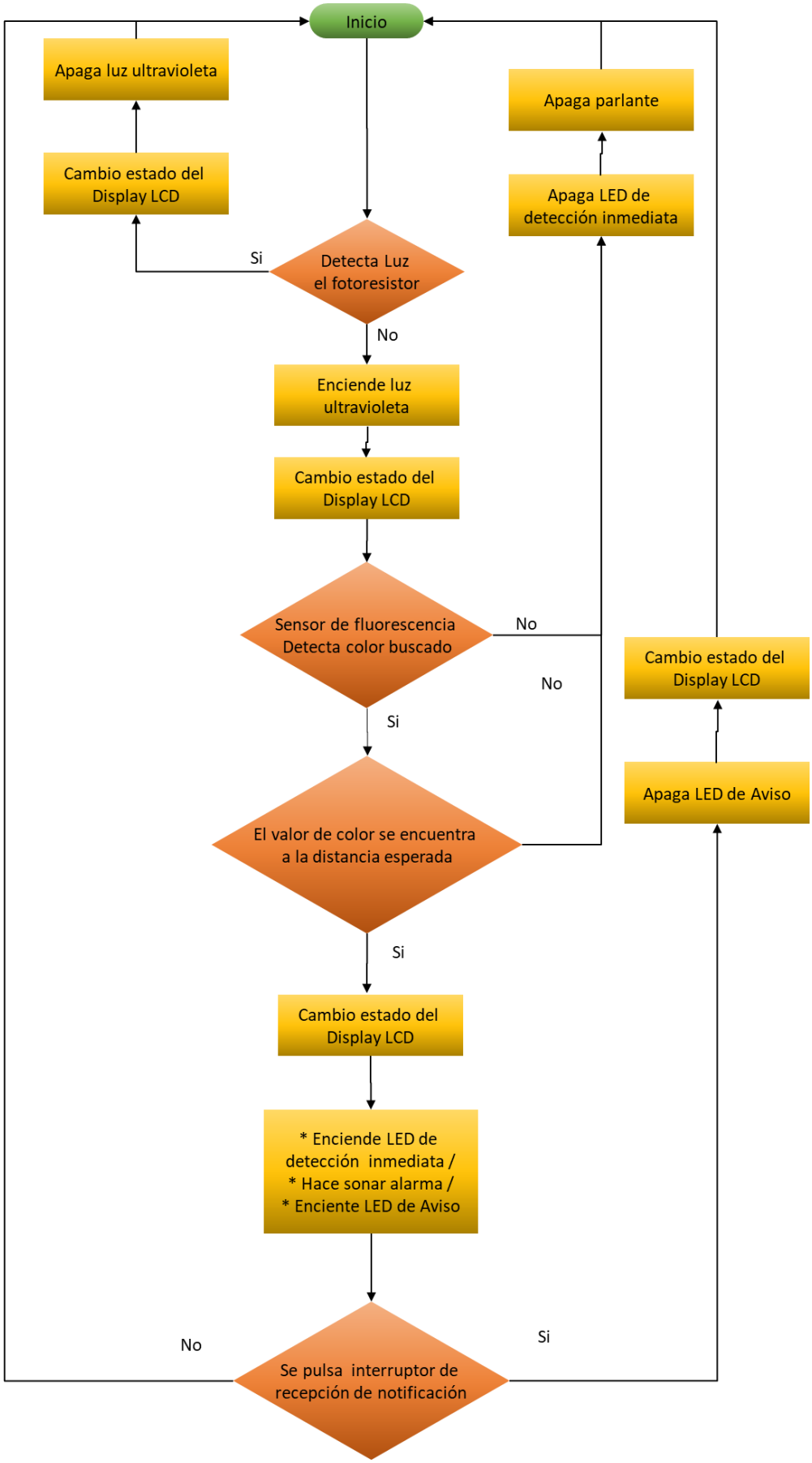


Fig. 81 Diagrama de flujo de sistema de detección con sensor de color y distancia

El funcionamiento de este sistema de alarma es el siguiente. En primer lugar, el sistema verifica la ausencia de luz natural, dado que solo en dichas condiciones va a poder utilizar la característica fluorescente de los escorpiones para su detección. En el caso de ausencia de luz natural, se enciende la luz ultravioleta e inicia la búsqueda del color específico de la fluorescencia de los escorpiones. Si encuentra un valor entre los posibles, se verifica que se encuentra a la distancia correspondiente para ese valor. Finalmente, en el caso que todo lo anterior se cumpla, se dará la alarma de la presencia de un escorpión delante del sistema.

Como se mostró en el diagrama de flujo, el sistema cuenta también con una pantalla LCD como interfaz de usuario, así se puede anunciar, no solo la detección, sino el estado en el que se encuentra el sistema, como por ejemplo, la inactividad por presencia de luz natural detectada con el fotorresistor. El sistema utiliza además tres diodos LEDs como indicadores para demarcar el estado del sistema, la detección en tiempo real, y la notificación de detección pasada, y utiliza dos pulsadores, uno para encendido y otro para recepción de la notificación de detección.

En la Fig. 82 se puede apreciar una simulación en Proteus del sistema funcionando con todos los componentes. Estos son la fuente de tensión (abajo a la izquierda); el fotorresistor con un amplificador en retroalimentación para generar una señal digital (abajo al centro); salida con amplificador para señal sonora (abajo a la derecha); el microcontrolador (en el centro); a su izquierda, el cristal externo y un potenciómetro que simula la señal analógica del sensor de color; a la derecha del microcontrolador se encuentran los tres LEDs de indicación (estado, detección pasada y presente) y el display LCD. Por último, se encuentra el sensor de distancia, emulado en el centro superior de la imagen, con una resistencia regulable que le indica la distancia), y en la parte superior derecha el sistema de accionamiento de encendido de la luz UV mediante un transistor conectado a un relé. Como se puede apreciar, en la simulación el sistema se encuentra en estado de detección, demostrado mediante el display y las señales lumínicas.

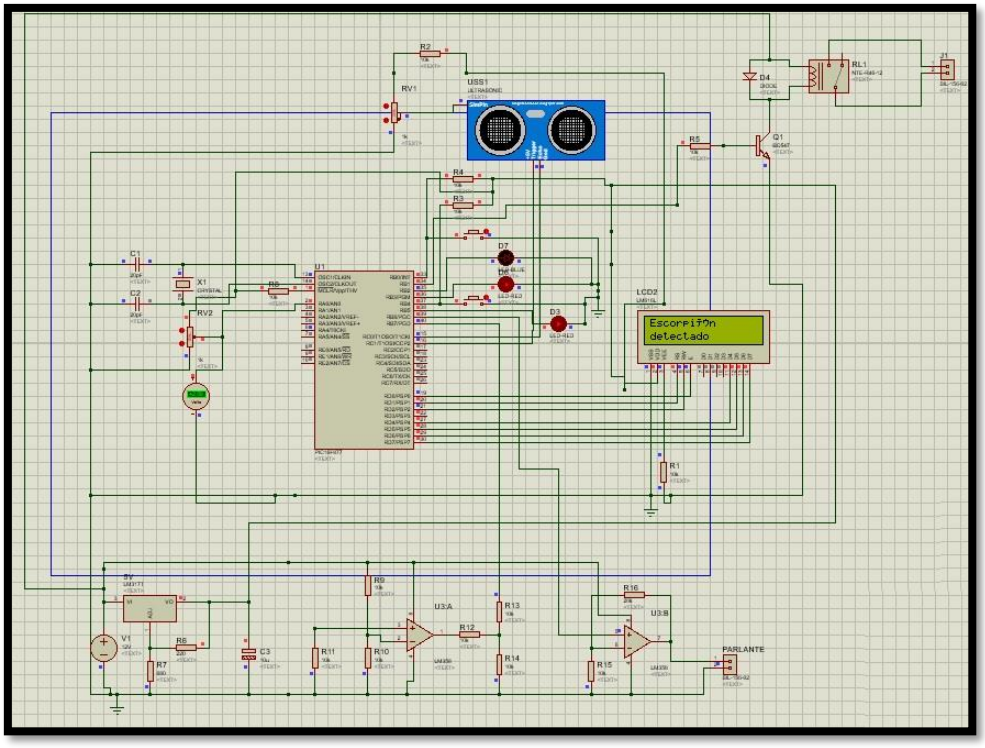


Fig. 82 Esquema de conexión en simulación de sistema de detección con sensor de color y distancia

En esta ocasión el microcontrolador utilizado fue el 16F877 [122], a diferencia del 16F819 [123], debido a la cantidad de pines disponibles para su uso. El proyecto original había demandado casi toda la funcionalidad del microcontrolador, por lo que no era posible incorporar el sensor de distancia y el display para este nuevo proyecto.

5.4.2. Instalación sistema alarma en Ministerio de Infraestructura

Dada la necesidad presente de detectar a los escorpiones peligrosos antes que puedan causar daño, o incluso la muerte, a personas de riesgo, en el año 2019 el Ministerio de Infraestructura de la Provincia de Buenos Aires decidió poner a prueba los sistemas que se estaban desarrollando para esta Tesis. El lugar elegido fue el Jardín de Infantes perteneciente a dicho Ministerio, el cual se ubica en la ciudad de La Plata en calle 8 entre 58 y 59. Gracias a este lugar de pruebas, donde frecuentemente se encontraban escorpiones del género *Tityus*, se pudo perfeccionar los diferentes sistemas puestos a prueba allí.

En una primera instancia se puso a prueba el sistema de detección de fluorescencia, el cual fue programado para que funcione de manera nocturna, usando las funciones de la librería “time.h”, para ser utilizado en ausencia de luz natural y con la iluminación ultravioleta necesaria para hacer fluorescer al escorpión. El sistema es controlado

mediante un software de escritorio remoto, en primera instancia TeamViewer y luego se utilizó Google Desktop.

Sin embargo, este sistema presentó un inconveniente al funcionar varias horas corridas, el uso de RAM de la computadora se saturaba y el equipo quedaba inutilizable hasta ser reiniciado. Se hicieron todas las modificaciones necesarias para liberar, en tiempo real, la RAM que ya no se utilizaría en el programa, pero no se logró agregar más de un par de horas al funcionamiento del mismo. Después de mucho análisis se llegó a la conclusión de que las librerías de OpenCV en lenguaje C no estaban liberando el uso de memoria de variables dentro de funciones que funcionaban en tiempo real.

Antes de descartar por completo el programa se decidió buscar alternativas de software, en nuestro caso CleanMem, para solucionar este inconveniente. El principal interés era garantizar los recursos necesarios para controlar la PC de manera remota y tener conocimiento del estado de funcionamiento del programa. Este programa es ideal dado que permite dar prioridad de uso de RAM a los procesos que uno quiera, e incluso crear un orden de prioridad para destacar, por sobre los demás, el programa de detección y el programa de control, dando prioridad a este último sobre el primero.

Aunque se haya solucionado este problema se decidió exportar el sistema de detección creado al lenguaje de programación Python, dado que las demás aplicaciones desarrolladas se implementaron en este lenguaje. Es por esta razón que se pudo realizar un híbrido entre detección de fluorescencia y de objetos. Otra ventaja del uso de Python es que, su compilación para exportación alberga todas las librerías necesarias para su correcto funcionamiento, a diferencia de la compilación en C con el IDE Code::Blocks que generaba la necesidad de tener que instalar todas las librerías en el equipo a utilizar, con todas las complicaciones que eso implica.

Para el sistema de alarma se utilizaron dos cámaras conectadas a un mismo equipo, ambas cámaras de calidad VGA con su correspondiente lámpara UV. Las zonas cubiertas comprendieron una cocina y una sala común.

Este sistema fue abandonado por diversas razones, todas ajenas a su desempeño, la principal fueron los problemas de iluminación, en el caso de una de las cámaras, la presencia de un guardia de seguridad de guardia nocturna en la sala común, lo cual imposibilitaba la detección dado que no se puede iluminar con UV una habitación donde hay una persona durante todo su período de funcionamiento. Y en el caso de la segunda cámara, la ubicación de la misma, se encontraba muy lejos del lugar que enfocaba, una detección a esa distancia (que no llegaba a ser correctamente iluminada por la luz UV) resultaba imposible, además de la iluminación artificial, que en varias oportunidades era dejada accidentalmente prendida, lo que anulaba toda posibilidad de detección.

