17/07/2024

Diagnóstico de paneles solares mediante tratamiento automatizado de imágenes y su aplicación a la automatización de programas de mantenimiento predictivo

Autor: Garrido Corrales, Luis
Supervisor: Emilio Manuel Domínguez Adán
Centro: Universidad Pontificia de Comillas

RESUMEN En un contexto donde la eficiencia y la seguridad de las instalaciones industriales son primordiales, la implementación de estrategias de mantenimiento predictivo se vuelve esencial. En el sector de la generación fotovoltaica, este mantenimiento es necesario para mantener bajos los costes variables y así conseguir rentabilizar las instalaciones y garantizar su viabilidad económica.

Este trabajo pretende presentar una alternativa a la metodología utilizada actualmente en las inspecciones termográficas de módulos fotovoltaicos en parques solares. Para ello, se ha propuesto la utilización de algoritmos de detección de objetos, basados en *Deep Learning*, para la identificación y clasificación de defectos en módulos fotovoltaicos a través del análisis de imágenes térmicas.

Los algoritmos elegidos, que son parte del estado del arte de la detección de objetos, son YOLOv5, Faster R-CNN y RetinaNet. Mediante el entrenamiento de estos modelos en imágenes térmicas de módulos fotovoltaicos, se pretende conseguir el diagnóstico automatizado de las instalaciones. Para ello, estos modelos han sido implementados en un entorno de programación en Python, con PyTorch como *framework* principal, y utilizando numerosas librerías de inteligencia artificial y procesamiento y gestión de imágenes. Para la comparación de las tres arquitecturas, se realizaron entrenamientos sobre el mismo dataset y con un mismo conjunto de hiperparámetros. Posteriormente, se realizó un entrenamiento más profundo y detallado para cada arquitectura, buscando maximizar el rendimiento de cada una mediante el ajuste y refinado de hiperparámetros.

Los resultados mostraron que los tres modelos son válidos para la aplicación propuesta, con valores de AP cercanos al 90% en la detección de paneles defectuosos y alrededor del 80% en el conjunto de los elementos a detectar (mAP). Faster R-CNN logró la mayor precisión, mientras que YOLOv5l demostró ser el más rápido, lo que lo hace adecuado para la detección en tiempo real. RetinaNet ofreció un equilibrio entre precisión y velocidad.

La integración de estas tecnologías demuestra un ejemplo claro de cómo el *Deep Learning* puede ser una herramienta poderosa en la gestión y mantenimiento de infraestructuras energéticas, promoviendo un entorno industrial más eficiente y sostenible. Además, estos modelos pueden combinarse con diversas tecnologías propias de la Industria Conectada, como IoT, Big Data y sistemas de gestión de energía, permitiendo la recopilación y análisis de datos en tiempo real para optimizar el rendimiento de las instalaciones.

PALABRAS CLAVE Mantenimiento predictivo, inspección termográfica, módulos fotovoltaicos, *Deep Learning*, YOLOv5, Faster R-CNN, RetinaNet,, Industria Conectada, IoT, *Big Data*, mAP

I. INTRODUCCIÓN

La energía solar fotovoltaica continua con su crecimiento exponencial en España. En 2023, aportó al *pool* energético un 13.99% de la energía total generada, alcanzando una producción anual de 37,332 GWh [1]. La energía renovable en general representó más del 50% de la generación eléctrica en España, destacando la combinación de energía eólica y solar como líderes en esta transición energética.

Como parte del mantenimiento predictivo de estas instalaciones se realizan frecuentes inspecciones que incluyen análisis eléctricos mediante la prueba de curva I-V, inspecciones visuales, pruebas de aislamiento, pruebas de continuidad del cableado o inspección de los inversores de corriente, entre otras.

Sin embargo, una de las pruebas más importantes y frecuentes, es la inspección termografica. Esta consiste en, mediante el uso de cámaras térmicas que captan la radiación del infrarrojo lejano (8 – 14 µm), medir la temperatura de los módulos de forma no invasiva, para así identificar anomalías térmicas como puntos calientes, fallos de conexiones y celdas defectuosas, que pueden disminuir significativamente el rendimiento de los paneles solares.

Este proyecto tiene como objetivo plantear una alternativa para la inspección termográfica de paneles fotovoltaicos utilizando algoritmos de detección de objetos de código abierto, gratuitos y fácilmente desplegables, específicamente YOLOv5, Faster R-CNN, y RetinaNet. También busca encontrar cual de estas tres arquitecturas estado del arte en la detección de objetos es más apropiada para esta aplicación.

II. ESTADO DE LA CUESTIÓN

A. Inspección Termográfica

La normativa de aplicación internacional en cuanto a inspecciones termográficas es la IEC 62446-3 [2]. Esta establece que existen dos tipos de inspecciones: las simplificadas, donde el objetivo es buscar indicios y patrones de averías de origen térmico; y la detallada, donde es necesario un diagnóstico completo de la instalación mediante mediciones precisas de la temperatura de los módulos.

Algunos de los patrones a identificar en la inspección simplificada son puntos calientes, cadenas y paneles completos en estado de circuito abierto o cortocircuito, o cajas de conexiones sobrecalentadas.

Estos fallos deben ser identificados mediante comparación con imágenes ejemplo y mediante el cálculo de la diferencia de temperatura con el resto del módulo o módulos, sin necesidad de conocer la temperatura absoluta. Algunas de las imágenes ejemplo se aprecian en la siguiente figura:

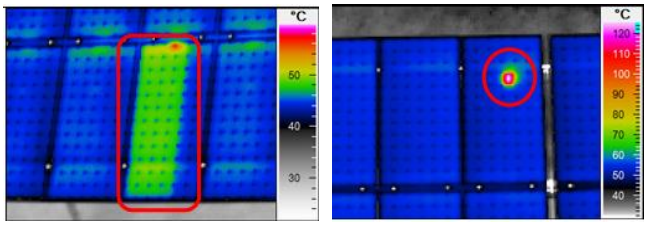

Figura 1. Patrones térmicos a identificar según IEC 62446-3[2]

La forma tradicional de llevar a cabo las inspecciones termográficas (tanto la detallada como la simplificada), y aún ampliamente utilizada, es mediante el uso de cámaras termográficas portátiles con las que se realizan inspecciones manuales. En ellas, un equipo de dos o más técnicos recorren la instalación solar midiendo la temperatura de los módulos en busca de alguna de las anomalías comentadas.

Este método sigue siendo el requerido por la normativa IEC para las inspecciones detalladas.

Sin embargo, en los últimos 10 años, el uso de drones equipados con cámaras termográficas ha ido sustituyendo el proceso manual de inspección de instalaciones fotovoltaicas. Diversas compañías ofrecen soluciones basadas en sobrevolar los módulos solares con drones obteniendo imágenes aéreas y perpendiculares a la superficie de estos. Los drones pueden barrer grandes áreas de paneles, capturando imágenes termográficas detalladas en tiempo real, lo que permite realizar inspecciones de manera más rápida, eficiente y con mayor cobertura en comparación con los métodos tradicionales.

A pesar de tener el contra del coste de la subcontratación de la tarea, o de la inversión inicial en drones y formación de pilotos, la rapidez y eficiencia convierten el uso de drones en el método hacia el cual se dirige la industria.

Las inspecciones termográficas con drones pueden realizarse de dos maneras principales. En el primer enfoque, los técnicos analizan las imágenes que los drones capturan, y aquellos paneles que presentan indicios de fallo son revisados manualmente. Este método combina la rapidez y amplitud de cobertura de los drones con la precisión y experiencia de los técnicos para confirmar y reparar las anomalías detectadas. De esta forma, se maximiza la eficiencia del mantenimiento al permitir intervenciones puntuales y específicas.

El segundo enfoque utiliza modelos de detección de fallos basados en inteligencia artificial (IA). Los drones capturan las imágenes termográficas y, o bien durante un post-procesamiento offline, o bien durante el propio tiempo de vuelo, se aplican algoritmos de IA que identifican automáticamente las anomalías térmicas. Estos modelos de IA generalmente son desarrollados por compañías para el propósito especifico de detección de averías de origen térmico, y su acceso al público está restringido. Con un buen entrenamiento y buenos ejemplos, los modelos pueden

aprender a detectar patrones y fallos con alta precisión, proporcionando reportes detallados y recomendaciones de mantenimiento sin necesidad de intervención humana directa. Aunque esta metodología oficialmente solo puede cumplir con inspecciones simplificadas, en la práctica pueden proporcionar a los gestores de los parques solares la información necesaria con la precisión suficiente para tomar decisiones correctivas. Este avance tecnológico permite una monitorización continua y automática de las instalaciones, optimizando la operación y mantenimiento de los sistemas fotovoltaicos.
El presente trabajo busca comprobar qué rendimiento pueden ofrecer los algoritmos de detección de objetos de código abierto para el diagnóstico de parques solares mediante el tratamiento automatizado de estas imágenes térmicas, como alternativa tanto a la inspección manual como a los modelos específicos y privados de las compañías de software.

