Universidad Autónoma de Yucatán
Facultad de Matemáticas

Detección del parásito del Chagas usando una red neuronal de aprendizaje profundo

T E S I S

PRESENTADA POR:

Gabriel Alejandro May Lozano

EN OPCIÓN AL GRADO DE:

Licenciado en Ciencias de la Computación

TUTOR:

Anabel Martín González

Mérida, Yucatán, 2022

A la Facultad y a la Universidad, por la formación y visión que me han dado.
A mis compañeros de la licenciatura de matemáticas, por motivarme en explorar más mi campo.
A mis compañeros de licenciatura, por el apoyo mutuo para poder seguir adelante.
A mis amistades, por ser una fuente de fortaleza y motivación.
A Rin, por ser mi principal apoyo en esta etapa universitaria.
A Nucita e Iskah, por demostrarme que se debe superar en la vida y en lo personal.
A mi familia, por ser mi principal sustento durante mi formación profesional.
En verdad, sin sus acciones no se hubiera podido llegar a este punto.

Reconocimientos

Extiendo mis agradecimientos al Centro de Cómputo de FMAT, en especial al maestro Gavino Silva por sus autorizaciones para poder ejecutar parte de este trabajo en una computadora personalizada por mí. Y también por su personal , por mencionar a Gabi que memorizó mis horarios de trabajo.

También a mi asesora, que tuvo fe en mí a pesar de mis contratiempos, por su apoyo y dirección para seguir adelante en los proyectos académicos bajo su supervisión.

También agradecer a Miguel Canul por haber aportado en proyectos previos que sirvieron como base este trabajo.

Resumen

La enfermedad de Chagas es un problema de salud pública, causada por el parásito *Trypanosoma cruzi* (*T. cruzi*). Esta condición ha provocado miles de muertes por año atribuidas a problemas cardíacos, digestivos, neurológicos o mixtos, y debido a que la mayoría de las personas infectadas no presentan síntomas, se considera una enfermedad potencialmente mortal y es necesario un diagnóstico rápido para una intervención adecuada.

Un análisis de sangre resulta ser el método preferido para generar un diagnóstico de la enfermedad; sin embargo, es un proceso tardado, ya que requiere de mucho esfuerzo por parte de expertos para analizar grandes cantidades de muestras en búsqueda de dicho parásito.

La implementación de sistemas automáticos que faciliten la detección del *T. cruzi* en imágenes de muestras de sangre capturadas por microscopio puede ser de gran utilidad para ahorrar tiempo y esfuerzo en el diagnóstico de la enfermedad.

Por lo tanto, en este trabajo de tesis se propone la implementación de una red neuronal convolucional como modelo de detección del parásito de la enfermedad del Chagas en imágenes de muestras sanguíneas.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Un diagnóstico médico es una de las tareas más complejas de realizar ya que se requiere del análisis de muchos factores como: anamnesis (es la información proporcionada por el paciente durante la entrevista clínica útil para analizar su situación clínica), síntomas, signos, exploración física, entre otros (1).

El análisis de dichos factores, en conjunto con la experiencia del médico, da como resultado un diagnóstico, el cual puede ser presuntivo o definitivo, y en ciertos casos, tiene que ser confirmado por exámenes médicos para llegar a un diagnóstico definitivo y poder tratar dicha afección. Por lo tanto, el uso de la tecnología para facilitar el diagnóstico de enfermedades es una necesidad en el siglo XXI, ya que la creciente población y demanda de los servicios médicos sobrepasa la capacidad de atención de los hospitales, dando como resultado tiempos de espera más largos y menos tiempo para la atención del paciente.

El uso de algoritmos de aprendizaje automático para el diagnóstico de enfermedades pueden reducir los tiempos de consulta. En afecciones muy comunes, estos algoritmos podrían incluso evitar la revisión de un especialista, ya que por medio de sistemas automatizados se podría dar un diagnóstico certero y un tratamiento óptimo.

En las zonas geográficas muy marginadas, que no cuentan con especialistas, un sistema computacional manejado por un técnico de la salud podría diagnosticar y tratar dichas enfermedades sin la necesidad de que el paciente se traslade (2).

1.1. Problemática

La enfermedad de Chagas es una afección parasitaria y potencialmente mortal causada por el parásito *Trypanosoma cruzi* o *T. cruzi*. De acuerdo con la Organización Mundial de la Salud (OMS), se calcula que en el mundo hay entre seis y siete millones de personas infectadas por el parásito *T. cruzi*. En 2005, la OMS reconoció que la enfermedad de Chagas era una enfermedad tropical desatendida (3).

La infección se transmite comúnmente a través de la picadura de un insecto triatomino infectado por el parásito. Este insecto triatomino, también conocido como chinche, habita principalmente en áreas rurales de América del Sur, América Central y México, siendo este último su principal hogar. Por otro lado, existen otras vías de transmisión como lo son congénita, trasplante de órganos o transfusión de sangre.

La enfermedad de Chagas tiene dos etapas o fases clínicas: una fase aguda y una fase crónica. Muchas personas (del 70 % al 80 % de los infectados) son asintomáticas toda su vida, pero entre un 20 % y 30 % de los afectados evolucionan a cuadros crónicos sintomáticos asociados a lesiones en el corazón, tubo digestivo y/o sistemas nerviosos (4).

El diagnóstico definitivo de infección por *T. cruzi* depende del resultado positivo de al menos dos pruebas serológicas diferentes, como son ELISA, Inmunofluorescencia Indirecta o Hemaglutinación Indirecta, que detectan anticuerpos específicos en el suero del paciente.

La enfermedad de Chagas puede tratarse para erradicar el parásito del sistema huésped, sin embargo es necesario detectarlo durante el inicio de la fase aguda dado que que un gran número de parásitos se encontrarían circulando en la sangre. La detección de parásitos a través de inspección microscópica de frotis de sangre es una técnica muy común y conveniente, pero en ocasiones, es un proceso que requiere de mucho tiempo y esfuerzo, ya que involucra el análisis de muchas muestras de sangre, provocando el retraso para obtener un diagnóstico eficaz y un pronto tratamiento de la enfermedad (5).

1.2. Estado de la técnica

El término «*estado de la técnica*» es utilizado en este trabajo, así como las expresiones *estado actual* o *últimos avances*, en sustitución de la frase *estado del arte*, calco en inglés de *state-of-the-art*, según sea el contexto de la lectura y siguiendo las recomendaciones hechas por la Real Academia de la Lengua Española (6).

Se han presentados propuestas de identificación del parásito *T. cruzi* mediante el uso de sistemas automáticos con el objetivo de eliminar o reducir el error humano en la observación de grandes volúmenes de muestras de sangre, especialmente durante la fase inicial de la enfermedad y en zonas altamente endémicas. Sin embargo, en el área de aprendizaje automático y de visión por computadora se han reportado un número limitado de trabajos para la detección y/o segmentación del parásito en imágenes de muestras sanguíneas.

1.2.1. Métodos basados en aprendizaje automático

En 2013, Uc, Brito y Ruiz proponen la detección del parásito de Chagas en imágenes de muestras de sangre mediante un algoritmo basado en el análisis del discriminante gaussiano (7). Su método consiste en analizar la información presente en el canal verde de las imágenes para extraer un vector con 121 características relevantes en zonas de la imagen donde existan posibles núcleos de parásitos. El problema de clasificación consiste en distinguir si el vector de características corresponde a un agente infeccioso de Chagas o no. Para ello, los autores implementaron el análisis de discriminante de Gauss para la construcción de dos modelos: $p(x|y = 1)$, que modela la distribución de características de lo que es parecido al parásito ($y = 1$), y $p(x|y = 0)$, que modela la distribución de características de lo que no es parecido al parásito ($y = 0$).

Los autores propusieron tres distintos modelos de experimentos, donde el experimento número 2 obtuvo el mejor resultado aplicando el análisis de componentes principales (PCA) para reducir el tamaño del vector de características de entrada. Las tasas de rendimiento final que reportaron fueron 0.0167 en falsos negativos, 0.1563 en falsos positivos, 0.8437 en verdaderos negativos y 0.9833 en verdaderos positivos.

En el trabajo presentado por Soberanis, Uc, Brito y Ruiz (8) se propone un algoritmo automatizado para detectar el agente infeccioso usando diversas técnicas de segmentación y clasificación en imágenes de muestras de sangre. Su base de datos cuenta con 120 imágenes, de las cuales 60 tienen presencia de parásitos.

Este trabajo fue desarrollado tres etapas. En una primera etapa de preprocesamiento aplican una máscara con el fin de conservar información presente del cuerpo del parásito mediante una umbralización sobre la imagen que contiene la diferencia de los canales azul y verde, previamente calculada. La etapa de segmentación consiste en separar el parásito del fondo con un clasificador gaussiano se entrena con ejemplos positivos (lo que representa un parásito) y negativos (lo que no lo representa).

En una última etapa las imágenes que se mantuvieron en la intersección pasan por

un clasificador de K vecinos más cercanos (9) previamente entrenado para realizar una clasificación binaria (si es o no es parásito). Después del entrenamiento y pruebas los autores reportaron 98 % de sensibilidad y 85 % de especificidad.

En 2014, Soberanis(10) propone una comparación de tres clasificadores, entre las máquinas de soporte vectorial (del inglés, *Support Vector Machine*, o *SVM*) (11), *AdaBoost* (12) y redes neuronales artificiales; para detectar y segmentar el parásito *T. cruzi* con el uso de la técnica de super pixeles. Los súper pixeles están formados por conjuntos de pixeles que describen regiones continuas delimitadas a lo largo de la imagen, y en su propuesta permite que se extraigan características considerando el vecindario y la relación de los pixeles dentro de un grupo.

Su metodología consiste en tres etapas, donde la primera se realiza un cómputo de súper pixeles, en la segunda se extraen las características óptimas, y en la última se ejecuta el entrenamiento de cada clasificador: redes neuronales usando el algoritmo de retro propagación; *AdaBoost* usando ensamble de perceptrones; y máquinas de soporte vectorial. Su base de datos empleada consiste en 900 imágenes de ejemplos positivos (con parásitos) y 900 de negativos (sin parásitos).

En una primera etapa de la experimentación se escoge la configuración del vector de características que mejor desempeño obtenga en los tres clasificadores propuestos por Soberanis. Posteriormente, compara el desempeño de sus clasificadores contra tres algoritmos en el estado del arte: clasificador gaussiano, clasificador de Bayes y el algoritmo propuesto por Soberanis en 2012 (13).

Al finalizar la experimentación se reporta que el clasificador gaussiano presentó el menor error cuadrático medio con 0.18568, seguido del clasificador *SVM* de Soberanis (con un tamaño de súper pixel de 100) que obtuvo 0.22635 y, por último, el clasificador usando redes neuronales con un error de 0.361.

En el trabajo de Uc, Brito y Ruiz (14) se realiza una comparación de dos algoritmos robustos, *AdaBoost* y *SVM* para la tarea de detección de *T. cruzi* en imágenes de muestras de sangre. Se hace uso de plantillas Haar (15) propuestas en este trabajo, que están diseñadas para representar la morfología común del parásito de *T. cruzi*. Su base de datos cuenta con 120 imágenes, de las cuales 60 tienen presencia de parásitos. Los autores realizaron una comparación del proceso de detección (*AdaBoost* + *SVM*) con una implementación para reconocer el parásito de Chagas usando un *único* clasificador *SVM*.

La clasificación usando AdaBoost más un postprocesamiento obtuvo los mejores resultados y reportó 100 % de sensibilidad y 93.25 % de especificidad.

En estos trabajos previos, se implementan algoritmos de aprendizaje automático que

se entrenan con características que los autores proponen después de analizar las imágenes. Si bien para algunos de estos trabajos ha resultado adecuado definir empíricamente las características a utilizar para la detección o segmentación del parásito, existen otras metodologías pertenecientes al área de aprendizaje profundo que aprenden a extraer automáticamente las características más importantes y las procesan según su requerimiento; de esta forma, se evita tener que proponer manualmente el conjunto de características que alimentarán al algoritmo de detección o segmentación (16).

Aunque estos trabajos no destacan por tener un alto desempeño de detección y reportan un elevado valor de falsos negativos (lo que indicaría que una persona no esté infectada cuando si lo está), puede ser considerado como un buen comienzo para apoyar en el diagnóstico de la enfermedad de Chagas.

1.2.2. Métodos basados en aprendizaje profundo

Recientemente, los algoritmos basados en las redes neuronales convolucionales de aprendizaje profundo para el análisis de imágenes biomédicas, se han convertido rápidamente en una metodología de uso debido al alto desempeño que se ha obtenido en los resultados de clasificación, detección y segmentación de objetos. Sin embargo, estas técnicas no han sido exploradas para la detección de la presencia de *T. cruzi*, donde el reto principal es contar con un gran número de imágenes de entrenamiento para que el algoritmo pueda aprender y obtener los resultados de detección y/o segmentación del parásito causante de la enfermedad estudiada (5).

Un aporte con la implementación de redes convolucionales se encuentra en el trabajo presentado en 2022 por Ojeda (17), donde plantea el uso de una red neuronal completamente convolucional para la segmentación del parásito de Chagas en imágenes de muestras de sangre, estando inspirada en la arquitectura *ResUnet* (18). Su modelo nombrado como *Res2Unet*, toma ventaja del modelo *U-Net* y de las conexiones de redes residuales (19), donde introduce las conexiones residuales dobles con el fin de mejorar la propagación de la información para un mejor rendimiento de segmentación.

Su base de datos consistió en 974 imágenes en espacio de color RGB y de resolución 2560× 1920 píxeles, donde se obtuvo un subconjunto de 1040 imágenes de tamaño 256 × 256 que fueron recortadas de la base de datos original. De este subconjunto de imágenes, 978 imágenes contenían parásitos y 62 imágenes no. Cabe mencionar que su subconjunto de imágenes fueron divididos en tres particiones: prueba, validación y entrenamiento, aplicando aumentación de datos a este último.

Se comparó el rendimiento del modelo *Res2Unet* propuesto con las arquitecturas *U-Net* y *ResUnet*, donde cada modelo se entrenó con las funciones de costo: entropía cruzada binaria, entropía cruzada binaria ponderada, entropía cruzada y de contornos activos. Para la evaluación de este rendimiento, se seleccionó el puntaje del coeficiente *Dice* (del inglés *Dice Coefficient*, DC) (20), el puntaje de intersección sobre la unión (IoU), los valores de precisión y de exhaustividad.

Los resultados obtenidos mostraron que el mejor modelo fue la propuesta con la función de costo de contornos activos, teniendo el mejor coeficiente *Dice*, reportando 0.84; y la mejor puntuación IoU, de 0.73, así como con los valores balanceados de precisión, reportando 0.80; y exhaustividad, de 0.76.

La propuesta de este proyecto busca una aportación dentro del estado de la técnica en la búsqueda y detección del parásito *T. cruzi* en imágenes de tomas de sangre.

1.3. Objetivos

Objetivo general

Este trabajo tiene por objetivo general la detección del parásito *Trypanosoma cruzi* en imágenes de muestras de sangre mediante la implementacion de una red neuronal de aprendizaje profundo.

Objetivos específicos

Los objetivos específicos de este trabajo son los siguientes:

- Etiquetar las imágenes de la base de datos para el entrenamiento de la red neuronal.
- Implementar el algoritmo de red neuronal *You Only Look Once* ($YOLO$) para la detección del parásito en imágenes.
- Comparar resultados obtenidos con algoritmos del estado de la técnica

Capítulo 2

Marco teórico

El desarrollo de este trabajo se encuentra fundamentado en dos temas fundamentales: el parásito causante de la enfermedad del chagas y los conceptos de una red neuronal de aprendizaje profundo.

2.1. Enfermedad de Chagas

Una enfermedad parasitaria o parasitosis es una enfermedad infecciosa causada por protozoos, vermes (cestodos, trematodos, nematodos) o artrópodos. Las parasitosis son estudiadas por la parasitología. No se consideran parasitosis las infecciones por hongos, bacterias o virus que, tradicionalmente, han sido estudiados por la microbiología.

Hay muchos tipos de análisis clínicos para diagnosticar enfermedades parasitarias. El tipo de análisis que solicite el médico especialista se basará en los signos y síntomas de pacientes afectados, para poder aplicar el tratamiento adecuado para combatir la enfermedad. El diagnóstico puede ser difícil, por lo que el desarrollo de nuevas técnicas de análisis es utilizadas para ayuda esa labor y proporcionar un tratamiento optimo a los pacientes.

La enfermedad de Chagas es una afección parasitaria, sistémica, crónica, transmitida por vectores y causada por el parásito *Trypanosoma cruzi*, abreviado como *T. cruzi*. La enfermedad lleva el nombre de Carlos Ribeiro Justiniano Chagas, médico e investigador brasileño que la descubrió en 1909.

Es una patología endémica en 21 países del continente americano, aunque las migraciones de personas infectadas pueden llevarla a países no endémicos de América y el Mundo.

La enfermedad tiene una mayor relevancia en las zonas rurales más pobres de América Latina, por lo que mantiene una firme vinculación con aspectos socio-económicos y culturales deficitarios, considerándosela una enfermedad desatendida (4).

El principal mecanismo de transmisión es vectorial, por hemípteros (chinches), de la subfamilia *Triatominae* de alimentación hematófaga (Ver Figura 2.1). Infectan personas expuestas a su picadura, al depositar sus heces infectadas en heridas de la piel o sobre mucosas. Otras modalidades de transmisión son transfusional, congénita, transplantes de órganos u oral. Aunque la mortalidad ha disminuido significativamente, la enfermedad puede causar consecuencias irreversibles y crónicas en el corazón, el sistema digestivo y el sistema nervioso.

Figura 2.1: Fotografía de un insecto triatomino (21)

El *T. cruzi* es un protozoario flagelado digenético del orden *Kinetoplastida* subgénero Schizotrypanum (ver Figura 2.2). El ciclo biológico se completa al infectar la sangre y otros tejidos de los reservorios (huépedes) y en el tubo digestivo del vector, donde sufre una serie de tranformaciones para poder seguir multiplicandose y seguir con la invasión al huésped.

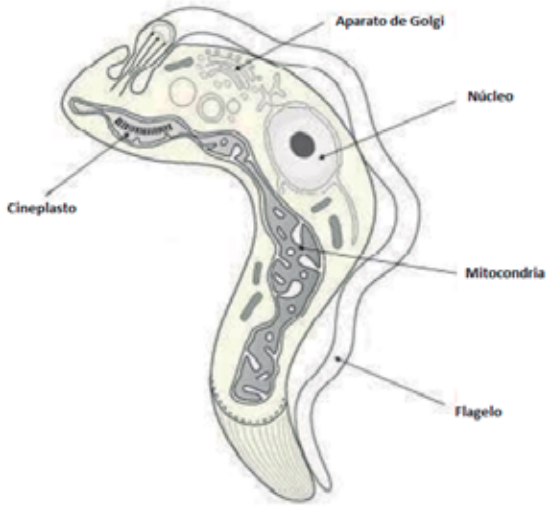


Figura 2.2: Estructura del parásito *T. cruzi* (22)

Diagnóstico

El diagnóstico de Chagas siempre es clínico, epidemiológico y de laboratorio (parasitología y serología).

Durante la fase aguda, la enfermedad de Chagas se puede diagnosticar mediante métodos parasitológicos en sangre, dada la riqueza de parasitemia. En la etapa aguda los estudios se centran en la búsqueda y reconocimiento del *T. cruzi* en un examen directo y tinción de extendidos de sangre (muestras sanguineas, ver Figura 2.3), y en determinar la seropositivización de la serología.

Para la etapa crónica de la enfermedad, el diagnóstico se fundamenta en la evaluación clínica, serología y antecedentes epidemiológicos.

En caso de contacto con estos insectos triatominos, es de suma importancia realizar estudios en búsqueda de parásitos en los primeros meses para obtener diagnósticos eficaces y un pronto tratamiento en caso de haber adquirido la enfermedad de Chagas. Se han reportado diferentes métodos para el diagnóstico de la enfermedad, donde se encuentra los métodos clínicos y recientemente los apoyados mediante el aprendizaje automático.

Un análisis de sangre es esencial para detectar y tratar con oportunidad el padecimiento. Existen otros métodos que requieren pruebas portátiles especializadas para la detección del parásito *T. cruzi* en una muestra de sangre. Uno de ellos es el denominado *Chagas*

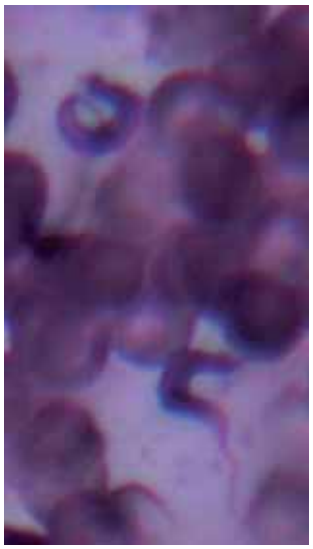


Figura 2.3: Presencia de parásitos *T. cruzi* en una muestra sanguínea

Stat-Pak, presentado por Ponce en 2005 (23), cuyo método tiene 99.6 % de sensibilidad y 99.9 % de especificidad; este rendimiento es comparable al que se obtiene con una prueba de *ELISA*. Otro método que requiere una prueba portátil es el denominado *Chagas Detect Plus* (24). Si bien estas pruebas portátiles para diagnosticar la enfermedad de Chagas son rápidas y eficaces, obligatoriamente se requiere *algún* método manual o serológico para emitir un diagnóstico preciso. En caso de un resultado positivo, el paciente debe iniciar el tratamiento correspondiente (24).

Para estos análisis clínicos, un método de aprendizaje automático podrá ayudar a automatizar las actividades de observación bajo microscopio, ya que engloba técnicas y herramientas tecnológicas que permite a las computadoras realizar tareas complejas con alta exactitud (25).

Contar con un sistema automático que identifique al parásito *T. cruzi* podría eliminar o reducir el error humano en la observación de grandes volúmenes de muestras de sangre, especialmente durante la fase inicial de la enfermedad y en zonas altamente endémicas.

2.2. Procesamiento de imágenes

El concepto de procesamiento de imágenes digitales (del inglés, *digital image processing*) es referido como a los procesos realizados por una computadora digital cuyas entradas y salidas son imágenes digitales. Algunos de estos procesos son el realce (del inglés, *enhancement*), filtrado y restauración de imágenes.

2.2.1. Imágenes digitales

Una imagen puede ser definida como una representación visual, un calco, dibujo o alguna percepción que se manifesta sobre un objeto o una situación. Puede ser una fotografía de una o varias personas, de animales, de una escena al aire libre, de una microfotografía de un componente electrónico, o el resultado de imágenes médicas. Incluso si la caracterización no es inmediatamente reconocible, no será solo un desenfoque aleatorio.

Una imagen digital está compuesta de un número finito de elementos, donde cada elemento tiene una localidad y un valor discreto. Estos elementos son conocidos como pixeles. El dominio de una imagen digital como una función $I(x, y)$ está dado por las dimensiones de la imagen en valores enteros y el punto inicial de la imagen está ubicado en la esquina superior izquierda de una imagen en la posición $(0, 0)$. La representación matemática de una imagen digital como un arreglo I de dimensiones $m \times n$ es de la siguiente forma:

$$I_{m,n} = \begin{bmatrix} p_{0,0} & p_{0,1} & \cdots & p_{0,n-1} \\ p_{1,0} & p_{1,1} & \cdots & p_{1,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m-1,0} & p_{m-1,1} & \cdots & p_{m-1,n-1} \end{bmatrix}$$

donde n es la cantidad de columnas de pixeles (ancho de imagen), m es la cantidad de filas de pixeles (altura o alto de imagen), y a es un pixel de la imagen con su correspondiente valor de intensidad y siendo frecuentemente codificado como un valor sin signo de 8 bits. El tamaño de la imagen denotado como $m \times n$ es referido como la resolución de la imagen, siendo este un dato fijo y se relaciona directamente con la calidad de una imagen.

Cuando el valor de un pixel es un único valor, entonces se habla de una imagen en escalas de grises, donde el valor del pixel representa un valor de escala de gris. Este valor

codificado en 8 bits genera 256 valores de intensidad donde el valor 0 representa el color negro, el valor 255 al color blanco; y los valores intermedios reprensentan tonalidades de gris creciente. En la Figura 2.4 se muestra un ejemplo de imagen en escalas de grises y valores correspondientes.

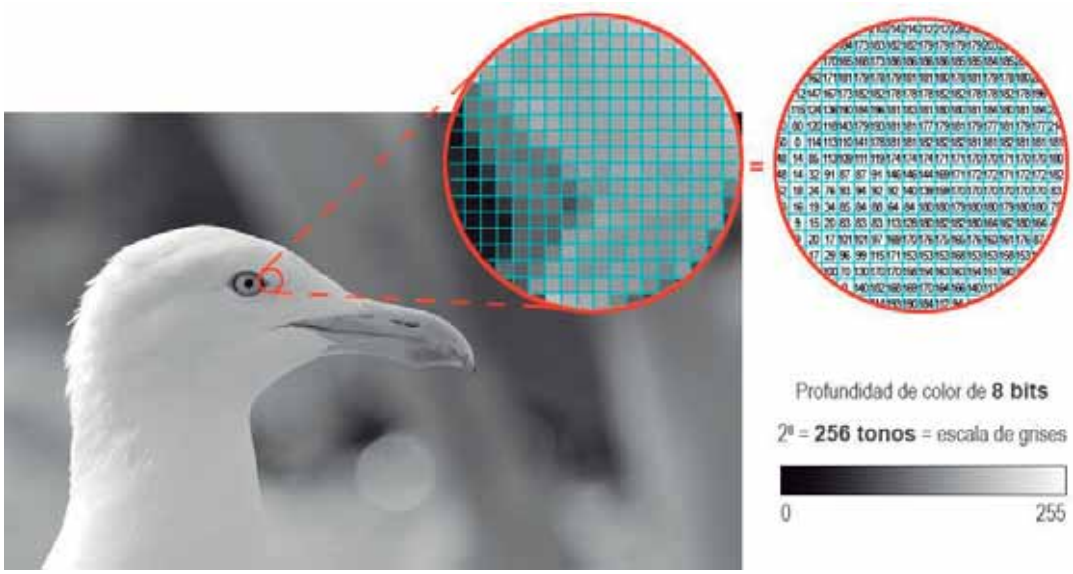


Figura 2.4: Representación gráfica de una imagen en escalas de grises (26)

Una imagen digital a color tiene una representación similar, donde el valor contenido en cada pixel es descrito y codificado en base a un espacio de color a utilizar. Un espacio de color es un sistema de interpretación del color, es decir, una organización específica de los colores en una imagen o video.