En un principio se iban a relocalizar las cámaras, pero los lugares de interés eran éstos dado que la mayor parte de los escorpiones fueron encontrados allí. Es por esta razón, que se decidió realizar las pruebas correspondientes con el sistema de detección por clasificador en cascada (Haar). Este sistema funcionó correctamente durante su etapa de ensayo, aunque afortunadamente para las personas que se encuentran allí, y

no para las pruebas, nunca se detectó un escorpión (TP: verdadero positivo). Lo que se puede rescatar es que, durante su período de funcionamiento nunca presentó un falso negativo (FN), es decir, que haya habido un escorpión y el sistema no lo haya detectado, lo que hubiese sido peligroso a nivel de seguridad sanitaria. Sin embargo, este sistema, como era de esperarse por la cantidad de movimiento que presenta este edificio durante el día, si presentó falsos positivos (FP), es decir, el aviso de detección de un objeto que no es un escorpión, que nunca ha superado los diez casos por día, lo cual consideramos aceptable dado que se apunta a un sistema de seguridad sanitaria, es decir que pretendemos tener un Recall cercano a 100%, a costa de reducir la Precisión (dentro de niveles aceptables).

Finalmente se implementó el sistema de detección con el modelo YOLOv3, el cual en una primera instancia presentó muchos inconvenientes con FP, la causa principal era el alto nivel de ruido de la imagen. Se solucionó este problema utilizando una función reductora de ruido gaussiano. Una vez funcionando, se lo testeó durante un periodo de tiempo similar al clasificador en cascada (de uno a dos meses), que con la misma suerte que el anterior no obtuvo TP ni FN, mientras si obtuvo FP, pero con menos frecuencia (en el orden de un caso cada dos días). Cabe destacar que han sucedido falsos positivos que llegaron a plantear dudas de si realmente el sistema se estaba equivocando debido a la mala resolución de la imagen, como es el caso de la Fig. 83, por lo que se puede concluir que en esos casos hasta una persona podría haberse confundido.

Fig. 83 Falsa detección en ambiente no controlado

Aunque claramente lo podemos considerar una mancha del suelo, dada la baja tasa de FP y a la resolución se llegó a discutir su origen. Cabe aclarar que la imagen ha sido recortada y ampliada para poder apreciarla mejor.

Uno podría pensar que dada la ausencia de TP estos ensayos no tienen relevancia para el análisis de desempeño de los sistemas entrenados, pero no es así, el hecho de saber la frecuencia de FP, combinado con los ensayos previamente realizados en ambientes controlados, nos dan un panorama bastante certero de qué sistema de

detección va a ser de mayor utilidad. En principio se puede pensar que el sistema más conveniente será YOLOv3, pero hay que ver los resultados de ambos sistemas, presentados en el próximo Capítulo para sacar conclusiones.

Con el propósito de escalar la implementación del sistema de detección dentro de las instalaciones del Jardín y del Ministerio de Infraestructura en sí, se nos propuso poder utilizar cámaras IP [124] para la detección de los escorpiones interconectando a la red LAN del Ministerio a fin de poder utilizar el sistema en cualquier sitio dentro de los alcances de la red, sin inconvenientes de conectividad de la unidad de procesamiento de las imágenes provistas por dichas cámaras.

Además, se albergó la posibilidad de utilizar el servidor del Ministerio, que no es requerido durante la noche, para realizar el procesamiento, y con su poder de cómputo, superior a una computadora como la utilizada, ser capaz de procesar múltiples cámaras en simultaneo, tema limitado a un máximo de dos cámaras por equipo con la computadora que se nos proporcionó para tal fin.

Las cámaras de IP que posee el Ministerio corresponden a cámaras viejas, reemplazadas por una nueva generación. Dada la fácil disponibilidad de éstas y la ausencia de gastos para su uso, nos proveyeron de una cámara para que adaptemos el sistema, en lugar de las Webcams conectadas por USB.

El uso de este tipo de cámara en un principio generó ciertos inconvenientes por su incompatibilidad a las nuevas versiones de explorador de internet para intentar acceder a su configuración y así reestablecer su dirección IP a fin de que se adapte a nuestra red de prueba. Para solucionarlo se debió recurrir al plugin “IE Tab” [125], el cual una vez instalado en Google Chrome permite colocar la IP de la cámara y entrar a su configuración, usuario y clave de por medio (admin y 12.345 por defecto).

Una vez resuelto esto, debimos modificar nuestro código Python, el cual ya no podía acceder a la imagen a través de “cv2.VideoCapture(0)” sino que ahora se debía usar un código que contemplase la dirección de red en la que se encuentra la cámara junto con su usuario y contraseña de acceso. La creación del objeto imagen que llamará al método “read()” para obtener la imagen quedaría de la siguiente manera.

```
cap = cv2.VideoCapture("rtsp://admin:12345@IP_seleccionada/Streaming
/channels/001/?transportmode=unicast")
```

En rojo usuario y contraseña, y en azul dónde hay que escribir la dirección LAN de la cámara.

Dada las particularidades del año 2020, como consecuencia de la pandemia COVID-19 y de la situación impuesta por ASPO y DISPO, los ensayos que se estaban llevando a cabo se vieron interrumpidos en reiteradas ocasiones debido, en un principio, a la imposibilidad de poder asistir al establecimiento para hacer modificaciones o reiniciar el equipo en caso de que se trabase por la ejecución de uno de los ensayos que se

estaban probando. Sin embargo, a pesar de ello, se pudieron llevar a cabo varios ensayos con diferentes mecanismos, lo cual nos permitió establecer cuál de los sistemas presentaba menor tasa de FP.

Capítulo 6: Resultados

6.1. Descripción

En esta sección se presentarán los resultados de los ensayos realizados con los diferentes métodos presentados. Al igual que durante el Capítulo 5, aquí también se divide este Capítulo en dos grupos principales: segmentación para la detección de objetos y clasificación de imágenes.

En primer lugar, se presentarán los resultados obtenidos con los sistemas de detección del objeto escorpión basados en el clasificador en cascada, los modelos YOLOv3, YOLOv4, y MobileNetV2, y la detección por fluorescencia. En particular, en lo que respecta a la detección de objetos con clasificador en cascada y con el modelo YOLOv3, si bien se calcularon sus respectivos Recall de detección, el estudio se centró en evaluar cómo la confirmación de detección por fluorescencia mejora sus funcionamientos reduciendo sus tasas de FP.

En segunda instancia, se presentarán los resultados obtenidos con los sistemas de clasificación de escorpiones. Por un lado, se presenta un análisis comparativo de los sistemas con LBPH y con VGG16 (con y sin uso de Data Augmentation), y por otro lado, se presenta el análisis de la aplicación de teléfonos inteligentes desarrollada con el modelo MobileNetV2.

6.2. Detección de objetos

6.2.1. Caso 1: Sistema de confirmación de detección mediante fluorescencia

En primer lugar, vamos a analizar el desempeño de los sistemas de detección a partir de las imágenes con o sin fluorescencia. Este análisis se centrará en un video de 5 cuadros por segundo, con un total de 45 segundos, y la métrica analizada será el “Recall”.

Dado que más de una detección por segundo no tiene sentido práctico a los fines de tiempo de reacción ante la presencia de un escorpión, nos conformaremos con que se dé una detección por segundo como mínimo y calculamos su desempeño en base a esto.

Con esta consideración agrupamos los datos de detección obtenidos por cuadro en grupos de 5 (1 segundo) y analizamos la proporción de cantidad de segundos con detección frente al total, obteniendo así un recall 73,33% (33 segundos con detección) para el sistema Haar, y un 82,22% (37 segundos con detección) para el sistema YOLOv3. Cabe recordar que ambos sistemas fueron entrenados con la misma base de datos. El

modelo YOLO demuestra su ventaja ante situaciones adversas de iluminación y orientación de cámara.

Por supuesto, al ser sistemas diferentes, los cuadros detectados del video utilizado no son los mismos, como tampoco lo son las coordenadas de lo detectado en la imagen, es por ello que es válido realizar nuevamente el análisis de funcionamiento de la confirmación de detección mediante la fluorescencia.

Recordemos que puede haber más de una detección por segundo, por lo que, dado el objetivo de este trabajo, vamos a analizar todos los cuadros en los que hubo detección. Dicho esto, se analizaron 348 cuadros del sistema Haar, y 262 cuadros del sistema YOLOv3.

Luego se realizó el análisis de desempeño del sistema de confirmación de detección mediante fluorescencia. Para ello se forzó la generación de FP en el sistema, esto se logró utilizando un escorpión real sin fluorescencia en un ambiente diurno bien iluminado. Recordemos que la confirmación por fluorescencia solo se puede dar en ausencia de luz natural, por lo que el sistema sólo deberá funcionar de noche o en un sitio oscuro y correctamente iluminado con UV.

Para estudiar la efectividad de este método de confirmación se analizaron las detecciones realizadas en un video, tanto con presencia de fluorescencia como con luz natural. De cada detección del objeto escorpión realizada correctamente, se analizó si se dio, o no, la correspondiente detección de fluorescencia para confirmación. La Tabla 10 muestras las métricas obtenidas de los ensayos con ambos sistemas.

MÉTODO	A	P	R	F1
HAAR + UV	0.99	0.99	1.00	0.99
YOLOV3 + UV	0.98	0.96	1.00	0.98

Tabla 10 Métricas de confirmación por fluorescencia de ambos métodos

Claramente los sistemas son excepcionalmente buenos, no solo su *accuracy* y precisión están por encima del 95%, sino que el recall en ambos es del 100%. Esto nos deja libres de falsos negativos, lo que nos da un sistema sumamente seguro a nivel sanitario, dado que nunca va a haber un caso en el que haya un escorpión en presencia de UV, que haya sido detectada su forma, en la cual no se detecte su fluorescencia.

Para entender mejor qué implica la reducción de FP a partir del uso del sistema de confirmación de detección por fluorescencia, se presentan a continuación los cálculos de mejora logrados con cada método de detección.

Al sistema de detección de objetos Haar lo forzamos a detectar mal, logrando así que haya 174 FP, es decir una precisión y *accuracy* del 50%, claramente porque el sistema por sí solo no es capaz de distinguir sus falsos positivos. Ahora bien, si al usar este sistema de confirmación reducimos los FP a solo dos, logrando mejorar así la

precisión al 98% y el *accuracy* al 99%, el sistema logró una mejora del 98% con el *accuracy* y un 96% con la precisión en su detección, sin alterar su recall, lo cual es sumamente importante para la seguridad sanitaria. Se puede observar la distribución de los casos analizados en la matriz de confusión confeccionada para tal fin (Fig. 84).

De la misma manera, el sistema de detección YOLO demostró mejoras similares al caso anterior. Cabe aclarar que la base de datos es igualmente simétrica, por lo que la mejora parte del 50%. Las mejoras obtenidas son del 96% para el *accuracy* y de un 92% para la precisión. Por lo que, se observa un menor incremento de la precisión y el *accucary*, respecto del sistema anterior, pero se destaca como se mantiene el recall en su 100%, lo que nos demuestra la ausencia de FN, como se puede ver en la Fig. 85 con la matriz de confusión.

