PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

LOCALIZACIÓN Y CLASIFICACIÓN DE ÁRBOLES Y EDIFICACIONES EN IMÁGENES AÉREAS EMPLEANDO APRENDIZAJE PROFUNDO

Tesis para obtener el Título profesional de Ingeniera Electrónica

AUTORA:

Pamela Enriquez Rodriguez

ASESOR:

Donato Andres Flores Espinoza

Lima, junio, 2024

Informe de Similitud

Yo, **Donato Andres Flores Espinoza**, docente de la Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú, asesor de la tesis titulada **LOCALIZACIÓN Y CLASIFICACIÓN DE ÁRBOLES Y EDIFICACIONES EN IMÁGENES AÉREAS EMPLEANDO APRENDIZAJE PROFUNDO**, de la autora **Pamela Enríquez Rodríguez**, dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 11%. Así lo consigna el reporte de similitud emitido por el software *Turnitin* el 07/06/2024.
- He revisado con detalle dicho reporte y la Tesis o Trabajo de Suficiencia Profesional, y no se advierte indicios de plagio.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha:

Lima, 7 de junio de 2024

Apellidos y nombres del asesor: Flores Espinoza, Donato Andres	
DNI: 06017817	Firma
ORCID: https://orcid.org/0000-0003-2092-7666	

Resumen

La presente tesis muestra el diseño de un detector de árboles y edificaciones en imágenes aéreas elaborado en base a algoritmos de aprendizaje profundo, cuyas redes troncales para la extracción de características son redes neuronales convolucionales. Este trabajo es parte de la tarea de automatización de un sistema de inspección de fajas de servidumbre que recibe imágenes capturadas por drones.

Inicialmente, el trabajo se ha centrado en el etiquetado de árboles y edificaciones en imágenes aéreas para la elaboración del *dataset*; para ello, se ha utilizado la herramienta *Image Labeler* de Matlab. Posteriormente, se dividió dicho conjunto de datos en data de entrenamiento (80%), validación (10%) y evaluación (10%); además de emplear la función *imageDataAugmenter* para incrementar la cantidad de imágenes disponible. Seguidamente, se procedió con el entrenamiento de la red bajo ciertos valores de hiperparámetros y; finalmente, se evaluó la eficacia del detector bajo ciertas métricas como precisión, sensibilidad y precisión promedio media.

Los resultados obtenidos muestran que el detector diseñado e implementado en Pytorch delimita correctamente la ubicación de los árboles y edificaciones en imágenes aéreas; además de etiquetarlos con su clase correspondiente. Esto se evidencia en los valores de precisión promedio del 70% para la clase árboles y del 63% para la clase edificaciones, logrando una precisión promedio media del 67%.

Agradecimientos

Quiero agradecer, en primer lugar, a Dios, por bendecir mi vida y guiarme a lo largo de mi existencia siendo mi apoyo y fortaleza en aquellos momentos de dificultad y debilidad.

También quiero agradecer a mis padres, Leo y Quely, por creer en mí y apoyar mis sueños incondicionalmente, por los consejos, valores y principios que me han inculcado. A mis hermanas, Irene y Diara, por ser parte de mis aventuras. Y a toda mi familia, quienes de alguna forma u otra manera me acompañaron en la realización de mis metas.

Además, quiero agradecer a Isabel, quien fue mi salvavidas en mis momentos más difíciles y quien me brindó las herramientas para enfrentar mis desafíos; a mi gran amiga Paula, quien fue parte de mi vida universitaria y con quien compartí largas horas de estudio.

Finalmente, quiero agradecer a mi asesor, el Ing, Andrés Flores, por su paciencia y guía durante el desarrollo de esta tesis.

Índice

_
_

Índice de Imágenes

Índice de Tablas

Introducción

Las áreas o fajas de servidumbre de redes de transmisión de energía eléctrica son usualmente invadidas por viviendas, vehículos y vegetación alta como los árboles, tal ocupación indebida impide que las empresas concesionarias realicen trabajos de mantenimiento en las instalaciones eléctricas; asimismo, tanto el crecimiento de árboles como la exposición cercana a las líneas energizadas son una fuente de riesgo eléctrico-mecánico que podría causar pérdidas humanas y materiales. Es por ello por lo que las concesionarias eléctricas deben realizar inspecciones periódicas de las fajas de servidumbre bajo su dominio, dicha tarea lo realizan sobrevolando el área de interés con drones que toman imágenes desde diversos ángulos y alturas. Sin embargo, a pesar del uso de drones, la identificación de objetos invasores se realiza de manera manual, la cual es una tarea ardua e ineficiente.

En esta tesis se plantea automatizar la tarea de inspección con un detector de árboles y edificaciones diseñado a partir de redes neuronales convolucionales, las cuales están basadas en algoritmos de aprendizaje profundo supervisado.

Capítulo 1

1.1 Marco problemático

Las redes de transmisión de energía eléctrica recorren ciudades y campos a lo largo del territorio nacional peruano para proveer de electricidad a las viviendas, industrias, comercios e instituciones públicas y privadas. Estas líneas de transmisión se encuentran conectadas entre torres, los cuales están dentro de una zona o faja de servidumbre definida como la proyección sobre el suelo de la faja ocupada por los conductores más la distancia de seguridad [1]. Esta última depende de la tensión nominal de las líneas tal como se muestra en la Tabla 219 del Código Nacional de Electricidad [2]. Asimismo, la representación gráfica de la faja de servidumbre se observa en la Figura 219-1 de dicho código.

Las fajas de servidumbre para líneas aéreas se establecen con el propósito de brindar facilidades para la instalación, operación y mantenimiento de las instalaciones eléctricas por parte de las empresas concesionarias, así como también para salvaguardar la seguridad pública, es decir, la integridad física de las personas y bienes frente a situaciones de riesgo eléctrico-mecánico [2] generados, por ejemplo, por los árboles o las ramas de estos cuando están demasiado cerca de las líneas eléctricas, lo cual produciría arcos eléctricos [3]. Es por ello que las concesionarias están obligadas a velar por el cumplimiento del espacio libre de los anchos mínimos de seguridad de las fajas de servidumbre sobre los bienes de dominio y uso público, tarea que de no cumplirse es sancionada con una multa según lo impuesto en la Escala de Multas y Sanciones emitidas por OSINERG.

La inspección periódica de las fajas de servidumbre se realiza a partir de las imágenes aéreas tomadas por drones sobre el área de interés desde diversos ángulos y alturas. El uso de los drones facilita parte de la tarea de inspección, pero esta continúa siendo sumamente laboriosa e ineficiente, pues las imágenes son inspeccionadas de manera manual. Por ello, la necesidad de automatizar dicha tarea mediante el uso de algoritmos de detección de objetos que involucra la localización y clasificación de estos es un reto que actualmente las concesionarias buscan implementar.

Existen algoritmos de localización y clasificación de objetos basados en aprendizaje automático que pueden ser supervisados y no supervisados, de los cuales la mayor parte de las investigaciones recurren a técnicas de clasificación supervisada que cuenta con procesos de entrenamiento y aprendizaje antes de la clasificación real de las imágenes; pero también se observa interés en las no supervisadas que no usan conjuntos de datos de entrenamiento etiquetados. Las técnicas supervisadas más empleadas han sido el árbol de decisión, las redes neuronales y las máquinas de vectores de soporte [4].

En esta tesis, se empleará algoritmos de aprendizaje automático supervisados para automatizar la tarea de detección de árboles y edificaciones en imágenes aéreas tomadas por un dron como parte de la tarea de automatización de la inspección de fajas de servidumbre. Específicamente, se hará uso de la técnica de aprendizaje supervisado basado en redes neuronales convolucionales (CNN), pues estas se distinguen de otros modelos en que no requieren un diseño manual de características de los datos para funcionar; las características se aprenden automáticamente mediante el proceso de optimización, también llamado entrenamiento. Y, en particular, las capas convolucionales que componen la red permiten aplicar convoluciones con filtros aprendidos para un mejor desempeño y eficiencia [5] .

1.2 Estado del arte

A continuación, se mostrará una revisión de algunas redes basadas en algoritmos de aprendizaje profundo para la detección (localización y clasificación) de objetos en imágenes aéreas. Las arquitecturas de dichas redes presentan en común tres etapas compuestas cada una de ellas por uno o varios módulos; la primera llamada *backbone*, compuesta por una red neuronal profunda para la extracción de características de las imágenes de entrada; la segunda, *neck*, encargada del refinamiento de las características extraídas en la etapa anterior; y la tercera, *head*, encargada de la clasificación y localización de los objetos de interés.

1.2.1 *End-to-End Edge Enhanced GAN and Object Detector (EESRGAN)* [6]

Red propuesta para detectar con alta precisión objetos pequeños en imágenes aéreas de baja resolución, cuya arquitectura *end-to-end* se basa en la mejora los bordes del contenido de las imágenes y la ampliación de la resolución de estas últimas.

La arquitectura de la red consta de dos módulos, un Generador y un Discriminador; el primero está conformado a su vez por un *Generator Network (G)* y un *Edge Enhancement Network (EEN);* y el segundo, por un *Discriminator* (D_{Ra}) y un Detector de Objetos (Det). La red *G* recibe las imágenes en baja resolución (LR) y las convierte en imágenes intermedias de super resolución (ISR), las cuales pasan por la red *EEN* y tras ella se obtiene las imágenes en super resolución (SR) que son la entrada del Detector de Objetos, el cual finalmente se encarga de la clasificación y localización de los objetos de interés. Las imágenes ISR también son la entrada para el D_{Ra} que se encarga de discriminarlas con las imágenes en alta resolución (HR) *Ground Truth*. El flujo de trabajo mencionado de la red *EESRGAN* se muestra en la Figura 1.

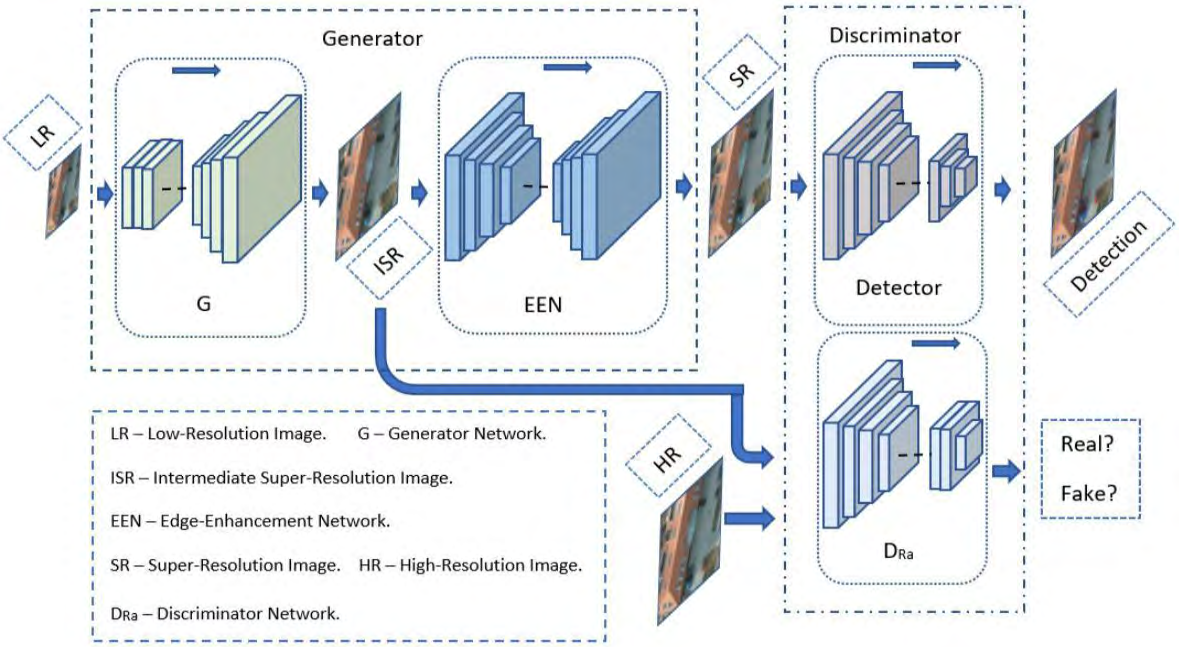


Figura 1. *EESRGAN Framework* [6].

En el módulo Generador, la red *G* está conformada por una seria de capas convolucionales y *Residual-in-Residual Dense Block (RRBD)*, esta último bloque es una red residual multinivel con conexiones densas, empleada estratégicamente ya que las conexiones densas incrementan la capacidad de la red y el escalado residual previene condiciones inestables durante el entrenamiento [6]; mientras, la segunda red *EEN*, encargada de remover el ruido y mejorar los bordes de la imagen, está conformada también por capas y bloques *RRBD*, pero antes de ellos, se emplea un operador Laplaciano que extrae los bordes originales de la imagen, esta información de los bordes es luego recién insertada en las capas y bloques *RRBD*; y tras ellos, se emplea una máscara sumado a una función de activación sigmoide que remueve el ruido de los bordes; finalmente, los bordes mejorados son insertados en la imagen original al cual también se le resta los bordes originales sustraídos con el operador Laplaciano.

Por otro lado, en el módulo Discriminador, D_{Ra} está basado en una red *ESRGAN* que emplea la *VGG-19* como red trocal, mientras que el Detector, en una red *Faster-RCNN* o una red *SSD*, ambas empleadas por separado para comparar resultados.

La red propuesta *EESRGAN* es entrenada y evaluada en dos *datasets*, *OGST* y *COWC*, el primero es un conjunto de datos elaborado manualmente que contiene imágenes de tanques y el segundo es un conjunto de datos abierto que contiene imágenes de carros; siendo la diferencia entre ambos en que los objetos de interés del primero son ligeramente más grandes que del segundo. La red *EESRGAN* empleando a la *Faster-RCNN* como Detector presenta un mejor rendimiento, ya que obtiene el mejor puntaje de AP@0.5:00.5:0.95 con valores de 95.5% y 83.2% en el *COWC* y *OGST dataset*, respectivamente.

1.2.2 ***VistrongerDet*** [7]

Red propuesta para la detección de objetos aéreos constituido por bloques mejorados respecto a los convencionales de un detector general en los tres niveles de su flujo de trabajo. El primero, un bloque *Feature Pyramid Network (FPN)* mejorado que emplea factores de fusión dinámicos entre sus capas a diferentes niveles y una máscara de supervisión para evitar el impacto generado por la amplia escala de objetos distribuidos desequilibradamente, y la degradación de los objetos diminutos y pequeños, respectivamente. El segundo, un bloque *Adjacent ROI Fusion (ARF)* más un módulo de mecanismo de atención espacial, donde *ARF* se encarga de fusionar los *ROIs Features* desde niveles adyacentes de modo que se aprovecha la fuerte correlación de los mapas de características entre estos niveles; mientras, el módulo de atención se encarga de mejorar la sensibilidad de las *ROIs Features* fusionadas anteriormente. Y

el tercero, un bloque *HEAD* mejorado que emplea dos ramas paralelas de clasificación, una para las categorías de cabeza y otra para las de cola; además, para resolver el problema de clasificación de categorías similares, emplea en cada rama, un *Group-Softmax* y *Multi-Label Classification*. El flujo de trabajo de la red *VistrongerDet* es ilustrada en la Figura 2.

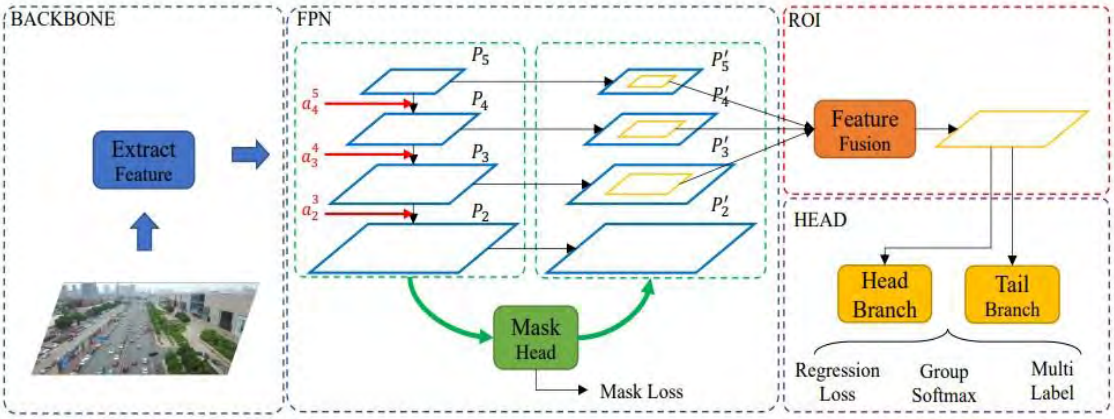


Figura 2. *VistrongerDet framework* [7].

El flujo de trabajo inicia con la extracción de los mapas de características de la imagen de entrada que son ingresados al primer bloque mejorado de la red, es decir, la red *FPN* mejorada que se encarga de fusionar dichos mapas de características mediante factores de fusión dinámicos, los cuales son calculados en base a la distribución de muestras de diferentes escalas en el conjunto de datos de entrenamiento [7]. Acto seguido, mediante el módulo *Spatial Attetion Fusion* (*SAF)*, los mapas resultantes de la red *FPN* mejorada se fusionan con los mapas de calor generados por la máscara de supervisión. Por cada mapa fusionado por *SAF*, se genera su respectivo *ROI Feature*, los cuales son ingresados al segundo bloque, el *ARF*, donde cada *ROI Feature* es mapeado en un nivel superior e inferior al que pertenece para luego integrarlos por una simple ponderación e ingresarlos a un módulo de mecanismo de atención espacial interno. Finalmente, los *ROIs Features* resultantes del módulo de atención ingresan al tercer bloque, la *HEAD*

mejorada, compuesta por ramas de *Fully Connected Layers* para clasificar los objetos de interés en objetos de cabeza o cola.

La red propuesta es entrenada y evaluada en *VisDrone2021-DET*, un conjunto de datos multiescala por su toma desde diversas alturas y ángulos; esta es evaluada bajo la métrica de mAP a diferentes umbrales de IoU, donde los resultados arrojan los siguientes valores: mAP@0.5 de 57.27%, mAP@0.75 de 34.81% y un mAP@0.5:0.05:0.95 de 33.85%.

1.2.3 ***Focus and Detect Network*** [8]

Focus and Detect (F&D) es una red de detección de objetos aéreos basada en la búsqueda y enfoque de regiones, método que permite con mayor precisión detectar objetos pequeños y medianos en imágenes densas y tomadas a gran escala, los cuales son características frecuentes de las imágenes aéreas. Esta red consta de dos subredes basadas en redes convencionales de detección de objetos, la primera subred llamada *Focus Stage* detecta las regiones focales a partir de un modelo gaussiano mixto, y la segunda, llamada *Detection Stage*, detecta objetos dentro de las regiones focales. La estructura del flujo de trabajo de la red propuesta se observa en la Figura 3, donde se visualiza que existe una etapa final de postprocesamiento que emplea los métodos de *Incomplete Box Suppression (IBS)* para suprimir los *Bounding Boxes (BB)* imcompletos generados por el solapamniento de las regiones focales y *Non-max Supression (NMS)* para eliminar el solapamiento de *BB* en un solo objeto.

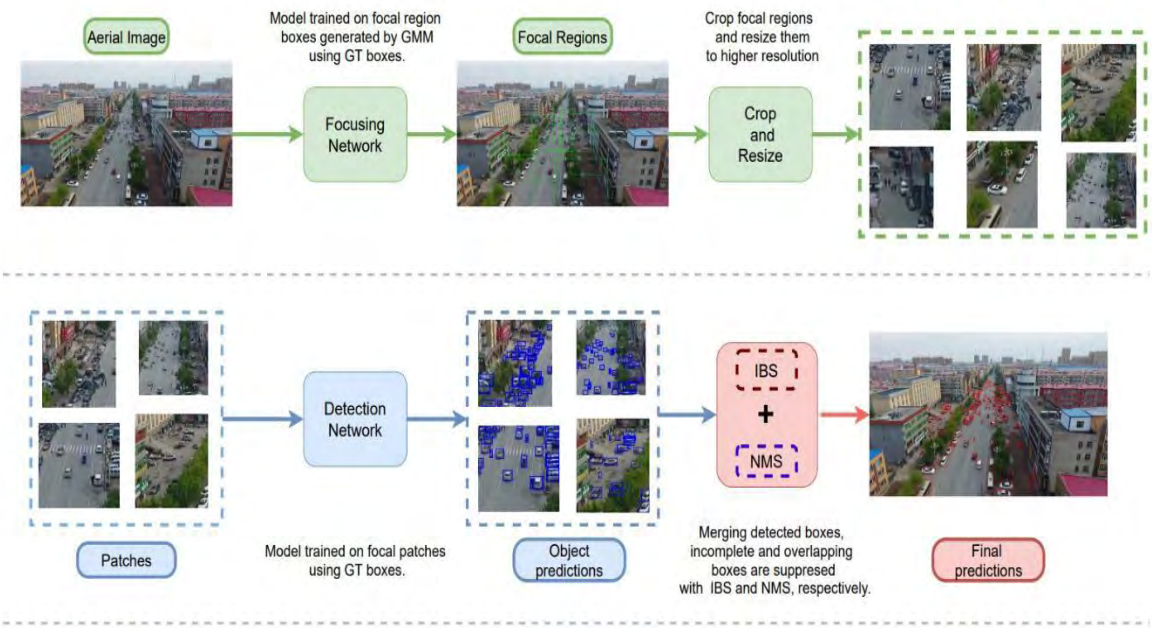


Figura 3. *Focus&Detect (F&D) framework* [8].

La subred *Focus Stage* consta de tres etapas, la primera, encargada de la generación de regiones focales propuestas que agrupan los *Ground Truth (GT) Boxes* usando un modelo gaussiano mixto, este modelo es un método de agrupamiento de aprendizaje máquina que recibe de entrada vectores distancia que se definen como vectores que salen de la cuadrícula predefinida de puntos y terminan en el centro de los *GT Boxes* [8], asimismo este modelo hace uso del algoritmo *Expectation Maximization (EM)* para modelar sus parámetros de media, varianza y covarianza. La segunda etapa extrae las características de la imagen de entrada empleando la red troncal ResNet-50 con sus tres últimas capas de convolución deformadas para una mejor transferencia de las características de las regiones focales; luego, emplea un *Feature Pyramid Network (FPN)* para explotar y refinar las características de las diferentes etapas de la red troncal; por último, la tercera etapa predice los *BB* de regiones focales empleando la *Generalized Focal Loss (GFL).*

Por otro lado, la subred *Detection Stage* presenta tres etapas similares a la anterior subred; la primera, recibe las regiones focales y las redimensiona a una mayor resolución; la segunda etapa extrae las características de dichas regiones empleando la red troncal *ResNetXt-101* con sus tres últimas capas de convolución deformadas; además, emplea un *FPN* para usar las características de las diferentes etapas de la red troncal. La tercera parte predice los *Bounding Boxes (BB)* de objetos empleando la *Generalized Focal Loss (GFL)*.

Esta red es entrenada y evaluada en dos *datasets*, *VisDrone2021-DET* y *UAVDT*, donde el segundo conjunto de datos en comparación al primero se centra demasiado en objetos pequeños. Las métricas empleadas para la evaluación de la red son tres: AP para diversas escalas de objeto, mAP a diferentes umbrales de IoU y el tiempo de ejecución promedio por imagen en la etapa de evaluación. Respecto a la primera métrica, los resultados arrojan un AP del 32.0% en la detección de objetos pequeños, un AP del 47.9% en objetos medianos, y un AP del 54.5% en objetos grandes. La segunda métrica empleada arroja los siguientes resultados, un mAP@0.5 de 66.12%, un mAP@0.75 de 44.64% y un mAP@0.5:0.05:0.95 de 42.06%, por último, la tercera métrica que evalúa la velocidad de detección de la red arroja un resultado de 1.362 s/img.

1.2.4 ***Eagle-YOLO*** [9]

Red inspirada en las características de visión de un águila para optimizar la red *You Only Look Once (YOLO),* y de esa manera presentar robustez al detectar objetos en imágenes aéreas con un complejo y variado *background*, con presencia de objetos multiescala con distribución desbalanceada entre ellos, en los cuales predominan los objetos de escala pequeña debido a la toma de *shots* desde alturas

considerables. Para lidiar con los problemas que conllevan las características mencionadas líneas arriba, la red emplea una arquitectura mejorada de *YOLOv5*, donde la etapa *backbone* para la extracción de características es la red *CSPDarknet53*, la etapa *neck*, para la mejora de estos mapas de características al fusionar información semántica de capas profundas e información de textura de capas superficiales, es la red *Bidirectional Feature Pyramid Network (Bi-FPN)*; y finalmente, la etapa *head*, para generar una matriz que almacena los vectores de regresión y las probabilidades de clases en cada nivel de la etapa anterior. El flujo de trabajo de la red propuesta se muestra en la Figura 4.