El espacio de color más utilizado es el *RGB* (sigla en inglés de *Red*-Rojo, *Blue*-Azul y *Green*-Verde), donde el contenido de cada pixel representa un vector de valores correspondiente a los colores en el orden rojo, azul y verde, originado un color de esta combinación en la imagen (Ver Figura 2.5).

Un pixel de una imagen *RGB* cuyo valor de intensidad de un color es de 8 bits puede generar, al igual a una imagen en escalas de gris, 256 valores de intensidad, pero al tratarse de 3 colores, en total la imagen a color es capaz de producir hasta $16,777,216$ tonalidades de colores diferentes.

Dado al gran número de tonalidades que imagen *RGB* genera, una representación

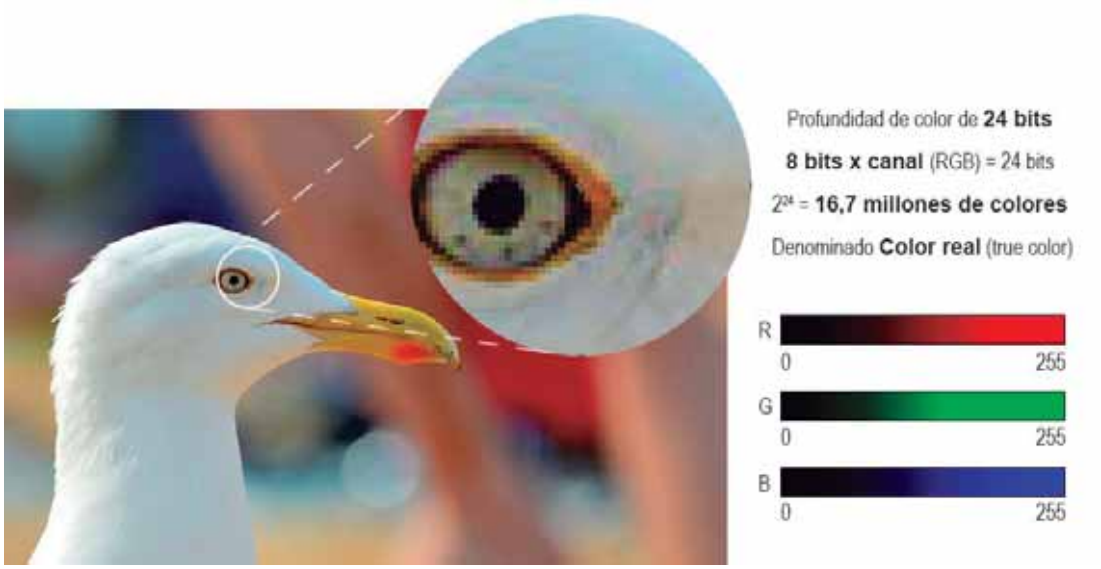


Figura 2.5: Representación computacional de una imagen en escalas de grises (26)

general de esta es mediante un arreglo tridimensional de tamaño $m \times n \times 3$, donde cada tercera dimensión corresponde a un canal de los colores en el orden rojo, verde y azul (Ver Figura 2.6).

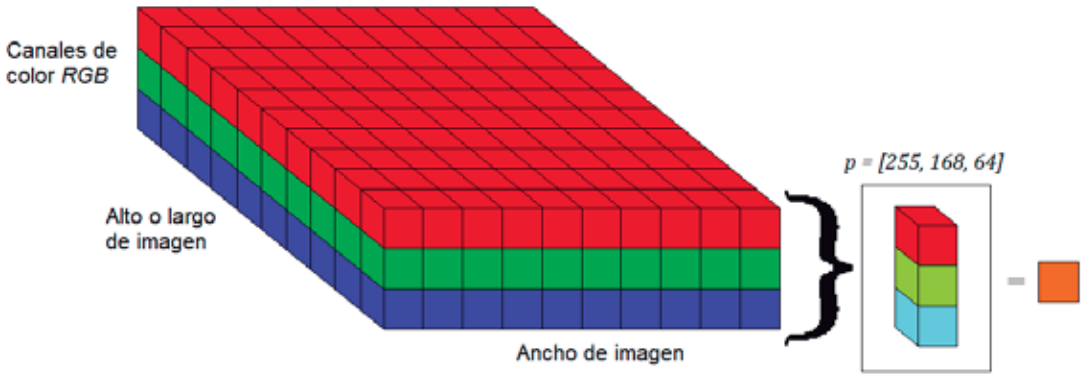


Figura 2.6: Ejemplo de visualización de una imagen como matriz tridimensional

2.2.2. Métodos de dominio espacial

Los métodos del procesado de imágenes se dividen en algoritmos de dominio de frecuencia (del inglés, *frequency domain*) y en algoritmos de dominio espacial (del inglés, *spatial domain*).

El término dominio espacial se refiere al plano de la imagen en la que se encuentra, y los métodos de procesamiento de imágenes de esta categoría se basan en la manipulación directa de cada pixel o un grupo de estos, denominado vecindario, de una imagen.

Se define como vecindario de un pixel p al conjunto de localizaciones de una imagen vecinos a dicho pixel. Generalmente, un pixel p con coordenadas (x, y) funge como centro de una pequeña región de la imagen mucho más pequeña de la misma y de forma rectangular para construir su vecindario. En la Figura 2.7 se visualiza un ejemplo de vecindario de una imagen

$p(x-1, y-1)$	$p(x, y-1)$	$p(x+1, y-1)$
$p(x-1, y)$	$p(x, y)$	$p(x+1, y)$
$p(x-1, y+1)$	$p(x, y+1)$	$p(x+1, y+1)$

Figura 2.7: Ejemplo de vecindad de tamaño 3×3 de un pixel p de una imagen

Los métodos de dominio espacial están basados en la expresión

$$g(x, y) = T\left[I(x, y)\right] \tag{2.1}$$

donde $I(x, y)$ es una imagen de entrada, $g(x, y)$ es una imagen de salida, y T es un operador sobre f definido sobre un vecindario del punto (x, y). Si el vecindario es del tamaño 1×1 (el menor posible); en este caso g depende solo del valor de f en el único punto (x, y) y T se convierte en una función de transformación de intensidad (sea nivel de gris o de color) de la forma

$$s = T(r)$$

donde s y r representan respectivamente la intensidad de g y f en cualquier punto (x, y).

Filtrado de imágenes

El filtrado de imágenes es una operación espacial que modifica una imagen reemplazando el valor de cada píxel por una función de los valores del píxel y sus vecinos denominado filtro. Si la operación realizada en los píxeles de la imagen es lineal, entonces el filtro se denomina filtro espacial lineal, de lo contrario, es un filtro espacial no lineal.

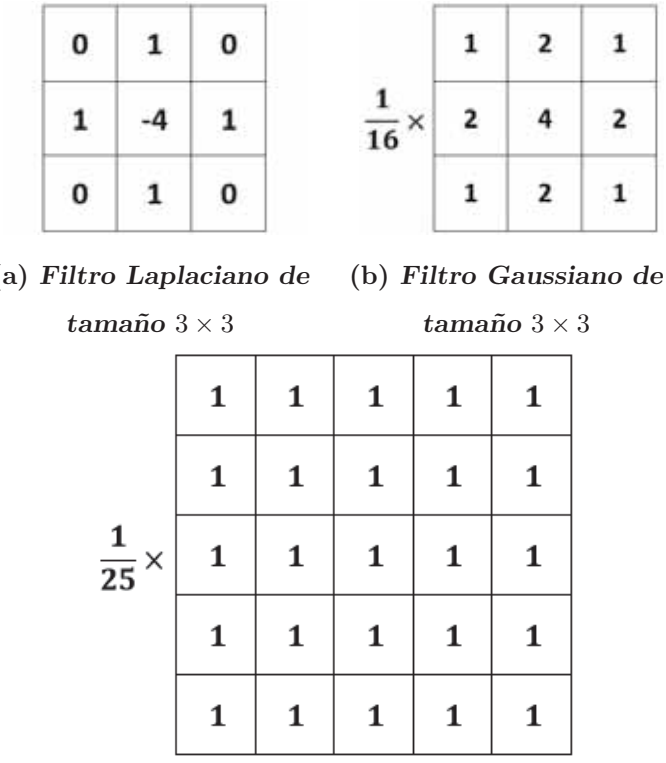
Un filtro espacial lineal realiza una operación de suma de productos entre una imagen f y un filtro k denominado como *kernel* (traducido como filtro núcleo del inglés). El *kernel* simplemente es una matriz cuyo tamaño define la vecindad de la operación y cuyos coeficientes determinan la naturaleza del filtro. Otros términos utilizados para referirse a un *kernel* como filtro espacial son máscara, plantilla, ventana y filtro *kernel*.

El *kernel* generalmente es definido como una matriz cuadrada de tamaño impar, siendo los de tamaño 3×3, y el 5×5 los más usados. De igual forma, se tiene que definir una tercera dimensión dependiendo del dato a convolucionar, siendo que para imágenes en escalas de grises una profundidad de 1 (*kernel* bidimensional), y para las imágenes *RGB* una profundidad de 3 (*kernel* tridimensional). Ejemplos se pueden visualizar en la Figura 2.8.

En general, el filtrado espacial lineal de una imagen de tamaño $M \times N$ con un *kernel* cuadrado de tamaño n esta dada por la expresión

$$F(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} k(s, t) I(x + s, y + t) \tag{2.2}$$

donde x y y varían de modo que el centro (origen) del *kernel* visita cada pixel en f una vez. Para un valor fijo de (x, y), F implementa la suma de productos de forma que el



(a) ***Filtro Laplaciano de tamaño*** 3×3

(b) ***Filtro Gaussiano de tamaño*** 3×3

(c) ***Filtro de la media de tamaño*** 5×5

Figura 2.8: Ejemplos de *kernels* bidimensionales para el filtrado de imágenes

centro del *kernel* empata con el valor fijo y así respectivamente para sus vecinos laterales y diagonales, pero para un núcleo de tamaño impar arbitrario.

La ecuación 2.2 también describe matemáticamente la correlación espacial, que consiste en mover el centro de un núcleo sobre una imagen y calcular la suma de productos en cada ubicación. En la Figura 2.9 ilustra tanto el filtrado espacial como la correlación con un *kernel* de tamaño 3.

La operación de la convolución es similar, con la diferencia en que el *kernel* de la correlación es rotado 180°. Por lo tanto, cuando los valores de un núcleo son simétricos con respecto a su centro, la correlación y la convolución dan el mismo resultado.

Con respecto a la expresión de la correlación de manera similar, la convolución de un

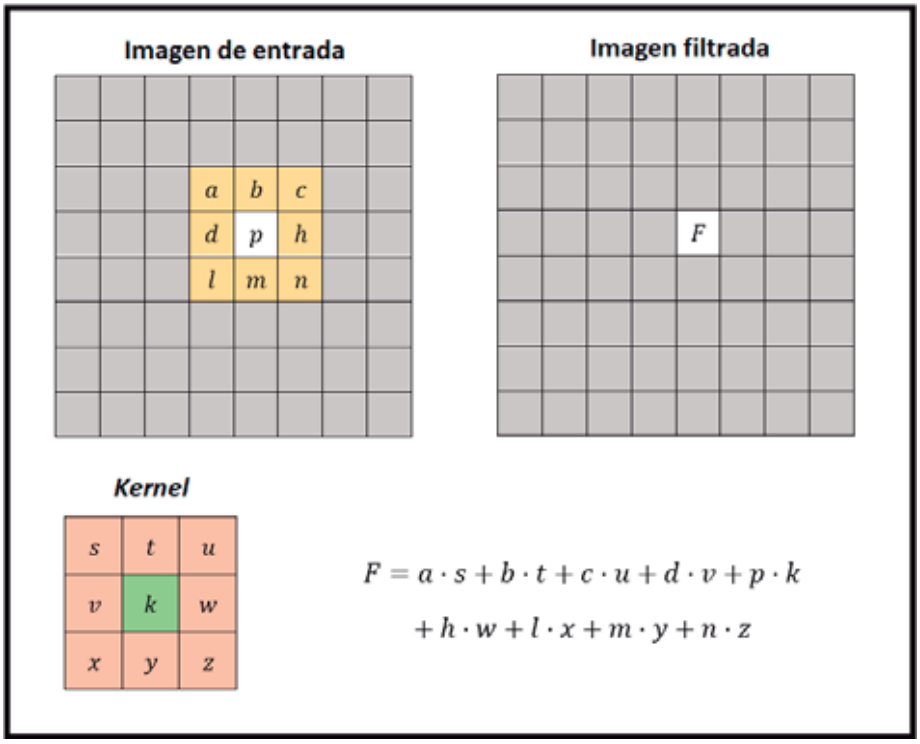


Figura 2.9: Visualización de filtrado con un *kernel* de tamaño 3

kernel k de tamaño $m \times m$ con una imagen $f(x, y)$, es expresada matemática como

$$\text{Conv}(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} k(s, t) f(x - s, y - t) \tag{2.3}$$

donde los signos de resta alinean las coordenadas de f y k cuando una de las funciones es rotada 180°. En la Figura 2.10 ilustra el proceso de la convolución.

La operación de convolución se repite para todos los pixeles de la imagen conforme el filtro de convolución recorre la imagen. Al tamaño del salto que realiza el filtro en la convolución se conoce como paso o, del inglés, *stride*. Con un salto de uno, el filtro avanza por la imagen realizando un salto entre cada columna de píxel. Si se aumenta el número de saltos, entonces la imagen de salida tendrá dimensiones aún más cortas. Un ejemplo de paso es visualizado en la Figura 2.11.

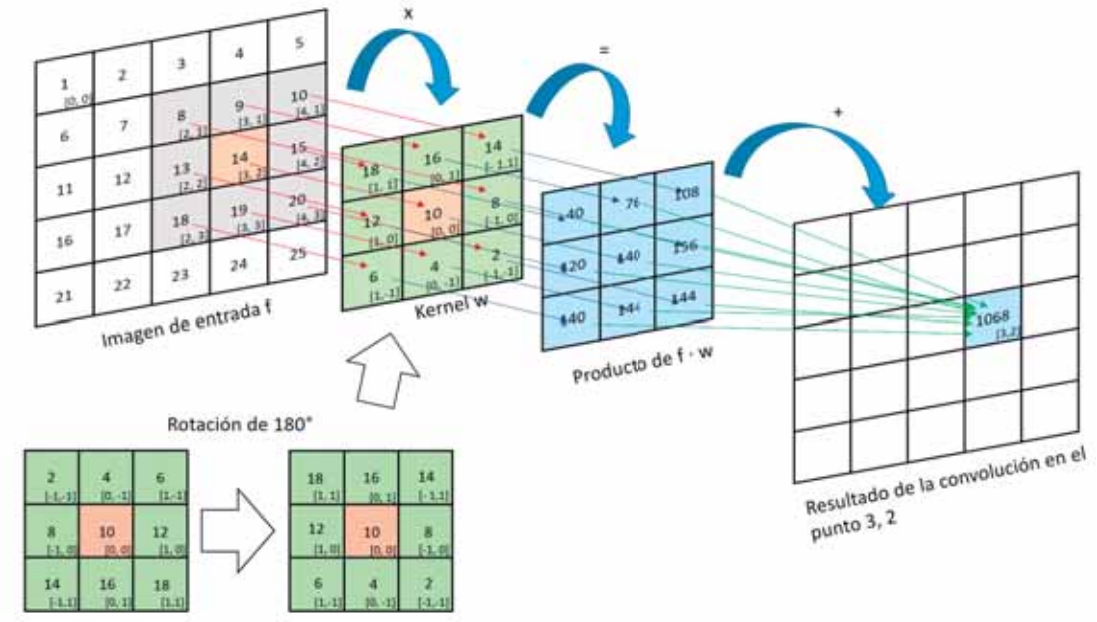


Figura 2.10: Visualización de la operación de convolución sobre un imagen. (27)

El cálculo de la operación toma como punto de inicio al pixel que empata con el centro del *kernel*, cumpliendo que todo la ventana es totalmente cubierta por una región de la imagen, pero se ignora el procesamiento de pixeles cuyos vecindarios no cumplen el tamaño del *kernel*, siendo principalmente pixeles cercanos a los bordes. Por lo que para «acompletar» esos vecindarios, se aplica el rellenado o, del inglés, *padding* de bordes de la imagen con valores definidos, para procesar todos los pixeles ignorados en el borde, y tomando como punto inicial de la convolución sería el pixel $(0, 0)$.

El número de bordes a rellenar depende del tamaño del *kernel*, donde su tamaño m implicar agregar $\frac{m-1}{2}$ filas y columnas a los bordes de la imagen, cuyos valores son determinados según el método empleado. Algunos de los métodos empleado son como el rellenado de ceros (del inglés *zero padding* y el más utilizado) y el rellenado espejo (replicando los valores de los bordes de la imágen). En la Figura 2.12 se presenta los métodos anteriormente mencionados de rellenados de imágenes.

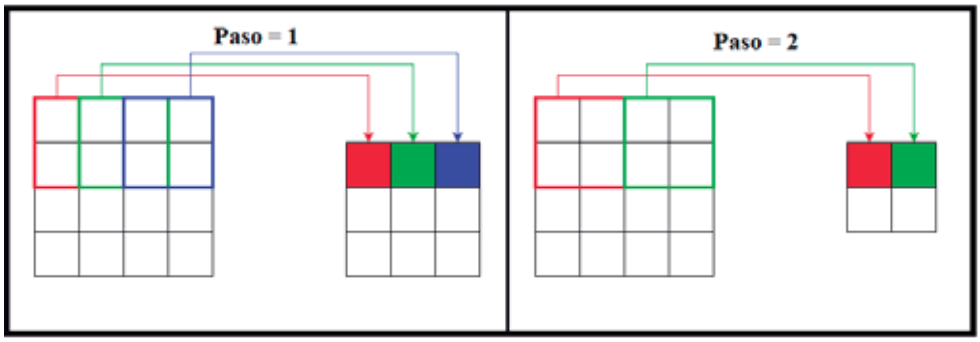


Figura 2.11: Ejemplo de aplicación de uno y dos pasos (28)

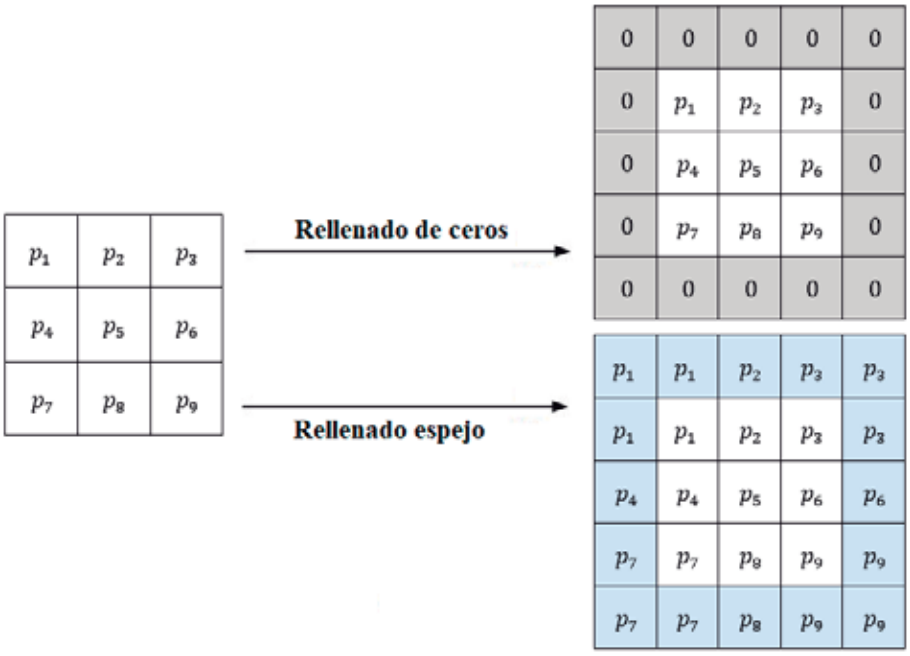


Figura 2.12: Ejemplo de aplicación de rellenado de bordes

El filtrado y convolución de imágenes es empleado para diversas aplicaciones, como la minimización de ruido, la aproximación de derivadas, detección de bordes en imágenes, entre otras (29).

2.3. Aprendizaje automático

El aprendizaje automático (también llamado aprendizaje máquina, del inglés *machine learning*) es una colección de algoritmos y herramientas dentro del campo de la inteligencia artificial (IA) que está interesado en responder cómo una computadora puede "aprender" tareas específicas de las observaciones del mundo real, tales como reconocer caracteres, apoyar el diagnóstico de personas con enfermedades graves, clasificar tipos de vino, separar algún material según su calidad (por ejemplo, la madera podría separarse según su debilidad, por lo que podría usarse más tarde para construir lápices o casas). Esas y muchas otras aplicaciones evidencian la utilidad del aprendizaje máquina para abordar problemas cotidianos y apoyar a los especialistas en la toma de decisiones, atrayendo a investigadores de distintas áreas del conocimiento (30)

El campo de aprendizaje automático generalmente se divide en tres categorías: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje reforzado. El aprendizaje supervisado utiliza conjuntos de datos etiquetados para categorizar o hacer predicciones; esto requiere algún tipo de intervención humana para etiquetar correctamente los datos de entrada. Por el contrario, el aprendizaje no supervisado no requiere conjuntos de datos etiquetados y, en cambio, detecta patrones en los datos, agrupándolos por cualquier característica distintiva. El aprendizaje por refuerzo es un proceso en el que un modelo aprende a ser más preciso para realizar una acción en un entorno basado en la retroalimentación para maximizar la recompensa.

Los algoritmos de aprendizaje automático aprovechan datos estructurados y etiquetados para hacer predicciones, lo que significa que las características específicas se definen a partir de los datos de entrada para el modelo y se organizan en tablas. Esto no significa necesariamente que no utilice datos no estructurados; solo significa que, si lo hace, generalmente pasa por un procesamiento previo para organizarlo en un formato estructurado (31).

2.3.1. Aprendizaje profundo

El aprendizaje profundo (del inglés, *deep learning*) es un enfoque especial del aprendizaje automático que cubre las tres categorías en las que se divide, y busca también extenderlas para abordar otros problemas de inteligencia artificial que generalmente no se incluyen en el aprendizaje automático, como la representación del conocimiento, el razonamiento, la

planificación, entre otras aplicaciones (32). Se intenta imitar el cerebro humano, aunque lejos de igualar su capacidad, lo que permite que los sistemas agrupen datos y hagan predicciones con una precisión increíble.

El aprendizaje profundo se distingue del aprendizaje automático clásico ya que elimina parte del preprocesamiento de datos que suele estar relacionado con el aprendizaje automático. Estos algoritmos pueden ingerir y procesar datos no estructurados, como texto e imágenes, y automatizan la extracción de funciones, eliminando parte de la dependencia de expertos humanos.

Desde la década de 2010, los avances tanto en los algoritmos de aprendizaje automático como en el hardware de la computadora han llevado a métodos eficientes para entrenar redes neuronales profundas que contienen muchas capas de unidades ocultas no lineales y una capa de salida muy grande. Para 2019, las unidades de procesamiento gráfico (GPU), a menudo con mejoras específicas para IA, habían desplazado a las CPUs como el método dominante para entrenar la IA en la nube comercial a gran escala (33).

2.3.2. Redes neuronales artificiales

Las redes neuronales artificiales (del inglés, *artificial neural networks*), también llamadas simplemente redes neuronales y referidas como RNA, son un subconjunto de modelos del aprendizaje automático y están en el núcleo de los algoritmos de aprendizaje profundo.

Neurona artificial

Una neurona biológica se compone principalmente de cinco partes, incluidas las dendritas, el soma, el núcleo, el axón y los axones terminales. Las dendritas actúan como entrada de la neurona. Están conectados a una entrada sensorial (como el ojo) o a otras neuronas a través de sinapsis. El soma recoge las entradas de las dendritas. Cuando las entradas pasan un cierto umbral, dispara una serie de impulsos a través del axón. A medida que se dispara la señal, el núcleo vuelve a su estado estacionario. Cuando llega a este estado, el disparo se detiene. Las señales disparadas se transmiten a otras neuronas a través de los axones. Finalmente, las conexiones sinápticas transmiten las señales de una neurona, mediante el axón terminal, a otra neurona, mediante sus propias dendritas. Ver Figura 2.13

Según las fuerzas sinápticas y la señal en un axón terminal, cada rama de dendritas aumenta o disminuye el potencial del núcleo. Además, la dirección de la señal es siempre

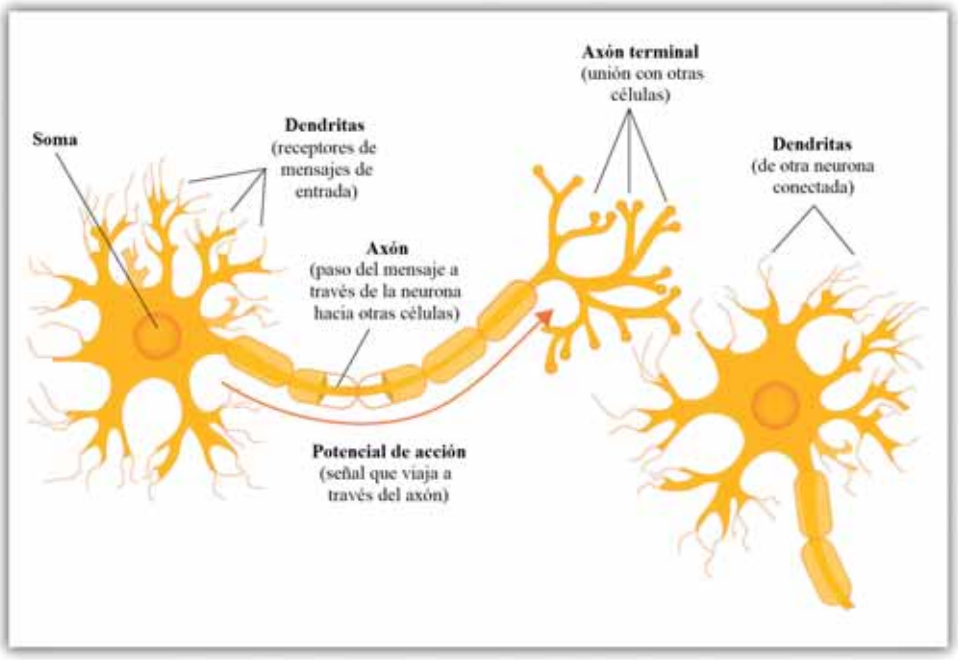


Figura 2.13: Diagrama de los componentes de una neurona biológica (34)

desde las terminales del axón hacia las dendritas. Eso significa que es imposible pasar una señal de las dendritas a las terminales de los axones. En otras palabras, el camino de una neurona a otra es siempre un camino de un solo sentido. Vale la pena mencionar que cada neurona puede estar conectada a miles de otras neuronas.