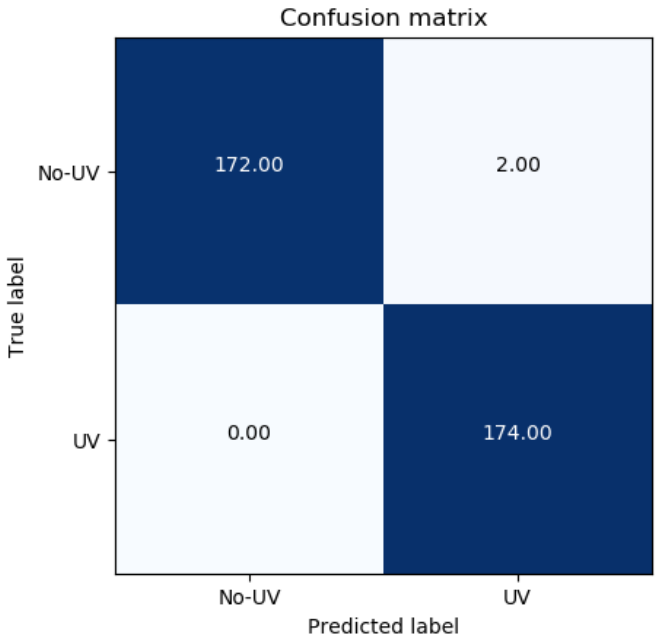


Fig. 84 Matriz de confusión Haar + UV

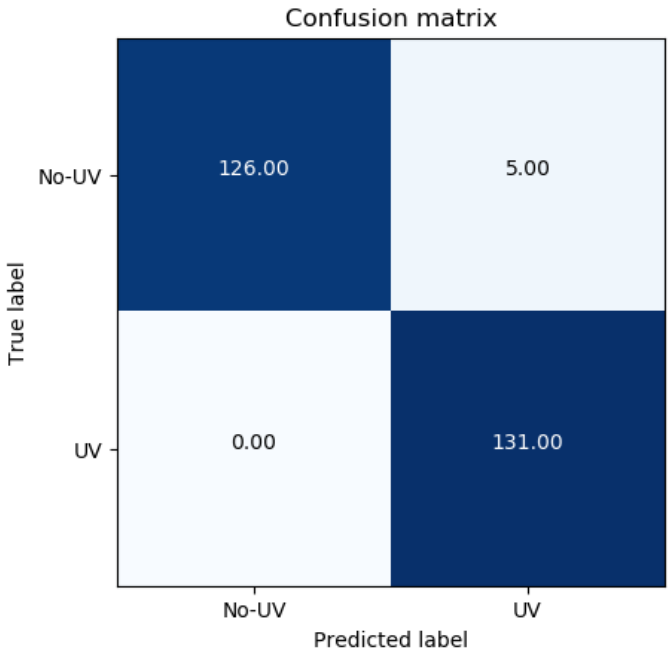


Fig. 85 Matriz de confusión YOLOv3 + UV

Claramente el sistema demuestra un excelente nivel de seguridad sanitaria, dado que, bajo ningún concepto, nos dirá que un escorpión no lo es cuando éste haya sido detectado por su forma y se encuentre en un entorno iluminado por UV.

6.2.2. Caso 2: Comparación entre dos sistemas de detección

En este caso se mostrará la comparación entre los sistemas de detección basados en los modelos YOLOv4 y MobileNetV2. Para analizar el desempeño de ellos, se consideraron 81 imágenes de la base de datos, las cuales no fueron utilizadas en el proceso de entrenamiento. Cabe aclarar que la selección de estas imágenes fue completamente aleatoria y realizada por el algoritmo de distribución de Roboflow.

Otra aclaración relevante que se debe mencionar es la ausencia de imágenes negativas (sin escorpión) para el entrenamiento. Esto se debe a que ambos métodos utilizados contaron con la utilización de algoritmos que generan sus propias imágenes negativas a partir de las secciones, dentro de las imágenes proporcionadas, que no poseen escorpiones. Esto garantiza el equilibrio de la base de datos entre clases. Las imágenes negativas utilizadas para el testeo fueron elegidas al azar de una base de datos propia de falsas detecciones y/o imágenes de entornos donde podría encontrarse el escorpión, incluyendo imágenes de otros arácnidos con los que podrían ser confundidos.

La Fig. 86 muestra las detecciones adecuadas de escorpiones utilizando los modelos YOLOv4 (imagen de la izquierda) y MobileNetV2 (imagen de la derecha). Además, se

puede ver una alta precisión en la detección del escorpión (98,96%) para el modelo MobileNetV2.

Fig. 86 Detección de escorpión con los modelos YOLOv4 (izquierda) y con MobileNetV2 (derecha)

Las matrices de confusión obtenidas durante las pruebas para los modelos YOLOv4 y MobileNetV2 se muestran en Fig. 87 y Fig. 88, respectivamente, donde los ejes verticales corresponden a los datos verdaderos y los ejes horizontales corresponden a las predicciones de los modelos.

La Tabla 11 muestra los valores de exactitud, precisión, Recall y Fmedida calculados mediante las ecuaciones de las métricas presentadas en la Sección 3.10, respectivamente, para ambos modelos considerados en este estudio. Estos resultados muestran que ambos sistemas de detección de objetos pueden detectar escorpiones con éxito. Además, los altos valores de Recall obtenidos indican que existen valores muy bajos de falsos negativos, lo que es fundamental para la seguridad sanitaria. En particular, el Recall del modelo MobileNetV2 (0,97) es mayor que el obtenido por el modelo YOLOv4 (0,90), lo que implica un sistema de detección más seguro.

Method	Accuracy (A)	Precision (P)	Recall (R)	$F_{measure}$
YOLOv4	0.88	0.93	0.90	0.92
MobileNetV2	0.91	0.92	0.97	0.94

Tabla 11 Métricas de los modelos de detección YOLOv4 y MobileNetV2

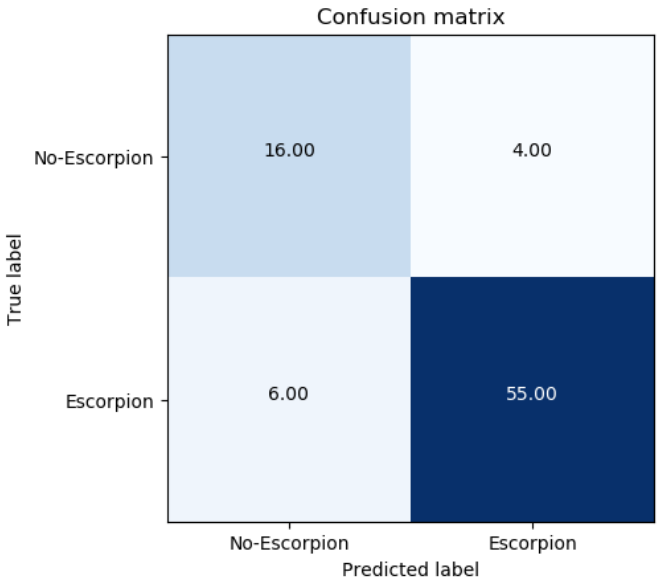


Fig. 87 Matriz de confusión YOLOv4

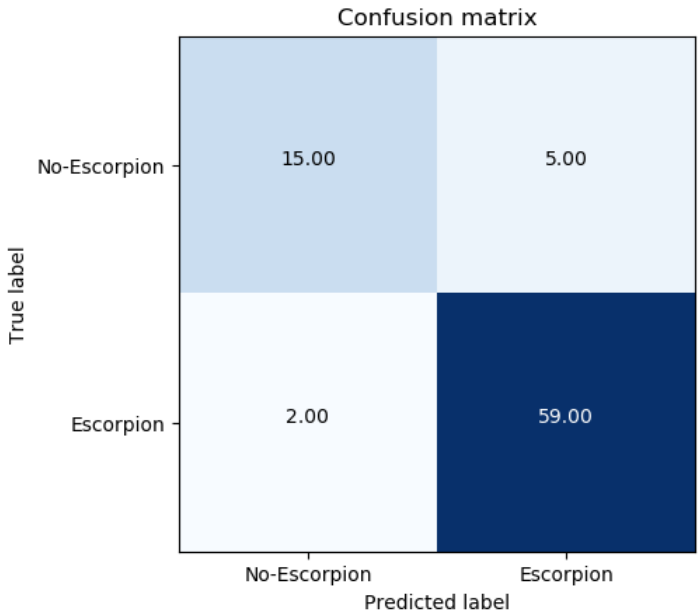


Fig. 88 Matriz de confusión MobileNetV2

La Fig. 89 muestra las curvas ROC para estos sistemas de detección. Se puede observar en esta figura que las áreas bajo las curvas ROC para los modelos YOLOv4 y MobileNetV2 son muy similares, con área de 85% y 86%, respectivamente, lo que implica una muy buena relación de sensibilidad y especificidad.

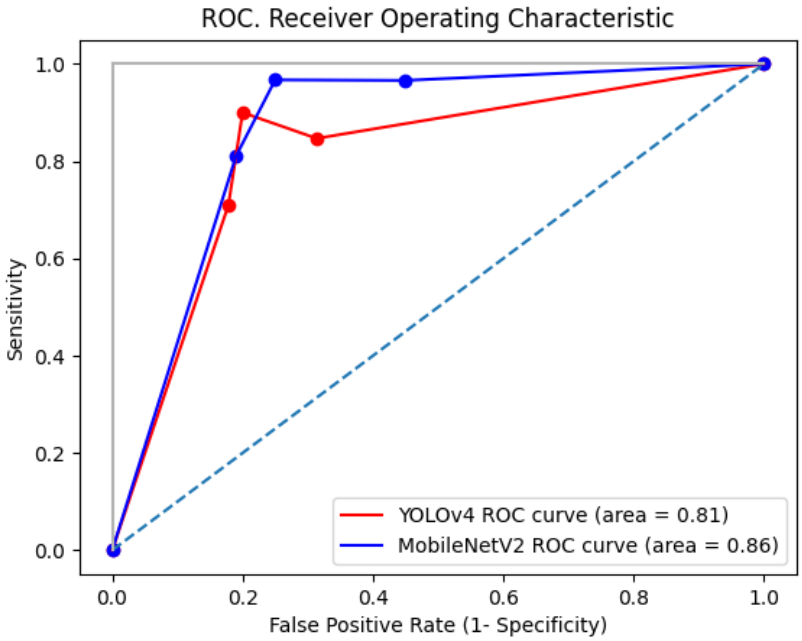


Fig. 89 Curvas comparativas ROC entre YOLOv4 y MobileNetV2

El sistema con el modelo MobileNetV2 resultó mejor al YOLOv4 para la detección de escorpiones. Sin embargo, se podría argumentar que hay diferente cantidad de épocas, incluso de tiempo de entrenamiento, pero si lo pensamos de este modo, incluso los modelos tienen estructuras neuronales muy diferentes; lo que hace válida esta comparación es que, ambos sistemas fueron entrenados hasta llegar al amesetamiento de su mejora, es decir, se los entrenó hasta que se obtuvo su máximo desempeño, sin llegar al sobre entrenamiento.

El modelo MobileNetV2, además de su buen desempeño ante las imágenes de testeo, ha demostrado una excelente capacidad de respuesta para la correcta detección de escorpiones situados en un ambiente no controlado, es decir, en múltiples posiciones no presentes en la base de datos original, con diversos objetos que pudiesen dificultar una clara detección (Fig. 90). En tal situación se observó una baja tasa de falsos positivos, los cuales son esperables en un ambiente muy diverso.

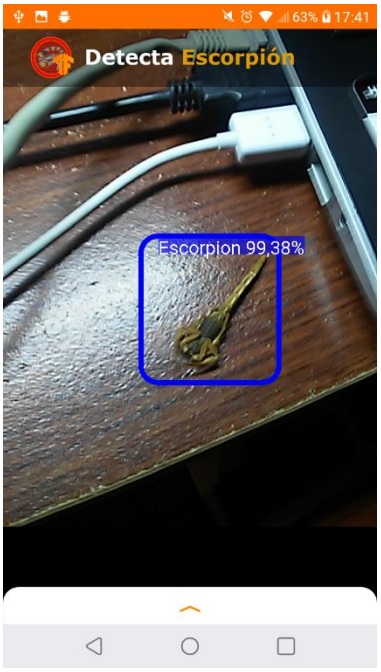


Fig. 90 Detección con la App "Detecta Escorpión"

Se pueden observar las diferencias dentro de la imagen, donde se cuenta con una parte totalmente a oscuras, otra con cables con sombras que se entre cruzan, brillo excesivo sobre la madera, incluso cerca del escorpión que se encuentra de costado. Sin importar lo anterior el sistema no solo detecta el escorpión correctamente, sino que no hace ninguna detección errónea en las zonas que podrían resultarle confusas al sistema dado que nunca se le presentaron imágenes parecidas en su entrenamiento.

