UNIVERSIDAD DE LA REPÚBLICA URUGUAY

FACULTAD DE INGENIERÍA

Procesamiento Automático de Imágenes para la Detección de Daños en Palas de Aerogeneradores

Informe de Proyecto de Grado presentado por

Jessica Arroyo, Martín Borba y Francisco Casarotti

en cumplimiento parcial de los requerimientos para la graduación de la carrera de Ingeniería en Computación e Ingeniería en Sistemas de Comunicación de Facultad de Ingeniería de la Universidad de la República

Supervisores

Facundo Benavides
Rafael Canetti

Montevideo, 9 de abril de 2024

Agradecimientos

Queremos expresar nuestra más sincera gratitud a todas las instituciones y personas cuya colaboración fue fundamental para la realización de esta tesis. La posibilidad de llevar a cabo este proyecto de investigación se debe en gran medida al generoso apoyo y a los recursos proporcionados por las distintas partes.

A nuestras familias, agradecemos profundamente su amor incondicional, paciencia y comprensión en los momentos en que nuestra atención estaba completamente sumergida en este proyecto. A nuestros padres, por ser nuestra inspiración y por su sacrificio constante, para proporcionarnos las oportunidades que hoy estamos agradecidos de tener.

A nuestros amigos, quienes han estado presentes en las alegrías y desafíos de este camino. Su compañía, risas y ánimos fueron un alivio necesario en los momentos de tensión académica.

Nuestros tutores, Facundo Benavides y Rafael Canetti, merecen un agradecimiento especial por su valiosa orientación y apoyo en este proyecto. Su experiencia y guía fueron esenciales para el desarrollo y finalización del mismo. Para aquellos interesados en explorar más a fondo la experiencia y la destacada contribución académica de nuestros tutores, les invitamos a visitar las páginas con sus publicaciones (Benavides, 2024) y (Canetti, 2024).

Nuestro agradecimiento se extiende también a ClusterUY (ClusterUY, 2024), plataforma de computación de alto desempeño, por facilitarnos su acceso. Esta entidad, caracterizada por su enfoque sin fines de lucro y gestión autosostenible, proporcionó el entorno adecuado para llevar a cabo una amplia variedad de pruebas y experimentos que enriquecieron nuestra experiencia de aprendizaje. Además, agradecer por su compromiso y apoyo a la comunidad científica e investigadora en Uruguay.

A Timbó (Timbó, 2024), la plataforma de acceso a recursos científicos y tecnológicos proporcionada por la Agencia Nacional de Investigación e Innovación (ANII, 2024) de Uruguay. Timbó desempeñó un papel crucial al permitirnos acceder a la última bibliografía y literatura científico-tecnológica a nivel mundial, proporcionando así una base sólida para nuestra investigación. También destacar el continuo compromiso con la comunidad de investigadores e innovadores. Ha logrado democratizar el acceso a la información, beneficiando no solo a este proyecto, sino también a la comunidad académica uruguaya en general.

Adicionalmente, agradecemos a la Universidad Técnica de Dinamarca (DTU, 2024) por brindar un conjunto de imágenes público que resultó invaluable en las fases iniciales de nuestro proyecto, funcionando como una guía para nuestros primeros entrenamientos.

Finalmente, deseamos expresar nuestro más sincero agradecimiento a Guillermo Carbajal (Carbajal, 2024) por su invaluable colaboración. Sus comentarios, respaldados por su experiencia en el tema, resultaron fundamentales para implementar cambios significativos que contribuyeron a la obtención de resultados más confiables.

Resumen

Este proyecto de grado aborda la detección de daños superficiales en las palas de aerogeneradores. Comienza con un resumen del marco teórico que abarca conceptos básicos sobre el aprendizaje automático, la noción de redes neuronales y específicamente convolucionales, junto con los componentes más importantes de su arquitectura. A su vez, se cubren nociones esenciales relacionadas con la detección de objetos, medidas de desempeño, funciones de costo, optimizadores y un análisis de los modelos utilizados para este proyecto. Posteriormente, se realiza un análisis de la literatura que detalla los métodos empleados por estudios similares y sus resultados correspondientes. Primero se identifica el propósito y protocolo a utilizar. Luego se proporciona un listado de los artículos encontrados y su nivel de relevancia con el proyecto en cuestión. Finalmente, se analizan los artículos y se proporciona en este informe un resumen de la información encontrada relacionada con los conjuntos de datos utilizados, clasificación de los daños, preprocesamiento de los datos, modelos, métricas y resultados obtenidos. También se contribuye con una tabla de la información más relevante encontrada para cada uno de los artículos. Luego se describe en una nueva sección las experimentaciones llevadas a cabo con diversos modelos de aprendizaje automático, incluyendo técnicas de aumento de datos y ajuste de hiperparámetros. Se comienza detallando la plataforma de ejecución y los conjuntos de datos utilizados. Luego se separan las experimentaciones en tres fases que dependen del conjunto de datos, el modelo a entrenar y las experimentaciones a realizar. La primera es una comparación de todos los modelos a evaluar, la segunda una comparación de los dos mejores de la fase anterior y la tercera la optimización del modelo YOLOv8n concluido como mejor. Como resultado, se logra alcanzar un $\text{mAP}_{0.5}$ de 0.5 con el modelo YOLOv8n, lo que representa una contribución en la reducción de la carga de trabajo de los inspectores, y mejora la eficiencia del mantenimiento preventivo.

Palabras clave: Detección de objetos, Aprendizaje automático, Redes neuronales convolucionales, Daños en palas de aerogeneradores, Visión por computadora

Índice general

Capítulo 1

Introducción

Contenido

En un mundo en constante crecimiento demográfico, la necesidad de adoptar fuentes de energía con un menor impacto ambiental se vuelve cada vez más importante. La energía eólica, que aprovecha la potencia del viento mediante turbinas, se presenta como una opción sostenible y renovable en contraste con los métodos de generación de energía más convencionales basados en combustibles fósiles (Wolniak y Skotnicka-Zasadzień, 2023). En el pasado, los costos de generar energía eólica, en comparación con sus beneficios, la volvían un método de generación de energía no rentable, pero en las últimas décadas, estos costos han disminuido significativamente, haciéndola cada vez más competitiva con otras fuentes de energía. Su abundancia y carácter inagotable la posicionan como una alternativa idónea. Es crucial resaltar que la generación de energía eólica contribuye de manera significativa a la reducción de la huella de carbono, a diferencia de los combustibles fósiles, al no emitir gases de efecto invernadero. Además, su creciente rentabilidad y capacidad para generar empleo la consolidan como una fuente de energía verdaderamente prometedora (Energy5, 2023), ya que la industria eólica genera una gran cantidad de empleos, tanto en la fase de construcción como en la de operación y mantenimiento.

En América Latina, como parte de la iniciativa de Acción Prioritaria (APS) liderada por la Agencia Internacional de Energía (IEA), la región está redoblando sus esfuerzos para hacer avanzar la transición hacia fuentes de energía renovable. Se proyecta que estas fuentes representen el 70 % del suministro energético para el año 2030, una década antes de lo estimado en el escenario de Pasos Estratégicos (STEPS) también delineado por la IEA. Además, se espera que esta proporción aumente a más del 90 % para el año 2050. En particular, se anticipa que la contribución de la energía solar, fotovoltaica y eólica a la generación de electricidad se duplique para el año 2030, pasando del actual 11 % al 40 % para el año 2050, según el informe de la IEA de 2023 (IEA, 2023).

En Uruguay, la utilización de los aerogeneradores comenzó a ser una realidad en el año 2009, cuando se inauguró el primer parque eólico comercial en el país, el cual está ubicado en Sierra de los Caracoles, Maldonado (Verri, 2021). Con esta inauguración también llegaron incentivos para la creación de nuevos parques y sus respectivas regulaciones (*Decreto No. 403/009*, 2009).
Hoy en día se cuenta con aproximadamente 43 parques eólicos, de los cuales el 20 % pertenece a UTE y el otro 80 % a inversiones privadas (Verri, 2021). Entre los años 2010 al 2016 se han invertido hasta US$ 7.8 billones en infraestructura energética. Se estima que en 2018 un 38 % de la electricidad generada provino de la energía eólica (UruguayXXI, 2019).

Según la revista científica de acceso abierto *Applied Sciences* (Zou y Cheng, 2022), se estima que el costo de operación y mantenimiento representa aproximadamente de un 20 % al 25 % del costo nivelado de electricidad (LCOE) total en los sistemas actuales de energía eólica.

A pesar de todas las ventajas antes nombradas, es importante destacar que los costos de operación y mantenimiento son elevados, por lo que la energía que generan estos molinos debe ser lo más eficiente

posible. Las palas, como uno de los componentes más relevantes, ejercen un impacto directo en la eficiencia de generación de energía. Estas estructuras son susceptibles a diversas condiciones ambientales, como vientos fuertes, tormentas de arena, descargas eléctricas, lluvia y nieve, entre otras (Zou y Cheng, 2022).

Debido a estas hostiles condiciones, es fundamental la inspección frecuente en busca de daños. Una práctica común para llevar a cabo la inspección es mediante imágenes de las palas, las cuales pueden ser capturadas por un observador en tierra o el uso de un dron. En este tipo de evaluación, se requiere la intervención de un especialista que analice las imágenes para identificar la presencia de posibles daños.

En este contexto, el aprendizaje profundo (*deep learning*) surge como una herramienta que puede potenciar las metodologías convencionales. Con un conjunto de imágenes lo suficientemente extenso, los modelos de aprendizaje profundo tienen la capacidad de reconocer características visuales asociadas a posibles daños en las palas (Iyer, Nguyen, y Khushu, 2022), brindando así un apoyo significativo a los operadores al permitirles identificar de manera más rápida y precisa estos defectos.

Sin embargo, ¿por qué es tan crucial prestar especial atención a la detección de posibles daños? Esta importancia radica, en parte, en la capacidad de abordar de manera temprana un desgaste leve, evitando así su agravamiento. Cuando no se aborda rápidamente, el desgaste en la pala puede deteriorarse gradualmente hasta llegar al punto de ruptura, generando pérdidas económicas significativas y potenciales riesgos de seguridad (Zou y Cheng, 2022).

Los métodos tradicionales de inspección incluyen la inspección visual por parte de un observador en tierra, que es simple y económica, pero puede carecer de precisión y ser subjetiva, y la inspección mediante drones, que ofrece imágenes de alta resolución mejorando la precisión, aunque requiere personal especializado y puede resultar costosa. Sin embargo, ambos métodos presentan limitaciones, como la subjetividad y posibles errores humanos en la inspección visual, el costo elevado de la inspección con drones, especialmente en parques eólicos extensos, y la cuestión de seguridad, ya que la inspección de las palas puede ser peligrosa, especialmente en condiciones climáticas adversas

1.1. Objetivos y resultados esperados

El objetivo de este proyecto de grado es comprender el estado del arte asociado a la problemática anteriormente descrita. Evaluando la factibilidad y conveniencia de reemplazar los métodos tradicionales de inspección superficial de palas de aerogeneradores, con soluciones basadas en sistemas de procesamiento automático de imágenes, con el fin de ayudar en la detección temprana de daños. Se realizó un análisis exhaustivo de la literatura existente sobre el tema, con énfasis en el procesamiento de las imágenes basado en aprendizaje automático. Esto con el propósito de desarrollar capacidades locales para la inspección frecuente del estado de las palas y garantizar el correcto funcionamiento de los aerogeneradores en el país. En general, se espera que este proyecto contribuya a la reducción de costos y el mejoramiento de la eficiencia en la inspección de palas de aerogeneradores en Uruguay, lo que podría tener un impacto positivo en la industria de la energía eólica a nivel local y regional.

1.2. Contribuciones

- Búsqueda, análisis y clasificación de documentos disponibles: La investigación inició con una búsqueda exhaustiva de la literatura científica relacionada con la detección de daños en palas. Se identificaron y revisaron diversos documentos académicos relevantes. La información extraída se organizó sistemáticamente en una tabla de análisis, que incluye detalles sobre los modelos utilizados, técnicas generales empleadas, clasificaciones aplicadas y resultados.

- Búsqueda y análisis de *datasets* disponibles: Se realizó una investigación para identificar *datasets* relevantes disponibles para el entrenamiento de modelos, algunos documentados en los textos científicos antes mencionados, otros disponibles en internet.

- Filtrado y selección de modelos: Este proceso permitió identificar aquellos modelos más prometedores que se adecuaban a las necesidades específicas de este proyecto. Posteriormente, se procedió a una evaluación comparativa para determinar cuáles de estos modelos ofrecían el rendimiento óptimo en

términos de precisión, eficiencia y escalabilidad. Para ello se utilizó uno de los conjuntos de datos disponibles comentados anteriormente.

- Refinamiento del *dataset* de DS.UY: Se llevó a cabo un proceso de refinamiento del *dataset* de DS.UY. Esto implicó la reanotación de objetos y la definición de cuadros delimitadores con el propósito de generar un recurso de mayor calidad y utilidad para investigaciones posteriores.
- Experimentación particular con DS.UY: Se dedicó un enfoque específico a la experimentación utilizando datos provenientes de DS.UY. Este enfoque permitió profundizar en aspectos específicos relacionados con este conjunto de datos, explorando sus características y desafíos particulares.
- Modelo final: Se brindó una versión final del modelo YOLOv8n, entrenado con hiperparámetros que optimizaron los resultados en el contexto de detección de daños en palas de aerogeneradores de DS.UY. El mismo es entrenado bajo un *framework* de código abierto.

1.3. Organización del documento

El documento está estructurado en diversos capítulos numerados que se detallan a continuación:

- **Introducción** (Capítulo 1): Establece los objetivos y resultados esperados, así como las contribuciones del estudio.
- **Marco Teórico** (Capítulo 2): Aborda conceptos fundamentales como la detección de objetos, medidas de rendimiento, funciones de costo, optimizadores y tipos de modelos.
- **Revisión de Antecedentes** (Capítulo 3): Examina el propósito, protocolos de investigación, búsqueda bibliográfica, conjuntos de datos, clasificación de daños, preprocesamiento de datos, modelos utilizados y resultados obtenidos en investigaciones previas.
- **Experimentación** (Capítulo 4): Detalla la plataforma utilizada, conjuntos de datos empleados y los procesos de experimentación. Incluyendo comparaciones, técnicas de aumento de datos y evaluaciones de modelos específicos.
- **Conclusiones y Trabajo Futuro** (Capítulo 5): Resume los hallazgos y plantea posibles direcciones para investigaciones futuras.

Capítulo 2

Marco teórico

Contenido

Este marco teórico proporciona una visión general de los elementos fundamentales que sustentan la detección de objetos, sirviendo como guía para el desarrollo de este proyecto de grado. Desde las anotaciones que facilitan el entrenamiento de modelos, técnicas de aumentación, hasta las redes neuronales y las redes convolucionales. La sección 2.1 abre la puerta al campo del aprendizaje automático, destacando los conceptos básicos de lo que es un modelo y el entrenamiento del mismo. Luego, se aborda el concepto de Redes Neuronales en la sección 2.2, proporcionando tanto una introducción general como una exploración de conceptos más avanzados, como son *Batch Normalization* y *Transfer Learning*. También se aborda el tema de Redes Neuronales Convolucionales (CNN) en la sección 2.3, explorando su funcionamiento y operaciones. En la sección 2.4 se ve la temática de Detección de Objetos, donde se incluyen conceptos básicos para esta área, como son IoU, *Anchor Boxes* y Aumentación de datos. En la sección 2.5, se introduce el concepto de la matriz de confusión, muy utilizada para las tareas de clasificación, y junto a ella las métricas más comunes para la evaluación de desempeño de los modelos. La sección 2.6 se adentra en funciones de costo, específicamente diseñadas para abordar aspectos de localización y clasificación. Se exploran optimizadores en la sección 2.7, desde el clásico Descenso por Gradiente hasta las variantes más modernas como Adam y AdamW. Finalmente, en la sección 2.8, se examina la diversidad de modelos, desde sus clasificaciones hasta aquellos evaluados en el contexto del proyecto.

2.1. Introducción al Aprendizaje Automático

El aprendizaje automático, una rama de la inteligencia artificial, se centra en desarrollar algoritmos y modelos capaces de aprender a partir de datos. En lugar de seguir instrucciones específicas, estos sistemas utilizan datos para mejorar su rendimiento, permitiéndoles aprender de la experiencia y optimizar su desempeño con nuevos datos. Este enfoque encuentra aplicación en diversos campos como el procesamiento de audio, imágenes y video, procesamiento de lenguaje natural, entre otros. El éxito del aprendizaje automático depende en gran medida de la calidad de los datos y la elección adecuada de algoritmos para el problema en cuestión. A continuación, se presentarán algunos conceptos básicos utilizados en esta temática.