Una neurona biológica puede ser formulada matemáticamente como:

$$f(x) = g\left(\sum_{i=0}^{N}(w_i \cdot x_i) + b\right) = g\left(W \cdot X^T + B\right) \tag{2.4}$$

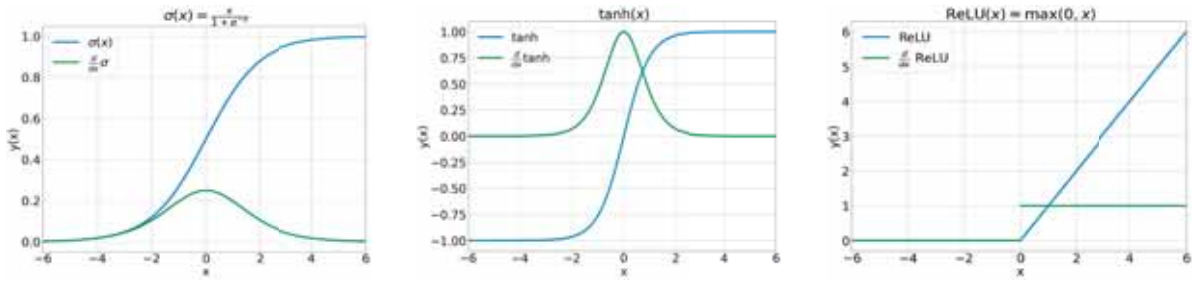
donde W es un vector de pesos sinápticos $w_1, w_2, \cdots w_i$ en la conexión neuronal, X son los datos de entrada $x_1, x_2, \ldots x_i$ a la neurona y b es el término de intersección llamado sesgo (del inglés *bias*), cuyo valor tiene el efecto de aplicar una fina transformación a la unidad de salida. Esta fórmula es referida como la definición de una neurona artificial.

Básicamente, una neurona artificial calcula la suma ponderada de las entradas. Esto imita el soma en la neurona biológica. La fuerza sináptica se modela usando los pesos w_i

y las entradas de otras neuronas o sensores se modelan usando los datos x_i. De forma vectorial, es representada como el producto punto entre W y X.

Además, g es una función no lineal que se denomina como función de activación, donde acepta como argumento un número real y devuelve otro número real después de aplicarle una transformación no lineal. La función de activación actúa como la función de umbral en la neurona biológica. Dependiendo del potencial del núcleo (es decir, $W \cdot X^T$), la función de activación devuelve un número real.

Tradicionalmente han fungido como funciones de activación las operaciones matemáticas como la logística o sigmoide, la tangente hiperbolica, la unidad lineal rectificada (o ReLU) (35) entre otros, dado que estas tiene un rango en una región limitada (Ver Figura 2.14).



(a) ***Función sigmoide*** **(b)** ***Función tangente hipérbolica*** **(c)** ***Función ReLU***

Figura 2.14: Funciones de activación usadas tradicionalmente en RNA

Estructura

Estas redes son definidas en una estructura de niveles denominados como capas de nodos, que contienen una capa de entrada, una o varias capas ocultas y una capa de salida. Cada nodo, o neurona artificial, se conecta a otro y tiene peso y un umbral asociados. Si la salida de un nodo individual está por encima del valor de umbral especificado, dicho nodo se activa y envía datos a la siguiente capa de la red. De lo contrario, no se pasan a la siguiente capa (31) (Ver Figura 2.15).

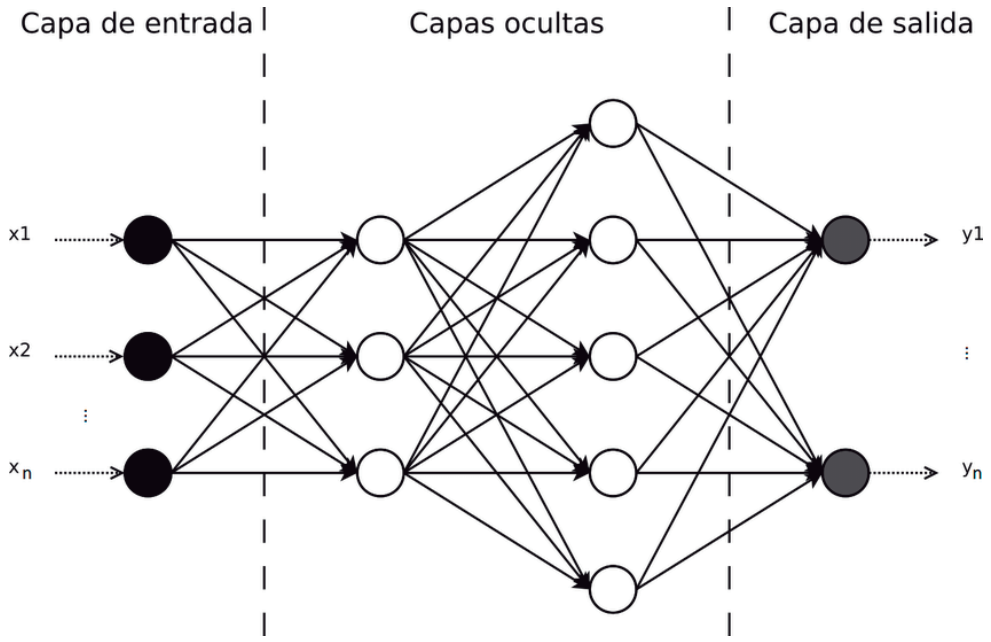


Figura 2.15: Arquitectura típica de una red neuronal formada por cuatro capas de nodos

Tipos de redes neuronales

Una RNA es clasificada por la conexión de las neuronas en sus respectivas entradas. Cada par de neuronas puede o no estar conectadas entre ellas. Dependiendo del número de conexiones que presente la red esta puede ser total o parcialmente conectada.

Otra forma de clasificación radica en la forma de conexión entre sus neuronas, pudiendo ser una red neuronal *feedforward*, traducido del inglés como red neuronal prealimentada o de avance o *FNN*, y una red neuronal recurrente (RNR). Específicamente, en una red neuronal *feedforward* las conexiones entre neuronas no forman un ciclo. Por el contrario, en las redes neuronales recurrentes, las conexiones entre neuronas son arbitrarias permitiendo formar un ciclo dirigido.

2.3.3. Redes neuronales convolucionales

Las redes neuronales convolucionales (del inglés *convolutional neural networks*), también conocidas como redes convolucionales y referidas como RNC, son un tipo especializado de red neuronal para procesar datos que tienen una topología similar a una malla o matriz. Los ejemplos incluyen datos de series temporales, que se pueden considerar como una cuadrícula vectorial que toma muestras a intervalos de tiempo regulares, y datos de imágenes, que se pueden considerar como una cuadrícula o arreglo bidimensional de píxeles.

Entre las aplicaciones que han logrado tener éxito las RNC se encuentran: clasificación de dígitos, detección y clasificación de objetos, es decir, además de clasificar al elemento estimar un cuadro delimitador que encierre al objeto; etiquetado de píxeles de una imagen, el cual consiste en etiquetar cada píxel de la imagen con una categoría del objeto al que pertenece; estimación de pose humana, que se enfoca en localizar las articulaciones humanas; detección de rostros, entre otras tareas (36).

Concepto

El término «red neuronal convolucional» indica que la red emplea la operación matemática de la convolución de los métodos de filtrado en imágenes. Es decir, las RNC son simplemente RNA que utilizan la convolución en lugar de la multiplicación general de matrices en al menos una de sus capas.

La idea propuesta de estas redes es basada en las ideas de David H. Hubel y Torsten Weisel presentadas en su artículo (37) de 1968 que les valió el premio Nobel de Fisiología y Medicina de 1981. Exploraron la corteza visual animal y encontraron conexiones entre actividades en un área pequeña pero bien definida del cerebro y actividades en pequeñas regiones del campo visual.

En algunos casos, incluso fue posible identificar neuronas exactas que estaban a cargo de una parte del campo visual. Esto los llevó al descubrimiento de los denominados campos receptivos (del inglés *receptive fields*), descrito como el vínculo entre partes de los campos visuales y las neuronas individuales que procesan la información, que contienen la presencia de estímulos que alteran las respuestas entre neuronas.

Dentro de las RNC, los *kernels* de filtrado también son referidos como *campos receptivos*, porque pueden ser vistos como identificadores de características, donde estos extraen características de la información contenida en todas las imágenes, como pudieran

ser curvas, bordes, entre otros atributos descriptivos, que puedan alterar la activación de neuronas. Por lo que, al ser una RNA, los *kernels* fungen como los pesos sinápticos de una neurona.

De manera que, el concepto de una neurona de una RNC puede ser escrito matemáticamente como sigue en la Ecuación 2.5.

$$S = \text{Activ}\left(\text{Conv}(X, W) + B\right) \tag{2.5}$$

donde:

- X son los datos de entrada, ya sea una imagen o un conjunto de mapas de características
- W es un *kernel* tridimensional con la misma profundidad de los datos de entrada X
- B son los sesgos de una neurona
- A es un mapa de activación, resultado de la convolución entre X y W
- $\text{Activ}(x)$ es una función de activación
- S es el resultado comnocido como mapa de características (del inglés, *features map*)

Estructura

De manera detallada, una RNC está compuesta generalmente de dos secciones, la primera denominada capas convolucionales cuyo objetivo es la extracción de características de la imagen de entrada; y la segunda sección contiene capas neuronales *feedforward* completamente conectadas con el propósito de generar predicciones acerca de la descripción de la imagen o el contenido de la misma. Una representación gráfica puede verse en la Figura 2.16.

Dentro de las capas convolucionales, los datos de entrada X se convolucionan con un conjunto de K filtros $\mathcal{W} = W_1, W_2, \ldots, W_K$ que se suman por sesgos $\mathcal{B} = B_1, B_2, \ldots, B_K$. La cantidad de kernels K en una etapa de convolución siempre es recomendado que sea un valor potencia de dos para aligerar los tiempos de cómputo.

Despues de las convoluciones, cada mapa de activación A_K es apilado tras otro, a lo largo de la dimensión de la profundidad para formar el volumen de salida completo para ser procesada con la función de activación.

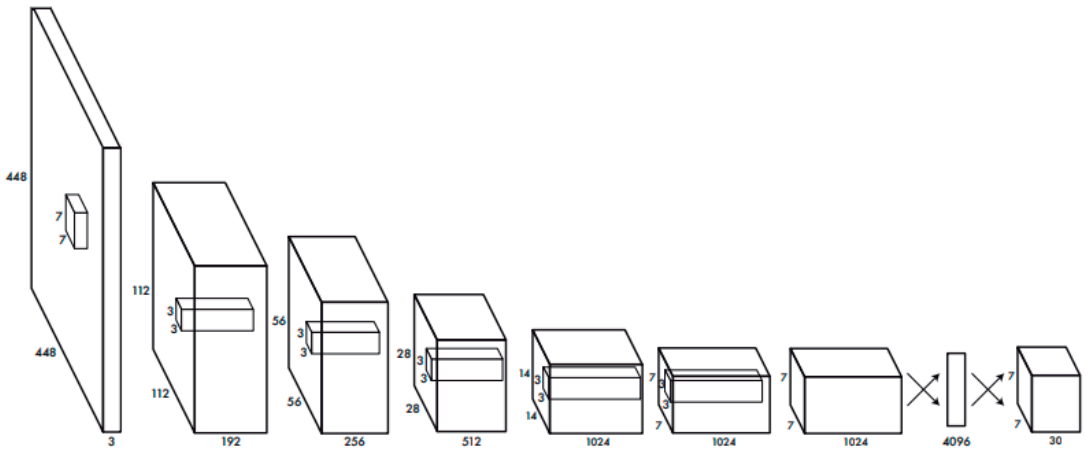


Figura 2.16: Diagrama de la estructutra de la red *YOLO* (38)

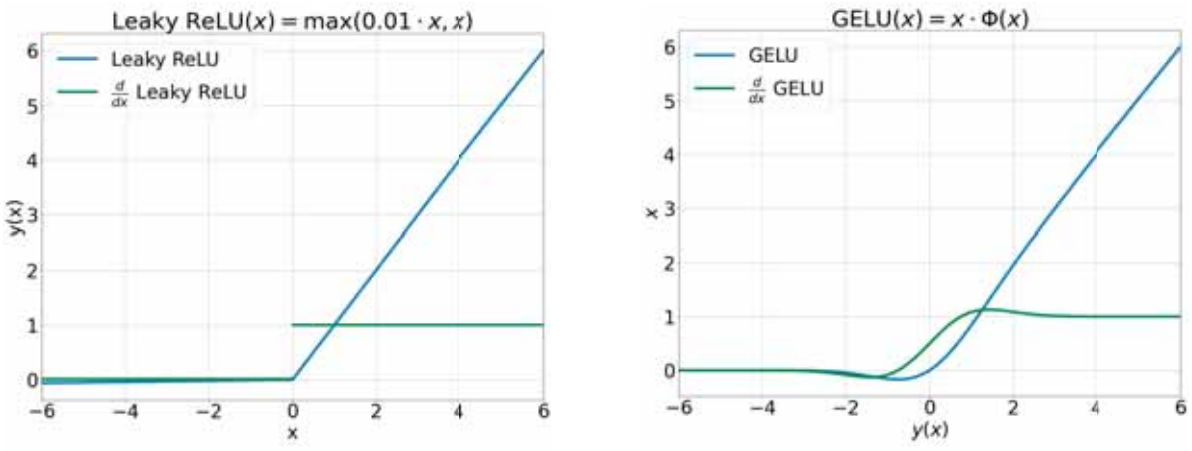
Cada capa en la red convolucional los *kernels* tiene una estructura de arreglo tridimensional, que tiene una altura, un ancho y una profundidad. La profundidad de una capa en una red neuronal convolucional no debe confundirse con la profundidad de la propia red. La palabra «profundidad» (cuando se usa en el contexto de una sola capa) se refiere a la cantidad de canales en cada capa, como la cantidad de canales de colores primarios (como el rojo, verde y azul en el espacio *RGB*) en la imagen de entrada.

La operación de convolución puede ser diferente entre cada capa convolucional con la modificación del tamaño del filtro, el paso y el rellenado de bordes.

Luego, los mapas de activación son procesados por una función de activación no lineal. Esta operación es usada para incrementar la *no linealidad* de la red sin afectar los campos receptivos de las capas convolucionales (5). Para cada función de activación, el dato de entrada x es un tensor que contiene cada valor de cada mapa de activación.

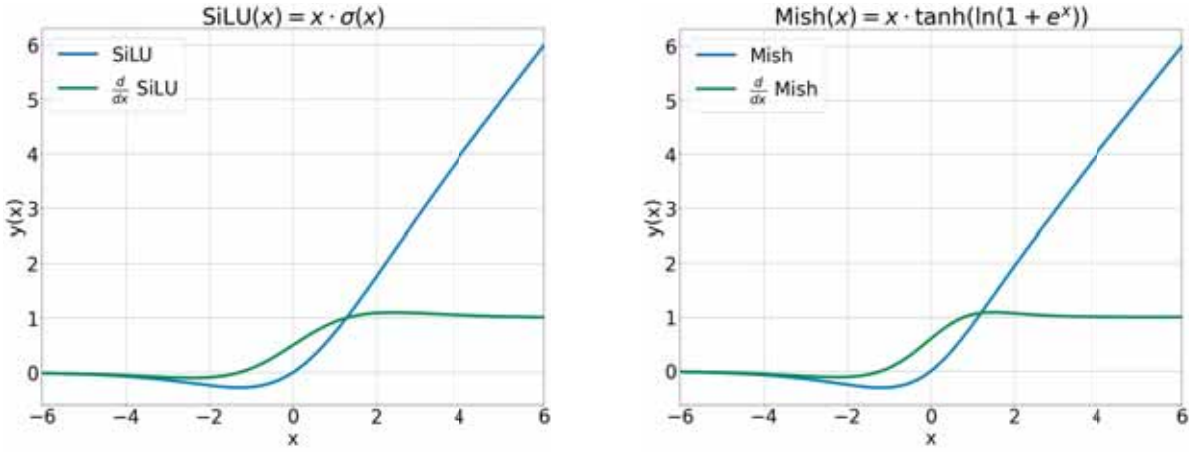
La función *ReLU*, la *LeakyReLU* (39) y la *GELU* (40) son consideradas como las tradicionales dentro de las RNC, siendo las más referenciadas dentro del estado de la técnica (41)(ver Figura 2.17).

También existen funciones que recientemente se han implementado dentro del estado de la técnica de RNC como propuestas de mejorar los resultados de las funciones anteriormente mencionadas pero con un mayor coste computacional, siendo las más usadas *SiLU* (42) y *Mish* (43) (ver Figura 2.18).



(a) ***Función*** **LeakyReLU** (b) ***Función*** **GELU**

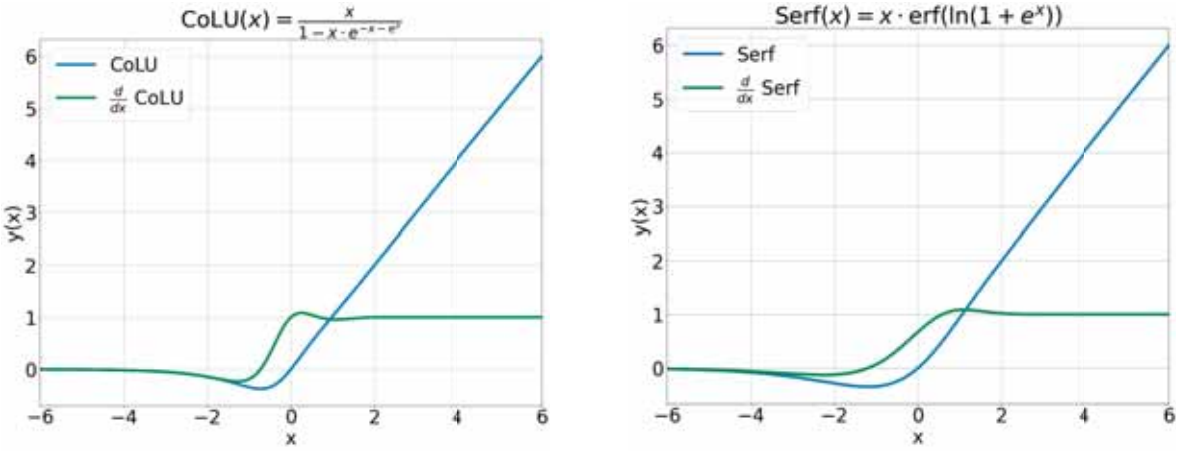
Figura 2.17: Funciones de activación usualmente referidas en RNC



(a) ***Función*** **SiLU** (b) ***Función*** **Mish**

Figura 2.18: Funciones de activación que recientemente son referidas dentro del estado de la técnica de RNC

Por último, se han desarrollado propuestas para el estado de la técnica en funciones de activación para ser implementadas en RNC, siendo ejemplos de estas las activaciones *CoLU* (44) y *Serf* (45) (ver Figura 2.19).



(a) ***Función*** **CoLU** **(b)** ***Función*** **Serf**

Figura 2.19: Funciones de activación del estado de la técnica diseñadas para RNC

Adicionalmente, dependiendo del diseño de la RNC, esta puede incorporar al término de una capa convolucional una operación de compresión (del inglés, *downsampling*) llamada *pooling*, traducido del inglés como método de reducción o agrupamiento; donde generalmente es tratada como una capa aparte de las capas convolucionales.

La capa de *pooling* tiene como propósito retener o comprimir la información importante de los datos de entrada, esto es reemplazando cada mapa de activación por otro que contiene elementos que se consideren que representan características relevantes del mapa, según el método empleado.

Los métodos de *max pooling* (traducido del inglés como reducción por máximos) y de *avg* o *average pooling* (traducido del inglés como reducción por promedio) se encuentran como los más utilizados en RNC, en donde ambos se genera una ventana de pixeles en cada mapa de características entrantes (generalmente de tamaño 2 y con un salto de uno o dos elementos) entregando como salida el elemento máximo (*max*) o el promedio del filtro (*avg*).

En la Figura 2.20 se emplea una visualización del funcionamiento de cada método de reducción.

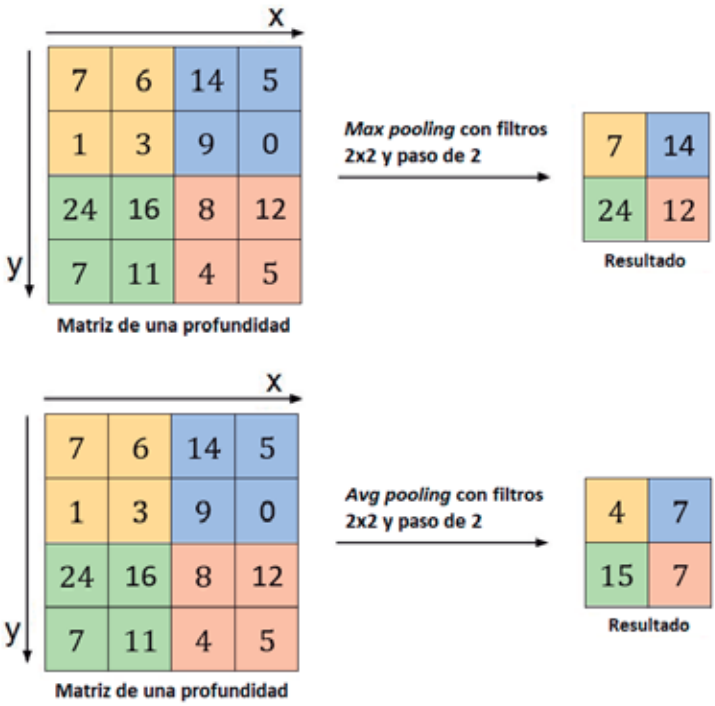


Figura 2.20: Ejemplos de las operaciones de *Max Pooling* y *Avg Pooling* en una matriz cuadrada de tamaño 4

2.4. Detección de objetos

La detección de objetos es una tecnología de la visión computacional en conjunto con el procesamiento de imágenes, que localiza e identifica objetos en una imagen. El rendimiento de la detección de objetos ha mejorado significativamente mediante la aplicación de tecnología del aprendizaje profundo.

La definición del problema de la detección de objetos es determinar la ubicación de los objetos en una imagen dada (localización de objetos) y a la categoría pertenece cada objeto (clasificación de objetos). Este problema es tratado en los modelos de detectores de objetos y se pueden dividir principalmente en tres etapas: selección de regiones informativas, extracción de características y clasificación.

2.4.1. Arquitectura

Un moderno detector de objetos generalmente se compone de un *backbone*, traducido como red troncal o columna vertebral, y un *head*, traducido como cabeza (de predicción) (46).

Un detector de objetos toma una imagen como dato de entrada y comprime las características a través de un *backbone* de red neuronal convolucional. La palabra *backbone* es referido a una red neuronal profunda compuesta principalmente por capas convolucionales, cuyo objetivo principal es extraer las características esenciales, por lo que su diseño y selección es un paso clave que mejorará el rendimiento de la detección de objetos (47).

Los detectores de objetos desarrollados en los últimos años a menudo insertan algunas capas entre el *backbone* y el *head*, y estas capas generalmente se usan para recopilar mapas de características de diferentes etapas para el dibujo de múltiples cuadros delimitadores. Es usualmente referido como el cuello (del inglés *neck*) de un detector de objetos. Por lo general, un cuello se compone de varias rutas de abajo hacia arriba y varias rutas de arriba hacia abajo (46).

La detección y predicción de clases del detector se realiza en el *head* o también referido como módulo detector. Estos módulos detectores se pueden dividir en dos clases principales según el método empleado, ya sean de los métodos de dos etapas y de los métodos de una etapa (del inglés, *two-stage and one-stage methods* respectivamente). Los métodos de dos etapas primero generan algunas propuestas de objetos candidatos y luego clasifican estas propuestas en categorías específicas. Los métodos de una etapa extraen y clasifican simultáneamente todas las propuestas de objetos. En la Tabla 2.1 se visualiza una recopilación de detectores de objetos en el estado de la técnica.

Una etapa	Dos etapas
YOLO (38)	*Faster R-CNN* (48)
Single Shot Detector (49)	*Mask R-CNN* (50)
FCOS (51)	*R-FCN* (52)
Contour Proposal Networks (53)	*RepPoints* (54)

Tabla 2.1: Recopilación de algunos detectores de objetos del estado de la técnica

Esta división del *head* se basa en el uso de redes neuronales en la detección profunda de objetos (del inglés *deep object detection*) y en términos generales, los métodos de dos etapas tienen una velocidad de detección relativamente más lenta y una precisión de detección más alta, mientras que los métodos de una etapa tienen una velocidad de detección mucho más rápida y una precisión de detección comparable (55)

En la Figura 2.21 se puede visualizar la composición en varias partes de un detector de objetos.

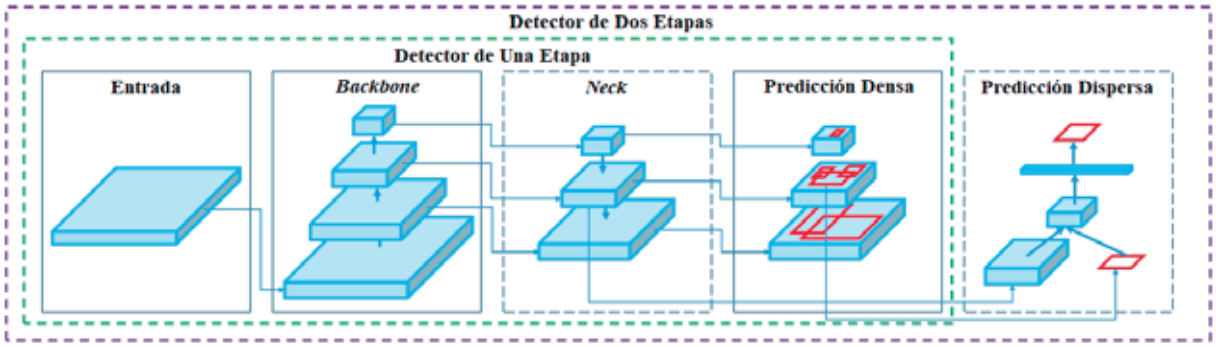


Figura 2.21: Estructura general de un detector de objetos, según lo presentado en el artículo de YOLOv4 (46)

2.4.2. Métricas de desempeño

Como todo algoritmo de aprendizaje automático, los detectores de objetos hacen uso de métricas para evaluar el desempeño en la tarea encomendada. Las métricas que se implementan en la mayoría de los detectores de objetos son las definidas por las competiciones de *PASCAL VOC* (56) y *MS COCO* (57).