Además de lo ante dicho, otra prueba de la excelente capacidad de respuesta de este sistema, es que no solo puede detectar correctamente en diferentes entornos, sino que además puede realizar múltiples detecciones de manera continua sin inconvenientes, como se muestra en la Fig. 91. En esta imagen, se puede observar la presencia de tres escorpiones de dos géneros diferentes detectados correctamente como escorpiones.

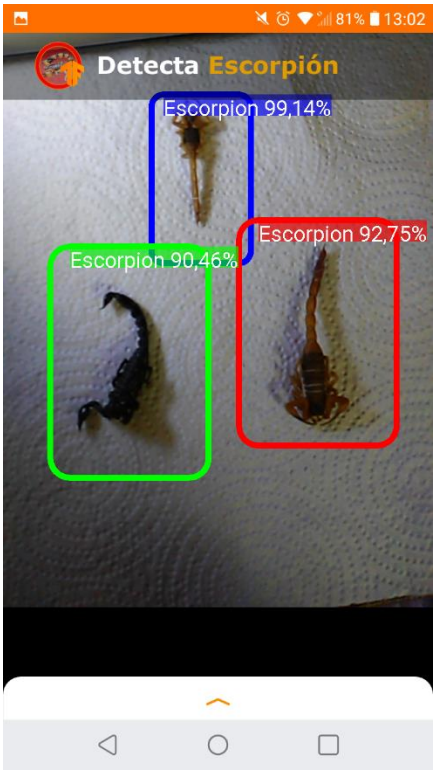


Fig. 91 Múltiples detecciones con la App "Detecta Escorpión"

Desde luego, hay situaciones que pueden confundir al detector, como bien se muestra en la Fig. 92. En dicha figura se muestra un pseudoescorpión (conocido como falso escorpión), el cual es detectado erróneamente como un escorpión con una certeza superior al 90%. Los pseudoescorpiones son arácnidos con la apariencia general de los escorpiones y se diferencian de ellos por la ausencia de cola, y la ligera diferencia en su forma. Por lo tanto, este error es más que aceptable y comprensible debido a que el modelo entrenado nunca recibió información de lo que es un pseudoescorpión, y sí se ha entrenado con la posibilidad de que el objeto (escorpión) sea tapado parcialmente, lo cual explicaría la ausencia de cola en la imagen.

Fig. 92 Detección de pseudo-escorpión como escorpión

6.3. Clasificación de escorpiones

6.3.1. Sistema de reconocimiento y clasificación de tres especies de escorpiones

6.3.1.1. Reconocimiento y clasificación de dos géneros de escorpiones: *Bothriurus* y *Tityus*

En esta sección se presenta la comparación de resultados obtenidos con los sistemas de clasificación de escorpiones basados en los modelos LBPH y VGG16 (con y sin uso de Data Augmentation), desarrollados para la diferenciación de dos géneros de escorpiones (*Bothriurus* y *Tityus*).

Las matrices de confusión obtenidas durante el testeo de los tres modelos desarrollados son presentadas en las Fig. 93, Fig. 94, y Fig. 95, donde el eje vertical corresponde a la etiqueta real de la imagen y el eje horizontal corresponde a lo predicho por los modelos.

En este caso, el verdadero positivo (TP) ocurre cuando el género *Tityus* es esperado y es correctamente clasificado por el sistema de decisión. Por su parte, un falso positivo (FP) ocurre cuando el sistema de manera errónea identifica al género *Bothriurus* como el género *Tityus*, mientras que el caso más peligroso ocurre con un falso negativo (FN),

dado que ocurre cuando el sistema falla y confunde al género *Tityus* por el género *Bothriurus*.

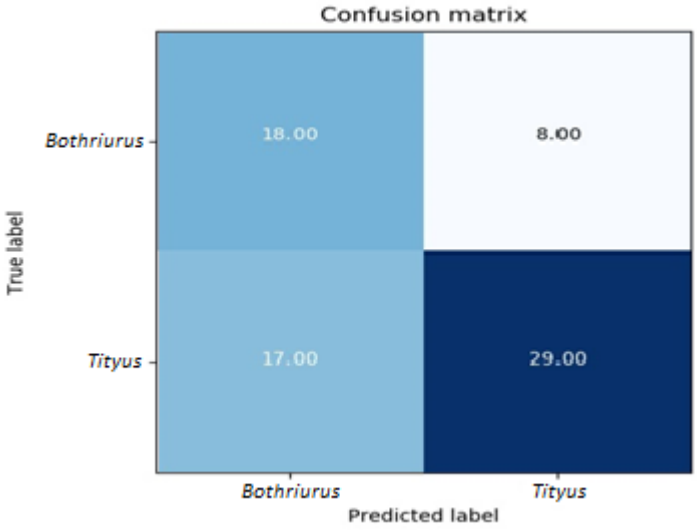


Fig. 93 Matriz de confusión Modelo LBPH

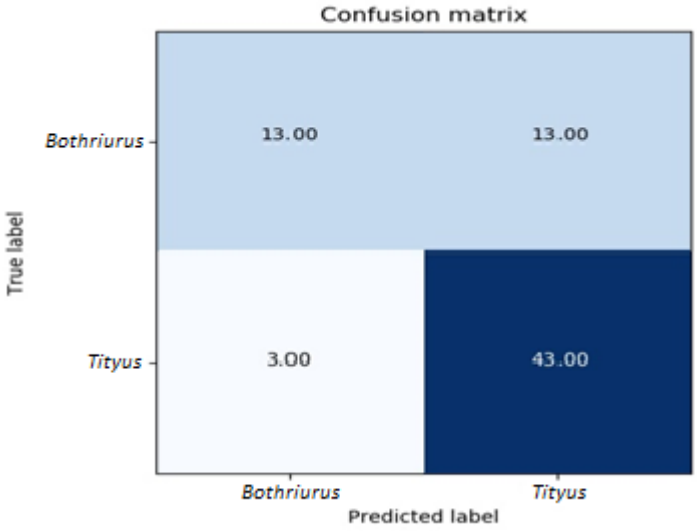


Fig. 94 Matriz de confusión Modelo DNN con TL (VGG16 sin DA)

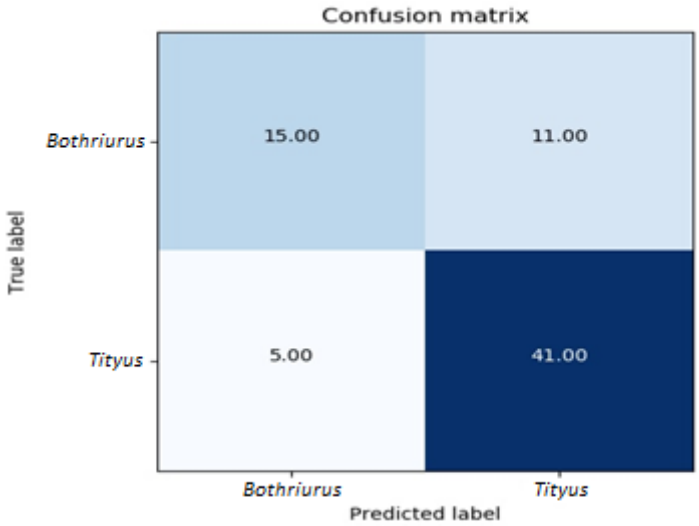


Fig. 95 Matriz de confusión Modelo DNN con TL y DA (VGG16 con DA)

La Tabla 12 muestra los valores de *accuracy*, precisión, y recall calculados con las ecuaciones mencionadas en la Sección 3.10. Como se puede ver, el desempeño del modelo LBPH es claramente menor que aquellos que utilizan redes neuronales densas (DNN). Pese a que el modelo LBPH tiene una buena precisión (comparable con los dos mejores modelos que se presentan en la tabla), el valor de R, asociado con el número de FN (*Tityus* identificados como *Bothriurus*) es mucho menor que los otros modelos, por lo que la seguridad sanitaria que este implica se vería comprometida con este modelo.

Method	Accuracy (A)	Precision (P)	Recall (R)	$F1_{measure}$
LBPH	0.65	0.78	0.63	0.70
DNN with TL	0.78	0.77	0.93	0.84
DNN with TL and DA	0.78	0.79	0.89	0.84

Tabla 12 Métricas de los tres modelos de clasificación desarrollados

Una mejora sustancial fue conseguida con ambos modelos DNN. En particular, el modelo DNN sin Data Augmentation (DA) obtuvo el valor R más elevado (0,93), sin embargo, una gran cantidad de épocas de entrenamiento fueron necesarias para llegar a este resultado. Valores de R cercanos a 0.9 o incluso superiores (por menores probabilidades de FN), brindan sistemas de reconocimiento eficientes para la seguridad sanitaria, debido a su gran capacidad para identificar el género *Tityus* de importancia

sanitaria. Si bien estos modelos tienen limitaciones para la identificación del género *Bothriurus*, como se mencionó anteriormente, este es un género no peligroso (sin importancia sanitaria). Esta limitación está relacionada con el hecho de que el número de FP es comparable al número de TN. Este resultado incide en los valores de A y P, que se acercan a 0,8.

Se ha realizado un análisis más detallado de los dos modelos más destacables a comparar con base en las curvas ROC, como se muestra en la Fig. 96. Se puede observar la considerable diferencia entre las áreas de estos dos modelos, lo que muestra claramente el mejor desempeño de un modelo que usa DNN para esta aplicación.

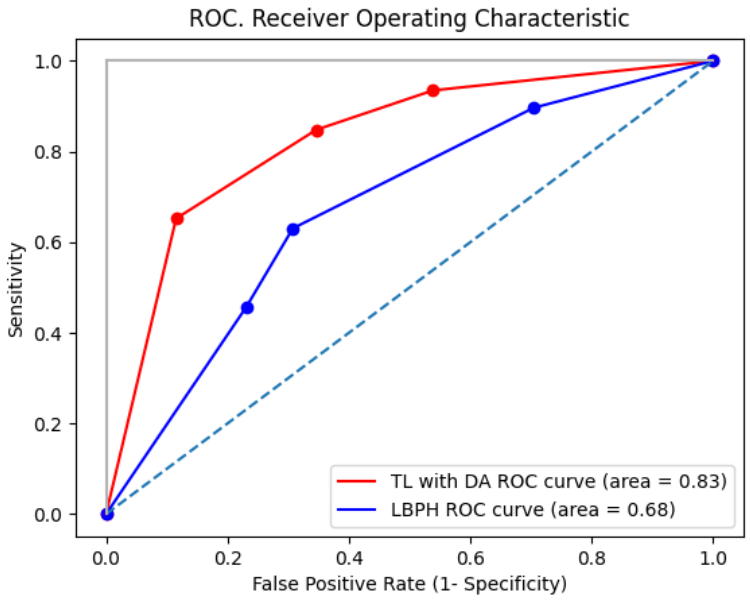


Fig. 96 Comparación entre curvas ROC de los dos modelos más relevantes a comparar

A partir de estos resultados, es posible establecer que ambos modelos de DNN se pueden utilizar de forma rápida y precisa para fines de reconocimiento y clasificación de géneros de escorpiones peligrosos (*Tityus*) y no peligrosos (*Bothriurus*). Sin embargo, el modelo DNN con TL y DA se consideró el mejor modelo debido a que la heurística de aumento de imagen garantiza un comportamiento robusto para un número significativo de imágenes de entrenamiento. En el caso específico de este modelo, para cada fase de entrenamiento, las imágenes generadas con la heurística de aumento de datos tienen características aleatorias, que provocan variaciones en las métricas calculadas. Por lo tanto, se realizaron treinta capacitaciones con el fin de estudiar la eficiencia del método propuesto. La Fig. 97 muestra la distribución de los resultados obtenidos para cada métrica en términos de diagrama de caja. En esta figura se presentan los valores de la mediana (M) y la desviación estándar (DE). También se puede observar que, para el modelo DNN con TL y DA y para un conjunto de entrenamiento dado, los valores de las métricas presentadas en la Tabla 12 son consistentes con los resultados mostrados en la Fig. 97.