2.1.1. Aprendizaje Supervisado

Los algoritmos de aprendizaje automático suelen clasificarse como supervisados o no supervisados. Se denominan supervisados cuando se proporcionan al algoritmo de aprendizaje entradas y salidas emparejadas. En la figura 2.1 se puede ver un esquema del mismo. Por otro lado, no supervisados es cuando se proporcionan datos sin etiquetas correspondientes (Szeliski, 2022).

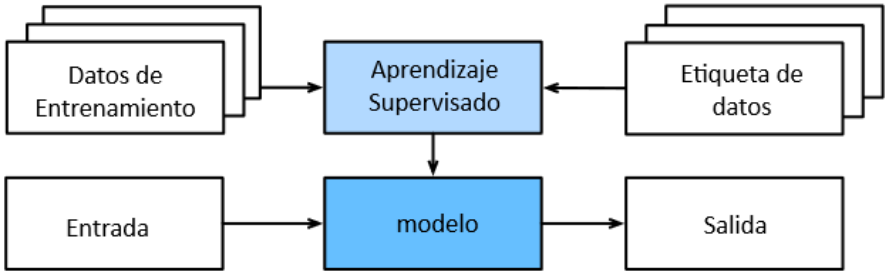


Figura 2.1: Esquema del aprendizaje supervisado.
(Szeliski, 2022)

El aprendizaje supervisado implica proporcionar al modelo pares de datos de entrada junto con sus salidas correspondientes. El modelo ajusta sus parámetros para minimizar la discrepancia entre sus predicciones y las salidas esperadas, evaluada mediante una función de costo. En la sección (2.6), se explorarán las funciones de costo más comunes en la detección de objetos. Este proceso de ajuste del modelo se conoce como entrenamiento.

2.1.2. Entrenamiento: iteración, época y tamaño de lote

Los tres conceptos más importantes necesarios para entender el proceso de entrenamiento son las iteraciones, las épocas (*epoch*) y el tamaño de lote (*batch size*). En general, no es posible utilizar todos los datos de entrenamiento simultáneamente para ajustar los modelos. Aquí entra en juego el concepto de tamaño de lote, el cual refiere a la cantidad de instancias de datos que se utilizan en una única iteración de entrenamiento. Una iteración refiere a cada paso individual en el proceso de entrenamiento, donde se alimenta un lote de datos al modelo, se calcula el error, y se ajustan los pesos del modelo en función de este error. Una vez que se realiza una pasada completa a través de todo el conjunto de datos de entrenamiento, se dice que se entrenó por una época. En otras palabras, una época es cuando el modelo ha visto y procesado todos los ejemplos de entrenamiento una vez. En la práctica, el entrenamiento suele consistir en varias épocas, donde el modelo se ajusta repetidamente a los datos para mejorar su rendimiento. Al realizar un entrenamiento se deben definir al menos dos de estos tres conceptos, generalmente se define la cantidad de épocas y el tamaño de lote, quedando así definidas la cantidad de iteraciones (Géron, 2022). En 2.1 se puede apreciar la relación entre las variables mencionadas.

$$Iteraciones\ totales = Épocas \times \left(\frac{Tamaño\ de\ lote}{Tamaño\ de\ datos\ de\ entrenamiento} \right) \tag{2.1}$$

2.1.3. Sobreajuste

El sobreajuste, también conocido como *overfitting*, es un fenómeno común en el aprendizaje automático que ocurre cuando un modelo se ajusta demasiado bien a los datos de entrenamiento, como se puede ver en la figura 2.2. En lugar de generalizar patrones útiles, el modelo memoriza los datos de entrenamiento específicos, lo que puede resultar en un rendimiento deficiente en nuevos datos no vistos (Géron, 2022). En contraposición, el subajuste se da cuando el modelo no adquiere suficiente comprensión de los datos de entrenamiento, resultando en un modelo demasiado básico que no logra abordar la complejidad de los datos.

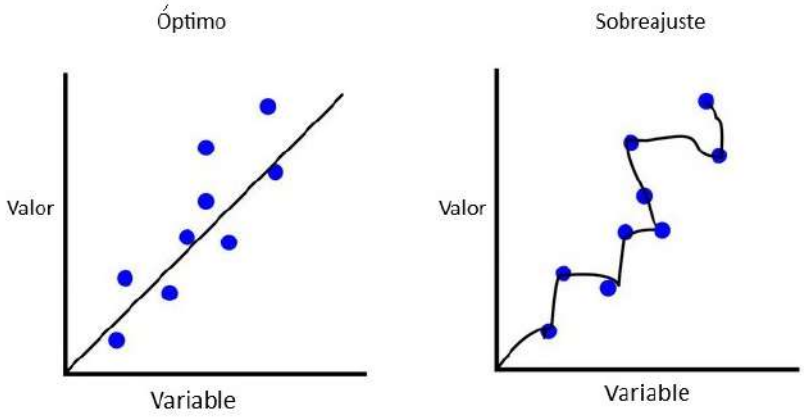


Figura 2.2: Sobreajuste vs óptimo.
(Siddhardhan, 2022)

El sobreajuste suele ser consecuencia de la complejidad excesiva del modelo en relación con la cantidad limitada de datos de entrenamiento. Los modelos demasiado complejos pueden capturar patrones específicos de los datos de entrenamiento que no son representativos del conjunto de datos en su totalidad. Es común ver este tipo de problemas en redes neuronales convoluciones (sección 2.3), dado la cantidad de parámetros que posee. Es fácil detectar este problema durante el entrenamiento comparando las gráficas de error para los conjuntos de *train* y *test*. Esto se puede apreciar mejor en la figura 2.3 donde al pasar más épocas, el error en el conjunto de *test* incrementa, mientras que en el conjunto de *train* decrementa.

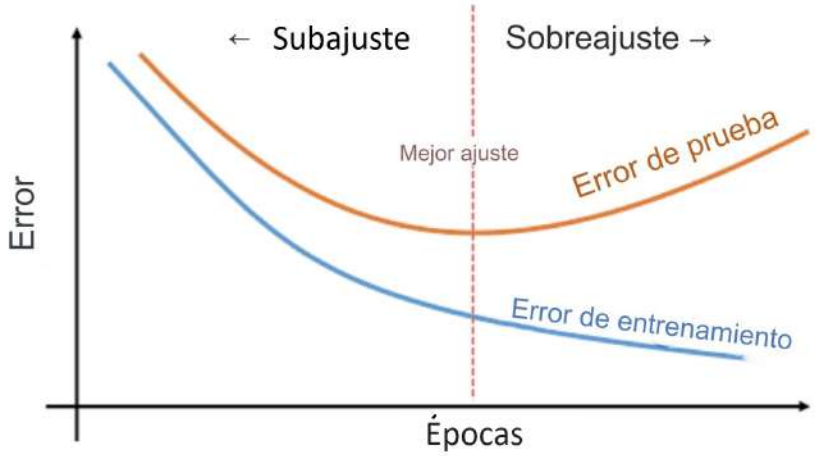


Figura 2.3: Error en los conjuntos de *train* y *test*.
(SETScholars, 2023)

Para contrarrestar esta problemática, se recurre a técnicas como la regularización, que restringen la flexibilidad del modelo durante el proceso de entrenamiento. Los hiperparámetros de regularización varían según el modelo empleado. En el contexto de *computer vision*, ejemplos de estas técnicas incluyen el *dropout* (sección 2.2.2), la normalización por lotes (sección 2.2.1), y la aumentación de datos (sección 2.4.5), entre otros.

2.2. Redes neuronales

Una red neuronal es un modelo computacional inspirado en el funcionamiento del cerebro humano. Está compuesta por nodos llamados neuronas, organizados en capas, que trabajan en conjunto para realizar tareas específicas, como detección y clasificación de imágenes, predicción de precios, etc.

Para entender un poco mejor el funcionamiento, es de ayuda entender cómo funciona una neurona a nivel biológico. Es una célula que posee muchas extensiones ramificadas llamadas dendritas, y una única y larga extensión llamada axón, como se puede ver en la figura 2.4. El axón está unido a las dendritas de otras células. Cuando una neurona genera impulsos eléctricos, estos impulsos viajan a lo largo del axón y liberan señales químicas a las dendritas de otras células, estas señales químicas son denominadas neurotransmisores. Cuando una neurona recibe una cantidad suficiente de estos neurotransmisores a través de las dendritas, genera sus propios impulsos eléctricos (Géron, 2022).

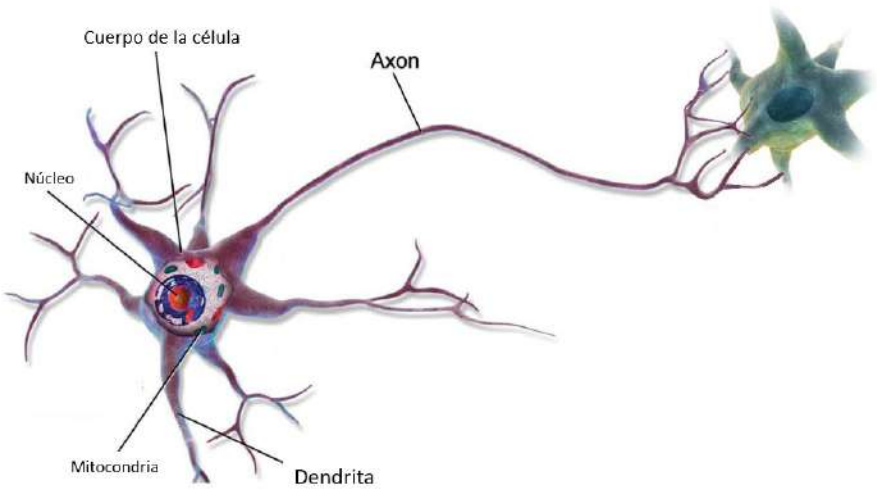


Figura 2.4: Neurona biológica.
(Géron, 2022)

Con base en este concepto es que se construyen las redes neuronales, la neurona recibe múltiples señales de entrada, las pondera mediante pesos en las conexiones, y luego aplica una "función de activación" para generar una salida final. La función de activación introduce no linealidades en el modelo. Estas no linealidades permiten a la red aprender patrones y relaciones más complejas en los datos, mejorando su capacidad para representar y generalizar información en tareas más desafiantes. La ecuación para el cálculo de la salida de estas neuronas puede ser vista en 2.2.

$$y = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{2.2}$$

- y es la salida del perceptrón.
- σ es la función de activación.
- x_i son las entradas.
- w_i son los pesos asociados a las entradas x_i.
- b es el sesgo del perceptrón.
- n es el número de entradas.

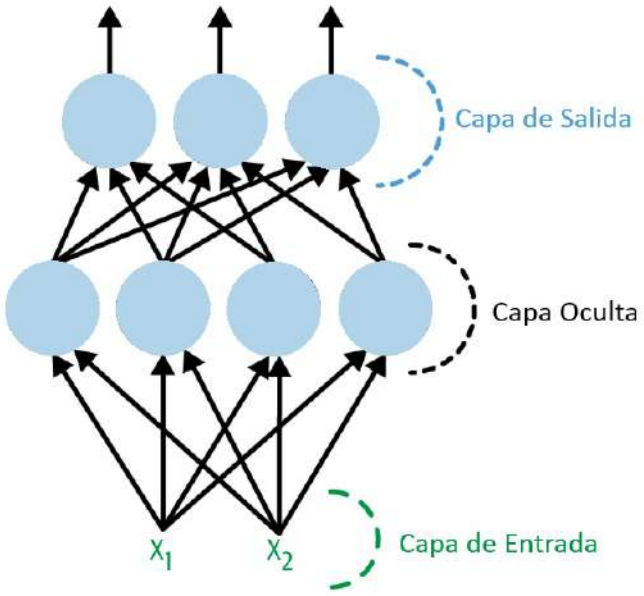


Figura 2.5: Red neuronal.
(Géron, 2022)

Esta salida "y" obtenida luego de aplicar la activación se convierte en la entrada para otras neuronas, permitiendo así la transmisión de información a lo largo de la red (Géron, 2022). Sin embargo, ¿cómo se estructuran estas interconexiones? Las arquitecturas de las redes neuronales generalmente se organizan en "capas", y podemos categorizar estas capas en tres grupos principales (ver figura 2.5): la capa de entrada, donde las entradas provienen del entorno externo a la red y la salida se dirige hacia otras neuronas; las capas ocultas, donde las conexiones tanto de entrada como de salida se establecen exclusivamente entre neuronas (pudiendo existir más de una capa oculta); y las capas de salida, donde las conexiones de entrada provienen de las neuronas ocultas y la salida serían los resultados de la red. Es importante destacar que las conexiones entre las neuronas están ponderadas, y de aquí en adelante se referirán como pesos o parámetros de la red.

La inicialización de estos pesos puede realizarse de manera aleatoria o utilizando heurísticas que consideran información clave de la red, como la función de activación y el tamaño de la misma. También es posible reutilizar pesos previamente entrenados en una tarea similar, como se detalla en la sección 2.2.3. El concepto de entrenar una red es el de ajustar estos pesos para que la red devuelva valores esperados. En términos más claros, durante la fase de entrenamiento, estos pesos experimentan ajustes con el objetivo de optimizarse, permitiendo al sistema aprender a ejecutar la tarea deseada. Esto se logra al alimentar a la red con datos de entrada y "corregirle" la salida. La corrección se logra con el uso de funciones de costo (sección 2.6) que comparan la salida predicha con la "verdad anotada" y se ajustan los pesos en consecuencia utilizando "*backpropagation*" (Géron, 2022). Después de la fase de entrenamiento de estas redes, el resultado valioso son los pesos ajustados a la tarea en particular. Es importante señalar que para obtener pesos óptimos se necesita contar con una cantidad de datos de alta calidad.

2.2.1. *Batch Normalization*

Introducida por Ioffe y Szegedy en 2015 (Ioffe y Szegedy, 2015), *Batch Normalization* es una técnica clave en el entrenamiento de redes neuronales profundas, que normaliza el resultado de las activaciones de una capa durante el proceso de entrenamiento. En pocas palabras, esta técnica consiste en normalizar la salida de una capa tomando en cuenta la media y la desviación estándar, estabilizando así la distribución de activaciones.

Al proporcionar una distribución más estable de activaciones, acelera la convergencia, permite tasas de aprendizaje más altas y ofrece cierta regularización sobre los pesos, reduciendo la necesidad de técnicas adicionales.

2.2.2. *Dropout*

Dropout es una de las técnicas de regularización para redes neuronales. Es un algoritmo bastante simple: en cada paso de entrenamiento, cada neurona (incluidas las neuronas de entrada, pero siempre excluyendo las neuronas de salida) tiene una probabilidad "p" de ser "eliminada temporalmente". Esto implica que será completamente ignorada durante una iteración del entrenamiento, pero puede estar activa durante la siguiente iteración. El hiperparámetro "p" se llama tasa de *dropout* y generalmente se establece entre el 10 % y el 50 %, y más cerca del 40 % y 50 % en redes neuronales convolucionales. Es importante señalar que al emplear la técnica de *dropout*, ninguna neurona en realidad es eliminada. Un ejemplo de esto puede verse en la figura 2.6 (Géron, 2022).

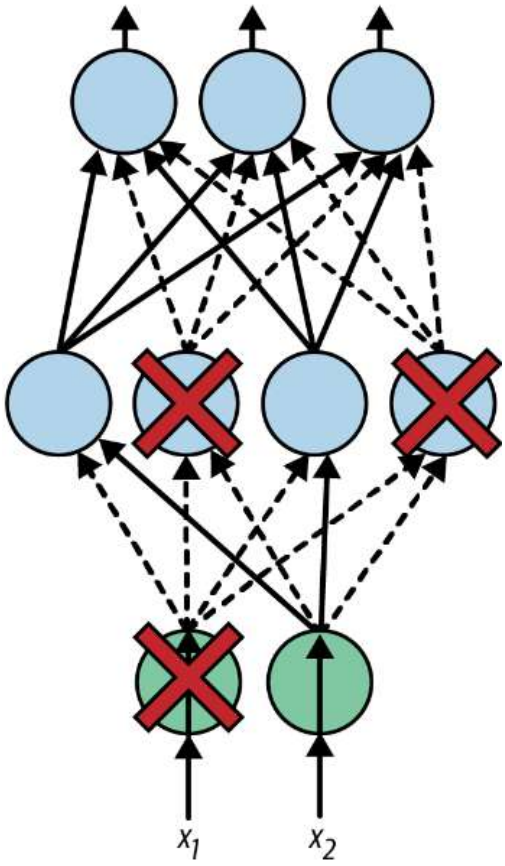


Figura 2.6: Ejemplo de una red neuronal con *dropout*.
(Géron, 2022)

Otra forma de entender el *dropout* es darse cuenta de que se genera una red neuronal única en cada iteración del entrenamiento. Dado que cada neurona puede estar presente o ausente, hay un total de 2^N posibles redes (donde N es el número total de neuronas eliminables). Esta es una cantidad tan grande que es casi imposible que se elija la misma red neuronal dos veces. En esencia, esto implica que luego de 10000 iteraciones de entrenamiento, se han entrenado 10000 redes neuronales diferentes, cada una con solo una instancia de entrenamiento. Estas redes neuronales obviamente no son independientes porque comparten muchos de sus pesos, pero son todas diferentes. La red neuronal resultante se puede ver como un conjunto promedio de todas estas redes neuronales más pequeñas (Géron, 2022).