Intersección sobre la Unión

La intersección sobre la unión, del inglés *intersection over unión* (IoU) y también conocida como índice Jaccard, mide la sobreposición entre los cuadros delimitadores predichos y los cuadros delimitadores *ground-truth* (traducido del inglés como verdad cimentada). Estos

último son los cuadros delimitadores etiquetados a mano en el conjunto de datos que especifican dónde está el objeto a detectar con su clase correspondiente y los cuadros delimitadores predichos provienen del modelo (ver Figura 2.22).

Figura 2.22: Un ejemplo de detección de una moneda en una imagen. El cuadro delimitador predicho se dibuja en colo rojo, mientras que el cuadro delimitador de *ground-truth* se dibuja en color verde

Matemáticamente, el IoU es el valor relativo del área de intersección de ambos cuadros delimitadores con respecto al área de unión de ambos, como se visualiza en la Figura 2.23 La puntuación de IoU está normalizada, es decir que está comprendida en el rango de 0 a 1 dado que el área de unión funge como denominador. El valor de IoU 0 representa que no hay superposición entre el cuadro delimitador predicho y el *ground-truth*, el valor 1 es el más óptimo, lo que significa que el cuadro delimitador predicho se superpone completamente al cuadro delimitador *ground-truth*; y una puntuación IoU mayor que 0.5 es considerado generalmente como una buena predicción.

El uso de esta métrica es una forma alterna al evaluar que la detección coincida «exactamente» con el *ground-truth*, dado que es un poco realista encontrar este resultado tomando

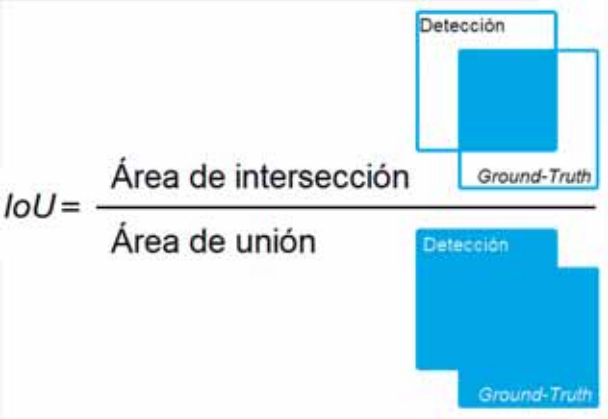


Figura 2.23: Diagrama de la operación de IoU (58)

en cuenta los múltiples parámetros del modelo de detección.

Matriz de confusión

Una matriz de confusión, o de desempeño, es un conjunto de valores relativos que ayudan a comprender como las predicciones son precisas con respecto a lo que se tiene el *ground-truth*.

En el contexto de detección de objetos, esto refleja como las detecciones predichas son tan similares con los objetos dentro de las imágenes mediante la puntuación IoU y un umbral que lo evalue (del inglés *IoU threshold*). A continuación, se definen los conceptos que conforman la matriz donde el termino de muestra positiva es la presencia del *ground-truth* correspondiente, y la muestra negativa en caso contrario.

- **Positivo Falso** (del inglés *False Positive* y con siglas FP): Predicción incorrecta en una muestra negativa. IoU igual a cero y con clase, menor que el umbral o clase incorrecta.
- **Negativo Falso** (del inglés *False Negative* y con siglas FN): Predicción errada en una muestra positiva. IoU igual a cero y sin clase.
- **Positivo Verdadero** (del inglés *True Positive* y con siglas PT): Predicción correcta en una muestra positiva. IoU igual o mayor que el umbral; y si clase es la correcta.

- **Negativo Verdadero** (del inglés *True Negative* y con siglas *TN*): Predicción inexistente en una muestra negativa. Este valor no es considerado dentro de las métricas de un detector de objetos.

Precisión y Exhaustividad

La precisión es el valor relativo de las predicciones correctas con respecto a las predicciones obtenidas. Su fórmula matemática es descrita en la Ecuación 2.6

$$P = \frac{TP}{TP + FP} \tag{2.6}$$

La exhaustividad, o del inglés *recall*, es el valor relativo de las predicciones correctas con respecto al *ground-truth* de la base de datos. Su fórmula matemática es escrita en la Ecuación 2.6.

$$R = \frac{TP}{TP + FN} \tag{2.7}$$

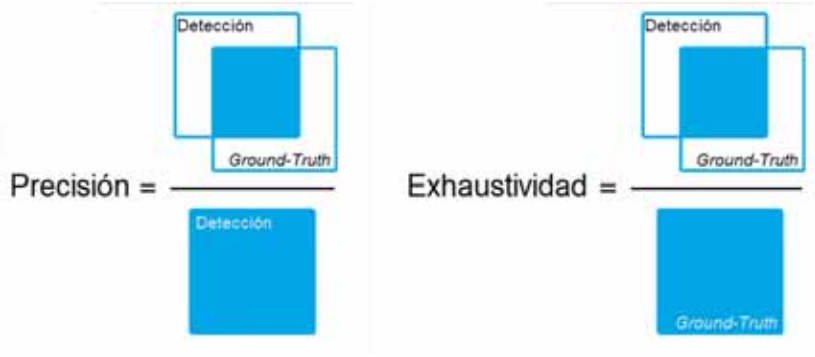


Figura 2.24: Diagrama de la métricas de precisión y exhaustividad (58)

La curva PR (del inglés *Precision-Recall curve*), es el resultado de generar la gráfica teniendo como ejes la precisión y la exhaustividad. Esta gráfica permite ver la relación en estas dos métricas de tal manera que el aumento de la exhaustividad visualiza una disminución de la precisión y viceversa. Por lo que el objetivo sería encontrar una alta precisión y una alta exhaustividad de tal manera que la curva este lo más cerca posible de la esquina superior derecha.

En la práctica, esta curva tiene una forma de *zigzag* o de picos de subida y bajadas, por lo que normalmente se procede a suavizar la curva buscando una forma más cercana a una escalera. El método más empleado es el reemplazar cada valor de precisión por el valor máximo de precisión correspondiente a un valor de exhaustividad mayor su propio valor de exhaustividad. Algunos autores refieren esta técnica como interpolación de la curva *PR* (59) (Ver figura 2.25).

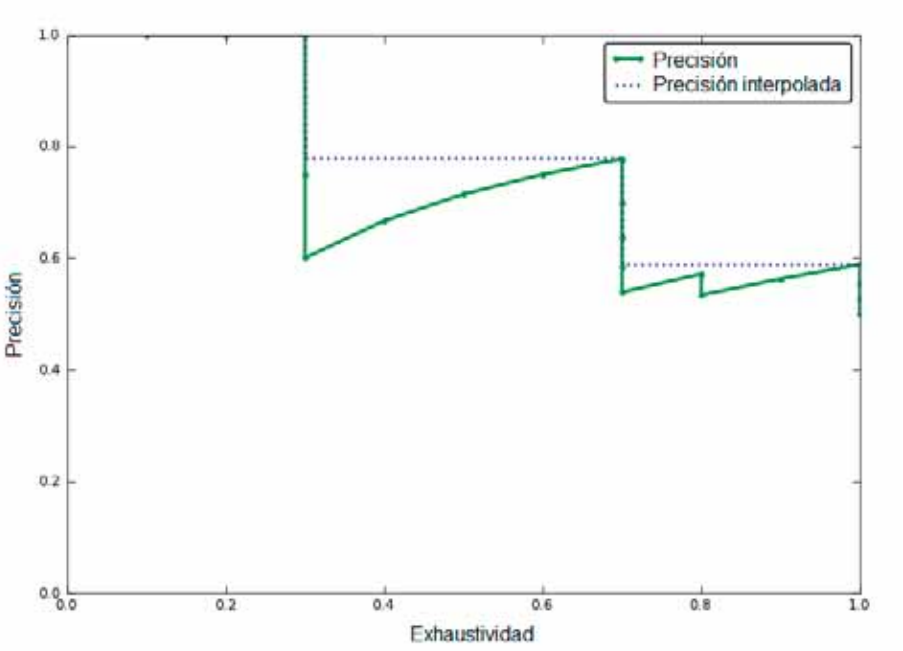


Figura 2.25: Ejemplo de curva *PR* y su interpolación (59)

Precisión Promedio

El valor de precisión promedio, del inglés *average precision* o *AP*, es el resultado de calcular el área bajo la curvar PR de una clase de objeto. El evaluador COCO, de la competencia de *MS COCO*, obtiene el valor de *AP* con el método de interpolación de 101 puntos, cuya fórmula es como sigue

$$AP = \frac{1}{101} \cdot \sum_{i=0}^{101} PR_i \tag{2.8}$$

De igual forma, *MS COCO* promedia el valor *AP* sobre un conjunto de diez umbrales de puntuación *IoU*, dentro del rango de `0.50` a `0.95` con un tamaño de paso de `0.05`; a diferencia del evaluador de *Pascal VOC* que utiliza únicamente el umbral de puntuación *IoU* `0.5`.

La métrica *mean Average Precision* (traducido del inglés como media de precisión promedio) o *mAP* es el valor relativo entra los valores AP con respecto al número total de clases a detectar. En *MS COCO*, utilizan la métrica `AP` con el mismo significado que `mAP`, con el promedio sobre todas las clases; por lo que no hace distinción entre ambos significados y se asume que la diferencia es clara por el contexto (60).

Se puede asumir la diferencia entre *AP* y *mAP*, donde se pudieran referenciar respectivamente para el único umbral de puntuación *IoU* `0.5` como `AP@.50` y `mAP@.50`; y respectivamente para los 10 umbrales de puntuación *IoU* mencionados ser referidos como `AP` y `mAP`. Adicionalmente, para enriquecer los resultados se realiza el cálculo sobre el umbral de puntuación *IoU* `0.75`, siendo referido como `AP@.75` Y `mAP@.75`.

2.5. Modelos *YOLO*

El término *YOLO* son siglas en inglés de *You Only Look One* (traducido como «*Solo miras una vez*»), y hace referencia a una familia de modelos de detección de objetos de una sola etapa.

2.5.1. Historia y desarrollo

En mayo de 2016, Joseph Redmon, junto con sus colaboradores, publicaron la primera versión de YOLO como el primer detector de objetos de una sola etapa en su artículo (38), que es capaz de ejecutarse en tiempo real con un desempeño comparable a métodos líderes y que trata la detección como un problema de regresión de cuadros delimitadores y probabilidades de clases asociadas. También propone una versión más pequeña llamada como *Fast YOLO* (traducido como YOLO rápido).

Posteriormente, Redmon y Farhadi presentaron en un artículo de diciembre de 2017 (61) dos nuevas variantes de YOLO, siendo el YOLO900 y el YOLOv2. Estos modelos eran idénticos, pero diferían en la estrategia de entrenamiento.

YOLOv2 fue entrenado en conjuntos de datos de detección como *Pascal VOC* y *MS COCO*. Al mismo tiempo, el YOLO9000 fue diseñado para predecir más de 9000 categorías de objetos diferentes entrenándolo conjuntamente en los conjuntos de datos *MS COCO* e *ImageNet*.

El modelo YOLOv2 mejorado utilizó varias técnicas novedosas para superar a los métodos de su tiempo como Faster R-CNN y SSD tanto en velocidad como en precisión. Una de esas técnicas fue el entrenamiento multiescala que permitió al modelo predecir en diferentes tamaños de entrada, lo que permitió una compensación entre velocidad y precisión.

De nueva cuenta, Redmon y Farhadi publicaron YOLOv3, en un nuevo artículo en abril de 2018 (62), donde hicieron muchos cambios de diseño con respecto a la arquitectura de red y adaptaron la mayoría de las otras técnicas del primer YOLO y especialmente de YOLOv2.

YOLOv3 introduce un nuevo *backbone* llamado `Darknet53`, que toma como base las ideas de la red *ResNet* (19) obteniendo una red convolucional mucho más grande, más precisa y más rápida; en comparación del `Darknet19` incluido en YOLOv2 (Ver Figura 2.26).

Estas tres primeras versiones desarrolladas están soportadas dentro un marco de trabajo (del inglés *framework*) de redes profundas llamado *Darknet* (63), también desarrollado por Redmon, de código abierto y escrito en lenguaje C y CUDA para el cómputo en CPUs y GPUs. *Darknet* ofrece programas para entrenar y ejecutar redes convolucionales profundas, inicialmente para acelerar la arquitectura de YOLOv2, pero posteriormente incluyó soporte para múltiples extractores de características y enfoques para la detección de objetos (64).

En febrero de 2020, Redmon anunció que se alejaba de la investigación de visión por computadora por razones de éticas. A partir de ese momento no se dejo claro si alguien podría o debería continuar usando el nombre *YOLO* para referirse a arquitecturas de nuevos modelos. Algunos han considerado a YOLOv3 como «el último» YOLO original (65).

De forma paralela, Glenn Jocher, fundador y CEO de Ultralytics (empresa enfocada en IA) y sin relación alguna con Redmon, desarrollo una popular implementación de YOLOv3 en PyTorch, biblioteca de visión computacional y procesamiento de lenguaje natural desarrollado por el Laboratorio de Investigación de Inteligencia Artificial de Facebook (o FAIR de su sigla en inglés) (66).

Capa	Filtro		Tamaño de Paso	Salida
	Cantidad	Tamaño		
Convolucional	32	3 × 3	1	224×224
Max Pool		2 × 2	2	112 × 112
Convolucional	64	3 × 3	1	112 × 112
Max Pool		2 × 2	2	56 × 56
Convolucional	128	3 × 3	1	56 × 56
Convolucional	64	1 × 1	1	56 × 56
Convolucional	128	3 × 3	1	56 × 56
Max Pool		2 × 2	2	28 × 28
Convolucional	256	3 × 3	1	28 × 28
Convolucional	128	1 × 1	1	28 × 28
Convolucional	256	3 × 3	1	28 × 28
Max Pool		2 × 2	2	14 × 14
Convolucional	512	3 × 3	1	14 × 14
Convolucional	256	1 × 1	1	14 × 14
Convolucional	512	3 × 3	1	14 × 14
Convolucional	256	1 × 1	1	14 × 14
Convolucional	512	3 × 3	1	14 × 14
Max Pool		2 × 2	2	7 × 7
Convolucional	1024	3 × 3	1	7 × 7
Convolucional	512	1 × 1	1	7 × 7
Convolucional	1024	3 × 3	1	7 × 7
Convolucional	512	1 × 1	1	7 × 7
Convolucional	1024	3 × 3	1	7 × 7
Convolucional	1000	1 × 1	1	7 × 7
Avg Pooling				1000
Softmax				

(a) ***Backbone Darknet19 (61)***

	Capa	Filtro		Tamaño de Paso	Salida
		Cantidad	Tamaño		
	Convolucional	32	3 × 3	1	256 × 256
	Convolucional	64	3 × 3	2	128 × 128
1 x	Convolucional	32	1 × 1	1	
	Convolucional	64	3 × 3	1	
	Residual				128 × 128
	Convolucional	128	3 × 3	2	64 × 64
2 x	Convolucional	64	1 × 1	1	
	Convolucional	128	3 × 3	1	
	Residual				64 × 64
	Convolucional	256	3 × 3	2	32 × 32
8 x	Convolucional	128	1 × 1	1	
	Convolucional	256	3 × 3	1	
	Residual				32 × 32
	Convolucional	512	3 × 3	2	16 × 16
8 x	Convolucional	256	1 × 1	1	
	Convolucional	512	3 × 3	1	
	Residual				16 × 16
	Convolucional	1024	3 × 3	2	8 × 8
4 x	Convolucional	512	1 × 1	1	
	Convolucional	1024	3 × 3	1	
	Residual				8 × 8
	Avg Pool		Global		
	Red conectada		1000		
	Softmax				

(b) ***Backbone Darknet53 (62)***

Figura 2.26: *Backbones* de YOLOv2 y YOLOv3

Luego, el 23 de abril de 2020, Bochkovskiy, Chien-Yao y Hong-Yuan publicaron YOLOv4 (46). En particular, Joseph Redmon se comprometió en Twitter para elogiar la cantidad de trabajo que Bochkovskiy ha realizado en una versión «canónica» del Darknet original (58) y dijo que «no importa lo que piense» sobre la implementación y el uso del término.

El 29 de mayo de 2020, Glenn Jocher creó un repositorio llamado YOLOv5 (67) que no contenía ningún código de modelo alguno, y el 9 de junio de 2020 agregó un mensaje de confirmación a su implementación de YOLOv3 titulado *YOLOv5 greetings*, traducido como «Saludos de YOLOv5».

La implementación de YOLOv5 de Jocher difiere de las versiones anteriores en algunas formas notables. Primero, Jocher no publicó (todavía) un artículo que respalde su versión de YOLO. En segundo lugar, Jocher implementó YOLOv5 de forma nativa en PyTorch, mientras que todos los modelos anteriores de la familia YOLO aprovechan Darknet.

En particular, a Jocher también se le atribuye la creación del aumento de datos de

mosaico implementado en su versión de YOLOv3, que es uno de los muchos aumentos de datos novedosos aprovechados en YOLOv4. Recibe un reconocimiento en el artículo de YOLOv4 (46).

Además de contar con una línea principal de desarrollo de YOLO (versiones numeradas), existen otros modelos de detección que toman como base alguna versión de YOLO. Algunos de esos trabajos, se enlistan en la Tabla 2.2.

Proyecto	Fecha de publicación	Autores	Marco de trabajo
PP-YOLO (68),	3 de agosto del 2020	Investigadores de Baidu Inc.	PaddlePaddle
Scaled-YOLOv4 (69)	15 de noviembre de 2020	Desarrolladores de YOLOv4	Pytorch y Darknet
PP-YOLOv2 (70)	21 de Abril del 2021	Investigadores de Baidu Inc. y de la Universidad Waseda	PaddlePaddle
YOLOR (71)	10 de mayo del 2021	Chien Yao, Hong Yuan y I Hau	PyTorch
YOLOX (72)	17 de julio del 2021	Zhen Ge, Songtao Liu y demás colaboradores	PyTorch y MegEngine

Tabla 2.2: Recopilación de trabajos basados en las redes *YOLO*

2.5.2. YOLOv4

El modelo YOLOv4 fue propuesto como un detector de objetos que pueda optimizar la computación paralela y mejorar la velocidad de detección de objetos (46)

Estructura

El diseño de este modelo fue construido contemplando la influencia de métodos del estado de la técnica agrupados en conjuntos denominados como *Bag of Freebies* (traducido como bolsa de regalos promocionales) o *BoF* y de *Bag of Specials* (traducido como bolsa de especiales) o *BoS*. Como resultado, la estructura de YOLOv4 (46) es descrita por sus autores como sigue:

- *backbone*: `CSPDarknet53` (73)
- *neck*: Módulo `SPP` (74) y red `PAN` (75)
- *head*: **YOLOv3** (62)

`CSPDarknet53` es el resultado de aplicar la técnica *CSP*, siglas en inglés de *Cross Stage Partial* (73), en los bloques residuales de `Darknet53`. Las redes *CSP* pueden mejorar el aprendizaje de una RNC mientras se reduce el monto de cálculo y el costo de memoria.

El módulo `SPP`, siglas en inglés de *Spatial Pyramid Pooling*, es utilizada para hacer más efectivo el campo receptivo y ayudar a separar las características contextuales, mientras que la red `PAN`, siglas en inglés de *Path Aggragation Network*, perimite acortar el camino que conecta la información de bajo nivel con la información de alto nivel (76).

El *BoF* es un conjunto de métodos que solo cambian la estrategia de entrenamiento o solo aumentan el costo de entrenamiento (nada que ver con la inferencia). Entre estos métodos se encuentran la pérdida CIoU (77), la eliminación de sensibilidad de malla (46), aumento de datos en mosaico, entre otros.

El *BoS* es un conjunto módulos de complemento y métodos de posprocesamiento que solo aumentan el costo de inferencia en una pequeña cantidad, pero pueden mejorar significativamente la precisión. YOLOv4 tienen como *BoS* función de activación Mish (43), conexiones CSP (73), DIoU-NMS (77), entre otros.

Un esquema gráfico puede ser visualizado en la Figura 2.27.

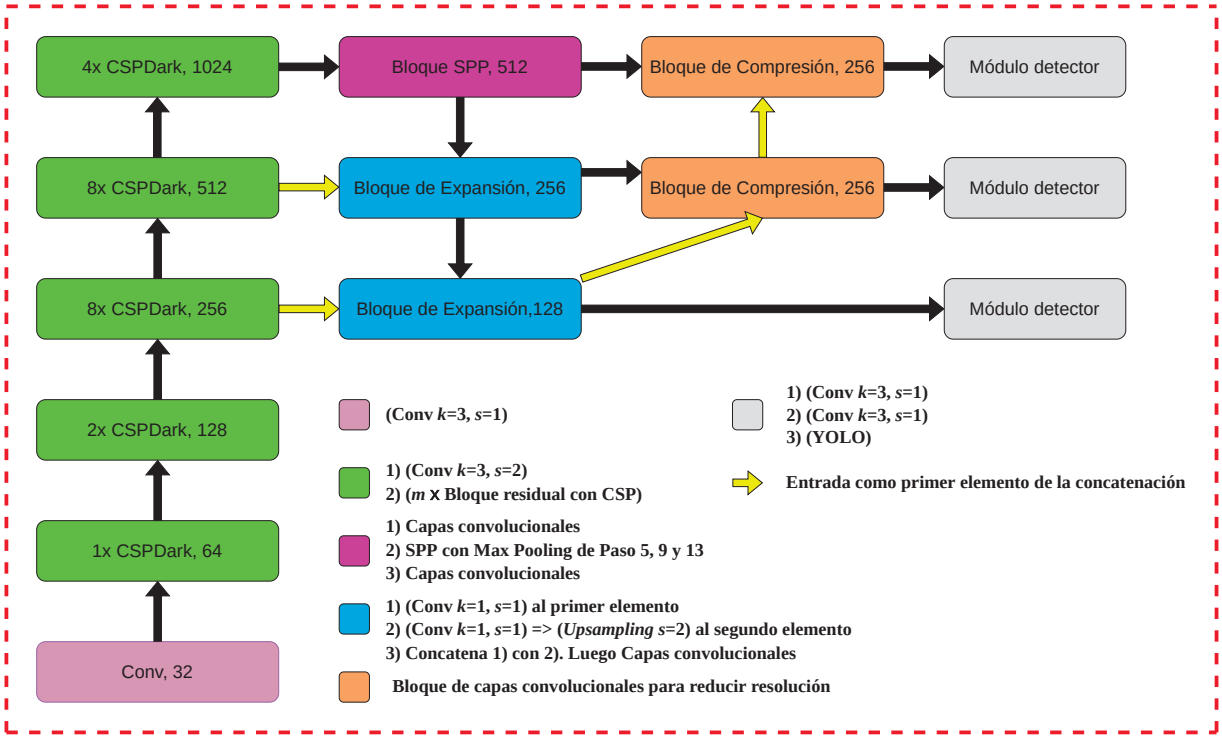


Figura 2.27: Esquema de la estructura del modelo YOLOv4

Funcionamiento

El modelo YOLO divide la imagen de entrada en una malla de SxS regiones o celdas. La altura de cada celda es igual a la altura de la imagen dividida entre S y su ancho es igual al ancho de la imagen dividida entre S, porque las dimensiones de la imagen deben ser divisibles entre S.

Si el centro de un objeto se encuentra en una de estas celdas, esa celda es etiquetada como la responsable de detectar ese objeto. Cada celda predice cuadros delimitadores y puntajes de confianza para esos cuadros. Esa confianza refleja que tan seguro está el modelo de que ese cuadro contiene un objeto y también que tan preciso se cree que el cuadro es lo que predice.

En el modelo YOLO, cada cuadro delimitador es predecido utilizando grupos o *clusters* de dimensiones como cuadros ancla o de anclaje (del inglés *anchors boxes*). El modelo

YOLOv4 hereda el *head* de YOLOv3, donde predice cuadros delimitadores en 3 escalas diferentes. El modelo extrae características de esas escalas utilizando un concepto similar a las redes piramidales de características, del inglés *feature pyramid networks* o *FPN* (78).

Para cada escala de detección, se hace uso de tres módulos detectores, en el caso de la famila YOLO es llamado módulo `yolo`, donde cada módulo realiza la detección con mapas de características en un determinado tamaño, siendo el caso en YOLOv3 y YOLOv4, detecciones de tamaño con un paso de 8, 16 y 32 respectivamente, lo que significa que la resolución de la imagen de entrada debe ser múltiplo de 32.

En cada módulo `yolo`, se predice un cuadro delimitador codificado como un tensor tridimensional, puntaje de objeto y predicción de clases. En los experimentos de YOLOv3 con *MS COCO* (57), se establecieron 3 cuadros en cada escala, por lo que el tensor tridimensional es de tamaño $N \times N \times [3 \times (4 + 1 + 80)]$ para 4 desplazamientos de cuadros delimitadores, 1 predicción de objeto y 80 predicciones de clase. (38)

El uso del algoritmo de *K-medias*, permitió estructurar cuadros delimitadores como cuadros anclas. Como en YOLOv3, YOLOv4 establece nueve cuadros anclas, divididos como grupos uniformes en tres escalas arbitrariamente escogidas, que son: $(12 \times 16), (19 \times 36), (40 \times 28), (36 \times 75), (76 \times 55), (72 \times 146), (142 \times 110), (192 \times 243), (459 \times 401)$.