Los resultados presentados en esta sección fueron publicados en una revista científica internacional [126].

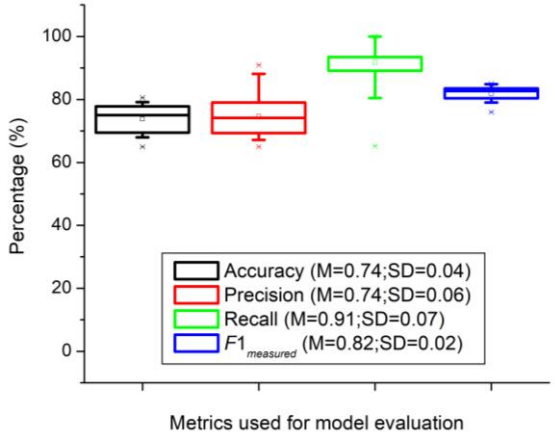


Fig. 97 Gráfico de cajas de la distribución de las métricas para treinta entrenamientos de Modelo DNN con TL y DA

6.3.1.2. Reconocimiento y clasificación de dos especies del género *Tityus*.

Como se mencionó anteriormente, dentro del género Tityus se pueden encontrar dos especies en la ciudad de La Plata: *Tityus carrilloi* y *Tityus confluens*, ambas de importancia sanitaria. Estas especies son demasiado similares entre sí, como se puede ver en la Fig. 98. La diferencia más significativa entre las dos especies son las tres franjas que se pueden ver en la parte superior de *Tityus carrilloi*. Otras diferencias menores pueden ser prácticamente imperceptibles para el sistema.

Fig. 98 Imágenes de Tityus carrilloi (arriba) y Tityus confluens (abajo). Las tres líneas en el lomo de Tityus carrilloi, son las diferencias más significativas de ambas especies

Con fines de investigación, los tres sistemas de clasificación de escorpiones basados en los modelos LBPH y VGG16 (con y sin uso de Data Augmentation), fueron entrenados para la identificación específica de ambas especies. En este caso, se ha utilizado un conjunto de datos de 76 imágenes del género *Tityus* (31 *carrilloi* y 45 de *confluens*) para entrenar y probar los modelos desarrollados.

El modelo LBPH y el modelo DNN con TL pero sin DA no pudieron diferenciar entre *Tityus carrilloi* y *Tityus confluens*. Sin embargo, el modelo con el enfoque de aumento de datos tuvo la capacidad de identificar adecuadamente ambas especies, lo que demuestra la utilidad de esta heurística. La arquitectura de red optimizada es la misma que se muestra en la Tabla 9. La red se entrenó durante 50 épocas. El tamaño del lote se estableció en 30 y el paso por época se estableció en 10, por lo que para entrenar el modelo se utilizaron 300 imágenes por época.

La matriz de confusión y la curva ROC obtenidas durante la prueba se muestran en la Fig. 99 y la Fig. 100, respectivamente. En este caso, TP ocurre cuando se espera encontrar un escorpión *Tityus confluens* y es correctamente reconocido por el clasificador; FP ocurre cuando se espera encontrar un escorpión *Tityus carrilloi*, pero es incorrectamente identificado como un *Tityus confluens*; mientras los *FN* ocurren cuando el sistema falla en reconocer a un escorpión *Tityus confluens*. Se obtuvieron valores de A = 0,72, P = 0,83 y R = 0,74 para este clasificador, para la exactitud, precisión y Recall, respectivamente. Aunque ambas especies son peligrosas para la salud humana, los resultados presentados en esta sección son particularmente útiles para fines de investigación y desarrollo biológicos, dado que pueden usarse para distinguir entre dos especies muy similares. Por lo tanto, la precisión es la métrica más relevante por considerar en este caso. Adicionalmente, para estudiar la eficiencia del método

propuesto, se utilizó un procedimiento similar al descrito previamente. Los resultados en términos de diagrama de caja se muestran en la Fig. 101.

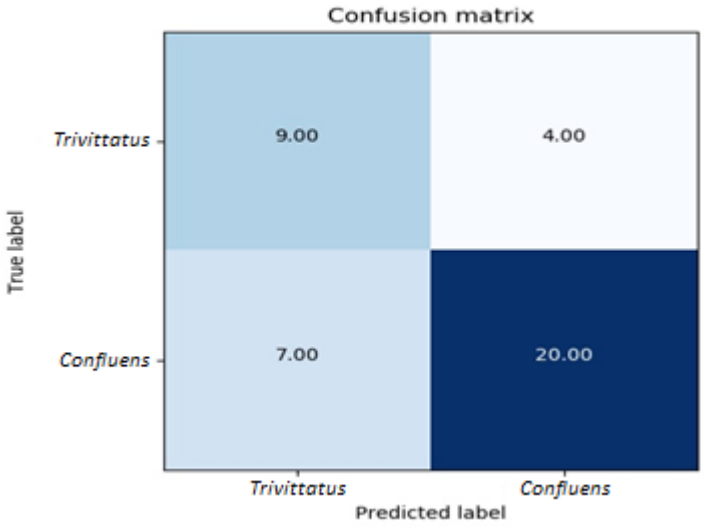


Fig. 99 Matriz de confusión Modelo DNN con TL y DA. Identificación dentro del género Tityus.

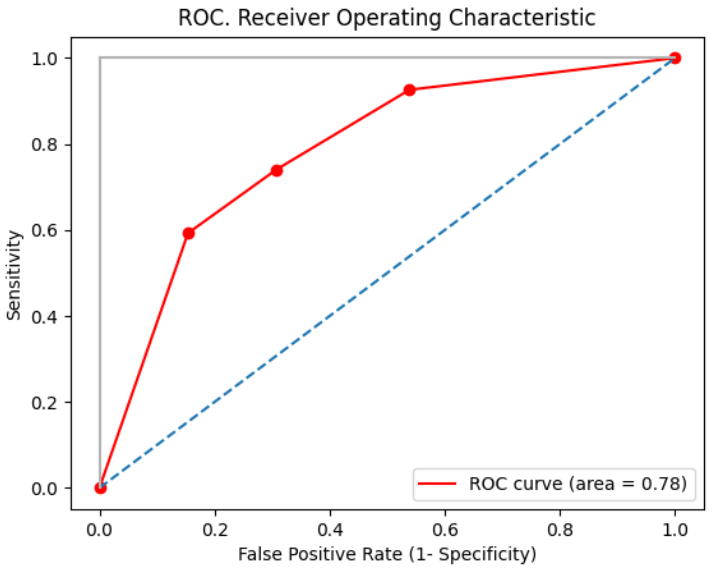


Fig. 100 Curva ROC Modelo DNN con TL y DA. Identificación dentro del género Tityus.

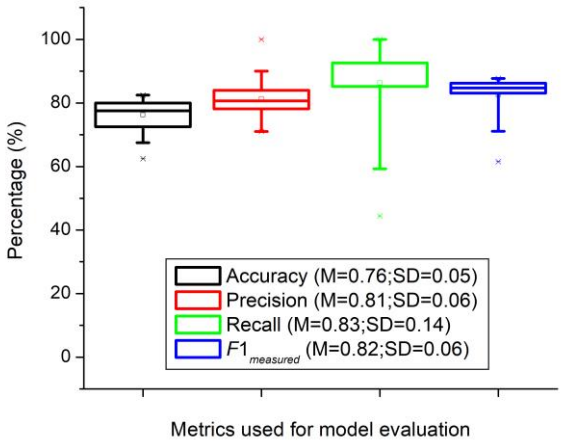


Fig. 101 Gráfico de cajas de distribución de métricas para treinta entrenamientos del Modelo DNN con TL y DA.

Los resultados de precisión obtenidos en este trabajo concuerdan muy bien con los reportados en [127], [128]. En estos trabajos, se muestran valores máximos de precisión cercanos al 72% y 78% utilizando arquitecturas de CNN para identificar razas de perros [127] y perros y gatos [128].

6.3.2. Sistema portátil de identificación de peligrosidad ante presencia de escorpiones

Finalmente, en esta sección se presentan los resultados obtenidos con el sistema de clasificación de escorpiones basado en el modelo MobileNetV2. Para realizar el análisis de desempeño del modelo entrenado se recurrió al 15% de la base de datos (126 imágenes), el cual no fue utilizado para el entrenamiento. Las métricas más relevantes obtenidas de cada clase respecto a las otras dos fueron, un R de 96% para *Tityus* (garantizando un buen nivel de seguridad sanitaria) y un A de 97% y 96% para *Tityus* y *Bothriurus* respectivamente (garantizando una buena discriminación respecto a la ausencia de escorpión en la imagen).

La clasificación de las muestras analizadas es mostrada en la Fig. 102 en la cual se puede apreciar la baja tasa de error que se comete en la clasificación.

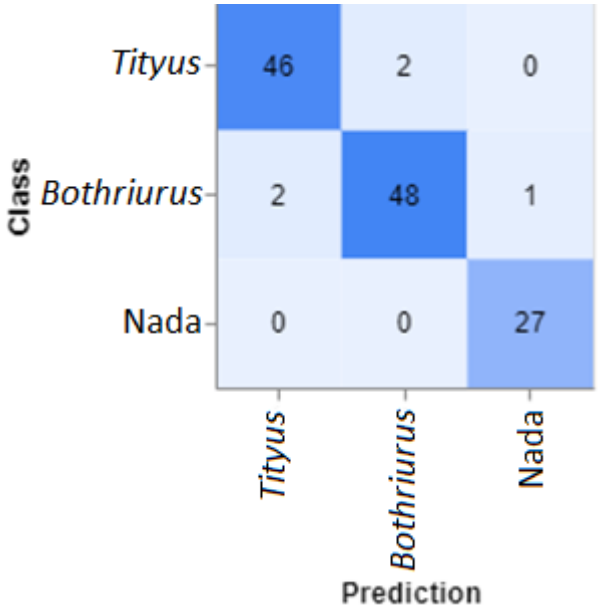


Fig. 102 Matriz de confusión Modelo MobileNetV2

Recordando que nuestro mayor interés es garantizar la correcta clasificación de "*Tityus*", dada su peligrosidad, podemos notar un detalle importante, ningún "*Tityus*" fue clasificado como Nada (ausencia de escorpión), por lo que, teniendo en cuenta la fórmula de Recall, R de "*Tityus*" frente a "Nada" es del 100%. Por otro lado, solo una imagen de "*Bothriurus*" ha sido clasificado como "Nada", dando un R del 97,96% entre estos dos.

Una vez aclarada la capacidad de distinguir a ambos escorpiones de un entorno vacío, analizaremos la capacidad de diferenciarlos, dado el objetivo principal de la app. Analizando estos dos géneros, con principal foco en el de importancia sanitaria (*Tityus*), podemos llegar a calcular un R de 95,83%, y dado que FP=FN entonces podemos decir que R=P=F1, por lo que se puede llegar a la conclusión de que el sistema clasifica correctamente las clases entrenadas, y por sobre todo que es un sistema seguro a nivel sanitario dada la baja tasa de FN en la detección de *Tityus*.

En la Fig. 103 se puede observar la correcta clasificación de los dos géneros de escorpiones por parte de la app desarrollada, demostrando además una alta precisión de clasificación.