2.2.3. *Transfer learning*

Transfer learning implica la reutilización de un modelo previamente entrenado como punto de partida para el entrenamiento de una nueva tarea. Durante el entrenamiento de la tarea inicial, el modelo ajusta

sus pesos para capturar patrones específicos presentes en los datos. Posteriormente, cuando se busca realizar *transfer learning* para una nueva tarea, los pesos aprendidos en la tarea original se utilizan como un punto de partida para los entrenamientos de la nueva tarea más específica. Este enfoque permite al modelo aprovechar la información valiosa y los patrones previamente capturados en la tarea original.

Además de la reutilización de pesos, se emplea una técnica conocida como *freezing*. Esta práctica implica "congelar" las capas inferiores del modelo durante el entrenamiento, es decir, no actualizar sus pesos. Este enfoque resulta beneficioso, especialmente cuando la tarea para la cual se preentrenaron los pesos guarda similitud con la nueva. Las capas inferiores suelen ser responsables del aprendizaje de las características más generales (por ejemplo bordes). Esto conlleva que, al entrenar el modelo, este se centre en aprender las características que son relevantes para la nueva tarea.

El hecho de llevar a cabo un entrenamiento con pesos preentrenados se le conoce como *fine-tuning*. Los pesos se refinan para adaptarse específicamente a la nueva tarea. La transferencia de pesos es esencialmente una forma de reutilizar el conocimiento adquirido por el modelo en una tarea, en otra tarea similar. Este proceso contribuye a mejorar el rendimiento del modelo en la nueva tarea, especialmente cuando la disponibilidad de datos de entrenamiento es limitada.

Aun así, entrenar desde pesos aleatorios no es peor que desde pesos preentrenados dadas las suficientes iteraciones (He, Girshick, y Dollár, 2018), el problema es que para entrenar desde cero un modelo, es necesaria una abundante cantidad de datos para no sobreajustar.

Generalmente, en tareas de detección de objetos es común utilizar los pesos preentrenados con el *dataset* de *COCO* (Lin y cols., 2014) e Imagenet (ImageNet, 2009).

2.3. Redes neuronales convolucionales

Las redes neuronales convolucionales, conocidas como *Convolutional Neural Networks* (CNN) en inglés, constituyen una categoría especializada de redes neuronales diseñadas específicamente para procesar datos dispuestos en forma matricial, como las imágenes, que se componen de cuadrículas de píxeles. Estas redes se destacan por su eficacia en la identificación de patrones complejos y en la ejecución de diversas tareas visuales, como el reconocimiento de objetos y la segmentación de imágenes. Esto se debe a su capacidad para capturar y procesar características visuales en varias regiones de una imagen, lo que las convierte en herramientas altamente efectivas en el ámbito del procesamiento visual (Géron, 2022).

Las CNN, al igual que las redes neuronales, son un modelo computacional inspirado en el funcionamiento del cerebro humano. Estas surgieron del estudio de la corteza visual del cerebro y se han utilizado en el area de *computer vision* desde la década de 1980. En los últimos tiempos, gracias al aumento en la potencia computacional y la cantidad de datos de entrenamiento disponibles, las CNN han logrado alcanzar muy buenos rendimientos en tareas visuales complejas.

Lo más importante para entender una CNN, es el concepto de la capa convolución. Cada capa es una grilla de neuronas que se comunican únicamente con cierto grupo de neuronas de la capa inferior. Viendo la figura 2.7, las neuronas en la primera capa no están conectadas a cada píxel individual de la imagen de entrada, sino solo a píxeles en sus regiones de influencia. A su vez, cada neurona en la segunda capa solo está conectada a neuronas ubicadas dentro de un pequeño rectángulo en la primera capa. Esta arquitectura permite que la red se concentre en características pequeñas de bajo nivel en la primera capa oculta, para luego ensamblarlas en características más grandes de nivel superior en la siguiente capa oculta, y así sucesivamente. Esta estructura jerárquica es común en imágenes del mundo real, esta es una de las razones por las cuales las CNN funcionan tan bien para el reconocimiento de imágenes.

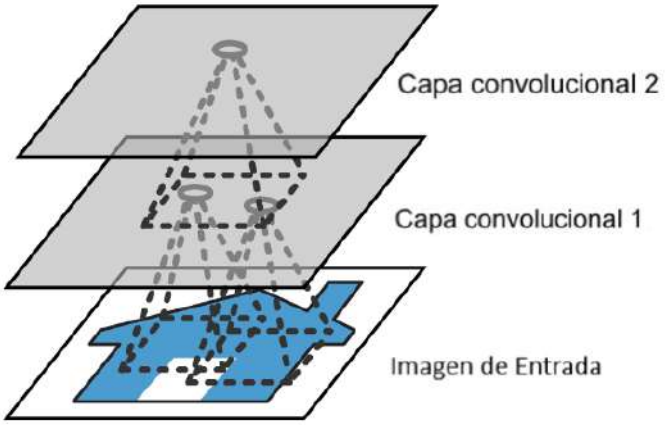


Figura 2.7: Imagen de entrada y capas de una red neuronal convolucional.
(Géron, 2022)

2.3.1. Operación de convolución

La operación clave en estas redes es, como su nombre lo indica, la convolución. Sabiendo esto, en 2.3 se resumen las explicaciones anteriores en una gran expresión matemática, se muestra cómo calcular la salida del resultado de una neurona dada en una capa de convolución (Géron, 2022).

$$zi,j,k = bk + \sum_{u=0}^{fh-1} \sum_{v=0}^{fw-1} \sum_{k'=0}^{fn-1} X_{i,j,k'} \times Wu,v,k,k' \qquad \text{con} \quad \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases} \tag{2.3}$$

- $Z_{i,j,k}$ es la salida de la neurona ubicada en la fila i, columna j en el mapa de características k de la capa de convolución (capa l).

- s_h y s_w corresponde al *stride* (sección 2.3.3) vertical y horizontal, f_h y f_w son la altura y anchura del campo receptivo, y fn' es el número de mapas de características en la capa anterior (capa $l-1$).
- $X_{i',j',k'}$ es la salida de la neurona ubicada en la capa $l-1$, fila i', columna j', mapa de características k' (o canal k' si la capa anterior es la capa de entrada).
- b_k es el término de sesgo para el mapa de características k (en la capa l). Puede pensarse como un ajuste que modifica el brillo general del mapa de características k.
- $w_{u,v,k',k}$ es el peso de conexión entre cualquier neurona en el mapa de características k de la capa l y su entrada ubicada en la fila u, columna v (en relación con el campo receptivo de la neurona) y el mapa de características k'.

2.3.2. *Kernel*

Un *kernel* es una matriz pequeña que se utiliza para realizar la operación de convolución sobre una imagen. Esta operación implica multiplicar la misma matriz en distintas posiciones de la imagen, generando así otra imagen como resultado. Las posiciones de aplicación del *kernel* cubren toda la imagen y están espaciadas de manera uniforme. La distancia entre estas posiciones se conoce como *"stride"*.

A medida que se desplaza el *kernel* para recorrer la matriz entera, se van multiplicando sus valores con los píxeles correspondientes en cada posición de la imagen. Esta operación genera un mapa de características (*feature map*), que resalta regiones específicas de la imagen que coinciden con patrones aprendidos por el *kernel*. Por ejemplo, un kernel puede detectar bordes, mientras que otro puede identificar texturas o esquinas. La imagen 2.8 demuestra el resultado de aplicar 2 *kernels* simples, uno horizontal y otro vertical, produciendo así 2 *feature maps* en los cuales se notan los bordes horizontales y verticales respectivamente.

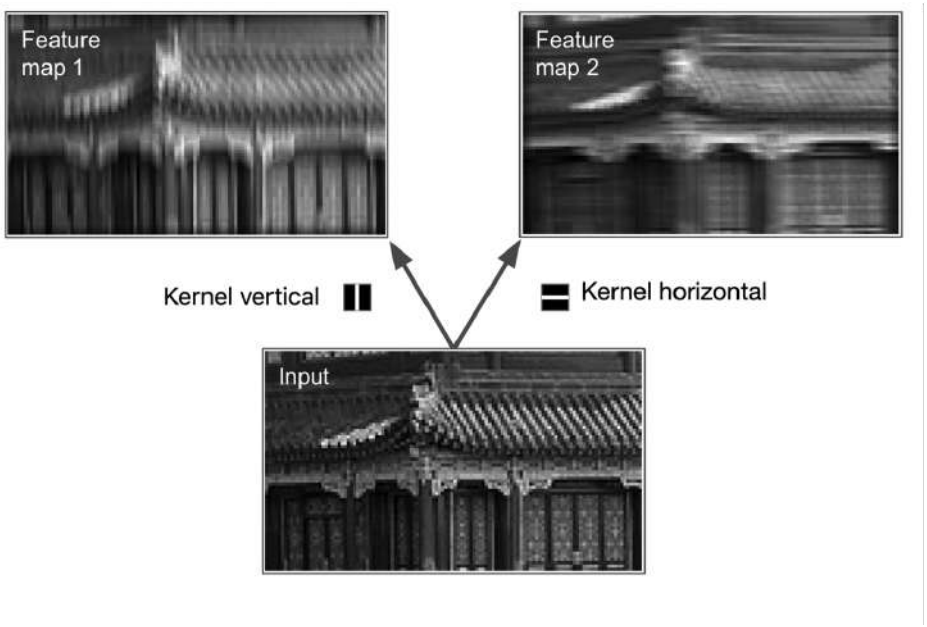


Figura 2.8: Resultado de aplicar 2 *kernels* simples, uno horizontal y otro vertical.
(Géron, 2022)

Lo que hace que las CNN sean poderosas es que, en lugar de tener que diseñar manualmente los *kernels*, a través del proceso de entrenamiento, las redes aprenden automáticamente los valores óptimos para los *kernels* de cada capa. Estos valores, los cuales son los pesos de las redes convolucionales, tienen la capacidad de detectar y representar características relevantes. El ajuste automático de pesos se logra mediante la propagación hacia atrás (*backpropagation*) del error utilizando algoritmos de optimización como el descenso por gradiente (sección 2.7.1).

2.3.3. *Stride*

Stride, que se traduce como "paso", indica la cantidad de píxeles que se saltan al aplicar un *kernel* sobre una imagen. En la figura 2.9, el *kernel* que se aplica sobre la imagen, tanto vertical como horizontalmente, es de 2 píxeles.

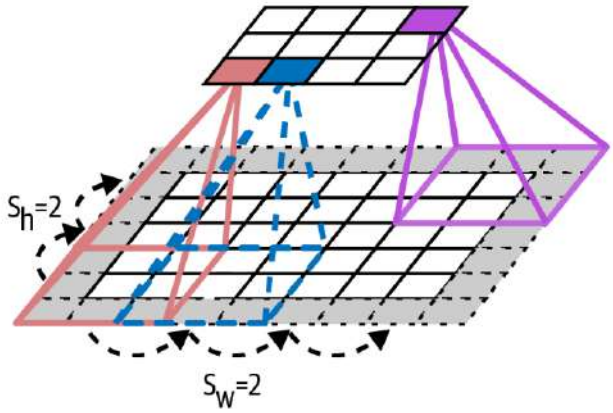


Figura 2.9: *Stride* de tamaño 2.
(Géron, 2022)

Cabe aclarar que el ejemplo mencionado anteriormente representa una imagen en blanco y negro. El formato más común de las imágenes no es este, sino que es el RGB (rojo, verde y azul por sus siglas en inglés), lo que implica que los *kernels* son tridimensionales. El tamaño de la salida de un kernel se determina por la altura, el ancho y la cantidad de canales de la entrada, junto con el tamaño del propio kernel.

2.3.4. *Pooling*

El *pooling* es una técnica empleada en el procesamiento de imágenes con el fin de simplificar la información sin perder los elementos más significativos. Se puede visualizar como una imagen dividida en pequeñas áreas o regiones, por ejemplo, de 3x3 píxeles. En este contexto, el *pooling* realiza el cálculo de un valor representativo para cada una de estas áreas, basándose en los valores de los píxeles que las componen. Este procedimiento contribuye a disminuir la cantidad de información y cálculos necesarios para comprender y procesar la imagen. Los dos tipos más comunes de *pooling* son el *average pooling*, que promedia los valores, y el *max pooling*, que selecciona el valor más grande.

Max Pooling: Para cada región, se selecciona el píxel con el valor más alto dentro de la misma. Este valor más alto se toma como la representación de esa región en la nueva imagen reducida, como se puede observar en la figura 2.10.

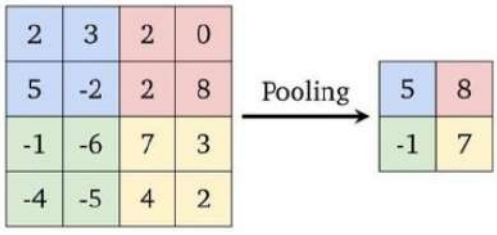


Figura 2.10: Resultado de aplicar *Max Pooling* 2x2.
(Guissous, 2019)

Avg Pooling: Al igual que antes, la imagen se divide en regiones, y para cada región, se calcula el promedio de los valores de los píxeles en esa región. Este valor promedio se utiliza como representante de esa región en la imagen reducida, como se puede observar en la figura 2.11.

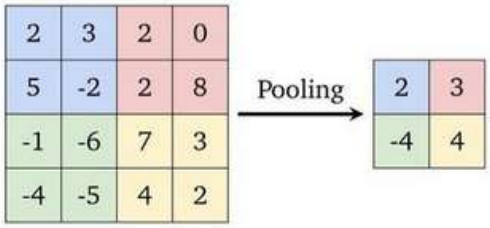


Figura 2.11: Resultado de aplicar *Avg Pooling* 2x2.
(Guissous, 2019)

2.3.5. *Padding*

Padding significa *relleno*. Se refiere al tipo de relleno que se agrega a los bordes de una imagen antes de aplicar operaciones que tengan como resultado una reducción de la dimensionalidad de la misma (por ejemplo, *pooling* y convoluciones). Esto se debe a que estas operaciones a menudo no representan adecuadamente los píxeles en los límites de la imagen resultante. Al agregar un relleno (*padding*), se reduce la incidencia de los efectos en los bordes en las imágenes (Szeliski, 2022). Se han desarrollado varias técnicas diferentes de relleno, como se observa en la figura 2.12. A su vez, en la misma se muestra el resultado de aplicar *padding* previo a una convolución que realiza un desenfoque gaussiano. A continuación, se describen las implicaciones de cada una de estas técnicas:

- *constant* (color de borde): establece todos los píxeles fuera de la imagen a un valor de borde en específico. Un caso particular de *constant* muy utilizado es *zero padding* el cual define el valor de borde en cero.
- *clamp* (de restricción): repite los píxeles del borde indefinidamente.
- *wrap* (cíclico): recorre la imagen de manera cíclica, repitiéndola como un mosaico en todas las direcciones.
- *mirror* (espejo): refleja los píxeles a través del borde de la imagen.

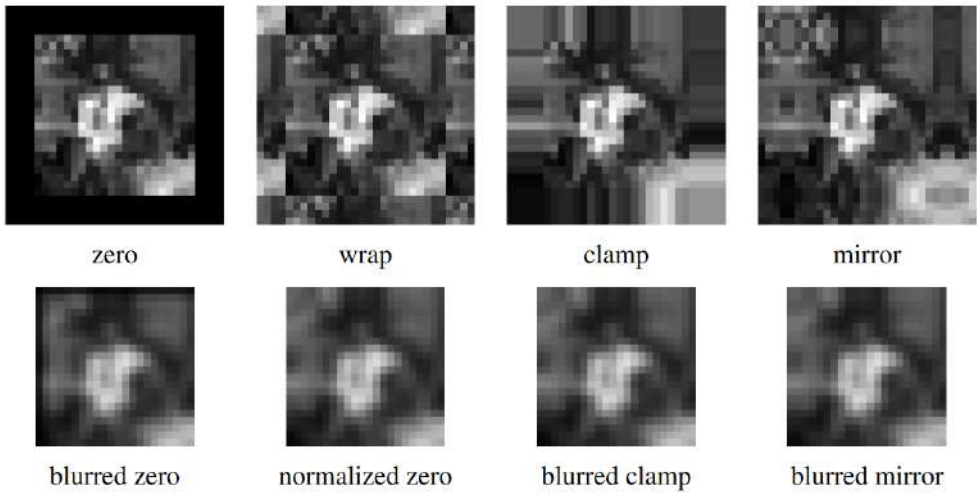


Figura 2.12: Resultados de aplicar desenfoque gaussiano con distinto *padding*. (Szeliski, 2022)

2.4. Detección de objetos

La detección de objetos es una tarea dentro del campo de visión por computadora (*computer vision*) que implica localizar y clasificar múltiples objetos dentro de una imagen (Géron, 2022). A diferencia de la clasificación de imágenes, que tiene como único objetivo asignar una sola etiqueta a una imagen completa, la detección de objetos va un paso más allá. Se detectan múltiples objetos y se predicen las coordenadas de los mismos junto con su etiqueta de clase correspondiente.