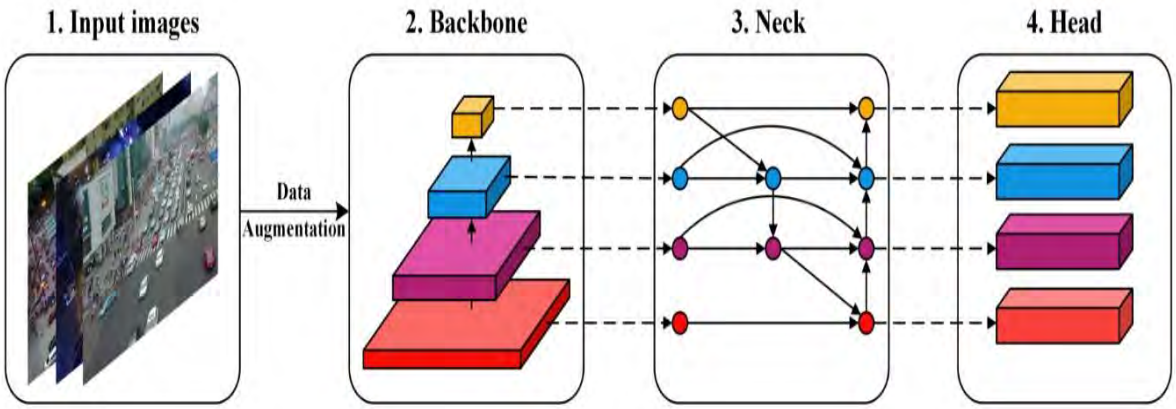


Figura 4. *Framework* de la red *Eagle-YOLO* [9].

El flujo de trabajo de la red inicia con la extracción de los mapas de características de la imagen de entrada mediante la red trocal *CSPDarknet53*, luego dichos mapas son ingresados la red *Bi-FPN* que fusiona los mapas de características por ponderación a diferentes escalas, a esta red se añade un *Large Kernel Attention Module (LKAM)* de modo que permita un enfoque en regiones de objetos y capture información semántica de estas. Este módulo genera mapas de atención a través de grandes filtros de convolución; sin embargo, para evitar el coste computacional, el módulo se descompone en *depth-wise convolution (DW-Conv)* con propiedades espaciales

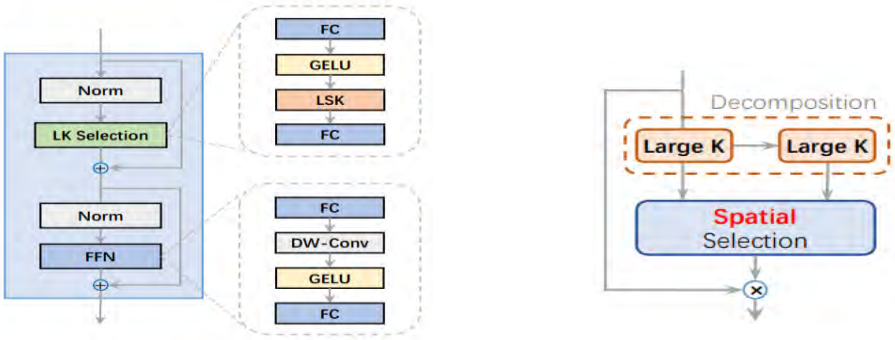
locales, *depth-wise dilation convolution (DW-D-Conv)* con propiedades espaciales de rango amplio y una convolución de 1x1. Finalmente, los mapas de características por cada nivel de la etapa anterior ingresan a una serie de capas convolucionales y se obtiene matrices que guardan los vectores de regresión y las probabilidades de clases. Cabe resaltar que se añadió una capa de bajo nivel a la red *Bi-FPN* que contiene información rica en detalles de objetos pequeños para la detección de estos con alta precisión y así mejore el rendimiento general del detector.

La red propuesta es entrenada y evaluada en *VisDrone2021-DET*, un conjunto de datos multiescala, donde predominan los objetos pequeños además de presentar una distribución desequilibrada por clases. La red es evaluada bajo la métrica de mAP que arroja los siguientes resultados: mAP@0.5 igual a 53.64% y mAP@0.5:0.05:0.95 igual a 32.91%.

1.2.5 *Detector Based on Large Selective Kernel Network (LSKNet)* [10]

Red basada en el aprovechamiento de la información contextual del objeto de interés en imágenes aéreas, siendo el rango de dicha información espacial variante según el tipo de objeto a detectar. Por ello, la red busca ajustar dinámicamente el campo receptivo de los mapas de características en su red troncal, y para ello emplea *Large Kernels Convolutions* y un módulo de mecanismo espacial selectivo, que en conjunto conforman el módulo *LSK*, el cual a su vez es insertado en el sub-bloque *Large Kernel Selection (LK Selection)* que junto al sub-bloque *Feed-forward Network (FFN)* conforman el bloque *LKSNet*. La red propuesta consta de varios de estos últimos bloques y es empleada como red troncal del flujo de trabajo de varios detectores de objetos. En la Figura 5a se observa la arquitectura del bloque *LSKNet* y en la Figura

5b, la del módulo *LSK*, además se detalla el *framework* del módulo en mención en la Figura 6.



(a))Arquitectura del bloque *LSKNet* (b)Arquitectura del módulo LSK

Figura 5. Arquitectura del bloque *LSKNet* y del módulo *LSK* [10].

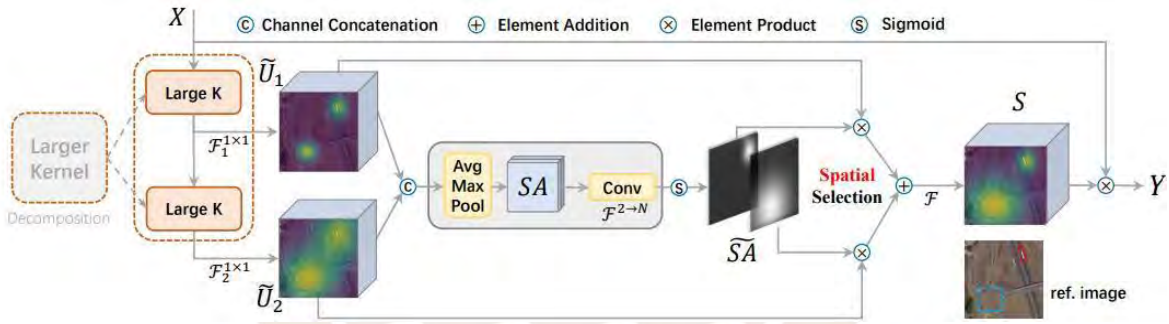


Figura 6. *Framework* del módulo *LSK* [10].

El módulo *LSK* consta de dos etapas, en la primera de ellas, la imagen de entrada se introduce a una secuencia de filtros convolucionales con un amplio rango de longitudes, pues se incrementa progresivamente el tamaño y el ratio de dilatación del filtro para así expandir rápidamente el campo receptivo del mapa de características, los cuales posteriormente son convolucionados de manera independiente con un filtro de 1x1, para luego ser concatenados y enviados a la segunda etapa que se basa en un módulo de mecanismo espacial selectivo. Dicha etapa inicia extrayendo la relación de características espaciales mediante un *average* y *maximum pooling* basados en canal; donde las salidas de estos descriptores de características espaciales son concatenadas y

luego convolucionadas en conjunto con un filtro 2xN, donde N es el número de filtros de la primera etapa. Acto seguido, se aplica una función sigmoide para obtener las máscaras individuales de selección espacial para cada mapa de características resultantes de la primera etapa. Finalmente, estos mapas ponderados son sumados y en conjunto multiplicados con la imagen de entrada.

La red propuesta es empleada como red troncal del detector de objetos orientados *O-RCNN*, siendo entrenada y evaluada en tres conjuntos de datos, de los cuales el *DOTA-v1.0* será seleccionado por su contenido de imágenes multiescala. La red es evaluada bajo tres métricas: AP para diversas escalas de objeto, mAP@0.5 y FPS. Respecto a la primera métrica, los resultados arrojan un AP del 82.50% en objetos pequeños, un AP del 79.9% en objetos medianos, y un AP del 81.99% en objetos grandes. La segunda métrica empleada indica un mAP@0.5 de 81.85%; y, por último, la tercera métrica que evalúa la velocidad de detección de la red arroja un resultado de 18.1 FPS. Cabe resaltar que la red mejora ligeramente su rendimiento tras ser ajustada con el método EMA [10], estos valores mejorados son los que se indicaron líneas arriba.

1.2.6 ***Network Based on Feature Alignment of Candidate Regions*** [11]

Red propuesta capaz de detectar objetos multiescala en imágenes aéreas, mostrando alta precisión en la detección de objetos pequeños tanto como las de mediana y larga escala. Para cumplir con dicho desafío, la red emplea tres módulos: el primero, *Attencion-Based Feature Alignment FPN (AFA-FPN)*, encargada de alinear y fusionar las características espaciales superficiales y las características semánticas profundas; el segundo, *Polarization Hybrid Domain Attetion (PHDA)*, que introduce un módulo de atención dual para eliminar el ruido en la imagen; y el tercero, *Rotation*

Branch, que permite la rotación del cuadro delimitador. La arquitectura completa de la red propuesta se observa en la Figura 7 que consta de 4 etapas, la primera de ellas está dada por la red troncal *Resnet-101* que se encarga de la extracción de características de la imagen de entrada, y las tres subsiguientes, conformadas por los tres módulos mencionados anteriormente. Tras la salida del tercer módulo, se aplica funciones para la clasificación de clase de los objetos dentro de las regiones de interés y la regresión de estos para un ajuste más acorde al *Ground Truth* de los cuadros delimitadores.

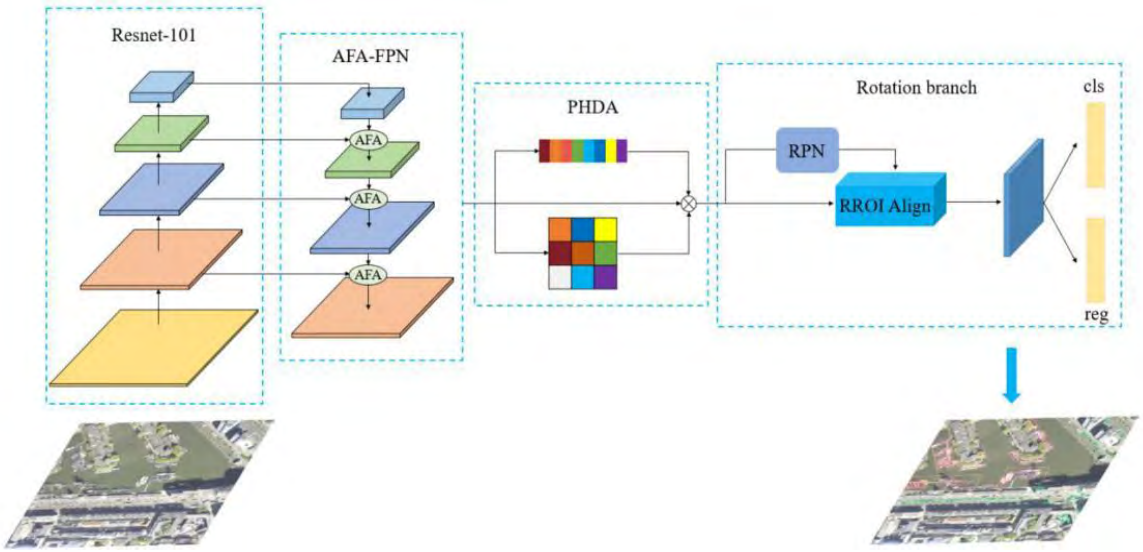


Figura 7. Arquitectura de la red *AFA-PHDA Network* [11].

El primer módulo *AFA-FPN* consta de dos submódulos, *Upsampling Calibration Path* y *Position Attention Path*, donde la entrada de ambos es la suma F^{i-1}_{sum} de un mapa de características superior F_i operado con la técnica de sobremuestreo bilinear tradicional, y el mapa de características inferior F_{i-1} de la red *FPN*. El primer submódulo permite sobremuestrear los límites del objeto con alta calidad y el segundo, permite resaltar los objetos de la imagen, el efecto de ambos permite que las características semánticas profundas converjan en determinadas posiciones de los objetos correspondientes en los

mapas de características superficiales [11].

El segundo módulo *PHDA* introduce un mecanismo dual basado en atención al canal y espacio a la salida del primer módulo; los resultados son fusionados junto a este último, y al resultado se le aplica una función de polarización para enfocarse en los objetos pequeños y reducir la interferencia del contexto irrelevante.

El tercer módulo, *Branch Rotation,* se divide en dos etapas, la primera llamada *RRoI Alignment* que se encarga de generar regiones de interés rotados a partir de los *RoIs* horizontales, y la segunda llamada *RRoI Rotation Position-Sensitive Pooling* que extrae las características invariantes de la rotación de los objetos orientados.

La red propuesta es entrenada y evaluada en el *DataSet RRSO* que es un conjunto de datos creado manualmente mediante recorte y escalamiento, este incluye 4000 imágenes de diversas escalas (pequeñas en su mayoría, medianas y grandes) de 15 clases de objetos. Las imágenes fueron tomadas desde diversas altitudes y orientaciones, y presenta escenarios complejos y simples, además de ser densos. La red es evaluada bajo tres métricas: AP para diversas escalas de objeto, mAP a diferentes umbrales de IoU y FPS. Respecto a la primera métrica, los resultados arrojan un AP del 78.21% en la detección de objetos pequeños, un AP del 84.77% en objetos medianos, y un AP del 80.43%, en objetos grandes. La segunda métrica empleada indica los siguientes resultados, un mAP@0.5 de 82.04%, un mAP@0.75 de 69.70% y un mAP@0.5:0.05:0.95 de 59.70%, y, por último, la tercera métrica que evalúa la velocidad de detección de la red arroja un resultado de 24.3 FPS.

Tabla 1. Cuadro comparativo del estado del arte [6], [7], [8], [9], [10], [11].

RED	Breve Resumen Metodología	Base de datos	Características Imágenes	Categorías	Requerimientos y consideraciones para el entrenamiento		
					Software	Hardware	Hiperparámetros
EESRGAN + FASTER-RCNN	Generador: Generator Network + EEN Discriminador: Discriminator D_{Ra} + Detector (Faster-RCNN / SSD)	COWC	Cantidad: 3340 64%/16%/20% Training/Validation/Testing Resolución: 256x256/64x64 (HR/LR) #Clases = 1	Car	Pytorch	2 NVIDIA GeForce Titan X	Adam ($\beta_1 = 0.9, \beta_1 = 0.999, \alpha = 0.001$) Decay Rate = ½ cada 50000 iteraciones
		OGST	Cantidad: 760 81%/9%/10% Training/Validation/Testing Resolución: 512x512/128x128 (HR/LR) #Clases = 1	Tank			
VistrongerDET	ResNet-50 + FPN enhancement+ ASF + HEAD enhancement (GS cls + ML cls)	VisDrone2021-DET*	Resolución: 1600x1050 Clases = 10	Pedestrian, People, Bicycle, Car, Van, Truck, Tricycle, Awning-Tricycle, Bus, Motor	Pytorch	8/1 GPUs	SGDM ($\alpha = 0.02$) Decay rate = 1/10 en 9th época, 1/100 en 12th época Batch Size = 2 #Épocas = 12 *Emplea Test Time Aumentation (TTA)
Focus and Detect	Focus Stage: ResNet-50 + FPN + Gaussiano Mixture Model +	VisDrone2021-DET*	Resolución Training: 400x1400 to 1200x1400 Resolución Testing: 1200x1400	VisDrone2021: Pedestrian, Person, Bicycle, Car, Van, Truck, Tricycle,	Pytorch	NVIDIA GeForce RTX 2080Ti	SGDM ($\beta = 0.9, \alpha = 0.01$) Penalty-L2 = 0.0001

	IBM + NMS	UAVDT	Resolución Training: 400x1000 to 800x1000 Resolución Testing: 600x1000	Awning-tricycle, Bus, Motor UAVDT: Vehicle (car, bus and truck)			Decay rate = 0.001 en 16th epoch y 0.0001 en 22th epoch #Épocas = 24
	Detection Stage: ResNetXt-101 + FPN	VisDrone2021-DET*	Resolución Training: 400x1400 to 1200x1400 Resolución Testing: 600x1000				
		UAVDT	Resolución Training: 400x800 to 800x800 Resolución Testing: 600x800				

Eagle-YOLO	CSPDarkNet53 + Bi-FPN(LKAM) + Head Large-size	VisDrone2021-DET*	Resolución Training/Testing: 1600x1600
			Clases: 10
				Pedestrian, Person, Bicycle, Car, Van, Truck, Tricycle, Awning-tricycle, Bus, Motor	Pytorch	NVIDIA GeForce RTX 3090 + CUDA v11.1 + CUDNN v8.2.1	Adam ($\alpha = 0.0003$)
							Batch size = 2
							#Épocas = 300
							Warmup = 2
							Function Loss: Eagle-IoU Loss

LKSNet + O-RCNN	Red Troncal: LK Selection (LSK module) + FFN Detector: O-RCNN	DOTAv1.0	Cantidad: 4000 #Instancias = 188282 Resolución: 124x124 #Clases = 15	Plane, Baseball Diamond, Storage Tank, Ship, Tennis court, Bridge, Ground Field Track, Harbor, Baseball Court, Small/Large Vehicle, Swimming Pool, Soccer Ball Field, Helicopter, Roundabout		8/1 NVIDIA GeForce RTX3090 for Training/ Testing	Adam ($\alpha = 0.0002$) Penalty-L2 = 0.05 Batch size = 8 #Épocas = 12

AFA-FPN + PHDA + Rotation Branch	Resnet-101 + AFA-FPN + PHDA + Rotation Branch	RSSO	Cantidad: 4000 60%/15%/25% Training/Validation/Testing Resolución: --- # Clases: 15	Plane, Baseball Diamond, Storage Tank, Ship, Tennis court, Bridge, Ground Field Track, Harbor, Baseball Court, Small/Large Vehicle, Swimming Pool, Soccer Ball Field, Helicopter, Roundabout	Pytorch	NVIDIA GeForce RTX 3090Ti	SGDM ($\beta = 0.9, \alpha = 0.01$) Batch Size = 32 #Iteraciones = 200000 Decay rate = 1/10 cada 50000 iteraciones Penalty-L2 = 0.0005 Loss Function: FL – Classification Loss CIoU – Regression Loss

*VisDrone2021-DET tiene emplea 6471 imágenes para entrenamiento, 548 para validación y 1580 para evaluación

Tabla 2. Cuadro comparativo de los resultados del rendimiento de las redes del estado del arte [6], [7], [8], [9], [10], [11].

RED	Dataset	Métricas para evaluación del rendimiento de la red							Resaltante	Limitaciones
		Precisión Promedio AP (%)			Precisión Media Promedio mAP (%)			Test Rate		
		$AP_{\square}$	$AP_{\square}$	$AP_{\square}$	@0.5	@0.75	@0.5: 0.05: 095			
EESRGAN + FASTER-RCNN	COWC	----	----	----	----	----	95.5	0.25s/img	Alta precisión al detectar objetos pequeños en imágenes de baja resolución.	Su precisión al detectar objetos medianos disminuye ligeramente, pero es mayor que los detectores convencionales como la Faster R-CNN.
	OGST	----	----	----	----	----	83.2			
VistrongerDet	VisDrone2021-DET	----	----	----	57.27	34.81	33.85	-----	Alta precisión al detector objetos diminutos, y comparado a los detectores convencionales Faster R-CNN y Cascade R-CNN, la red propuesta tiene una mejor precisión al detectar objetos pequeños, medianos y grandes. Asimismo, la red presenta una gran robustez al detectar objetos cuyo número de muestras en el	Su velocidad de detección se eleva por los módulos de mejora empleados en comparación con los detectores convencionales Faster R-CNN y Cascade R-CNN

									Dataset son muy bajos.	
Focus and Detect	VisDrone2021-DET	32.0	47.9	54.5	66.12	44.64	42.06	1.362s /img	Alta precisión al detectar objetos pequeños en imágenes densas y tomadas a gran escala. Asimismo, su precisión mejora ligeramente respecto a detectores convencionales al detectar objetos medianos.	Su precisión al detectar objetos grandes no es el mejor del estado del arte, pero es competitivo. Sin embargo, la velocidad de detección es baja.
	UAVDT	----	----	----	----	54.16	----	----		
Eagle-YOLO	VisDrone2021-DET	----	----	----	53.64	-- --	32.91	----	Alta precisión al detectar objetos pequeños en contextos complejos y detectar objetos de la categoría tail.	Su velocidad de detección es baja debido a las operaciones de punto flotante GPLOs en comparación a la red base YOLOv5x.
LKSNet + O-RCNN	DOTAv1.0	82.5	79.9	81.89	----	----	81.85	18.1 FPS	Alta precisión al detectar objetos pequeños y grandes en imágenes aéreas con entornos densos y cuyos objetos pueden obstruirse entre sí. Asimismo, su velocidad es mayor a los detectores convencionales debido a la disminución de número total de parámetros y la del número de operaciones de punto flotante FLOPs.	Su precisión al detectar objetos medianos disminuye ligeramente, pero sigue siendo competitivo con detectores convencionales.

AFA-FPN + PHDA + Rotation Branch	RSSO	78.21	87.77	80.43	82.04	69.70	59.70	24.3 FPS	Alta precisión al detectar objetos pequeños en diversos escenarios (simples, complejos, a gran escala, multiescala de objetivos, objetos pequeños densos). Asimismo, su precisión mejora ligeramente respecto a detectores convencionales al detectar medianos y grandes objetos.	Su velocidad de detección no es tan alta, pero es suficiente para aplicaciones prácticas de ingeniería.

1.3 Justificación

Actualmente, el estado del arte muestra investigaciones sobre desarrollos de algoritmos de aprendizaje profundo para la detección de objetos en imágenes áreas con un mAP@0.5 que oscilan entre el 53% y el 83% [6], [7], [8], [9], [10], [11]. Dichos algoritmos se caracterizan por enfocarse en el mejoramiento de la extracción de características de las imágenes de entrada empleando como base las redes neuronales convolucionales. Las implementaciones de estos algoritmos están orientadas a la inspección de zonas urbanas, campos de agricultura, torres y líneas de alta tensión, etc. Sin embrago, no existe documentación sobre inspección automatizada de fajas de servidumbre; por ello, en esta tesis se plantea diseñar un detector de árboles y edificaciones en imágenes aéreas basado en algoritmos de aprendizaje profundo supervisado como parte de la tarea de automatización de la inspección de fajas de servidumbre.

1.4 Objetivos

1.4.1 Generales

- Desarrollar un sistema de inspección automatizado basado en algoritmos de aprendizaje profundo supervisado para la detección de árboles y edificaciones en imágenes aéreas

1.4.2 Específicos

- Etiquetar una base de datos de imágenes aéreas para el entrenamiento, validación y evaluación de las redes *Faster R-CNN* y *YOLOv4*
- Implementar las redes *Faster R-CNN* y *YOLOv4* en Matlab y Pytorch
- Entrenar las redes *Faster R-CNN* y *YOLOv4* para la detección de objetos multiclase como árboles y/o edificaciones
- Evaluar las redes *Faster R-CNN* y *YOLOv4* con las imágenes de prueba y comparar los resultados con el estado del arte

Capítulo 2

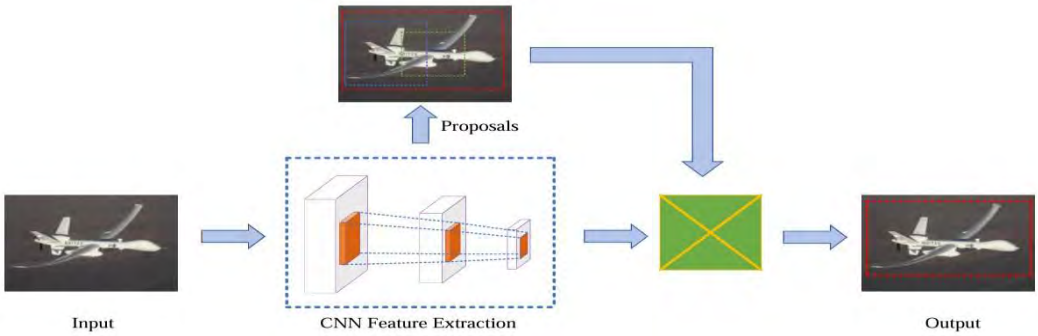
Las redes propuestas para el desarrollo de esta tesis están basadas en la *Faster R-CNN* (detector de dos etapas) y *YOLOv4* (detector de una sola etapa), las cuales presentan una arquitectura con tres fases principales: extracción de los mapas de características, la generación de las regiones de interés y, localización y clasificación de los objetos de interés. La primera sección del presente capítulo desarrolla, en primer lugar, las diferencias de los detectores de una y dos etapas; luego, una síntesis de cada fase de las redes resaltando las capas que la conforman; y, por último, explica el algoritmo usado para el proceso de entrenamiento y los ajustes de los hiperparámetros involucrados en dicho proceso. La segunda sección detalla de forma particular mediante un diagrama de bloques a las redes propuestas como solución, redes basadas en la *Faster R-CNN* y *YOLOv4*.

