Universidad Autonóma de Yucatán

Facultad de Matemáticas

Detección de gránulos de polen de cuatro especies vegetales de importancia apícola en el estado de Yucatán usando redes neuronales convolucionales

TESIS

Presentada por:

Miguel Ángel Canul Chin

En opción al título de:

Ingeniero en Computación

Asesora:

Dra. Anabel Martín González

Mérida, Yucatán, México

2022

A la Facultad de Matemáticas.

Agradecimientos

Quiero agradecer a mi madre, a ella le debo todo lo que he podido lograr, por enseñarme a nunca rendirme en nada por más difícil que pareciera. Por apoyarme económicamente, por su incansable esfuerzo para asegurarse de que nunca nos faltara nada a mi y a mi hermana.

A mi asesora, la Dra. Anabel Martín, por proporcionarme los conocimientos necesarios para realizar este trabajo de tesis y por hacer que me interesara más en el tema del aprendizaje profundo.

Resumen

En el estado de Yucatán se encuentran alrededor de 2400 especies vegetales que representan aproximadamente el 10 % de toda la flora mexicana y aproximadamente 600 son de importancia melífera. La miel que se produce en el estado de Yucatán contribuye al 30 % del total de la producción nacional, además, tiene propiedades que favorecen su consumo, como el sabor, color y aroma. Debido a estas propiedades importantes, cerca del 90 % de la miel se destina al mercado internacional. Las mieles provenientes de la floración del tajonal (*Viguiera dentata*) y t´sit´silche´ (*Gymnopodium floribundum*) son las que mayor contribución tienen en la producción apícola de la península.

Los gránulos de polen son estructuras microscópicas que presentan características que resultan de gran ayuda en diversas áreas. Una de las características más importantes es la diversidad morfológica que presentan los gránulos de polen, estos adoptan formas lo suficientemente diferentes entre especies, lo que hace posible identificar el tipo de miel y qué planta los produjo. Sin embargo, el estudio de gránulos de polen requiere una contabilización e identificación de estos, este conteo e identificación de los gránulos de polen son tareas muy lentas (que pueden tomar incluso meses) y difíciles, pues, requiere de expertos altamente entrenados para preparar las muestras y posteriormente reconocerlos por medio de inspección visual, utilizando microscopios ópticos. En el estado de Yucatán, y en México en general, es muy escaso este tipo de personal.

Automatizar este proceso mediante el uso de redes neuronales convolucionales capaces de clasificar y localizar múltiples gránulos de polen en una imagen en tan sólo milisegundos sería de gran utilidad en esta rama.

En este trabajo de tesis se propone un modelo de detección y clasificación de gránulos de polen basado en las arquitecturas YOLOV4 y Res2Unet, obteniendo un mAP@0.5 de 98.83 % con menos parámetros que la versión original de YOLOV4. Le llamamos al modelo WRes2Net.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En el estado de Yucatán se han registrado alrededor de 2400 especies vegetales (aproximadamente el 10 % de toda la flora mexicana) y aproximadamente 600 son de importancia melífera. La miel que se produce en el estado de Yucatán contribuye al 30 % del total de la producción a nivel nacional, además tiene ciertas propiedades como el color, sabor y aroma que favorecen su consumo, por lo que cerca del 90 % se destina al mercado internacional. Las mieles provenientes de la floración del tajonal (*Viguiera dentata*) y t´sit´silche´ (*Gymnopodium floribundum*) son las que mayor contribución tienen en la producción apícola de la península (alrededor del 90 %) [1].

Es la palinología, rama de la botánica, la encargada de estudiar los gránulos de polen, estas estructuras microscópicas presentan características que resultan ser de gran ayuda en otras áreas, por ejemplo, la investigación criminal, proveer datos para el estudio del cambio climático, ayudar a la exploración petrolera para encontrar posibles yacimientos de petróleo [2, 3]. Una de las características más importantes es la gran diversidad morfológica que presentan los gránulos de polen, estos adoptan formas lo suficientemente diferentes entre especies, lo que hace posible identificar el tipo de miel y qué planta los produjo.

Sin embargo, el estudio de los gránulos de polen requiere una contabilización e identificación de estos, este conteo e identificación de gránulos de polen son tareas muy lentas (que pueden tomar meses) y difíciles, pues, aún se sigue haciendo por medio de inspección visual, utilizando microscopios ópticos, y solamente por expertos altamente entrenados. En el estado de Yucatán, y en México en general, es muy escaso este tipo de personal.

Automatizar este proceso mediante la utilización de algoritmos computacionales capaces de clasificar y localizar múltiples gránulos de polen en una imagen en tan sólo milisegundos sería de gran utilidad en esta rama.

Avances recientes tanto en la arquitectura y proceso de manufactura de las GPUs (Unidades de procesamiento grafico, por sus siglas en inglés) como en las redes neuronales completamente convolucionales (*FCNN* por sus siglas en inglés) han hecho factible la utilización de las *FCNN* , dado que, son más eficientes y ya no se requieren dispositivos especializados o demasiado costosos para poder entrenarlos y utilizarlos, reduciendo el tiempo que puede llevar meses a unas cuantas horas.

1.1. Antecedentes

Actualmente existen pocos trabajos en la detección y clasificación de gránulos de polen que utilicen *FCNNs*, la mayoría de los métodos existentes se concentran principalmente en la clasificación. Los métodos principales utilizados en la automatización de la detección y clasificación de gránulos de polen, mediante algoritmos computacionales, realizan un preprocesado a las imágenes para extraer características que pudieran ser relevantes para la clasificación, estos métodos suelen ser: de morfología, basados en texturas e híbridos. El primero se centra en las características visuales como la forma, circularidad de los gránulos, perímetro, área, características geométricas en tres dimensiones, componentes radiales y angulares de una representación de voxel de los gránulos de polen, los cambios a lo largo del contorno de los gránulos de polen, entre otros. El segundo método estudia la textura de la superficie de los gránulos utilizando características obtenidas a partir de la matriz de coocurrencia de niveles de gris (GLCM por sus siglas en inglés). El tercer método aborda el estudio combinando varias características como, características geométricas, descriptores de Fourier, características basadas en el color, características de textura y morfológicas [4]. Estas características extraídas se ingresan a algoritmos de clasificación para intentar discriminar clases de polen utilizando dicha información.
Existen otras técnicas para extraer información relevante presente en una imagen pero el problema de todos estos métodos es que al final se deben escoger manualmente qué características son relevantes para la clasificación, resulta difícil saber si se están considerando todas las características posibles que en verdad ayuden a discriminar entre una clase de polen y otra.
En [5] utilizan características de la textura de segundo orden a partir de las matrices de coocurrencia de nivel de gris de imágenes de polen para posteriormente usar esos datos como entrada en una red neuronal artificial para clasificar los gránulos, consiguiendo una exactitud del 88 %.
En [6] proponen un método no supervisado para analizar gránulos de polen en imá-

genes sin etiquetar, usan una base de datos de 650 imágenes de gránulos de polen recolectadas de miel, primero se aplica un submuestreo para reducir el tamaño de la imagen proveniente del microscopio, posteriormente utilizan una red neuronal convolucional basada en YOLO [7] para obtener un conjunto de cuadros delimitadores para todos los pólenes presentes, obteniendo así las ubicaciones de los pólenes, estos luego se recortan y se amplifican para recombinarse en la imagen original, una vez obtenidos estos recortes cada uno pasa por un codificador basado en la arquitectura de la red convolucional VGG16 y por último se utiliza el algoritmo de K-medias para agrupar los resultados.

En [8] utilizan un ensamble de redes neuronales convolucionales para clasificar imágenes de polen, usando la base de datos Pollen-13k, que consiste en 13000 imágenes de polen, consiguiendo una exactitud del 97.28 % en el reconocimiento.

En [4] utilizan un modelo con seis capas convolucionales y una capa completamente conectada para clasificar un total de 30 tipos distintos de polen, alcanzando un 94 % de precisión. La base de datos utilizada consiste en imágenes tomadas usando un microscopio óptico y un microscopio de barrido de electrones. En [9] se comparan dos versiones de YOLOV5 (una con más capas convolucionales que la otra) junto con Fast R-CNN y RetinaNet. Los modelos se entrenaron en una base de datos llamada ABC-PollenOD. Ambos modelos de YOLOV5 se preentrenaron por 300 épocas en la base de datos COCO y se entrenaron como máximo por 500 épocas para la detección de gránulos polen. Las versiones de YOLOV5 superaron a los otros métodos, obteniendo 86.8 % y 92.4 % para los conjuntos de prueba.

1.2. Objetivos

1.2.1. Objetivos generales

El objetivo general de este trabajo de tesis es la detección de cuatro tipos de pólenes de especies de importancia apícola en el estado de Yucatán mediante la implementación de redes neuronales convolucionales.

1.2.2. Objetivos específicos

Los objetivos específicos son los siguientes:

- Preparar las imágenes de la base de datos para el entrenamiento de las redes neuronales.
- Implementar los algoritmos You Only Look Once(YOLO) versiones 4 y 7.
- Cambiar el Backbone de YOLO (tanto en la versión 4 como en la versión 7) por una versión modificada de la Res2Unet.
- Evaluar el desempeño de las redes modificadas con sus versiones originales.
- Comparar los resultados entre estos modelos.

Capítulo 2

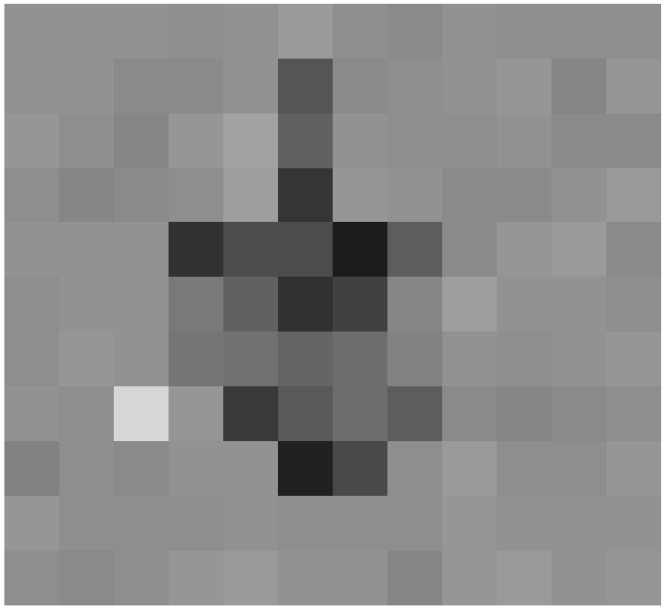
Marco Teórico

2.1. Procesamiento digital de imágenes

Una imagen digital puede representarse matemáticamente como una función bidimensional discreta y finita $f[x, y]$, donde los pares $[x, y]$ representan la posición (píxel) y f es la intensidad o nivel de gris del píxel en tal posición. [10].
En una computadora una imagen digital en escala de grises cada píxel se puede representar como un entero de 8 bits, con esto se pueden representar un rango amplío de tonalidades de gris (de 0 a 255). La figura 2.1 muestra un ejemplo de como se representa una imagen digital.
El procesamiento digital de imágenes es un conjunto de técnicas y métodos para manipular o transformar imágenes de tal manera que en esta se pueda mejorar la información presente, se puedan resaltar ciertas características u objetos de interés y procesarlas para su almacenamiento o transmisión. Estas imágenes se componen de un conjunto de elementos finitos llamados píxeles, que tienen una ubicación particular y un valor asociado. Estas técnicas y métodos son ampliamente utilizados en la electrónica, medicina, visión computacional, reconocimiento de patrones, entre otros.

2.2. Espacios de color

Existen varias formas de representar imágenes digitales a color en una computadora, estas diferentes formas se conocen como espacios de color [11]. Cada espacio de color tiene distintos parámetros para producir los colores.