El modelo predice cuatro coordenadas para cada cuadro delimitador: t_x, t_y son el centro de la predicción, y t_w, t_h las medidas de ancho y altura del cuadro. Si la celda está desplazada desde la esquina superior izquierda de la imagen por (c_x, c_y) y el cuadro delimitador *prior* tiene ancho p_w y alto p_h, entonces las predicciones corresponden a las transformaciones matemáticas, como se describen en el conjunto de Ecuaciones 2.9, así como en la Figura 2.28.

$$\begin{aligned} b_x &= \alpha_i \cdot \sigma(t_x) - \frac{\alpha_i - 1}{2} + c_x \\ b_y &= \alpha_i \cdot \sigma(t_y) - \frac{\alpha_i - 1}{2} + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \end{aligned} \tag{2.9}$$

Para el cálculo de b_x y b_y se agrega un escalador mayor y cercano a uno con el fin de eliminar la sensibilidad de la malla, en cada módulo `yolo` i. Luego de obtener las predicciones, lo siguiente es eliminar los cuadros delimitadores menos convenientes, es decir, aquellos que contengan un valor de probabilidad por debajo de un umbral fijado. A los cuadros que sobreviven a este procedimiento se les aplica una supresión no máxima

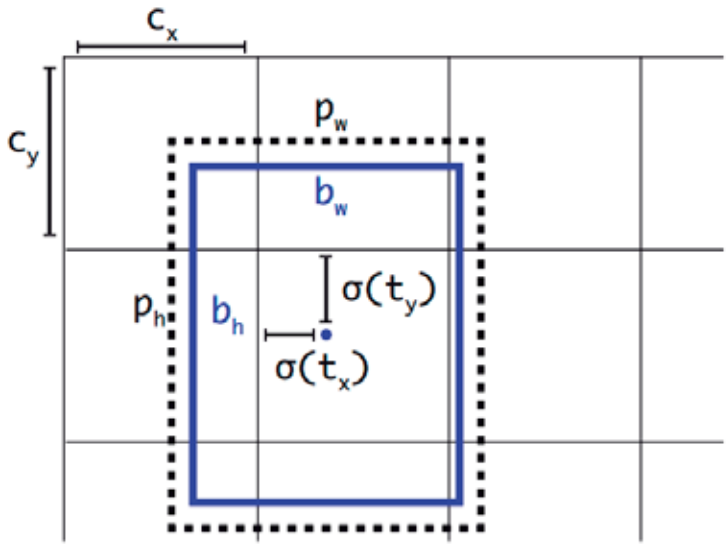


Figura 2.28: Transformación de coordenadas de los cuadros delimitadores en *YOLO*, donde el cuadro delimitador predicho azul se ajusta con el cuadro ancla asociado (61)

(del inglés *Non-Max Suppression*) para eliminar las posibles detecciones duplicadas de los objetos y dejar únicamente la más acertada (Ver Figura 2.29).

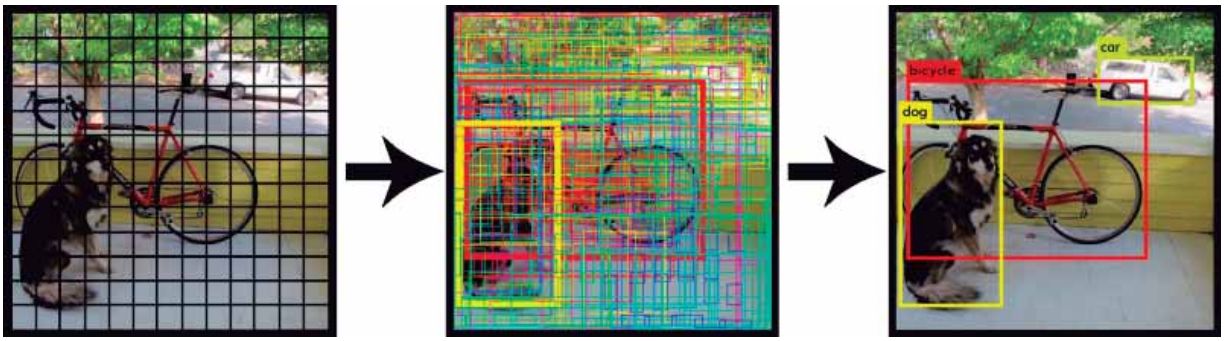


Figura 2.29: Proceso de detección en *YOLO*, eliminando cuadros delimitadores no necesarios (38)

Cada cuadro predice las clases que puede contener el cuadro delimitador mediante

la clasificación multietiqueta, usando clasificadores logísticos independientes. Durante el entrenamiento, la función de pérdida de la entropía cruzada binaria (del inglés *Binary Cross Entropy*) es usada para las predicciones de clase.

Función de pérdida

YOLOv4 usa la siguiente función de perdida

$$\mathcal{L}_Y = \mathcal{L}_{CIoU} - \sum_{i=0}^{S^2}\sum_{j=0}^{B} I_{ij}^{\text{obj}} \cdot \text{BCE}\left(C_i, \hat{C}_i\right) - \lambda_{\text{noobj}} \cdot \sum_{i=0}^{S^2}\sum_{j=0}^{B} I_{ij}^{\text{noobj}} \cdot \text{BCE}\left(C_i, \hat{C}_i\right) - \sum_{i=0}^{S^2} I_{ij} \sum_{c \in clases} \left[\text{BCE}\left(p_i(c), \hat{p}_i\right)\right] \quad (2.10)$$

donde

- I_{ij}^{obj} es un penalizador que determinar que si en la celda s contiene un cuadro delimitador responsable j de un objeto y encuentra su adecuado con respecto al *ground-truth*. Por el contrario, I_{ij}^{noobj} penaliza para las celdas que no contienen objetos.
- λ_{noobj} es un ponderador que se asigna a celdas que no contienen objetos a encontrar.
- C_i son las medidas de cuadro delimitador predicho y $\hat{C}_i$ las medidas del cuadro delimitador del *ground-truth*
- $p_i(c)$ es la predicción de la clase asigna al objeto encontrado; y $\hat{p}_i$ es la etiqueta *ground-truth* de la clase del objeto encontrado.
- $\mathcal{L}_{CIoU}$ es la perdida de *IoU* completa (del inglés *Complete IoU Loss*) (77)
- BCE es la función de perdida de la entropía cruzada binaria.

2.6. Aprendizaje transferido

El aprendizaje transferido (del inglés *Transfer Learning*), es conocido como la mejora del aprendizaje en una tarea nueva, a través de la transferencia del conocimiento adquirido de una tarea similar que ya ha sido aprendida. En otras palabras, es la aplicación que se obtiene de un contexto a otro contexto.

En las redes neuronales, el uso del aprendizaje transferido es visto en la forma de Modelos pre entrenados, generados de un entrenamiento previo de una red neuronal sobre un conjunto de datos bastante grande, como Imagenet, usando sus parámetros para una nueva tarea de interés de la red. Aplicar el conocimiento de un modelo podría ayudar a reducir el tiempo de capacitación y los problemas de aprendizaje profundo al tomar los parámetros existentes para resolver problemas de datos "pequeños".

Capítulo 3

Metodología

El proceso de desarrollo de este trabajo es descrito a continuación, donde es presentando dentro de subsecciones para una mejor estructuración de lectura.

3.1. Modelo propuesto

En este trabajo se propone una versión modificada del modelo de detección YOLOv4, siendo referido como modelo `uY4TC`, que implementa algunos métodos del estado de la técnica, con el fin de alcanzar un mejor resultado de detección. Un esquema de funcionamiento del modelo propuesto es visualizado en la Figura 3.1

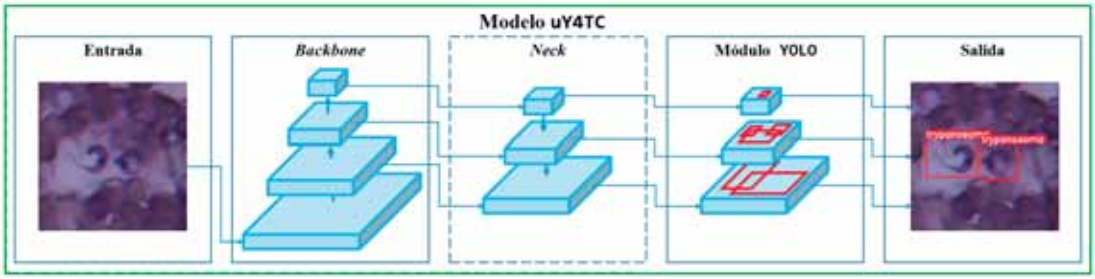


Figura 3.1: Esquema de funcionamiento del modelo propuesto `uY4TC`

Por lo que, para poder lograr los resultados esperados, es requerido contar con una

base de datos etiquetada de la clase objetivo, en este caso el parásito *T. cruzi*, desarrollar e implementar el modelo propuesto, contrastando sus resultados con los de otros modelos del estado de la técnica.

3.2. Base de datos

El conjunto de datos que se usará en este trabajo como muestras consiste de 772 imágenes en formato RGB de tamaño 2560 × 1920 píxeles. Las imágenes fueron tomadas de muestras de sangre de ratones infectados con el parásito *T. cruzi* en el Centro de Investigaciones Regionales "Dr. Hideyo Noguchi"de la Universidad Autónoma de Yucatán (Ver Figura 3.2).

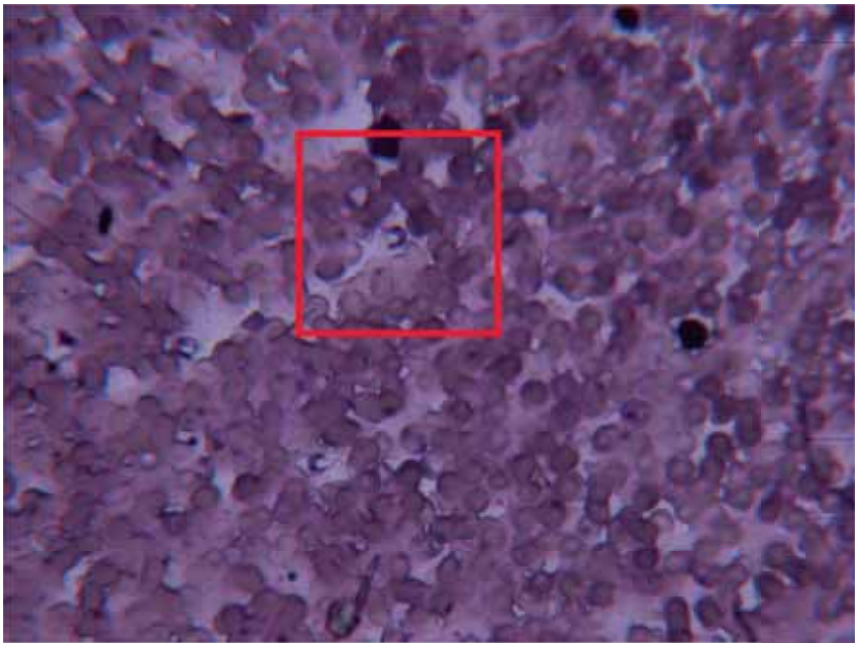


Figura 3.2: Imagen de muestra sanguínea con presencia del parásito *Trypanosoma cruzi* remarcado con un cuadro rojo

3.3. Etiquetado de la base de datos

Para que sea posible la detección del parásito es necesario etiquetar el objeto en las tomas sanguíneas, es decir, indicar en que posición de la imagen se encuentra, ya sea uno o varios parásitos *T. cruzi*. Este proceso de etiquteado se realiza con el programa *LabelImg*.

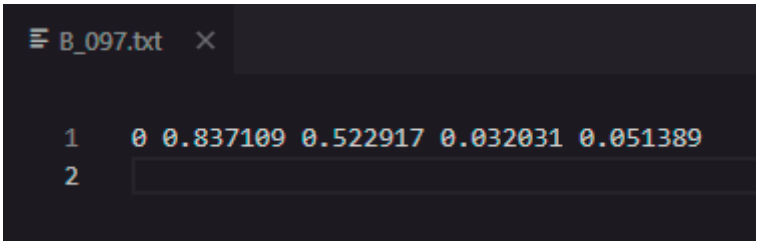
Este programa genera un archivo `.txt` por cada archivo de imagen en el mismo directorio y mismo nombre de la imagen; que contiene la información de cada parásito por cada línea, estructurada como sigue:

```
<clase del objeto> <x_centro> <y_centro> <ancho> <altura>
```

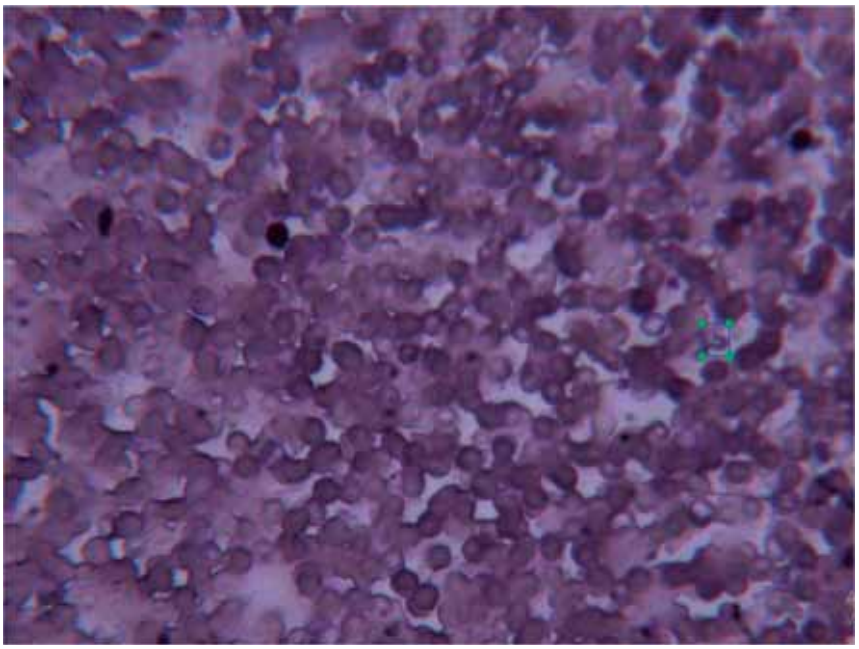
donde:

- `<clase del objeto>` es un identificador entero que va desde `0` hasta `total de clases de objetos - 1`
- `<x_centro> <y_centro> <ancho> <altura>` son valores flotantes relativos al ancho y alto de la imagen en el intervalo $(0.0, 1.0]$.

Por ejemplo, para la imagen `B_097.jpg` se crea el archivo `B_097.txt` que contendrá la información de cada parásito que contiene (Ver Figura 3.3).



(a) ***Información de parásito presente en la imagen***



(b) ***Imagen etiquetada***

Figura 3.3: Etiquetado de la imagen `B_097.jpg` que contiene la ubicación de un parásito con el programa *LabelImg*

3.4. División y aumentado de datos

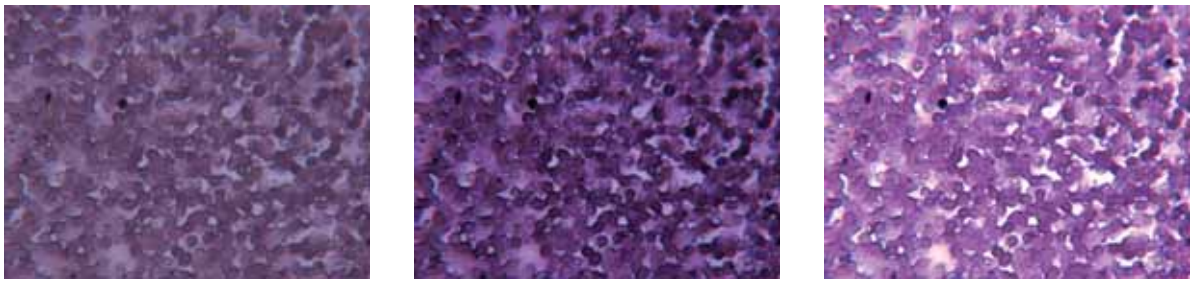
En la Tabla 3.1 se presenta las estadísticas de la base de datos, donde se presenta una división de la misma en tres conjuntos, el de entrenamiento, el de validación y el de prueba, con sus respectivas cantidades de imágenes e instancias de objeto, para este trabajo serían la cantidad de parásitos presentes en todas las imágenes de la base de datos etiquetada.

Conjunto	Imágenes	Instancias
Entrenamiento	512	838
Validación	130	219
Prueba	130	210
Total	772	1261

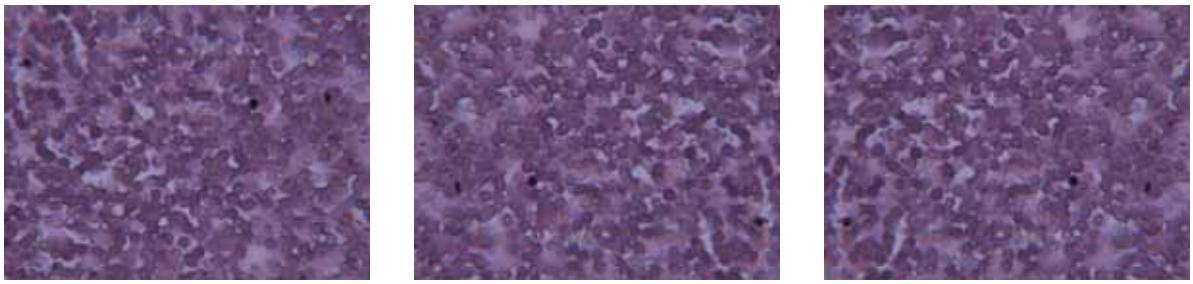
Tabla 3.1: Estadísticas de la base de datos construida

Adicionalmente, se aplicó un aumentado de datos al conjunto de entrenamiento, donde por cada imagen, se generaron cinco copias adicionales presentando una manipulación de los datos de su original, simulando posibles casos alternativos en el mundo real como se describen a continuación (ver Figura 3.4):

- Bajo contraste y de brillo, simulando un pequeño exceso en el colorante en la tinción de la muestra sanguínea.
- Alto contraste y de brillo, simulando un exceso de iluminación durante una observación microscópica.
- Volteo horizontal, vertical y de ambos, simulando una posición alternativa del cubreobjetos.



(a) ***Imagen original*** (b) ***Imagen de bajo contraste y brillo*** (c) ***Imagen de alto contraste y brillo***



(d) ***Imagen volteada horizontalmente*** (e) ***Imagen volteada verticalmente*** (f) ***Imagen volteada en ambos sentido***

Figura 3.4: Aumentado de datos sobre la imagen de entrenamiento `B_097.jpg`

El resultado de haber aplicado un aumentado de datos fue por cada imagen del conjunto de entrenamiento, esta tendría 5 imágenes hermanas; dando en total al conjunto de entrenamiento fueron de 3072 imágenes y de 5028 instancias de objeto. Para los conjuntos de validación y prueba no se aplicó aumentación de datos.

3.5. Estructura de modelo propuesto

Para este trabajo, se presenta un modelo de detección YOLOv4 modificado para orientarlo a la detección de parásitos, en sus componentes, son nuevos cuadros anclas y modificación de la función de pérdida y de activación, y de las ecuaciones de regresión.

3.5.1. Cuadros Anclas

Los autores refieren que una buena precisión en el modelo para orientarlo en la detección de objetos específicos, es necesario redefinir los valores de ancho y altura de los cuadros anclas predefinidos en el modelo.

Para esta redefinición se capturan los valores normalizados de ancho y alto de los cuadros delimitadores, generados en el proceso de etiquetado para el conjunto *ground-truth*, y luego son procesados en un algoritmo de *K-medias* para encontrar nueve centroides dentro del conjunto de puntos definidos en el plano ancho-altura de los cuadros anclas.

Para estos centroides obtenidos, sus coordenadas son utilizadas como base para la redefinición de los cuadros anclas del modelo, donde son ordenados de orden ascendentes tomando como base el área que abarcan. En la Ver Figura 3.5 se visualizan los centroides escogidos después de algunas ejecuciones del *K-medias* y en la Tabla 3.2 se detallan de forma numérica y ordenada.

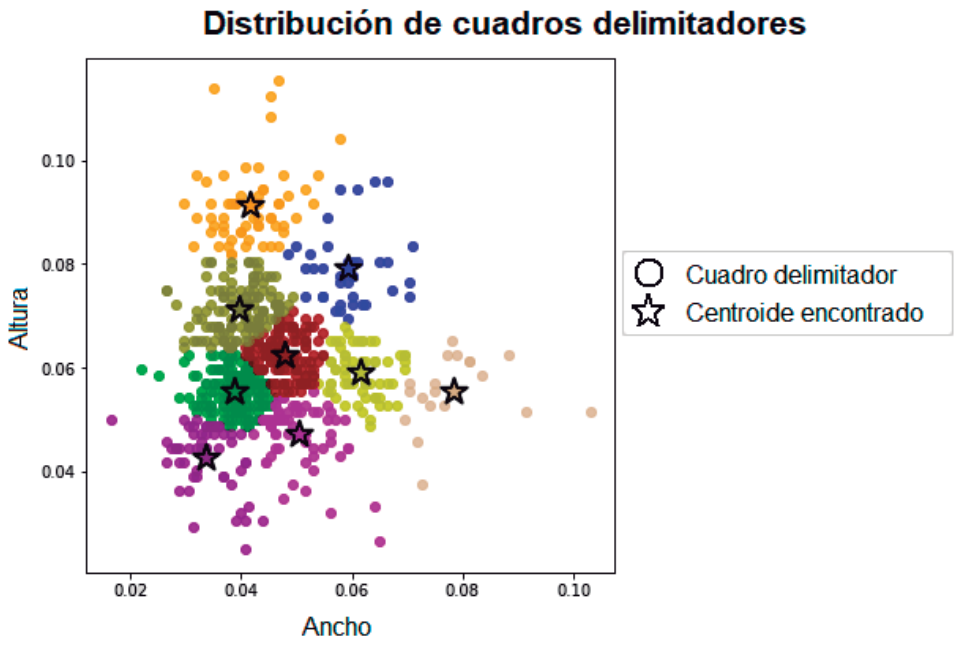


Figura 3.5: Resultado escogido de la ejecución de *K-medias* sobre las medidas relativas de los cuadros delimitadores del *ground-truth*

Orden	Ancho	Altura
0	0.03375583	0.042741
1	0.03877362	0.05542086
2	0.05032241	0.04726632
3	0.03970064	0.07116206
4	0.04774952	0.06258292
5	0.06148821	0.05899361
6	0.04158557	0.09130236
7	0.078495	0.05548247
8	0.05926949	0.07920416

Tabla 3.2: Centroides escogidos para calcular nuevos cuadros anclas

Como se requieren cuadros anclas de medida entera, los centroides obtenidos son multiplicados por una serie de cuatro números que representan resoluciones de imágenes que YOLO recomienda para su ejecución:

- 416, resolución usada por defecto en *Darknet*. Usada para los centroides `0` y `1`.
- 512, resolución adicional recomendada en *Darknet*. Usada para los centroides `2` y `3`.
- 640, resolución usada por defecto en la implementación de YOLOv3 del repositorio `ultralytics`. Usada para los centroides `4` y `5`.
- 1024, alta resolución para una mejor detección de objetos. Usada para los últimos tres centroides.

Estas números multiplicadores, permiten acortar el rango de áreas de los cuadros anclas originales del modelo $[192, 184059]$, al rango de los nuevos cuadros anclas $[238, 4860]$ descritos en la Tabla 3.3.

Orden	Ancho	Altura
0	14	17
1	16	23
2	25	24
3	20	36
4	30	40
5	39	37
6	42	93
7	80	56
8	60	81

Tabla 3.3: Cuadros anclas implementados

3.5.2. Función de pérdida *IoU*

Para la función de pérdida, se reemplaza la funcion de pérdida de IoU completa por la pérdida de IoU eficiente (79). Además, se ignora el cálculo de la pérdida de clase, dado que se cuenta con una única clase, por lo que la nueva fórmula de la función de pérdida de YOLOv4 quedaría rescrita como sigue en la Ecuación 3.1.

$$\mathcal{L}_Y = \mathcal{L}_{EIoU} - \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{obj}} \cdot \text{BCE}\left(C_i, \hat{C}_i\right) - \lambda_{\text{noobj}} \cdot \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{noobj}} \cdot \text{BCE}\left(C_i, \hat{C}_i\right) \quad (3.1)$$

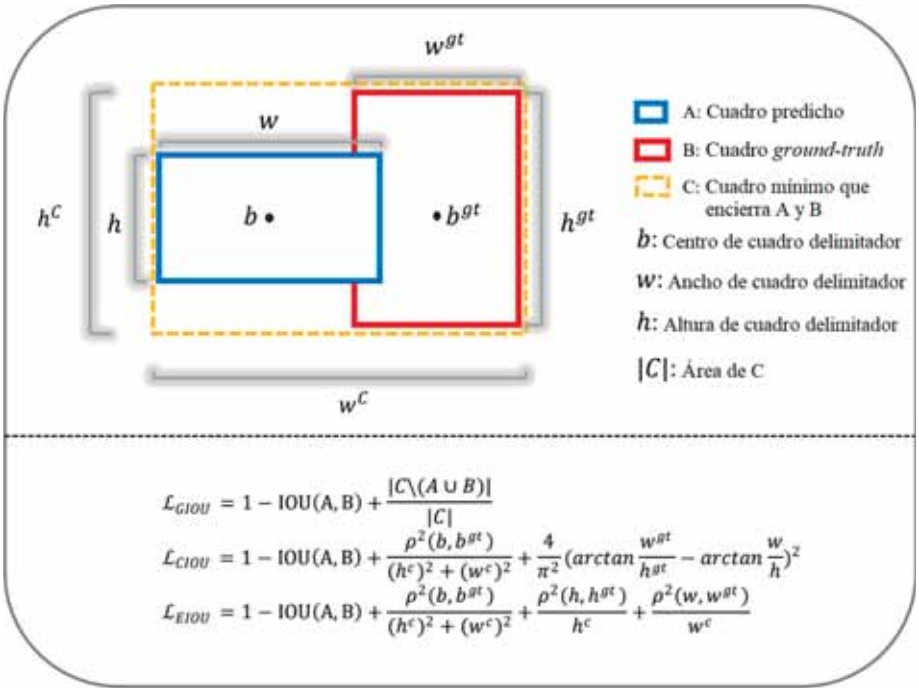


Figura 3.6: Esquema de funcionamiento de la pérdida de *IoU* eficiente (79)

3.5.3. Funciones de activación

Se hace el reemplazo de dos funciones de activación en el modelo. La función de activación *Mish* es remplazada por la función *Serf* en las capas convolucionales definidas en el *backbone*. La implementación se describe en la Ecuación 3.2

$$\text{Serf}(x) = x \cdot \text{erf}\left(\ln 1 + e^x\right) \tag{3.2}$$

donde erf() es la función de error. De igual forma, se implementa su derivada (45) como se describe en la Ecuación 3.3.

$$\text{Serf}'(x) = \frac{2}{\sqrt{\pi}} e^{-\ln^2 1+e^x} \cdot x \cdot \sigma(x) + \text{erf}\left(\ln 1 + e^x\right) \tag{3.3}$$

A su vez, para las capas convolucionales correspondientes al *head* del modelo, donde

tienen implementado la función de activación *LeakyReLU* con parámetro `0.1`, es sustituida por la función de activación *SiLU* (42)

3.5.4. Ecuaciones de regresión

Se sustituyen las ecuaciones de regresión por las definidas en YOLOv5, donde el escalador es establecido como $\alpha = 2$, quedando rescritas como siguen en el conjunto de Ecuaciones 3.4 para cada módulo `yolo`.

$$\begin{aligned} b_x &= 2 \cdot \sigma(t_x) - 0.5 + c_x \\ b_y &= 2 \cdot \sigma(t_y) - 0.5 + c_y \\ b_w &= (2 \cdot \sigma(t_w))^2 \cdot p_w \\ b_h &= (2 \cdot \sigma(t_h))^2 \cdot p_h \end{aligned} \tag{3.4}$$

3.6. Implementación

La implementación del modelo propuesto es construido en el lenguaje Python3 y el uso de la bibiloteca PyTorch, en lugar del uso del marco de trabajo *Darknet*, donde su manipulación requería de mucha mano técnica para la construcción de módulos necesarios para este trabajo.