2.1 Marco teórico

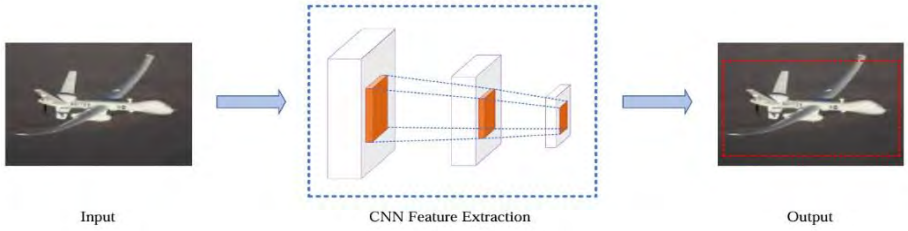
2.1.1 Detectores de una etapa vs Detectores de dos etapas

El estado del arte actual de los detectores de objetos basados en redes neuronales convolucionales se clasifica en dos categorías, las de una sola etapa basadas en tareas de regresión y las de dos etapas, en búsqueda de regiones candidatas. En cuanto a las redes de dos etapas, estas generan múltiples cajas candidatas para proporcionar predicciones aproximadas de las posiciones de los objetivos, que luego se refinan mediante el entrenamiento para lograr una alta precisión de detección y una localización exacta. Sin embargo, debido a la complejidad del algoritmo y el requisito de realizar la detección en dos pasos secuenciales, la velocidad de detección es relativamente más lenta [12]. Respecto a la primera, son redes que emplean la regresión

para la clasificación de objetos y la localización de las cajas candidatas, reduciendo de esta manera la complejidad de la estructura de la red y por ende la velocidad de detección, aunque sacrifica la precisión comparada a la red de dos etapas. A continuación, se observa en la Figura 8, los *frameworks* de estos dos tipos de detectores.



(a) *Framework* del detector de dos etapas



(b) *Framework* del detector de una etapa

Figura 8. *Frameworks* de los detectores de una y dos etapas [13].

2.1.2 Extracción de mapas de características

La fase de extracción de características está compuesta por filtros convolucionales o *kernels*, cada uno de los cuales activa ciertas características de las imágenes [14]. Los núcleos o *kernels* son matrices de pesos que se deslizan por la imagen de entrada que es una matriz de píxeles, por cada paso o *stride* que determina el intervalo de saltos del centro del *kernel*, se realiza un conjunto de operaciones matemáticas (producto escalar sumado un valor constante de *bias*) para producir un único valor de salida, y tras deslizarse el *kernel* por toda la imagen de entrada, se genera

una matriz convolucionada o mapa de características a la cual se le aplica la función de activación que reduce el problema de la gradiente de fuga presente en las redes neuronales profundas. El desvanecimiento de la gradiente o gradiente de fuga aparece en el entrenamiento de la red con el uso del algoritmo de aprendizaje supervisado llamado *backpropagation*; en este método, los pesos de los filtros de la red neuronal reciben una actualización proporcional a la derivada parcial de la función de pérdida con respecto a los pesos actuales del filtro en cada iteración del entrenamiento. El problema aparece en las redes neuronales profundas en las que la gradiente se vuelve muy pequeña, lo que evitará que los pesos se actualicen y por ende no se realice una predicción correcta de clase.

La arquitectura de una red convolucional profunda está compuesta por varias capas convolucionadas seguidas de funciones de activación, y a medida que se profundiza en la red, se reduce la dimensión de la matriz convolucionada o mapa de características (ancho y alto que se derivan en función del *stride*), pero aumenta en profundidad, la cual depende del número de filtros a aplicarse a dicho mapa. Cabe resaltar que varias redes presentan, inmediatamente después de cada función de activación, una capa *Pooling* que se encarga de reducir las dimensiones de los mapas de características resultantes con la finalidad de reducir el tiempo de procesamiento mediante la compresión de datos. Las capas más usadas son la *max-pooling* o la *average-pooling*, de los cuales la primera extrae los valores máximos de un conjunto de píxeles por donde se desliza la capa *Pooling*; y la segunda extrae los valores promedios. La arquitectura completa de esta etapa de extracción de los mapas de características se muestra en lado derecho de la Figura 9.

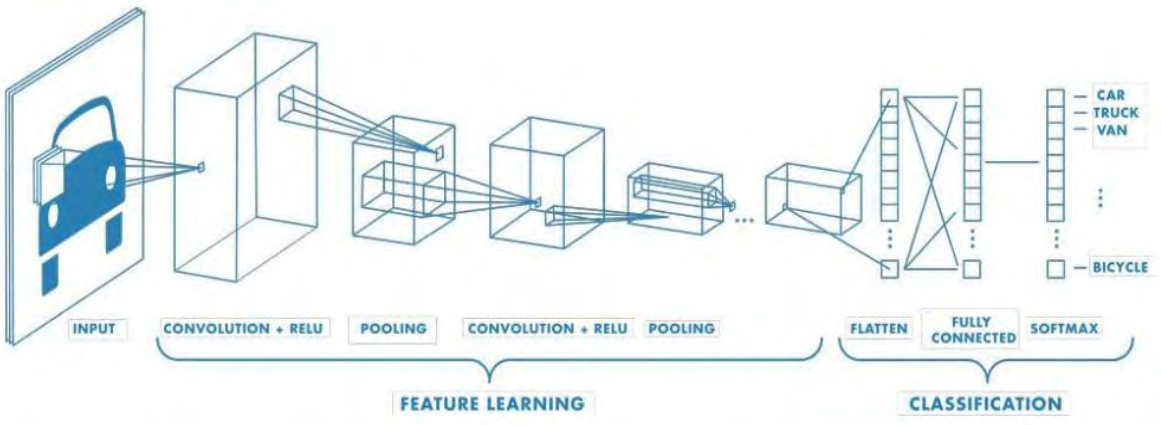


Figura 9. Arquitectura de una red neuronal convolucional típica [14].

En el Capítulo 3 donde se desarrolla el diseño de las redes propuestas como solución, para la fase de extracción de características, se plantea la red troncal *Resnet-50* para el detector de dos etapas, y la red *CPSDarknet5*3 para el detector de una sola atapa, ambas son descritas a continuación:

- ***Residual Network (Resnet)***

  Residual Network se trata de una red neuronal que propone introducir bloques residuales cuyas capas intermedias aprenden una función residual con referencia a la entrada del bloque, dicha función puede ser vista como un paso de refinamiento en el que se aprenda a cómo ajustar el mapa de características de entrada para obtener entidades de mayor calidad [15].

  Los bloques residuales para las *Resnet* de 50/101/152 capas incorporan el "cuello de botella" que consiste en aplicar dos convoluciones 1 x 1 para reducir/restaurar la dimensionalidad al comienzo/final del bloque respectivo [16] tal como se muestra en los recuadros punteados de la Figura 10, los cuales representan a los bloques residuales en mención.

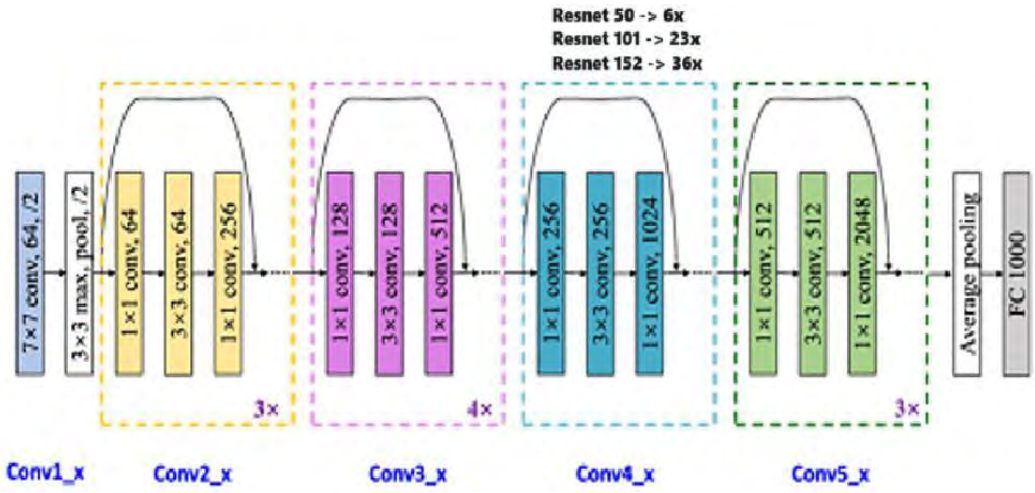


Figura 10. Arquitectura de *Resnet*-50/101/152.

- ***CPSDarknet53***

CSPDarknet53 es una red neuronal convolucional que usa la red *Darknet-53* como base, esta a su vez está compuesta por una *DenseNet*, y la *Cross Stage Partial Network (CPSNet)* como estrategia de partición de esta última de modo que el flujo de la gradiente puedo propagarse a través de diferentes rutas de la red y converja más rápido. La arquitectura de la *CSPDarknet53* lo constituye una capa convolucional seguido de una función de activación *Mish*; y 5 módulos *CSPBlock* continuos que, a medida que se profundiza la red, aumenta en capas apiladas de 1, 2, 8, 8 y 4 tal como se visualiza en el bloque de la izquierda en la Figura 11. Cada uno de estos *CSPBlock* divide la capa base en dos partes, una de ellas pasa por los bloques densos y la otra pasa directamente a concatenarse con la salida de la primera.

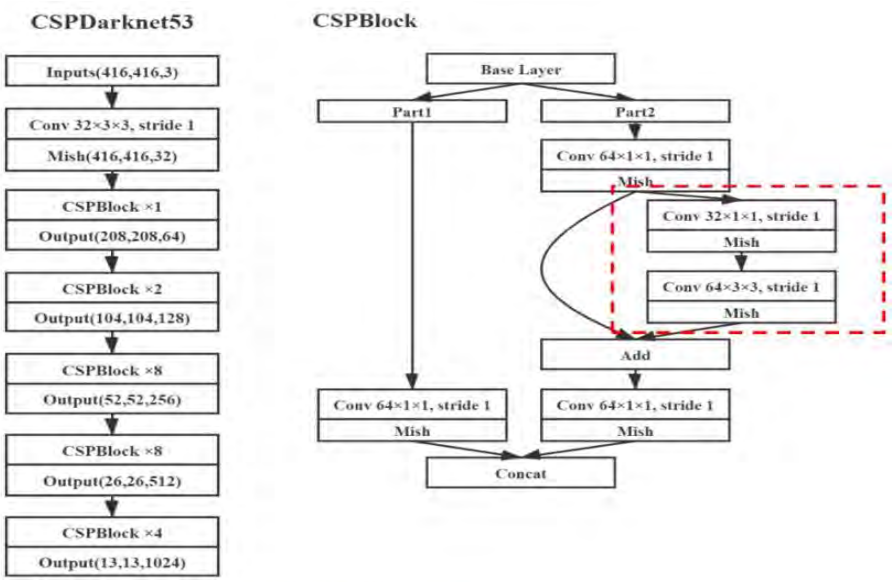


Figura 11. Arquitectura de la *CSPDarknet53* [17].

2.1.2.1 Mejoramiento de los mapas de características

Esta post-fase complementa a la primera, pues mediante un *Feautre Pyramid Network (FPN)* se mejora la fase de extracción de entidades creando una pirámide de mapas de características sólida en todos sus niveles, donde cada nivel puede ser empleado para detectar objetos a diferentes escalas. Para ello, combina características semánticamente fuertes de baja resolución con características semánticamente débiles de alta resolución a través de una vía de arriba hacia abajo y conexiones laterales tal como se observa en la Figura 12. La vía ascendente es la red convolucional habitual para la extracción de características, cuyo resultado es una jerarquía de mapas de características a varias escalas con un paso de submuestreo de 2. Por su lado, la vía descendente presenta una jerarquía de mapas de características de menor a mayor resolución mediante el sobremuestreo continuo que inicia con el último mapa de entidades obtenido de la red troncal. Cada mapa de la jerarquía descendente

se fusiona mediante una conexión lateral con el mapa de la misma resolución de la vía ascendente, el cual previamente para evitar un coste computacional se somete a una convolución de 1x1; y a la salida de cada fusión se le somete a una convolución de 3x3 para reducir el efecto *aliasing* del muestreo ascendente [18].

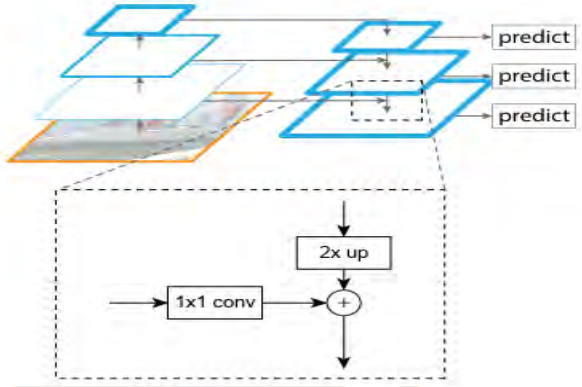


Figura 12. *Framework* de la *Feature Pyramid Network (FPN)* [18].

En el Capítulo 3 donde se desarrolla el diseño de las redes propuestas como solución se plantea las redes *Path Aggregation Network (PAN)* y *Spatial Pyramid Pooling Network (SPP-Net)* para el mejoramiento de características del detector de una sola etapa, ambas son descritas a continuación:

- ***Spatial Pyramid Pooling Network (SPP-Net)***

  La red *SPPNet* permite aumentar el campo receptivo de la red troncal *CSPDarknet53*, para ello toma los mapas de características de la salida de esta útlima y emplea ventanas de agrupación de diversos tamaños y *strides* variables de modo que, en la salida, a diferencia de la *SPP* original, se generen mapas de características del mismo tamaño y número de canales que su entrada. A continuación, todos estos mapas generados se concatenan con los mapas de entrada de la red *SPP* como

se observa en la Figura 13.

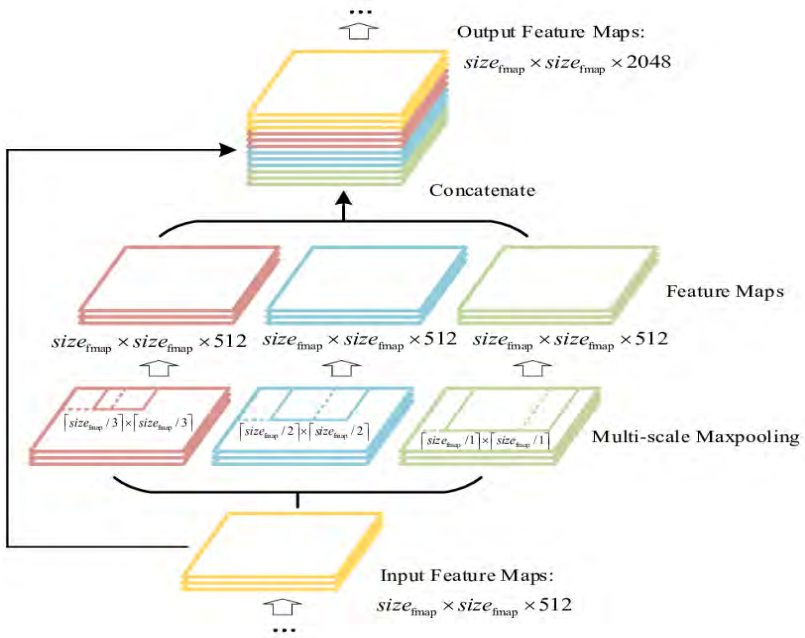


Figura 13. *Framework* de la *SPP* modificada [19].

- ***Path Aggregation Network (PANet) modificado***

La red *PANet* presenta la misma arquitectura de la *Feature Pyramid Network (FPN)* con la diferencia que esta adiciona una ruta ascendente más con la finalidad de facilitar el flujo de información y así detectar objetos de mayor escala ubicados en las capas superiores de manera rápida, entonces la ruta ascendente adicionada es una conexión de acceso directo que consta de menos de 10 capas en comparación con las de más de 100 capas que llegan a constituir la ruta ascendente de la *FPN* [20]. A continuación, los mapas generados de la ruta ascendente adicionada se les aplica un *ROI Align* para extraer mapas de entidades de un tamaño fijo y luego fusionarlos adaptativamente con la operación máxima de modo que la red extraiga información de

todos los niveles y presente mejores resultados. Cabe recalcar que la versión original de la *PANet* fue modificada de modo que las conexiones laterales se dan mediante una concatenación en remplazo de la adición, dicha conexión lateral ascendente se muestra entre la parte “a” y “b” de la Figura 14.

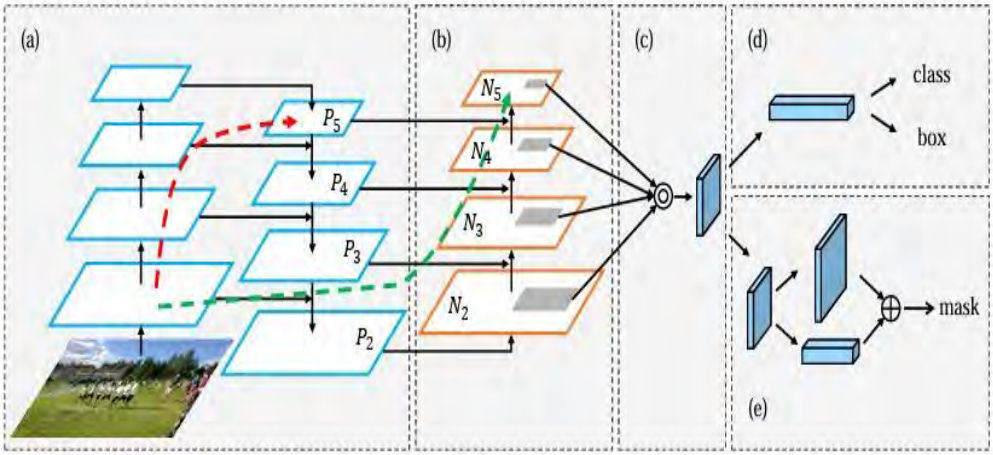


Figura 14. *Framework* de la *PANet* modificada [20].

2.1.3 Generación de Propuestas de Cuadros Delimitadores

2.1.3.1 Detector de dos etapas - Generación de Regiones de Interés (RoI)

Esta fase en un detector de dos etapas está compuesta por una *Region Proposal Network (RPN)*, la cual generará las regiones de interés que contienen objetos de la imagen de entrada. Para ello, la red define un conjunto de “k” cajas de anclaje por cada ancla o ubicación de *pixel* (centro de la caja) en la imagen de entrada teniendo en cuenta “n” parámetros de escala (tamaño de la imagen) y “n” relaciones de aspecto (relación entre ancho y alto de la imagen), pues de este modo se generan cajas de acuerdo con las dimensiones de la imagen de entrada. La red inicia deslizando una ventana 3x3 con 512

unidades sobre los mapas de características de la última capa de la red troncal; luego se emplea dos capas convolucionales de 1x1 para verifica si los “k” cajas de anclaje contienen realmente un objeto o no y se redefinan las coordenadas de las anclas para generar cuadros delimitadores candidatos como regiones de interés. Una de las capas convolucionales es la capa de clasificación que genera 2k puntajes y determina si la caja dada es de primer plano (tiene un objeto o parte del objeto dentro de ella) o de fondo (no tiene un objeto dentro de ella), y la segunda es la capa de regresión de coeficientes para cuadros con la finalidad de ajustar los objetos en las cajas de primer plano, esta capa genera 4k salidas con las coordenadas de las cajas propuestas (centro x e y, alto y ancho). Y es así, finalmente, de ese modo que se genera las regiones de interés llamadas *ROIs.* La arquitectura de la *RPN* descrita anteriormente se puede observar en la Figura 15.

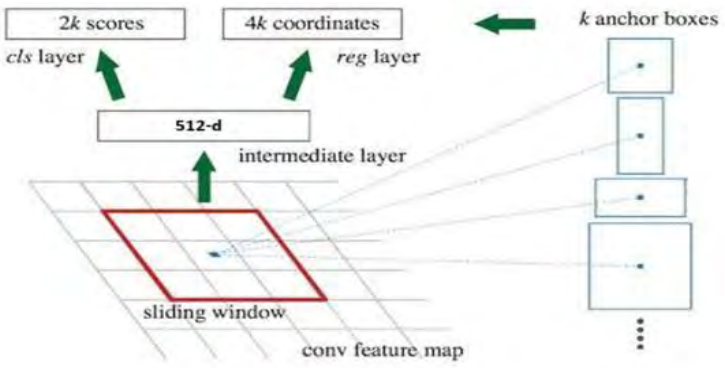


Figura 15. Arquitectura de la *Region Proposal Network (RPN)* [21].

2.1.3.2 Detector de una etapa- Propuestas de cuadros delimitadores

Dado que el detector de una etapa se encarga en una sola red de la extracción de características, la regresión de las cajas y la clasificación de los objetos, en la salida de la red se genera una matriz de SxS cuadrículas, donde

"S" es el tamaño del ancho y largo del último mapa de características. Cada cuadrícula predice "k" cajas delimitadoras, y las representa con un vector que consta de 5 componentes, 4 de ellos son las coordenadas del *bounding box* (x,y,w,h) y una, el *confidence object* que indica la probabilidad que el *bounding box* contenga un objeto ; cabe resaltar que las cuadrículas que contienen el centro de los objetos presentes en la imagen son las responsables de detectar dichos objetos. Asimismo, adyacente a cada vector de cuadrícula, está el vector de probabilidad de clases que consta de tantos elementos como objetos de interés a detectar se quiere. Los vectores en mención se pueden observar en el conjunto de cubos rosados en la Figura 16.

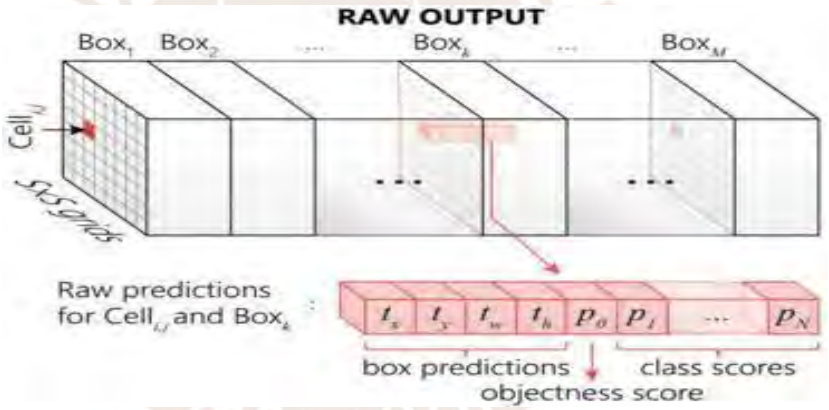


Figura 16. Salida de la red *YOLO* [21].

2.1.4 Localización y clasificación de objetos de interés

Para la red de dos etapas, la etapa de localización y clasificación de los objetos de interés se basa en una *Fully Connected (FC) Layer,* la cual es una capa que multiplica la entrada recibida por una matriz de pesos, y a continuación, agrega un vector de sesgo [22], siendo la entrada recibida, la salida aplanada en un vector unidimensional de la capa *ROI Pooling*, la cual busca redimensionar a un tamaño fijo

las propuestas de región proyectadas estratégicamente en los mapas de características multinivel tras la salida de la *Feature Pyramid Network (FPN)*. A la salida de la capa *FC*, se generan dos vectores, uno para la tarea de clasificación y otro para la de regresión. Por su parte, la red de una etapa realiza la tarea de localización y clasificación directamente en su matriz tridimensional compuesta de SxS celdas con una profundidad de (5+C) x B, donde "C" es el número de clases a identificar y "B" el número de *bounding boxes* para cada celda de la matriz. En la salida de ambos detectores se utiliza dos funciones; la primera, la función de activación *softmax* para obtener la probabilidad de que la entrada esté en una clase particular; la segunda, la función *Box regression* para extraer los coeficientes de desplazamientos de los cuadros predichos respecto a los reales.