Las coordenadas que definen la ubicación de los objetos en la imagen se denominan cuadros delimitadores, conocidos en inglés como *bounding boxes*. La explicación detallada sobre estos cuadros y su codificación se presenta más adelante en la sección de anotaciones 2.4.1.

Con el objetivo de realizar estas tareas de localización y clasificación, se entrenan modelos basados en redes neuronales convolucionales. Estos modelos tienen como entrada una imagen con los objetos en cuestión y como salida, una lista de predicciones.

Estas predicciones incluyen tres datos esenciales: en primer lugar, los cuadros delimitadores; en segundo lugar, el identificador de la clase detectada dentro del cuadro; y finalmente, lo que se conoce como el *objectness score*, que representa la probabilidad de que un objeto exista en una región de interés dada.

2.4.1. Anotaciones

La anotación de datos, en el ámbito de la detección de objetos, implica la tarea de categorizar y localizar los objetos presentes en una imagen, con el propósito de facilitar el entrenamiento de modelos de aprendizaje automático.

En esta sección, se examinará la anotación de los cuadros delimitadores y su codificación en la detección de objetos. Además, se menciona la segmentación como otra forma de anotación que permite una comprensión más detallada de los objetos en una escena visual y potencialmente útil para la detección de objetos.

Cuadros delimitadores (*Bounding boxes)*

Para lograr la detección precisa de un objeto específico en una imagen, es esencial inicialmente establecer de manera única su ubicación. La detección de objetos se materializa mediante la definición de un rectángulo en la imagen, lo cual implica la utilización de al menos cuatro valores escalares. En situaciones donde hay múltiples clases de objetos, se requiere un escalar adicional para identificar la categoría a la que pertenece. Al determinar una zona rectangular en una imagen se define lo que se le llama cuadros delimitadores o *bounding boxes*. En general, los mismos se codifican con las coordenadas de dos esquinas opuestas del rectángulo [Xmin, Ymin, Xmax, Ymax], como se puede ver en la figura 2.13, pero existen muchas más combinaciones.

Antes de explicar los principales tipos de anotaciones, es importante destacar que en la convención utilizada sobre las imágenes, el cruce de ejes se da en la esquina superior izquierda de las mismas, es decir, el primer píxel de la esquina superior izquierda tiene la coordenada (0,0). Existen varias convenciones de cuadros delimitadores, a continuación se mencionarán brevemente las principales.

XYminXYmax: Esta convención utiliza las coordenadas (Xmin, Ymin) para el vértice superior izquierdo del rectángulo y las coordenadas (Xmax, Ymax) para el vértice inferior derecho del mismo. Este formato es utilizado por *PASCAL Visual Object Classes (VOC)* (Everingham, Van Gool, Williams, Winn, y Zisserman, 2010).

XYminWH: Aquí, las coordenadas (Xmin, Ymin) nuevamente representan el vértice superior izquierdo del rectángulo. Además, se agrega el ancho (*Width*) y la altura (*Height*) del rectángulo, en lugar de las coordenadas del vértice inferior derecho. Este formato es utilizado por *COCO (Common Objects in Context)* (Lin y cols., 2014).

XYcentWH: En esta convención, se utilizan las coordenadas (Xcent, Ycent) para representar el centro del rectángulo, y nuevamente se incluyen el ancho y la altura.

La mayoría de las veces, el formato utilizado para las coordenadas está dado en píxeles. Es decir, los valores de X varían entre 0 y W, y los de Y, entre 0 y H, siendo W y H el ancho y alto de la imagen respectivamente. Pero también existe otro, el formato YOLO, donde los valores de las coordenadas X e Y se

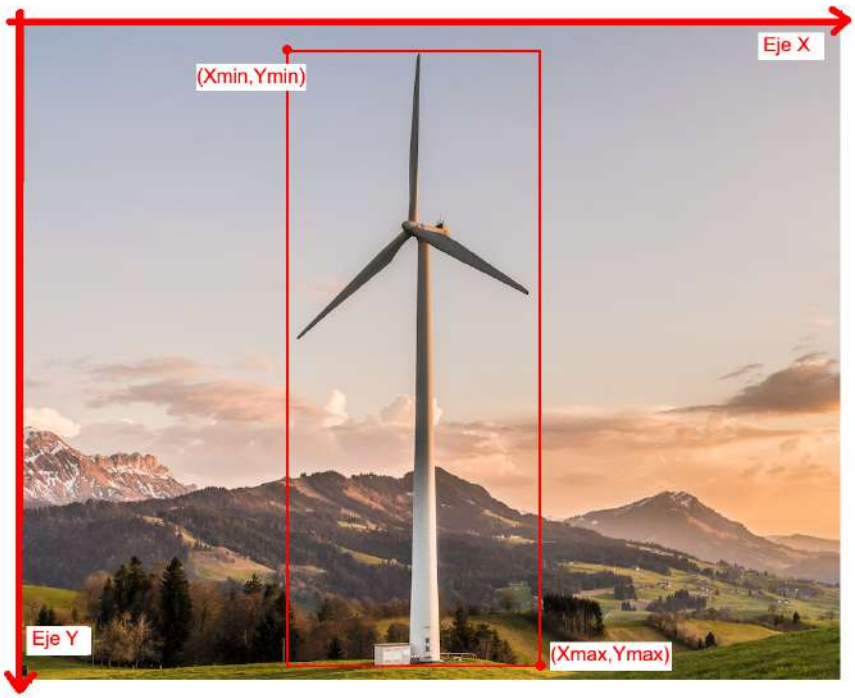


Figura 2.13: Visualización de anotación [Xmin,Ymin,Xmax,Ymax].
(Pexels, 2017)

normalizan entre 0 y 1 en relación con las dimensiones de la imagen. Entonces, en lugar de utilizar valores de píxeles absolutos, se utilizan valores relativos a la altura y ancho de la imagen. Esto permite de alguna forma "independizarse" del tamaño de las imágenes en las anotaciones.
Este formato es nombrado así porque está fuertemente relacionado con el modelo de detección YOLO, abordado en la subsección 2.8.2.

Segmentación

En función de la tarea, es crucial seleccionar la anotación más adecuada. En este proyecto grado, centrado en la tarea de detección de objetos, se ha elegido emplear anotaciones de cuadros delimitadores, dado que para la detección temprana de daños es suficiente con los mismos. A pesar de esto, es importante destacar que existen otros formatos de anotaciones disponibles.

La segmentación de instancias va más allá de la simple detección, el resultado de los modelos de segmentación es un conjunto de máscaras o segmentos que clasifican a nivel de píxel cada instancia, acompañados de etiquetas de clase y puntuaciones de confianza, un ejemplo puede ser visto en la figura 2.14. Este enfoque es valioso cuando se busca no solo la ubicación de los objetos en una imagen, sino también la comprensión de su forma precisa (Ultralytics, 2023). Estos modelos de segmentación generalmente tienen más parámetros y son más costosos computacionalmente (memoria requerida, tiempo y operaciones de punto flotante), tanto para el entrenamiento como para la inferencia.

En la tabla 2.1 se puede visualizar una comparación de las métricas obtenidas para los modelos YOLOv8n y YOLOv8n-seg, para tareas específicas como detección de objetos y segmentación de instancias, respectivamente. La misma fue realizada por Ultralytics sobre el dataset de COCO, que contiene imágenes de tamaño 640x480 píxeles.

La decisión de no emplear anotaciones segmentadas no solo se basó en su costo computacional más elevado, sino también en la falta de conjuntos de datos debidamente anotados. La carencia de *datasets* de entrenamiento apropiados, que contengan anotaciones de segmentación de instancias, dificultó su incorporación en el marco de este proyecto.

Modelo	parámetros (M)	Velocidad en A100 (ms)	FLOPs (B)
YOLOv8n	3.2	0.99	8.7
YOLOv8n-seg	3.4	1.21	12.6

Tabla 2.1: Comparación de modelos de YOLOv8 para detección y segmentación.

Figura 2.14: Ejemplo de instancias segmentadas.
(Bolya, Zhou, Xiao, y Lee, 2019)

2.4.2. Intersección sobre unión (IoU)

Como su nombre lo indica, la métrica de *Intersection over Union* (IoU) resulta de dividir el área de la intersección entre dos *bounding boxes* por el área de su unión, proporcionando un valor en el rango de 0 a 1. Este valor escalar indica cuánta coincidencia existe entre los dos rectángulos, siendo 0 indicativo de ninguna superposición y 1 indicativo de una coincidencia perfecta. En la figura 2.15 se puede visualizar una representación gráfica de esta métrica.

En general, esta medida es utilizada para decidir cuándo una predicción es válida. Si el IoU entre el cuadro delimitador de la predicción y el cuadro delimitador de un objeto anotado de la misma clase es mayor a cierto umbral, se considera como correcta la predicción. El rango de umbrales más común se da generalmente entre 0.5 y 0.95. Además de ser usado como métrica para determinar la correctitud de una predicción, es también usado en algoritmos cómo el de *Non-Max Supression* explicado en la subsección 2.4.4. Por otro lado, existen generalizaciones de esta métrica que la adaptan para ser una función de localización viable, como se explica en la subsección 2.6.1.

2.4.3. *Anchor boxes*

Los *anchor boxes*, o cuadros ancla, son componentes fundamentales en ciertos modelos de detección de objetos basados en redes convolucionales. Consisten en rectángulos predefinidos de diversos tamaños y relaciones de aspecto que desempeñan un papel clave en la segmentación de la imagen en regiones candidatas para la extracción de características y, finalmente, la detección.

Estos cuadros ancla actúan como puntos de referencia para las predicciones del modelo de detección. Cada *anchor box* implica una suposición sobre el tamaño y la forma de un objeto en una imagen, lo que permite al modelo realizar predicciones de manera más eficiente al tener en cuenta una variedad de posibles ubicaciones y tamaños de objetos.

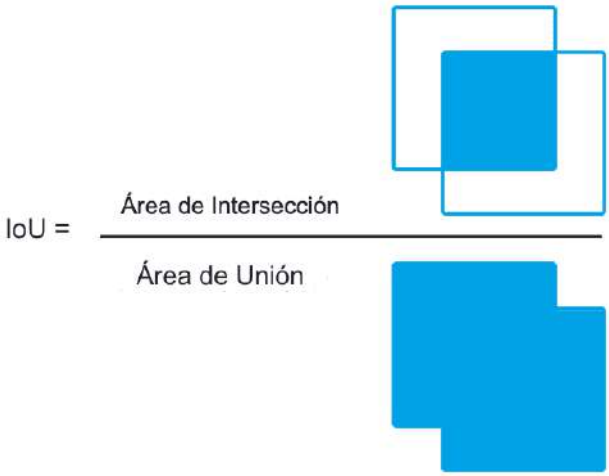


Figura 2.15: Visualización de IoU.
(Pyimagesearch, 2016)

2.4.4. *Non-Max Supression*

Esta operación, tal como sugiere su denominación, tiene como objetivo eliminar las predicciones de cuadros delimitadores que puedan presentar superposiciones. En diversas situaciones, es común que las predicciones se acumulen en una única instancia, generando una redundancia de cuadros. Por ende, la finalidad de esta operación es suprimir de manera eficiente estas superposiciones, evitando la presencia innecesaria de cuadros delimitadores.
Un posible algoritmo para implementar esta operación de forma iterativa se encuentra detallado en (Géron, 2022). En cada iteración, se selecciona la predicción con mayor *objectness score* y es comparada con el resto de predicciones de esa clase, aquellas que superen un IoU mayor a cierto umbral con esta predicción son descartadas, dado que se asume que predicen el mismo objeto. En la figura 2.16, se destaca en verde la anotación con mayor *objectness score* que permanece después de aplicar este algoritmo, mientras que las anotaciones en rojo son descartadas. Este parámetro de límite para el IoU es configurable en la mayoría de los modelos de detección, entre ellos está por ejemplo FasterRCNN sobre el cual se detalla en más profundidad en la sección 2.8.2.

Para abordar el problema de detección de daños superficiales, resulta lógico establecer límites reducidos para el IoU, ya que se tiene conocimiento de que no hay instancias superpuestas. En términos generales, los daños no tienden a solaparse entre sí, a diferencia de escenarios distintos, como la detección de multitudes, donde sí se pueden presentar superposiciones.

2.4.5. Aumentación de datos *(Data Augmentation)*

La aumentación de datos es una estrategia esencial en el entrenamiento de modelos de detección de objetos, especialmente cuando se trabaja con conjuntos de datos limitados. Esta técnica de generación de datos artificiales se ha empleado en diversas áreas y, desde hace bastante tiempo, en la detección de objetos, como se evidencia en el trabajo pionero de AlexNet (Krizhevsky, Sutskever, y Hinton, 2012).

Consiste en la creación de imágenes artificiales a partir de imágenes reales, introduciendo variaciones y perturbaciones que enriquecen la diversidad del conjunto de entrenamiento. En la imagen 2.17 se pueden observar varios ejemplos de aumentaciones. La creación de estas imágenes artificiales se hace a través de transformaciones de las originales. Existen transformaciones geométricas, transformaciones cromáticas, inclusive transformaciones que agregan ruido aleatorio o gaussiano, entre otras. Esta técnica no solo mejora la capacidad del modelo para generalizar a nuevas situaciones, sino que también ayuda a mitigar el riesgo de sobreajuste.

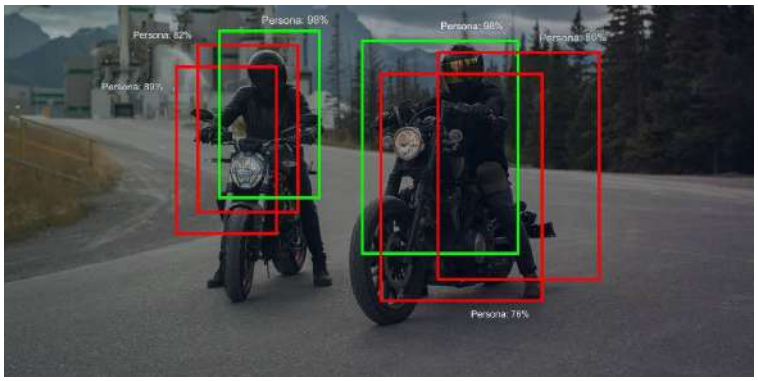


Figura 2.16: Anotaciones después de correr Non-Max Supression.
(Analyticsvidhya, 2020)

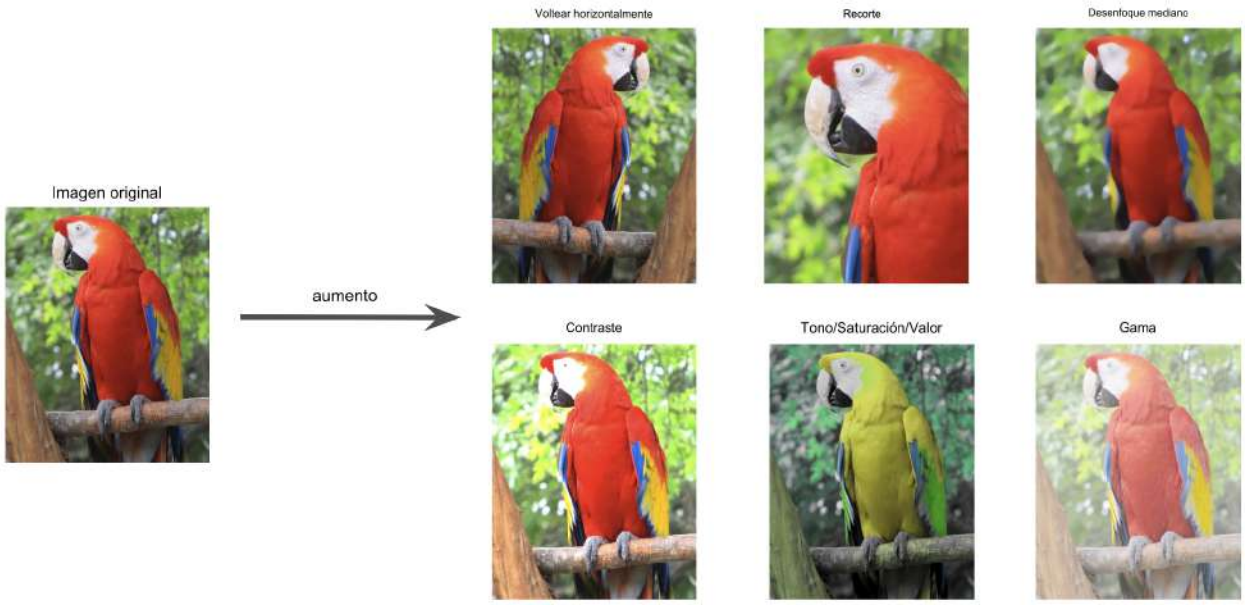


Figura 2.17: Ejemplos de aumentación de una imagen.
(Buslaev y cols., 2020)

2.5. Medidas de desempeño

En este capítulo, se introducirán los conceptos clave necesarios para definir las medidas de rendimiento más comunes empleadas en la evaluación de modelos de aprendizaje automático, con un enfoque particular en modelos de detección de objetos.