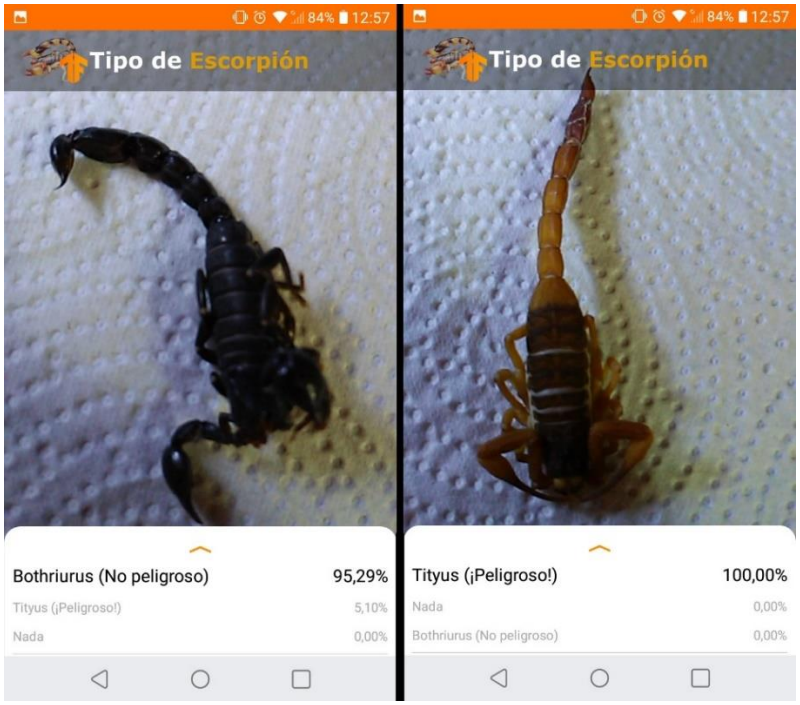


Fig. 103 Clasificación de dos escorpiones con la App "Peligroso?"

Considerando solamente las imágenes correspondientes a los dos géneros de escorpiones (*Tityus* y *Bothriurus*), es posible comparar los resultados obtenidos con el modelo MobileNetV2, con aquellos presentados en la Tabla 11 para los modelos LBPH y VGG16 (con y sin uso de Data Augmentation). De esta manera, la Tabla 13 resume los resultados para las cuatro métricas consideradas. Estos resultados muestran que el modelo MobileNetV2 es claramente más eficiente que los otros para discriminar entre escorpiones peligrosos (*Tityus*) y no peligrosos (*Bothriurus*).

Method	Accuracy (A)	Precision (P)	Recall (R)	$F_{measure}$
LBPH	0.65	0.78	0.63	0.70
DNN with TL	0.78	0.77	0.93	0.84
DNN with TL and DA	0.78	0.79	0.89	0.84
MobileNetV2	0.96	0.96	0.96	0.96

Tabla 13 Comparación de métricas para cuatro modelos diferentes

Capítulo 7: Conclusiones

A lo largo del presente trabajo de Tesis Doctoral se han estudiado diferentes modelos de Aprendizaje Profundo, los cuales han sido implementados en sistemas de detección, clasificación y reconocimiento de escorpiones, de manera temprana, precisa, no invasiva y segura, desarrollados con fines de prevención para permitir el control sanitario. A continuación, se presentan las conclusiones de la Tesis y las líneas de trabajo a futuro.

7.1. Conclusiones

Durante el desarrollo de esta tesis se llegaron a muchas conclusiones, en esta sección intentaremos mencionarlas a todas.

En primer lugar, se puede destacar que las heurísticas de detección de objeto y clasificación implementadas obtuvieron resultados satisfactorios para cumplir su propósito y han superado las expectativas en cuestión de prevención de incidentes con escorpiones de importancia sanitaria.

En segundo lugar, hay que destacar que todos los métodos y programas implementados son gratuitos y de código abierto, lo que demuestra el libre acceso a estas poderosas herramientas capaces de procesar imágenes de la manera más compleja en la actualidad. No solo el software libre provee estas herramientas, sino que, si se buscan las innovaciones que se están llevando a cabo en esta y otras temáticas, se llega a la conclusión de que, el código abierto se encuentra en la vanguardia del desarrollo de las nuevas tecnologías. Por otro lado, para utilizar librerías de este tipo hay que tomar recaudos respecto a la compatibilidad entre las mismas (como se vio en el Capítulo 5), siempre se recomienda no usar la última versión disponible sino una versión anterior, llamada estable por estar más probada y con sus errores o fallas (*bugs*) solucionados. Además, en caso de tener algún percance es probable que su solución se encuentre disponible en la misma página de la librería o en un foro de consultas.

En tercer lugar, podemos mencionar una temática, que muchos recién iniciados en el tema clasificarían erróneamente de menor importancia, como lo es la selección de la base de datos que se utilizará para entrenar. En nuestro entrenamiento, nuestra base de datos debe consumirnos aproximadamente el 80% del tiempo, y al ser ésta compuesta por imágenes, no podemos caer en el error de solo obtener muchas fotografías del objeto a detectar o clasificar y comenzar a entrenar la red neuronal, porque lo único que conseguiremos es perder el 100% del tiempo que íbamos a utilizar y tener que volver a empezar. Una base de datos de imágenes que hemos obtenido nosotros mismos requiere ser visualizada una a la vez por una persona que determine imparcialmente si esta sirve o no, está repetida o no; de nada nos sirve tener imágenes repetidas o ininteligible porque solo nos hará gastar más tiempo de entrenamiento, sin

mejorarlo o en el peor de los casos empeorándolo. Un ejemplo de un modelo mal entrenado por falta de diversidad de imágenes sería cargarle imágenes de escorpiones con fondo blanco, en este caso el sistema tenderá a detectar solo los escorpiones que tengan fondo blanco e ignorará a todos los demás. Otro ejemplo, pero de imágenes con mala calidad con los escorpiones apenas visibles, sería que el sistema se le proporcionarían imágenes tan generalizadas que no sabría distinguir un escorpión de cualquier otro objeto. Por ello es fundamental dedicar el tiempo suficiente a nuestra base de datos, porque un mal modelo no necesariamente implica una red neuronal defectuosa o falta/exceso de entrenamiento, puede ser una base de datos no acorde a las necesidades del sistema. Es por lo mencionado que nuestra base de datos se obtuvo de múltiples fuentes con diferentes especímenes en variadas condiciones, así poder garantizar la diversidad de nuestra base de datos, la cual debió ser complementada con técnicas de data augmentation las cuales posibilitaron imágenes en condiciones no contempladas durante la adquisición. Complementariamente se verificó de manera sistemática el estado de las imágenes obtenidas para garantizar el correcto entrenamiento.

En cuanto a la capacidad de cómputo, en cuarto lugar, podemos mencionar que debido a la posibilidad de utilizar Colaboratory, hoy en día, no es necesario contar con equipo con gran capacidad de cómputo para llevar a cabo proyectos de esta envergadura. Por otra parte, se puede destacar la diferencia considerable entre el entrenamiento de un clasificador de imagen frente a un detector de objetos en lo que respecta a capacidad y tiempo de cómputo que requieren. Mientras que los clasificadores se podían entrenar con nuestros equipos en pocos minutos, los sistemas de detección de objetos hubiesen demorado semanas en ellos, y terminaron demorando solo unos días con la ayuda de este entorno de desarrollo web.

Con respecto a los modelos empleados se pueden mencionar varios puntos:

- El clasificador en cascada (Haar) para la detección de objetos es una herramienta que brinda una solución eficaz para la detección en tiempo real debido al bajo consumo de cómputo en su implementación. Sin embargo, su entrenamiento requiere mucho tiempo de procesamiento en comparación con otros métodos más eficaces, pero más pesados de implementar.
- El modelo YOLOv3 demostró ser mejor para detectar objetos que el sistema Haar, con una detección continua en el tiempo. Sin embargo, demostró demandar más recursos para su implementación, y aunque su entrenamiento nos demandó menos tiempo requirió mayor poder de cómputo dado que se debió utilizar GPU para el mismo.
- La detección de la fluorescencia demostró ser una buena herramienta para mejorar el funcionamiento de los detectores de objetos (Haar y YOLOv3) realizando un sistema de confirmación mediante la misma. Esto solo podrá ser empleado en aquellos detectores que puedan ser implementados en ambientes en constante ausencia de luz natural y con iluminación UV.

- El modelo MobileNetV2 no solo demostró ser un método eficaz en la detección de objetos, sino que, además, demostró ser el sistema más práctico para su implementación en un smartphone, gracias al formato TensorFlow Lite.
- El modelo LBPH, aunque muy útil en su campo para identificar rostros, demostró no ser idóneo para la identificación de escorpiones dado su bajo nivel de prevención frente a incidentes de importancia sanitaria en comparación con los modelos que utilizan Deep Learning.
- El modelo VGG16 es un sistema eficaz para la diferenciación de escorpiones, no solamente entre dos géneros diferentes (*Bothriurus* y *Tityus*), sino que también entre especies del mismo género (*Tityus carrilloi* y *Tityus confluens*).
- El modelo MobileNetV2 además de ser un eficaz detector del objeto escorpión, demostró una excelente capacidad de respuesta como clasificador en ambientes no controlados, ideal para la diferenciación de escorpiones por su peligrosidad y por su practicidad a la hora de implementarlo en un smartphone.
- El sistema de detección mediante un sensor de color analógico ha mejorado su rendimiento mediante la implementación de un sensor de distancia, por reducir con este los falsos positivos. Por lo que se pudo proponer un sistema de detección eficaz y sencillo para corta distancia de operación y para ambientes sin luz natural.

Es necesario destacar que ninguno de los modelos de Aprendizaje Profundo analizados en este trabajo ha sido previamente utilizados para la detección y/o clasificación de escorpiones. Sin embargo, se han utilizado estos modelos en otros seres vivos ajenos a este tipo de arácnidos, un ejemplo de ello son los insectos.

Por último, cabe destacar la importancia de las heurísticas de Transfer Learning y Data Augmentation, las cuales resultaron esenciales para realizar un entrenamiento más rápido y mejorar la calidad de la base de datos, respectivamente. Sin Transfer Learning los modelos no hubiesen tenido ningún punto de referencia para comenzar su entrenamiento, por lo que éste habría tomado mucho más tiempo para llegar a los valores de desempeño que se lograron. En lo que respecta a Data Augmentation, posibilitó tener una base de datos más grande y heterogénea, lo que proporciona mayor variedad de posibilidades de detección a los sistemas a entrenar. Cabe aclarar que, el sistema YOLOv3 ya contempla esta última heurística dentro de su algoritmo de entrenamiento.

7.2. Líneas de trabajo a futuro

Como era de esperar, esta tesis ha generado múltiples líneas a futuro. En esta sección mencionaremos las más significativas.

En lo que respecta a la detección y clasificación de escorpiones, con heurísticas de Machine Learning y Deep Learning, se pueden mencionar varias opciones, entre ellas se encuentra la implementación de otros modelos diferentes a los utilizados para el

entrenamiento de los mismos y hacer un estudio comparativo de estos, tales como Inception, ResNet, R-FCN, entre otros.

Cuando analizamos los entrenamientos que hemos realizado y las bases de datos que se han implementado, se puede llegar a la conclusión de que, de tener una capacidad de cómputo mayor se podría procesar una base de datos mayor, más heterogénea en sus entornos, lo cual podría mejorar aún más nuestro sistema ya eficiente. Es por ello que, una posible línea a futuro consiste en obtener una base de datos más grande, en múltiples condiciones de entorno y posición del espécimen, y con esta nueva base de datos, entrenar un modelo con mayor cantidad de capas hasta llegar al límite de entrenamiento posible, y así poder analizar la efectividad de este nuevo sistema, que indudablemente será aún mejor que los buenos resultados que hemos obtenido.

Otra línea a futuro posible es la implementación de un sistema de segmentación de imágenes, el cual consiste en que la detección del escorpión se demarque por su contorno, demostrando así qué tan precisa es la detección, y en caso de tratarse de una falsa detección, poder visualizar exactamente cuál es la causante del error dentro de la imagen.

Finalmente, durante el trascurso de esta tesis se desarrolló la detección, clasificación y reconocimiento de escorpiones, en particular de las especies presentes en el partido de La Plata. Dado que esta tesis logró demostrar la factibilidad de la implementación de los sistemas empleados para cumplir el propósito de prevenir incidentes con escorpiones de importancia sanitaria, una línea a futuro que se puede desarrollar es el entrenamiento de una mayor cantidad de especies, para así extender la región biogeográfica donde se puedan aplicar los sistemas desarrollados. Para ello se necesitarán imágenes de los escorpiones que habitan en dichas regiones, para lo cual se deberá contar con acceso a base de datos locales o con la cooperación de los institutos de aracnología respectivos para la adquisición de las imágenes necesarias para el entrenamiento y la realización de los respectivos ensayos de desempeño.