(a) Imagen en escala de grises de 11x12 píxeles

145	145	144	143	145	153	140	138	144	141	143	143
144	145	139	139	146	85	136	140	144	148	133	149
150	141	132	148	162	98	144	143	143	145	138	139
140	135	139	143	156	55	149	147	138	139	147	152
145	144	146	48	76	78	28	92	138	148	155	138
142	147	147	123	96	51	64	133	156	144	147	143
142	149	144	116	112	100	110	128	146	140	144	148
144	140	215	149	57	88	108	94	136	133	139	142
130	142	138	145	147	32	75	143	153	142	141	151
150	142	143	140	145	140	142	143	148	147	147	147
143	139	141	148	152	145	146	133	150	153	145	150

(b) Representación matricial de la imagen (a)

Figura 2.1: Representación de una imagen digital en escala de grises.

2.2.1. Espacio de color RGB

Las pantallas electrónicas utilizan el espacio de color RGB, pues, estas están compuestas de pequeños elementos llamados píxeles, que a su vez se componen de 3 pequeños diodos emisores de luz (LED por sus siglas en inglés), cada uno de estos pequeños diodos puede producir un tipo específico de luz (Rojo, Verde y Azul) y mediante una combinación de intensidades de estos se pueden producir millones de colores [12]. Sin embargo, en ciertas ocasiones resulta más conveniente representar la misma imagen en un espacio de color distinto [13, 14, 15, 16]. Para fines de procesamiento de imágenes, resulta más costoso computacionalmente utilizar este espacio de colores, pues, hay que manipular 3 canales distintos (rojo, verde y azul) para afectar la imagen como se desea.

2.2.2. Espacio de color HSI

El espacio de color HSI (*Hue*, *Saturation*, *Intensity*) es una forma de representar una imagen digital a color que facilita la manipulación de imágenes mediante su tinte o matiz (*Hue*), la saturación (*Saturation*) y su intensidad (*Intensity*) [16]. Otra ventaja es que el espacio de color HSI se asemeja de una mejor manera a la forma en la que el ojo humano se comporta. Este espacio de color se define usando un sistema de coordenadas cilíndricas. Las componentes H, S e I se pueden calcular a partir de las coordenadas rectangulares R, G y B como se muestra en las ecuaciones 2.1,2.2 y 2.3.

$$I = \frac{R + B + G}{3} \tag{2.1}$$

$$S = \frac{2}{\sqrt{6}} \times \sqrt{(R^2 + G^2 + B^2 - RG - GB - BR)} \tag{2.2}$$

$$H = \arctan\left(\sqrt{3}\frac{G - B}{2R - G - B}\right) \tag{2.3}$$

2.3. Detección de objetos

La detección de objetos tiene sus orígenes en la visión computacional, cuyo objetivo es intentar encontrar objetos en una imagen, clasificarlos y encontrar el mejor conjunto de recuadros delimitadores que los contengan. Las técnicas clásicas para la detección de objetos realizan una extracción manual de características, por ejemplo, en [17] se usan técnicas de procesamiento de imágenes para obtener los contornos de ciertos

objetos, para así poder estudiarlos y de cierta manera caracterizarlos para poder discriminar entre el objeto de interés y lo demás. Otros utilizan características rectangulares precalculadas como ventanas deslizantes sobre una imagen en conjunto con métodos de clasificadores en cascada para detectar rostros humanos [18]. De igual manera en [19, 20] se utilizan características que se obtienen al transformar una imagen aplicando algoritmos de procesamiento de imágenes.

En los últimos años ha sido cada vez más popular el uso de redes neuronales artificiales y redes neuronales convolucionales para realizar esta tarea, especialmente estas últimas debido a que son capaces de aprender qué características son las más relevantes para los objetos de interés.
Existen básicamente dos tipos de arquitecturas de aprendizaje profundo para la detección de objetos, las de detección en múltiples etapas y las de una etapa. AlexNet, OverFeat y R-CNN fueron las primeras redes neuronales convolucionales usadas para la clasificación y detección de objetos [21, 22, 23]. Modelos como R-CNN realizan la detección en dos etapas, primero entrenan una red neuronal convolucional para extraer características importantes, luego estas características son ingresadas en un clasificador, como una máquina de soporte vectorial, y posteriormente se entrena un regresor, que utiliza ventanas deslizantes en toda la imagen, para encontrar los mejores rectángulos delimitadores para los objetos y finalmente se combinan los resultados de ambos para obtener la detección. Este tipo de detectores, aunque obtienen mejores resultados en exactitud que las técnicas tradicionales, entrenarlas toma demasiado tiempo, requieren de una gran cantidad de imágenes etiquetadas y el tiempo de detección normalmente está en orden de los segundos.

Por otra parte están los métodos de una sola etapa como [24, 7], los cuáles reestructuran la detección para utilizar una red neuronal convolucional al inicio como extractor de características y una red neuronal completamente conectada al final para obtener una predicción tanto de la clase como de los rectángulos delimitadores al mismo tiempo sin pasar por un clasificador extra, pues, la red completamente conectada utiliza la información directamente de las capas anteriores para hacer la predicción. Estos métodos tienen la ventaja de ser más rápidos al ser entrenados y al momento de hacer predicciones o inferencias sobre nuevas imágenes, el tiempo de detección de estos modelos normalmente está en el orden de milisegundos, haciéndolos buenos candidatos para tareas de detección de objetos en tiempo real.

2.4. Redes neuronales convolucionales

Los métodos previos (o clásicos) de reconocimiento de patrones para la detección y reconocimiento de objetos se entrenaban en CPUs (Unidades Centrales de Procesamiento) que se encontraban en las súper computadoras de la época, esto era posible debido a que los algoritmos no eran muy complejos, tenían pocos parámetros y se usaban imágenes de baja resolución. Sin embargo, conforme los modelos crecían y se hacían más complejos era más evidente que el *hardware* limitaba su alcance y precisión.

Las redes neuronales convolucionales (presentadas por primera vez por Yann LeCun [25] para el reconocimiento de caracteres escritos a mano) son un tipo especial de redes neuronales capaces procesar datos mediante el uso de una operación matemática llamada **convolución** [26].
Con la aparición de las redes neuronales convolucionales y los avances en hardware (principalmente las unidades de procesamiento gráfico) se aceleró el avance en las redes neuronales convolucionales para tareas de clasificación y detección de objetos en imágenes, permitiendo considerablemente el aumento del número de parámetros entrenables y la paralelización de estos algoritmos.

2.4.1. Convolución

Una convolución, en una dimensión, se puede escribir matemáticamente como:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{2.4}$$

La ecuación 2.4 asume que las funciones x y w son continuas, pero en realidad cuando realizamos operaciones en una computadora estos valores no son *continuos*, más bien son discretos. La versión discretizada de la ecuación 2.4 es:

$$s[t] = x[t] * w[t] = \sum_{a} x[a]w[t-a] \tag{2.5}$$

Dado que en la detección de objetos se trabaja con imágenes, la entrada de una red convolucional es normalmente un vector bidimensional y aplicamos la convolución en los ejes x e y de la imagen. La versión en dos dimensiones de la ecuación 2.5 queda de la siguiente forma:

$$S[i,j] = I[i,j] * K[i,j] = \sum_{m}\sum_{n} I[m,n]K[i-m, j-n] \tag{2.6}$$

Donde I es el valor de la imagen en el píxel (m, n) y K es el kernel utilizado para realizar la convolución en la imagen.

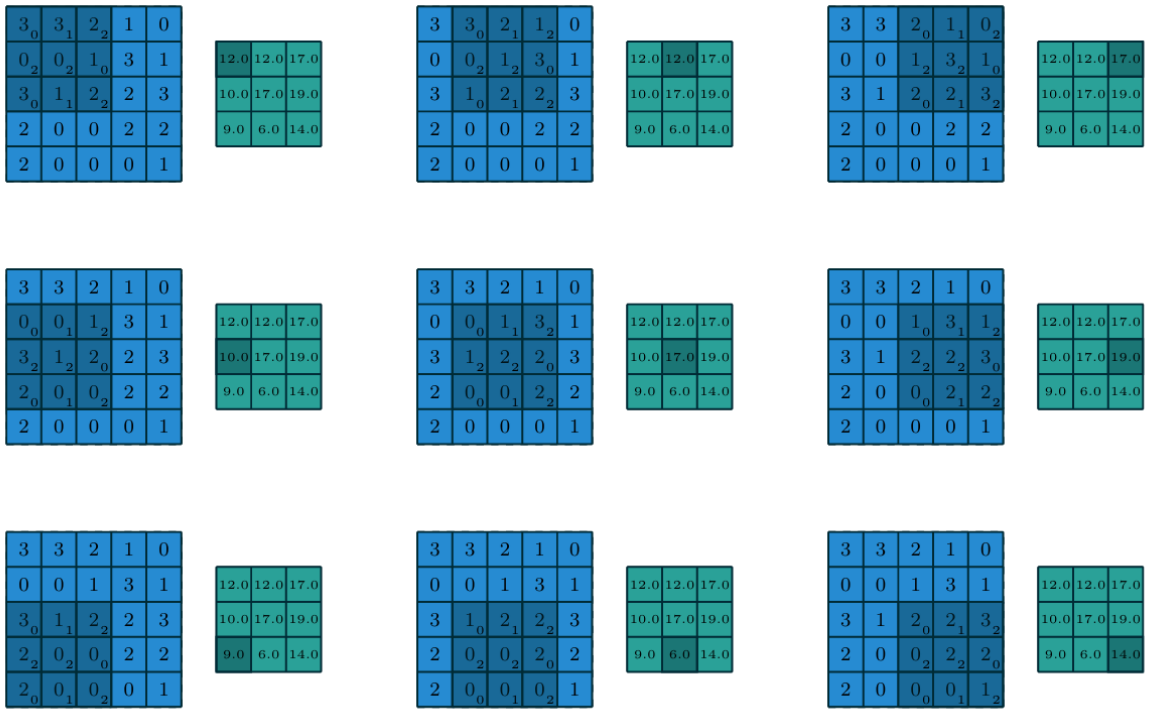


Figura 2.2: Ejemplo de una convolución en una matriz de 5×5 con un kernel de 3×3.

En la figura 2.2 se muestra un ejemplo de cómo se produce un mapa de características con una convolución de una matriz de entrada de 5×5 y un kernel de 3×3.

2.4.2. Kernel

El kernel es una matriz especial utilizada en una convolución a manera de ventana deslizante. Una red neuronal convolucional está compuesta de varias capas que son el resultado de realizar una convolución con estas matrices (kernels) y con capas previas, a las capas que se obtienen como resultado de de una convolución también se les llama mapas de características.
Existen distintos tipos de kernels, en la figura 2.3 se muestra un ejemplo de un kernel de tamaño 3×3 que puede ser utilizado para *suavizar* una imagen mediante una convolución. En el ejemplo anterior el kernel tiene valores fijos para resaltar características determinadas en una imagen (o mapa de características), sin embargo, en una red neuronal convolucional los valores individuales de cada kernel son parámetros que se pueden modificar por algún algoritmo de optimización.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Figura 2.3: Kernel de 3×3 usado para aplicar un filtro de suavizado de imagen.

2.4.3. Stride

En la figura 2.2 el kernel se desplaza un sólo píxel a la vez haciendo un producto punto entre la entrada y el kernel, sin embargo, esto no tiene porque ser así. Se puede especificar cuántos píxeles se desplazará el kernel en el mapa de características de entrada. Se puede especificar, por ejemplo, un stride de $s = 2$ para reducir la *resolución* del mapa de características resultante a la mitad.

2.4.4. Max pooling

Existen distintos métodos de pooling pero el más común es el *max pooling*. El objetivo principal del max pooling es reducir el tamaño de un mapa de características, y por tanto, reducir también el número de parámetros que tiene que modificar la red. El *max pooling* funciona como una ventana deslizante de tamaño N×N (normalmente de 2×2) sobre un mapa de características, y tomando el mayor de los N×N valores. La figura 2.4 muestra un ejemplo de una operación de max pooling de tamaño 3×3 en un mapa de características de tamaño 5×5.