B. Estudios previos

Existen estudios que aplican diversos algoritmos de detección de objetos para el análisis de paneles solares. El más destacado es el de Pathaka et al. [5], donde se utiliza un enfoque de *Deep Learning* para la localización de puntos calientes y clasificación de fallos en paneles solares mediante el procesamiento de imágenes térmicas. En este estudio, se comparan modelos de Machine Learning y Deep Learning, siendo estos últimos superiores en clasificación y detección de defectos. Entre los modelos de *Machine Learning* se utilizaron el *Random Forest*, XGBoost, SVM y *K-Means*.
Como modelos de Deep Learning destacaron ResNet-50, que alcanzó ResNet-50 alcanzó la puntuación F1 más alta del 85.37%, en la clasificación de fallos, y Faster R-CNN, que obtuvo un MAP del 67% en la identificación de las regiones de interés del panel defectuoso.
El estudio también destacó que la clasificación de fallos específicos en los paneles solares es más desafiante que simplemente identificar si un panel es defectuoso o no. Esto se debe a la variabilidad y complejidad de los distintos tipos de fallos que pueden ocurrir en los módulos fotovoltaicos.

C. Tecnología de reconocimiento de objetos basada en deep learning

Los algoritmos basados en deep learning para la clasificación y regresión de objetos presentan una estructura común, formada por una base o *backbone*, un cuello o *neck*, y una cabeza o *head*. La base está compuesta por una red CNN encargada de extraer características de las imágenes de entrada. El cuello busca la agregación de los mapas de características, combinando las salidas de diferentes escalas del *backbone*. Por último, la cabeza se encarga de la predicción final, compuesta por una confianza de clase y las coordenadas de la caja delimitadora.
En cuanto al enfoque o metodología para implementar estas tres estructuras, los algoritmos actuales pueden dividirse en dos categorías. La primera de ellas es el algoritmo de dos etapas o *two-stage detector*, representado por arquitecturas como R-CNN, Fast R-CNN y Faster R-CNN. Estos algoritmos consiguen realizar la detección en dos pasos. El primero consiste en una búsqueda selectiva para generar posibles regiones objetivo, y luego completar la clasificación y regresión de las cajas delimitadoras en dichas regiones. Este método tiene alta precisión, pero también limita la velocidad de detección.

El otro enfoque es el de una sola etapa o *one-stage detector*, representado principalmente por RetinaNet, SSD y YOLO. Los algoritmos de una sola etapa utilizan una única red para predecir directamente los cuadros delimitadores de los objetos y las puntuaciones de probabilidad de clase a partir de las imágenes originales. La velocidad de detección es superior al evitar la preselección de posibles regiones de interés.
Sin embargo, la precisión del algoritmo de una sola etapa para la detección de no es tan buena como la del algoritmo de dos etapas, especialmente para muestras difíciles.
Usar unos modelos u otros supone un compromiso entre precisión y velocidad de detección, que obliga a evaluar distintas arquitecturas para encontrar la óptima para la aplicación propuesta.

En este trabajo, se han utilizado y comparado los algoritmos Faster R-CNN (2-*stage*), YOLOv5 (*1-stage*). y RetinaNet (*1-stage*).

Faster R-CNN

Faster R-CNN es una arquitectura de dos etapas ampliamente utilizada por su alta precisión en la detección de objetos. El proceso de detección de Faster R-CNN se lleva a cabo en dos pasos. Primero, la red convolucional base genera mapas de características a partir de las imágenes de entrada, extrayendo las propiedades necesarias para la posterior detección de objetos. Posteriormente la estructura FPN o *Feature Pyramid network* combina por suma los mapas generados a diferentes escalas por la red base, ajustando las dimensiones. De esta forma, los mapas de características poseen la información semántica que aportan las capas más profundas junto con la resolución espacial de las capas más próximas. Estos mapas de características enriquecidos son alimentados a la Red de Propuesta de Regiones (RPN), encargada de producir regiones de interés RoI (Regions of Interest) que tienen una alta probabilidad de contener un objeto.
En la segunda etapa, estas propuestas de regiones se utilizan para realizar la clasificación y la regresión. Se aplica un RoI *pooling* para ajustar todas las regiones propuestas a un tamaño fijo. Luego, estas regiones ajustadas se pasan a través de redes neuronales totalmente conectadas.
La red de clasificación cuenta con una función de activación Softmax, que asigna probabilidades a cada clase, incluyendo la clase de fondo. En paralelo, una red de regresión refina las

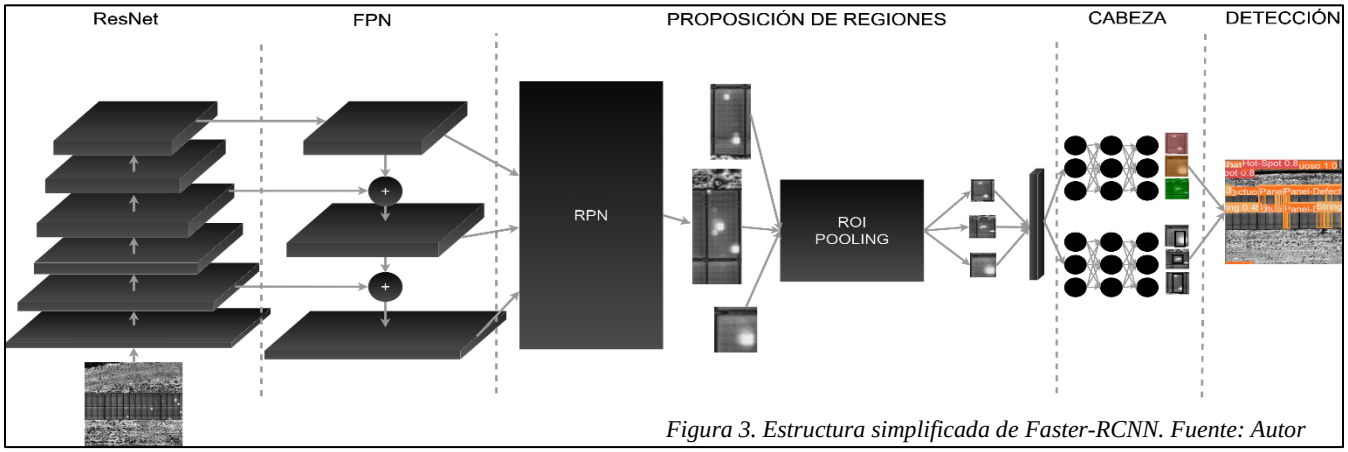

Figura 3. Estructura simplificada de Faster-RCNN. Fuente: Autor

coordenadas de los cuadros delimitadores ajustándolos para que se alineen mejor con los objetos detectados.

Esta combinación de clasificación y regresión permite a Faster R-CNN lograr una alta precisión en la detección de objetos, aunque a costa de una mayor complejidad computacional y menor velocidad en comparación con los algoritmos de una sola etapa.

YOLOv5

YOLOv5 es un algoritmo de detección de objetos de una sola etapa conocido por su rapidez y precisión. A diferencia de las arquitecturas de dos etapas como Faster R-CNN, YOLOv5 realiza la detección en un solo paso sin necesidad de generar regiones de interés.

El backbone de Yolov5 es la CNN CSPDarknet, basada en ResNet, que utiliza conexiones residuales para evitar la pérdida de gradientes. Esta CNN se encarga de extraer las características iniciales de las imágenes de entrada. CSPDarknet incorpora además convoluciones de punto y profundidad y una separación de caminos mediante los bloques CSP que reduce la redundancia y mejora la eficiencia computacional. Estas características permiten que el *backbone* procese la información de manera más rápida y eficiente.

El *neck* utiliza una combinación de Spatial Pyramid Pooling (SPP) y *Path Aggregation Network* (PANet) para combinar y enriquecer las características extraídas. SPP maneja objetos de diferentes tamaños y escalas al agregar características espaciales, mientras que PANet mejora la información de características de los niveles inferiores y superiores, concatenando los mapas de características de distintas escalas procedentes de la CNN base. PANet, de forma similar a la *Feature Pyramid Network* (FPN) utilizada en Faster R-CNN, optimiza la precisión y la localización de objetos pequeños y medianos, proporcionando una mejor representación de las características a diferentes niveles de la red.