2.5.1. Matriz de confusión

La matriz de confusión, como su nombre lo indica, sirve para mostrar de alguna forma cuánto una clase es confundida con otra. Cada columna de la matriz representa las predicciones de cada clase, mientras que cada fila representa las verdaderas instancias de la clase.

El ejemplo más básico de una matriz de confusión ocurre cuando hay solo dos clasificaciones posibles. En el contexto de la detección de objetos, esto sucede cuando solo se está tratando de detectar una única clase en una imagen. En esta situación, hay dos resultados posibles: o se detecta el objeto, o no se detecta y se clasifica como fondo. La matriz de confusión para este caso se define en 2.4.

$$\begin{bmatrix} TP & FP \\ FN & TN \end{bmatrix} \tag{2.4}$$

Donde:

- TP = verdaderos positivos.
- FP = falsos positivos.
- FN = falsos negativos.
- TN = verdaderos negativos.

A partir de esta definición, si se extiende la idea, se puede definir la matriz teniendo N clases, lo que implica un tamaño de matriz $N + 1 \times N + 1$. En la figura 2.18 se puede ver un ejemplo de una matriz de confusión de tamaño 3x3.

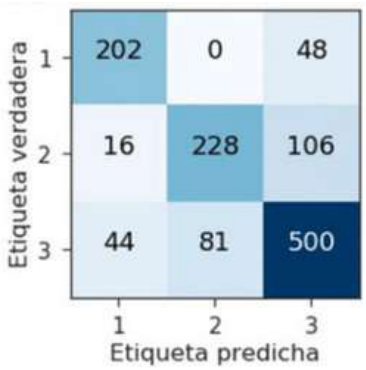


Figura 2.18: Ejemplo de matriz de confusión de tamaño 3x3.
(Garcia, 2018)

El resultado ideal de una matriz de confusión sería una matriz diagonal, lo que significa que todas las predicciones son correctas. En otras palabras, todos los elementos predichos como pertenecientes a una clase específica coinciden con la verdadera pertenencia a esa clase.

Exactitud (*Acurraccy*)

Es una medida que evalúa la proporción de predicciones correctas realizadas por un modelo, en relación con el total de predicciones realizadas sobre todas las clases. Se calcula con 2.5.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.5}$$

La exactitud es una buena métrica que comprende la capacidad modelo en general para las predicciones, pero puede no ser muy representativa en casos de clases desbalanceadas. Esto se mitiga calculando la exactitud de forma separada para cada clase.

Precisión

La precisión es una métrica que mide con qué frecuencia un modelo de aprendizaje automático predice correctamente una clase en particular. Se puede calcular con 2.6 dividiendo el número de predicciones correctas entre el total de predicciones para una clase en particular.

$$\text{Precisión} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.6}$$

Sensibilidad (*Recall*)

La sensibilidad es una métrica que mide con qué frecuencia un modelo de aprendizaje automático identifica correctamente las instancias positivas, de entre todas las muestras positivas en el conjunto de datos para una clase en particular. Se define en 2.7.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.7}$$

Curva PR (*Precision-Recall Curve*)

La curva de Precisión-Sensibilidad (PR) es una representación gráfica del rendimiento de un modelo de clasificación en diferentes niveles de umbral de confianza. Esta curva ilustra cómo varía la precisión y la sensibilidad a medida que se ajusta el umbral de confianza del modelo. Generalmente, si el *objectness score* pasa el umbral de confianza definido, la predicción se toma como válida, de lo contrario se descarta.

La figura 2.19 muestra un ejemplo de cómo se ve una curva PR para la predicción de una clase. A mayor umbral de confianza se exige al modelo mayor seguridad en la predicción, por lo tanto, su precisión es alta. Por otro lado, bajar el umbral tiene como consecuencia un empeoramiento en la precisión a cambio de un mayor recall. Esto ocurre porque las predicciones, que previamente no se consideraban válidas, ahora sí lo son. Esta mayor consideración conlleva un aumento en la cantidad de errores y, por ende, a una disminución en la precisión del modelo.

F1 *Score*

Es una métrica comúnmente utilizada en problemas de clasificación para evaluar el rendimiento de un modelo, especialmente cuando las clases están desbalanceadas.
Esta métrica, que se calcula con 2.8, combina la *precisión* y *recall* en un único valor, proporcionando una medida más equilibrada del rendimiento del modelo.

$$F1\ Score = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.8}$$

2.5.2. Métricas para detección de objetos

Es fundamental destacar que las métricas discutidas anteriormente son específicas de problemas de clasificación. Sin embargo, en el caso de la detección de objetos, además de la clasificación, también entra en juego la localización. Como se explicó en la subsección 2.4.2, se utiliza un umbral de IoU para determinar la precisión de la localización. Si la localización no es lo suficientemente precisa, a pesar de que la clasificación sea correcta, se toma la predicción como un falso positivo. Por lo tanto, es posible definir $\text{Precision}_{\text{umbral}}$ y $\text{Recall}_{\text{umbral}}$, donde el umbral puede variar entre cero y uno. En la literatura, estos umbrales suelen estandarizarse en 0.5 o en un rango de 0.5 hasta 0.95.

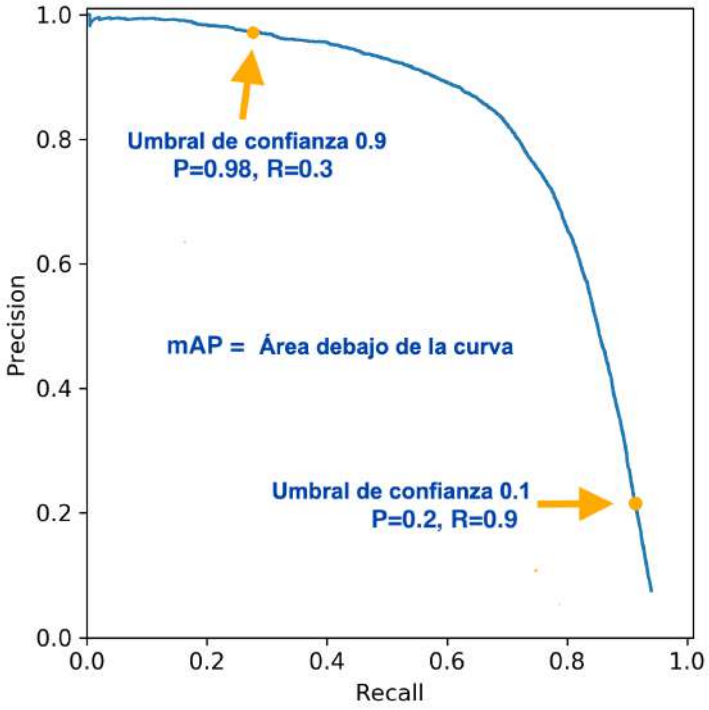


Figura 2.19: Ejemplo de curva PR.

mAP ***(Mean Average Precision)***

Esta métrica, definida como 2.9, es comúnmente utilizada para evaluar el rendimiento de modelos de detección de objetos en problemas de visión por computadora. Proporciona una medida de la precisión promedio a lo largo de múltiples clases y umbrales de confianza.

$$mAP = \frac{1}{|C|} \sum_{i=1}^{|C|} AP_i \tag{2.9}$$

Donde C es el número total de clases y AP_i, definido en 2.10, es la precisión promedio para la clase i.

$$AP = \int_0^1 \text{precision}(r)\, dr \tag{2.10}$$

Donde precision(r) es la precisión con un umbral de confianza r.

$\text{mAP}_{0.5}$ y $\text{mAP}_{0.5:0.95}$: Según la definición de la página oficial de COCO, esta métrica se calcula como la media de *Average Precision* (AP) sobre todas las clases en un rango de umbrales de IoU, desde 0.5 hasta 0.95. Esto proporciona una evaluación más completa del rendimiento del modelo, considerando diferentes niveles de superposición entre predicciones e instancias reales. Exigir un mayor umbral de IoU implica que el modelo sea más preciso en la localización de las predicciones.

2.6. Funciones de costo

En el campo de la detección de objetos, así como en el de aprendizaje automático, las funciones de costo son cruciales en el proceso de entrenamiento de modelos. Su papel fundamental radica en medir la discrepancia entre las predicciones generadas por el modelo y las etiquetas reales asociadas a los objetos en las imágenes. Dada la naturaleza de las anotaciones, tenemos dos categorías de funciones de costo, las de localización y las de clasificación.

2.6.1. Funciones de costo para localización

Estas funciones de costo tienen como entrada la *bounding box* predicha y la real previamente anotada. Como salida, se tiene un escalar que representa qué tan distante se encuentra la predicción con la anotación real.

Error Absoluto *(L1 Loss)*

La función de costo L1, también conocida como Error Absoluto o Error Absoluto Medio (MAE, por sus siglas en inglés), se calcula en 2.11 como la media de las diferencias absolutas entre los valores predichos y reales. Esta métrica mide la magnitud promedio de los errores sin considerar su dirección. Es una medida intuitiva, pero tiene la desventaja de no ser derivable en cero.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |x_i - \hat{x}_i| + |y_i - \hat{y}_i| + |\text{width}_i - \hat{\text{width}}_i| + |\text{height}_i - \hat{\text{height}}_i| \tag{2.11}$$

Dónde:

n es el número total de obeservaciones.

$x_i, y_i, \text{width}_i.\text{height}_i$ son las coordenadas y dimensiones reales de la bounding box i.

$\hat{x}_i, \hat{y}_i, \hat{\text{width}}_i, \hat{\text{height}}_i$ son las predicciones correspondientes para la bounding box i.

Error Cuadrático *(L2 Loss)*

La función de costo de Error Cuadrático Medio (MSE, por sus siglas en inglés) busca minimizar la suma de los cuadrados de las diferencias entre las localizaciones predichas y los valores reales de localización, como se observa en 2.12.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} \left((x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\text{width}_i - \hat{\text{width}}_i)^2 + (\text{height}_i - \hat{\text{height}}_i)^2\right) \tag{2.12}$$

Donde:

n es el número total de observaciones.

$x_i, y_i, \text{width}_i, \text{height}_i$ son las coordenadas y dimensiones reales de la bounding box i.

$\hat{x}_i, \hat{y}_i, \hat{\text{width}}_i, \hat{\text{height}}_i$ son las predicciones correspondientes para la bounding box i.

Dado que se toma el cuadrado de la diferencia entre las predicciones, el error crece cuadráticamente, lo cual para el caso de *outliers* (valores atípicos) provoca que estos términos influyan fuertemente en el error final. Esta situación presenta desventajas, dado que se busca en general un modelo robusto, capaz de exhibir un rendimiento sólido tanto en términos globales como en la mayoría de los casos.

L1 suave *(Smooth L1 Loss)*

L1 suave *(Smooth L1 Loss)* es una combinación entre la función de costo L1 (Error Absoluto) y la función de costo L2 (Error Cuadrático). Esta función, definida en 2.13, se introdujo para abordar algunos

de los desafíos asociados con L2, como la sensibilidad a *outliers* y el problema de L1 que no es derivable en cero. En la figura 2.20 se puede comparar la función L1, L2 y *Smooth* L1.

$$\text{Smooth L1}(x) = \begin{cases} \frac{1}{2} \cdot L2(x) & \text{si } |x| < 1 \\ L1(x) - \frac{1}{2} & \text{en otro caso} \end{cases} \tag{2.13}$$

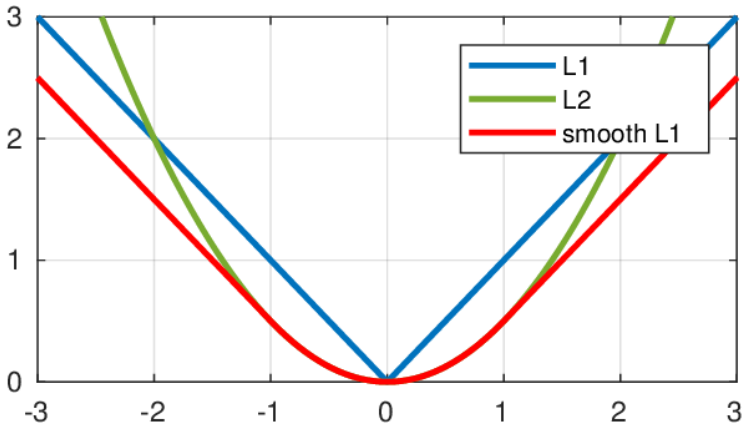


Figura 2.20: L1, L2 y Smooth L1.
(Feng, Kittler, Awais, Huber, y Wu, 2018)

***Generalized-IoU* (GIoU)**

La mejor forma de optimizar un modelo basándose en una métrica es utilizar la métrica en sí como función de costo. El problema con utilizar la métrica IoU como función de costo es que en casos donde no hay solapamiento entre las predicciones y anotaciones el resultado es 0, volviéndose prácticamente imposible optimizar con descenso por gradiente. El siguiente documento (Rezatofighi y cols., 2019) ataca este problema y las debilidades de IoU generando una nueva métrica generalizada GIoU. En la figura 2.21 se puede apreciar la ecuación junto a una intuición visual.

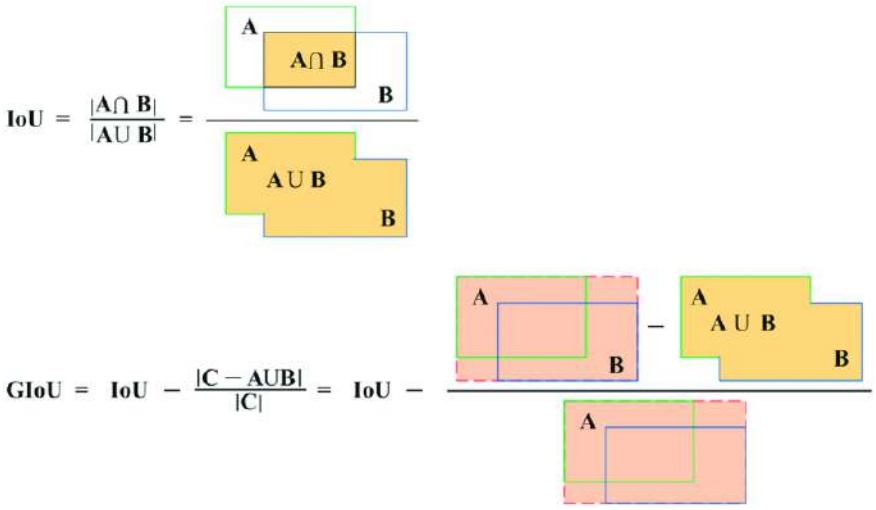


Figura 2.21: GIoU vs IoU.
(Fu y cols., 2023)

La misma mantiene propiedades deseables de una métrica cómo la invarianza en distintas escalas y la no-negatividad, entre otras. Esta función de costo, a pesar de ser un poco más compleja de calcular, a diferencia

de otras cómo L1 o MSE, permite una mejor localización según el documento. Además de GIoU se han propuesto otras métricas como Distance-IoU (DIoU) (Zheng y cols., 2019), Complete-IoU (CIoU) (Zheng y cols., 2021) las cuales buscan continuar mejorando la eficiencia y convergencia de los modelos.

2.6.2. Funciones de costo para clasificación

Entropía cruzada

La Entropía Cruzada, también conocida como pérdida de entropía cruzada o *cross-entropy loss*, se ha establecido como una opción común y efectiva para problemas de clasificación, incluidos aquellos relacionados con la detección de objetos. Esta métrica, definida en 2.14, cuantifica la discrepancia entre la distribución de probabilidad que el modelo predice para las clases y la distribución real de clases en el conjunto de datos.

$$H(y, \hat{y}) = -\sum_{i} y_i \log(\hat{y}_i) \tag{2.14}$$

Donde:

- y_i es la etiqueta verdadera para la clase i.
- $\hat{y}_i$ es la probabilidad predicha por el modelo para la clase i.