2.1.4.1 Función de activación *Softmax*

La función *Softmax* convierte un vector real en un vector de probabilidades categóricas cuyos valores de salida están en el rango de 0 a 1 y la suma de todas ellas es 1. Esta función clasifica el objeto de interés en la etiqueta de clase donde presenta el valor máximo de probabilidad.

2.1.4.2 Función *Bounding Box Regression (BBR)*

La función *Bounding Box Regression* predice un valor de número real continuo a partir de un vector de características dado [23]. Dado que la localización de las cajas se especifican con un conjunto de coordenadas en la parte superior izquierda (x,y) y un ancho (w) y alto (h) en píxeles, lo que busca la función *BBR* es calcular el error entre las coordenadas predichas respecto a la *ground truth* que viene a ser la caja etiquetada como verdadera con la

finalidad de encajar el objeto en la caja delimitadora propuesta.

2.1.5 Proceso de entrenamiento de la red

Para llevar a cabo el proceso de entrenamiento de la red y de esta forma, la red aprenda a realizar de la manera más óptima la tarea para que fue destinada; primero, se debe separar el conjunto de datos de imágenes en datos de entrenamiento, validación y evaluación, de las cuales; el primero, permite el proceso de aprendizaje del sistema de forma iterativa; el segundo, averigua qué hiperparámetros se necesitan variar en el sistema para disminuir el error de generalización; y el tercero, evalúa la red una vez finalizado y optimizado el proceso de aprendizaje. En el caso de la tarea de detección de objetos de dos etapas, se integra dos redes; una red basada en propuesta de regiones para generar cuadros candidatos que delimitan con rectángulos los objetos presentes en la imagen; y una red de clasificación de clase y localización de dichos objetos; por tanto, se tendrá dos redes por entrenar de manera independiente para luego fusionarlas y reentrenarlas ajustando los pesos de las capas exclusivas de cada red, cabe recalcar que ambas comparten la red troncal para la extracción de los mapas de características. El entrenamiento en ambas redes se realiza en una sola etapa con una función de pérdida total multitarea que integra la pérdida de clasificación y la de regresión, y además solo para el caso de la red de una sola etapa, la pérdida de confidencia.

La técnica más usada para el proceso de entrenamiento es la Retropropagación o *Backpropagation*, el entrenamiento con este algoritmo consta de dos fases; la primera fase es una propagación desde los inputs hasta generar las salidas de la red [24], en esta fase los pesos de los *kernels* se inicializan en cualquier valor, estos se convolucionan con sus respectivas matrices de entrada y pasan por una función de activación además

de capas *Pooling*. La segunda fase compara las salidas con las etiquetas *Ground Truth* y se calcula el error mediante una función de coste o pérdida, llamado como este último en el caso de una clasificación multietiqueta. Existen diversas funciones de coste independientes para la clasificación y localización de objetos empleadas en la evaluación de las soluciones candidatas, y justamente la técnica de retropropagación busca minimizar el valor de dichas funciones ajustando los pesos de cada capa de la red y; para ello, se usa métodos de optimización basados en diferentes variantes del método de la gradiente descendente. Este último junto a los métodos de regularización y la tasa de aprendizaje forman parte de los hiperparámetros de una red que se ajustan para mejorar su desempeño. En cuanto a los métodos de regularización, estos se emplean para evitar el sobreajuste, el cual originaría que la red pierda la capacidad de generalización al clasificar debido a una especialización de los pesos a los datos de entrenamiento [24]. A continuación, se detallará sobre las funciones de coste, los modelos de optimalización y regularización más usados en el entrenamiento de redes neuronales.

2.1.5.1 Funciones de Coste

2.1.5.1.1. Funciones de Coste de Clasificación

- ***Cross Entropy Loss***

  Esta función de coste busca penalizar las predicciones erróneas más que premiar las correctas. La penalización es de naturaleza logarítmica, lo que produce una puntuación grande para las diferencias cercanas a 1 y una puntuación pequeña para las diferencias que tienden a 0. La entropía cruzada se define en la

Ecuación 1, donde y_i es la etiqueta verdadera y p_i es la probabilidad softmax para la clase i.

$$CE = -\sum_{i=1}^{C} y_i \log(p_i) \quad (1)$$

- ***Focal Loss***

En esta función se añade un factor de modulación $(1 - p_t)^\gamma$ y un parametro de ponderación α_t a la función de *cross entropy* para penalizar las predicciones incorrectas sin preocuparse demasiado por las muestras bien clasificadas y tomando en cuenta la frecuencia de aparición de ejemplos positivos y negativos. Por tanto, esta función es una pérdida de entropía cruzada escalada dinámicamente, pues pondera la contribución de cada muestra a la pérdida en función del error de clasificación. La *focal loss* se muestra en la Ecuación 2.

$$FL = -\alpha_t (1 - p_t)^\gamma \log(p_t), \gamma \in [0, 5] \quad (2)$$

2.1.5.1.2. Funciones de Coste de Regresión

- ***Squared Loss***

También conocida como L_2 representa la suma de los cuadrados de las diferencias entre los valores reales $y^{(i)}$ y los predichos. $h(x^{(i)})$ correspondientes a los datos $x^{(i)}$. La combinación de todos los valores de pérdida L_2 del *dataset* se llama *Mean Squared Error (MSE)* [25] y se define como en la Ecuación 3.

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad (3)$$

- ***Absolute Loss***

También conocida como $\mathcal{L}_1$, representa el valor absoluto de la resta entre los valores reales y_i y los predichos $\hat{y}_i$ correspondientes a los datos x_i. La combinación de todos los valores de pérdida $\mathcal{L}_1$ del *dataset* se llama *Mean Absolute Error (MAE)* [25] y se define como en la Ecuación 4.

$$\mathcal{L}(y, \hat{y}) = -\sum_{i=1}^{n} |y_i - \hat{y}_i| \quad (4)$$

- ***Smooth*** $\mathcal{L}_1$

Es una combinación de $\mathcal{L}_2$ y $\mathcal{L}_1$, de modo que cuando el error se hace mínimo, se use el *MSE* y cuando sea grande, el *MAE*. Lo pequeño que tiene que ser el error absoluto para que se vuelva cuadrático depende de un hiperparámetro δ [25], según se muestra en la Ecuación 5.

$$\mathcal{L}_{\delta}(y, \hat{y}) = \begin{cases} \frac{1}{2}(y - \hat{y})^2, & |y - \hat{y}| \leq \delta \\ \delta |y - \hat{y}| - \frac{1}{2}\delta^2, & \text{en otro caso} \end{cases} \quad (5)$$

- ***IoU Loss* y sus variantes**

El cálculo de pérdida *IoU* es en base al área de superposición entre el cuadro real y el predicho, por tanto, solo funcionan para cuadros superpuestos. Para mejorar la brecha que presenta *IoU Loss*, se plantea la *Generalized IoU (GIoU) Loss* que se basa en la anterior,

pero busca aumentar el tamaño de la caja predicha y moverla lentamente hacia la caja real para que ambas se superpongan. Sin embargo, este traslado puede llevar varias iteraciones; por ello, se plantea la *DIstance IoU (DIoU) Loss* que calcula la distancia normalizada entre el punto central del cuadro real y predicho, de modo que permite acelarar el movimiento planteado en *GIoU* y con una regresión más precisa. Todas las funciones de pérdida mencionadas se complementan en una sola función llamada *Complete IoU (CIoU)* que utiliza tres factores geométricos: el área de superposición, la distancia entre los puntos centrales y la relación de aspecto [26]. Los resultados de la *CIoU* superan a la *GIoU* en términos de velocidad de convergencia y precisión de regresión como se observa en la Figura 17; y está dada bajo la Ecuación 6 (1-2).

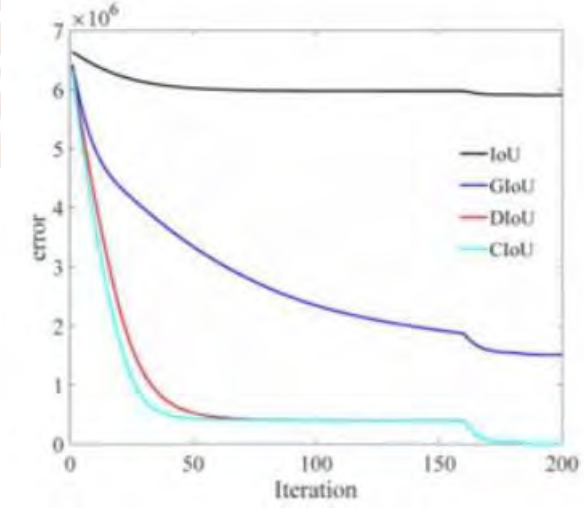


Figura 17. Curvas de errores de regresión para diferentes funciones de pérdidas *IoU* [26].

$$\theta_{l+1} = \theta_l + \Delta\theta_{l+1} + \varepsilon(\theta, \quad (6.1)$$

$$\Delta\theta_{l+1} = \alpha - I\frac{\partial}{\partial} + \varepsilon(\theta, \quad (6.2)$$

2.1.5.2 Optimizadores

- ***Stochastic Gradient Descent with Momentum (SGDM)***

El algoritmo de gradiente de descenso con momento restringe la oscilación del descenso de la gradiente en una dirección para que este converja más rápido; para ello, en cada paso de actualización de pesos, al valor del gradiente actual se suma el promedio ponderado exponencial de las gradientes previas multiplicada por

un factor de impulso (β), el cual oscila entre 0 y 1, siendo 0.9 el valor más usado. El optimizador *SGDM* se define bajo la Ecuación 7 (1-2).

$$\theta_\theta = \theta * \theta_{\theta-\theta} + (\theta - \theta) * \quad (7.1)$$

$$\theta_\theta = \theta_{\theta-\theta} - \theta * \theta_\theta \quad (7.2)$$

- ***Root Mean Square Propagation (RMSProp)***

Optimizador que utiliza una tasa de aprendizaje adaptativa de modo que esta cambia con el tiempo; asimismo, emplea una media móvil de gradientes al cuadrado para normalizar el degradado y está dado bajo la Ecuación 8.1 y 8.2. Dicha normalización equilibra el tamaño del paso, disminuyéndolo para gradientes grandes para evitar la explosión y aumentándolo para gradientes

pequeños para evitar la desaparición [27]. El optimizador *RMSProp* se define bajo la Ecuación 8 (1-2).

$$\text{(8.1)}$$

$$\text{(8.2)}$$

- ***Adaptive Moment Estimation (Adam)***

La optimización Adam combina las ventajas de los algoritmos *RMSProp* y *SGD Momentum*, entonces el algoritmo de este optimizador calcula las tasas de aprendizaje adaptativo individuales para cada peso a partir de las estimaciones de los primeros (la media, β_1) y segundos (varianza no centrada, β_2) momentos de los gradientes. El valor inicial de las medias móviles y los valores de β_1 y β_2 cercanos a 1 generan un sesgo en las estimaciones de lo momentos hacia cero; para ello, se calcula las estimaciones sesgadas.

Los valores iniciales recomendados para configurar el algoritmo *Adam* son $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ y $\varepsilon = 10^{-8}$, esta última variable es usada para evitar cualquier división por cero en la implementación. El optimizador *Adam* se define bajos la Ecuación 9 (1-4).

$$\text{(9.1)}$$

$$\text{(9.2)}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \qquad \hat{s}_t = \frac{s_t}{1-\beta_2^t} \tag{9.3}$$

$$\theta_t = \theta_{t-1} - \frac{\eta}{\sqrt{\hat{s}_t + \varepsilon}} * \hat{v}_t \tag{9.4}$$

Desde el punto de vista práctico como se observa en la Figura 18, a menudo, al principio del proceso de entrenamiento, *SGD* va en la dirección equivocada, mientras que *RMSProp* enfila hacia la dirección correcta. Sin embargo, este es sensible al ruido, por lo que generalmente rebota significativamente alrededor del punto óptimo cuando está cerca de un mínimo local. Siendo así, *Adam* una mejor opción al presentar una convergencia rápida y menos ruidosa.

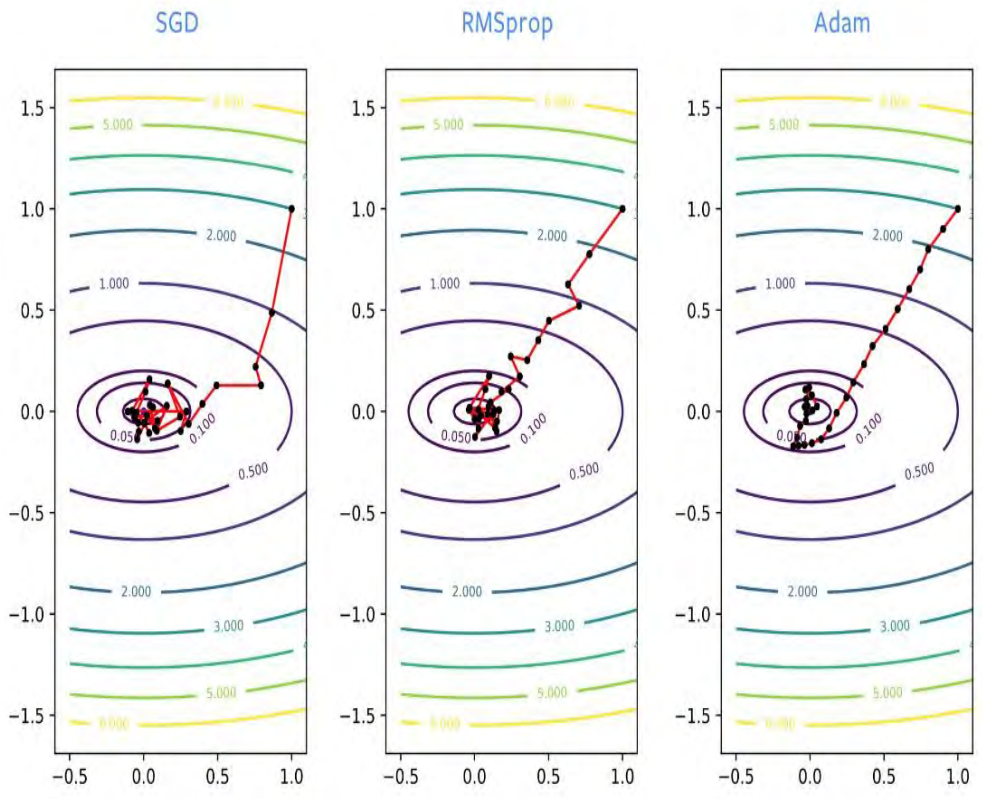


Figura 18. *SGD vs RMSProp vs Adam* [28].

2.1.5.3 Regularización

- **Regularización de peso**

La regularización de peso impone restricciones a la capacidad del modelo mediante la penalización de pesos de los parámetros, obligándolos a tomar valores pequeños. Se tienen dos métodos cuya diferencia radica en conceptos matemáticos, la

regularización L1 y L2, cuyo factor de regularización será λ.

Regularización L1: La penalización L1 corresponde a la suma de los valores absolutos de los parámetros del modelo, llegando a generar modelos dispersos donde algunos parámetros pueden convertirse en cero y eliminarse. Esta regularización se define bajo la Ecuación 10 (1-2).

$$L_{\square} = \mathbf{Error}(yy) + \lambda * ||\square||_{\square} \quad \textbf{(10.1)}$$

$$||\square||_{\square} = \sum |\square_i| \quad \textbf{(10.2)}$$

Regularización L2: La penalización agregada corresponde a la suma de los cuadrados de los parámetros del modelo, sin embargo, L2 no es capaz de generar modelos dispersos, todos los coeficientes se reducen por el mismo factor, pero no se elimina ninguno. Esta regularización se define bajo la Ecuación 11 (1-2).

$$L_{\square} = \mathbf{Error}(yy) + \lambda * ||\square||_{\square} \quad \textbf{(11.1)}$$

$$||\square||_{\square} = \sum (\square_i)^{\square} \quad \textbf{(11.2)}$$

- **Aumento de datos**

El aumento de datos consiste aumentar la variedad de inputs que recibe la red, y una forma de lograrlo es usando transformaciones al conjunto de datos de entrenamiento tales como rotación, reflexión, traslación, etc.

- ***Dropout***

Este método permite desactivar cierto número de neuronas de la red de forma aleatoria en cada iteración de esta durante el entrenamiento, lo cual fuerza a la red a aprender características más robustas de los *imputs*. *Dropout* trabaja con un parámetro de probabilidad de que las neuronas se queden desactivadas, el cual toma valores de 0 a 1; donde los valores cercanos a cero indican menos neuronas desactivas y los valores cercanos a uno indican más neuronas desactivadas.

- **Normalización por lotes**

La técnica de normalización por lotes consiste en insertar una capa de normalización llamada *Batch Norm (BN)* entre dos capas ocultas con la finalidad de normalizar las salidas de la capa previa antes de pasarlas como entrada a la capa posterior. El proceso de normalización, tal como se muestra en la Figura 19, implica calcular para cada vector de activación, la media y varianza de todos los valores del *batch*, y luego con dichos resultados normalizar cada salida del vector en mención, restándola por la

media y a la sustracción dividiéndola por la varianza. Para predecir los mejores resultados, se requiere desplazar (media diferente) y reescalar (varianza diferente) los valores de la capa de activación normalizada, para ello se multiplica dichos valores por un factor gamma y al producto se le añade un factor beta. Ambos elementos son parámetros entrenables. Asimismo, para la etapa de testeo se debe considerar que al tener una sola muestra que ingresa a la red, se debe emplear la media móvil exponencial de la media y la varianza guardadas de la etapa de entrenamiento.

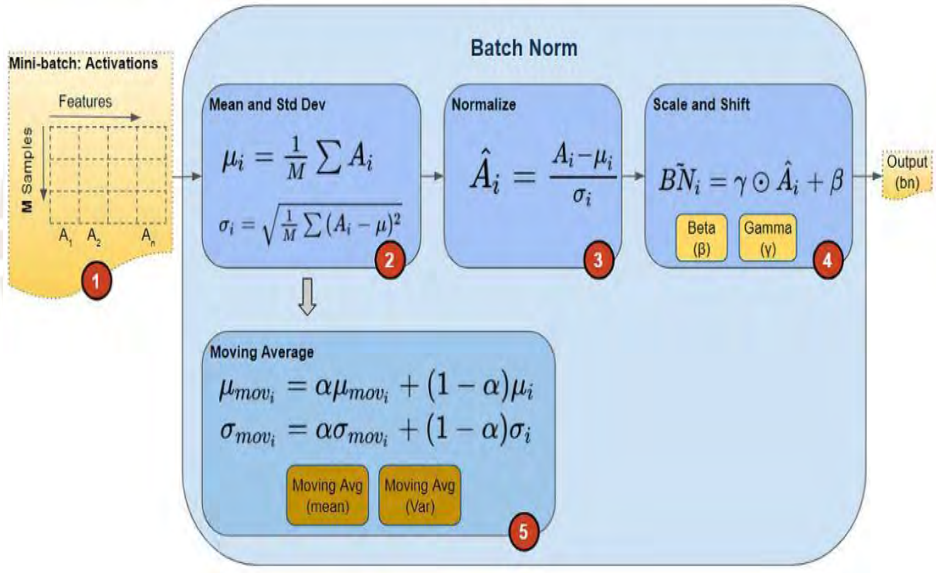


Figura 19. Cálculos realizados en la capa *Batch Norm (BN)* [29].

2.2 Modelo de solución

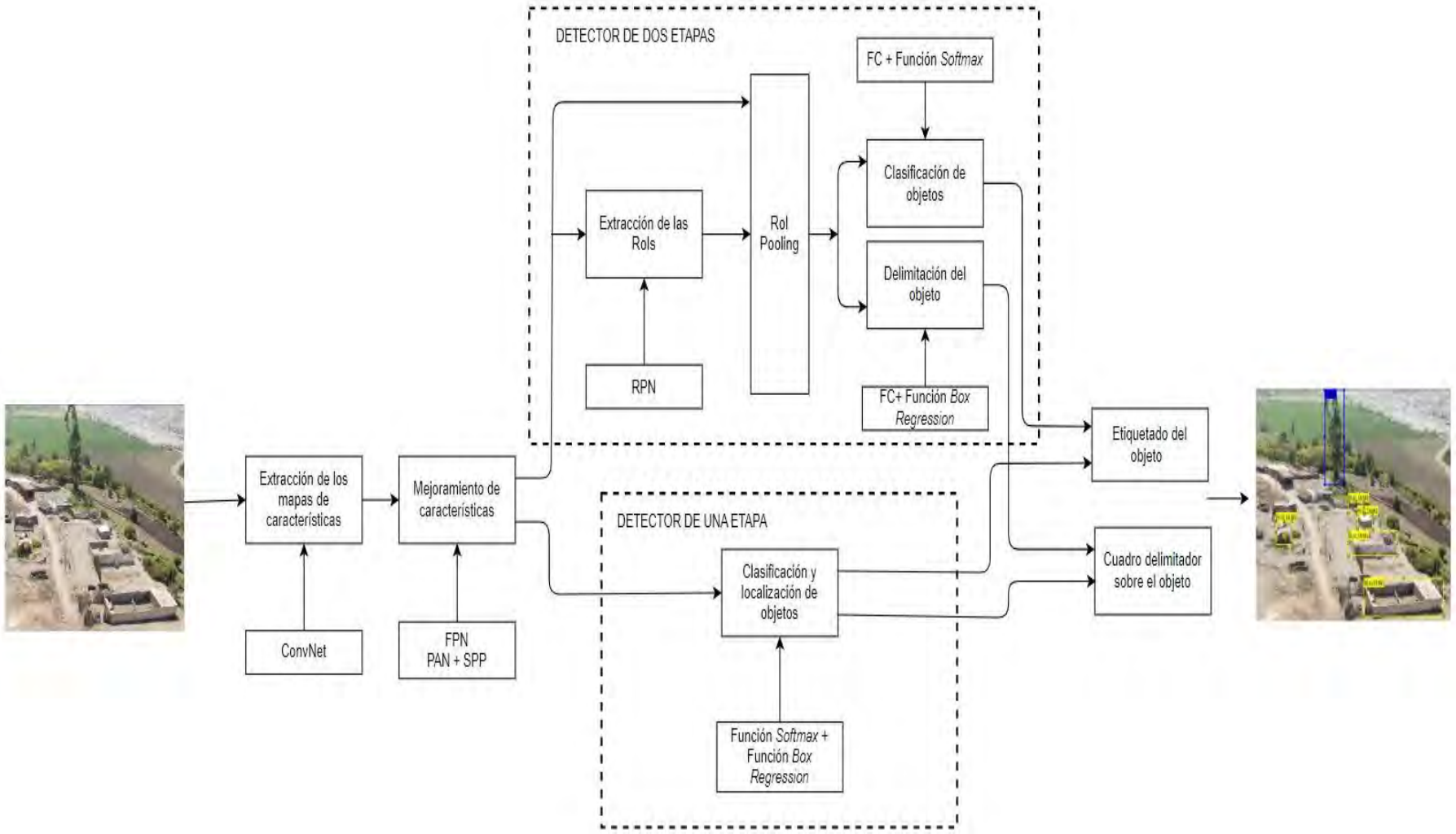


Figura 20. Diagrama de bloques de la arquitectura del modelo de solución (Elaboración propia).

Capítulo 3

El presente capítulo tiene como objetivo el diseño de las redes propuestas para la detección de árboles y edificaciones en imágenes aéreas; para ello, primero se detalla los requerimientos y consideraciones necesarias tanto en *software* como en *hardware*, luego se describe cada etapa del diseño de las redes, comenzando con la etapa de recolección y etiquetado; luego, la etapa de entrenamiento y; por último, la etapa de evaluación. En la segunda etapa se abordará los hiperparámetros a ajustar para la optimización de la red; y en la tercera, se detallará las métricas de evaluación del desempeño de la red.

3.1 Requerimientos y consideraciones

Para la implementación, entrenamiento y evaluación de la red propuesta, se emplea tanto el *software* Matlab como el *framework* Pytorch haciendo uso de sus librerías de procesamiento de imágenes, visión por computadora y computación paralela para habilitar el uso de GPUs. Asimismo, se emplea dos conjuntos de imágenes, el primero llamado "DataSetImages" de 1560 imágenes, las cuales fueron tomadas por drones pertenecientes a la empresa StatKraft Perú, a la universidad PUCP y también extraídas de un video aéreo de *YouTube* tomadas sobre una casa de campo; el segundo, llamado "DatasetVariados" consta de 350 imágenes extraídas en un mínimo porcentaje del primer *dataset* y en su mayoría de videos aéreos de *YouTube*. Por el lado del hardware, el *software* Matlab corre en una computadora con GPU modelo NVIDIA RTX 3060 de 12GB, memoria de video mínima para trabajar con una gran cantidad de datos durante el entrenamiento. Se prefiere un GPU marca NVIDIA debido a sus librerías CUDA, una plataforma de computación paralela mucho más potente que OpenCL, librería utilizada por la marca AMD [30]. Y respecto al *framework* Pytorch este corre en el servicio gratuito en la nube *Google Colab*, el cual permite trabajar

con una GPU modelo Tesla T4 de 15GB.