Finalmente, la head consiste en varias capas de salida que predicen directamente las cajas delimitadoras, las clases de los objetos y las puntuaciones de confianza asociadas. La velocidad de YOLOv5 se debe a su diseño eficiente que evita el uso de RPN y ROI *pooling*, lo que simplifica el proceso de detección. A pesar de tener más parámetros que RetinaNet, YOLOv5 es más rápido debido a su arquitectura optimizada y a técnicas avanzadas de implementación, como el uso de

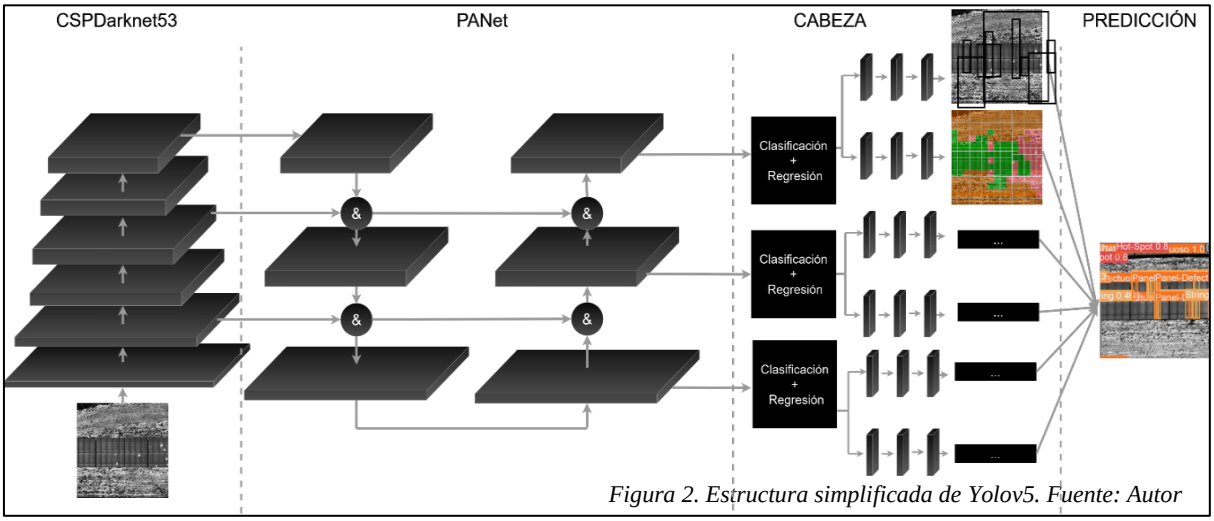

Figura 2. Estructura simplificada de Yolov5. Fuente: Autor

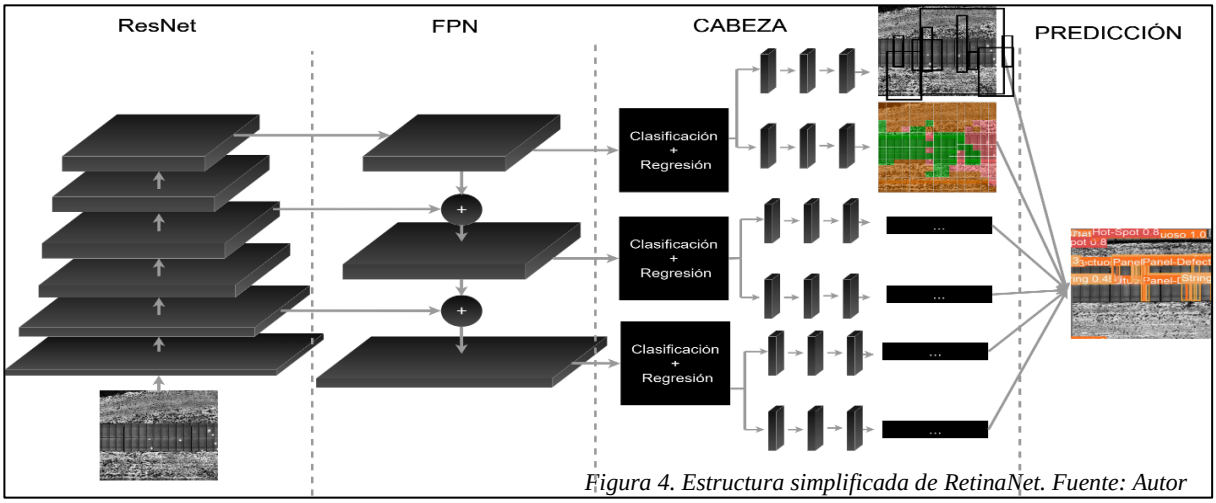

Figura 4. Estructura simplificada de RetinaNet. Fuente: Autor

convoluciones de punto y profundidad y la separación de caminos en su *backbone* CSPDarknet, lo que reduce la

redundancia y mejora la eficiencia computacional. Esta combinación de precisión y velocidad hace que YOLOv5 sea especialmente adecuado para aplicaciones en tiempo real.

RetinaNet

RetinaNet es un algoritmo de detección de objetos de una sola etapa que destaca por su capacidad para manejar desequilibrios de clases en los datos de entrenamiento. La estructura de RetinaNet se compone de tres partes principales: el *backbone*, el *neck* y el *head*. Como backbone utiliza ResNet para extraer características iniciales de las imágenes de entrada, proporcionando una base robusta para la detección. Luego, el cuello incorpora una *Feature Pyramid Network* (FPN) que combina características de múltiples escalas para mejorar la precisión y la resolución espacial de las detecciones. Finalmente, la cabeza de RetinaNet se divide en dos subredes independientes: una para la clasificación de objetos y otra para la regresión de las cajas delimitadoras. Una innovación clave de RetinaNet es la inclusión de la *Focal Loss* en la subred de clasificación, que asigna diferentes pesos a las muestras difíciles como forma de resolver el problema del desequilibrio de clases en la detección de objetos. Gracias a esta combinación de técnicas, RetinaNet logra una alta precisión en la detección de objetos, comparable a muchos algoritmos de dos etapas, pero con una velocidad de inferencia que permite su uso en aplicaciones en tiempo real.

III. METODOLOGÍA EMPLEADA

A. Entorno de trabajo

El código necesario para el desarrollo del proyecto está escrito en su totalidad en Python, lenguaje interpretado ampliamente utilizado en *Data Science* y *Machine Learning*. Se ha usado Anaconda para la gestión del entorno virtual, facilitando la administración de dependencias y librerías. Para la implementación de las arquitecturas de detección de objetos,

se ha optado por el *framework* PyTorch, conocido por su flexibilidad y eficiencia en cálculos tensoriales.
Se han utilizado además bibliotecas complementarias s para tareas como gestión y preprocesamiento de imágenes, aceleración de hardware con GPU o visualización de resultados.
En cuanto a hardware, el proyecto se ha realizado enteramente en local, haciendo uso de un PC personal con las siguientes características: OS, Win11; GPU, Nvidia RTX 4070 Super; CPU, AMD Ryzen 7 7800X3D; CPU @ 4.20 GHz. Esto ha permitido hacer uso de Cuda para acelerar los cálculos y aumentar la eficiencia de los entrenamientos.

B. Recopilación de imágenes

Debido a imposibilidad de crear de un dataset propio de imágenes térmicas de módulos defectuosos para este proyecto, estas han sido recopiladas de internet, de datasets públicos procedentes de fuentes como Flir, Mendeley Data, o Roboflow[4]. Se ha priorizado la selección imágenes aéreas y procedentes de fuentes de confianza, siempre garantizando

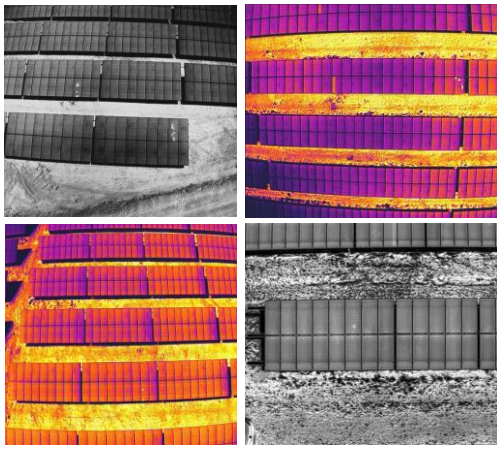

Figura 5. Ejemplo de imágenes térmicas utilizadas

que se cumplen los requisitos establecidos en la IEC 62446-3 acerca de la resolución mínima y ángulo de inclinación respecto a los módulos. Algunas de ellas son:

C. Preprocesado de las imágenes

Tras la recopilación de imágenes y antes del entrenamiento, se llevó a cabo un pre-procesado, que incluyó tareas como el escalado de las imágenes a un tamaño fijo de 640x640.
Además, como puede observarse en la Ilustración anterior, algunas de las imágenes recolectadas tenían una representación en escala de grises y otras tenían aplicado un mapa de color térmico. Para uniformizar el dataset se realizó la conversión a imágenes en escala de grises, es decir, en representación de 1 solo canal.

Dado que el método de reconocimiento de objetos usado en este experimento es un tipo de aprendizaje supervisado, es necesario obtener la información de etiquetado de las imágenes, que incluye la categoría del objeto y la ubicación en la imagen.
Par facilitar esta tarea de etiquetado de las imágenes, se ha hecho uso de la plataforma Roboflow [4] para mejorar la eficiencia del etiquetado en nuestro experimento. Esta plataforma presenta una buena interfaz gráfica interactiva con variedad de atajos de teclado y además permite exportar las etiquetas en los formatos requeridos por los modelos. De esta forma se crean las cajas delimitadoras que los modelos de *supervised learning* necesitan como referencia o *Ground truth*.

Como clases a detectar se incluyeron: 'Panel defectuoso', 'Punto Caliente/*Hot-Spot'*, 'Cadena/*String*' y 'Caja de conexiones'. Las tres últimas representan averías de diversos tipos y 'Panel defectuoso' detecta todos aquellos paneles que tienen al menos una de las tres averías presentes. Estas clases permiten detectar todas las anomalías referidas en la normativa IEC 62446-3.