2.4.5. Zero padding

Debido al proceso de convolución y que en el ejemplo 2.2 el kernel tiene un tamaño de 3×3 el tamaño resultante del mapa de características cambia de la siguiente manera, para el ancho $w_n = (w - (k - 1))$ y para el alto $h_n = (h - (k - 1))$, es decir, el mapa de características resultante tendrá un tamaño de (w_n, h_n). Para evitar que cambie de tamaño el mapa de características resultante se agregan ceros alrededor de cada mapa de características al que se va a aplicar la convolución, a este proceso se le conoce como *zero padding*. La figura 2.5 muestra como se aplica el *zero padding* en una matriz de 5×5 y en la figura 2.6 se muestra una convolución en una matriz de 5×5 con *zero padding*, un kernel de 3×3 y stride de $s = 2$.

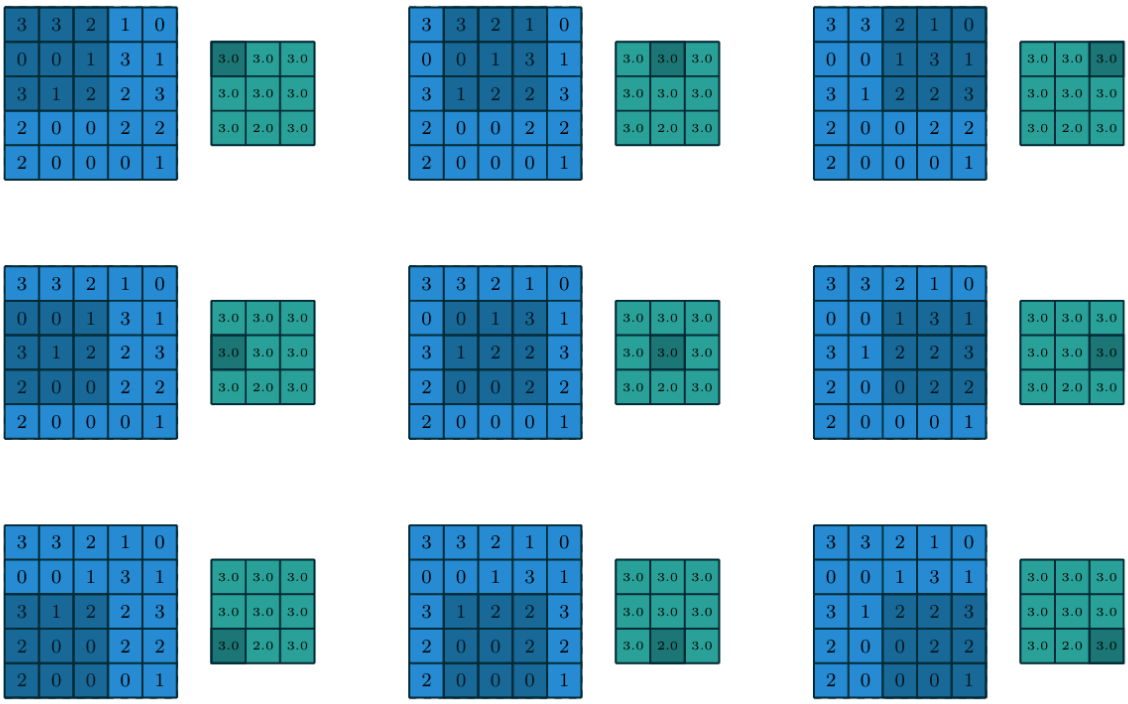


Figura 2.4: Ejemplo de una operación de max pooling.

0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	145	145	144	143	145	153	140	138	144	141	143	143	0
0	144	145	139	139	146	85	136	140	144	148	133	149	0
0	150	141	132	148	162	98	144	143	143	145	138	139	0
0	140	135	139	143	156	55	149	147	138	139	147	152	0
0	145	144	146	48	76	78	28	92	138	148	155	138	0
0	142	147	147	123	96	51	64	133	156	144	147	143	0
0	142	149	144	116	112	100	110	128	146	140	144	148	0
0	144	140	215	149	57	88	108	94	136	133	139	142	0
0	130	142	138	145	147	32	75	143	153	142	141	151	0
0	150	142	143	140	145	140	142	143	148	147	147	147	0
0	143	139	141	148	152	145	146	133	150	153	145	150	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 2.5: Representación del *zero padding*

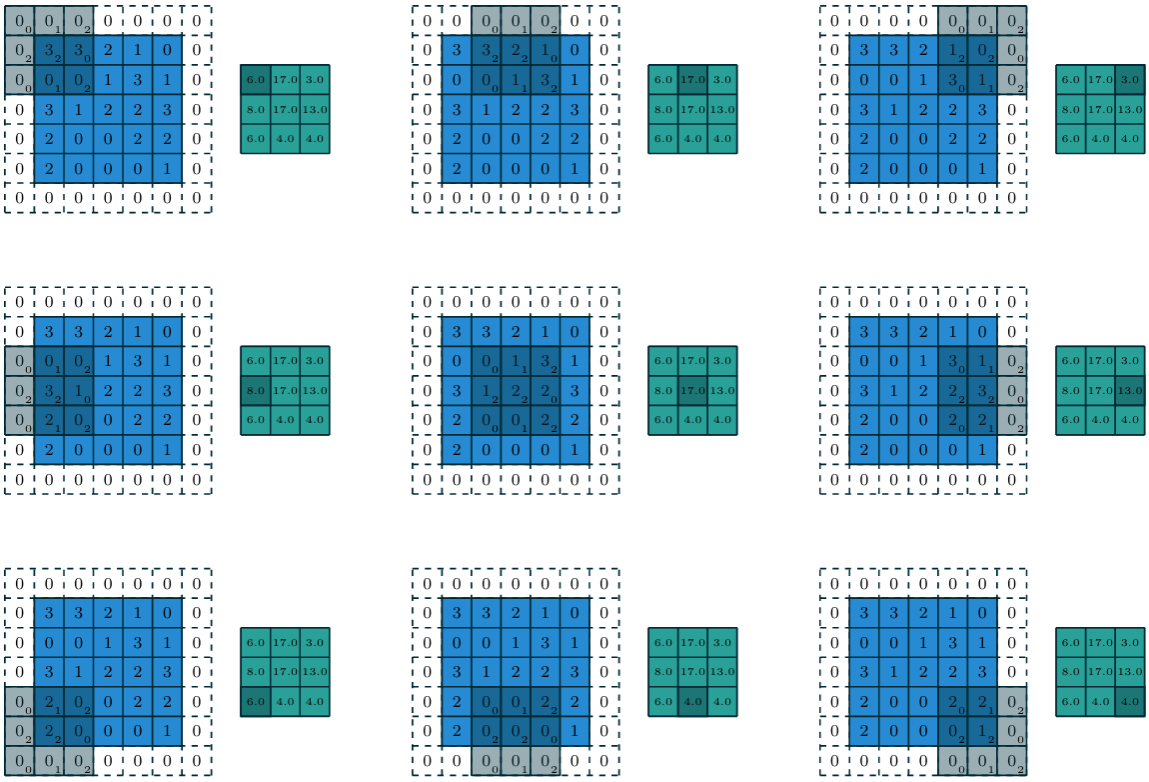


Figura 2.6: Ejemplo de una convolución con *zero padding*.

2.4.6. Funciones de activación

Las tareas que se intentan resolver mediante el uso de redes neuronales convolucionales normalmente no son problemas *lineales*, de ser así se utilizarían otros métodos que son buenos para encontrar soluciones a problemas más simples como, por ejemplo, un problema de regresión logística.
Para evitar que la arquitectura se mantega lineal y permitirle agregar complejidad, se utilizan un conjunto de funciones no lineales que ayudan al aprendizaje de las capas convolucionales y de toda al red [27, 28, 29, 30, 31]. En la ecuación 2.7 se describe de manera general este proceso.

$$g(z) = g(W[i,j] * I[i,j] + b) \tag{2.7}$$

2.4.6.1. ReLU

De entre las funciones de activación más populares en las capas de redes neuronales convolucionales está la Rectified Linear Unit (ReLU) [32]. Las funciones ReLU han sido

ampliamente usadas en el campo del aprendizaje profundo debido a que aceleran el aprendizaje de las redes neuronales convolucionales, suelen tener mejor rendimiento y capacidad de generalización a comparación de las funciones de activación sigmoidal y tangente hiperbólica. ReLU tiene ciertas limitaciones, por ejemplo, es más fácil que ocurra un *overfitting* [33] a comparación de la función sigmoidal y al transformar las partes negativas a cero de los pesos, contribuye al desvanecimiento del gradiente, y por tanto, imposibilitando el aprendizaje de la red.
Existen múltiples variantes de esta función que intentan remediar algunos de sus problemas, por ejemplo, leaky ReLU, ELU, PReLU, RReLU, etc [27]. En la ecuación 2.8 se muestra como se define matemáticamente la función de activación ReLU.

$$g(z) = max(0, z) \tag{2.8}$$

La ecuación 2.8 implica que todas las entradas positivas se reasignan en cero y las entradas positivas quedan sin alterarse. La figura 2.7 muestra la gráfica de la función de activación ReLU.

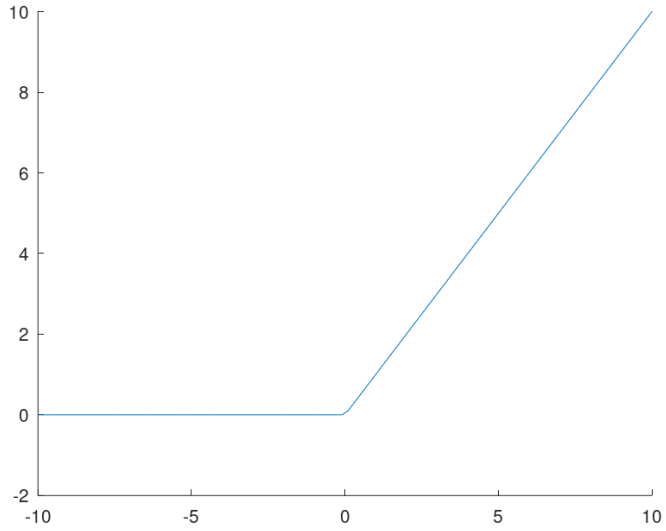


Figura 2.7: Gráfica de la función ReLU.

2.4.6.2. Mish

Mish es una función de activación relativamente nueva que tiene las propiedades de ser auto-regularizable y mejorar la precisión en tareas de clasificación y detección de objetos, además de minimizar los defectos de otras funciones de pérdida [34]. En la

ecuación 2.9 se define matemáticamente la función de activación Mish.

$$g(z) = ztanh(softplus(z)) \tag{2.9}$$

donde $softplus(z) = ln(1 + e^x)$. En la figura 2.8 se muestra la gráfica de la función de activación Mish.

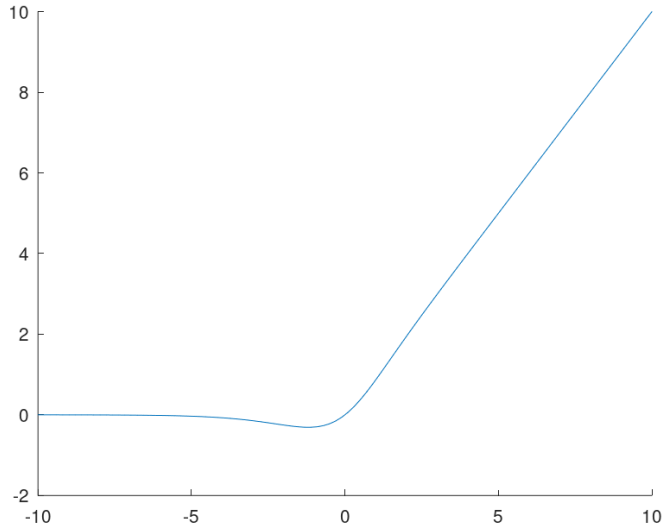


Figura 2.8: Gráfica de la función Mish.