3.2 Etapas de la propuesta de solución

La propuesta de solución planteada consta de 3 etapas tal como se muestra en el esquema de la Fig. 21: etapa de recolección y etiquetado, etapa de entrenamiento y etapa de evaluación.

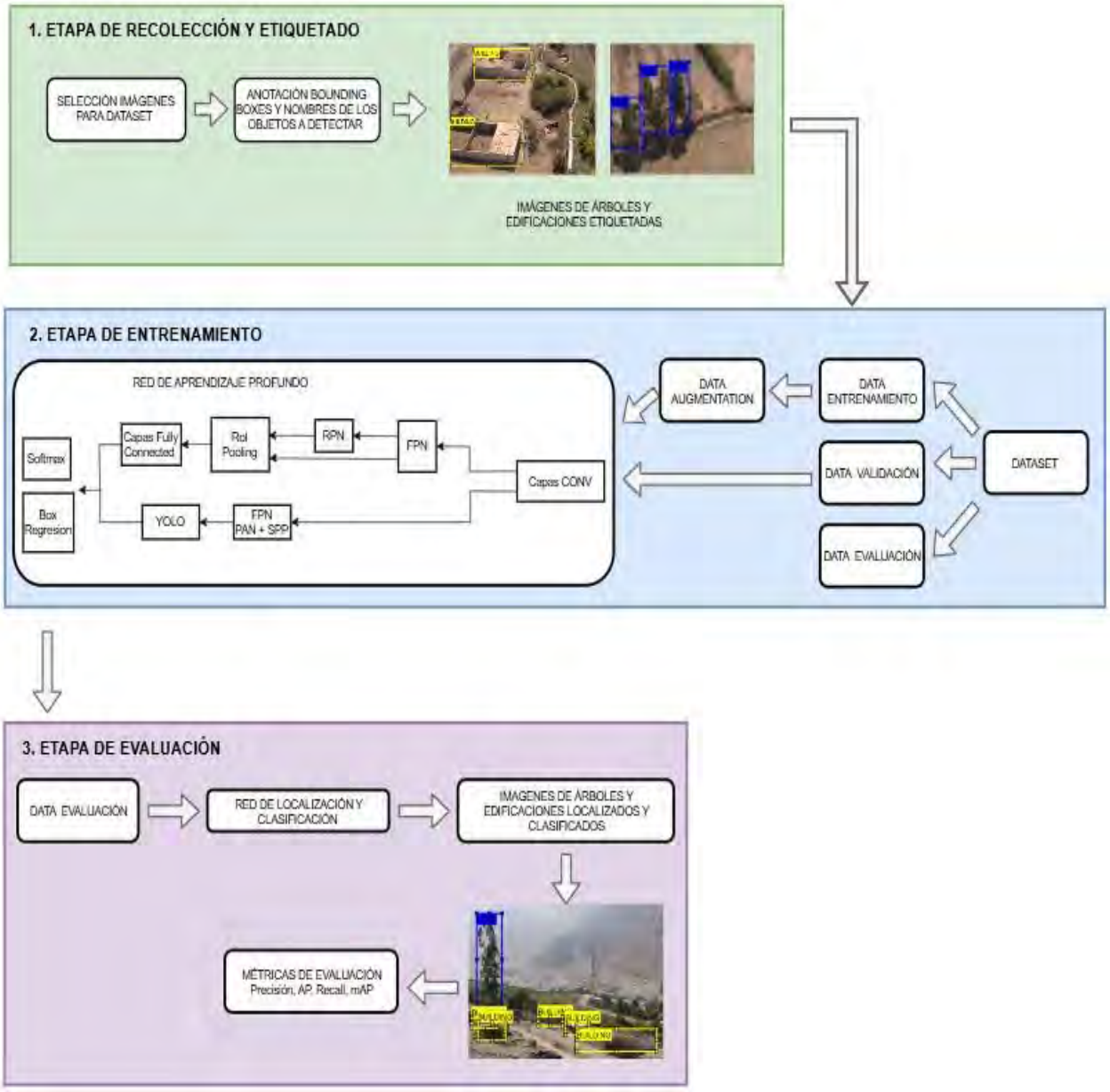


Figura 21. Flujo de etapas de la solución propuesta. (Elaboración propia)

3.2.1 Etapa de recolección y etiquetado

Se creó dos bases de datos “DataSetImages” y “DatasetVariados” debido a que se requería un conjunto de imágenes de árboles y edificaciones en fajas de servidumbre, las cuales no se pudieron encontrar en *datasets* de imágenes aéreas ya existentes y de libre acceso.

Para la creación de las bases de datos, primero se seleccionó las imágenes proporcionadas por la empresa Statkraft, la universidad PUCP y las extraídas de los videos aéreos en *YouTube*, con la finalidad de recopilar todas aquellas donde se note claramente algún(os) árboles y/o edificación(es); luego, se procedió con el etiquetado de las imágenes seleccionadas para ambos *datasets* por medio de la herramienta *Image Labeler*, una *app* de Matlab que permite realizar anotaciones para crear los *bounding boxes*, que son los cuadros delimitadores del objeto en interés, y definir la clase de los objetos que se desean etiquetar, los cuales en caso del desarrollo de la tesis son etiquetados como “TREE” y “BUILDING”. La base de datos “DataSetImages” tiene un total de 1560 imágenes de tres resoluciones: 5280 x 2970 pixeles (Statkraft), 1920 x 1088 pixeles (PUCP), 1280 x 720 (*YouTube*); y “DatasetVariados”, un total de 350 imágenes de diversas resoluciones cuyo ancho varía entre los 640 y 5280 píxeles y el alto, entre los 368 y 2970 píxeles. Ambos *datasets* son divididos cada uno en tres partes: el 80% del total está destinado para el entrenamiento, el 10%, para la validación y el 10% restante, para la evaluación. Respecto a los objetos presentes en las imágenes, estos varían en tamaño según la altura y el ángulo en que se realizó la toma, por ello, la resolución de algunos es de pocos píxeles en ciertas imágenes, mientras en otras ocupan gran parte de estas. Asímismo, hay imágenes con tomas de largas de escenas donde abundan los objetos pequeños y densos, llegando incluso estos mismos a

obstruirse entre sí. Además, cabe recalcar que la diferencia entre ambos *datasets* radica en la diversidad de contextos en los que fueron tomadas las imágenes, siendo "DatasetVariados" el que contiene la mayor diversidad de tomas en zonas rurales y urbanas.

3.2.2 Etapa de entrenamiento

Tras el etiquetado de las imágenes de los *datasets* creados, se procede a dividirlas a cada uno por separado en tres grupos: entrenamiento, validación y evaluación. Sin embargo, como el número total de imágenes solo asciende a 1560 para el primero y 350 para el segundo, es necesario emplear la técnica de aumento de datos mediante un conjunto de opciones de preprocesamiento para el aumento de imágenes durante cada época del entrenamiento, tales como la reflexión, rotación, recorte y cambio de tonalidad. Luego de la división de las imágenes en cada *dataset*, se procede a entrenar las redes propuestas, de las cuales el detector de dos etapas emplea como red troncal para la extracción de características a la *Resnet-50* y el detector de una sola etapa emplea la *CPSDarknet53*; asimismo, para el mejoramiento de características, el primero emplea *Feature Pyramid Network (FPN)*; y el segundo, la *Path Aggregation Network (PAN)* y *Spatial Pyramid Pooling Network (SPP-Net)* en su arquitectura. En esta etapa de entrenamiento de las redes propuestas se ajustan sus hiperparámetros para optimizarlas y así obtener una mejor precisión en la detección de los objetos de interés.

3.2.2.1 Ajuste de hiperparámetros

Para el entrenamiento de la red se configura sus hiperparámetros de modo que se optimice el proceso de entrenamiento de este con la finalidad que se obtenga su mejor rendimiento, siendo estos hiperparámetros [31] los

siguientes:

- **Tamaño de lote (*batch_size*):** Número de muestras que están presentes en una pasada tipo *Forward* y *Backward* en la etapa de entrenamiento
- **Iteraciones (*Iterations*):** Número de pasadas o *batches* necesarios para completar una época
- **Épocas (*Epoch*):** Es un tipo de *Forward* y *Backward* para todas las muestras de entrenamiento
- **Tasa de aprendizaje (*learning rate*):** Controla cuánto se está ajustando los pesos o parámetros de la red con respecto a la gradiente de pérdida
- **Optimizador:** Algoritmo que ajusta los pesos en la retropropagación para minimizar la función de coste, prácticamente casi todas pertenecen a la familia denominada Gradiente Descendiente Estocástico (SGD), entre las más importantes tenemos a *Adam*, *RMSprop* y *SGDM*, las cuales se detallaron en el Capítulo 2.

3.2.3 Etapa de evaluación

La red entrenada debe ser evaluada para determinar su rendimiento en la tarea de detección de árboles y edificaciones; para ello se introduce a la red las imágenes de evaluación y se analiza los resultados en base a métricas de desempeño, las cuales se describen a continuación con sus respectivas ecuaciones con cada parámetro en función de Verdaderos positivos (TP), Verdaderos Negativos (TN), Falsos positivos (FP) y Falsos negativos (FN).

- **Precisión (AP):** Mide la tasa entre los verdaderos positivos y el número de muestras predichas como positivas.

$$\text{Precisión} = \frac{VP}{VP + FP} \times 100$$

- **Sensibilidad o *Recall (R)*:** Tasa entre los verdaderos positivos y las muestras que efectivamente son positivas, lo cual determina cuán eficaz es la red para detectar muestras verdaderamente positivas.

$$\text{Sensibilidad} = \frac{VP}{VP + FN} \times 100$$

- **Exactitud o *Accuracy*:** Representa la tasa de predicciones correctas frente al total de predicciones hechas.

$$\text{Exactitud} = \frac{VP + VN}{VP + FP + VN + FN} \times 100$$

- **Especificidad:** Mide la proporción de verdaderos negativos que se identifican realmente como tales, su métrica es opuesta a la de la sensibilidad.

$$\text{Exactitud} = \frac{VN}{VP + FP} \times 100$$

- **Precisión promedio (AP):** Métrica que indica si la red puede identificar correctamente todas las muestras positivas sin marcar demasiados falsos positivos. Esta métrica se calcula como el área bajo la curva que mide el equilibrio entre la

precisión y la sensibilidad.

- **Precisión promedio media (mAP):** Mide el valor promedio de los AP de cada clase de objeto a detectar en la red, un alto valor de mAP indica un mejor rendimiento del detector para todo el conjunto de datos empleado.

Las métricas mencionadas anteriormente se pueden asociar y visualizar en un gráfico denominado matriz de confusión tal como se observa en la Figura 22.

<table>
<tr><td colspan="2" rowspan="2">Matriz de confusión</td><td colspan="2">Estimado por el modelo
Negativo (N) | Positivo (P)</td><td colspan="2"></td></tr>
<tr><td rowspan="2">Real</td><td>Negativo</td><td>a: (TN)</td><td>b: (FP)</td><td></td><td></td></tr>
<tr><td>Positivo</td><td>c: (FN)</td><td>d: (TP)</td><td>Precisión
("precision")
Porcentaje predicciones positivas correctas:</td><td>d/(b+d)</td></tr>
<tr><td colspan="2"></td><td>Sensibilidad, exhaustividad
("Recall")
Porcentaje casos positivos detectados</td><td>Especifidad
(Specifity)
Porcentaje casos negativos detectados</td><td colspan="2">Exactitud
("accuracy")
Porcentaje de predicciones correctas
(No sirve en datasets poco equilibrados)</td></tr>
<tr><td colspan="2"></td><td>d/(d+c)</td><td>a/(a+b)</td><td colspan="2">(a+d)/(a+b+c+d)</td></tr>
</table>

Figura 22. Matriz de confusión y métricas asociadas [32].

Capítulo 4

En el presente capítulo se exponen y se comentan los resultados de las redes propuestas como solución en el Capítulo 2 de esta tesis. Dichas redes se entrenan con dos conjuntos de datos etiquetados que presentan objetos multiescala y de contextos variados. Asimismo, en cada entrenamiento de las redes, se especifican los hiperparámetros como tamaño de lote, taza de aprendizaje, número de épocas, optimizador, etc. Por último, cada resultado es analizado y comparado entre ellos y las desarrolladas en el estado del arte del Capítulo 1.

4.1 Consideraciones de implementación

Las imágenes provienen de tomas hechas por drones pertenecientes a la empresa StatKraft Perú, a la universidad PUCP, además de ser extraídas de videos aéreos de *YouTube*. El primero de ellos posee una cámara DJI modelo FC6520 que captura imágenes con una resolución de 5280 x 2970 pixeles; y el segundo, una cámara DJI modelo Phantom 4 que captura imágenes de 1920 x 1088 pixeles, mientras que las imágenes extraídas de los videos de *YouTube* tienen resoluciones variadas en las que predomina las de 1920 x 1088 píxeles. Respecto a los objetos de interés, todas las imágenes presentan tanto árboles como edificaciones de diversos tamaños en diversos contextos rurales y urbanos, representando algunos de ellos porciones pequeñas de la imagen total. Ante esta particularidad, se dividió el *dataset* en dos paquetes, el primero, bajo la excepción que no se contaba aún con suficientes fuentes para la extracción de imágenes, consta de tres tipos de escenarios, pero con una considerable cantidad de imágenes; el segundo, con la finalidad de suplir la escasez de escenarios de la primera, consta de diversos escenarios, pero una cantidad menor de imágenes. Por cada *dataset* se realizó dos tipos de experimentos; para el primero, el primer experimento realiza un preprocesamiento de imágenes mediante recortes de regiones de

interés antes de ser ingresadas a la red *Faster RCNN* con la finalidad de extraer eficientemente las características de los objetos contenidos en la imagen, mientras el segundo experimento emplea la versión ligera llamada *YOLOv4-Tiny* cuya subred *Feature Pyramid Network (FPN)* mejora su capacidad al detectar objetos multiescala. Respecto al segundo *dataset*, el primer experimento emplea la red *Faster RCNN* mejorada con la subred *FPN*; y el segundo, la red *YOLOv8l*, que es una versión mejorada de la *YOLOv4*.

Tabla 3. Tabla de *Sets* de Imágenes utilizados

Set de imágenes		Cantidad de imágenes	Resolución de imágenes	Cantidad de árboles etiquetados	Cantidad de edificaciones etiquetados
DataSetImages	Statkraft	1560	5280 x 2970	9644	9686
	Jardines_PUCP		1920 x 1088		
	Youtube		1280 x 720		
DataSetImages_Zoom*	Statkraft	1560	5280 x 2970	2871	3333
	Jardines_PUCP		1920 x 1088		
	Youtube		1280 x 720		
DatasetVariados	Youtube	350	$w_{min,max}$ = [640, 5280]	6891	6086
	DataSetImages (mínimo porcentaje)				

***** DataSetImages_Zoom = DataSetImages recortado

(a). Imagen aérea proporcionada por la empresa Statkraft con resolución de 5280 x 2970

(b). Imagen aérea proporcionada por la universidad PUCP con resolución de 1920 x 1088

(c). Imagen aérea extraída de *YouTube* con resolución de 280 x 720

(d). Imágenes recortadas

Figura 23. Imágenes de “DataSetImages”

Figura 24. Imágenes de “DatasetVariados”

4.2 Descripción de la implementación

Esta sección está dividida en dos partes, ambas muestran los resultados del detector con las arquitecturas base *Faster RCNN* y *YOLOv4*, pero es la segunda parte donde se emplea las versiones mejoradas de las arquitecturas de ambas redes. En la primera prueba se entrena, valida y evalúa la red independientemente con los conjuntos de datos "DataSetImages" y "DataSetImages_Zoom" usando el software de Matlab, mientras, en el segundo experimento se emplea, "DatasetVariados" en el *framework* Pytorch.

4.2.1 Primera prueba

4.2.1.1 Experimentos con *Faster RCNN*

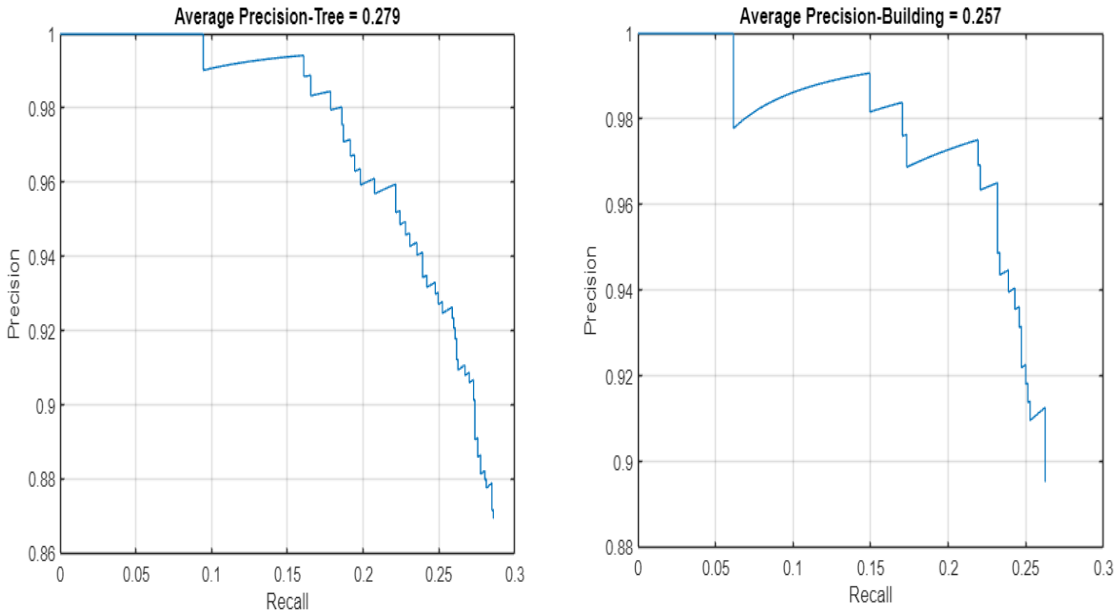
- ***Primer experimento: DataSetImages***

  En el primer experimento de la primera prueba con la red *Faster RCNN* se trabajó con el conjunto de datos "DataSetImages" para el entrenamiento, validación y evaluación de la red cuyos hiperparámetros fueron configurados según la Tabla 4, con dicho detector se obtuvo los resultados más bajos en cuanto a la precisión promedio en ambas clases, dando un resultado del 28% para árboles y un 26% para edificaciones como se observa en la Figura 25a-b; logrando una precisión promedio media de 27%. Esto debido al área de pocos píxeles que ocupan los objetos de interés respecto al área total de la imagen tal como se corrobora en la Figura 23a-b, esta característica perjudica al aprendizaje de la red durante su entrenamiento pues las imágenes serán redimensionadas a una resolución de 224 x 224 antes de ser introducidas a la red troncal,

dicho redimensionamiento trae como consecuencia una pérdida de información de los objetos de interés, ya que al disminuir su resolución se pierden detalles de ellos y por tanto la red no aprende a diferenciar sus características relevantes y su precisión será baja. Como resultado de un bajo rendimiento, se marcan demasiados Falsos Positivos y Falsos Negativos (Figura 26a-b) comparados a los Verdaderos Positivos.

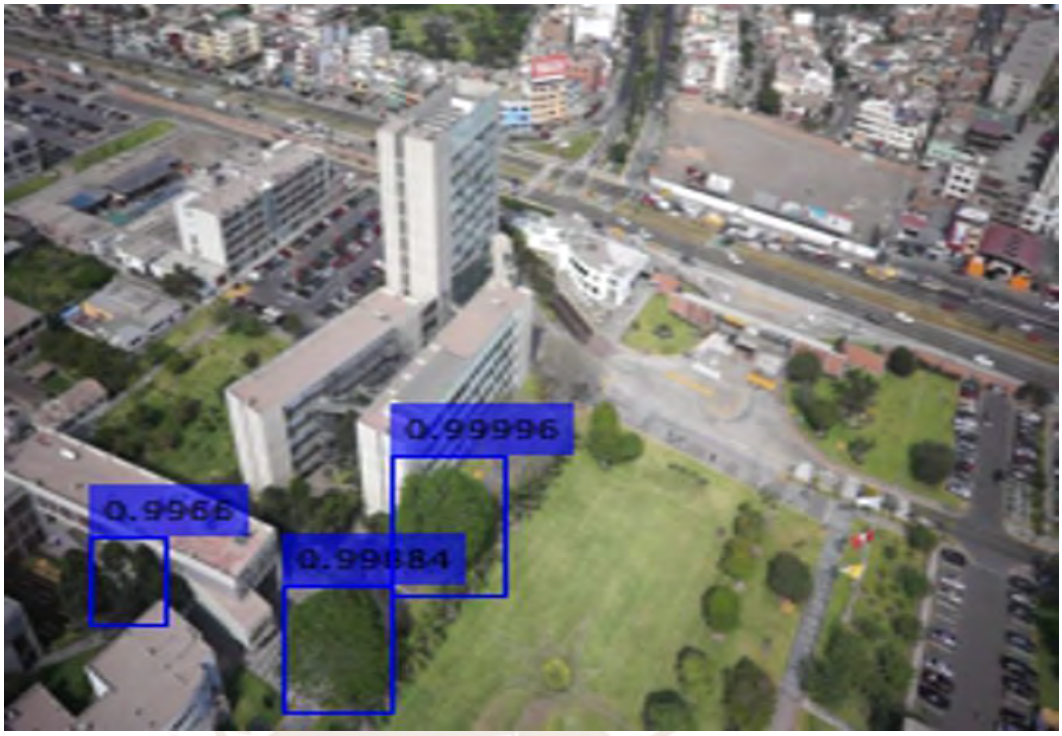
Tabla 4. Configuración de Hiperparámetros – *Faster RCNN*

Hiperparámetro	Valor
Optimizador	SGDM
Learning Rate (lr)	0.01
Factor de impulso (β)	0.9
Batch Size	4
Épocas	20



(a) mAP de la clase "*TREE*" (b) mAP de la clase "*BUILDING*"

Figura 25. Precisión promedio – *Faster RCNN*



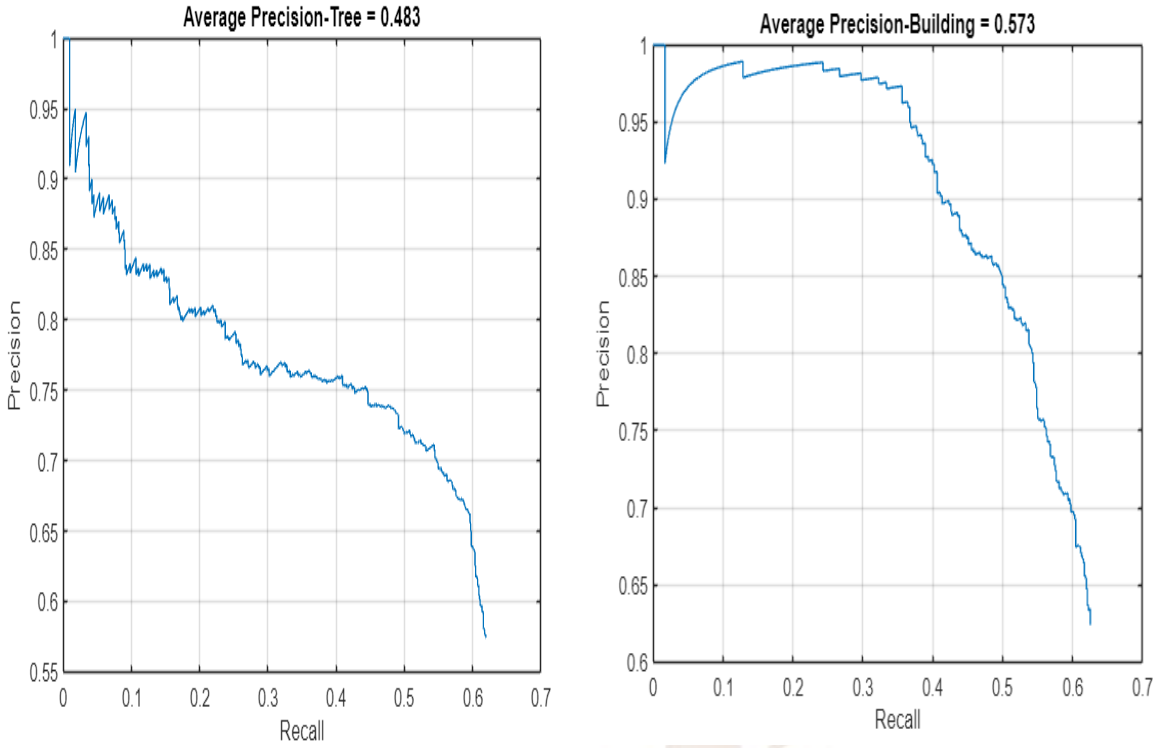
(a)