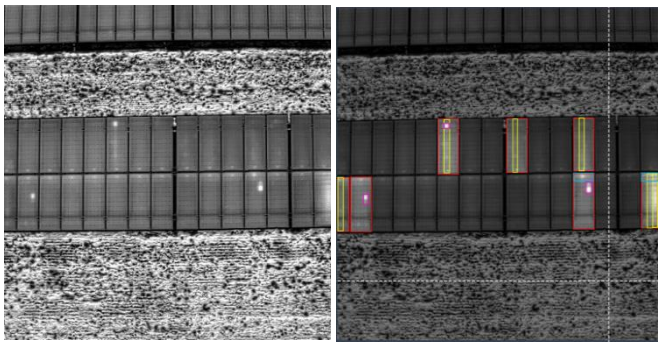

Figura 6. Ejemplo de etiquetado con Roboflow. Fuente: Autor

Tras el etiquetado, se observó que el dataset presentaba desequilibrios en el número de muestras, especialmente en la clase 'Caja de Conexiones', que aparece un número de veces muy inferior a las demás. A pesar de que es recomendable evitar desequilibrios, se mantuvo para evaluar el desempeño de los diferentes modelos ante una situación común en detección multiclase como es esta.

Por último, se recurrió a técnicas de aumento de datos, para ampliar el reducido tamaño del dataset. De esta forma, se generaron versiones alternativas de las imágenes originales aplicando transformaciones como la rotación o la introducción de ruido. Con ello, Se consiguió pasar de un dataset de 220 imágenes a uno de 440.

D. Diseño de los modelos

Para las tres arquitecturas, se usaron modelos preentrenados con el conjunto de datos Ms. COCO, que contiene 200.000 imágenes con 1.5M de objetos anotados pertenecientes a 80 clases. y luego se ajustaron al dataset específico de imágenes térmicas de paneles solares. El conjunto de datos se dividió según la proporción 90% para entrenamiento y 10% para test. Dentro del 90% de entrenamiento, se aplicó validación cruzada para maximizar el uso de un dataset limitado.

Para comparar los resultados de los tres modelos de detección de objetos, se aplicaron una serie de métricas estándar comúnmente utilizadas para evaluar estos modelos. Las posibles salidas de la red, tras la comparación con las etiquetas reales pueden clasificarse en: verdaderos positivos (TP), falsos positivos (FP), falsos negativos (FN) y verdaderos negativos (TN). Si el objeto detectado se clasifica correctamente, las coordenadas centrales de la caja delimitadora de detección y las dimensiones de la caja están dentro de los límites tolerables, el resultado de detección se registra como TP. FP se refiere a un error en la clasificación del objeto detectado o en la posición o dimensiones del marco. Por último, cualquier objeto presente en las etiquetas reales o *Ground truth* y para el que el modelo no prediga una caja apropiada de la clase correcta será considerado un FN.
Los TN recogen todas aquellas cajas delimitadoras que el modelo ha predicho como vacías, de forma correcta.

Con estos valores se calcula la precisión, definida como la proporción de predicciones correctas de entre todas las predicciones de la presencia del objeto, y se define como:

$$Precisión = \frac{TP}{TP + FP}$$

Modelos con precisión alta son fiables cuando predicen un objeto en la imagen, pero para ello pueden dejar pasar muchos objetos, detectando únicamente aquellos más evidentes o fáciles.
El *recall* o sensibilidad, por su parte, mide la proporción de objetos que el modelo detecta de entre el total de muestras que contienen este objeto, y se define como:

Figura 7. Número de muestras por clase. [4]

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

Modelos con sensibilidad alta tienden a no pasar por alto ninguno de los objetos a detectar, pero para ello pueden clasificar muchas áreas vacías de las imágenes como una de las clases.

Es por esto que es necesario combinar precisión y sensibilidad en un único método evaluación del rendimiento, como el AP o *Average Precision*. El cálculo del AP consiste en calcular el área bajo la curva de precisión y sensibilidad, que se traza para distintos umbrales de confianza en las predicciones hechas durante la validación realizada después de cada época de entrenamiento.

$$AP = \int_0^1 p(r)dr$$

El AP se calcula para cada clase a predecir, y con todos estos valores se calcula el mAP (*mean Average Precision*) como la media aritmética de los AP. Tanto el mAP como los AP individuales se han utilizado para evaluar el rendimiento de los modelos.

$$mAP = \frac{1}{n}\sum_{i=1}^{n} AP_i$$

A la par que la precisión, es importante evaluar la velocidad de inferencia de los modelos para determinar su viabilidad en aplicaciones en tiempo real. Como medida de la velocidad de inferencia de los modelos se ha utilizado tanto el tiempo necesario para la predicción por imagen, como su inversa los FPS.
Dado que esta medida es extremadamente dependiente del hardware utilizado se han calculado haciendo uso de la aceleración con GPU y también sin ella.

E. Entrenamientos realizados

En primer lugar, se realizaron dos entrenamientos para comparar un modelo de cada arquitectura, y así comprobar su rendimiento en cuanto a precisión y velocidad de inferencia. Dado que Yolov5 tiene diferentes versiones (s, m, l, x), con diferentes profundidades, y Faster R-CNN y Retina pueden ser implementados con diferentes *backbones*, se han agrupado los modelos elegidos en dos experimentos, de tal forma que los modelos comparados presenten un número de similar de parámetros totales.
Primer entrenamiento:

- Yolov5l con CSPDarknet53 y PANet: 46M de parámetros
- Faster R-CNN con ResNet50 y FPN: 41M de parámetros
- RetinaNet con ResNet50 y FPN: 38M de parámetros.

Segundo entrenamiento:

- Yolov5x con CSPDarknet53 y PANet: 87M de parámetros
- Faster R-CNN con ResNet101 y FPN: 60M de parámetros
- RetinaNet con ResNet101 y FPN: 63M de parámetros.

Como se ha mencionado, se ha hecho uso de validación cruzada. El 90% del dataset original, compuesto por 220 imágenes, ha sido dividido en 5 lotes. Para cada una de las dos versiones de cada arquitectura se han entrenado 5 modelos, utilizando 4 de los 5 lotes como datos de entrenamiento y el restante como validación. Los datos de entrenamiento y validación se han ido rotando entre los 5 modelos, asegurando que cada lote se utilice una vez como conjunto de validación y que el aumento de datos solo afecta al lote de entrenamiento en cada caso.
Los hiperparámetros elegidos para estos entrenamientos fueron:

TABLA I
HIPERPARÁMETROS UTILIZADOS

Hiperparámetro	**Valor**
Tasa de aprendizaje / learning rate	0.01 – 0.0001 (lineal)
Momentum	0.9
Tamaño del lote/ Batch Size	10
Numero de épocas / Epochs	500
Decaimiento de pesos / Weight decay	0.001
Tamaño de las anclas /Anchor Size	[0.5, 1, 2]
Paciencia / Early Stopping	50

Posteriormente, se realzaron entrenamientos separados a cada arquitectura tratando de ajustar los hiperparámetros para obtener los mejores resultados para esta aplicación.

IV. RESULTADOS OBTENIDOS

F. Comparación de Arquitecturas

El entrenamiento del primer conjunto de modelos arrojó los siguientes resultados en cuanto a AP para cada clase y mAP total:

TABLA II
RESULTADOS AP

Modelo	Media mAP	Punto Caliente	Caja Conexiones	Panel Defectuoso	String
F-RCNN /ResNet50	0.826	0.846	0.771	0.909	0.775
YOLOv5l	0.781	0.822	0.751	0.895	0.661
RetinaNet /ResNet50	0.793	0.785	0.769	0.884	0.735

Y para el segundo conjunto:

TABLA III
RESULTADOS AP

Modelo	Media mAP	Punto Caliente	Caja Conexiones	Panel Defectuoso	String
F-RCNN /ResNet101	0.820	0.849	0.760	0.904	0.762
YOLOv5x	0.743	0.828	0.798	0.893	0.489
RetinaNet /ResNet101	0.785	0.788	0.817	0.890	0.645

Las tablas completas con los resultados para los 5 entrenamientos de cada modelo se muestran al final del documento. En ellas, puede observarse que la variabilidad en los resultados obtenidos a través de los distintos pliegues (*k-folds*) es notable. Esto puede atribuirse al tamaño reducido del conjunto de datos (220 imágenes) y al hecho de que, aunque cada pliegue comparte aproximadamente el 80% del conjunto de entrenamiento, las diferencias en el 20% restante pueden introducir variaciones significativas en los resultados. Esto subraya la importancia de ampliar el conjunto de datos para obtener resultados más consistentes y representativos.

Faster R-CNN con ResNet50 mostró la mayor precisión general (mAP) entre todas las arquitecturas probadas, destacándose en clases como "Panel Defectuoso" y "Punto Caliente". En la clase "Caja de Conexiones", que está subrepresentada, Faster R-CNN mantuvo una precisión razonablemente alta, reflejando su capacidad para generalizar bien incluso con datos limitados. La clase *String*, que presenta cajas delimitadoras alargadas, y que es la más difícil de disitnguir con respecto al fondo, también fue detectada de manera efectiva, aunque con más variabilidad.