2.4.7. Redes neuronales residuales

Las redes neuronales convolucionales tienen ciertas limitaciones conforme se van haciendo cada vez más profundas. Conforme las arquitecturas se vuelven más profundas (tienen más capas convolucionales) sufren de un fenómeno conocido por degradar los mapas de características, lo que provoca que tengan un error más alto [35].
Una red residual se forma conectando las capas anteriores a una cierta cantidad de convoluciones directamente con el resultado de capas posteriores, sumando elemento a elemento estas capas [36, 37, 38, 39]. Matemáticamente se puede expresar esto como

$$y = F(x) + x \tag{2.10}$$

donde $F(x)$ es el resultado de n convoluciones previas y x es la entrada previo a las n convoluciones. La figura 2.9 muestra la estructura de un bloque residual. Este método, al tomar características desde capas anteriores a las convoluciones, ayuda a minimizar

el efecto de desvanecimiento y explosión del gradiente [40, 41, 42, 43] y suelen tener mejor precisión con menos capas que su equivalente sin conexiones residuales.

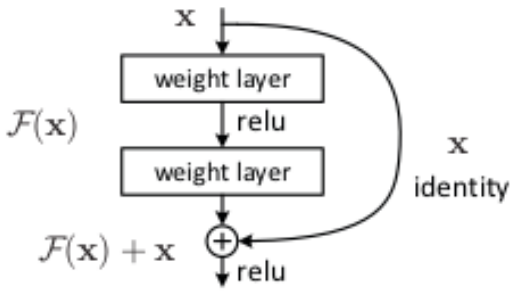


Figura 2.9: Representación de un bloque residual.

2.4.8. Función de pérdida

En el aprendizaje profundo una función de pérdida actúa como una medida de que tan lejos se está del objetivo (en este caso clasificación o detección de objetos), a esta medida se le llama *error* porque compara la salida de la red con el *ground truth*, es decir, qué tanto se acerca la red a los valores esperados (en este caso las imágenes marcadas por un humano) [44]. Existen varias funciones que se pueden utilizar como funciones de pérdida [45] y se debe elegir la adecuada para el problema que se esté intentando resolver. El resultado de esta función se utiliza en un algoritmo de optimización para poder encontrar valores de las entradas que hagan mínimo el error.

2.4.9. Descenso de gradiente

Para que una red neuronal pueda aprender, es necesario que se encuentren los pesos de cada capa que hacen mínimo el error, dado por la función de pérdida. El algoritmo más común para optimización en el aprendizaje profundo es el descenso de gradiente [46].
Para determinar el valor de los siguientes pesos se utiliza el gradiente del hiperplano que se quiere minimizar. El gradiente es la derivada parcial de la función de pérdida con respecto a todos los pesos a actualizar, como se muestra en la ecuación 2.11.

$$\frac{\partial J(\theta_0, \theta_1, ..., \theta_n)}{\partial \theta_i} \tag{2.11}$$

Utilizando las derivadas parciales con la función de pérdida, los nuevos pesos se actualizan como se muestra en la ecuación 2.12.

$$\theta_{\text{i nuevo}} = \theta_{\text{i anterior}} - \alpha \frac{\partial J(\theta_0, \theta_1, ..., \theta_n)}{\partial \theta_i} \tag{2.12}$$

Donde α el factor que indica que tanto peso se le dará al gradiente para actualizar los pesos.

2.4.10. Entrenamiento

Para poder entrar un modelo es necesario tener en cuenta varios elementos:

- Subconjuntos de la base de datos.
- Tamaño del lote (*batch size*).
- Iteraciones.

Lo ideal sería que una vez que nuestro modelo se entrene en una base de datos, se obtengan nuevas imágenes que no estén presentes en la base de datos donde se entrenó el modelo, sin embargo, no siempre es posible obtener más imágenes que con las que se cuenta. Debido a esta limitación se suele subdividir la base de datos existente en tres subconjuntos: uno de entrenamiento, uno de validación y uno de pruebas. Para una correcta comprobación del modelo se crean tres subconjuntos a partir de la base de datos, un subconjunto de entrenamiento, uno de validación y uno de pruebas. El subconjunto de entrenamiento se utiliza para actualizar los pesos del modelo utilizando el descenso de gradiente. El subconjunto de validación se utiliza para variar los hiperparámetros del modelo. Y por último el subconjunto de pruebas sirve para evaluar el modelo.
Al momento de entrenar una red neuronal en una computadora con GPU, se requiere una cierta cantidad de memoria para cargar los pesos y las imágenes que serán procesadas en la memoria RAM de la GPU. Debido a que la cantidad de memoria RAM presente en una GPU es limitada, y para que el proceso del descenso de gradiente pueda actualizar sus pesos y pueda converger rápidamente, en lugar de esperar a que se procesen todas las imágenes en el conjunto de entrenamiento para poder actualizar los pesos, las imágenes se procesan por lotes y por cada lote los pesos se actualizan. En una iteración se procesa un lote de imágenes y se actualizan los pesos.

2.4.11. Aumentación de datos

En ocasiones no es posible tener tantas imágenes como se quiere, sin embargo, los algoritmos para tareas como clasificación o detección de objetos requieren de bastantes muestras para poder generalizar y abstraer características importantes de los objetos. Debido a esto existen técnicas de procesamiento digital de imágenes que pueden ayudar a incrementar de manera artificial el número de imágenes con las que se cuenta.
Estas transformaciones alteran la imagen sin modificar la información relevante que se encuentra presente.
Algunos ejemplos de estas transformaciones son:

- Traslación
- Rotación
- Saturación
- Brillo
- Contraste
- CutMix [47]
- Blur

2.4.12. Red neuronal convolucional Res2Unet

La Res2Unet, propuesta en [48], es una versión inspirada en la arquitectura UNet [49]. La Res2Unet es una red neuronal completamente convolucional (ver Figura 2.11) diseñada para la segmentación semántica de objetos, específicamente para segmentar imágenes del parásito *Trypanosoma cruzi*. Esta versión modificada de la UNet modifica la función de pérdida original e introduce un método basado en la aproximación de ecuaciones diferenciales para construir los bloques residuales.
En la ecuación 2.13 se describe matemáticamente el método de Heun, utilizado para aproximar soluciones de ecuaciones diferenciales. Esta misma ecuación se implementa utilizando una suma entre una serie de capas convolucionales y la capa de entrada a ese conjunto de convoluciones (Figura 2.10).

$$y = x + hF(x) \tag{2.13}$$

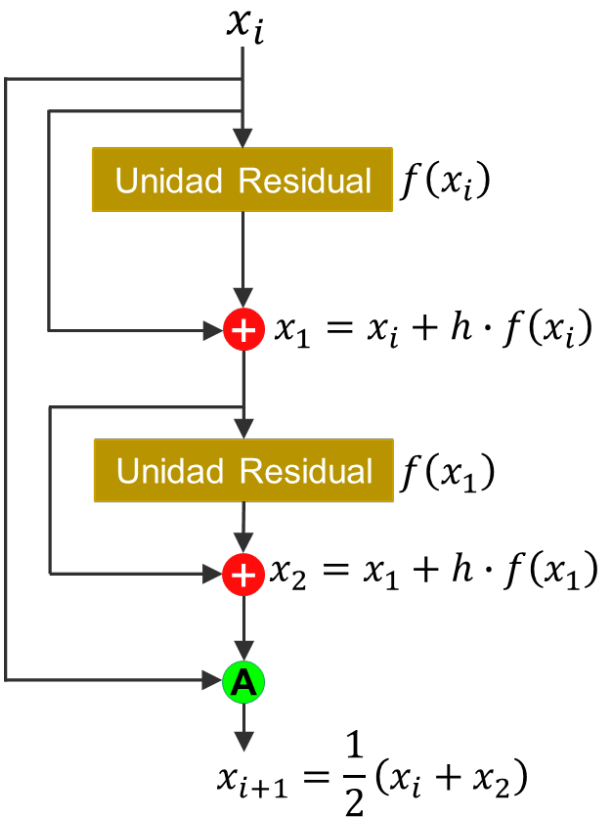


Figura 2.10: Representación del método de Heun en la Res2UNet.

La Res2UNet se compone de tres partes principales: codificador, puente y decodificador. El codificador es un módulo diseñado para la extracción de características de la imagen de entrada. Sin embargo, conforme se efectúan las convoluciones en el codificador, el tamaño de los mapas de características disminuye. Dado que se requiere que la salida de la red sea la imagen original segmentada, es necesario regresar la imagen al tamaño original, esto se logra mediante los módulos puente y decodificador, los cuáles utilizan una serie de convoluciones transpuestas para esto.

Las Unidades Residuales están estructuradas de la siguiente manera. La Unidad Residual 1 se compone de una capa de convolución, seguido de una capa de *batch normalization*, una capa de activación ReLU y una capa de convolución. La Unidad Residual 2 consiste en una capa de *batch normalization*, seguida de una capa de activación ReLU, una capa de convolución, una capa de *batch normalization*, una capa de activación ReLU y una capa de convolución (ver Figura 2.12).

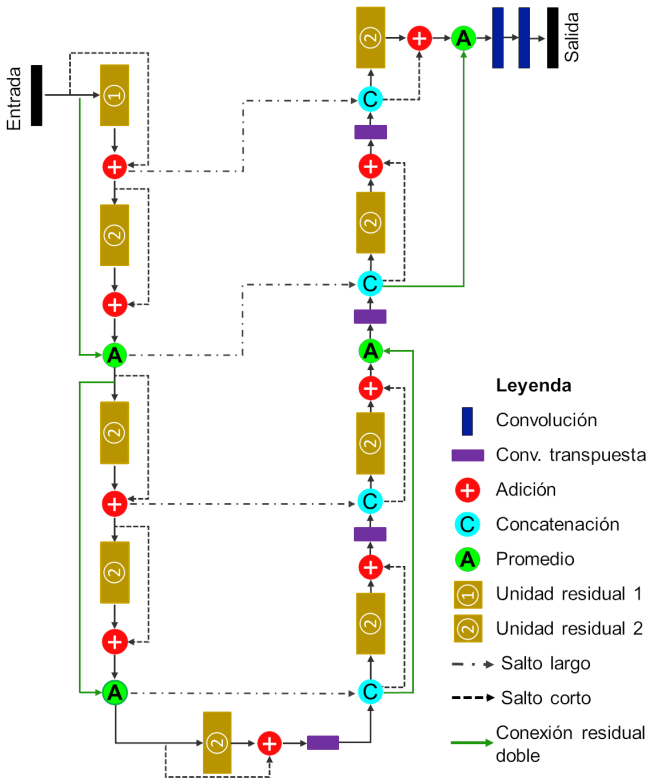


Figura 2.11: Diagrama de la arquitectura de la Res2UNet.

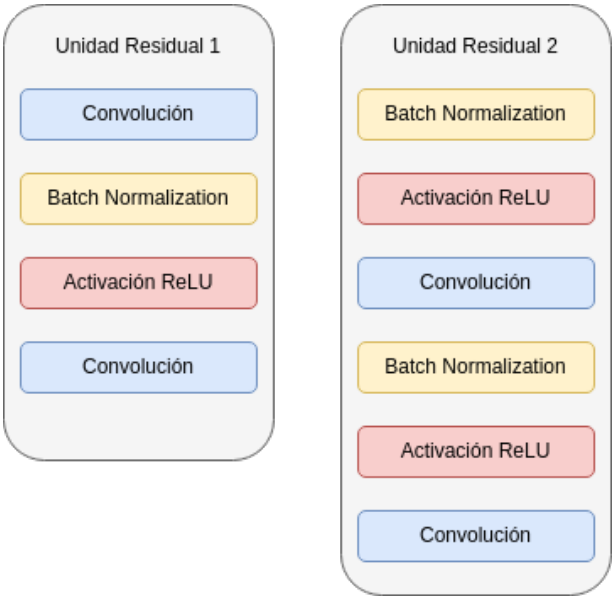


Figura 2.12: Estructura de las unidades residuales en la Res2UNet.

2.4.13. Red neuronal convolucional YOLO

YOLO (*You Only Look Once* por sus siglas en inglés) es una red neuronal convolucional para la detección de objetos en tiempo real de una sola etapa, presentada por primera vez en [7]. En su primera versión, realiza la detección subdividiendo la imagen de entrada en un arreglo de celdas de tamaño $\mathbf{S}\times\mathbf{S}$ y por cada una de estas celdas predice dos recuadros delimitadores, la probabilidad de que en el recuadro esté el objeto de interés y la probabilidad de la clase. La función de pérdida utilizada es la de la ecuación 2.14, junto con una red completamente conectada al final de las capas convolucionales para realizar las predicciones tanto de la clase como del recuadro delimitador. La parte convolucional de la red usada para extraer características está basada en la arquitectura de Googlenet. En la figura 2.13 se muestra como está estructurada la arquitectura de la primera versión de YOLO.