Pérdida Focal (*Focal loss*)

La Pérdida Focal es una variante de la función de pérdida de entropía cruzada utilizada en problemas de clasificación, diseñada para abordar el desequilibrio entre clases. Esta función fue introducida por primera vez en el trabajo "Focal Loss for Dense Object Detection" (Lin, Goyal, Girshick, He, y Dollár, 2018) junto a la arquitectura de RetinaNet. Este desequilibrio se da porque en general hay una mayor cantidad de instancias negativas (fondo) en comparación con las positivas (objetos de interés). Esta función se centra en mejorar el proceso de aprendizaje, asignando menor peso a los ejemplos negativos fácilmente clasificables y mayor peso a los ejemplos positivos considerados más difíciles. De este modo, *Focal loss* busca contrarrestar la influencia dominante de las instancias negativas en la función de costo. Cabe destacar que esta función de costo focal constituye una variante de la función de entropía cruzada y su definición puede verse en 2.15.

$$\text{Focal Loss}(p_t) = -\alpha_t (1 - p_t)^{\gamma} \log(p_t) \tag{2.15}$$

Donde:

- p_t es la probabilidad de la clase verdadera.
- α_t es el factor de equilibrio que se utiliza para contrarrestar el desequilibrio de clases.
- γ es un parámetro que ajusta la facilidad o dificultad con la que se clasifican los ejemplos.

2.7. Optimizadores

Un optimizador es un algoritmo que se utiliza para ajustar los parámetros de un modelo (pesos) con el objetivo de minimizar una función de costo. El algoritmo de optimización guía el ajuste de los pesos mediante la actualización iterativa de los mismos, tomando en cuenta la derivada de la función de costo con respecto a sus pesos.

Existen diversos optimizadores, cada uno con enfoques y propiedades específicas. Entre los optimizadores más comunes, se destacan el descenso por gradiente, descenso por gradiente en lotes, descenso por gradiente estocástico, descenso por gradiente con momentum y Adam. Estos optimizadores difieren en la manera en que adaptan la tasa de aprendizaje o incorporan "momento" para acelerar o suavizar las actualizaciones de parámetros durante el entrenamiento del modelo.

2.7.1. Descenso por gradiente

El descenso por gradiente es un algoritmo genérico de optimización versátil que se utiliza para encontrar soluciones óptimas en diversos problemas. La esencia del descenso por gradiente radica en la actualización de los parámetros en función del gradiente de la función de costo. El proceso comienza con la inicialización del vector de parámetros θ, por ejemplo, de forma aleatoria. A continuación, el algoritmo avanza de manera incremental en la búsqueda de un mínimo, tomando pequeños pasos dirigidos en la dirección opuesta al gradiente para reducir la función de costo. Un factor crítico en el descenso por gradiente es la tasa de aprendizaje, un hiperparámetro que determina la magnitud de cada paso. Una tasa de aprendizaje demasiado pequeña implica numerosas iteraciones y un tiempo considerable para converger, mientras que una tasa demasiado alta puede llevar a saltos excesivos, incluso resultando en la divergencia del algoritmo. Además, es importante destacar que no todas las funciones de costo tienen una forma regular y suave, lo que complica la convergencia hacia el mínimo (Géron, 2022).

2.7.2. Descenso por gradiente por Lotes

En el descenso por gradiente por lote, el vector de gradiente, representado como $\nabla J(\theta)$, comprende todas las derivadas parciales de la función de costo, una por cada parámetro del modelo. Es importante destacar que esta implementación implica realizar cálculos sobre el conjunto completo de entrenamiento en cada paso del descenso de gradiente. Por esta razón, el algoritmo recibe el nombre de "descenso de gradiente por lotes", ya que utiliza la totalidad del conjunto de datos de entrenamiento en cada iteración. Como consecuencia, resulta ser bastante lento en conjuntos de entrenamiento muy grandes (Géron, 2022).

En 2.16 se muestra cómo se actualiza el parámetro θ.

$$\theta = \theta - \eta \cdot \nabla J(\theta) \tag{2.16}$$

Dónde:

- θ son los parámetros del modelo.
- η es la tasa de aprendizaje.
- $J(\theta)$ es la función de costo.
- $\nabla J(\theta)$ es el gradiente de la función de costo con respecto a los parámetros.

2.7.3. Descenso por gradiente Estocástico (SGD)

El principal inconveniente del descenso por gradiente por lotes radica en que utiliza todo el conjunto de entrenamiento para calcular gradientes en cada paso, lo que lo hace muy lento cuando el conjunto de entrenamiento es grande. En cambio, el descenso por gradiente estocástico adopta un enfoque más dinámico. Selecciona una instancia aleatoria del conjunto de entrenamiento en cada paso para calcular los gradientes, como se observa en 2.17. Esta técnica acelera significativamente el algoritmo.

No obstante, la naturaleza estocástica (aleatoria) de este algoritmo introduce variaciones que no se observan en el descenso por gradiente por lotes. En lugar de disminuir de manera uniforme hasta alcanzar

el mínimo, la función de costo experimenta fluctuaciones, oscilando en una dirección general descendente. Con el tiempo, converge hacia una posición muy cercana al mínimo, pero persiste en fluctuar en lugar de estabilizarse. Cuando el algoritmo finaliza, los valores finales de los parámetros son buenos, pero pueden no ser óptimos en el sentido estricto (Géron, 2022).

$$\theta = \theta - \eta \cdot \nabla J_i(\theta) \tag{2.17}$$

2.7.4. *Momentum*

El algoritmo de descenso por gradiente tradicional no considera los valores previos de los gradientes, lo que implica que avanza lentamente cuando se encuentra con gradientes locales pequeños. En contraste, la optimización mediante *momentum* otorga especial importancia a los gradientes anteriores. Para explicarlo de manera más sencilla, el gradiente puede considerarse como la "velocidad" con la que se mueve hacia un óptimo local. Pero, al aplicar el *momentum*, estamos considerando no solo la velocidad actual (es decir, el gradiente actual), sino también cómo ha estado cambiando esa velocidad en el pasado, lo que puede verse como una "aceleración".

Ahora bien, para simular algún tipo de mecanismo de fricción y prevenir un crecimiento excesivo del *momentum*, el algoritmo incorpora un nuevo hiperparámetro β a 2.17, denominado *coeficiente de fricción*, dando como resultado 2.18. Este coeficiente debe ajustarse en un rango entre 0 (alta fricción) y 1 (sin fricción) (Géron, 2022).

$$\begin{aligned} m &\leftarrow \beta \cdot m - \eta \nabla_\theta J(\theta) \\ \theta &\leftarrow \theta + m \end{aligned} \tag{2.18}$$

2.7.5. Adam y AdamW

Adam y AdamW son algoritmos de optimización avanzados y sus fórmulas matemáticas son bastante más complejas que las de métodos más simples.

Adam combina la adaptación de la tasa de aprendizaje con la idea de utilizar momentos para seguir la dirección y velocidad del gradiente. Calcula medias móviles exponenciales de los gradientes y sus cuadrados, siendo eficiente computacionalmente y adecuado para problemas con grandes conjuntos de datos o alta dimensionalidad.

AdamW es una variante de Adam, que introduce la penalización directa en la magnitud de los pesos (*weight decay*), mejorando la generalización del modelo y abordando problemas de estabilidad en la optimización de pesos, lo que ayuda a prevenir el sobreajuste.

2.8. Modelos

2.8.1. Tipos de modelos

Existe una amplia variedad de modelos de detección de objetos, los cuales pueden clasificarse en dos subconjuntos: detectores de una sola etapa (*one stage detector*) o de dos etapas (*two stage detector*).

One stage detector

Los detectores de una etapa, como YOLO (sección 2.8.2) y SSD (sección 2.8.2), realizan la detección de objetos "en una sola pasada" a través de la imagen. Estos modelos generan predicciones de detección directamente utilizando una red neuronal convolucional, sin etapas adicionales de procesamiento. Se caracterizan por ser eficientes.

Two stage detector (Region-CNN)

Los detectores de dos etapas, como Faster R-CNN (sección 2.8.2), se estructuran en dos fases esenciales: la generación de propuestas de regiones (*region proposal generation*) y la clasificación/regresión de estas regiones. En el pasado, la primera etapa solía ser implementada empleando algoritmos como *Selective Search* (R. B. Girshick, Donahue, Darrell, y Malik, 2014), pero en la actualidad, las redes neuronales también han asumido un papel predominante en este proceso.

En la segunda etapa, estas regiones propuestas son sometidas a un proceso de clasificación y regresión final mediante otra red neuronal. Este enfoque puede resultar en una mayor precisión en la detección al emplear regiones propuestas más refinadas. Sin embargo, suelen exhibir un tiempo de ejecución más lento en comparación con los detectores de una sola etapa.

2.8.2. Modelos evaluados

A continuación, se presenta un resumen de los modelos utilizados en el proyecto, los cuales fueron seleccionados tras la revisión de la literatura en la sección 3. Esta revisión permitió identificar y elegir los modelos más prometedores en relación con los objetivos específicos del proyecto.

You Only Look Once(YOLO)

Este modelo abreviado por la frase en inglés de "Solo se mira una vez" como indica su nombre, es un modelo de detección en una sola etapa. El primer modelo se remonta al año 2016 (Redmon, Divvala, Girshick, y Farhadi, 2016), el mismo se destacó por tener una muy buena eficiencia sin perder mucha precisión, algo sumamente importante para aplicaciones de detección en tiempo real. En la figura 2.22, se presenta de manera general una representación visual de cómo opera el modelo. Inicialmente, YOLO fragmenta la imagen de entrada en una cuadrícula de dimensiones S x S. Posteriormente, una red neuronal convolucional procesa la imagen globalmente, extrayendo características relevantes. En una única evaluación de la red, se realizan predicciones simultáneas de múltiples cajas delimitadoras (*bounding boxes*) y las respectivas probabilidades de clasificación asociadas a esas cajas. Para refinar los resultados, se implementa el algoritmo de *Non-Max Suppression*, el cual elimina detecciones redundantes y conserva únicamente las más confiables, estas detecciones finales son la salida del modelo.

Single Shot Multibox Detector (SSD)

El modelo SSD de detección de objetos fue presentado por primera vez en 2015 (Liu y cols., 2015) y demostró ser altamente efectivo y eficiente. Es un detector de una etapa, esto lo hace extremadamente rápido tanto para predecir, así como para entrenar. SSD utiliza una arquitectura de red neuronal convolucional y tiene la capacidad de detectar objetos en diferentes escalas mediante la incorporación de múltiples capas de detección.

A nivel de varios *feature maps*, SSD predice múltiples cuadros delimitadores (*bounding boxes*) y las probabilidades asociadas a la presencia de diferentes categorías de objetos. Esto se logra mediante la aplicación de

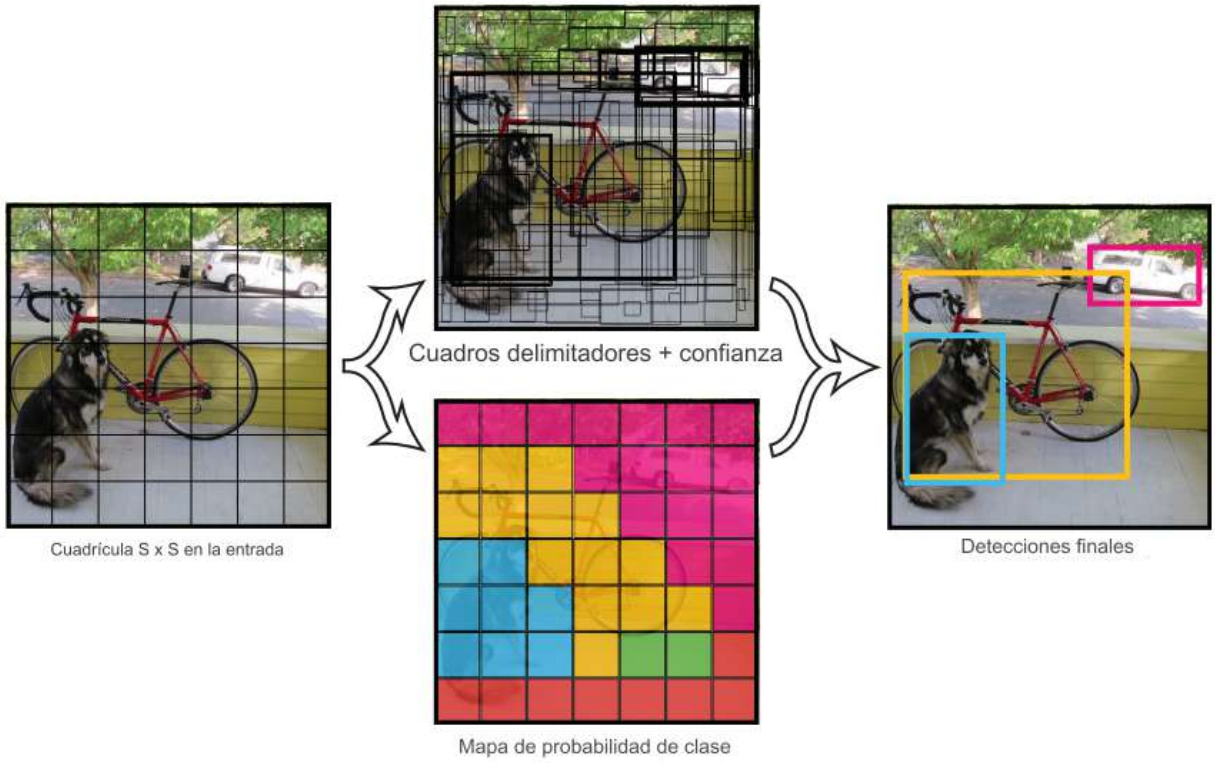


Figura 2.22: Representación visual del modelo YOLO.
(Redmon y cols., 2016)

filtros convolucionales en diferentes escalas y aspectos de los *feature maps*. Finalmente, se aplica el algoritmo *Non-Max Suppression* para filtrar las detecciones redundantes y generar las detecciones finales de objetos.

En la figura 2.23 se encuentra la arquitectura del modelo presentada en (Liu y cols., 2015), comparada con el modelo YOLO. Ambos modelos utilizan una cuadrícula de celdas para dividir la imagen, pero su enfoque difiere ligeramente. SSD emplea múltiples *anchor boxes* por celda para manejar objetos de diferentes escalas y aspectos, mientras que YOLO predice un número fijo de bounding boxes directamente desde las celdas de la cuadrícula.

Faster RCNN

Este modelo introducido inicialmente en (Ren, He, Girshick, y Sun, 2016), forma parte de la categoría de modelos de detección de objetos de dos etapas. El mismo es el resultado de una mejora del modelo anterior conocido como Fast R-CNN (R. B. Girshick, 2015). En este refinamiento, se realiza una modificación en la predicción de las regiones de interés, proponiendo abordar el problema mediante la incorporación de una red convolucional en lugar de depender de métodos convencionales de la época, como el algoritmo *Selective Search*.

La red propuesta es denominada *Region Proposal Network* (RPN) y es un componente esencial que utiliza anclas de diferentes escalas y aspectos, para generar propuestas de regiones directamente desde las características convolucionales de la imagen. Esta innovación elimina la necesidad de métodos externos para la generación de propuestas y permite un entrenamiento *end-to-end* más efectivo.

Faster R-CNN es compatible con varios extractores de características. Estos extractores son representados en la figura 2.24 por las capas convolucionales, y se encargan de generar los mapas de características (*feature maps*) para que luego, la RPN pueda generar propuestas de regiones que posiblemente contengan objetos. Posteriormente, una capa de clasificación asigna categorías a las propuestas de regiones y predice las ubicaciones precisas de los objetos dentro de estas regiones. Para facilitar este proceso, se utiliza ROI

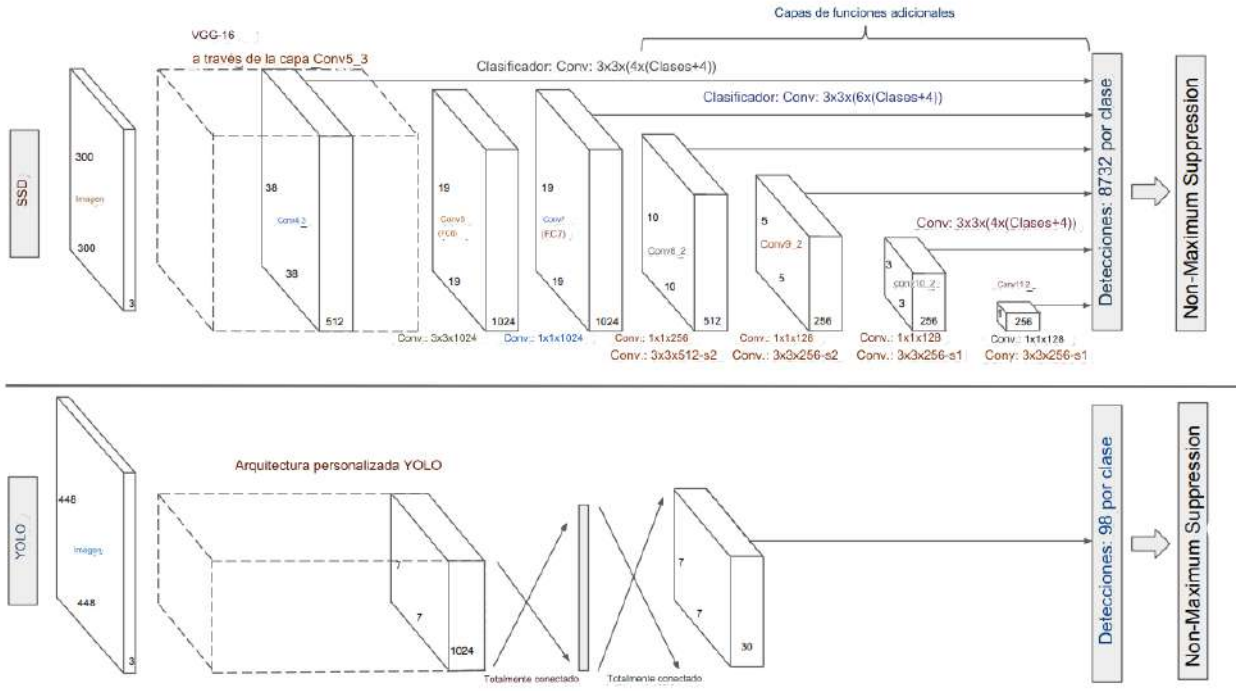


Figura 2.23: Arquitectura del modelo SSD comparada con el modelo YOLO.