YOLOv5l presentó un rendimiento razonable en términos de mAP, con una media que varía entre 0.716 y 0.839. Es notablemente eficaz en la detección de "Panel Defectuoso", con mAPs superiores a 0.900 en varios pliegues. Sin embargo, la clase *String* presentó mayores problemas, con APs que oscilan entre 0.566 y 0.768, lo cual se puede atribuir a la dificultad de detectar cajas muy alargadas y de área pequeña. Analizando las métricas de pérdidas se observó que la pérdida de *objectness* es con diferencia la más elevada en validación, es decir, el modelo tiene mayor dificultad en diferenciar los objetos del fondo más que en clasificarlos y localizarlos.

RetinaNet con ResNet50 ofreció un rendimiento equilibrado, con una media de mAP que varía entre 0.711 y 0.829. Aunque ligeramente inferior en precisión general comparado con Faster R-CNN y YOLOv5, RetinaNet mostró consistencia y menos variabilidad. En la detección de "Caja de Conexiones" y *String*, RetinaNet superó a YOLOv5, debido a su uso de la pérdida focal, que maneja mejor los desequilibrios en las clases. La capacidad de RetinaNet para manejar mejor las muestras difíciles y subrepresentadas le da una ventaja en estas categorías específicas.

Los modelos YOLOv5x y Faster R-CNN con ResNet101, a pesar de su mayor profundidad y número de parámetros, mostraron un rendimiento inferior en comparación con sus versiones más ligeras, siendo las diferencias más pequeñas en el caso de Faster -RCNN. YOLOv5x obtuvo medias de mAP que oscilan entre 0.677 y 0.824, con una notable disminución en la clase *String*, donde los APs fueron consistentemente bajos (por debajo de 0.553). RetinaNet con ResNet101 tuvo un mAP que varía entre 0.700 y 0.835, siendo más efectivo en la detección de "Punto Caliente" y "Caja de Conexiones". La reducción de rendimiento en estos modelos más complejos puede atribuirse al tamaño limitado del conjunto de datos, que no proporciona suficiente información para aprovechar completamente las capacidades de estas arquitecturas más profundas, conduciendo a un sobreajuste que resulta en un rendimiento inferior. Este fenómeno puede observarse en las curvas de pérdidas, que descienden consistentemente para los lotes de entrenamiento, mientras que para las imágenes de validación alcanzan rápidamente un mínimo y comienzan a crecer de vuelta.

Una vez comparados los resultados, se entrenó un modelo de cada arquitectura comentada, esta vez utilizando el conjunto de los 5 pliegues (90% del dataset total) como lote de entrenamiento. Sobre el lote de test, que comprende el 10% restante del dataset, se realizó la inferencia, para medir la velocidad en la predicción de los modelos. Los resultados obtenidos para el tiempo de inferencia y los fotogramas por segundo (FPS) para cada modelo fueron los siguientes:

TABLA IV
RESULTADOS INFERENCIA

Modelo	Tiempo/Imagen (ms)	FPS
YOLOv5l (GPU)	12,91	77,46
F-RCNN / ResNet50 (GPU)	32,63	30,64
RetinaNet / ResNet50 (GPU)	27,68	36,13
YOLOv5l (CPU)	295,36	3,39
F-RCNN / ResNet50 (CPU)	574,97	1,74
RetinaNet / ResNet50 (CPU)	412,14	2,43

TABLA V
RESULTADOS INFERENCIA

Modelo	Tiempo/Imagen (ms)	FPS
YOLOv5x (GPU)	19,98	50,05
F-RCNN / ResNet101 (GPU)	48,93	20,44
RetinaNet / ResNet101 (GPU)	42,27	23,66
YOLOv5x (CPU)	500,66	2,00
F-RCNN / ResNet101 (CPU)	695,38	1,44
RetinaNet / ResNet101 (CPU)	627,37	1,59

En este caso, los resultados son los esperados. Yolov5, que alcanza hasta los 77 FPS, es notablemente más rápido en las predicciones, llegando a doblar la velocidad de inferencia de RetinaNet en ambos conjuntos. Este resultado confirma la eficiencia de YOLOv5 en aplicaciones de tiempo real, donde la rapidez en la detección sea el elemento crítico.
El modelo Faster R-CNN con ResNet50, que obtuvo los mejores resultados de precisión, presentó un tiempo promedio de inferencia por imagen de 32.63 milisegundos, con una tasa de 30.64 FPS. Esta velocidad sigue siendo aplicable en detección de objetos en vídeo a tiempo real, pero prueba como la arquitectura, pero es significativamente más lenta en comparación con YOLOv5l.
RetinaNet con ResNet50 ofreció un rendimiento equilibrado, con un tiempo intermedio, aunque más cercano a los tiempos de la arquitectura de dos pasos que a Yolo.
Un resumen del rendimiento de todos los modelos, en precisión y velocidad puede observarse en la siguiente figura.

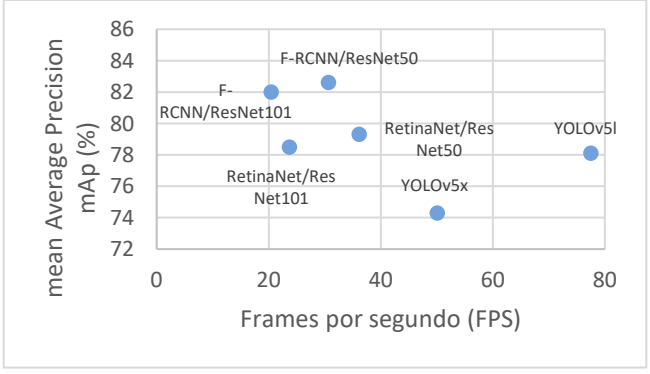

Figura 8. Comparación modelos. Fuente: Autor

G. Ajuste de Hiperparámetros

Tras la comparación inicial de los modelos, se decidió intentar afinar los hiperparámetros de cada arquitectura por separado. Este enfoque se adoptó para maximizar el rendimiento y obtener una mejor generalización de los modelos, considerando las limitaciones y características específicas del conjunto de datos. Durante este proceso y por simplicidad, únicamente se consideró el primer pliegue o *k-fold*, buscando mejorar los resultados que obtuvo para cada arquitectura durante el estudio inicial.

Para ajustar los hiperparámetros, primero se estudiaron las curvas de pérdidas y mAP obtenidas durante la comparación.
En primer lugar, se observa los modelos no llegaban a las 500 épocas antes de estancarse. En base a esto, se ha experimentado con rebajas en la tasa de aprendizaje (learning rate), manteniendo un decrecimiento lineal y también implementando un LRonPlateau.

También se ha iterado con valores de decaimiento de pesos (*weight decay*) y *momentum*.
Además, dado el rendimiento deficiente en la clase *String*, se probó a modificar la relación de aspecto de las anclas, de forma que sean más alargadas. Para ello, se aplicó el algoritmo de agrupación *K-means*, sobre todas las cajas de las etiquetas.
Atendiendo a las funciones de pérdidas, se puede observar diferentes comportamientos en los tres modelos. En Yolov5 la pérdida de *objectness* (diferenciación de objeto con respecto al fondo) está siendo la mayor contribuyente a la pérdida total. En RetinaNet, que no tiene perdida de *objectness* como tal, se observa esto en la perdida de clasificación que agrupa lo que en Yolov5 y Faster R-CNN son las pérdidas de *objectness* y clasificación. En Faster R-CNN la función de pérdidas está más distribuida entre sus componentes, aunque son las pérdidas de detección y clasificación de objetos las que antes comienzan a crecer en la validación.
Puede deducirse que el problema principal que experimentan los modelos es distinguir un objeto con respecto al fondo, más que la clasificación o el posicionamiento de las cajas delimitadoras.
Para paliar este efecto, se decidió modificar las ganancias de las pérdidas de cada modelo.
Para Yolov5 la función de perdidas puede definirse como:

$$L_{total} = \lambda_{box} * L_{box} + \lambda_{obj} * L_{obj} + \lambda_{cls} * L_{cls}$$

Donde cada termino hace referencia a la pérdida por error en las coordenadas de la caja delimitadora, error en la detección del objeto respecto al fondo y error en la clasificación en una de las clases, cada una multiplicada por una ganancia que mide su influencia en la pérdida final. Para el caso de Faster R-CNN, la pérdida total es:

$$L_{total} = \lambda_{obj} L_{clsRpn} + \lambda_{boxrpn} L_{regRpn} + \lambda_{cls} L_{clsDet} + \lambda_{boxdet} L_{regDet}$$

En este caso hay cuatro términos, correspondientes a la clasificación de y regresión de las regiones propuestas por la RPN y posteriormente a la clasificación y corrección de las cajas por las subredes densas.
Para RetinaNet:

$$L_{total} = \lambda_{cls} * L_{cls} + \lambda_{box} * L_{box}$$

Siendo *Lcls* la función de pérdida local que sustituye a la entropía cruzada tradicional.