En su primera versión alcanza un mAP bastante cercano a métodos más lentos y de dos etapas, como en [23] y sus derivadas, pero con mayor eficiencia. Al ser de una sola etapa y aplicar la detección en toda la imagen, es decir, tiene todo el *contexto* y no utilizar una ventana deslizante para proponer regiones de un tamaño fijo como métodos previos, esta red predice menos falsos positivos sobre el fondo de la imagen.

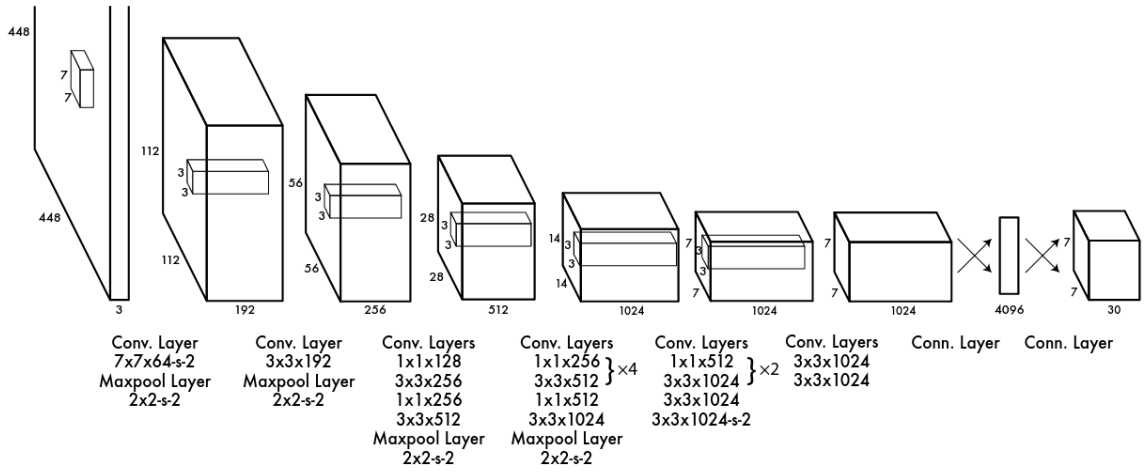


Figura 2.13: Arquitectura de YOLO versión 1.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(x_i - \tilde{x}_i)^2 + (y_i - \tilde{y}_i)^2 \right]$$
$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\tilde{w}_i})^2 + \left(\sqrt{h_i} - \sqrt{\tilde{h}_i} \right)^2 \right]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \tilde{C}_i)^2$$
$$+ \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \tilde{C}_i)^2$$
$$+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in clases} (p_i(c) - \tilde{p}_i(c))^2 \quad (2.14)$$

Existen varias familias de arquitecturas modificadas de YOLO que mejoran progresiva y significativamente el rendimiento y precisión de esta [50, 51, 47, 52, 53, 54] o incluso las reutilizan para otras tareas como la segmentación [55] y la conducción autónoma de vehículos [56], pero las que son relevantes para este trabajo son las correspondientes a YoloV4 y YoloV7 [47, 52, 54], debido a que son las relativamente más recientes al momento de realizar el presente trabajo de tesis.

2.5. Métricas de desempeño

Para poder comparar diferentes modelos de manera objetiva es necesario un método que pueda medir de manera cuantitativa el desempeño de cada uno de los modelos a comparar. A estos métodos se les llama métricas de desempeño. Existen una gran variedad de métricas de desempeño, cada una diseñada adecuadamente para evaluar modelos según la tarea que tienen que realizar [57, 58, 59]. Por ejemplo, para la segmentación de imágenes médicas la métrica más comúnmente utilizada es la métrica del coeficiente de *DICE* (también llamada F_1-Score) [58] y para detección de objetos normalmente se utilizan el mAP (*mean Average Precision*) y el IoU (*Intersection over the Union*) [60, 59].

Varias de estas métricas se definen utilizando los verdaderos positivos (*True Positivies*) y falsos positivos (*False Positives*) y falsos negativos (*False Negatives*). Para poder determinar si una detección se hizo de manera correcta o incorrecta se utiliza como referencia el *IoU* y un umbral para este. Por ejemplo, si se utiliza un umbral del 0.5 en

el *IoU*, se considerará una detección como correcta si se cumple que $IoU_{objeto} \geq 0{,}5$. El valor de verdaderos positivos representa el número de instancias detectadas correctamente por el modelo (usando como referencia el *ground truth*), el valor de falsos positivos representa el número de instancias detectadas erróneamente por el modelo que no están presentes en el *ground truth* (y que podrían no ser realmente los objetos de interés) y por último el valor de falsos negativos representa el número de instancias que no se detectaron, o se ignoraron pero que si estaban presentes en el *ground truth*, por el modelo.

2.5.1. Precisión

Esta métrica indica qué tan bueno es el modelo para realizar detecciones correctas, es decir, es el porcentaje de detecciones correctas sobre todas las detecciones realizadas.

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{Todas las detecciones}} \tag{2.15}$$

2.5.2. Recall

Esta métrica indica qué tan bueno es el modelo para realizar detecciones de objetos verdaderos, es decir, es el porcentaje de detecciones correctas sobre todos los *ground truth*.

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{Todos los } ground\ truth} \tag{2.16}$$

2.5.3. F1-Score

Esta métrica sirve para evaluar el modelo tomando en cuenta el *precision* y el *recall* como igual de imporatantes.

$$F_1 Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{2.17}$$

2.5.4. mAP

Esta métrica es la media aritmética de todos los *Average Precision*, se calcula utilizando el área bajo la curva PR (*Precision Recall*).

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \tag{2.18}$$

$$AP_i = \sum_n (R_{n+1} - R_n) max_{R:R \geq R_{n+1}} P(R) \tag{2.19}$$

Donde AP_i es el *Average precision* para la *i-ésima* clase y N es el número total de clases evaluadas.

2.5.5. IoU

Esta métrica sirve como base para poder determinar si un objeto se predijo correctamente o no.

$$J(B_p, B_{gt}) = IoU = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})} \tag{2.20}$$

Capítulo 3

Metodología

3.1. Base de datos

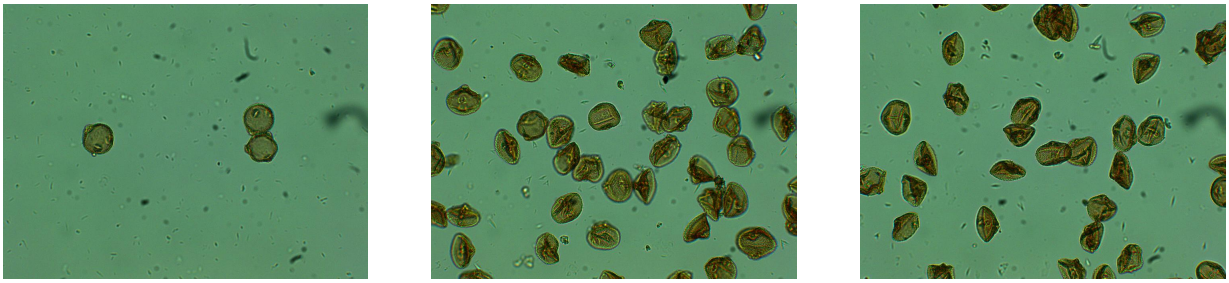
Las imágenes fueron proporcionadas de la base de datos del laboratorio de Ciencia de Alimentos del Campo Experimental Mocochá, del Centro de Investigación regional Sureste, INIFAP. Está compuesta de un total de 370 imágenes de gránulos de polen tomadas con un microscopio óptico (marca Motic provisto de cámara y software para captura de imágenes (Moticam 10)) en los objetivos de 10X y 40X; todas las imágenes fueron capturadas en formato RGB y con una resolución de 3664×2748. Las base de datos se divide en cuatro tipos de gránulos de polen de especies de importancia apícola en el estado de Yucatán: box catzín (*Acacia gaumeri*), chaká (*Bursera simaruba*), jabín (*Piscidia piscipula*) y tajonal (*Viguiera dentata*). El cuadro 3.1 muestra el número de objetos presentes por clase en la base de datos.

Clase	**Instancias por clase**	**Imágenes por clase**
Box catzín	265	96
Chaká	334	126
Jabín	601	52
Tajonal	426	96

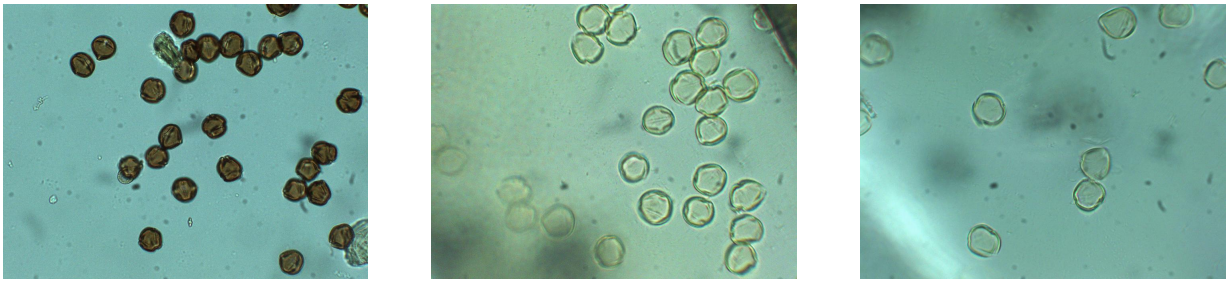
Cuadro 3.1: Número de objetos y número de imágenes por clase.

Identificar la especie de gránulos de polen a partir de una imagen de microscopio óptico y posteriormente contarlos es una tarea bastante difícil, especialmente cuando algunas especies se ven bastante diferentes a distintas escalas. En la figura 3.1 se muestran las imágenes de los gránulos de polen de las cuatro especies tomadas en objetivos 10X y 40X.

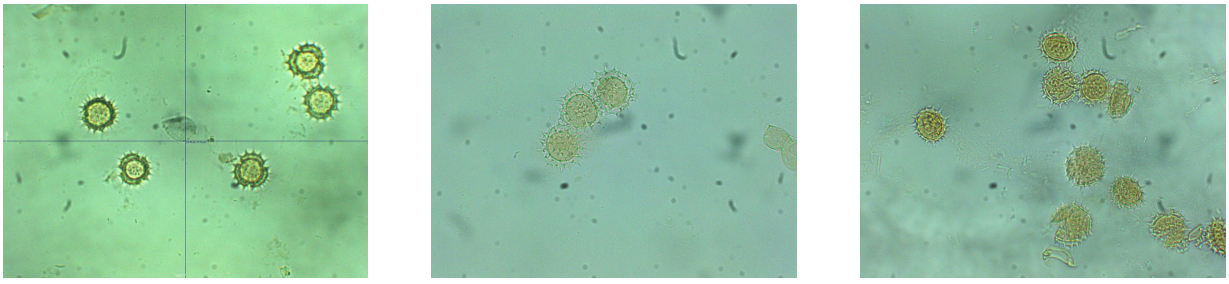
(a) Box catzín



(b) Chaká



(c) Jabín



(d) Tajonal

Figura 3.1: Algunas imágenes de la base de datos de gránulos de polen de las especies *Box catzín*, *Chaká*, *Jabín* y *Tajonal*.

Esta base de datos se dividió en tres subconjuntos: entrenamiento, validación y pruebas. Se escogieron las siguientes proporciones para la división: 80 % para entrenamiento, 10 % para validación y 10 % para pruebas; se escogieron dichas proporciones con el fin de intentar utilizar la mayor cantidad posible de imágenes para entrenamiento, debido a que la cantidad total de imágenes es limitada (370 imágenes en total).
Además, como se puede observar en el cuadro 3.1, también existe un desequilibrio en

el número de imágenes por clase. Para evitar que en los tres distintos subconjuntos se omita alguna clase (por ejemplo que en el conjunto de entrenamiento no aparezcan imágenes de Jabín) se realizó una división por clase en lugar de tomar las imágenes de toda la base de datos. El cuadro 3.2 muestra como se subdividió de la base de datos en los tres subconjuntos.