Bibliografía

[1] "Guía de Prevención, Diagnóstico, Tratamiento y Vigilancia Epidemiológica de los Envenenamientos por Arañas," Accessed: Jun. 09, 2022. [Online]. Available: http://www.msal.gov.ar/redartox.

[2] "Vecinos en guardia por la aparición de alacranes en La Plata," *Diario El Día*, La Plata, Buenos Aires, Argentina, Jan. 30, 2020.

[3] "La UNLP advierte sobre cómo actuar ante la presencia de escorpiones," *Universidad Nacional de La Plata*, La Plata, Buenos Aires, Argentina, Apr. 2019.

[4] S. González, L. Giambelluca, and A. González, "Escorpiones de La Plata: aproximacion a sus hábitos y distribución," *SEDICI*, 2019, [Online]. Available: http://sedici.unlp.edu.ar/handle/10915/73514.

[5] A. A. Ojanguren-affilastro, C. Bizzotto, L. C. Lanari, and A. R. De Roodt, "Presencia de Tityus confluens Borelli en la ciudad de Buenos Aires y expansión de la distribución de las especies de importancia médica de Tityus (Scorpiones; Buthidae) en la Argentina," *Rev. del Mus. Argentino Ciencias Nat. nueva Ser.*, vol. 21, no. 1, pp. 101–112, 2019.

[6] F. L. Giambelluca, J. A. Rapallini, M. Cappelletti, and L. A. Giambelluca, "Alarma detectora de escorpiones," Universidad Nacional de La Plata, 2018.

[7] M. J. Hutt and P. J. Houghton, "A survey from the literature of plants used to treat scorpion stings," *J. Ethnopharmacol.*, vol. 60, no. 2, pp. 97–110, 1998, doi: 10.1016/S0378-8741(97)00138-4.

[8] E. Blanco and G. Salas, *Arácnidos guía de campo*. Proyecto para la divulgación del conocimiento científico, 2007.

[9] "The Global Biodiversity Information Facility." https://www.gbif.org/es/.

[10] "QGIS." [Online]. Available: https://www.qgis.org/en/site/.

[11] "Tityus Koch," *gbif.org*, 1836. https://www.gbif.org/species/2123713.

[12] Ehrenberg, "Leiurus quinquestriatus," *gbif.org*, 1828. https://www.gbif.org/species/6893601.

[13] Lutz and Mello, "Tityus serrulatus," *gbif.org*, 1922. https://www.gbif.org/species/6893860.

[14] Olivier, "Androctonus crassicauda," *gbif.org*, 1807. https://www.gbif.org/species/6894171.

[15] Ehrenberg, "Androctonus," *gbif.org*, 1828. https://www.gbif.org/species/4656247.

[16] Leach, "Buthus," *gbif.org*, 1825. https://www.gbif.org/species/3253203.

[17] Hemprich and Ehrenberg, "Leiurus Ehrenberg," *gbif.org*, 1828. https://www.gbif.org/species/3253202.

[18] Vachon, "Mesobuthus," *gbif.org*, 1950. https://www.gbif.org/species/4404674.

[19] Pocock, "Parabuthus," *gbif.org*, 1890. https://www.gbif.org/species/4656188.

[20] Academia de Ciencias de California, "iNaturalist," 2008. https://www.inaturalist.org/.

[21] SmugMug, “Flickr,” 2004. https://www.flickr.com/.

[22] J. Dorsey, “Twitter,” 2006. https://twitter.com/.

[23] A. de Roodt *et al.*, “Expansión de la distribución de escorpiones del género Tityus C. L. Koch 1836 en Argentina: Implicancias sanitarias,” *Acta toxicológica argentina*, vol. 27, no. 3, pp. 109–119, 2019.

[24] F. T. Abushama, “On the behaviour and sensory physiology of the scorpion,” *Anim. Behav.*, vol. 12, p. 140, 1964.

[25] “Bothriurus Peters,” *gbif.org*, 1861. https://www.gbif.org/species/4656407.

[26] Pocock, “Centruroides suffusus,” *gbif.org*, 1902. https://www.gbif.org/species/6894260.

[27] Ewing, “Centruroides sculpturatus,” *gbif.org*, 1928. https://www.gbif.org/species/6894244.

[28] L. Giambelluca, S. González, G. Reboredo, S. Rodriguez Gil, and A. González, “Evolución y evaluación de la aparición de escorpiones en la ciudad de La Plata (Buenos Aires, Argentina),” *Rev. del Mus. La Plata*, vol. 3, p. 102R:102R, 2018.

[29] G. R. C. Blass and D. D. Gaffin, “Light wavelength biases of scorpions,” *Anim. Behav.*, vol. 76, no. 2, pp. 365–373, 2008, doi: 10.1016/j.anbehav.2007.12.022.

[30] G. R. . Blass and D. D. Gaffin, “Sensory ecology and orientational behaviors,” *Scorpion Biol. Res. (Ed. by P. H. Brownell G. A. Polis), Oxford Univ. Press*, pp. 159–183, 2001.

[31] E. A. Camp and D. D. Gaffin, “Escape Behavior Mediated by Negative Phototaxis in the Scorpion Paruroctonus utahensis,” *Am. Arachnol. Soc.*, vol. 27, no. 3, pp. 679–684, 1999.

[32] A. Fasel, P. A. Muller, P. Suppan, and E. Vauthey, “Photoluminescence of the African scorpion ‘Pandinus imperator,’” *J. Photochem. Photobiol. B Biol.*, vol. 39, no. 1, pp. 96–98, 1997, doi: 10.1016/S1011-1344(96)00016-4.

[33] L. M. Frost, D. R. Butler, B. O’Dell, and V. Fet, “A coumarin as a fluorescent compound in scorpion cuticle,” *Scorpions 2001; Memoriam Gary A P.*, p. 363e368, 2001.

[34] L. D. Honetschlager, “A new method for hunting scorpions,” *Turtox News*, vol. 43, p. 69, 1965.

[35] S. J. Stachel, S. A. Stockwell, and D. L. Van Vranken, “The fluorescence of scorpions and cataractogenesis,” *Chem. Biol.*, vol. 6, no. 8, pp. 531–539, 1999, doi: 10.1016/S1074-5521(99)80085-4.

[36] C. T. Kloock, “Aerial insects avoid fluorescing scorpions,” *Euscorpius*, vol. 21, pp. 1–7, 2005, doi: 10.18590/euscorpius.2005.vol2005.iss21.1.

[37] L. A. Giambelluca, Francisco Luis; Osio, Jorge; Cappelletti, Marcelo; Rapallini, José A.; Giambelluca, “Alarma detectora de escorpiones utilizando procesamiento digital de imágenes,” *IX Congr. Microelectrónica Apl.*, 2018.

[38] C. T. Kloock, “A comparison of fluorescence in two sympatric scorpion species,” *J. Photochem. Photobiol. B Biol.*, vol. 91, no. 2–3, pp. 132–136, 2008, doi: 10.1016/j.jphotobiol.2008.02.008.

[39] N. F. Hadley and S. C. Williams, “Surface Activities of Some North American Scorpions in

Relation to Feeding," vol. 49, no. 4, pp. 726–734, 1968.

[40] A. H. Shehab, Z. S. Amr, and J. A. Lindsell, "Ecology and biology of scorpions in Palmyra, Syria," *Turkish J. Zool.*, vol. 35, no. 3, pp. 333–341, 2011, doi: 10.3906/zoo-0904-19.

[41] P. Brownell and R. D. Farley, "Orientation to vibrations in sand by the nocturnal scorpion Paruroctonus mesaensis: Mechanism of target localization," *J. Comp. Physiol. □ A*, vol. 131, no. 1, pp. 31–38, 1979, doi: 10.1007/BF00613081.

[42] M. J. B. Eberhard, D. Lang, B. Metscher, G. Pass, M. D. Picker, and H. Wolf, "Structure and sensory physiology of the leg scolopidial organs in Mantophasmatodea and their role in vibrational communication," *Arthropod Struct. Dev.*, vol. 39, no. 4, pp. 230–241, 2010, doi: 10.1016/j.asd.2010.02.002.

[43] A. M. Aibinu, B. A. Sadiq, E. Joseph, H. B. Salau, and M. J. E. Salami, "Development of an intelligent scorpion detection technique using vibration analysis," *2014 Int. Conf. Comput. Inf. Sci. ICCOINS 2014 - A Conf. World Eng. Sci. Technol. Congr. ESTCON 2014 - Proc.*, pp. 2–5, 2014, doi: 10.1109/ICCOINS.2014.6868374.

[44] L. Eduardo Acosta and E. A. Maury, "Stridulation in Timogenes elegans (Mello-Leitao) (Scorpiones, Bothriuridae)," *Bol. La Soc. Biol. Concepcion, Chile*, vol. 61, pp. 29–38, 1990.

[45] A. Ojanguren Affilastro, "Estudio monográfico de los escorpiones de la República Argentina," *Rev. ibérica Aracnol.*, no. 11, pp. 75–246, 2005.

[46] N. Nilsson, *Principles of Artificial Intelligence*. 1980.

[47] R. E. López Briega, "Introducción a la inteligencia artificial," 2017. https://relopezbriega.github.io/blog/2017/06/05/introduccion-a-la-inteligencia-artificial/.

[48] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science (80-.).*, vol. 349, no. 6245, pp. 255–260, Jul. 2015, doi: 10.1126/science.aaa8415.

[49] E. Alpaydin, *Machine Learning: The New AI*. MIT Press, 2016.

[50] J. G. Carbonell, "Machine learning research," *ACM SIGART Bull.*, vol. 18, no. 77, pp. 29–29, 1981, doi: 10.1145/1056743.1056744.

[51] F. Chollet, *Deep learning with python*. Manning Shelter Island, 2018.

[52] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[53] E. R. Davies, *Computer Vision*, 5th ed. Academic Press, 2017.

[54] A. Dairi, F. Harrou, Y. Sun, and M. Senouci, "Obstacle Detection for Intelligent Transportation Systems Using Deep Stacked Autoencoder and k-Nearest Neighbor Scheme," *IEEE Sens. J.*, vol. 18, no. 12, pp. 5122–5132, 2018, doi: 10.1109/JSEN.2018.2831082.

[55] W. Budiharto, A. A. S. Gunawan, J. S. Suroso, A. Chowanda, A. Patrik, and G. Utama, "Fast Object Detection for Quadcopter Drone Using Deep Learning," *2018 3rd Int. Conf. Comput. Commun. Syst. ICCCS 2018*, pp. 367–371, 2018, doi: 10.1109/CCOMS.2018.8463284.

[56] V. D. Nguyen, H. Van Nguyen, D. T. Tran, S. J. Lee, and J. W. Jeon, "Learning Framework for Robust Obstacle Detection, Recognition, and Tracking," *IEEE Trans. Intell. Transp.*

Syst., vol. 18, no. 6, pp. 1633–1646, 2017, doi: 10.1109/TITS.2016.2614818.

[57] H. Kim *et al.*, “Image-based monitoring of Jellyfish using deep learning architecture,” *IEEE Sens. J.*, vol. 16, no. 8, pp. 2215–2216, 2016, doi: 10.1109/JSEN.2016.2517823.

[58] B. K. Tripathi, “On the complex domain deep machine learning for face recognition,” *Appl. Intell.*, vol. 47, no. 2, pp. 382–396, 2017, doi: 10.1007/s10489-017-0902-7.

[59] V. Andrearczyk and P. F. Whelan, “Using filter banks in Convolutional Neural Networks for texture classification,” *Pattern Recognit. Lett.*, vol. 84, pp. 63–69, 2016, doi: 10.1016/j.patrec.2016.08.016.

[60] Y. Xu, Y. Wang, J. Yuan, Q. Cheng, X. Wang, and P. L. Carson, “Medical breast ultrasound image segmentation by machine learning,” *Ultrasonics*, vol. 91, pp. 1–9, 2018, doi: 10.1016/j.ultras.2018.07.006.