Tras numerosas iteraciones variando los hiperparámetros comentados, los mejores resultados para cada arquitectura fueron:

TABLA VI
RESULTADOS AP – FASTER R-CNN

Modelo	Media mAP	Punto Caliente	Caja Conexiones	Panel Defectuoso	String
F-RCNN /ResNet50	0.838	0.862	0.780	0.913	0.797

Para unos hiperparámetros:

TABLA VII
HIPERPARÁMETROS UTILIZADOS

Hiperparámetro	**Valor**
Tasa de aprendizaje / learning rate	LRonPlateau(0.001/0.75 after 30 Epochs)
Optimizador / Optimizer	SGD – Momentum 0.85
Tamaño del lote/ Batch Size	16
Numero de épocas / Epochs	500
Decaimiento de pesos / Weight decay	0.005
Ganancia en la detección de objetos (λ_{obj})	1.5
Ganancia en la clasificación de objetos (λ_{cls})	0.8
Relación de aspecto /Anchor Aspect Ratio	['0.918', '0.767', '2.678'] ['0.129', '0.297', '0.117'] ['1.186', '0.502', '0.400'] ['0.390', '0.386', '0.870']
Paciencia / Early Stopping	100

TABLA VIII
RESULTADOS AP – YOLOV5

Modelo	Media mAP	Punto Caliente	Caja Conexiones	Panel Defectuoso	String
YOLOv5l	0.794	0.831	0.762	0.896	0.688

Para unos hiperparámetros:

TABLA IX
HIPERPARÁMETROS UTILIZADOS

Hiperparámetro	**Valor**
Tasa de aprendizaje / learning rate	LRonPlateau(0.005/0.75 after 30 Epochs)
Optimizador / Optimizer	SGD – Momentum 0.9
Tamaño del lote/ Batch Size	16
Numero de épocas / Epochs	500
Decaimiento de pesos / Weight decay	0.0025
Ganancia en la detección de objetos (λ_{obj})	1.2
Ganancia en la clasificación de objetos (λ_{cls})	0.75
Relación de aspecto /Anchor Aspect Ratio	['0.918', '0.767', '2.678'] ['0.129', '0.297', '1.200'] ['1.186', '0.452', '0.870']
Paciencia / Early Stopping	100

TABLA X
RESULTADOS AP – RETINANET

Modelo	Media mAP	Punto Caliente	Caja Conexiones	Panel Defectuoso	String
RetinaNet /ResNet50	0.805	0.801	0.783	0.883	0.752

Para unos hiperparámetros:

TABLA XI
HIPERPARÁMETROS UTILIZADOS

Hiperparámetro	**Valor**
Tasa de aprendizaje / learning rate	LRonPlateau(0.001/0.75 after 30 Epochs)
Optimizador / Optimizer	SGD – Momentum 0.9
Tamaño del lote/ Batch Size	16
Numero de épocas / Epochs	500
Decaimiento de pesos / Weight decay	0.0025
Ganancia en la clasificación de objetos (λ_{cls})	1.15
Relación de aspecto /Anchor Aspect Ratio	['0.400', '0.976', '0.614'] ['2.678', '0.150', '0.862'] ['0.373', '0.117', '1.242'] ['0.889','0.498', '0.390'] ['0.386', '0.880','0.128']
Paciencia / Early Stopping	100

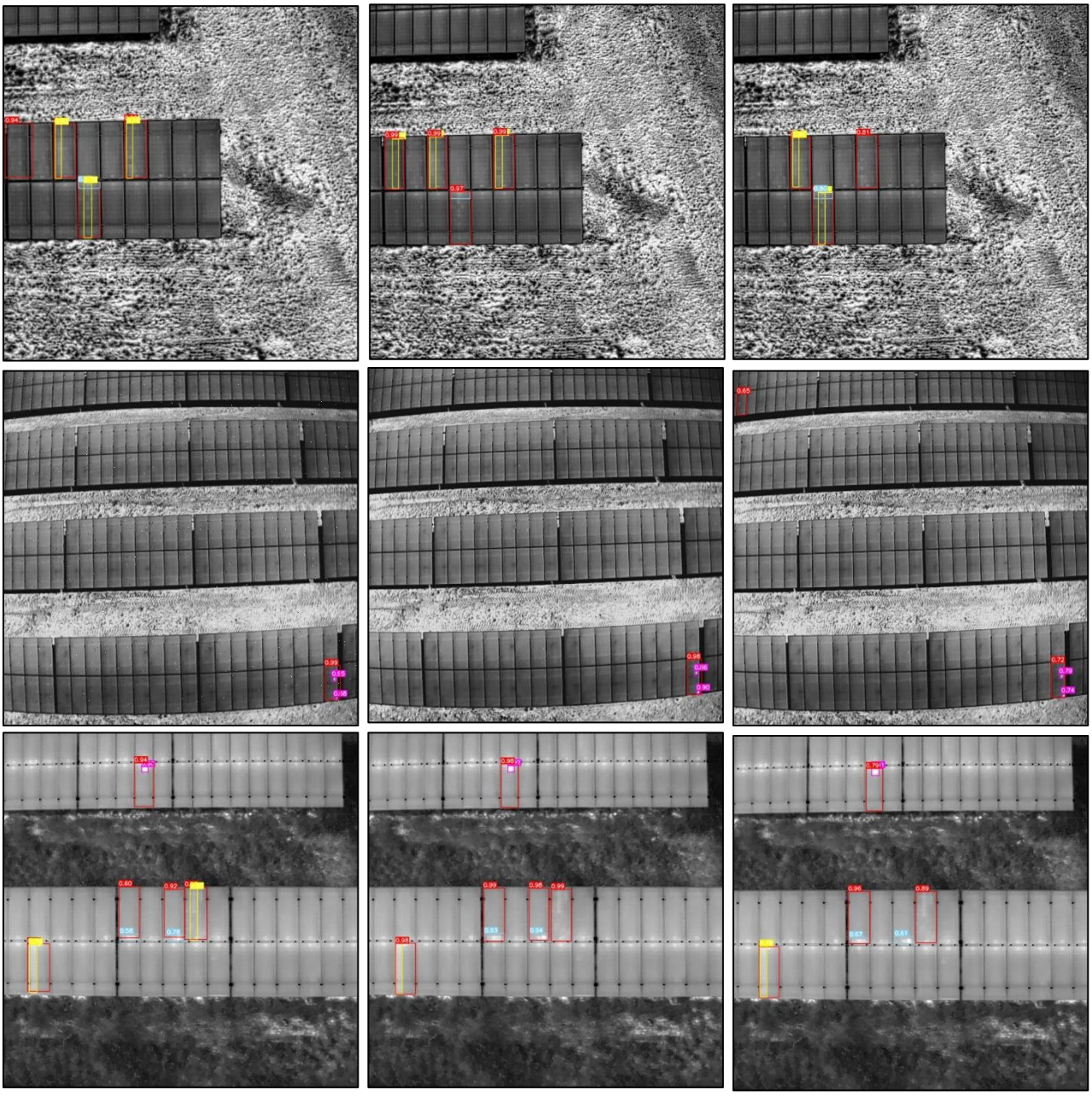

Figura 9. Resultados de la inferencia de los tres modelos.

H. Inferencia

Tras el refinado de los modelos mediante el ajuste de los hiperparámetros, puede verse su capacidad de detección en las muestras del test set apreciables en la Figura 9, donde en la primera columna se muestran los resultados de Faster R-CNN, en la segunda los de YOLOv5 y en la tercera los de RetinaNet. Se aplicó NMS (*Non-Max Suppression*) para mantener únicamente la mejor predicción para cada objeto detectado.

En rojo los modelos tratan de predecir los paneles defectuosos, en amarillo las cadenas averiadas, en azul las cajas de conexiones y en morado los puntos calientes.

V. CONCLUSIONES

El proyecto concluye que los modelos de detección automatizada de objetos son una alternativa eficaz para el proceso de mantenimiento de instalaciones fotovoltaicas, cumpliendo con la normativa IEC 62446-3 y permitiendo reducir la necesidad de intervención humana o la dependencia de las compañías propietarias de los modelos de análisis mediante IA. Los modelos evaluados han demostrado ser efectivos para la detección de anomalías térmicas en paneles

solares, presentando diferentes capacidades en cuanto a velocidad de inferencia y precisión.

Para un diagnóstico común, el modelo Faster R-CNN es el más adecuado, ya que se suelen utilizar estrategias de post-procesado offline de las imágenes capturadas y no detección en tiempo real.
Esta arquitectura permite identificar con precisión adecuada tanto los paneles defectuosos, como los elementos causantes dentro de cada módulo.
Si se requiere detección en tiempo real, YOLOv5 ofrece un mayor margen de desarrollo debido a su rapidez de inferencia, proporcionando precisión adecuada en la detección de paneles defectuosos, aunque no es tan fiable en la localización de las causas.
RetinaNet, por su parte, mejora significativamente el rendimiento de Yolov5 en las clases difíciles, y se encuadra entre los modelos anteriores en capacidad de detección y velocidad de inferencia, pudiendo ser una alternativa eficaz si se busca el equilibrio entre ambos factores.

Esta tecnología basada en *Deep Learning*, al ser de código abierto y fácilmente desplegable, democratiza el acceso a soluciones avanzadas, permitiendo a empresas de diferentes tamaños y sectores mejorar su eficiencia, reducir costos y aumentar la fiabilidad de sus operaciones.
La implementación en el diagnóstico de paneles solares es un claro ejemplo de cómo estas técnicas pueden optimizar la gestión de recursos energéticos y contribuir a un futuro industrial más sostenible y eficiente.