Clase	**Entrenamiento**	**Validación**	**Pruebas**
Box catzín	76	9	11
Chaká	100	12	14
Jabín	41	5	6
Tajonal	76	9	11

Cuadro 3.2: Distribución del número de imágenes por subconjunto.

El procedimiento para generar los tres subconjuntos fue siguiente:

1. Para cada clase se obtuvieron las rutas de todas las imágenes de una clase, se revolvieron y se guardaron en un archivo de texto, por ejemplo, para la clase *box catzín* se generó un archivo llamado *Boxcatzin.txt* que contiene la ruta completa de todas las imágenes presentes en la base de datos de esa clase.

2. Se calculó para entrenamiento y validación (el resto se usa para pruebas), según un porcentaje dado, cuantas líneas de los archivos *clase.txt* (donde clase representa el nombre de la clase a usar) se usarán. Debido a que en, por ejemplo, el archivo *Boxcatzin.txt* cada línea contiene la ruta a una imagen de esta clase.

3. Se producen los archivos *train.txt*, *valid.txt* y *test.txt*.

3.2. Modelo propuesto

La arquitectura propuesta en este trabajo está inspirada en la arquitectura de la Res2Unet y YOLOV4. Se reemplazó la CSPDarknet-53 (*backbone*), presente en la versión original de YOLOV4, por nuestro modelo WRes2Net y los demás módulos (*neck* y *head*) quedaron intactos.
Originalmente la Res2Unet se compone de tres partes principales: el codificador, el puente y el decodificador. En esta versión modificada de la Res2Unet se eliminó la parte del decodificador (que originalmente era para la segmentación), se agregaron 2 capas convolucionales extra al inicio de la red antes de alimentar al codificador,

se eliminaron los valores fijos de h del método Heun (Método de Euler mejorado o modificado [48]) y se introdujo un tipo de conexión residual (*shortcut*) con pesos entrenables para cada capa que se suma, permitiéndo a la red encontrar los valores óptimos del antes hiperparámetro h. Además, todas las funciones de activación ReLU se reemplazaron por funciones de activación Mish, debido a las ventajas que esta presenta sobre ReLU.

3.2.1. Arquitectura

Originalmente los bloques residuales propuestos en la Res2Unet se componen de una capa convolucional, una capa separada de *batch normalization*, una capa separada de activación ReLU y al final una capa convolucional. La Res2Unet originalmente utiliza una imagen de entrada de 256×256 para la segmentación, sin embargo, en esta versión para la detección de objetos se redimensionaron las imágenes a una resolución de 416×416 para obtener mayor información sobre el objeto y poder detectar objetos de distintos tamaños. Además se utilizó una versión de una capa convolucional que combina *batch normalization* y la activación Mish en dentro de la misma capa. Se agregaron dos capas convolucionales de treinta y dos filtros y kernel de 3×3 antes del codificador, todas las capas tienen *batch normalization* y activación Mish [39]. El cuadro 3.3 muestra la estructura del *backbone* propuesto.

La capa de suma (*shortcut*) se modificó para tener pesos por cada capa que se suma y además se normalizan estos pesos con una función ReLU y al final una función de activación Mish (Figura 3.2). En la ecuación 3.1 se describe este proceso de normalización en la capa de suma.

$$W_i = \frac{ReLU(w_i)}{\sigma} \tag{3.1}$$

Donde N es el número de capas que se suman y $\sigma = \sum_{i=1}^{N} ReLU(w_i)$. De manera que la modificación de la ecuación 2.13 queda como se muestra en la ecuación 3.2.

$$y = Mish\left(W_1 x + W_2 F(x)\right) \tag{3.2}$$

3.2.2. Detalles de implementación

Se entrenó la red en el conjunto de entrenamiento para aprender los pesos adecuados para la detección y clasificación, el conjunto de validación sirvió para ajustar los

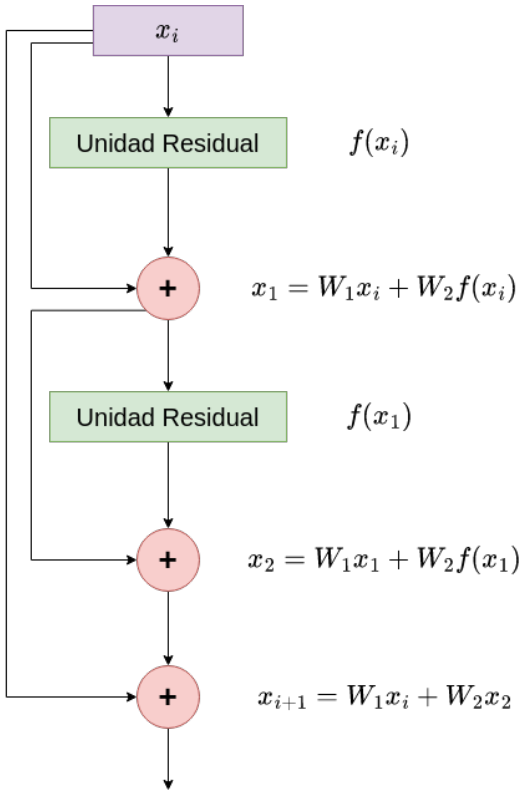


Figura 3.2: Representación del método utilizado en las capas de suma (shortcut) en la WRes2Net.

Capa	Filtros	Kernel / Stride	Pad	Entradas	Salida
Conv1	32	3×3 / 1	1	Imagen de entrada (416×416×3)	416×416×32
Conv2	32	3×3 / 2	1	Conv1 (416×416×32)	208×208×32
Conv3	32	3×3 / 1	1	Conv2 (208×208×32)	208×208×32
Conv4	64	3×3 / 1	1	Conv3 (208×208×32)	208×208×64
Conv5	64	1×1 / 1	0	Conv2 (208×208×32)	208×208×64
shortcut1				Conv5-4 (208×208×64)	208×208×64
Conv6	64	3×3 / 2	1	shortcut1 (208×208×64)	104×104×64
Conv7	128	3×3 / 1	1	Conv6 (104×104×64)	104×104×128
Conv8	128	1×1 / 2	0	shortcut1 (208×208×64)	104×104×128
shortcut2				Conv8-7 (104×104×128)	104×104×128
Conv9	128	1×1 / 2	0	Conv2 (208×208×32)	104×104×128
shortcut3				Conv9-shortcut2 (104×104×128)	104×104×128
Conv10	128	3×3 / 2	1	shortcut3 (104×104×128)	52×52×128
Conv11	256	3×3 / 1	1	Conv10 (52×52×128)	52×52×256
Conv12	256	1×1 / 2	0	shortcut3 (104×104×128)	52×52×256
shortcut4				Conv12-11 (52×52×256)	52×52×256
Conv13	256	3×3 / 2	1	shortcut4 (52×52×256)	26×26×256
Conv14	512	3×3 / 1	1	Conv13 (26×26×256)	26×26×512
Conv15	512	1×1 / 2	0	shortcut4 (52×52×256)	26×26×512
shortcut5				Conv15-14 (26×26×512)	26×26×512
Conv16	512	1×1 / 4	0	shortcut3 (104×104×128)	26×26×512
shortcut6				Conv16-shortcut5 (26×26×512)	26×26×512
Conv17	512	3×3 / 2	1	shortcut6 (26×26×512)	13×13×512
Conv18	1024	3×3 / 1	1	Conv17 (13×13×512)	13×13×1024
Conv19	1024	1×1 / 2	0	shortcut6 (13×13×512)	13×13×1024
shortcut7				Conv19-18 (13×13×1024)	13×13×1024

Cuadro 3.3: Estructura del *backbone* del modelo propuesto.

hiperparámetros (como el número de capas, número de filtros por capa, función de activación, etc). Finalmente se usó el conjunto de pruebas para obtener el desempeño final de la red, mediante el cálculo de las métricas de desempeño. Se comparó el desempeño del modelo propuesto con la versión original de YoloV4 y YoloV7.
Durante el entrenamiento se generaron imágenes adicionales utilizando aumentación de datos de manera aleatoria aplicando una variación de la saturación, hue y exposición (intensidad luminosa) de las imágenes del conjunto de entrenamiento, además las imágenes fueron redimensionadas a un cuarto de su tamaño original para que ocuparan menos memoria al momento de ser cargadas para el entrenamiento. Todos los modelos se entrenaron por un total de 10000 iteraciones, utilizando descenso de gradiente estocástico (SGD por sus siglas en inglés) con reinicios [61], con un *learning rate* inicial de 0.0026, un *momentum* de 0.949 y un *decay* de 0.0005.

Para la implementación de la Res2Unet modificada, se utilizó el *framework* de Darknet[1] [62]. Todos los experimentos se ejecutaron en una máquina con el sistema operativo PopOS 22.04 (con la versión del kernel de linux 5.15.23-76051523-generic), con la versión del controlador de Nvidia 515.65.01, la versión de CUDA[2] 11.2, la versión de CUDNN[3] 8.0.4, un procesador AMD Ryzen 7 3700X, una GPU Nvidia RTX 2070 SUPER con 8GB de VRAM y 32 GB de RAM.

3.3. Evaluación experimental

Se realizaron varios experimentos con distintos valores de h para determinar si esta tenía algún efecto en la detección de objetos, se probó con el valor $h = 0{,}1$, $h = 0{,}5$, $h = 2$ y sin el valor h. Entrenar el modelo con la WRes2Net como *backbone* en YoloV4 toma aproximadamente 14 horas. Los pesos entrenados del modelo suman aproximadamente 190 megabytes de datos. La elección de eliminar el valor h, que multiplicaba los valores de los pesos de ciertas capas e introducir la versión de la capa *shortcut* con normalización y pesos por capa fue de manera experimental a manera de prueba y error.

[1] El repositorio original fue abandonado por su autor original y el usuario AlexeyAB continua el desarrollo de este *framework*, por lo que se utilizó el siguiente repositorio: `https://github.com/AlexeyAB/darknet`

[2] CUDA es un conjunto de herramientas desarrollada por el fabricante de GPUs Nvidia, que permite hacer uso más eficiente de sus GPUs para realizar operaciones de álgebra lineal en paralelo. `https://developer.nvidia.com/cuda-toolkit`

[3] CUDNN es una biblioteca de redes neuronales profundas desarrollada por el fabricante Nvidia, implementa varias funciones "primitivas"(como convolución, pooling, normalización y capas de activación) para facilitar y hacer más eficiente la implementación de algoritmos de aprendizaje profundo usando GPUs. `https://developer.nvidia.com/cudnn`