Con respecto a la base de datos, el conjunto de entrenamiento es empleado para el entrenamiento, el conjunto de validacion es utilizada para evaluaciones de rendimiento que permiten determinar si el apendizaje es el adecuado, y el conjunto de evaluación es usado para la detección y el cálculo de métricas de desempeño como resultados finales de experimentación.

3.6.1. Código base de *YOLOv5*

El modelo propuesto tomará como código base el desarrollado para YOLOv5, en su versión de lanzamiento `6.0` (80), tomando como referencia el diseño del código de YOLOv3 con la versión del repositorio de *Ultralytics* (81), compatible con dicha versión de YOLOv5.

Los archivos de configuración para este modelo son construidos en base a los desarrollados por Chien-Yao en su versión de YOLOv4 en PyTorch (82), en la rama denominada `u5`, especificamente la configuración del modelo `Large`. La compatibilidad del modelo con el código de YOLOv5 se persigue con el objetivo principal de un mejor rendimiento de memoria en GPU, dado que la impementación de funciones de activación no nativas de PyTorch requiere del uso de más memoria, con respecto a la rama `master` de dicho repositorio.

Este proceso de compatibilidad provoca un reacomodo de módulos definidos del modelo con respecto a su definición original, como sucede con YOLOv3 de *ultralytics*, que es estructurado de manera diferente al modelo YOLOv3 original en *Darknet* (63) diseñado por Redmon en 2018.

3.6.2. Plataforma

Para el entrenamiento y posterior detección de parásitos, se utlizó la plataforma de *Google Colaboratory*, que proporciona el uso de máquinas virtuales con tarjetas gráficas en la nube para acelerar la ejecución del algoritmos de visión computacional, así como también la ejecución de cuadernos `.ipynb` (*Python Jupyter Notebook*)

La especificaciones de programación del modelo se realizó con Python3.7, PyTorch versión 1.11 y en conjunto con el paquete de de herramientas CUDA versión 1.13. Las especificaciones del equipo de cómputo proporcionado por la plataforma fue con un procesador *Intel Xeon* de 2.20GHz, espacio de disco de 78GB y tarjeta gráfica *Tesla T4* de 15GB.

3.6.3. Entrenamiento de datos aumentados

Durante el entranimiento, el modelo implementa aumentación de datos sobre el conjunto de entrenamiento en tiempo de entrenamiento. Estas operaciones son volteos, escalas, traslación y manipulación de colores sobre un mosaico construido como dato de entrenamiento. Este mosaico es generado sobre puntajes de probabilidad, donde se busca que no se generen datos repetidos durante las épocas del entrenamiento (Ver Figura 3.7).

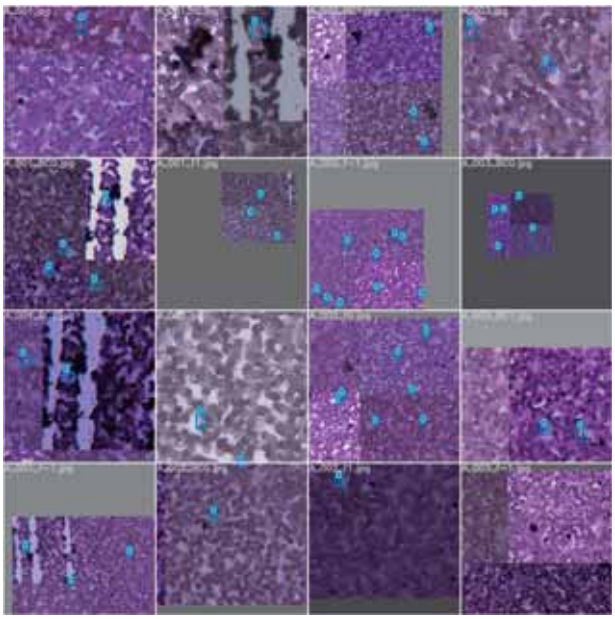


Figura 3.7: Ejemplo de aumentación de datos durante el entrenamiento

3.6.4. Validación

El modelo fue evaluado al término de cada época (una época es el momento que el modelo entrena visualizando todas las imágenes del conjunto entrenamiento) con el cálculo de las métricas *mAP* de *MS COCO* para medir su rendimiento sobre el conjunto de validación.

La evaluación determina un punto de alto de entrenamiento cuando las métricas ya no siguen aumentando su puntuación significativamente o un estancamiento de las mismas, donde dentro de este rango de este evento, se determina que los pesos de entrenamiento adecuados son aquellos con la puntuación *mAP* más alta como prioridad, aun cuando se registra las métricas `mAP@.50` y `mAP@.75`.

3.6.5. Implementación y comparación con otros modelos

Para una evaluación final de los experimentos, se realizó de la implementación de 4 modelos adicionales para una comparación de resultados. En total, este trabajo se ejecutaron 6 modelos con el uso de modelos preentrandos (aprendizaje tranferido), quedando divididos en 2 grupos, acorde a su diseño de origen.

El primer grupo continen a los modelos YOLOv3, YOLOv4, `uY4TC` y YOLOv5; siendo referidos como modelos *uYOLO*, dado a que fueron implementados con el código fuente de YOLOv5 y evitando la confusión con la implementación oficial en *Darknet*.

Cabe mencionar que, en el caso de YOLOv3 se implementa la estructura definida por Redmon y no la proporcionada en el repositorio de `ultralytics`, para el caso de YOLOv5 se usa el modelo `Large`, y los modelos YOLOv4 fue adaptada su estructura para el código (82), construyendo tanto los archivos de configuración como los pesos de modelos preentrenados que fueron adaptados de los pesos preentrenados de *Darknet*.

El segundo grupo contienen a los modelos de Faster R-CNN y Mask R-CNN, referidos como modelos *Detectron2* (83), dado que se encuentran implementados en el marco de trabajo homónimo. La base de estos métodos hacen uso de un *backbone* combinado entre *ResNet-50* (19) y *FPN* (78) preconstruido en el marco de trabajo.

Los modelos fueron implementados con una tasa de aprendizaje de 0.01, usando el optimizador del descenso de gradiente estocástico (del inglés *Stochastic Gradient Descent* o *SGD*), con la base de datos aumentada localmente y aplicando en tiempo de entrenamiento otra aumentación de datos adicional, según lo construido en los marcos de trabajo en donde se encuentran construidos los modelos: `ultralytics` o *Detectron2*.

Capítulo 4

Resultados y discusión

A continuación se presentan los resultados de la experimentación de los modelos durante la fase de entrenamiento, así como los resultados de rendimiento alcanzados después del aprendizaje.

Los modelos fueron entrenados con un tamaño de lote de 12 imágenes durante 50 épocas o 12800 iteraciones. Las estadísticas que registran la ejecución de los modelos *uYOLO* son guardados en archivos `CSV` y sus pesos en formato `.pt`; mientras que para los modelos *Detectron2* sus resultados son en formato `JSON` y pesos en formato `.pth`

Cabe aclarar, que en la presentación de las métricas de desempeño, tanto *AP* como *mAP* en este trabajo tiene el mismo significado como el mismo valor, dado que se trata de la detección de una sola clase. Sin embargo, se elige el uso del término `mAP` para enfatizar el cálculo final para su reporte. Además, estos valores `mAP` se escriben en puntos porcentuales. A su vez, los resultados de las métricas correspondientes al modelo Mask R-CNN son referentes al desempeño de predicción de los cuadros delimitadores, ignorando los cálculos de *mAP* correspondientes a la evaluación de la segmentación.

4.1. Validación de modelos

En la Tabla 4.1 se visualizan los resultados obtenidos del cálculo de las métricas `mAP` sobre el conjunto de validación de la base de datos de cada modelo al final del entrenamiento, resaltando en **negritas** los valores óptimos de cada columa, que representa cad métrica `mAP`.

Modelo	mAP	mAP@.50	mAP@.75
YOLOv3	52.262	91.458	56.556
YOLOv4	52.037	92.412	53.795
uY4TC	**53.146**	**93.297**	53.572
YOLOv5	52.098	92.167	51.345
Faster R-CNN	51.868	84.827	**59.641**
Mask R-CNN	51.74	87.193	58.509

Tabla 4.1: Relación de estadísticas obtenidas de los modelos implementados sobre el conjunto de validación

De las estadísticas presentadas en la Tabla 4.1, el modelo propuesto uY4TC toma los valores más altos, tanto para la métrica mAP, como para la métrica mAP@.50, sin embargo para la métrica mAP@.75 no consigue el mejor resultado, donde el modelo Faster R-CNN lo consigue por mucho margen.

4.2. Resultados de implementación

En esta sección se describen los resultados de desempeño de los modelos implementados métricas mAP y la visualización de cuadros delimitadores predichos, en imágenes pertenecientes al conjunto de prueba de la base de datos.

4.2.1. Métricas de rendimiento

En la Tabla 4.2 se enlistan los resultados del cálculo de las métricas *mAP* sobre el rendimiento de detección de cada modelo, redondeadas a procentajes y donde se resaltan en **negritas** el valor óptimo alcanzado en la información de cada columna.

Los resultados de la Tabla 4.2, los modelos *uYOLO*, se presentan dos resultados de rendimiento, donde cada uno representa una resolución para la imagen de entrada de los

Modelo	**Resolución**	mAP	mAP@.50	mAP@.75
YOLOv3	640	46.211	86.519	39.469
YOLOv3	960	46.764	86.171	46.445
YOLOv4	640	46.35	85.901	45.872
YOLOv4	960	48.259	87.193	50.757
uY4TC	640	**46.55**	***89.301***	38.636
uY4TC	960	***48.769***	**87.801**	45.921
YOLOv5	640	45.875	83.941	41.96
YOLOv5	960	45.234	84.079	39.908
Faster R-CNN	-	45.768	76.468	47.983
Mask R-CNN	-	46.298	79.664	**49.119**

Tabla 4.2: Relación de estadísticas obtenidas de los modelos implementados sobre el conjunto de prueba

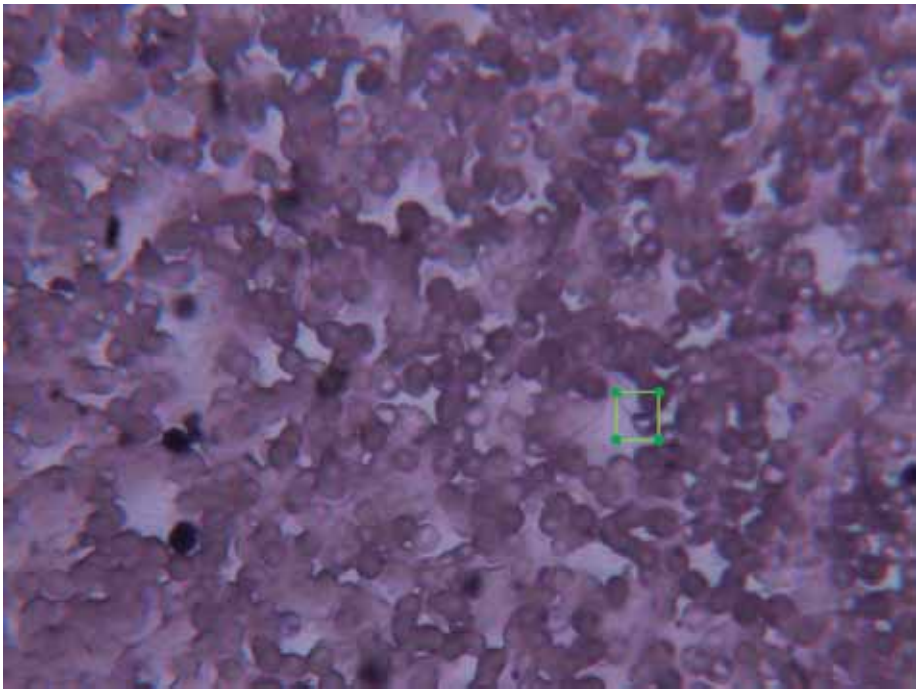
modelos, la primera es la resolución 640 de entrenamiento y validación, y la segunda una resolución con mejor definición de 960 para evaluar a los pesos obtenidos en el entrenamiento. Para los resultados de los modelos *Detectron2* se obtienen con la resolución de imagen original.

El modelo uY4TC propuesto logra los mejores resultados para mAP y mAP@.50 para las dos resoluciones presentadas. Sin embargo, para mAP@.75 en ambas resoluciones se obtienen valores muy bajos, siendo los mejores resultados con esta métrica los obtenidos con los modelos *Detectron2*. A pesar de ello, el modelo propuesto aún con la resolución de entrenamiento, logra los resultados que permiten establecer al modelo propuesto como el más competente para la detección del parásito *T. cruzi* en imágenes de tomas de sangre.

4.2.2. Detecciones

A continuación, se presentan una selección de imágenes, pertenecientes al conjunto de prueba, en las cuales se pueden visualizar las detecciones de parásitos realizadas por cada modelo.

En cada Figura, presenta una serie de imágenes, conformada por la imagen de toma de muestra de sangre remarcada con el cuadro *ground-truth* indicando la presencia del parásito, junto con 6 grupos de subimágenes que muestran la detección de los modelos YOLOv3, YOLOv4, `uY4TC`, YOLOv5, Faster R-CNN y Mask R-CNN (en ese orden) remarcado por los cuadros delimitadores generada en cada modelo, y en el caso de Mask R-CNN también se visualiza el resultado de la segmentación.



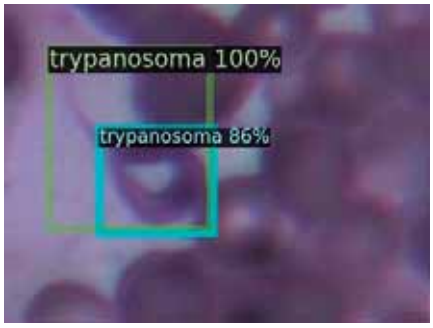
(a) Ground-truth

(b) ***YOLOv3***

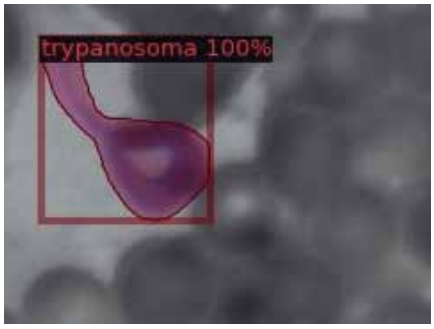
(c) ***YOLOv4***

(d) **uY4TC**

(e) ***YOLOv5***

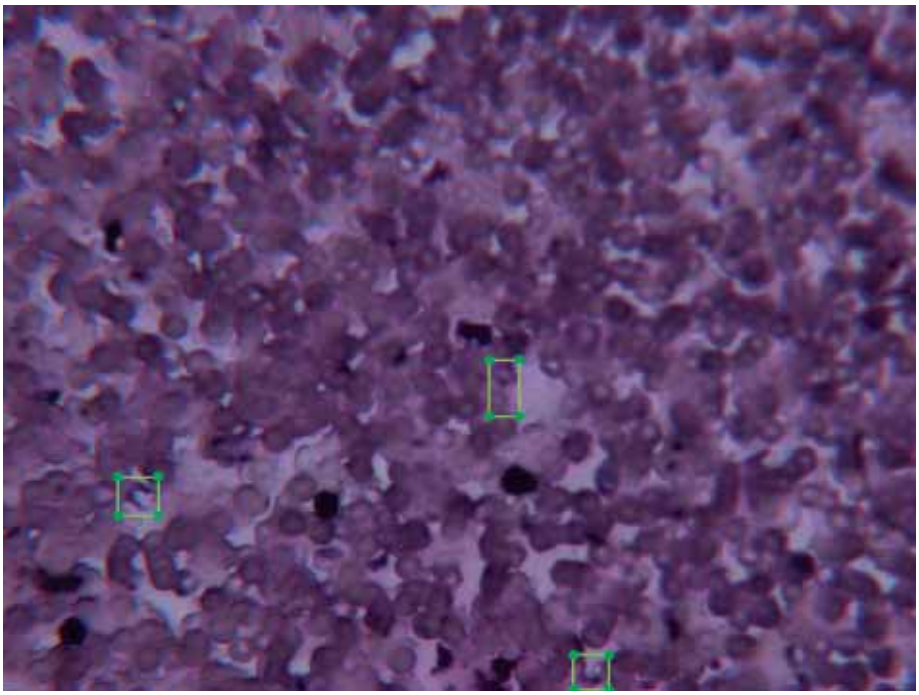


(f) ***Faster R-CNN***



(g) ***Mask R-CNN***

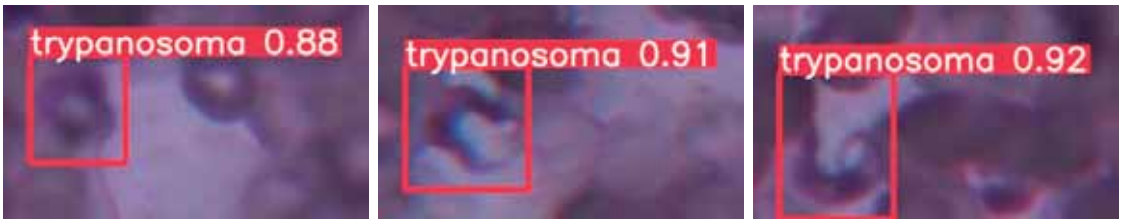
Figura 4.1: Resultados de detección en la imagen de prueba `A_041.jpg`



(a) Ground-truth

(b) *YOLOv3*

(c) *YOLOv4*



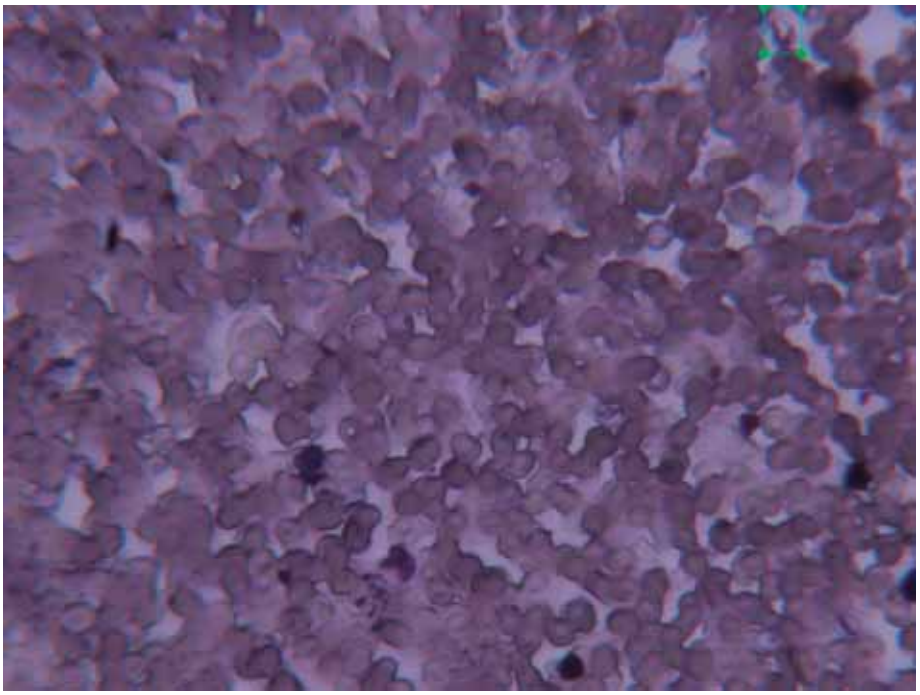
(d) ***uY4TC***

(e) ***YOLOv5***

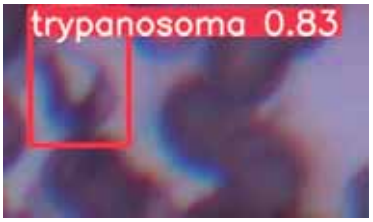
(f) ***Faster R-CNN***

(g) ***Mask R-CNN***

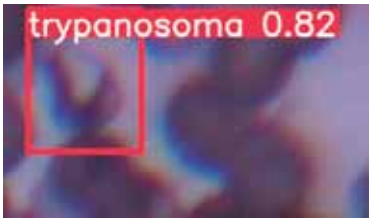
Figura 4.2: Resultados de detección en la imagen de prueba `A_087.jpg`