Como próximos pasos, se recomienda ampliar el conjunto de datos para incluir una mayor variedad de imágenes térmicas capturadas bajo diferentes condiciones ambientales y operativas. Dado el tamaño reducido del conjunto de entrenamiento actual, el ajuste de hiperparámetros ha tenido una influencia limitada en el rendimiento; por lo tanto, obtener más muestras sería preferible para mejorar la robustez y precisión de los modelos. Con un conjunto de datos más extenso, que contenga numerosos ejemplos de cada clase, se podría alcanzar un nivel de inspección más detallado, mejorando así la eficacia y eficiencia en las operaciones de mantenimiento de las instalaciones fotovoltaicas.

VI. REFERENCIAS

[1] Generación solar fotovoltaica en España 2010-2023. (2024, May 22). Statista. https://es.statista.com/estadisticas/1004390/generacion-solar-fotovoltaica-en-espana/

[2] International Electrotechnical Commission (IEC), 2017. IEC 62446-3: Photovoltaic (PV) systems - Requirements for testing, documentation and maintenance - Part 3: Photovoltaic modules and plants - Outdoor infrared thermography. Geneva: IEC.

[3] Pathak, S.P., Patil, S., and Patel, S. (2022). Solar panel hotspot localization and fault classification using deep learning approach. *Procedia Computer Science*, 204, pp.698-705. Available at: https://www.sciencedirect.com/science/article/pii/S1877050922008225

[4] Roboflow. (n.d.). Roboflow: Accelerate Computer Vision. Available at: https://app.roboflow.com/
Elsevier. (2024). Mendeley Data. Available at: https://data.mendeley.com/
Teledyne FLIR LLC. (2024). FLIR | The World's Sixth Sense. Available at: https://www.flir.com/

[5] Ren, S., He, K., Girshick, R., & Sun, J. (2015) 'Faster R-CNN: Towards real-time object detection with region proposal networks', Advances in neural information processing systems (NeurIPS).

[6] Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020) 'YOLOv4: Optimal speed and accuracy of object detection', arXiv:2004.10934.

[7] Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollár, P. (2017) 'Focal loss for dense object detection', IEEE transactions on pattern analysis and machine intelligence.

[8] Tan, L., Huangfu, T., Wu, L. & Chen, W. (2021) 'Comparison of RetinaNet, SSD, and YOLO v3 for real-time pill identification', *BMC Medical Informatics and Decision Making*, 21(1), p. 324. Available at: https://doi.org/10.1186/s12911-021-01691-8

VII. ANEXOS

I. Abreviaturas utilizadas

CPU: Central Processing Unit. Unidad de procesamiento central.

CNN: Convolutional Neural Network. Red neuronal convolucional.

FPN: Feature Pyramid Network. Red de pirámide de características.

FPS: Frames Per Second. imágenes por segundo, utilizado para medir la velocidad de inferencia de los modelos.

GPU: Graphics Processing Unit. Unidad de procesamiento gráfico.

IoT: Internet of Things. Internet de las cosas, red de dispositivos físicos interconectados.

mAP: Mean Average Precision. Media de las precisiones promedio calculadas para cada clase.

PANet: Path Aggregation Network. Red de agregación de caminos utilizada en modelos de detección de objetos para mejorar la información de características.

RPN: Region Proposal Network. Red de propuestas de regiones utilizada en modelos de detección de objetos para generar regiones de interés.
TP: True Positive. Verdadero positivo, cuando un objeto es correctamente detectado y clasificado.

FP: False Positive. Falso positivo, cuando un objeto es incorrectamente detectado o clasificado.

FN: False Negative. Falso negativo, cuando un objeto no es detectado, aunque esté presente.

TN: True Negative. Verdadero negativo, no aplicable en detección de objetos.

YOLO: You Only Look Once. Algoritmo de detección de objetos de una sola etapa conocido por su rapidez y precisión.

NMS: Non-Maximum Suppression. Supresión de no máximos, algoritmo para eliminar las detecciones redundantes.

J. Resultados de la comparación entre modelos

Modelo	Lote de validación	Media (mAP)	Punto Caliente	Caja de conexiones	Panel Defectuoso	String
YOLOv5l	1	0.791	0.847	0.785	0.900	0.629
	2	0.716	0.731	0.722	0.854	0.725
	3	0.723	0.715	0.738	0.875	0.546
	4	0.839	0.886	0.767	0.935	0.740
	5	0.836	0.932	0.745	0.948	0.663
F-RCNN /ResNet50	1	0.845	0.87	0.798	0.93	0.779
	2	0.829	0.852	0.810	0.915	0.739
	3	0.742	0.752	0.626	0.784	0.784
	4	0.858	0.876	0.836	0.955	0.766
	5	0.845	0.878	0.768	0.942	0.794
RetinaNet / ResNet50	1	0.811	0.819	0.792	0.930	0.703
	2	0.762	0.764	0.729	0.905	0.648
	3	0.711	0.663	0.724	0.806	0.655
	4	0.829	0.854	0.820	0.939	0.705
	5	0.815	0.825	0.752	0.926	0.755

TABLA IX
RESULTADOS AP PARA LOS 5 PLIEGUES (FOLDS)

Modelo	Lote de validación	Media (mAP)	Punto Caliente	Caja de conexiones	Panel Defectuoso	String
YOLOv5x	1	0.729	0.811	0.755	0.919	0.431
	2	0.677	0.719	0.688	0.848	0.453
	3	0.692	0.757	0.881	0.869	0.459
	4	0.824	0.924	0.904	0.920	0.547
	5	0.795	0.93	0.764	0.933	0.553
F-RCNN /ResNet101	1	0.840	0.878	0.789	0.925	0.768
	2	0.827	0.856	0.799	0.920	0.732
	3	0.742	0.752	0.626	0.784	0.784
	4	0.855	0.881	0.836	0.953	0.749
	5	0.837	0.882	0.750	0.940	0.778
RetinaNet / ResNet101	1	0.778	0.774	0.805	0.924	0.608
	2	0.700	0.690	0.694	0.814	0.600
	3	0.835	0.842	0.905	0.921	0.672
	4	0.821	0.86	0.897	0.858	0.669
	5	0.791	0.776	0.784	0.928	0.675

TABLA XIII
RESULTADOS AP PARA LOS 5 PLIEGUES (FOLDS)