Capítulo 4

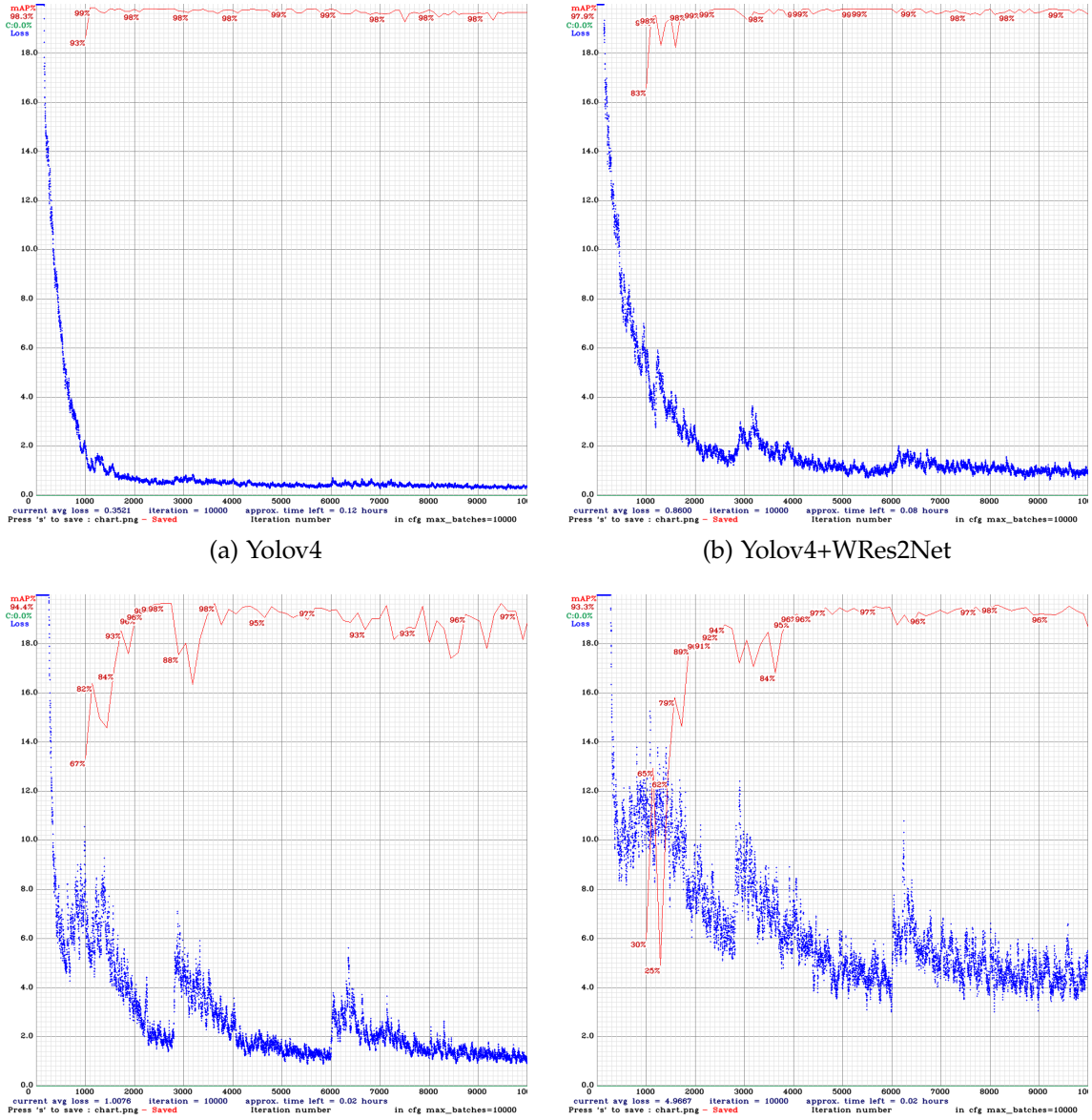
Resultados y discusiones

4.1. Entrenamiento

Luego de entrenar los modelos por 10000 iteraciones se obtuvieron las gráficas del valor de la función de pérdida, junto al cálculo del mAP (sobre el conjunto de validación) cada 100 iteraciones, que se muestran en la figura 4.1. Como se puede observar el modelo YoloV4 con WRes2Net converge rápidamente, aún con menos parámetros que el *backbone* original de YoloV4.
En el cuadro 4.1 se puede observar los valores finales de los pesos aprendidos para cada capa de *shortcut*.

Modelo	**Número de capa**	W_1	W_2
YoloV4+WRes2Net	6	0.990074	1.014566
	11	0.983940	1.023335
	14	0.987035	1.015991
	19	0.984751	1.014417
	24	1.000256	1.022918
	27	0.981539	1.018605
	32	0.997042	1.005113
YoloV7+WRes2Net	6	0.918814	1.142164
	11	0.914263	1.212030
	14	0.870816	1.149832
	19	0.831479	1.166121
	24	1.015152	1.208013
	27	0.799539	1.199945
	32	0.903181	1.106653

Cuadro 4.1: Valores finales de los pesos aprendidos en las capas *shortcut*.



(a) Yolov4 (b) Yolov4+WRes2Net

(c) Yolov7 (d) Yolov7+WRes2Net

Figura 4.1: Gráficas de entrenamiento de los diferentes modelos entrenados.

4.2. Validación

Después de entrenar los modelos varias veces cambiando distintos hiperparámetros se obtuvieron las métricas para el conjunto de validación, que ayudaron a verificar si los cambios en los hiperparámetros mejoraron o empeoraron el modelo. Estas se pueden observar en el cuadro 4.2. Estas nos indican que tan bien se comporta nuestro modelo en imágenes que no están presentes en el conjunto de entrenamiento (nuevas). Aunque nuestro modelo tiene menos parámetros, logra un mejor *IoU* y F_1-*Score* que

los otros modelos, lo que la hace más eficiente.

Modelo	Precision	Recall	mAP@0.5 (%)	IoU (%)	F1-Score
Yolo V4	0.91	**0.99**	**99.20**	80.57	0.95
Yolo V4+WRes2Net	**0.96**	0.98	99.01	**86.98**	**0.97**
Yolo V7	0.92	0.91	97.81	79.48	0.92
Yolo V7+WRes2Net	0.88	0.85	97.75	76.09	0.87

Cuadro 4.2: Resultados de desempeño en el conjunto de validación de los modelos comparados.

4.3. Pruebas

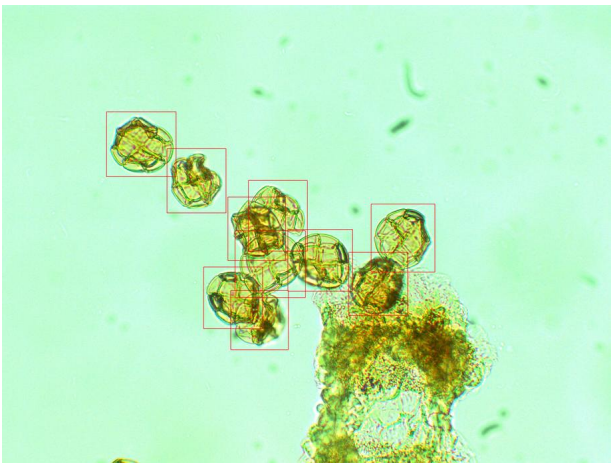
Finalmente, una vez realizadas las modificaciones y ajustes necesarios que resultaran en una mejora de nuestro modelo en el conjunto de validación y escogiendo este último como modelo final para ser evaluado, se calcularon las métricas en el conjunto de pruebas para saber si nuestro modelo mantiene el desempeño observado en el conjunto de validación.

Se puede observar que nuestro modelo logra un mejor desempeño que los demás modelos en la mayoría de las métricas en el conjunto de prueba (Cuadro 4.3), lo que comprueba que nuestro modelo funciona. En el cuadro 4.6 se muestra un desglose de los *Average Precision*, los verdaderos positivos y falsos positivos por cada clase para los cuatro modelos. A pesar tener aproximadamente 17 Millones de parámetros menos que YOLOV4, el modelo WRes2Net consigue un rendimiento bastante cercano al de YOLOV4, aumentando la velocidad de detección sin comprometer de manera significativa la precisión.

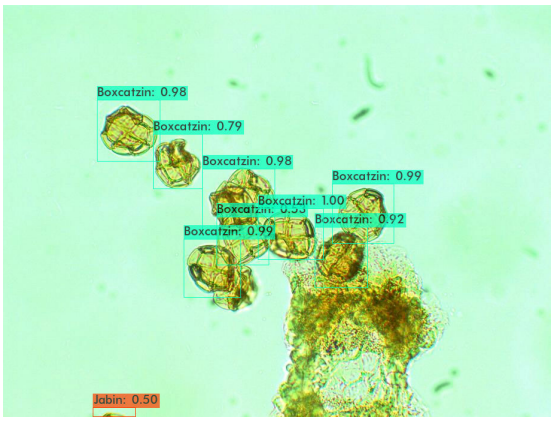
Modelo	Precision	Recall	mAP@0.5 (%)	IoU (%)	F1-Score
Yolo V4	0.92	0.99	98.68	81.72	0.96
Yolo V4+WRes2Net	**0.93**	0.99	**98.83**	**83.86**	0.96
Yolo V7	0.90	0.85	93.81	75.88	0.87
Yolo V7+WRes2Net	0.89	0.89	94.91	76.86	0.89

Cuadro 4.3: Resultados de desempeño en el conjunto de pruebas de los modelos comparados.

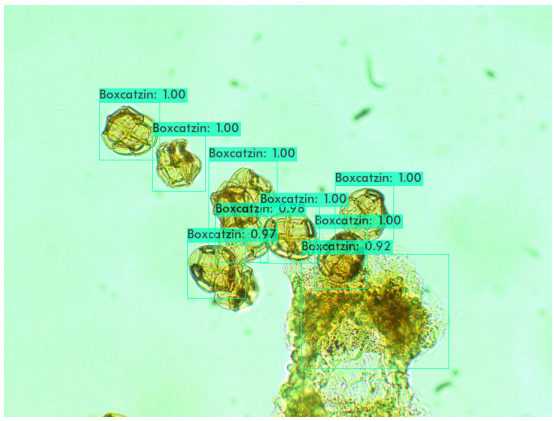
En la Figura 4.2 se observa el resultado de la detección de los distintos modelos comparados sobre una imagen de gránulos de polen de la especie Box catzín.



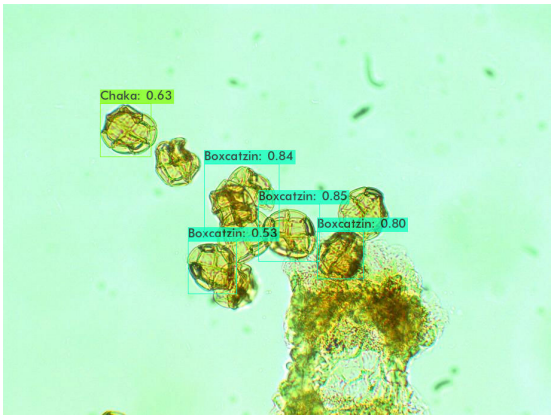
(a) Ground Truth



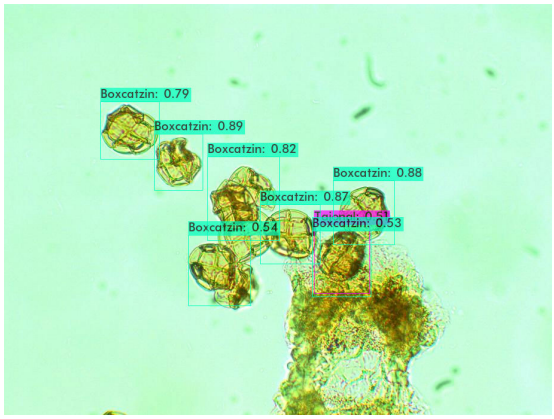
(b) YoloV4



(c) YoloV4+WRes2Net



(d) YoloV7



(e) YoloV7+WRes2Net

Figura 4.2: Comparación de detecciones de los diferentes modelos en una muestra de gránulos de box catzín.

En el cuadro 4.4 se observa el número de parámetros de los *backbones* de los modelos comparados. Adicionalmente el cuadro 4.5 muestra la velocidad de cada modelo para

procesar y realizar inferencias sobre las imágenes.

Backbone	**Número de parámetros entrenables**
CSPDarknet-53 (YoloV4)	27,624,520 ($\approx$ 27,5M)
WRes2Net	10,211,726 ($\approx$ 10,2M)
CSPVoVNet (Yolov7)	13,361,792 ($\approx$ 13,3M)

Cuadro 4.4: Número de parámetros entrenables de los *backbones* de los modelos comparados.

Modelo	**Imágenes por segundo**	**BLFOPs**
YoloV4	62.50	59.58
YoloV4+WRes2Net	111.11	39.21
YoloV7	43.47	103.27
YoloV7+WRes2Net	52.63	69.74

Cuadro 4.5: Velocidad de los modelos medida en imágenes por segundo y billones de operaciones de punto flotante por segundo (BFLOPs).