(*Region of Interest*) Pooling, que ajusta las regiones propuestas a un tamaño fijo, lo que extrae características relevantes de manera eficiente. Esta técnica asegura una representación estándar de las características de las regiones propuestas, lo que facilita su procesamiento para la red convolucional y contribuye en eficiencia.

RetinaNet

RetinaNet es un modelo de detección de objetos de una sola etapa, el cual fue introducido en 2018 en (Lin y cols., 2018). Como se puede observar en la figura 2.25 tiene una arquitectura integrada que comprende un *backbone*, y dos subredes para tareas específicas.

El extractor de características empleado por RetinaNet es una *Feature Pyramid Network* (FPN), que genera un mapa de características convolucionales para la imagen de entrada. Este extractor de características es utilizado en varios modelos y tiene, como fuerte, la capacidad de detección de objetos en distintas escalas. Esto último se logra gracias la combinación de información en distintos mapas de características como se puede ver en (a) y (b) de la figura 2.25 correspondientes la arquitectura.

En cuanto a las subredes, una se encarga de la clasificación de objetos mediante convoluciones en la salida de la red principal, asignando clases a cada *anchor box* generado. La segunda subred se dedica a la regresión de cuadros delimitadores, ajustando los cuadros generados por la red principal para mejorar su precisión. Este enfoque unificado permite a RetinaNet realizar de manera efectiva la detección y clasificación de objetos en diferentes escalas y contextos en una única arquitectura.

Dentro del documento publicado en 2018, además de proponer esta arquitectura, se propuso una nueva función de costo de clasificación, esto para abordar el desafío del desequilibrio de clases en la detección de objetos (Lin y cols., 2018). Esta función de costo está más detallada en 2.6.2.

Otro punto interesante abordado en el trabajo es el aumento de *Anchor boxes*, se sugiere en el documento avanzar a detección de objetos más “densa”, y se propone utilizar 100000 *Anchor boxes* para la detección. Este número es significativo si se lo compara con anteriores modelos de la época.

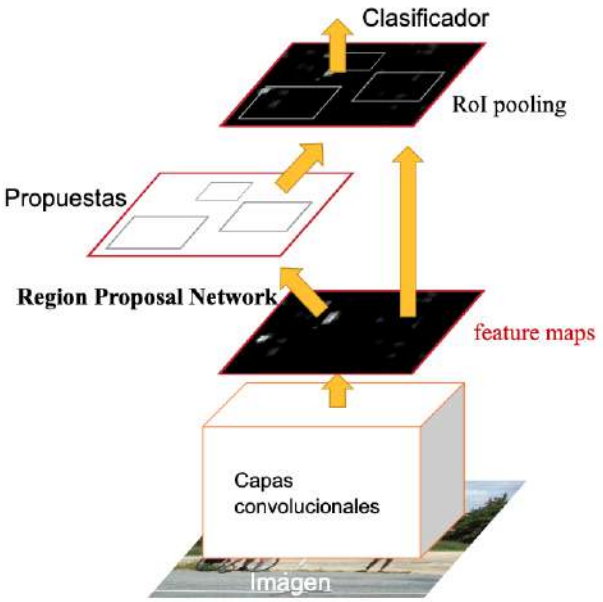


Figura 2.24: Imagen de la arquitectura general de Faster RCNN.
(Ren y cols., 2016)

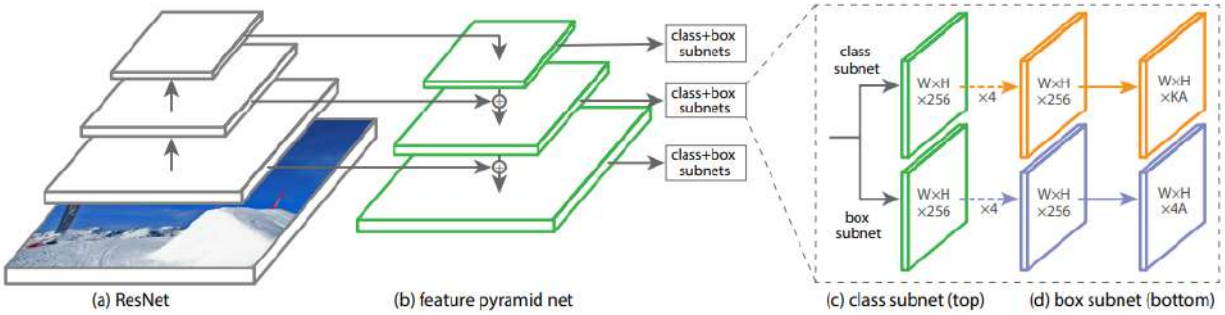


Figura 2.25: Imagen de la arquitectura general de RetinaNet.
(Lin y cols., 2018)

EfficientDet

EfficientDet fue desarrollado para abordar la creciente importancia de la eficiencia de los modelos en el área de *computer vision* (Tan, Pang, y Le, 2020). Para lograr esto, los autores del artículo estudiaron sistemáticamente las opciones de diseño de arquitectura de redes neuronales para la detección de objetos y propusieron varias optimizaciones clave para mejorar la eficiencia.

La arquitectura de EfficientDet mostrada en 2.26 sigue el paradigma de los detectores de una sola etapa y consta de tres componentes principales: el *backbone* EfficientNet, la red de características bidireccionales fusionadas (BiFPN) y dos redes más, una para predicción de cuadros delimitadores y otra para predicción de clases.

El *backbone* EfficientNet es una red neuronal convolucional pre-entrenada en ImageNet que se utiliza para extraer características de la imagen de entrada. La red BiFPN (*Bidirectional Feature Pyramid Network*) es una red que fusiona características de múltiples escalas para mejorar la precisión de la detección de objetos. Como se puede observar en la figura 2.26, la red toma características de distintas capas del *backbone* EfficientNet y aplica repetidamente la fusión de características bidireccionales de arriba hacia abajo (flechas azules) y de abajo hacia arriba (flechas rojas). Las redes de predicción de cuadros delimitadores y clases, toma las características fusionadas de la red BiFPN y producen predicciones de cuadros delimitadores y

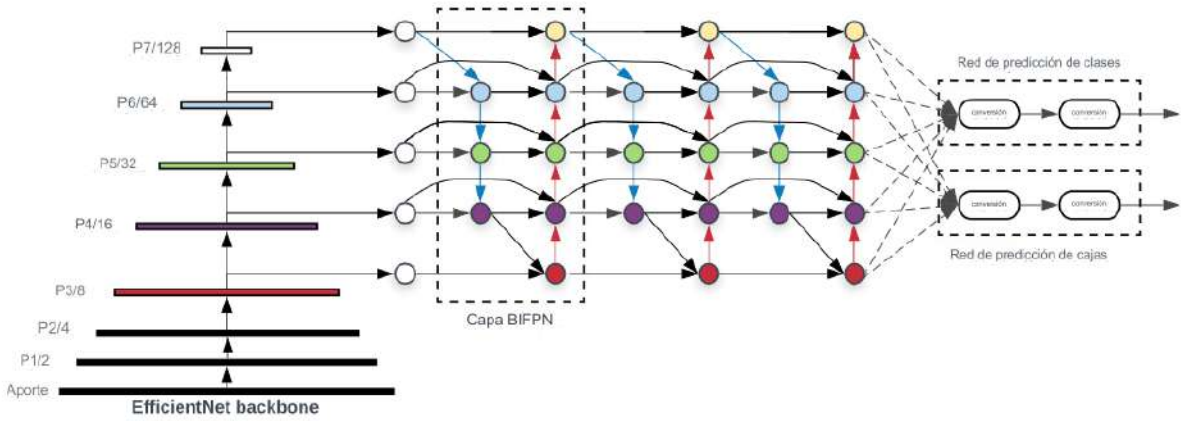


Figura 2.26: Imagen de arquitectura general de EfficentDet.
(Tan y cols., 2020)

clases para cada objeto detectado en la imagen.

DETR (DEtection TRansformer)

Detr (Carion y cols., 2020) es un novedoso modelo perteneciente a los modelos de una sola etapa y está basado en la arquitectura de redes neuronales *transformers*. Estas arquitecturas tienen un mecanismo de atención que posibilita que la red se centre en distintas partes de la entrada, con ponderaciones sobre las mismas (Vaswani y cols., 2017). Cabe destacar que la cantidad de predicciones que puede realizar, está limitada por un número fijo, esto significa que, por ejemplo, se le puede determinar encontrar cuatro objetos, y si solo encuentra tres, habrá una predicción que será del estilo *no object*.

La arquitectura de DETR 2.27 consta de tres componentes principales que trabajan en conjunto para lograr la detección de objetos de extremo a extremo de manera eficiente y precisa. Estos componentes son:

- ***Backbone***: Se utiliza una red neuronal convolucional (CNN) convencional como *backbone* para extraer una representación de características compacta de la imagen de entrada.
- ***Encoder-Decoder Transformer***: La red *Transformer*, conocida por su capacidad para modelar dependencias a larga distancia, se utiliza como un codificador-decodificador en DETR. El codificador procesa las características de la imagen generadas por el *backbone* junto con codificaciones posicionales espaciales, mientras que el decodificador genera la salida basándose en la codificación de estas características. Los *Transformers* utilizan mecanismos de atención para capturar las relaciones entre las distintas partes de la entrada y la salida.
- ***FeedForward Network* (FFN)**: Una red neuronal *feedforward* (Sazli, 2006) simple opera sobre la salida del decodificador y se encarga de realizar la predicción final de detección. Esta red toma la salida del decodificador y genera las predicciones de clase y cuadros delimitadores para los objetos detectados en la imagen.

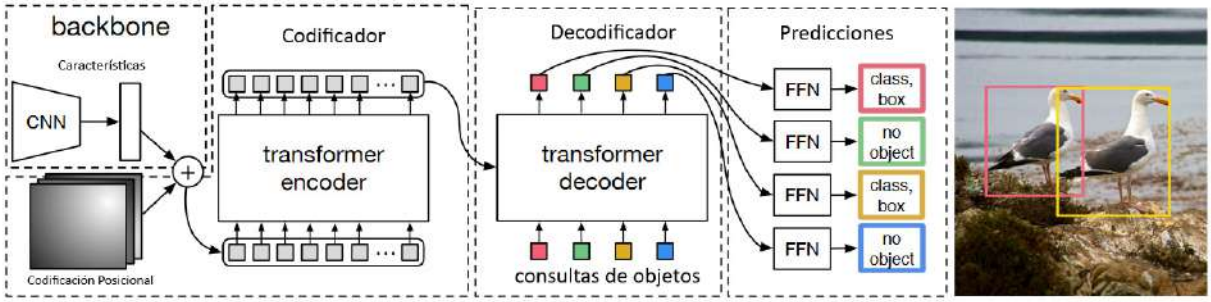


Figura 2.27: Arquitectura general de DETR.
(Carion y cols., 2020)

Capítulo 3

Revisión de antecedentes

Contenido

El objetivo de este capítulo es proporcionar al lector una visión actualizada sobre los avances en el procesamiento automático de imágenes, centrándose en la detección de daños en las palas de aerogeneradores. Con el fin de llevar a cabo una revisión sistemática de la literatura, se optó por utilizar parcialmente la metodología propuesta por (Okoli, 2015).

3.1. Identificación del propósito

El propósito de esta revisión es explorar el estado del arte de la computación visual y la detección de objetos, con el objetivo final de desarrollar un sistema de detección de daños superficiales en palas de aerogeneradores.

3.2. Protocolo de relevamiento

Para construir un protocolo a utilizar al momento de revisar la literatura se plantearon las siguientes preguntas:

- ¿Sobre qué *dataset* se trabaja?
- ¿Qué clasificaciones de daños superficiales se utilizan?
- ¿Qué criterios se utilizan para trabajar con los datos?
- ¿Qué modelos se utilizan?
- ¿Qué criterios se utilizan para cuantificar el desempeño de los métodos y cuáles son los resultados?

3.3. Búsqueda y filtrado de la literatura

Para llevar a cabo la investigación, se empleó Google Scholar (*Google Scholar*, 2024), un motor de búsqueda que alberga información académica en una amplia gama de formatos y campos de estudio. Es importante destacar que, aunque se encontraron diversos artículos, no todos estaban disponibles de manera pública. Con el fin de acceder a aquellos que tenían acceso restringido, se recurrió a FOCO (Timbó, 2024), una plataforma uruguaya que proporciona acceso gratuito a recursos científicos y tecnológicos a nivel global. Las palabras clave seleccionadas para la búsqueda fueron las siguientes: ***wind turbine, inspection, blade, object detection, damage detection, machine learning, computer vision.*** Además, se definió un intervalo de años para la inclusión de artículos, desde el 2016 hasta el 2023.

En la siguiente fase, que corresponde al proceso de filtrado, se llevó a cabo una lectura rápida de los artículos, asignándoles a cada uno un índice de relevancia:

1. **Poco relevante**: Este artículo aborda la detección de daños en palas de aerogeneradores, sin embargo, no se centra en imágenes ni en técnicas de inteligencia artificial.

2. **Relevante**: Este artículo se enfoca en la detección de objetos, aunque no específicamente en palas de aerogeneradores.

3. **Muy relevante**: Este artículo destaca por utilizar métodos de aprendizaje automático aplicados a imágenes de palas de aerogeneradores dañadas.

En esta fase, se recopiló información adicional acerca del tipo de artículo. Se distinguió entre revisiones bibliográficas, que ofrecen una síntesis y análisis de la literatura existente sobre un tema, y casos de estudio, que exploran situaciones prácticas mediante la aplicación de conceptos teóricos en entornos reales. Además, se consideró el año de publicación y el número de citas para organizar los artículos en nuevas categorías. Vale la pena destacar que se hicieron excepciones, como en el caso del artículo (He, Zhang, Ren, y Sun, 2015), el cual fue incluido en la lista a pesar de no cumplir con la restricción de la ventana temporal establecida. Esto se debe a que el artículo, reconocido por su relevancia en el procesamiento de imágenes, está excepcionalmente bien redactado y marca un hito significativo en la introducción de redes residuales.

Como complemento a este análisis, se ha generado la tabla 3.1, que presenta la clasificación de los artículos según su relevancia. La tabla proporciona una visión resumida de la importancia asignada a cada artículo, según los criterios previamente discutidos. Se priorizaron los artículos más recientes, seguidos por los que tienen más relevancia y finalmente más citas.

3.4. *Datasets*

La mayoría de los artículos examinados mencionan la cantidad de imágenes utilizadas y cómo se obtuvieron, pero no están disponibles públicamente. Sin embargo, uno de los artículos consultados (Shihavuddin y cols., 2019), proporciona un *dataset* público que contiene 701 imágenes de alta resolución que no pasaron por el proceso de anotación, recolectadas entre 2017 y 2018 en Roskilde, Dinamarca (Shihavuddin y cols., 2021). Estas imágenes fueron tomadas en alta resolución y a una distancia considerable, como se puede ver en la figura 3.1. El conjunto de datos abarca diversos tipos de daños en las palas de los aerogeneradores, tales como generadores de vórtices con dientes faltantes, erosión en el borde de ataque, grietas, receptores de rayos dañados y superficies faltantes, entre otros.

A partir de este *dataset*, otro estudio (Foster y cols., 2022) dividió las imágenes originales de tamaño 5280x2970 en imágenes más pequeñas de tamaño 586x371 y descartó las imágenes sin superficie de pala, un ejemplo se puede ver en la imagen 3.2. Este conjunto de datos consta de 13470 imágenes y está disponible en Kaggle [1].