(b)

Figura 26. Resultados del detector *Faster RCNN*

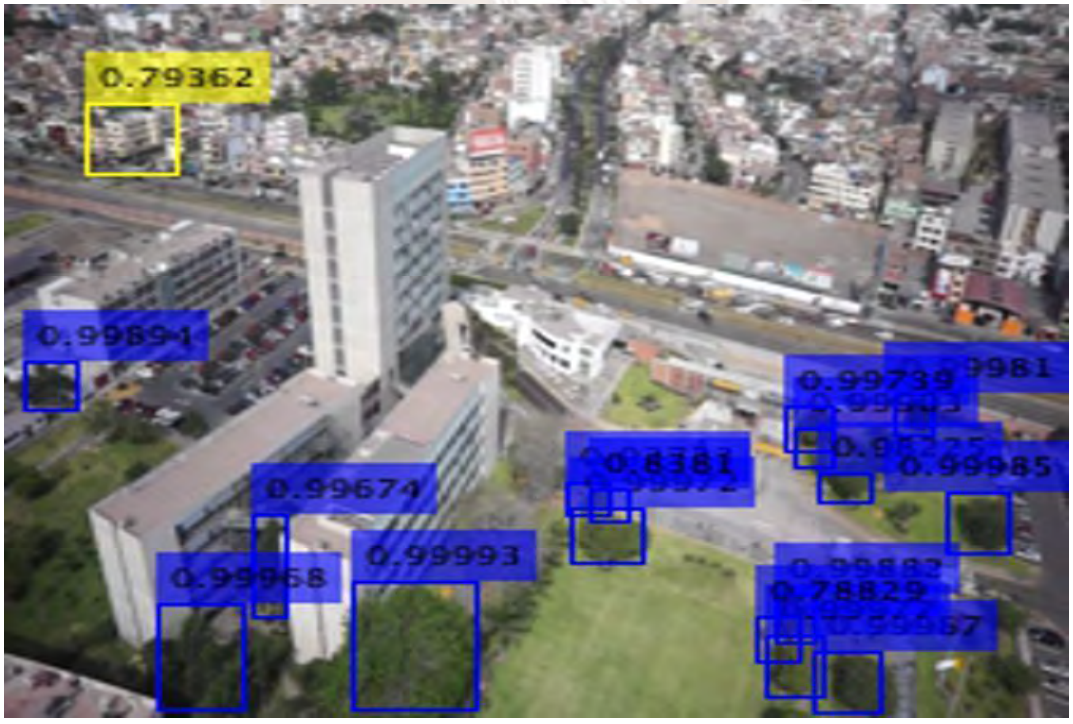
- ***Segundo experimento: DataSetImages_Zoom***

En el segundo experimento de la primera prueba con la red *Faster RCNN*, cuyos hiperparámetros son los mismo de la Tabla 4, se emplea el conjunto de datos "DataSetImages_Zoom", el cual está conformado por imágenes recortadas del conjunto de datos "DataSetImages", recortes enfocados en las regiones de interés donde se encuentran los objetos a detectar, con la finalidad de que las imágenes ingresadas para el entrenamiento de la red no pierdan información relevante de sus características tras su redimensionamiento a un resolución menor de 224 x 224. Los resultados del detector mejoraron duplicando prácticamente su desempeño en comparación a los resultados del primer experimento tal como se observa en la Figura 27a-b, dando un valor de precisión promedio del 48% en la clase de árboles y del 57% en la de edificaciones, logrando una precisión promedio media de 53%. Aún se marcan muchos Falsos Positivos, pero disminuyen los Falsos Negativos (Figura 28a-b) en comparación al primer experimento. Cabe resaltar que, durante la etapa de prueba, las imágenes del *Test* son las mismas que el "DataSetImages" con la diferencia que cada una de ellas antes de ingresar a ser evaluada por la red, es dividida en nueve partes para no ser afectada por el redimensionamiento a 224 x 224.

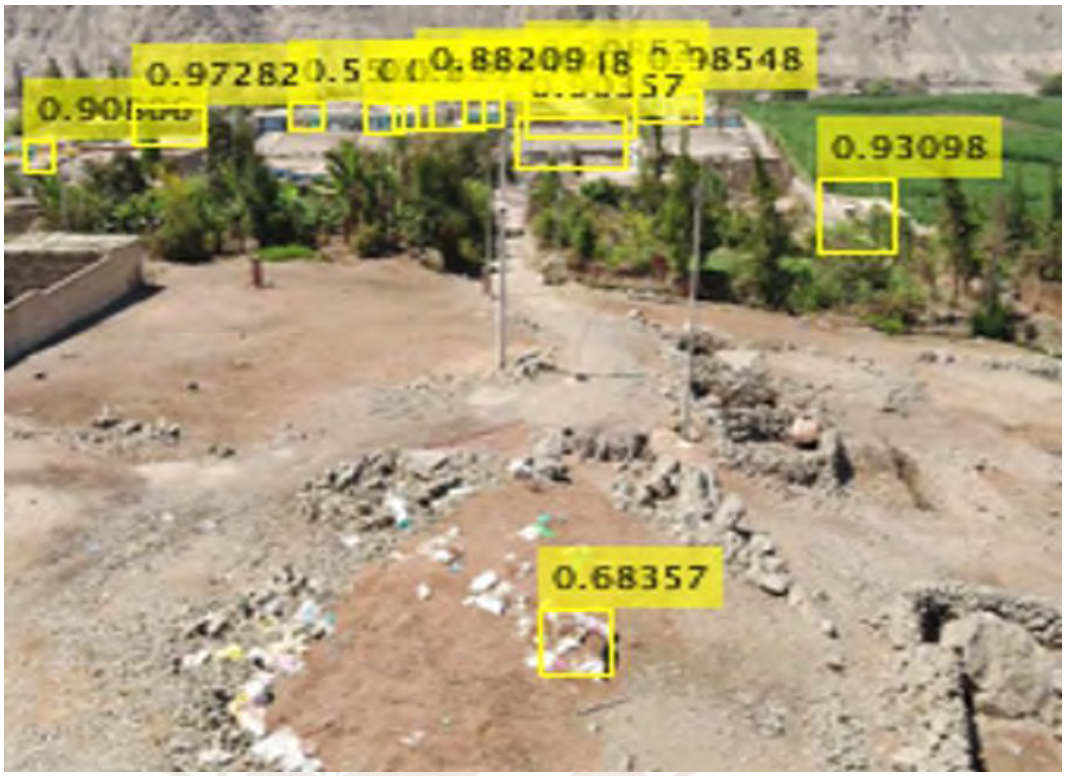


(a) mAP de la clase "*TREE*" (b) mAP de la clase "*BUILDING*"

Figura 27. Precisión promedio – *Faster RCNN* mejorada



(a)



(b)

Figura 28. Resultados de la *Faster RCNN* mejorada

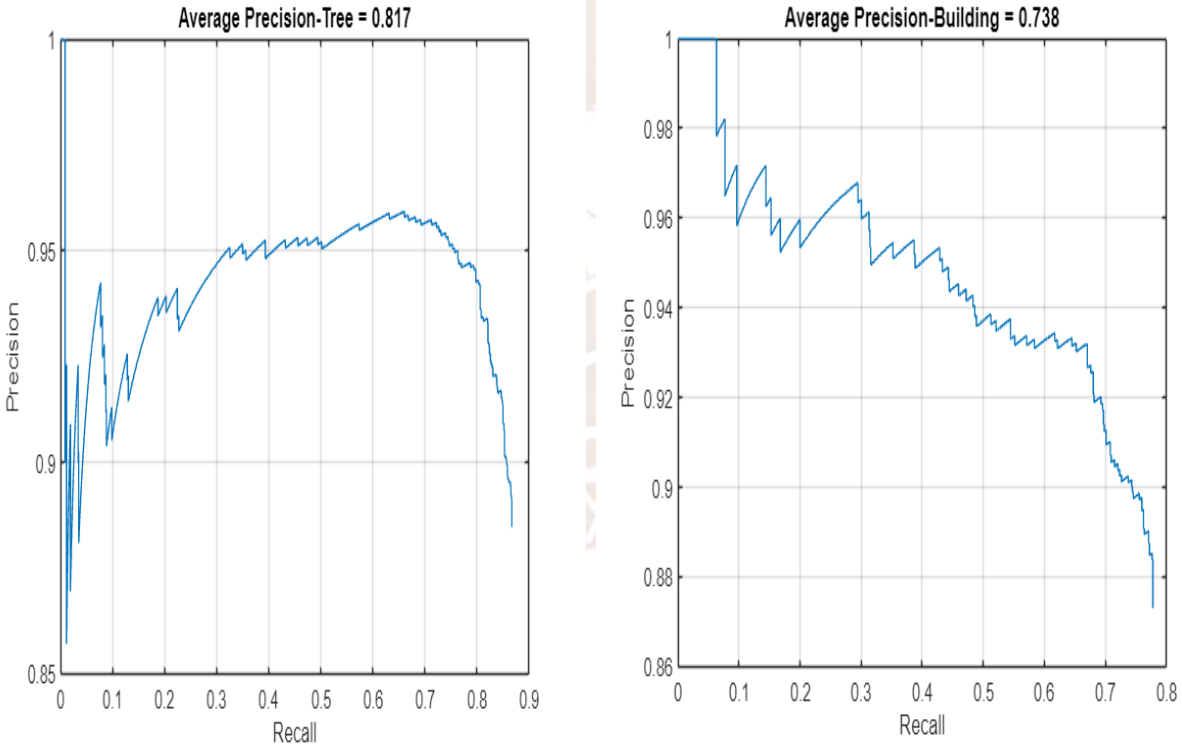
4.2.1.2 Experimentos con *YOLOv4*

En el experimento de la primera prueba con la red *YOLOv4* se emplea el conjunto de datos "DataSetImages" para el entrenamiento, validación y evaluación de la red, y la versión ligera de esta última, *YOLOv4-Tiny*. Por hacer una comparación justa, se introducen a la red las imágenes redimensionadas a una menor resolución de 224 x 224 y los hiperparámetros de entrenamiento son los que se detallan en la Tabla 5. Los resultados muestran una mejora notable respecto a los experimentos con la red *Faster RCNN*, obteniendo resultados de precisión promedio de 82% para la clase de árboles y del 74% para la clase de edificaciones tal como se muestra en la Figura 29a-b, logrando una precisión promedio media del 78%. El detector

marca los objetos de interés sin tantos Falsos Positivos ni Falsos negativos (Figura 30a-b).

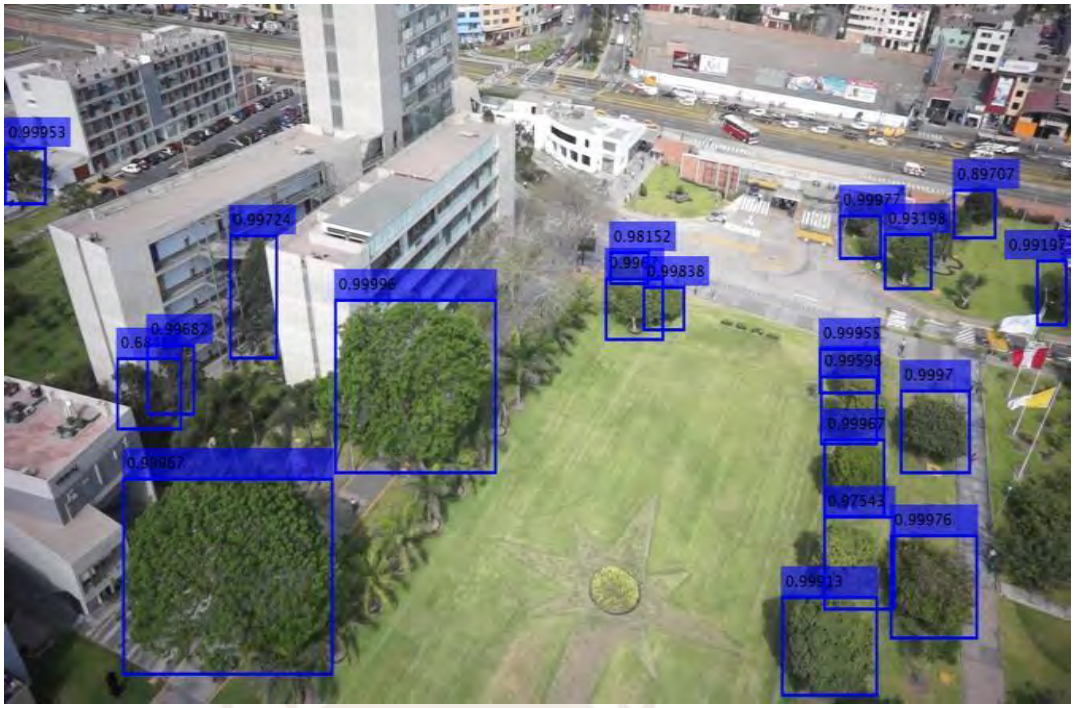
Tabla 5. Configuración de Hiperparámetros – *YOLOv4-Tiny*

Hiperparámetro	Valor
Optimizador	Adam
Learning Rate (lr)	0.0001
GradientDecayFactor (β_1)	0.9
SquaredGradientDecayFactor (β_2)	0.99
Batch Size	4
Épocas	70



(a) mAP de la clase "*TREE*" (b) mAP de la clase "*BUILDING*"

Figura 29. Precisión promedio – *YOLOv4-Tiny*



(a)

(b)

Figura 30. Resultados de la *YOLOv4-Tiny*

4.2.2 Segunda prueba

4.2.2.1 Experimento con *Faster RCNN*

En el experimento de la segunda prueba con la red *Faster RCNN*, se emplea el conjunto de datos "DatasetVariados" para el entrenamiento, validación y evaluación de la red, y la versión mejorada de esta última, *Faster RCNN_FPN*, cuyos hiperparámetros de entrenamiento se detallan en la Tabla 6. La resolución de las imágenes se redimensiona a 640 x 640 para ser introducidas a la red, el cual logra detectar los objetos con una precisión promedio media del 61% tal como se observa en la Figura 31. Este valor de mAP disminuye debido a la que el detector marca Falsos Positivos y Falsos Negativos como se observa en la Figura 32a-d y se corrobora en la matriz de confusión de la Figura 33. La velocidad de la red al entrenar es de prácticamente 17 minutos y, al evaluar es de 7 segundos para un total de 35 imágenes de prueba. En la gráfica de pérdidas de entrenamiento y validación, Figura 34, se observa que a partir de la época 10 la red empieza a sufrir de sobreajuste por lo cual se entrena a la red como máximo para esa cantidad límite de épocas, logrando una buena generalización de esta como se observa en la Figura 35a-b donde se introduce a la red imágenes totalmente nuevas.

Tabla 6. Configuración de Hiperparámetros – *Fater RCNN_FPN*

Hiperparámetro	Valor
Optimizador	AdamW
Learning Rate (lr)	0.0001
GradientDecayFactor (β_1)	0.9
SquaredGradientDecayFactor (β_2)	0.99
Weight Decay	0.01

Lr_Schedule Batch Size	Factor decaimiento = 0.1 Paso decaimiento = 3 2
Épocas	10

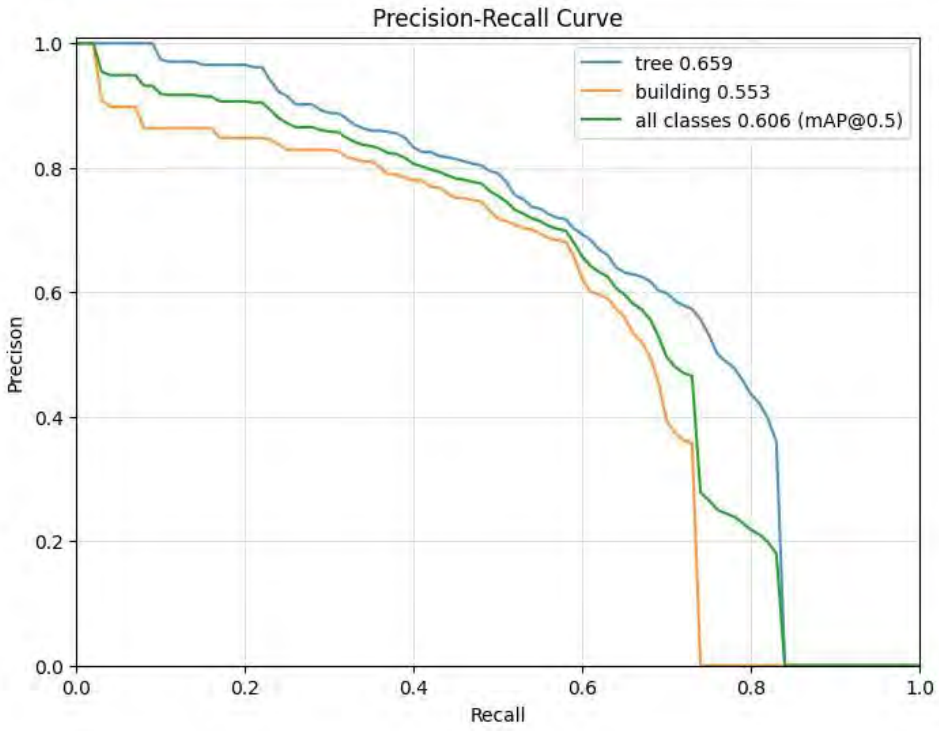
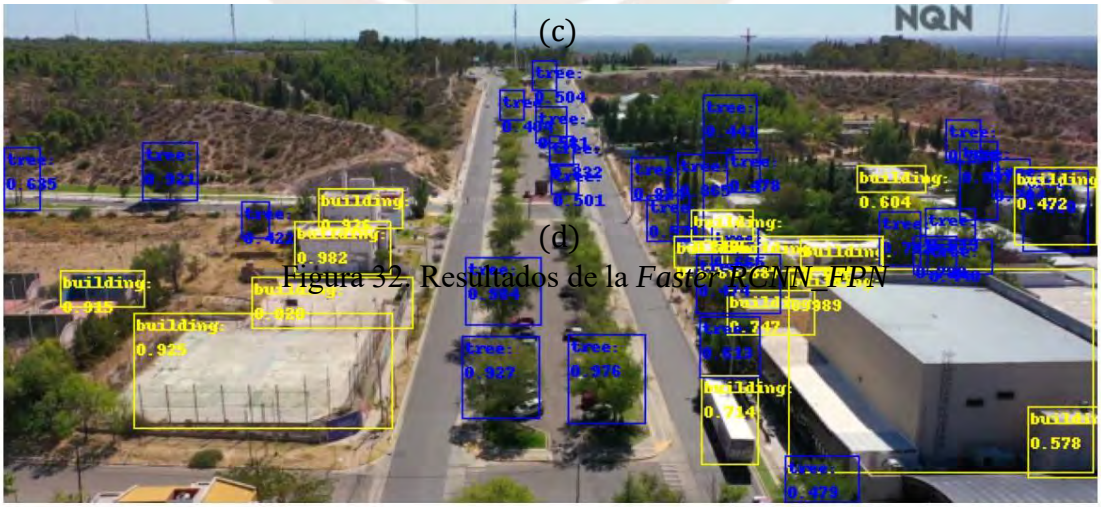


Figura 31. Precisión promedio – *Faster RCNN_FPN*

(a)

(b)

(c)



(d)

Figura 32. Resultados de la *Faster RCNN FPN*

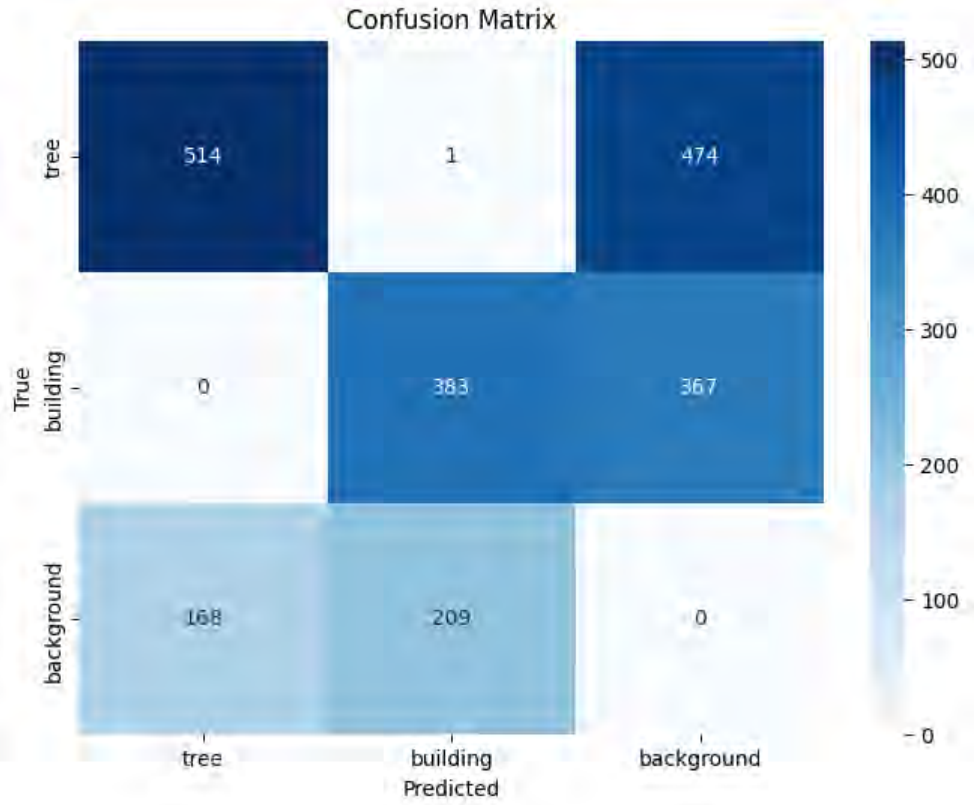


Figura 33. Matriz de confusión - *Faster RCNN_FPN*

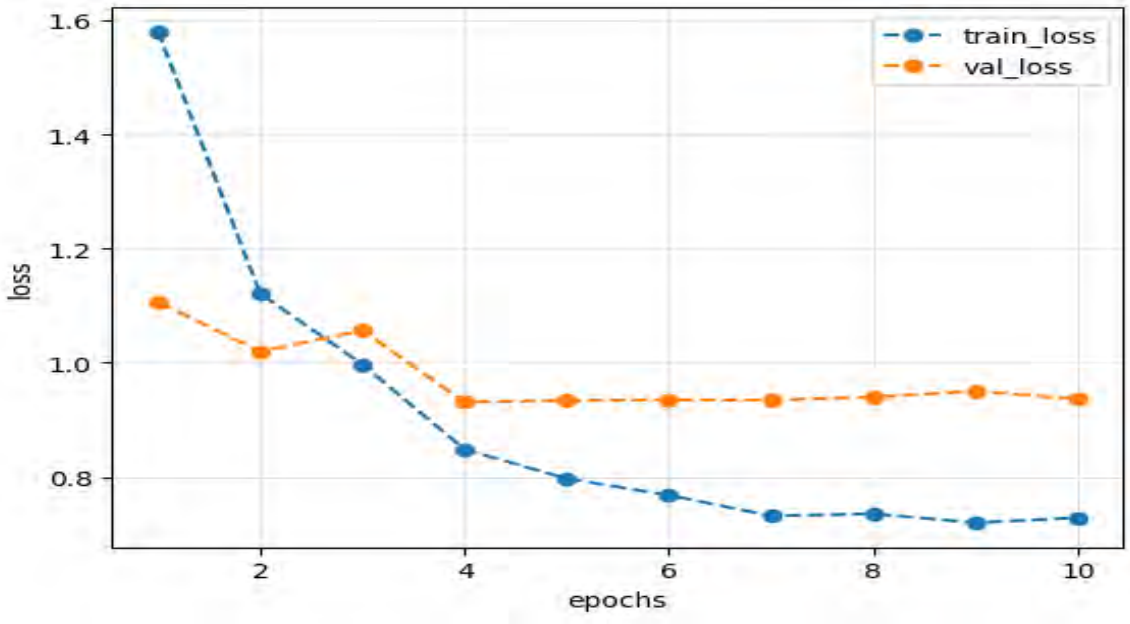
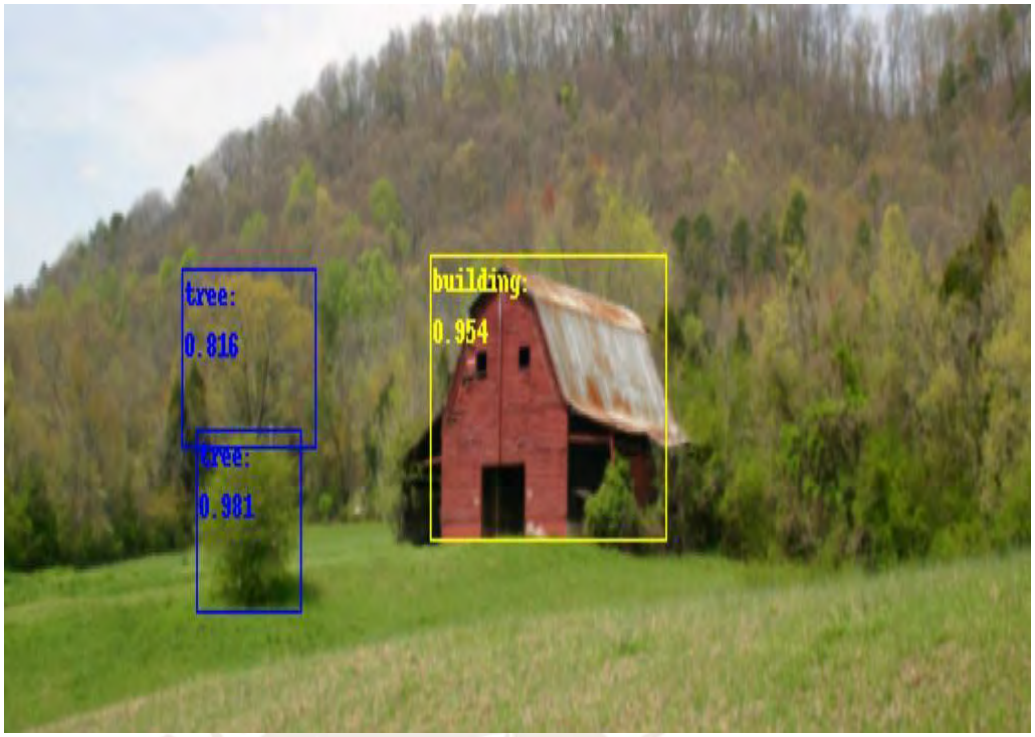


Figura 34. Curva de pérdidas de entrenamiento y validación - *Faster RCNN_FPN*



(a)

(b)

Figura 35. Resultados data nueva - *Faster RCNN_FPN*

4.2.2.2 Experimento con *YOLOv4*

En el experimento de la segunda prueba con la red *YOLOv4*, se emplea el conjunto de datos "DatasetVariados" para el entrenamiento, validación y evaluación de la red, y la versión mejorada de la esta última, *YOLOv8l*, cuyos hiperparámetros de entrenamiento se detallan en la Tabla 7. La resolución de las imágenes se redimensiona a 640 x 640 para ser introducidas a la red, la cual logra detectar los objetos con una precisión promedio del 70% para la clase árboles y del 63% para la clase edificaciones, logrando una precisión promedio media del 67% tal como se observa en la Figura 36. Este valor de mAP disminuye debido a que el detector marca Falsos Positivos y Falsos Negativos como se observa en la Figura 37a-d pero en menor medida que la *Faster RCNN_FPN* y se corrobora en la matriz de confusión de la Figura 38. La velocidad de la red al entrenar es de prácticamente 28 min 19 seg y, al evaluar es de 275 ms para un lote de 35 imágenes de prueba. En la gráfica de pérdidas de entrenamiento y validación, Figura 39, se observa que a partir de la época 50 la red empieza a sufrir de sobreajuste por lo cual se la entrena como máximo para esa cantidad límite de épocas, logrando una buena generalización de esta como se observa en la Figura 40a-b donde se introduce a la red imágenes totalmente nuevas.