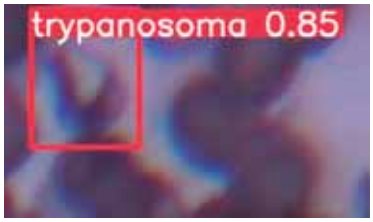
(a) Ground-truth



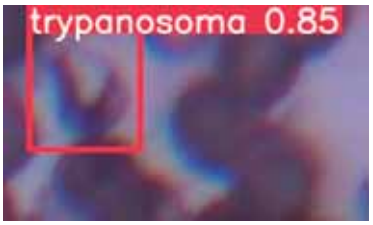
(b) *YOLOv3*



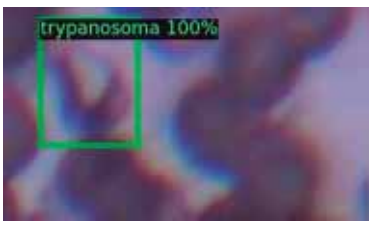
(c) *YOLOv4*



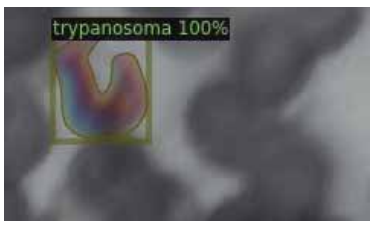
(d) uY4TC



(e) *YOLOv5*

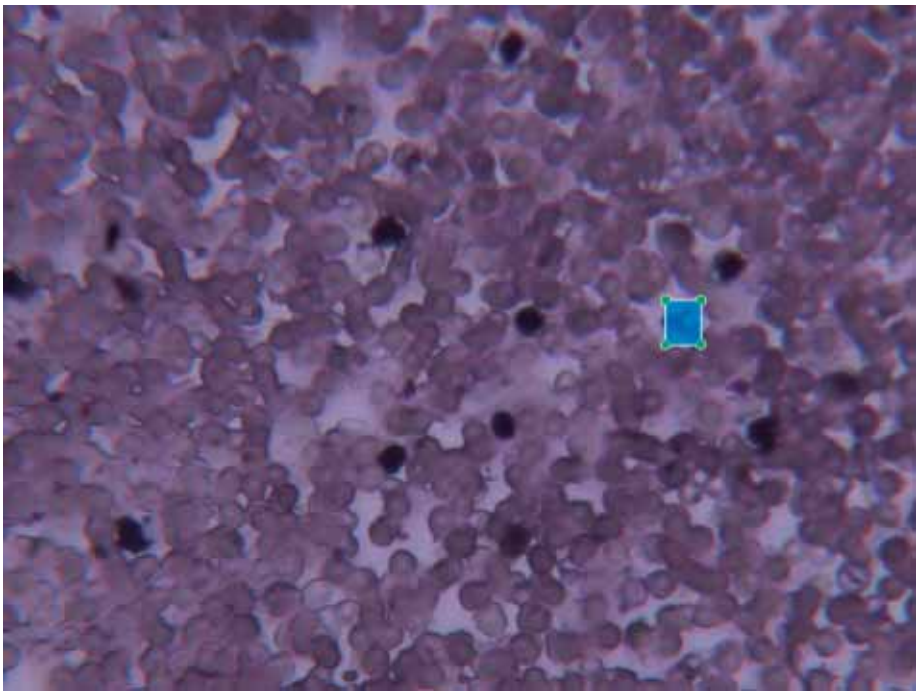


(f) *Faster R-CNN*

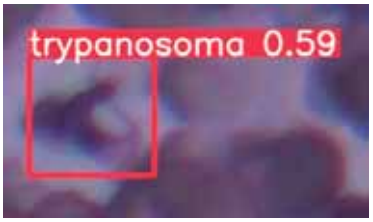


(g) *Mask R-CNN*

Figura 4.3: Resultados de detección en la imagen de prueba `A_138.jpg`

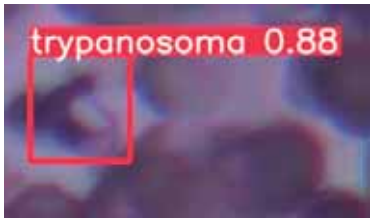


(a) Ground-truth



(b) *YOLOv3*

(c) *YOLOv4*



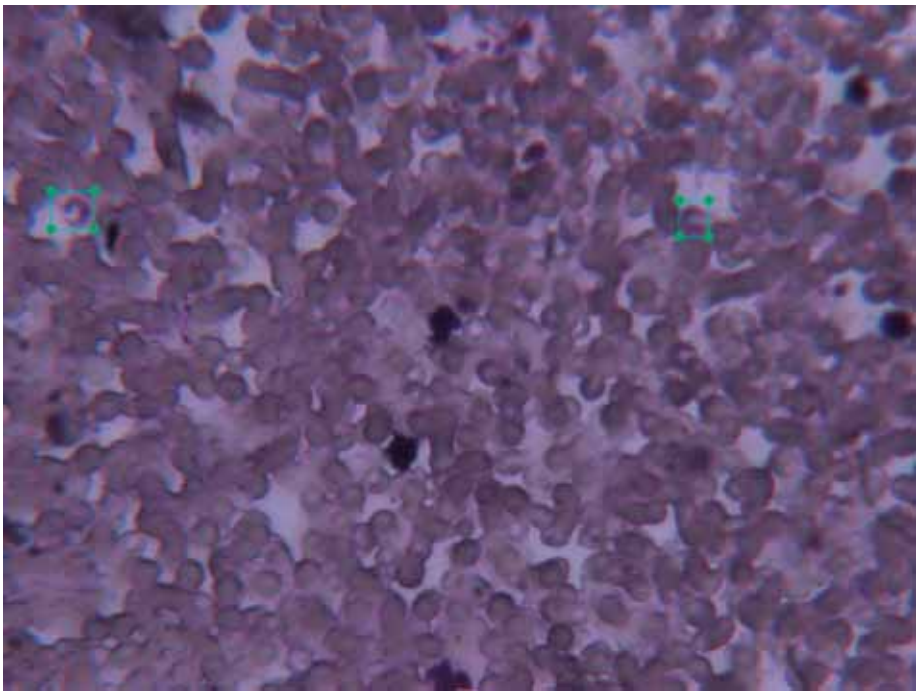
(d) uY4TC

(e) *YOLOv5*

(f) *Faster R-CNN*

(g) *Mask R-CNN*

Figura 4.4: Resultados de detección en la imagen de prueba `A_160.jpg`

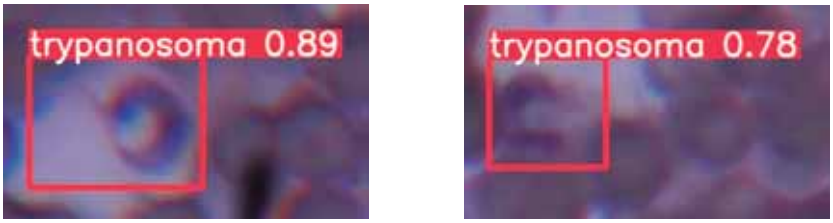


(a) Ground-truth

(b) *YOLOv3*

(c) *YOLOv4*

(d) uY4TC

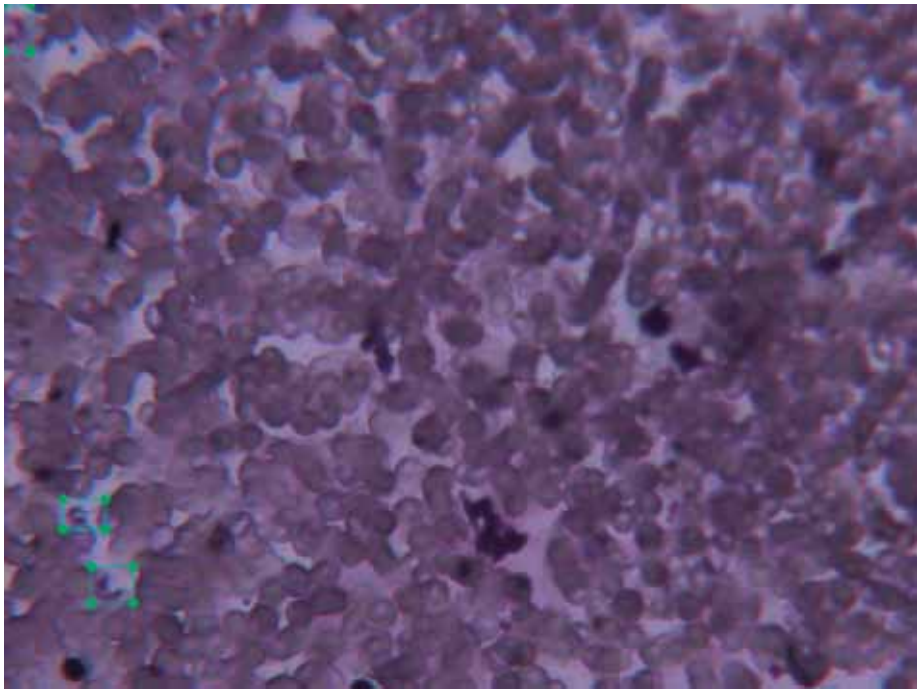


(e) *YOLOv5*

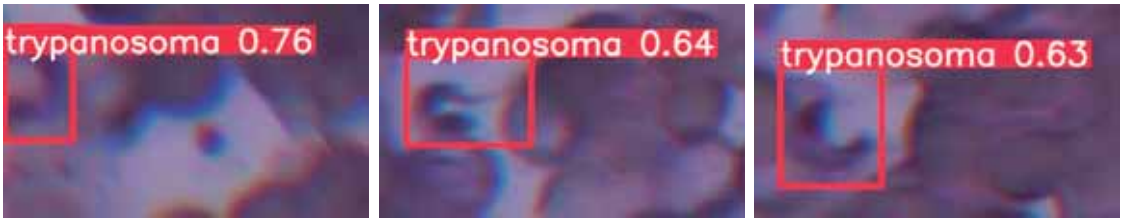
(f) *Faster R-CNN*

(g) *Mask R-CNN*

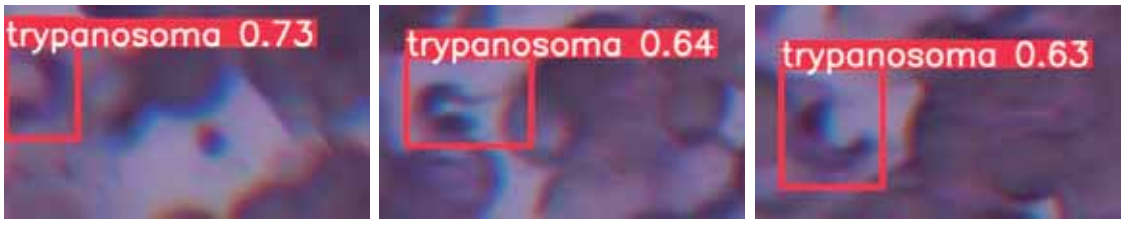
Figura 4.5: Resultados de detección en la imagen de prueba `A_238.jpg`



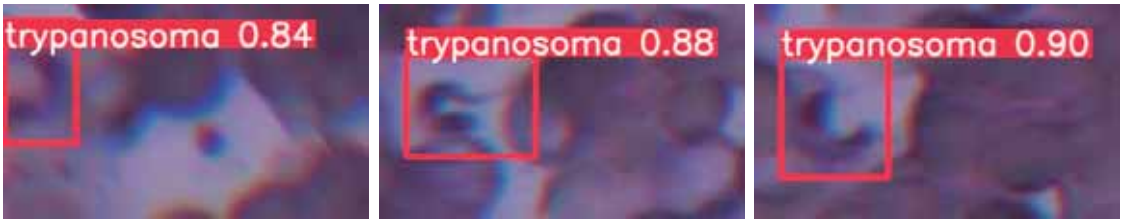
(a) Ground-truth



(b) *YOLOv3*

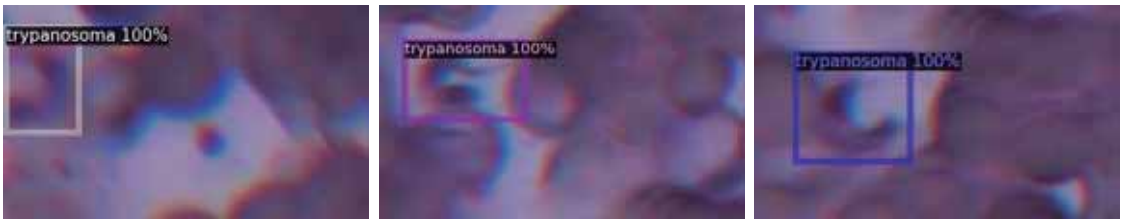


(c) *YOLOv4*

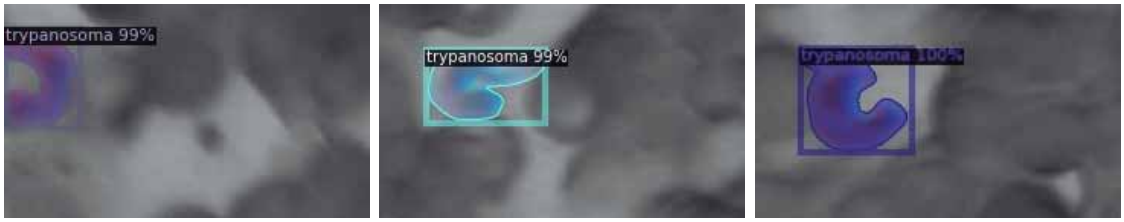


(d) **uY4TC**

(e) ***YOLOv5***

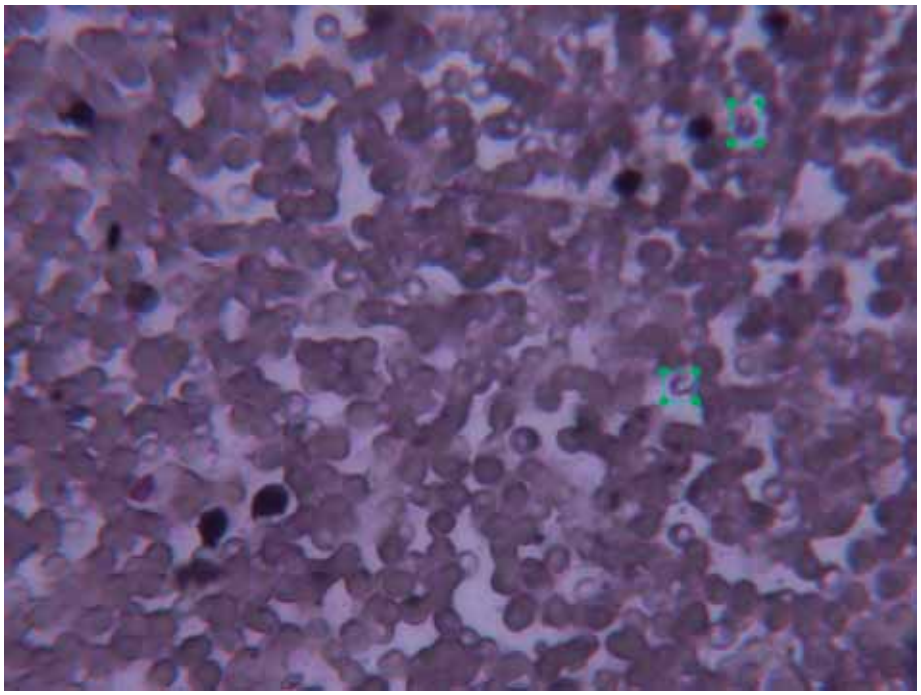


(f) ***Faster R-CNN***

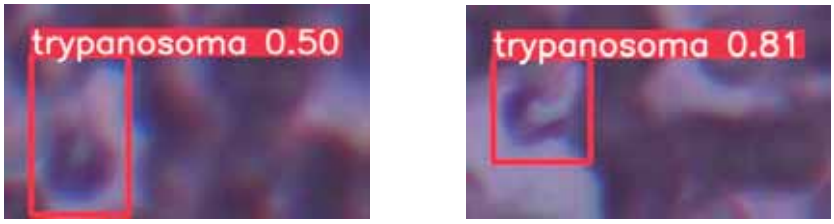


(g) ***Mask R-CNN***

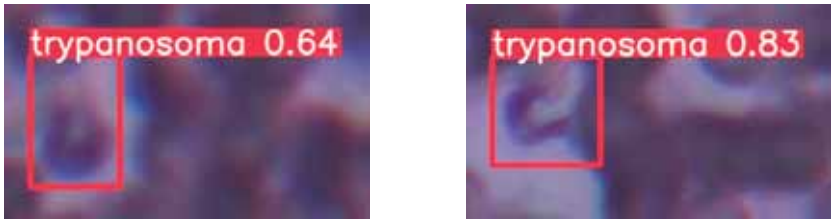
Figura 4.6: Resultados de detección en la imagen de prueba `A_284.jpg`



(a) Ground-truth



(b) *YOLOv3*

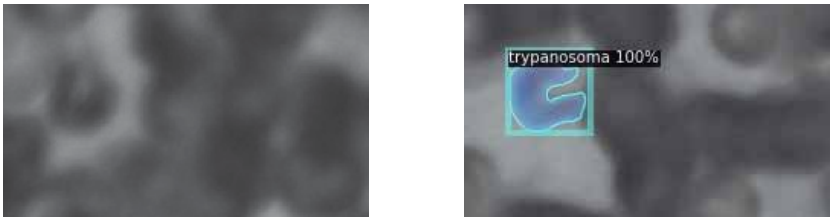


(c) *YOLOv4*

(d) uY4TC

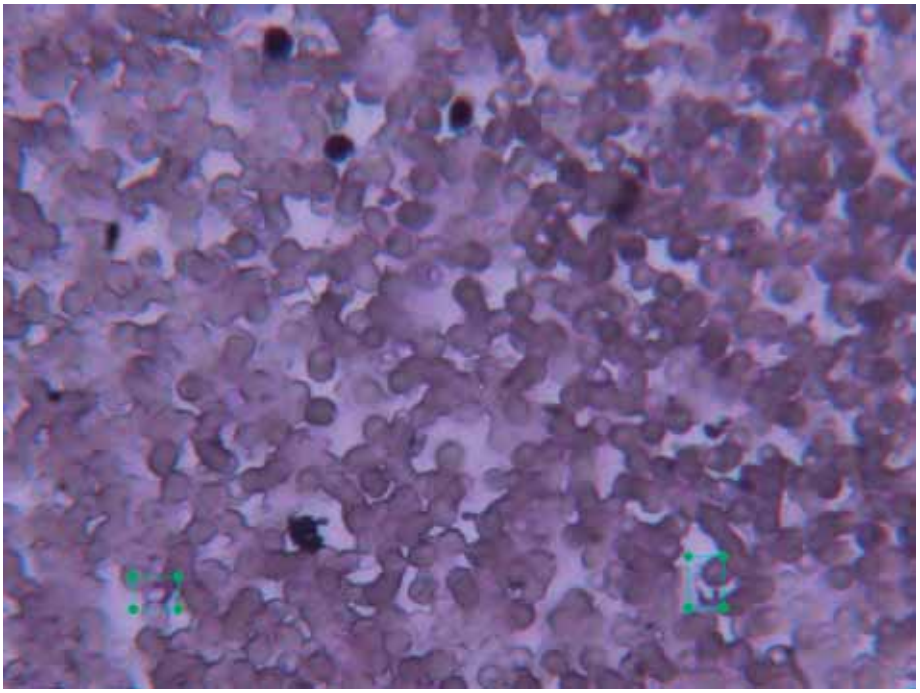
(e) *YOLOv5*

(f) *Faster R-CNN*



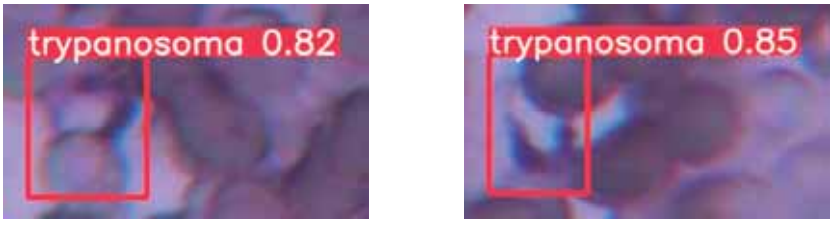
(g) *Mask R-CNN*

Figura 4.7: Resultados de detección en la imagen de prueba `A_373.jpg`



(a) Ground-truth

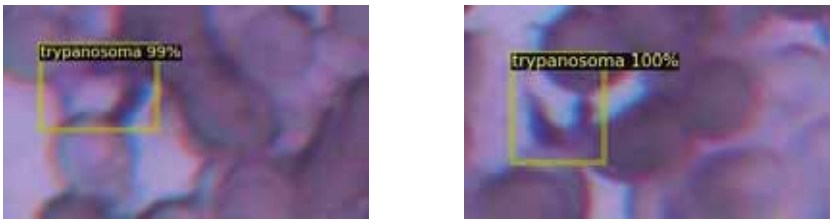
(b) *YOLOv3*



(c) *YOLOv4*

(d) uY4TC

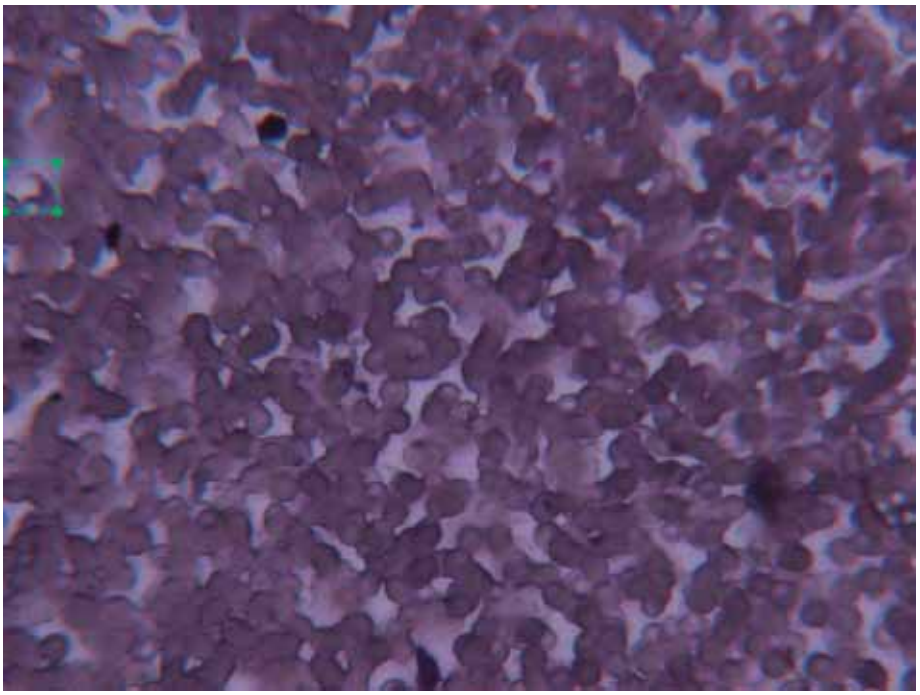
(e) *YOLOv5*



(f) *Faster R-CNN*

(g) *Mask R-CNN*

Figura 4.8: Resultados de detección en la imagen de prueba `B_084.jpg`

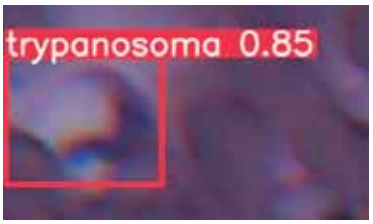


(a) Ground-truth

(b) *YOLOv3*

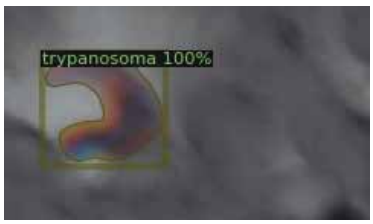
(c) *YOLOv4*

(d) uY4TC



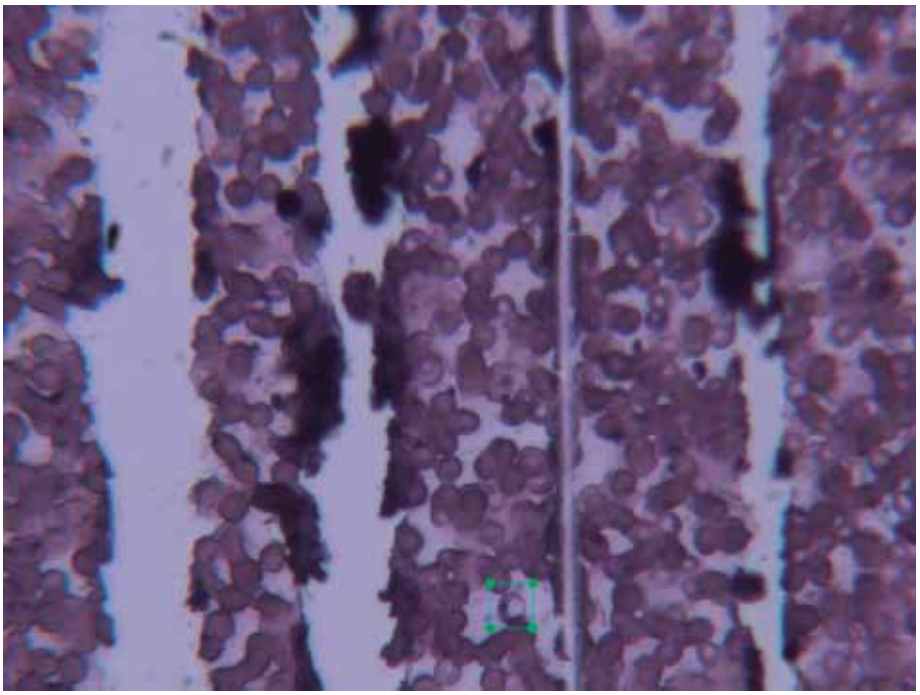
(e) *YOLOv5*

(f) *Faster R-CNN*



(g) *Mask R-CNN*

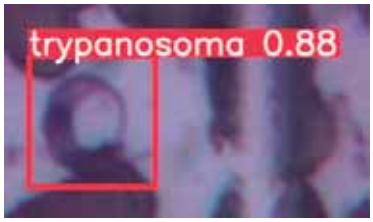
Figura 4.9: Resultados de detección en la imagen de prueba `B_330.jpg`



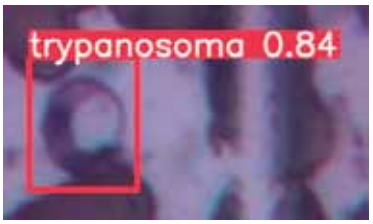
(a) Ground-truth

(b) *YOLOv3*

(c) *YOLOv4*

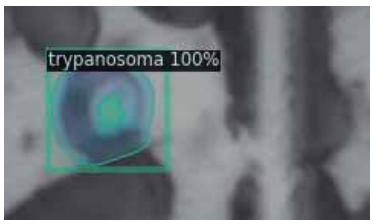


(d) uY4TC



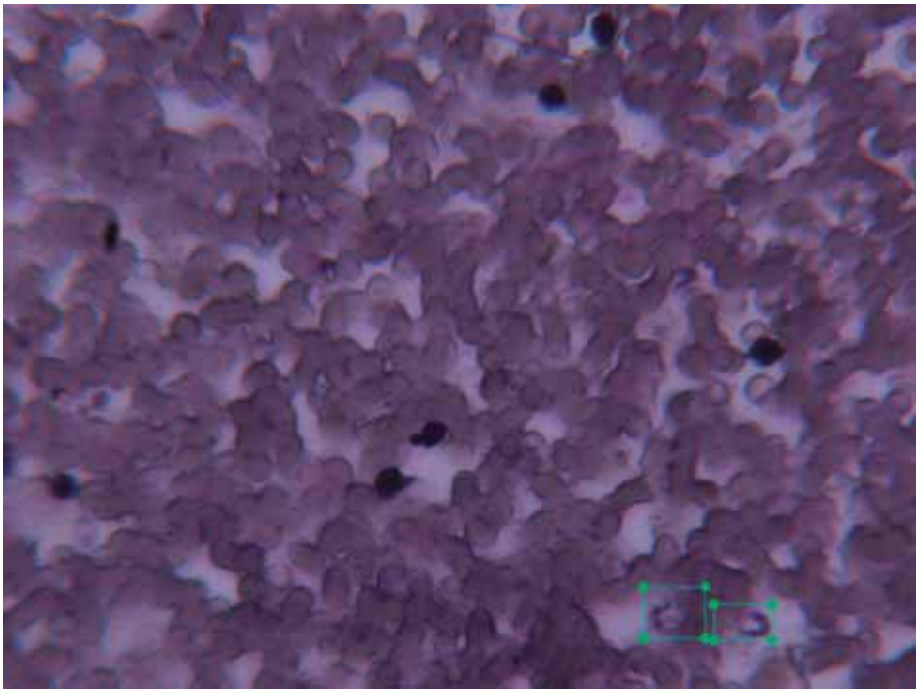
(e) *YOLOv5*

(f) *Faster R-CNN*

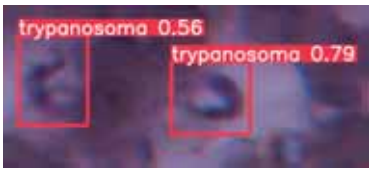


(g) *Mask R-CNN*

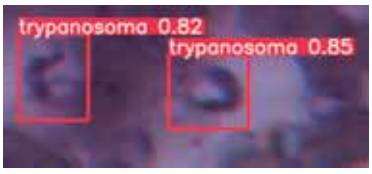
Figura 4.10: Resultados de detección en la imagen de prueba `B_460.jpg`