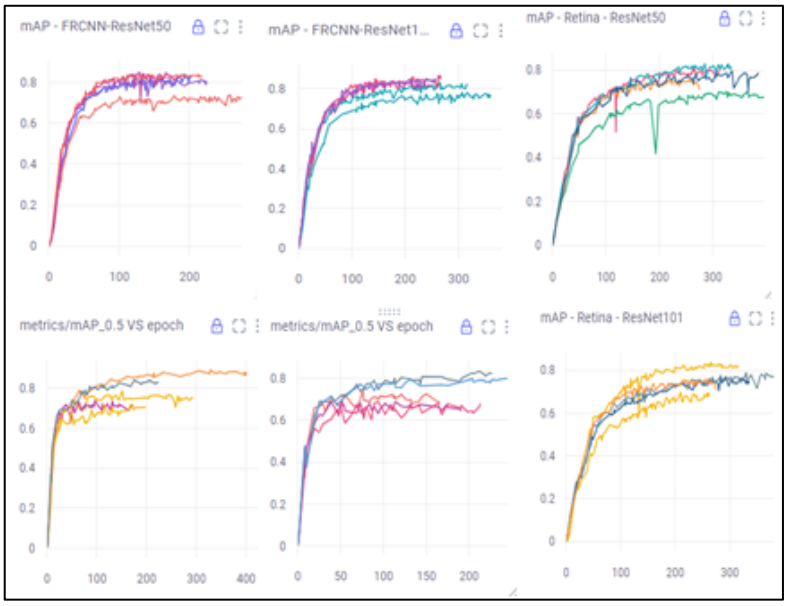

Figura 10. mAPs de la comparación de modelos

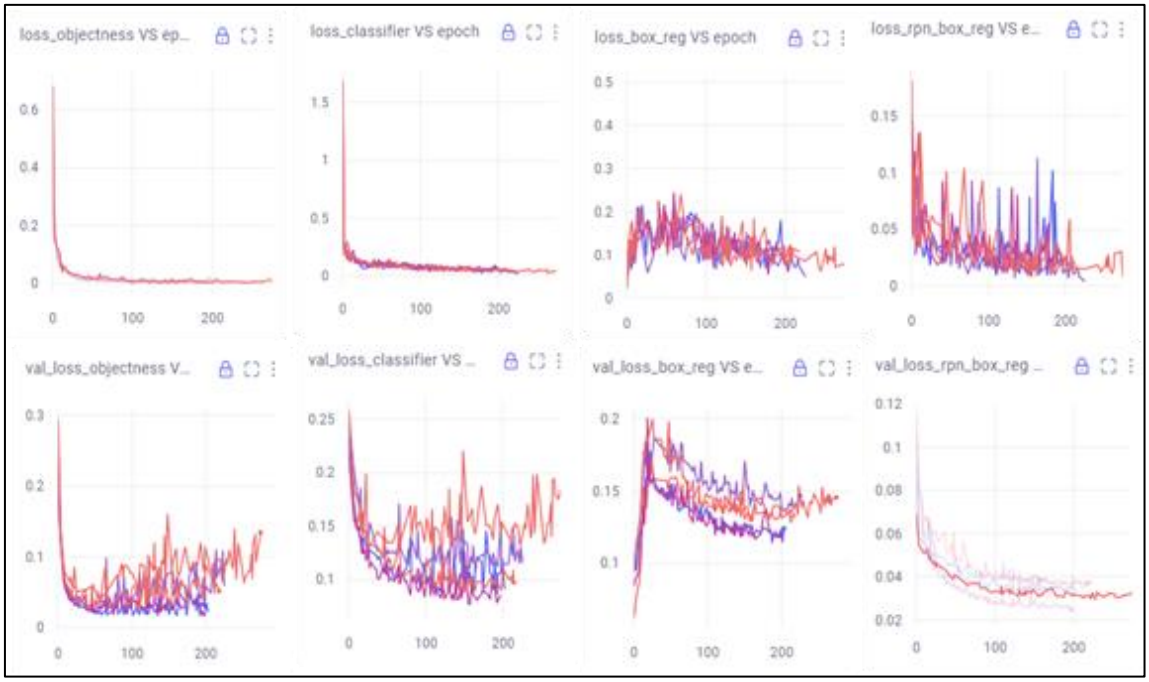

Figura 11. Funciones de pérdida para Faster R-CNN / ResNet50

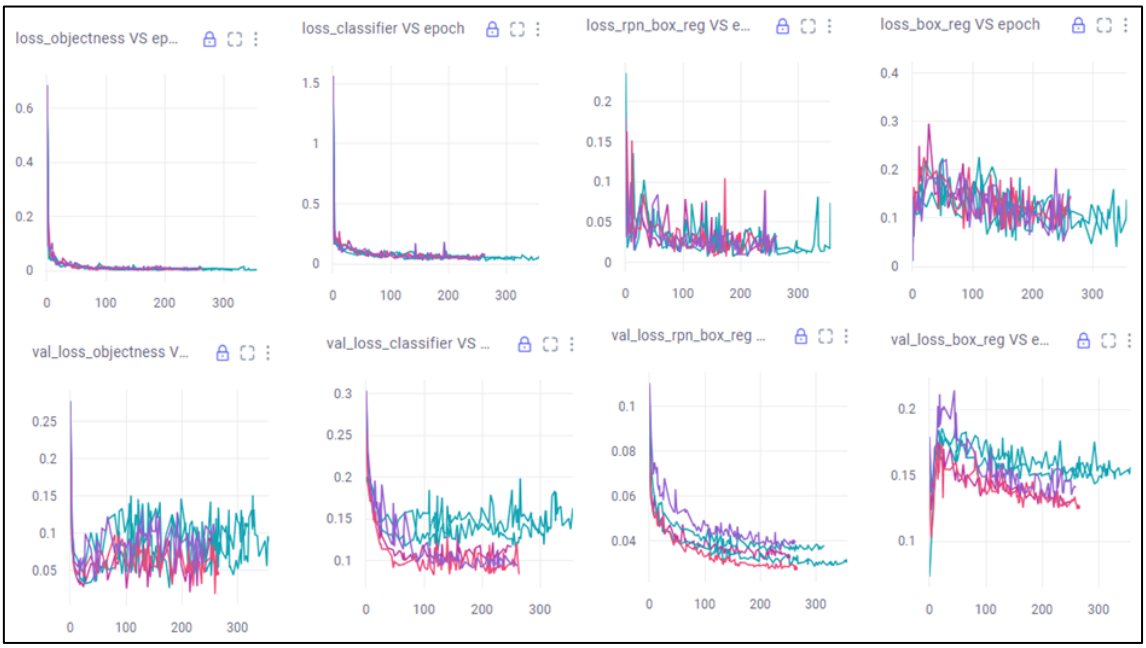

Figura 12. Funciones de pérdida para Faster R-CNN / ResNet101

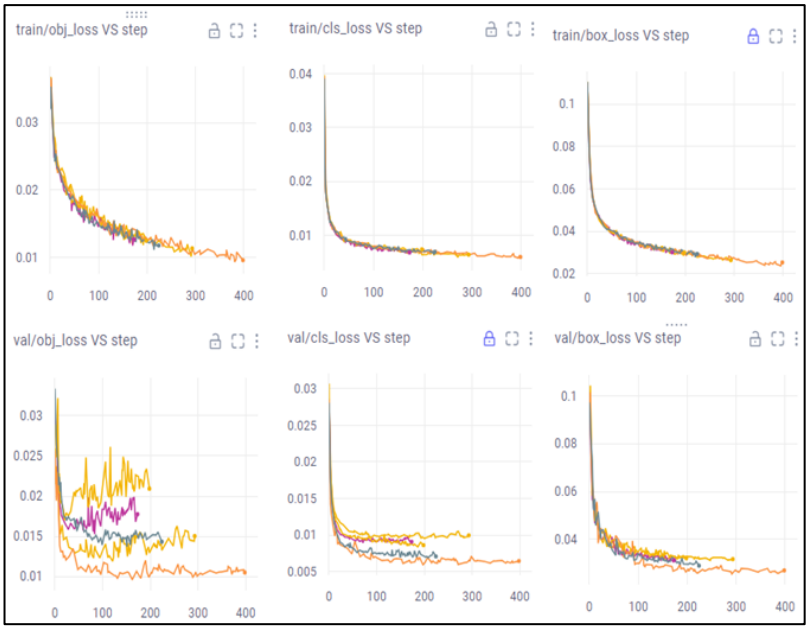

Figura 13. Funciones de pérdida para Yolov5l

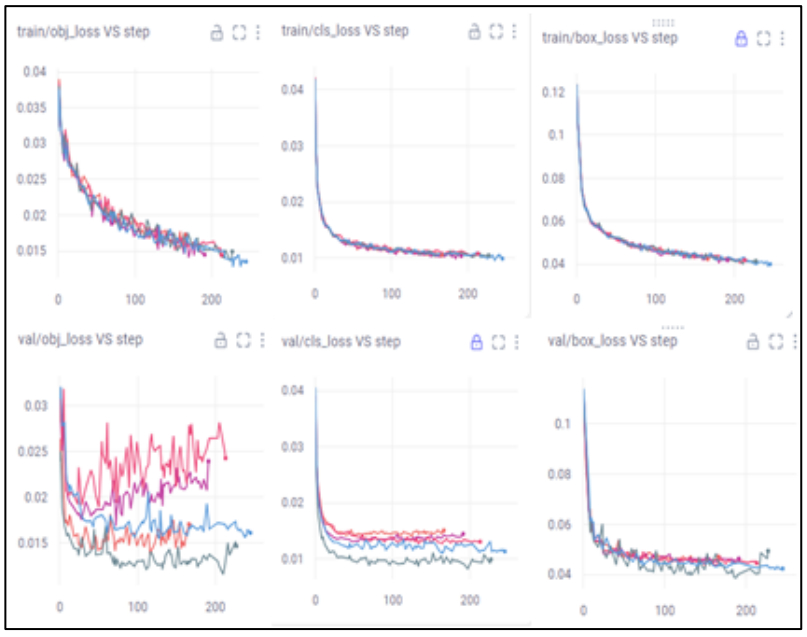

Figura 14. Funciones de pérdida para Yolov5x

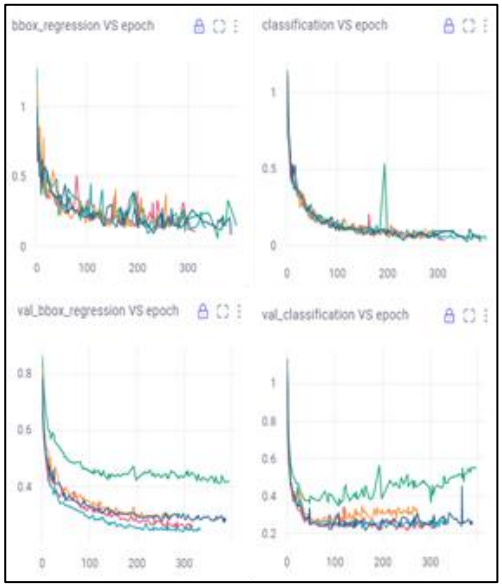

Figura 15. Funciones de pérdida para RetinaNet / ResNet50

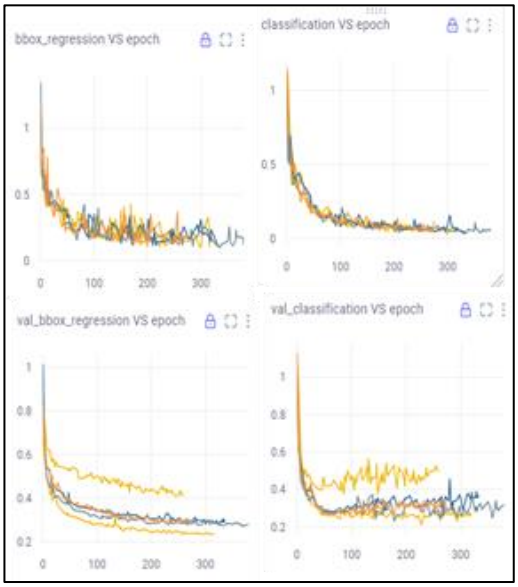

Figura 16. Funciones de pérdida para RetinaNet / ResNet101