Tabla 7. Configuración de Hiperparámetros – *YOLOv8l*

Hiperparámetro	Valor
Optimizador	AdamW
Learning Rate (lr)	0.001
GradientDecayFactor (β_1)	0.937

SquaredGradientDecayFactor (β_2)	0.990
Weight Decay	0.005
Batch Size	4
Épocas	50

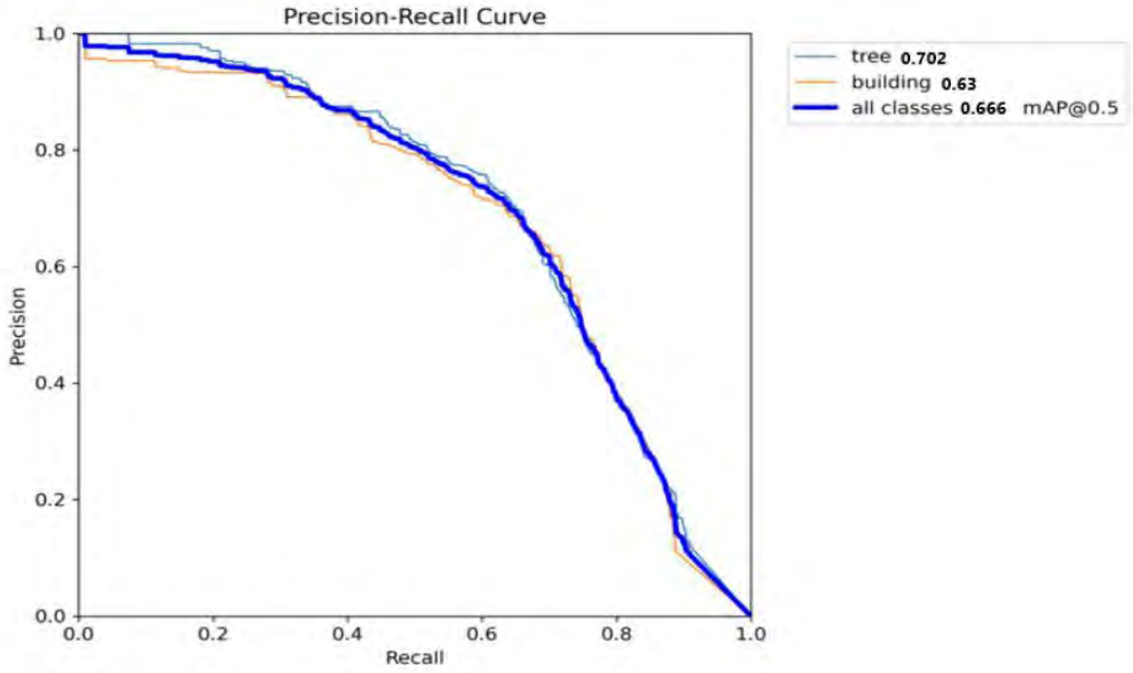
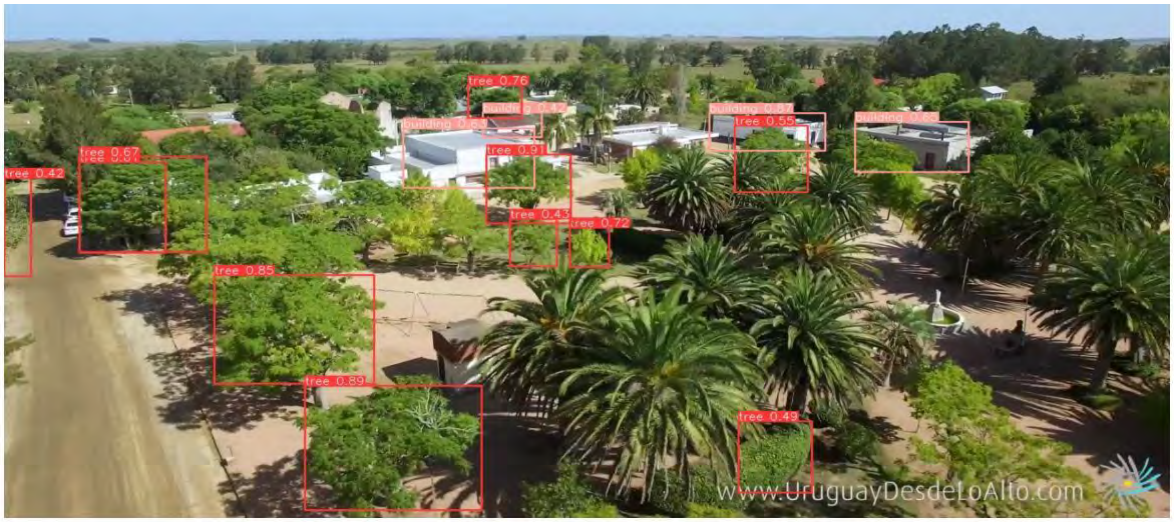
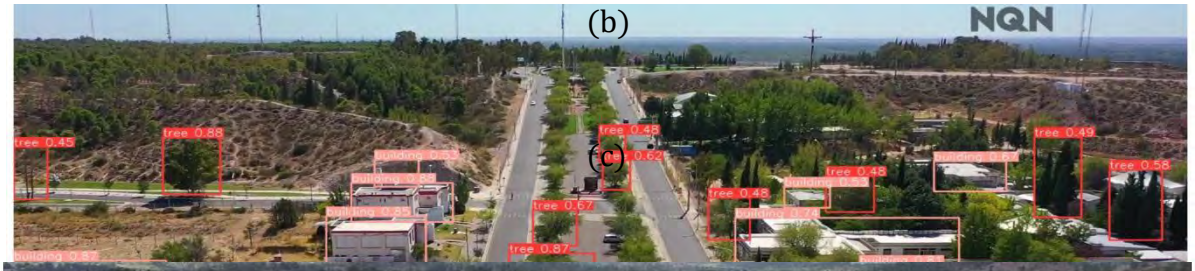


Figura 36. Precisión promedio – *YOLOv8l*

(a)



(b)



(c)

(d)

Figura 37. Resultados de la *YOLOv8l*

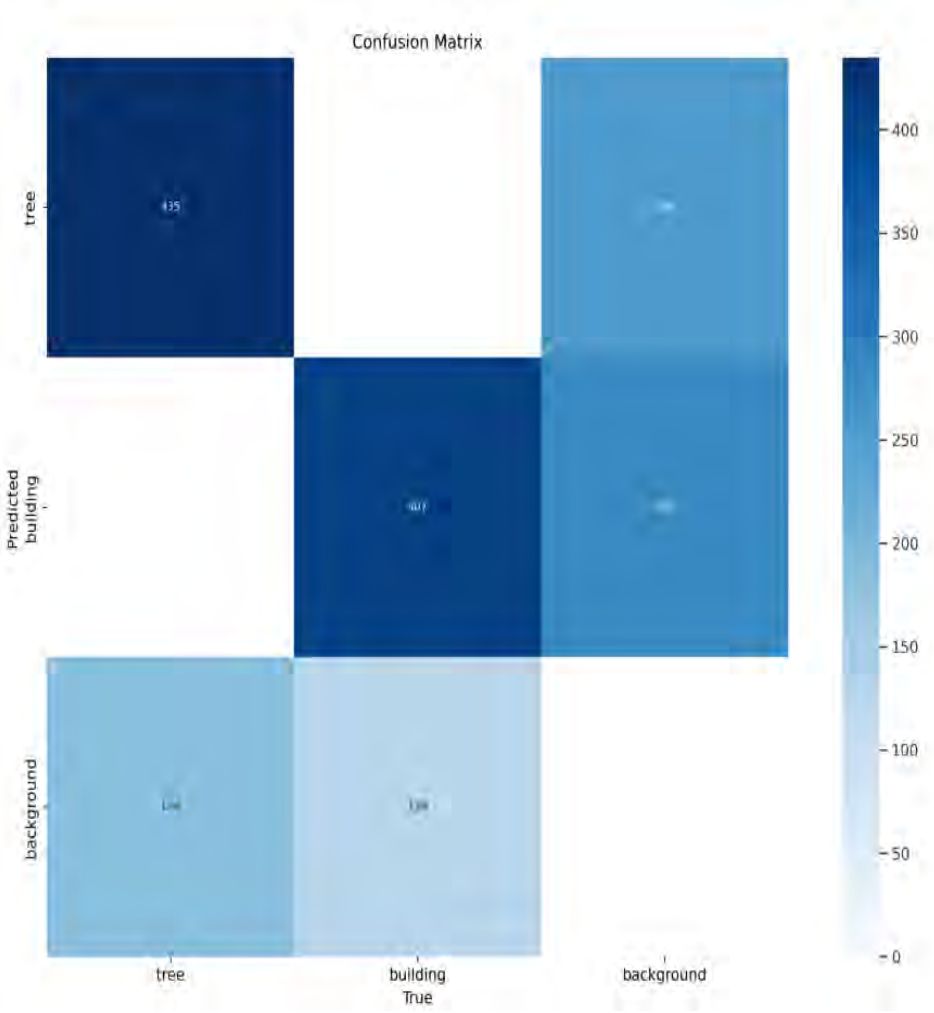


Figura 38. Matriz de confusión - *YOLOv8l*

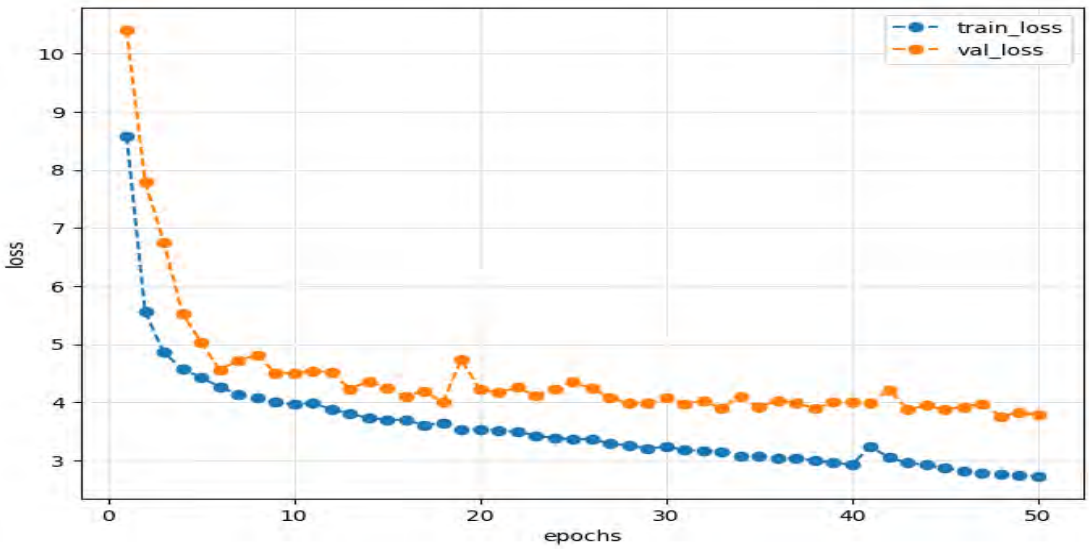


Figura 39. Curva de pérdidas de entrenamiento y validación - *YOLOv8l*

(a)

(b)

Figura 40. Resultados data nueva - *YOLOv8l*

4.3 Análisis de resultados

En esta sección se comparan los resultados obtenidos en ambas pruebas mostrando un resumen en la Tabla 8 de los diversos experimentos realizados por prueba con las redes base *Faster RCNN* y *YOLOv4*. Asimismo, el detector con mejor rendimiento será comparado en la Tabla 9 con los detectores del estado del arte presentados en el Capítulo 1.

Tabla 8. Comparación de resultados de las pruebas realizadas con las redes base *Faster RCNN* y *YOLOv4*

RED	Arquitectura	Dataset	mAP(%) @0.5
FASTER RCNN	Resnet50 + RPN + FC Layer	DataSetImages	27%
FASTER RCNN_ZOOM	Resnet50 + RPN +FC Layer	DataSetImages_Zoom	53%
YOLOv4-Tiny	CSPDarknet53-Tiny + FPN + Yolov3(2 salidas)	DataSetImages	78%
FASTER RCNN_FPN	Resnet50 + FPN + RPN + FC Layer	DatasetVariados	61%
YOLOv8l	C2fDarknet53 + SPPF + BiFPN (C2f) + Yolov3(3 salidas)	DatasetVariados	67%

Tabla 9. Comparación de resultados de las redes con mejor rendimiento y las redes del estado del arte

RED	Arquitectura	Clases a detectar	mAP(%) @0.5
YOLOv4-Tiny	CSPDarknet53-Tiny + FPN + Yolov3 (2 salidas)	Tree, Building	78%
YOLOv8l	C2fDarknet53 + SPPF + BiFPN + Yolov3 (3 salidas)	Tree, Building	67%
EESRGAN + FASTER-RCNN	Generator Network + EEN + D_{Ra} + Detector (Faster-RCNN / SSD)	Car	95.5% @0.5:0.05:0.95
		Tank	83.2% @0.5:0.05:0.95

VistrongerDET	ResNet-50 + FPN enhancement+ ASF + FC Layer	VisDrone2021-DET*	57.27%
Focus and Detect	Focus Stage: ResNet-50 + FPN + Gaussiano Mixture Model Detection Stage: ResNetXt-101 + FPN	VisDrone2021-DET*	66.12%
Eagle-YOLO	CSPDarkNet53 + Bi-FPN(LKAM) + YOLOv3 Large-size	VisDrone2021-DET*	53.64%
LKSNet + O-RCNN	LK Selection (LSK module) + FFN + O-RCNN	DOTAv1.0**	81.85%
AFA-FPN + PHDA + Rotation Branch	Resnet-101 + AFA-FPN + PHDA + Rotation Branch	RSSO**	82.04%

*VisDrone2021-DET: Pedestrian, Person, Bicycle, Car, Van, Truck, Tricycle, Awning-tricycle, Bus, Motor
** DOTAv1.0 = RSSO: Baseball Diamond, Storage Tank, Ship, Tennis court, Bridge, Ground Field Track, Harbor, Baseball Court, Small/Large Vehicle, Swimming Pool, Soccer Ball Field, Helicopter, Roundabout

De la Tabla 7, se concluye que, para ambos conjuntos de datos que se diferencian en la diversidad de escenarios de las imágenes tomadas, las redes basadas en *YOLOv4* obtienen la mejor precisión promedio media a un IoU de 0.5, alcanzando la *YOLOv4-tiny*, la cual es entrenada y evaluada en "DataSetImages", un mAP de 78% y la *YOLOv8l*, entrenada y evaluada em "DatasetVariados", un mAP de 67%. Ciertamente la red *YOLOv8l* presenta una arquitectura más robusta y se espera que tenga un mejor rendimiento que la *YOLOv4-Tiny* pero la diferencia de los resultados radica en el tipo de *dataset* empleado y la cantidad de imágenes que conforman dicho conjunto de datos. "DatasetVariados" presenta escenarios más variados y complejos que "DataSetImages" y la cantidad de imágenes que la conforman es prácticamente 4.5 veces menor en volumen; por ello, el mAP es menor, pero tiene una mejor generalización al ser entrenado en diversos contextos urbanos y rurales. Asimismo, al ser ambas redes *YOLO*, redes de una sola etapa, ambos detectores presentan un menor tiempo de evaluación que las redes basadas en

Faster RCNN. Respecto a esta rede de dos etapas, los resultados son menores debido a su poca capacidad para detectar objetos multiclase donde abundan las de escala pequeña; por ello, la red base *Faster RCNN* es la que menor mAP presenta, dicho valor se trató de mejorar empleando dos técnicas, una de preprocesamiento mediante el recorte de las regiones de interés que agrupan los objetos a detectar, y la otra que implica un cambio en la arquitectura con la adición de una red *FPN* tras la red troncal *Resnet.* Ambas técnicas mejoraron los resultados de la red base *Faster RCNN*, pero las redes basadas en *YOLOv4*, presentan mayores valores de mAP, ya que esta última presenta una red troncal más eficiente, una red *FPN* mejorada y en su salida cabezales de dos escalas para *YOLOv4-Tiny* y de tres escalas para *YOLOv8l*.

Respecto a la Tabla 8, los detectores basados en *YOLOv4* son tan eficientes y precisos como los del estado del arte, comparando sus valores de mAP con los detectores *VistrongerDET*, *Focus and Detect*, *Eagle-YOLO* (basado en la *Yolov5x*), con la diferencia que estas últimas son capaces de ubicar y clasificar una mayor cantidad de clases de objeto. Además, cabe resaltar, que estas redes fueron entrenadas en un conjunto de datos de un gran volumen de imágenes llamado VisDrone2021-DET, el cual emplea 6471 imágenes para entrenamiento, 548 para validación y 1580 para evaluación. Dicho volumen de datos es sobredimensionado respecto al conjunto de datos empleado en esta tesis "DataSetImages" y "DatasetVariados" que se elaboraron manualmente con la aplicación *Image Labeler* del *software* de Matlab.

En resumen, analizando los resultados y los factores de entrenamiento, se elige como propuesta de solución a la red *YOLOv8l* al presentar un alto valor de mAP de 67%, una velocidad de detección competitiva de 7.85ms por imagen y su buena generalización al detectar los objetos de interés en data totalmente nueva para la red.

Conclusiones

- El detector de árboles y edificaciones propuesto como modelo de solución construido a partir de redes neuronales convolucionales profundas cumple con el objetivo de determinar la ubicación de estos dos objetos mencionados en imágenes aéreas tomadas por drones; además de etiquetarlos con su clase correspondiente. Aunque existe presencia de Falsos Positivos y Falsos Negativos en ciertas imágenes, la precisión al detectar los árboles es del 70.2% y al detectar edificaciones es del 63%, resultados que indican que la red detecta correctamente los objetos de interés.

- En la tarea de detección de árboles y edificaciones en imágenes aéreas, donde la mayoría de los objetos de interés presenta una resolución pequeña respecto a la dimensión total de la imagen; se presenta muchos Falsos Positivos y Falsos Negativos que disminuyen el valor de la precisión promedio media de la red con un valor de apenas 27%; por ello, se adiciona a la red troncal una red mejorada basada en la *FPN* que permita fusionar características de alto nivel semánticamente fuertes, ideales para una correcta clasificación de los objetos de interés, con características de baja resolución ricas en información espacial que ayudan a ubicar o localizar con precisión dichos objetos.

- Fue crucial el aumento de la cantidad de imágenes de los *datasets* durante el entrenamiento de la red pues se contaba con un número reducido de imágenes en comparación con los conjuntos de datos del estado del arte, lo cual influye en la precisión de la red al detectar los objetos de interés, árboles y edificaciones.

Recomendaciones

- El detector propuesto como solución fue construido en base a la red *YOLOv8l*, la cual fue entrenada, validada y evaluada con el conjunto de datos "DatasetVariados" cuyas imágenes presentan una variedad de escenarios urbanos y rurales, pero cuyo volumen de data es reducido, apenas oscilando a las 350 imágenes. Por ello, se recomienda realizar más tomas aéreas con drones en diversas zonas de preferencia sobre áreas de fajas de servidumbre; además se puede seleccionar las imágenes de los datasets de acceso libre donde se visualice los objetos de interés, árboles y edificaciones, para su posterior etiquetado.
- Los hiperparámetros de entrenamiento pueden ser más exhaustivos de tal manera que la precisión de la red mejore. Para ello, sobre todo, se puede variar los optimizadores, emplear un *Learning Schedule* con *warmup* y *decay rate*, y agregar reguladores como *dropout*.
- Para el mejoramiento de la extracción de características se puede emplear módulos de atención espacial, tales como el módulo *PHDA* de la red *AFA-PHDA Network* o el módulo *LKAM* de la red *Eagle-YOLO* con la finalidad de extraer información espacial reduciendo el ruido de *background*, es decir, enfocándose en las zonas potenciales de objetos, pero que generen apenas un pequeño coste computacional.
- Si bien es cierto que el detector propuesto como solución está basado en un modelo supervisado, para trabajos futuros sería dable emplear modelos no supervisados para evitar el trabajo laborioso del etiquetado de los objetos de interés en las imágenes del conjunto de datos creado.