[61] X. Li, H. Shen, L. Zhang, H. Zhang, Q. Yuan, and G. Yang, “Recovering quantitative remote sensing products contaminated by thick clouds and shadows using multitemporal dictionary learning,” *IEEE Trans. Geosci. Remote Sens.*, vol. 52, no. 11, pp. 7086–7098, 2014, doi: 10.1109/TGRS.2014.2307354.

[62] Y. Sun, Y. Liu, G. Wang, and H. Zhang, “Deep Learning for Plant Identification in Natural Environment,” *Comput. Intell. Neurosci.*, vol. 2017, p. 7361042, 2017, doi: 10.1155/2017/7361042.

[63] A. M. Dawud, K. Yurtkan, and H. Oztoprak, “Application of Deep Learning in Neuroradiology: Brain Haemorrhage Classification Using Transfer Learning,” *Comput. Intell. Neurosci.*, vol. 2019, p. 4629859, 2019, doi: 10.1155/2019/4629859.

[64] S. Cui, Y. Zhou, Y. Wang, and L. Zhai, “Fish Detection Using Deep Learning,” *Appl. Comput. Intell. Soft Comput.*, vol. 2020, p. 3738108, 2020, doi: 10.1155/2020/3738108.

[65] D. Khemasuwan, J. S. Sorensen, and H. G. Colt, “Artificial intelligence in pulmonary medicine: computer vision, predictive model and COVID-19,” *Eur. Respir. Rev.*, vol. 29, no. 157, p. 200181, Sep. 2020, doi: 10.1183/16000617.0181-2020.

[66] A. Esteva *et al.*, “Deep learning-enabled medical computer vision,” *npj Digit. Med.*, vol. 4, no. 1, p. 5, Dec. 2021, doi: 10.1038/s41746-020-00376-2.

[67] J. Zhao, R. Masood, and S. Seneviratne, “A Review of Computer Vision Methods in Network Security,” *IEEE Commun. Surv. Tutorials*, vol. 23, no. 3, pp. 1838–1878, 2021, doi: 10.1109/COMST.2021.3086475.

[68] A. Saxena, D. K. Gupta, and S. Singh, “An Animal Detection and Collision Avoidance System Using Deep Learning,” 2021, pp. 1069–1084.

[69] G. K. Verma and P. Gupta, “Wild Animal Detection Using Deep Convolutional Neural Network,” 2018, pp. 327–338.

[70] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, 2001, vol. 1, pp. I–I, doi: 10.1109/CVPR.2001.990517.

[71] S. López-Tapia, R. Molina, and N. Pérez de la Blanca, “Using machine learning to detect and localize concealed objects in passive millimeter-wave images,” *Eng. Appl. Artif. Intell.*, vol. 67, no. October 2016, pp. 81–90, 2018, doi: 10.1016/j.engappai.2017.09.005.

[72] D.-P. Fan, G.-P. Ji, M.-M. Cheng, and L. Shao, “Concealed Object Detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, pp. 1–1, 2021, doi: 10.1109/TPAMI.2021.3085766.

[73] S. Bozinovski, “Reminder of the first paper on transfer learning in neural networks, 1976,” *Inform.*, vol. 44, no. 3, pp. 291–302, 2020, doi: 10.31449/INF.V44I3.2828.

[74] D. A. V. Dyk and X. L. Meng, “The art of data augmentation,” *J. Comput. Graph. Stat.*, vol. 10, no. 1, pp. 1–50, 2001, doi: 10.1198/10618600152418584.

[75] S. FROHWIRTH-SCHNAT, “DATA AUGMENTATION AND DYNAMIC LINEAR MODELS,” vol. 15, no. 2, pp. 183–202, 1992.

[76] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J. Big Data*, vol. 6, no. 1, 2019, doi: 10.1186/s40537-019-0197-0.

[77] A. Díez Herranz and M. Tobal González, “Las curvas ROC en la evaluación de las pruebas diagnósticas.,” *Med. Clin. (Barc).*, vol. 108, no. 1, 1997.

[78] F. Berzal, “Redes Neuronales & Deep Learning,” *Granada*, 2018.

[79] M. Olafenwa and J. Olafenwa, *ImageAI Documentation*. 2020.

[80] CEPAVE_CONICET-UNLP, “¿Es araña o escorpión? - Apps in Google Play,” 2017. https://play.google.com/store/apps/details?id=com.mattone.nico.bichoscepave&hl=es_AR&gl=US.

[81] M. Olafenwa, “LabelImg.” https://readthedocs.org/projects/labelimg/.

[82] J. Nelson, “LabelImg,” *Roboflow*, 2020. https://blog.roboflow.com/labelimg/.

[83] “Roboflow.” https://docs.roboflow.com/.

[84] B. Yang and S. Chen, “A comparative study on local binary pattern (LBP) based face recognition: LBP histogram versus LBP image,” *Neurocomputing*, vol. 120, pp. 365–379, 2013, doi: 10.1016/j.neucom.2012.10.032.

[85] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–14, 2015.

[86] X. Yao, “Transfer Learning,” *Yaonotes Blog*, 2020. https://blog.yaonotes.org/2020/06/12/transfer-learning/.

[87] A. G. Howard *et al.*, “MobileNets: Efficient convolutional neural networks for mobile vision applications,” *arXiv*, 2017.

[88] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” *arXiv*, pp. 4510–4520, 2018.

[89] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[90] J. Redmon and A. Farhadi, “YOLO v.3,” *Tech Rep.*, pp. 1–6, 2018, [Online]. Available: https://pjreddie.com/media/files/papers/YOLOv3.pdf.

[91] A. Bochkovskiy, C. Y. Wang, and H. Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv*, 2020.

[92] P. Raybaut, *Spyder Documentation*, 3rd ed. 2017.

[93] Google, “What is Colaboratory?” https://colab.research.google.com/notebooks/intro.ipynb.

[94] E. Bisong, *Google Colaboratory BT - Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners*. 2019.

[95] T. Carneiro, R. V. M. Da Nobrega, T. Nepomuceno, G. Bin Bian, V. H. C. De Albuquerque, and P. P. R. Filho, “Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications,” *IEEE Access*, vol. 6, pp. 61677–61685, 2018, doi: 10.1109/ACCESS.2018.2874767.

[96] M. Carney *et al.*, “Teachable machine: Approachable web-based tool for exploring machine learning classification,” *Conf. Hum. Factors Comput. Syst. - Proc.*, 2020, doi: 10.1145/3334480.3382839.

[97] OpenCV, “Haar Training PEN.” https://github.com/opencv/opencv/files/3076684/HaarTrainingPEN.zip.

[98] “Cascade Classifier Training.” https://docs.opencv.org/3.4.9/dc/d88/tutorial_traincascade.html.

[99] “Cascade Classifier.” https://docs.opencv.org/3.4.9/db/d28/tutorial_cascade_classifier.html.

[100] “Keras. Documentation for individual models.” https://keras.io/applications/.

[101] “Tensorflow,” [Online]. Available: https://www.tensorflow.org/.

[102] NVIDEA, “CUDA Toolkit Documentation.” https://docs.nvidia.com/cuda/.

[103] T. Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, “Focal Loss for Dense Object Detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 2, pp. 318–327, Aug. 2017, doi: 10.1109/TPAMI.2018.2858826.

[104] S.-H. Tsang, “Review: YOLOv3 — You Only Look Once (Object Detection),” *towardsdatascience.com*, 2019. https://towardsdatascience.com/review-yolov3-you-only-look-once-object-detection-eab75d7a1ba6.

[105] C. Y. Wang, A. Bochkovskiy, and H. Y. M. Liao, “Scaled-YOLOv4: Scaling cross stage partial network,” *arXiv*, 2020.

[106] Q. Chen, “A Basic Introduction to OpenCV for Image Processing,” *Univ. Ottawa*, 2007.

[107] R. Igual and C. Medrano, “Tutorial de OpenCV,” *Lab. visión por Comput.*, 2008.

[108] G. . Bradski and A. Kaebler, *Learning OpenCV*. 2008.

[109] M. Notes, “Diseño de módulo de Pre procesamiento y Extracción de características físicas de una imagen,” *Esc. Politécnica Nac.*, vol. Cap. 3, pp. 54–55, 2006.

[110] C. F. Calderón, “Visión Artificial–Umbralización,” *Univ. Pontif. Javeriana*, pp. 1–11, 2011.

[111] F. L. Giambelluca, “Base de datos de escorpiones Tityus y Bothriurus.,” *Roboflow*, 2021. https://app.roboflow.com/ds/AI5quAoBA3?key=cHjhZBUwKB.

[112] R. Sanders, “The pareto principle: Its use and abuse,” *J. Consum. Mark.*, vol. 4, no. 1, pp. 37–40, 1987, doi: 10.1108/eb008188.

[113] “Solución incompatibilidad de App Detección de Objetos con TensorFlow Lite,” 2020. https://github.com/tensorflow/examples/compare/master...cachvico:darren/fix-od.

[114] Google, “Repositorio TensorFlow,” 2015. https://github.com/tensorflow/.

[115] X. Wang, X. Duan, and X. Bai, “Deep sketch feature for cross-domain image retrieval,” *Neurocomputing*, vol. 207, pp. 387–397, 2016, doi: 10.1016/j.neucom.2016.04.046.

[116] G. Sun, L. Liang, T. Chen, F. Xiao, and F. Lang, “Network traffic classification based on transfer learning,” *Comput. Electr. Eng.*, vol. 69, pp. 920–927, 2018, doi: 10.1016/j.compeleceng.2018.03.005.

[117] R. J. Chalakkal, W. H. Abdulla, and S. S. Thulaseedharan, “Quality and content analysis of fundus images using deep learning,” *Comput. Biol. Med.*, vol. 108, no. March, pp. 317–331, 2019, doi: 10.1016/j.compbiomed.2019.03.019.

[118] V. Cheplygina, M. de Bruijne, and J. P. W. Pluim, “Not-so-supervised: A survey of semi-supervised, multi-instance, and transfer learning in medical image analysis,” *Med. Image Anal.*, vol. 54, pp. 280–296, 2019, doi: 10.1016/j.media.2019.03.009.

[119] D. Han, Q. Liu, and W. Fan, “A new image classification method using CNN transfer learning and web data augmentation,” *Expert Syst. Appl.*, vol. 95, pp. 43–56, 2018, doi: 10.1016/j.eswa.2017.11.028.

[120] B. Liu, X. Wang, R. Kwitt, and N. Vasconcelos, “Feature Space Transfer for Data Augmentation Mandar Dixit Microsoft,” *Cvpr*, pp. 9090–9098, 2018, [Online]. Available: http://openaccess.thecvf.com/content_cvpr_2018/papers/Liu_Feature_Space_Transfer_CVPR_2018_paper.pdf.

[121] F. L. Giambelluca, “Alarma detectora de escorpiones - Patent 20170100767,” 20170100767, 2017.

[122] Microchip, “PIC16F87X Data Sheet,” 2001.

[123] Microchip, “PIC16F818/819 Data Sheet,” pp. 1–390, 2004.

[124] HIKVISION EUROPE B.V., “HIKVISION Camera URL User Guide.”

[125] Blackfish Software, “IETab Documentation.” https://www.ietab.net/ie-tab-documentation.

[126] F. L. Giambelluca, M. A. Cappelletti, J. Osio, and L. A. Giambelluca, “Novel Automatic Scorpion Detection and Recognition System based on Machine Learning Techniques,” *Mach. Learn. Sci. Technol.*, Dec. 2020, doi: 10.1088/2632-2153/abd51d.

[127] R. Kumar, M. Sharma, K. Dhawale, and G. Singal, “Identification of Dog Breeds Using Deep Learning,” *Proc. 2019 IEEE 9th Int. Conf. Adv. Comput. IACC 2019*, pp. 193–198, 2019, doi: 10.1109/IACC48062.2019.8971604.

[128] J. Wang and P. Luis, “The Effectiveness of Data Augmentation in Image Classification using Deep Learning,” *Comput. Vis. Pattern Recognit.*, 2017.