Modelo	**Tajonal**			**Jabín**			**Chaká**			**Box catzín**		
	AP (%)	**TP**	**FP**	**AP (%)**	**TP**	**FP**	**AP (%)**	**TP**	**FP**	**AP (%)**	**TP**	**FP**
YoloV4	99.36	57	4	98.58	77	8	**99.76**	67	5	**97.01**	31	2
YoloV4+WRes2Net	**99.82**	57	2	**99.90**	77	5	99.18	66	9	96.43	31	2
YoloV7	99.79	55	3	95.62	66	4	87.31	52	15	92.52	26	0
YoloV7+WRes2Net	98.70	55	7	94.89	67	2	93.96	56	11	92.07	30	7

Cuadro 4.6: Resultados de *Average precision* (AP), verdaderos positivos (TP) y falsos positivos (FP) por clase en el conjunto de pruebas de los modelos comparados.

Capítulo 5

Conclusiones

En el presente trabajo de tesis se propuso un *backbone* para la detección y clasificación automática de gránulos de polen de cuatro especies vegetales de importancia apícola en el estado de Yucatán.
Nuestro modelo propuesto, **YoloV4+WRes2Net** alcanza un rendimiento muy cercano a la versión original de YOLOV4 además muestra ser más rápido y más eficiente con los parámetros aprendidos que otros modelos con los que se probó, además de reducir el número de parámetros entrenables y así reduciendo a su vez el tiempo de entrenamiento y detección. Sin embargo, debido a que la base de datos aún es muy pequeña (370 imágenes), se requieren más muestras para que el modelo generalice mejor. En el cuadro 4.3 se observa que utilizar nuestro modelo como *backbone* mejora el mAP@0.5 y el IoU tanto en la YOLOV4 como en la YOLOV7. Además se observó que el valor del hiperparámetro *h*, que originalmente estaba presente para la tarea de segmentación, no funcionó muy bien para esta tarea de detección y se cambió para ser un parámetro entrenable. En el cuadro 4.1 se observa que la mayoría de los pesos aprendidos en las capas de suma son cercanos a uno y sugiere que los pesos o el hiperparámetro h casi no obtuvo efecto.

5.0.1. Trabajo a futuro

Los resultados obtenidos en este trabajo son el comienzo para continuar el desarrollo de un sistema automático para la detección, clasificación y conteo de gránulos de diversas especies vegetales de importancia apícola en el estado de Yucatán, sin la necesidad de *hardware* costoso.
Aumentar el tamaño de la base de datos, incluyendo imágenes que contengan gránulos de polen mezclados e imágenes que no contengan gránulos de polen podría mejorar

la generalización del modelo, ya que la base de datos actual solamente contiene una clase de polen en cada imagen. Adicionalmente incrementar el número de gránulos de polen presentes en cada imagen también resultaría beneficioso.
Por último, utilizar una operación de concatenación en lugar de las dos sumas cortas es algo que no se probó y sería interesante investigar el comportamiento en ese caso. Utilizar funciones de activación *Mish* resultó en una mejora de los modelos. Otra opción a probar sería alguna otra combinación de módulos para el *neck* (como SPP, SAM, PANet, CSPNet, etc).

Bibliografía

[1] Alejandra Vanessa Castillo Cázaresa, Yolanda Beatriz Moguel Ordóñezb, Moisés Alberto Cortés Cruzc, Elsa Espinosa Huertaa, Miguel Enrique Arechavaleta Velascod, and María Alejandra Mora Avilésa. Composición botánica de mieles de la península de yucatán, mediante qpcr y análisis de curvas de disociación. Rev. mex. de cienc. pecuarias, 7(4):489–505, 2016.

[2] Saqer S. Alotaibi, Samy M. Sayed, Manal Alosaimi, Raghad Alharthi, Aseel Banjar, Nosaiba Abdulqader, and Reem Alhamed. Pollen molecular biology: Applications in the forensic palynology and future prospects: A review. Saudi Journal of Biological Sciences, 27(5):1185–1190, 2020.

[3] Mohamed Zobaa, Francisca Oboh-Ikuenobe, and Michael Zavada. Applications of palynology for hydrocarbon exploration: case studies from egypt, eastern tennessee (usa) and the gulf of mexico. 06 2009.

[4] Amar "Daood, Eraldo Ribeiro, and Mark"Bush. "pollen grain recognition using deep learning". In George "Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Fatih Porikli, Sandra Skaff, Alireza Entezari, Jianyuan Min, Daisuke Iwai, Amela Sadagic, Carlos Scheidegger, and TobiasÏsenberg, editors, .ªdvances in Visual Computing", pages "321–330", Çham", "2016". "Springer International Publishing".

[5] Leyra Martínez, Pedro Arguijo, Antonio Hiram Vázquez López, and Roberto Ángel Meléndez Armenta. Clasificación de granos de polen: Efecto de la distancia inter-pixel en glcm. Research in Computer Science, 149(8):753–762, 2020.

[6] Peter He, Gerard Glowacki, and Alexis Gkantiragas. Unsupervised representations of pollen in bright-field microscopy. CoRR, abs/1908.01866, 2019.

[7] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. CoRR, abs/1506.02640, 2015.

[8] Jaideep Murkute. Robust pollen imagery classification with generative modeling and mixup training. CoRR, abs/2102.13143, 2021.

[9] Elżbieta Kubera, Agnieszka Kubik-Komar, Paweł Kurasiński, Krystyna Piotrowska-Weryszko, and Magdalena Skrzypiec. Detection and recognition of pollen grains in multilabel microscopic images. Sensors, 22(7), 2022.

[10] Rafael C. Gonzalez and Richard E. Woods. Digital image processing. Prentice Hall, Upper Saddle River, N.J., 2008.

[11] George H. Joblove and Donald Greenberg. Color spaces for computer graphics. In Proceedings of the 5th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '78, page 20–25, New York, NY, USA, 1978. Association for Computing Machinery.

[12] Mark D Ricker. Pixels, bits, and guis: The fundamentals of digital imagery and their application by plant pathologists. Plant disease, 88(3):228–241, 2004.

[13] S Chitra and G Balakrishnan. Comparative study for two color spaces hscbcr and ycbcr in skin color detection. Applied Mathematical Sciences, 6(85):4229–4238, 2012.

[14] Kyungjun Lee and Jechang Jeong. Multi-color space network for salient object detection. Sensors, 22(9), 2022.

[15] Samruddhi Y. Kahu, Rajesh B. Raut, and Kishor M. Bhurchandi. Review and evaluation of color spaces for image/video compression. Color Research & Application, 44(1):8–33, 2019.

[16] E. Welch, R. Moorhead, and J.K. Owens. Image processing using the hsi color space. In IEEE Proceedings of the SOUTHEASTCON '91, pages 722–725 vol.2, 1991.

[17] Vittorio Ferrari, Tinne Tuytelaars, and Luc Van Gool. Object detection by contour segment networks. In Proceeding of the European Conference on Computer Vision, volume 3953 of LNCS, pages 14–28. Elsevier, June 2006.

[18] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001, volume 1, pages I–I, 2001.

[19] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In 2008 IEEE Conference on Computer Vision and Pattern Recognition, pages 1–8, 2008.

[20] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, pages 886–893 vol. 1, 2005.

[21] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks, 2013.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 25. Curran Associates, Inc., 2012.

[23] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CoRR, abs/1311.2524, 2013.

[24] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. CoRR, abs/1512.02325, 2015.

[25] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4):541–551, 1989.

[26] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016.

[27] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. CoRR, abs/1811.03378, 2018.

[28] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. CoRR, abs/1710.05941, 2017.

[29] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. Neurocomputing, 503:92–108, 2022.

[30] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. towards data science, 6(12):310–316, 2017.

[31] Bin Ding, Huimin Qian, and Jun Zhou. Activation functions and their characteristics in deep neural networks. In 2018 Chinese Control And Decision Conference (CCDC), pages 1836–1841, 2018.

[32] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In ICML, pages 807–814, 2010.

[33] Rich Caruana, Steve Lawrence, and C. Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In T. Leen, T. Dietterich, and V. Tresp, editors, Advances in Neural Information Processing Systems, volume 13. MIT Press, 2000.

[34] Diganta Misra. Mish: A self regularized non-monotonic neural activation function. CoRR, abs/1908.08681, 2019.

[35] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. CoRR, abs/1512.03385, 2015.

[36] Ke Zhang, Miao Sun, Tony X. Han, Xingfang Yuan, Liru Guo, and Tao Liu. Residual networks of residual networks: Multilevel residual networks. CoRR, abs/1608.02908, 2016.

[37] Andreas Veit, Michael J. Wilber, and Serge J. Belongie. Residual networks are exponential ensembles of relatively shallow networks. CoRR, abs/1605.06431, 2016.

[38] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. CoRR, abs/1605.07146, 2016.

[39] Falong Shen and Gang Zeng. Weighted residuals for very deep networks. CoRR, abs/1605.08831, 2016.

[40] David Balduzzi, Marcus Frean, Lennox Leary, J. P. Lewis, Kurt Wan-Duo Ma, and Brian McWilliams. The shattered gradients problem: If resnets are the answer, then what is the question? In Doina Precup and Yee Whye Teh, editors, Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 342–350. PMLR, 06–11 Aug 2017.

[41] Boris Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, Advances in Neural Information Processing Systems, volume 31. Curran Associates, Inc., 2018.

[42] Alexander Rehmer and Andreas Kroll. On the vanishing and exploding gradient problem in gated recurrent units. IFAC-PapersOnLine, 53(2):1243–1248, 2020. 21st IFAC World Congress.

[43] Hong Hui Tan and King Hann Lim. Vanishing gradient mitigation with deep learning neural network optimization. In 2019 7th International Conference on Smart Computing & Communications (ICSCC), pages 1–4, 2019.

[44] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. Loss functions for neural networks for image processing. CoRR, abs/1511.08861, 2015.

[45] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. CoRR, abs/1702.05659, 2017.

[46] Nan Cui. Applying gradient descent in convolutional neural networks. Journal of Physics: Conference Series, 1004(1):012027, apr 2018.

[47] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. CoRR, abs/2004.10934, 2020.

[48] Allan E. Ojeda Pat. Res2unet: Una red completamente convolucional para la segmentación del parásito trypanosoma cruzi. Master's thesis, Universidad Autónoma de Yucatán, Septiembre 2020.

[49] .º. Ronneberger, P.Fischer, and T. Brox". ü-net: Convolutional networks for biomedical image segmentation". In "Medical Image Computing and Computer-Assisted Intervention (MICCAI)", volume "9351.ºf "LNCS", pages "234–241". "Springer", "2015". "(available on arXiv:1505.04597 [cs.CV])".

[50] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. CoRR, abs/1612.08242, 2016.

[51] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. CoRR, abs/1804.02767, 2018.

[52] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. CoRR, abs/2011.08036, 2020.

[53] Chuyin Li, Lu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, L. Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. Yolov6: A single-stage object detection framework for industrial applications. ArXiv, abs/2209.02976, 2022.

[54] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. ArXiv, abs/2207.02696, 2022.

[55] Amirkoushyar Ziabari, Derek C. Rose, Abbas Shirinifard, and David J. Solecki. Yolo2u-net: Detection-guided 3d instance segmentation for microscopy. ArXiv, abs/2207.06215, 2022.

[56] Siyuan Liang, Hao Wu, Li Zhen, Qiaozhi Hua, Sahil Garg, Georges Kaddoum, Mohammad Mehedi Hassan, and Keping Yu. Edge yolo: Real-time intelligent object detection system based on edge-cloud cooperation in autonomous vehicles. ArXiv, abs/2205.14942, 2022.

[57] Zhaobin Wang, E. Wang, and Ying Zhu. Image segmentation evaluation: a survey of methods. 53:5637–5674, 2020.

[58] Abdel Aziz Taha and Allan Hanbury. Metrics for evaluating 3d medical image segmentation: analysis, selection, and tool. 15:29, 2015.

[59] Haidi Zhu, Haoran Wei, Baoqing Li, Xiaobing Yuan, and Nasser Kehtarnavaz. A review of video object detection: Datasets, metrics and methods. Applied Sciences, 10(21), 2020.

[60] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. A survey on performance metrics for object-detection algorithms. In 2020 International Conference on Systems, Signals and Image Processing (IWSSIP), pages 237–242, 2020.

[61] Ilya Loshchilov and Frank Hutter. SGDR: stochastic gradient descent with restarts. CoRR, abs/1608.03983, 2016.

[62] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013–2016.