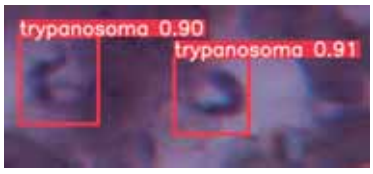
(a) Ground-truth



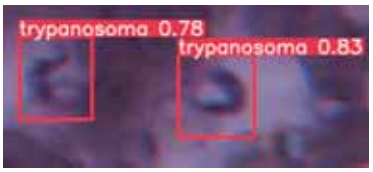
(b) *YOLOv3*



(c) *YOLOv4*

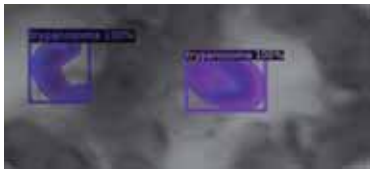


(d) uY4TC



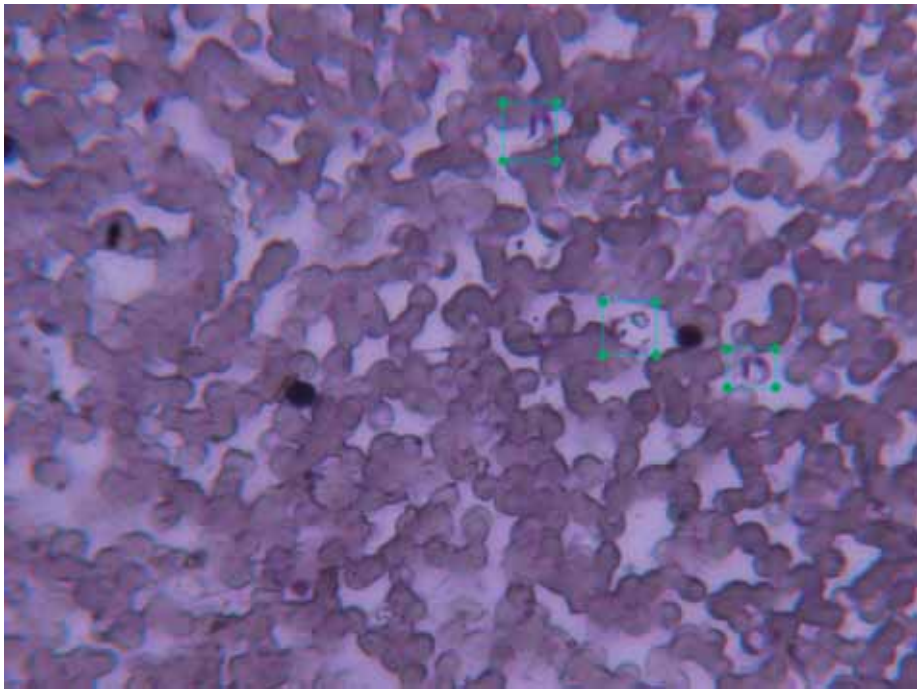
(e) *YOLOv5*

(f) *Faster R-CNN*



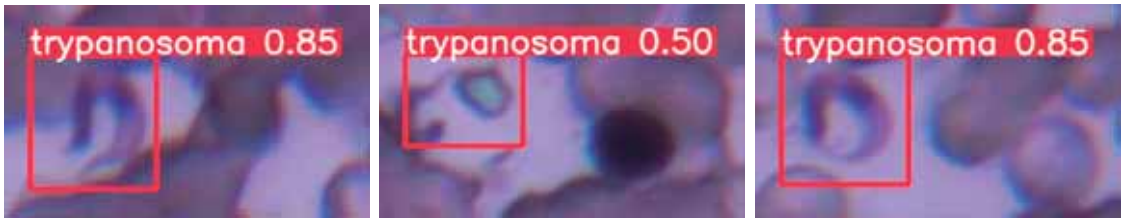
(g) *Mask R-CNN*

Figura 4.11: Resultados de detección en la imagen de prueba `B_475.jpg`



(a) Ground-truth

(b) *YOLOv3*



(c) *YOLOv4*

(d) uY4TC

(e) *YOLOv5*

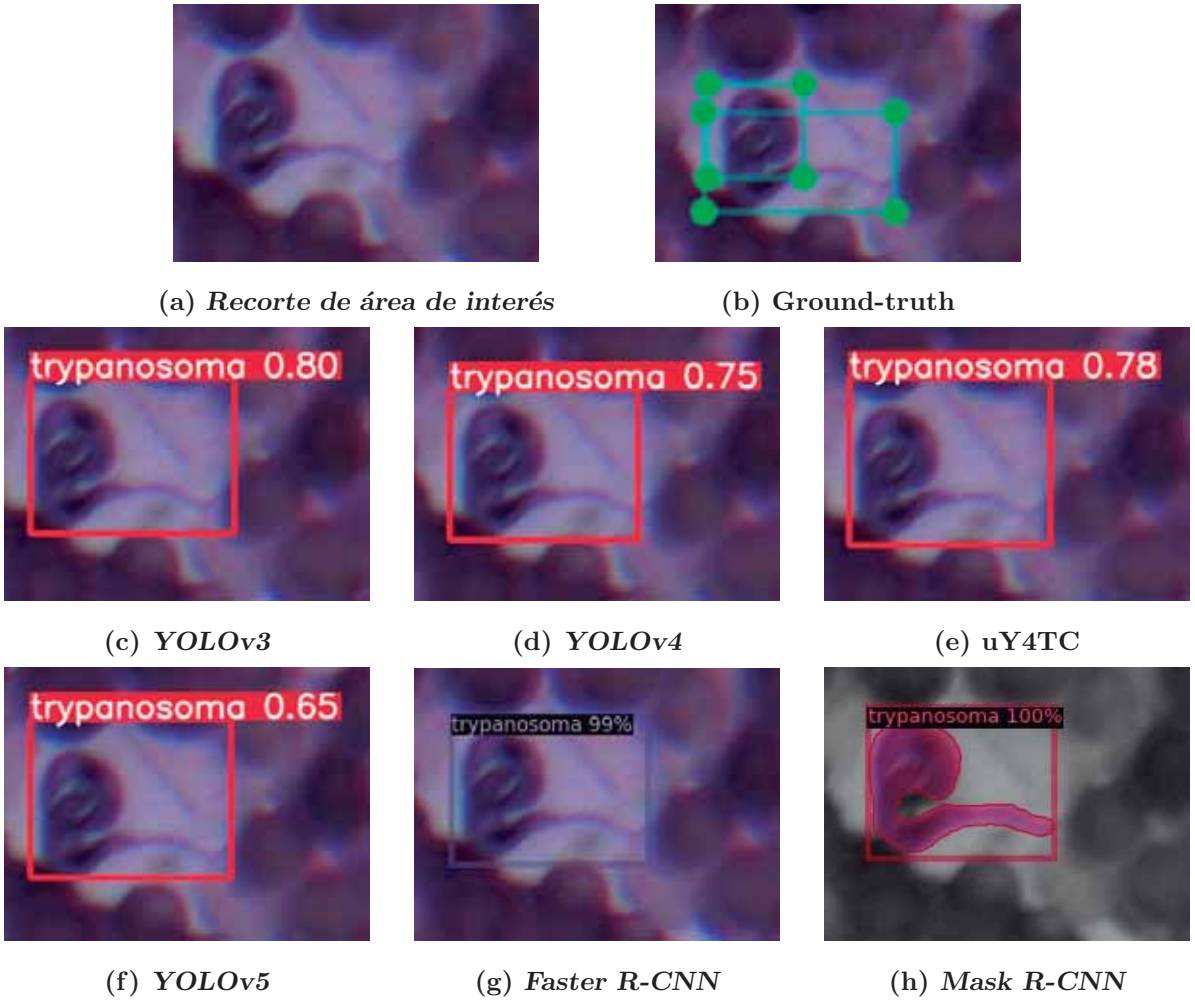
(f) *Faster R-CNN*

(g) *Mask R-CNN*

Figura 4.12: Resultados de detección en la imagen de prueba `B_562.jpg`

4.2.3. Detección interesante

En la detección con las imágenes del conjunto de prueba, se pudo notar una detección interesante del parásito *T. cruzi*, donde todos los modelos implementados, incluido la segmentación, detectan exitosamente a los parásitos presentes en la imagen de la toma de sangre, sin embargo esta detección es reportada como la detección de un único parásito.



(a) ***Recorte de área de interés*** **(b) Ground-truth**

(c) ***YOLOv3*** **(d)** ***YOLOv4*** **(e) uY4TC**

(f) ***YOLOv5*** **(g)** ***Faster R-CNN*** **(h)** ***Mask R-CNN***

Figura 4.13: Resultados de detección en la imagen de prueba `B_245.jpg`

Capítulo 5

Conclusiones

En este trabajo se realizó la implementación de un modelo de detección de objetos, referida como `uY4TC`, para la detección del parásito *Trypanosoma cruzi* en imágenes de muestras de sangre. Este modelo es una versión modificada de YOLOv4 que toma como base el código fuente de YOLOv5 (80) y la configuración de YOLOv4 de Chien-Yao (82), e incorpora métodos del estado de la técnica, siendo estos la función de pérdida de IoU eficiente y la función de activación *Serf*; así como la redefición de nuevos cuadros anclas con el fin de obtener detecciones óptimas del parásito *T. cruzi*.

La construcción de una base de datos con un conjunto de entrenamiento tratado con una aumentación de datos, permitió obtener resultados favorables, donde el modelo propuesto registró el mejor desempeño de detección en comparación con los demás modelos implementados, por lo que se puede tomar el modelo `uY4TC` como el más competente en la detección del parásito *T. cruzi*.

Además, se consigue aportar al estado de la técnica una base de datos de imágenes etiquetadas para la detección del parásito *T. cruzi* y un modelo de detección de objetos eficiente en la tarea de detección del parásito *T. cruzi* en imágenes de muestras sanguíneas, asistiendo a los especialistas en el desarrollo del diagnóstico y del tratamiento de la enfermedad de Chagas.

5.1. Comentarios

En el desarrollo de este trabajo, se estableció un punto de alto en el entrenamiento un patrón de estancamiento de las métricas `mAP`, en vez del tradicional punto bajo de la técnica de la validación cruzada. Aunque en el modelo propuesto presentaba un muy ligero repunte de sobreajuste, los demás modelos *uYOLO* aún no lo presentaban, por lo que debería ser interesante seguir con el entrenamiento hasta alcanzar dicho punto de sobreajuste.

También, se esperaraba que las primeras propuestas se llevarán dentro del marco de trabajo *Darknet* (58), pero en agunos experimentos previos con este marco de trabajo, la evaluación de desempeño de cada época podría tomar alrededor de una hora con el uso de CUDA y cerca de media hora más con el uso de CPU. Se utiliza el verbo podría ya que es lo que se llego a medir con la primera, y en otras la segunda, y luego de que el algoritmo terminaba abruptamente en la evaluación de la segunda o tercera época por la excepción de CUDA por falta de memoria.

La documentación refiere esto al gran conjunto de datos que se maneja y el manejo de los lotes y el cálculo de mini lotes (del inglés *mini-batch*). Pero aun configurando los parametros corrrespondientes, hasta incluso la reducción del conjunto de entrenamiento, los tiempos de evaluación de una época se pudieron reducir hasta 30 minutos

Pero la recolección de datos de interés sobre la implemntación en *Darknet* es tediosa dado que el marco de trabajo solo imprime en pantalla los resultados y no los guarda en un archivo externo, tanto para el entrenmiento como las evaluaciones de desempeño.

Con lo comentado anteriormente, se exploro otras alternativas de implementación como pudieran ser PyTorch y TensorFlow, con la primera más tentativa dado que se tenía conocimiento de la versión de YOLOv3 (versión de repositorio `9.6`) (81), y de YOLOv5 (versión de repositorio `6.0`), ambos modelos en el repositorio de `ultralytics` de la plataforma *GitHub* y bajo la supervisión de Glen Jocher.

Un factor que también aportó en la elección de PyTorch fue durante la búsqueda de modelos para la contrastación de resultados, en donde se encontró el marco de trabajo *Detectron2* (83) de Facebook, versión `0.6`, que también se encontraba implementado en PyTorch. Por lo que, siguiendo con la metodología, al final se construyó con una red basada en el código de YOLOv5 ya en su versión `6.0`, referida como modelo `uY4TC`.

Se debe mencionar que en un principio, se diseño una primera propuesta de modelo que incorporaría la implementación de la pérdida focal (del inglés, *focal loss*) y de la nueva función de activación *CoLU*. Con respecto a la pérdida focal, esta generó una disminución del `mAP` de unos tres puntos, quedando por debajo de todas los modelos *uYOLO*. No se

pudo realizar una investigación sobre este problema, pero puede ser referido a los mismos planteamientos que menciona Redmon en su intento de implementarla en YOLOv3 (62). En la búsqueda de una posible modificación en la función de pérdida, se optó por la complementación de la pérdida de *IoU* eficiente.

En el caso de la función *CoLU*, se propuso su implementación dentro de las capas convolucionales ajenas al *backbone* pero en experimentos previos, esta función generaba un peligroso sobreajuste del cálculo de números, ya que en su definición (44) se utiliza una función exponencial dentro de otra exponencial provocando un número casi imposible de calcular. La biblioteca de PyTorch en estas situaciones reescribe el resultado como `inf`, provocando varios errores de cálculo y provocando mapas con valores `NaN` (del inglés *Not A Number*). Tal vez cuando procesa ese `inf` en una fracción se vuelve cero, pero cuando se toma el cero en otra división, frecuentemente en la derivada, ya no es posible procesar un adecuado cálculo. Esto se podría arreglar con el uso de umbrales, pero dado al tiempo consumido con varias evaluaciones con diferentes umbrales, se decidió reemplazar su implementación por la función *SiLU*, dado que es la función de activación principal de YOLOv5.

5.2. Trabajo a futuro

Se propone un conjunto de actividades como trabajo a futuro, retomando fundamentalmente lo desarrollado y alcanzado en este trabajo.

Primordialmente, se debe atender a una posible implementación de este trabajo dentro del marco de trabajo *Darknet*, donde se agregarían la evaluación de la validación cruzada y el cálculo de las precisiones promedio con el umbral `0.75` y sobre los diez umbrales.

Para enriquecer la relación de resultados de los experimentos, se plantea experimentar con otros modelos del estado de la técnica como pudieran ser las versiones YOLO desarrollados por Chien-Yao (Scaled-YOLOv4 y YOLOR), que reportan mejores resultados sobre YOLOv4, o incluso poder construir o modificar el modelo propuesto `uY4TC` sobre estas versiones con la posibilidad de alcanzar mejores resultados.

Otro punto a alcanzar, es el de poder construir una adecuada implementación de *CoLU*, donde permitiría rescatar la propuesta de su implementación en la red `uY4TC`. El producto de esta acción, que sustituiría la función de activación *SiLU* por *CoLU*, tendría como nombre complementario provisional *Serf-CoLU*, terminología inspirada en los modelos series de YOLOv4 como `YOLOv4-Mish`.

Entonces, el modelo construido en este trabajo podría ser referido como la modelo `uY4TC Serf-SiLU`, donde además se desarrollarían dos modelos adicionales, donde el primero hace uso únicamente de la función *Serf*, así como el segundo modelo solo de la función *CoLU*, así como un posible traspaso en *Darknet*.

Bibliografía

[1] N. González, V. Estrada, and A. Febles, "Estudio y selección de las técnicas de Inteligencia Artificial para el diagnóstico de enfermedades," *Revista de Ciencias Médicas de Pinar del Río*, vol. 22, no. 3, pp. 534–544, 2018.

[2] J. Rodríguez, R. Biswal, and E. Cruz, "Algoritmos de aprendizaje automático de vanguardia para el diagnóstico de enfermedades," *Research in Computing Science*, vol. 148, pp. 455–468, 12 2019.

[3] OMS, "La enfermedad de Chagas (tripanosomiasis americana)." https://www.who.int/es/news-room/fact-sheets/detail/chagas-disease-(american-trypanosomiasis), 2021. Consultado el 7/12/21.

[4] OPS, "Enfermedad de Chagas." https://www.paho.org/es/temas/enfermedad-chagas, 2020. Consultado el 7/12/21.

[5] A. Ojeda-Pat, A. Martin-Gonzalez, and V. Uc-Cetina, "Revisión de métodos de aprendizaje automático para detectar al parásito de la enfermedad de Chagas," *Investigación y Ciencia de la Universidad Autónoma de Aguascalientes*, pp. 91–98, 06 2020.

[6] RAE and ASALE, "Diccionario panhispánico de dudas." https://www.rae.es/dpd/, 2005. Consultado el 7/12/21.

[7] V. Uc-Cetina, C. Brito-Loeza, and H. Ruiz-Piña, "Chagas Parasites Detection through Gaussian Discriminant Analysis," *Abstraction and Application*, vol. 8, pp. 6–17, 08 2013.

[8] R. Soberanis-Mukul, V. Uc-Cetina, C. Brito-Loeza, and H. Ruiz-Piña, "An automatic algorithm for the detection of Trypanosoma cruzi parasites in blood sample images," *Computer Methods and Programs in Biomedicine*, vol. 112, no. 3, pp. 633–639, 2013.

[9] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.

[10] R. D. Soberanis Mukul, "Algoritmos de segmentación de Trypanosoma cruzi en imágenes de muestras sanguíneas," Master's thesis, Universidad Autónoma de Yucatán, 2014.

[11] V. N. Vapnik, "Statistical learning theory," 1998.

[12] P. Bartlett, Y. Freund, W. S. Lee, and R. E. Schapire, "Boosting the margin: A new explanation for the effectiveness of voting methods," *The annals of statistics*, vol. 26, no. 5, pp. 1651–1686, 1998.

[13] R. D. Soberanis Mukul, "Detección de Trypanosoma Cruzi en Imágenes obtenidas a partir de muestras sanguíneas," Master's thesis, Universidad Autónoma de Yucatán, 2012.

[14] V. Uc-Cetina, C. Brito-Loeza, and H. Ruiz-Piña, "Chagas parasite detection in blood images using AdaBoost," *Computational and mathematical methods in medicine*, vol. 2015, p. 139681, 2015.

[15] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, 2001.

[16] D. Kumari and S. Chokkalingam, "Deep Convolutional Neural Networks (CNN) for Medical Image Analysis," *SSRN Electronic Journal*, 01 2019.

[17] A. Ojeda-Pat, A. Martin-Gonzalez, C. Brito-Loeza, H. Ruiz-Piña, and D. Ruz-Suarez, "Effective residual convolutional neural network for Chagas disease parasite segmentation," *Medical and biological engineering and computing*, March 2022.

[18] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds.), (Cham), pp. 234–241, Springer International Publishing, 2015.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[20] T. J. Sørensen, *A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on Danish commons.* København, 1948.

[21] F. Guhl, "Photograph showing an adult specimen of Triatoma dimidiata from Colombia." https://es.wikipedia.org/wiki/Archivo:Triatoma_dimidiata-adult.jpg, 2008.

[22] J. F. Ortega Campos, "Estudio fisicoquímico y biológico de complejos metálicos de platino y paladio con ligantes bioactivos contra *Trypanosoma cruzi*," Master's thesis, Universidad de Chile, 2020.

[23] C. Ponce, E. Ponce, E. Vinelli, A. Montoya, V. de Aguilar, A. Gonzalez, B. Zingales, R. Rangel-Aldao, M. J. Levin, J. Esfandiari, E. S. Umezawa, A. O. Luquetti, and J. F. da Silveira, "Validation of a Rapid and Reliable Test for Diagnosis of Chagas' Disease by Detection of ¡i¿Trypanosoma cruzi¡/i¿-Specific Antibodies in Blood of Donors and Patients in Central America," *Journal of Clinical Microbiology*, vol. 43, no. 10, pp. 5065–5068, 2005.

[24] K. Egüez, J. Alonso-Padilla, C. Terán Calderón, Z. Chipana, W. García, F. Torrico, J. Gascón, D. Lozano, and M.-J. Pinazo, "Rapid diagnostic tests duo as alternative to conventional serological assays for conclusive Chagas disease diagnosis," *PLoS neglected tropical diseases*, vol. 11, p. e0005501, 04 2017.

[25] J. Sidey-Gibbons and C. Sidey-Gibbons, "Machine learning in medicine: a practical introduction," *BMC Medical Research Methodology*, vol. 19, 03 2019.

[26] M. Alonso, "Ciclos Formato de Imagen." https://wiki.ead.pucv.cl/Ciclo_Formatos_de_imagen_-_Macarena_Alonso, 2020.

[27] B. Medina, "Convolución en 2D (Filtrado espacial)." https://bryanmed.github.io/conv2d/, 2019.

[28] T. Ganegedara, "Intuitive understanding of 1D, 2D, and 3D convolutions in convolutional neural networks." https://stackoverflow.com/q/42883547, 2019. Modificado y consultado el 7/12/21.

[29] C. Solomon and T. Breckon, *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. Wiley Publishing, 1st ed., 2011.

[30] R. F. de Mello and M. A. Ponti, *Machine Learning - A Practical Approach on the Statistical Learning Theory*. Springer, 2018.

[31] IBM, "Redes Neuronales." https://www.ibm.com/mx-es/cloud/learn/neural-networks, 2020. Consultado el 7/12/21.

[32] S. Skansi, *Introduction to Deep Learning: From Logical Calculus to Artificial Intelligence*. Springer Publishing Company, Incorporated, 1st ed., 2018.

[33] A. Trask, *Grokking Deep Learning*. USA: Manning Publications Co., 1st ed., 2019.

[34] J. Walinga, "Diagram of basic neuron and components." https://commons.wikimedia.org/wiki/File:Components_of_neuron.jpg, 2014. Modificado y consultado el 7/12/21.

[35] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, pp. 947–951, 2000.

[36] I. d. J. Ávila Uc, "Red Neuronal Convolucional para Deteccion y Clasificación de Señales de Tránsito," Bachelor's Thesis, Universidad Autónoma de Yucatán, 2020.

[37] Y. A. Abramovich, C. D. Aliprantis, and O. Burkinshaw, "Another characterization of the invariant subspace problem," *Operator Theory in Function Spaces and Banach Lattices.* The A.C. Zaanen Anniversary Volume*, Operator Theory: Advances and Applications*, vol. 75, pp. 15–31, 1995. Birkhäuser Verlag.

[38] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[39] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier Nonlinearities Improve Neural Network Acoustic Models," in *In Proceedings of International Conference on Machine Learning (ICML)*, 2013.

[40] D. Hendrycks and K. Gimpel, "Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units," *ArXiv*, vol. abs/1606.08415, 2016.

[41] Papers with Code team, "Activation Functions." https://paperswithcode.com/methods/category/activation-functions, 2021. Consultado el 7/12/21.

[42] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning," *Neural networks : the official journal of the International Neural Network Society*, vol. 107, pp. 3–11, 2018.

[43] D. Misra, "Mish: A self regularized non-monotonic neural activation function," *arXiv preprint arXiv:1908.08681*, 2019.

[44] A. Vagerwal, "Deeper Learning with CoLU Activation," vol. abs/2112.12078, 2021.

[45] S. Nag and M. Bhattacharyya, "SERF: Towards better training of deep neural networks using log-Softplus ERror activation Function," *ArXiv*, vol. abs/2108.09598, 2021.

[46] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," 2020.

[47] Z.-Q. Zhao, P. Zheng, S. tao Xu, and X. Wu, "Object detection with deep learning: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, pp. 3212–3232, 2019.

[48] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, pp. 1137–1149, 2015.

[49] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," in *ECCV*, 2016.

[50] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2980–2988, 2017.

[51] Z. Tian, C. Shen, H. Chen, and T. He, "FCOS: Fully Convolutional One-Stage Object Detection," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9626–9635, 2019.

[52] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks," *ArXiv*, vol. abs/1605.06409, 2016.

[53] E. Upschulte, S. Harmeling, K. Amunts, and T. Dickscheid, "Contour proposal networks for biomedical instance segmentation," *Medical image analysis*, vol. 77, p. 102371, 2022.

[54] Z. Yang, S. Liu, H. Hu, L. Wang, and S. Lin, "RepPoints: Point Set Representation for Object Detection," *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 9656–9665, 2019.

[55] X. Jiang, A. Hadid, Y. Pang, E. Granger, and X. Feng, *Deep Learning in Object Detection and Recognition*. Springer Publishing Company, Incorporated, 1st ed., 2019.

[56] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision*, vol. 111, pp. 98–136, Jan. 2015.

[57] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *ECCV*, 2014.

[58] A. Bochkovskiy, "AlexeyAB/Darknet." https://github.com/AlexeyAB/darknet, 2020. Consultado el 7/12/21.

[59] S. McCann, "Average Precision." https://sanchom.wordpress.com/tag/average-precision/, 2011. Modificado y consultado el 7/12/21.

[60] COCO Consortium, "COCO Common Objects in Context." https://cocodataset.org, 2015. Consultado el 7/12/21.

[61] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6517–6525, 2017.

[62] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," 2018.

[63] J. Redmon, "Darknet: Open Source Neural Networks in C." http://pjreddie.com/darknet/, 2013–2016.

[64] G. A. Chávez Fragoso, "Reconocimiento de marcadores con redes profundas," Master's thesis, Centro de Investigación y de Estudios Avanzados del Instituo Politécnico Nacional - Unidad Zacatenco, 2020.

[65] J. Nelson and J. Solawetz, "Responding to the Controversy about YOLOv5." https://blog.roboflow.com/yolov4-versus-yolov5/, 2020. Consultado el 7/12/21.

[66] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019.

[67] G. Jocher, L. Changyu, A. Hogan, L. Y. , changyu98, P. Rai, and T. Sullivan, "ultralytics/yolov5: Initial Release," June 2020.

[68] X. Long, K. Deng, G. Wang, Y. Zhang, Q. Dang, Y. Gao, H. Shen, J. Ren, S. Han, E. Ding, and S. Wen, "PP-YOLO: An Effective and Efficient Implementation of Object Detector," *ArXiv*, vol. abs/2007.12099, 2020.

[69] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-YOLOv4: Scaling Cross Stage Partial Network," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13024–13033, 2021.

[70] X. Huang, X. Wang, W. Lv, X. Bai, X. Long, K. Deng, Q. Dang, S. Han, Q. Liu, X. Hu, D. Yu, Y. Ma, and O. Yoshie, "PP-YOLOv2: A Practical Object Detector," *ArXiv*, vol. abs/2104.10419, 2021.

[71] C.-Y. Wang, I.-H. Yeh, and H. Liao, "You Only Learn One Representation: Unified Network for Multiple Tasks," *ArXiv*, vol. abs/2105.04206, 2021.

[72] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "YOLOX: Exceeding YOLO Series in 2021," *ArXiv*, vol. abs/2107.08430, 2021.

[73] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, "CSPNet: A New Backbone that can Enhance Learning Capability of CNN," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1571–1580, 2020.

[74] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, pp. 1904–1916, 2015.

[75] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path Aggregation Network for Instance Segmentation," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8759–8768, 2018.

[76] B. Gong, D. Ergu, Y. Cai, and B. Ma, "Real-Time Detection for Wheat Head Applying Deep Neural Network," *Sensors*, vol. 21, no. 1, 2021.

[77] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression," in *AAAI*, 2020.

[78] T.-Y. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature Pyramid Networks for Object Detection," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 936–944, 2017.

[79] Y.-F. Zhang, W. Ren, Z. Zhang, Z. Jia, L. Wang, and T. Tan, "Focal and Efficient IOU Loss for Accurate Bounding Box Regression," *ArXiv*, vol. abs/2101.08158, 2021.

[80] G. Jocher, A. Stoken, A. Chaurasia, J. Borovec, NanoCode012, TaoXie, Y. Kwon, K. Michael, L. Changyu, J. Fang, A. V, Laughing, tkianai, yxNONG, P. Skalski, A. Hogan, J. Nadar, imyhxy, L. Mammana, AlexWang1900, C. Fati, D. Montes, J. Hajek, L. Diaconu, M. T. Minh, Marc, albinxavi, fatih, oleg, and wanghaoyang0106, "ultralytics/yolov5: v6.0 - YOLOv5n 'Nano' models, Roboflow integration, TensorFlow export, OpenCV DNN support," Oct. 2021.

[81] G. Jocher, Y. Kwon, guigarfr, perry0418, J. Veitch-Michaelis, Ttayu, D. Suess, F. Baltacı, G. Bianconi, IlyaOvodov, Marc, e96031413, C. Lee, D. Kendall, Falak, F. Reveriano, FuLin, GoogleWiki, J. Nataprawira, J. Hu, LinCoce, LukeAI, NanoCode012, NirZarrabi, O. Reda, P. Cohen, P. Skalski, SergioSanchezMontesUAM, S. Song, and T. M. Shead, "ultralytics/yolov3: v9.6.0 - YOLOv5 v6.0 release compatibility update for YOLOv3," Nov. 2021.

[82] C.-Y. Wang, "WongKinYiu/PyTorch_YOLOv4." https://github.com/WongKinYiu/PyTorch_YOLOv4, 2020. Consultado el 7/12/21.

[83] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, "Detectron2." https://github.com/facebookresearch/detectron2, 2019.