Anexos

A. Código de la red *YOLOv8l* en Pytorch

```
[ ] import numpy as np
    import pandas as pd
    import os
    import random
    import shutil
    import glob
    import yaml
```

Data directory

```
[ ] images_dir = "/content/data"
```

Load the dataset

```
[ ] import json

    json_dir = "/content/drive/My Drive/TESIS/Dataset/data.json"

    # Open the JSON file
    with open(json_dir, 'r') as file:
        data = json.load(file)
```

```
[ ] #Muestro info de la primera imagen como ejemplo
    data[0]
```

Class ids

```
[ ] class_id = {
        "TREE" : 0,
        "BUILDING" : 1
    }
```

```
[ ] !unzip -qq "/content/drive/My Drive/TESIS/Dataset/data.zip" -d "/content"
```

Dataframe from json file

```
[ ] import pandas as pd

    def create_dataframe(json_data, class_id, images_dir):
        data_dict = {
            'filename': [],
            'label': [],
            'class_id': [],
            'width': [], #image width
            'height': [], #image height
            'bboxes': []
        }

        for entry in json_data:
            img_name = entry['imageFilename']
            boxes = entry['Boxes'] # (xmin, ymin, width, height)
            labels = entry['Labels']
            image = Image.open(images_dir + "/" + img_name)
            img_width, img_height = image.size

            if isinstance(labels, list):
                for i in range(len(labels)):
                    data_dict['filename'].append(img_name)
                    data_dict['label'].append(labels[i])
                    data_dict['class_id'].append(class_id[labels[i]])
                    data_dict['bboxes'].append(boxes[i])
                    data_dict['width'].append(img_width)
                    data_dict['height'].append(img_height)
            else:
                data_dict['filename'].append(img_name)
                data_dict['label'].append(labels)
                data_dict['class_id'].append(class_id[labels])
                data_dict['bboxes'].append(boxes)
                data_dict['width'].append(img_width)
                data_dict['height'].append(img_height)

        return pd.DataFrame(data_dict)
```

```
[ ] data_df = create_dataframe(data, class_id, images_dir)
```

```
[ ] data_df.head()
```

```
[ ] data_df[data_df['filename'] == 'DJI_281.jpeg']
```

Dataset information

```
[ ] #Número de objetos etiquetados en la Dataset
    print(f'Dataset size: {data_df.shape}')
```

```
[ ] #Valindo data
    print("NaN values:", end = '\n')
    data_df.isna().sum()
```

```
[ ] data_df.info()
```

```
[ ] # Contabilizando objetos etiquetados por clase definida
    data_df.label.value_counts()
```

Each image can contain multiple objects

```
[ ] #Múltiples objetos etiquetados por imagen
    data_df.filename.value_counts()
```

```
[ ] #Número de imágenes en Dataset
    print(f'Unique images: {len(data_df.filename.unique())}')
```

Data visualization

```
[ ] def show_random_images_with_bbox(df, images_dir, figsize=(10,10)):
        all_images = os.listdir(images_dir)
        random_image_filename = random.sample(all_images, 9)

        fig, ax = plt.subplots(nrows=3, ncols=3, figsize=figsize)
        for i, filename in enumerate(random_image_filename):
            selected_df = df[df['filename'] == filename]

            image = Image.open(images_dir + '/' + filename)

            ax.flat[i].imshow(image)
            ax.flat[i].axis(False)

            image_bboxes = []
            for df_index in range(0, len(selected_df)):
                color = "g" # with_mask
                if selected_df.iloc[df_index].class_id == 0: color = "b" #trees
                elif selected_df.iloc[df_index].class_id == 1: color = "y" #buildings

                #x_min, y_min, x_max, y_max = selected_df.iloc[df_index].bboxes
                x_min, y_min, width, height = selected_df.iloc[df_index].bboxes
                x_max = x_min + width
                y_max = y_min + height

                rect = patches.Rectangle([x_min, y_min], x_max-x_min, y_max-y_min,
                                         linewidth=2, edgecolor=color, facecolor="none")
                ax.flat[i].add_patch(rect)

    show_random_images_with_bbox(data_df, images_dir, figsize=(20, 10))
```

Convert coordinates required by YOLO

The current coordinates that we have for each bounding box have the following format: `[x_min, y_min, width, height]`, where

- xmin, ymin: coordinates of top left corner of the bounding box/rectangle.
- width, height: width and height of the bounding box.

However, YOLO requires: `[x_center, y_center, width, height]`. Thus, we will convert the coordinates to the required format.

```
[ ] # Convert the bbox format to yolo as the current one is on pascal_voc format
    def convert_bbox_to_yolo_format(bbox_array, img_width, img_height):
        x_min, y_min, w, h = bbox_array
        x_max = x_min + w
        y_max = y_min + h

        # Center coordinate
        x_center = ((x_max + x_min) / 2) / img_width
        y_center = ((y_max + y_min) / 2) / img_height

        # Width and height
        width = (x_max - x_min) / img_width
        height = (y_max - y_min) / img_height

        return [x_center, y_center, width, height]
```

```
[ ] # Example
    convert_bbox_to_yolo_format(data_df.iloc[0]['bboxes'], data_df.iloc[0]['width'], data_df.iloc[0]['height'])
```

Data preparation

YOLO requires that the dataset be in a specific format. More details here.

Create directories required by YOLO

```
[ ] train_path = "/content/datasets/train"
    valid_path = "/content/datasets/valid"
    test_path = "/content/datasets/test"
```

```
os.mkdir("/content/datasets")
os.mkdir(train_path)
os.mkdir(valid_path)
os.mkdir(test_path)
```

Split the dataset

```
[ ] def split_data(df, train_ratio, val_ratio, test_ratio, random_state=42):
        assert train_ratio + val_ratio + test_ratio == 1.0, "The sum of train_ratio, val_ratio, and test_ratio should be 1.0."
        # Split the data into training and remaining data
        train_df, remaining_df = train_test_split(df, test_size=val_ratio+test_ratio, random_state=random_state)
        # Split the remaining data into validation and test sets
        relative_test_ratio = test_ratio / (val_ratio + test_ratio)
        val_df, test_df = train_test_split(remaining_df, test_size=relative_test_ratio, random_state=random_state)
        return train_df, val_df, test_df
```

```
[ ] # train: 80%, validation: 10%, test: 10%
    train_df, val_df, test_df = split_data(data_df.filename.unique(), 0.8, 0.10, 0.10)

    print(f'Number of training examples: {train_df.shape}')
    print(f'Number of validation examples: {val_df.shape}')
    print(f'Number of test examples: {test_df.shape}')
```

Copy image files to each folder

```
[ ] def copy_image_file(image_items, folder_name, img_dir):
        for image in image_items:
                image_path = img_dir + "/" + image
                new_image_path = os.path.join(folder_name, image)
                shutil.copy(image_path, new_image_path)

    copy_image_file(train_df, train_path, images_dir)
    copy_image_file(val_df, valid_path, images_dir)
    copy_image_file(test_df, test_path, images_dir)
```

Create labels file

Labels for each image must have the following format: <class_id> <x_center> <y_center> <width> <height>

```
[ ] def create_label_file(data_df, image_items, folder_name):
        for image in image_items:
            fileName = Path(image).stem
            df = data_df[data_df['filename'] == image]
            with open(folder_name + "/" + fileName +'.txt', 'w') as f:
                for i in range(0, len(df)):
                    bbox = convert_bbox_to_yolo_format(df.iloc[i]['bboxes'], df.iloc[i]['width'], df.iloc[i]['height'])
                    bbox_text = " ".join(map(str, bbox))
                    txt = str(df.iloc[i]['class_id'])+ " " + bbox_text
                    f.write(txt)
                    if i != len(df) - 1:
                        f.write("\n")

    create_label_file(data_df, train_df, train_path)
    create_label_file(data_df, val_df, valid_path)
    create_label_file(data_df, test_df, test_path)
```

```
[ ] def walk_through_dir(filepath):
        for dirpath, dirnames, filenames in os.walk(filepath):
            print(f"There are {len(dirnames)} directories and {len(glob.glob(filepath + '/*.jpeg', recursive = True))} images in '{dirpath}'.")
            print(f"There are {len(dirnames)} directories and {len(glob.glob(filepath + '/*.jpg', recursive = True))} images in '{dirpath}'.")
            print(f"There are {len(dirnames)} directories and {len(glob.glob(filepath + '/*.JPG', recursive = True))} images in '{dirpath}'.")

    walk_through_dir(train_path)
    walk_through_dir(valid_path)
    walk_through_dir(test_path)
```

Create YAML file

```
[ ] classes = list(data_df.label.unique())
    class_count = len(classes)
    urban_yaml = f"""
        train: train
        val: valid
        test: test
        nc: {class_count}
        names:
            0 : tree
            1 : building
        """

    with open('urban.yaml', 'w') as f:
        f.write(urban_yaml)

    %cat urban.yaml
```

YOLOv8 installation

```
[ ] %pip install ultralytics
```

```
[ ] import ultralytics
    from ultralytics import YOLO
    ultralytics.checks()
```

Model

Load pre-trained model

There are pre-trained models with different sizes, we will load the smalles one. More information about the models here.

```
[ ] # Pre-trained model
    model = YOLO("yolov8l.pt")
```

```
[ ] model.info()
```

Method to visualize the image with predictions generated by yolo

```
[ ] def visualize_predictions(image_path, figsize=(10,10)):
        predicted_image = Image.open(image_path)
        plt.figure(figsize=figsize)
        plt.imshow(predicted_image)
        plt.title("Prediction")
        plt.axis(False)
        plt.show()
```

Fine-tuning

We can change the hyperparameters according to the following: https://docs.ultralytics.com/usage/cfg/

```
[ ] # Fine-tuning
    # Hyperparameters: https://docs.ultralytics.com/usage/cfg/
    results = model.train(data="urban.yaml", epochs=50)#, lr0=1e-3, optimizer='AdamW')
```

```
[ ] results.curves
```

```
['Precision-Recall(B)',
 'F1-Confidence(B)',
 'Precision-Confidence(B)',
 'Recall-Confidence(B)']
```

Results

Specify the directory of the training results, initially they will be located in `runs/detect/train/` directory, after training for the second time the results will be located in `runs/detect/train2/`, etc.

```
[ ] results_dir = "runs/detect/train/" # Change directory every time you train the model
```

The following function will show an image. The results of yolo include images of metrics and curves

```
[ ] def show_image(path, figsize=(20,10)):
        img = Image.open(path)
        plt.figure(figsize=figsize)
        plt.imshow(img)
        plt.axis(False)
        plt.show()
```

Confusion Matrix

```
[ ] show_image(results_dir + "confusion_matrix.png")
```

```
[ ] show_image(results_dir + "F1_curve.png", figsize=(10,10))
```

```
[ ] show_image(results_dir + "PR_curve.png", figsize=(10,10))
```

```
[ ] show_image(results_dir + "P_curve.png", figsize=(10,10))
```

```
[ ] show_image(results_dir + "results.png", figsize=(15,15))
```

Loss curves (total)

```
[ ] import matplotlib

    def plot_loss_curve(train_results, val_results, figsize=(8, 6), show_all_epochs=False):
        x = list(range(1, len(train_results) + 1))
        plt.figure(figsize=figsize)
        plt.plot(x, train_results, 'o--', label='train_loss')
        plt.plot(x, val_results, 'o--', label='val_loss')
        plt.xlabel("epochs")
        plt.ylabel("loss")
        plt.grid(color='lightgrey', linestyle='-', linewidth=0.5)
        plt.legend()

        if show_all_epochs:
            locator = matplotlib.ticker.MultipleLocator(2)
            plt.gca().xaxis.set_major_locator(locator)
            formatter = matplotlib.ticker.StrMethodFormatter("{x:.0f}")
            plt.gca().xaxis.set_major_formatter(formatter)
        plt.show()
```

```
[ ] results_path = results_dir + "results.csv"
    results_df = pd.read_csv(results_path)
    results_df.head()
```

```
[ ] train_loss_cols = results_df.columns[1:4]
    val_loss_cols = results_df.columns[8:11]
    print(f'Train loss columns: {train_loss_cols}')
    print(f'Validation loss columns: {val_loss_cols}')

    # Get values
    train_total_loss = results_df[train_loss_cols].sum(axis=1).values
    val_total_loss = results_df[val_loss_cols].sum(axis=1).values
```

```
[ ] plot_loss_curve(train_total_loss, val_total_loss, figsize=(8, 6), show_all_epochs=False) #show_all_epochs (x-axis)
```

Predictions for validation images

```
[ ] val_label = Image.open(results_dir + "val_batch0_labels.jpg")
    val_pred = Image.open(results_dir + "val_batch0_pred.jpg")

    plt.figure(figsize=(20,10))
    plt.imshow(val_label)
    plt.title("Label")
    plt.axis(False)
    plt.show()

    plt.figure(figsize=(20,10))
    plt.imshow(val_pred)
    plt.title("Prediction")
    plt.axis(False)
    plt.show()
```

```
[ ] val_label = Image.open(results_dir + "val_batch1_labels.jpg")
    val_pred = Image.open(results_dir + "val_batch1_pred.jpg")

    plt.figure(figsize=(20,10))
    plt.imshow(val_label)
    plt.title("Label")
    plt.axis(False)
    plt.show()

    plt.figure(figsize=(20,10))
    plt.imshow(val_pred)
    plt.title("Prediction")
    plt.axis(False)
    plt.show()
```

Load fine-tuned model

```
[ ] #https://docs.ultralytics.com/usage/cfg
    predictions_path = "/content/prediction_results"
    if not os.path.exists(predictions_path):
        os.mkdir(predictions_path)
```

```
[ ] # Pretrained model
    model = YOLO(model=results_dir + "weights/best.pt")
    #model = YOLO(model="runs/detect/train4/weights/best.pt")
```

Train results

```
[ ] train_results = model.val(data="urban.yaml", split='train')
```

Validation results

```
[ ] val_results = model.val(data="urban.yaml", split='val')
```

Test results

```
[ ] test_results = model.val(data="urban.yaml", split='test')
```

Predictions

```
[ ] filename = 'DJI_200.jpeg'
    prediction_results = model.predict(images_dir + "/" + filename, save=True, line_width=3, conf=0.4)
```

```
[ ] results_dir_pred = "runs/detect/predict2/" # Usar directorio donde se encuentran las predicciones
```

```
[ ] visualize_predictions(results_dir_pred + filename,figsize=(15,15))
```

Predictions for test images

```
[ ] filenames = glob.glob(test_path+"/*.jpg", recursive=False)
    test_image1 = cv2.imread(filenames[0])
    test_image2 = cv2.imread(filenames[1])
    prediction_results = model.predict([test_image1, test_image2], save=True, line_width=3,conf=0.4,project="runs/detect/predict2",name = "TestImages")
```

```
0: 640x640 5 trees, 25 buildings, 1: 640x640 4 trees, 6 buildings, 120.9ms
Speed: 3.4ms preprocess, 60.4ms inference, 2.1ms postprocess per image at shape (1, 3, 640, 640)
Results saved to runs/detect/predict2/TestImages
```

```
[ ] results_dir_pred = "runs/detect/predict2/TestImages/"
```

```
[ ] filename = 'image0.jpg'
    #filename=os.path.basename(filenames[0])
    visualize_predictions(results_dir_pred + filename, (15,15))
```

```
[ ] filename = 'image1.jpg'
    visualize_predictions(results_dir_pred + filename, (10,10))
```

```
[ ] filenames = glob.glob(test_path+"/*.jpg", recursive=False)
    test_image3 = cv2.imread(filenames[0])
    test_image4 = cv2.imread(filenames[1])
    test_image5 = cv2.imread(filenames[2])
    test_image6 = cv2.imread(filenames[3])
    test_image7 = cv2.imread(filenames[4])
    prediction_results = model.predict([test_image3,test_image4,test_image5,test_image6,test_image7], save=True, line_width=2,conf=0.4, project="runs/detect/predict3",name = "TestImages")
```

```
[ ] results_dir_pred = "runs/detect/predict3/TestImages/"
```

```
[ ] filename = "image0.jpg"
    #filename=os.path.basename(filenames[0])
    visualize_predictions(results_dir_pred + filename, (15,15))
```

```
[ ] filename = 'image1.jpg'
    #filename=os.path.basename(filenames[0])
    visualize_predictions(results_dir_pred + filename, (15,15))
```

```
[ ] filename = 'image2.jpg'
    #filename=os.path.basename(filenames[0])
    visualize_predictions(results_dir_pred + filename, (20,20))
```

```
[ ] filename = 'image3.jpg'
    #filename=os.path.basename(filenames[0])
    visualize_predictions(results_dir_pred + filename, (20,20))
```

```
[ ] filename = 'image4.jpg'
    #filename=os.path.basename(filenames[0])
    visualize_predictions(results_dir_pred + filename, (20,20))
```

Test with external images

```
[ ] filename = 'https://www.knoxmercury.com/wp-content/uploads/2016/10/East-Knox-County-Barn-793x450.png'
    prediction_results = model.predict(filename, save=True, line_width=2, project="runs/detect/predict4",name = "NewImages")
```

```
[ ] results_dir_pred = "runs/detect/predict4/NewImages/"
```

```
[ ] filename = 'East-Knox-County-Barn-793x450.png'
    visualize_predictions(results_dir_pred + filename, (10,10))
```

```
[ ] filename = 'https://previews.123rf.com/images/creativenature/creativenature1306/creativenature130600076/20183272-carretera-nacional-a-trav%C3%A9s-de-una-zona-rural-en-francia-europa.jpg'
    prediction_results = model.predict(filename, save=True, line_width=2,project="runs/detect/predict4",name = "NewImages")
```

```
[ ] results_dir_pred = "runs/detect/predict4/NewImages2/"
    filename = '20183272-carretera-nacional-a-través-de-una-zona-rural-en-francia-europa.jpg'
    visualize_predictions(results_dir_pred + filename, (15,15))
```

```
[ ] #Guardar Archivo de parámetros con los mejores valores tras entrenar la red
    shutil.make_archive("/content/drive/My Drive/TESIS/Train", 'zip', "runs/detect/train")
```

Bibliografía

[1] E. Memor, G. General, O. Supervisor, I. Legal, L. Marco, and O. Reguladores, "Organismo Supervisor De La Inversion En Energia Osinerg N ° 005-2004-Os / Cd," pp. 1–23, 2004.

[2] D. Oficial and E. Peruano, "CÓDIGO NACIONAL DE ELECTRICIDAD (SUMINISTRO 2011)," no. Suministro 2011, 2012.

[3] S. Vemula and M. Frye, "Mask R-CNN powerline detector: A deep learning approach with applications to a UAV," in *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, Institute of Electrical and Electronics Engineers Inc., Oct. 2020. doi: 10.1109/DASC50938.2020.9256456.

[4] I. Gil-Leiva, J.-V. Rodríguez-Muñoz, and P. M. Díaz-Ortuño, "Técnicas y usos en la clasificación automática de imágenes," pp. 1–14, 2019, doi: 10.31229/osf.io/d6ax4.

[5] Facundo Manuel Quiroga, "Medidas de Invarianza y Equivarianza a Transformaciones en Redes Neuronales Convolucionales. Aplicaciones al reconocimiento de formas de mano.," 2020.

[6] J. Rabbi, N. Ray, M. Schubert, S. Chowdhury, and D. Chao, "Small-Object Detection in Remote Sensing Images with End-to-End Edge-Enhanced GAN and Object Detector Network," *Remote Sens (Basel)*, vol. 12, no. 9, Mar. 2020, doi: 10.3390/RS12091432.

[7] J. Wan, B. Zhang, Y. Zhao, Y. Du, and Z. Tong, "VistrongerDet: Stronger Visual Information for Object Detection in VisDrone Images," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2021-October, pp. 2820–2829, 2021, doi: 10.1109/ICCVW54120.2021.00316.

[8] O. Can Koyun, R. Kevser Keser, and gur Töreyin, "FOCUS-AND-DETECT: A SMALL OBJECT DETECTION FRAMEWORK FOR AERIAL IMAGES *", doi: 10.1016/j.image.2022.116675.

[9] L. Liao, L. Luo, J. Su, Z. Xiao, F. Zou, and Y. Lin, "Eagle-YOLO: An Eagle-Inspired YOLO for Object Detection in Unmanned Aerial Vehicles Scenarios," *Mathematics*, vol. 11, no. 9, May 2023, doi: 10.3390/MATH11092093.

[10] Y. Li, Q. Hou, Z. Zheng, M.-M. Cheng, J. Yang, and X. Li, "Large Selective Kernel Network for Remote Sensing Object Detection," Mar. 2023, Accessed: Oct. 30, 2023. [Online]. Available: https://arxiv.org/abs/2303.09030v2

[11] J. Wang, F. Shao, X. He, and G. Lu, "A Novel Method of Small Object Detection in UAV Remote Sensing Images Based on Feature Alignment of Candidate Regions," *Drones*, vol. 6, no. 10, Oct. 2022, doi: 10.3390/DRONES6100292.

[12] "(PDF) A Survey of Small Object Detection Based on Deep Learning in Aerial Images."

[13] X. Wang, A. Wang, J. Yi, Y. Song, and A. Chehri, "Small Object Detection Based on Deep Learning for Remote Sensing: A Comprehensive Review," *Remote Sensing 2023, Vol. 15, Page 3265*, vol. 15, no. 13, p. 3265, Jun. 2023, doi: 10.3390/RS15133265.

[14] "Redes neuronales convolucionales - MATLAB & Simulink." Accessed: Jun. 04, 2021. [Online]. Available: https://la.mathworks.com/discovery/convolutional-neural-network-matlab.html

[15] A. Moreno, "Clasificación de imágenes usando redes neuronales convolucionales en Python," *Universidad de Sevilla*, p. 80, 2019, [Online]. Available: http://bibing.us.es/proyectos/abreproy/92402/fichero/TFG-2402-ARTOLA.pdf

[16] V. De la Fuente, "Diseño evolutivo de arquitecturas de Deep Learning para detección de vías de transporte.," p. 120, 2019, [Online]. Available: https://oa.upm.es/55751/1/TFM_VICTOR_DE_LA_FUENTE_CASTILLO.pdf

[17] P. Xu *et al.*, "On-board real-time ship detection in hisea-1 sar images based on cfar and lightweight deep learning," *Remote Sens (Basel)*, vol. 13, no. 10, May 2021, doi:

10.3390/RS13101995.
[18] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection,” Dec. 2016, Accessed: Dec. 13, 2023. [Online]. Available: https://arxiv.org/abs/1612.03144v2
[19] “Achieving Optimal Speed and Accuracy in Object Detection (YOLOv4) - PyImageSearch.” Accessed: Jan. 31, 2024. [Online]. Available: https://pyimagesearch.com/2022/05/16/achieving-optimal-speed-and-accuracy-in-object-detection-yolov4/
[20] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path Aggregation Network for Instance Segmentation,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 8759–8768, Mar. 2018, doi: 10.1109/CVPR.2018.00913.
[21] “Easiest RPN explaination, the core of Faster R-CNN. | by Kai | LSC PSD | Medium.” Accessed: Nov. 01, 2023. [Online]. Available: https://medium.com/lsc-psd/easiest-rpn-explained-the-core-of-faster-r-cnn-3b0168c3e650
[22] “Capa totalmente conectada - MATLAB - MathWorks América Latina.” Accessed: Jun. 05, 2021. [Online]. Available: https://la.mathworks.com/help/deeplearning/ref/nnet.cnn.layer.fullyconnectedlayer.html
[23] N. Dickerson, “Refining Bounding-Box Regression for Object Localization”, Portland State University , 2017. Accessed: Jul. 24, 2023. [Online]. Available: https://pdxscholar.library.pdx.edu/open_access_etds/3940/
[24] V. Senén, “Diseño, implementación y evaluación de una red neuronal convolucional de regresión en clasificación de naranjas,” 2019, [Online]. Available: https://riunet.upv.es/bitstream/handle/10251/155034/20901959L_TFG_15936111059957994150375869586263.pdf?sequence=1&isAllowed=y
[25] “Loss Functions. Loss functions explanations and… | by Tomer Kordonsky | Artificialis | Medium.” Accessed: Dec. 16, 2023. [Online]. Available: https://medium.com/artificialis/loss-functions-361b2ad439a0
[26] “Different IoU Losses for Faster and Accurate Object Detection | by Renu Khandelwal | Analytics Vidhya | Medium.” Accessed: Dec. 18, 2023. [Online]. Available: https://medium.com/analytics-vidhya/different-iou-losses-for-faster-and-accurate-object-detection-3345781e0bf
[27] “Adam y RMSprop Optimizer.” Accessed: Nov. 30, 2021. [Online]. Available: https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be
[28] “Técnicas de Optimización II · Aprendizaje Profundo.” Accessed: Aug. 28, 2023. [Online]. Available: https://atcold.github.io/pytorch-Deep-Learning/es/week05/05-2/
[29] “Norma por lotes explicada visualmente: cómo funciona y por qué las redes neuronales la necesitan | por Ketan Doshi | Hacia la ciencia de datos.” Accessed: Sep. 11, 2023. [Online]. Available: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739
[30] “Las mejores GPU para inteligencia artificial | Observatorio IA.”
[31] W. N. CHOQUEHUAYTA, “‘DETECCIÓN DE EMBARCACIONES UTILIZANDO DEEP LEARNING E IMÁGENES SATELITALES ÓPTICAS.’”, Accessed: Sep. 18, 2021. [Online]. Available: http://repositorio.unsa.edu.pe/bitstream/handle/20.500.12773/12247/ISnichw.pdf?sequence=1&isAllowed=y
[32] “Cómo interpretar la matriz de confusión: ejemplo práctico.” Accessed: Feb. 11, 2024. [Online]. Available: https://telefonicatech.com/blog/como-interpretar-la-matriz-de-confusion-ejemplo-practico