De la totalidad de las imágenes, únicamente 2996 se encuentran con instancias de anotación, en la figura 3.3a se puede apreciar un gráfico de torta mostrando la distribución porcentual entre imágenes con y sin instancias anotadas. Cabe destacar que a partir de ahora, al referirse a imágenes sin anotación, se refiere a

[1] https://www.kaggle.com/datasets/ajifoster3/yolo-annotated-wind-turbines-586x371

Autores y fecha	Relevancia	Tipo	Citas	Fuente
(Aird, Barthelmie, y Pryor, 2023)	3	Caso de Estudio	0	MDPI
(C. Zhang, Yang, y Yang, 2022)	3	Caso de Estudio	4	MDPI
(Zou, Wang, Bi, y Sun, 2022)	3	Caso de Estudio	1	MDPI
(Song y cols., 2022)	3	Revisión	1	MDPI
(Zou y Cheng, 2022)	3	Caso de Estudio	1	MDPI
(Gu, Liu, y Li, 2022)	3	Caso de Estudio	0	MDPI
(Lv y cols., 2022)	3	Caso de Estudio	0	Science Direct
(Kabir, Aslansefat, Gope, Campean, y Papadopoulos, 2022)	3	Caso de Estudio	0	Cornell University
(Iyer y cols., 2022)	2	Caso de Estudio	0	Cornell University
(Foster y cols., 2022)	1	Caso de Estudio	187	IEEE
(Zhu, Liu, Li, Wan, y Cai, 2022)	1	Caso de Estudio	0	MDPI
(Nguyen, Iyer, y Khushu, 2022)	3	Caso de Estudio	0	Cornell University
(Guo, Liu, Cao, y Jiang, 2021)	3	Caso de Estudio	26	Science Direct
(Shihavuddin y cols., 2021)	3	Caso de Estudio	21	Science Direct
(J. Zhang, Cosma, y Watkins, 2021)	3	Caso de Estudio	19	MDPI
(Sarkar y Gunturi, 2021)	3	Caso de Estudio	10	Springer
Patel, Sharma, y Dhiman, 2021	3	Caso de Estudio	2	Cornell University
(Pizarro Muñoz, 2020)	3	Caso de Estudio	1	Univ. politécnica de valencia
(Bahaghighat, Xin, Motamedi, Zanjireh, y Vacavant, 2020)	3	Caso de Estudio	8	MDPI
(Yang, Dong, Zhao, y Chen, 2020)	3	Caso de Estudio	8	IEEE
(Tan y cols., 2020)	2	Caso de Estudio	3311	Cornell University
(Du y cols., 2020)	1	Revisión	176	Science Direct
(Shihavuddin y cols., 2019)	3	Caso de Estudio	71	MDPI
(Reddy, Indragandhi, Ravi, y Subramaniyaswamy, 2019)	3	Caso de Estudio	88	Science Direct
(Peng y Liu, 2018)	3	Caso de Estudio	22	Wiley
(Moreno, Peña, Toledo, Treviño, y Ponce, 2018)	3	Caso de Estudio	5	IEEE
(Crous, Intelligentie, y Visser, 2018)	3	Caso de Estudio	1	Computer Science
(Wang, Zhang, Xu, y Liu, 2018)	1	Caso de Estudio	2	IEEE
(Wang y Zhang, 2017)	3	Caso de Estudio	152	IEEE
(He, Gkioxari, Dollár, y Girshick, 2017)	2	Caso de Estudio	11450	IEEE
(Lin y cols., 2017)	2	Caso de Estudio	8894	IEEE
(Yu, Cao, Liu, Yang, y Bai, 2017)	1	Caso de Estudio	539	IEEE
(He, Zhang, Ren, y Sun, 2016)	2	Caso de Estudio	3	IEEE
(He y cols., 2015)	2	Caso de Estudio	161381	Cornell University

Tabla 3.1: Clasificación de artículos por relevancia.

imágenes que pasaron por el proceso de anotación, pero no se encontraron instancias de objetos, es decir, imágenes de palas en buenas condiciones. A partir de este punto se hará referencia a este último *dataset* por el nombre de "Nordtank".

A su vez, este *dataset* posee únicamente dos clases, Daño y Suciedad. Es importante destacar que estas clases se encuentran desequilibradas como se puede ver en la figura 3.3b, teniendo una mayor cantidad de anotaciones de Daño que de Suciedad.

Además, otro artículo consultado (Zou y cols., 2022) proporciona un conjunto de datos con un total de 56 imágenes que no pasaron por el proceso de anotación. Estas imágenes también se centran en las palas de los aerogeneradores y se pueden encontrar en GitHub (KaKoYu007, 2022), en la figura 3.4 se tiene un ejemplo.

Otra fuente importante corresponde a Mendeley, en donde se brinda un amplio conjunto de datos (Nikolov, Nielsen, Garnæs, y Madsen, 2020). No obstante, es importante destacar que todas las imágenes incluidas muestran palas que fueron retiradas del molino, y algunas presentan anotaciones físicas escritas a mano al lado de los daños. Esta problemática se puede apreciar en la imagen 3.5. Dada la diferencia de contexto en la que se sacan las imágenes, y la presencia de anotaciones físicas, hace que no sean útiles.

En una etapa avanzada del proyecto, fue proporcionado un conjunto de datos con imágenes representativas de la realidad local, el cual será referenciado de ahora en más como DS.UY. El conjunto de datos original consistía en dos conjuntos de imágenes anotadas proporcionadas, los cuales fueron unificados en uno solo para una mejor gestión y análisis. Las clases de daños del conjunto son grieta (*Crack*, CR), receptor de rayos (*Lightning Receptor*, LR), erosión del borde de ataque (*Leading Edge Erosion*, LEE), impacto de rayo

Figura 3.1: Imagen de tamaño original tomada en Roskilde, Dinamarca.
(Shihavuddin y cols., 2019)

Figura 3.2: Imagen recortada tomada en Roskilde, Dinamarca.
(Foster y cols., 2022)

(*Lightning Strike*, LS) y desprendimiento de la cinta del borde de ataque (*Tape Erosion*, TE). Un ejemplo de estos daños se puede observar en la tabla 3.2.

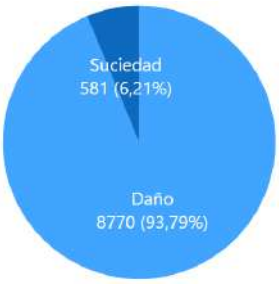
Cabe destacar que las anotaciones de ambos conjuntos se encontraban en formatos variados, algunas anotaciones con cuadros delimitadores y otras con polígonos o segmentos. Al unificarse los mismos se optó por normalizar todas las anotaciones, llevándolas a cuadros delimitadores.

3.5. Clasificación de daños

Como se puede observar en la tabla 3.3, los artículos revisados cubren una amplia variedad de daños. No obstante, la categoría de daño más frecuente es la de Grieta, como se evidencia en trabajos como (C. Zhang y cols., 2022), (Zou y Cheng, 2022), (Lv y cols., 2022), (Iyer y cols., 2022), (Shihavuddin y cols., 2021), (J. Zhang y cols., 2021), (Sarkar y Gunturi, 2021), (Patel y cols., 2021), (Pizarro Muñoz, 2020) y (Guo y cols., 2021). Además, de entre todos los artículos relevados, se estudió la frecuencia con la que un daño aparece en los mismos. En la gráfica de barras 3.6 se puede apreciar aquellos que aparecen en al menos dos artículos.

Del artículo (Foster y cols., 2022) se pueden apreciar tres de estos daños: Grieta, Receptor de rayo dañado (LR), Receptor de rayo (LR), VG Dañado y VG sano. Estos daños se pueden visualizar en la figura 3.7.

(a) Distribución porcentual entre imágenes con y sin instancias de anotaciones.



(b) Distribución porcentual de clases.

Figura 3.3: Distribución porcentual de imágenes y clases del *dataset* de Nordtank.

Figura 3.4: Ejemplo de imagen del repositorio de GitHub.
(KaKoYu007, 2022)

Un ejemplo de *Leading Edge Erosion* (LEE) y hueco se puede encontrar en el artículo (Aird y cols., 2023). Este artículo toma un enfoque particular y va más allá de la simple detección de clases de daños por tipo, ya que busca diagnosticar la gravedad del daño. Este enfoque categoriza los tipos de daño en dos clases distintas: los superficiales, como picaduras y rascaduras, y los profundos, como oquedades y delaminación, asignando clases de objetos distintas a un mismo tipo de daño. Esto permite una evaluación más precisa de la magnitud del daño. Un ejemplo de esto puede verse en la figura 3.8.

Por último, encontramos la categoría conocida como *Tape Erosion* (TE), la cual se presenta en la tabla 3.2. Este tipo de deterioro se refiere al desgaste progresivo de la cinta que protege el borde de ataque de las palas, resultado de la interacción de diversos factores ambientales.

Figura 3.5: Ejemplo de imagen con anotación física.
(Nikolov y cols., 2020)

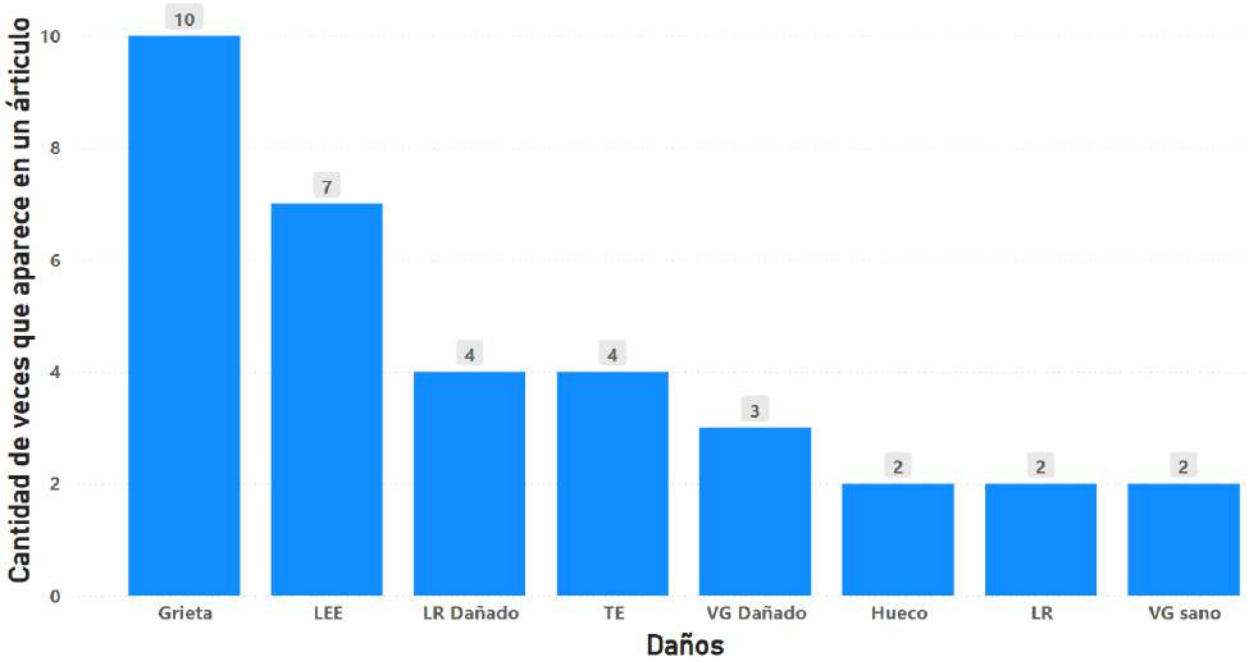


Figura 3.6: Daños con mayor frecuencia de aparición en artículos.

Clasificación	Daño	Descripción
CR		Grieta normalmente provocada por desgaste de la pala.
LEE		Erosión en el borde de ataque, normalmente provocado por el impacto del viento.
LR		Daño en el receptor de rayos, provocado por el impacto de un rayo.
LS		Daños en forma de grieta provocados por impactos de rayos.
TE		Desprendimiento de la cinta que cubre el borde de ataque, provocado por el impacto del viento.

Tabla 3.2: Descripción de los tipos de daño.

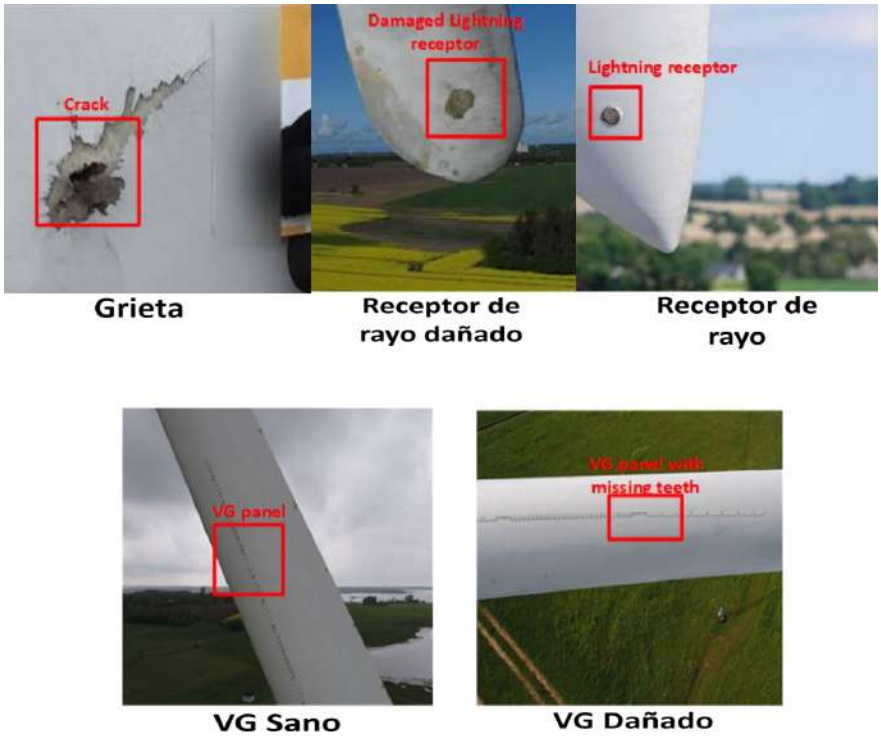


Figura 3.7: Ejemplos de daño de grieta, LR, LR Dañado, VG Dañado y VG sano. (Foster y cols., 2022)

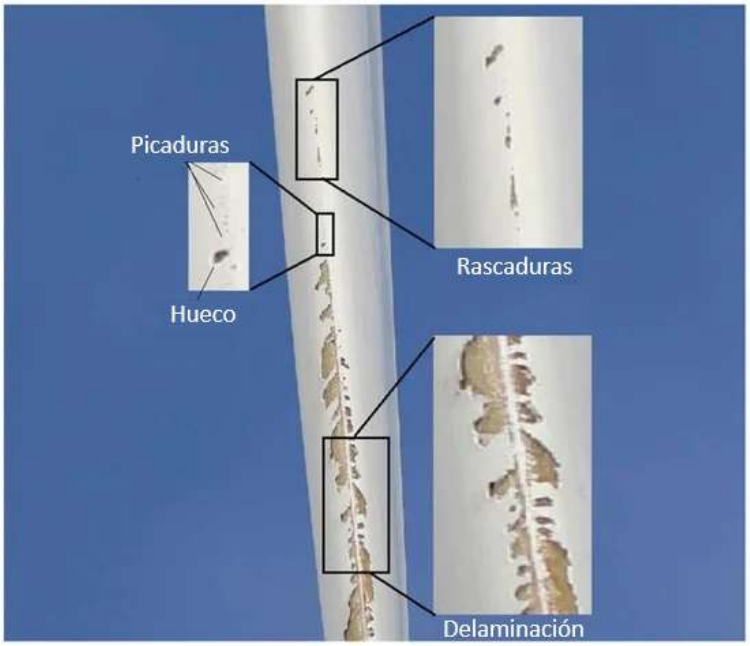


Figura 3.8: Ejemplo de LEE con distinta gravedad. (Aird y cols., 2023)

3.6. Preprocesamiento de los datos

El objetivo final del preprocesamiento de datos es proporcionar un conjunto de datos limpio, bien estructurado y adaptado para maximizar la eficiencia y eficacia de los modelos a entrenar. Las técnicas observadas se pueden agrupar en tres clasificaciones según su propósito, aquellas destinadas a limpiar el *dataset* (como por ejemplo la eliminación de imágenes sin palas (Foster y cols., 2022)), aquellas destinadas a mejorar la eficiencia (por ejemplo realizar una reducción de dimensionalidad (Nguyen y cols., 2022)) y por último aquellas destinadas a aumentar y diversificar el conjunto de entrenamiento. De este último grupo se identificaron diversos ajustes, los cuales están vinculados a simular cambios en las condiciones en las que se capturaron las fotografías, como se puede observar en la columna "Técnicas de preprocesamiento" de la tabla 3.3. Estos ajustes abarcan modificaciones en las condiciones cromáticas y de iluminación, como el contraste, la saturación y el brillo. Asimismo, se observaron variaciones relacionadas con el ángulo de captura, como giros de la imagen, cambios en la escala y ajustes en la perspectiva. Otras transformaciones no tan comunes tenían la introducción de ruido en sus imágenes (Zou y Cheng, 2022) y (C. Zhang y cols., 2022). En la figura 3.9 se pueden visualizar ejemplos de aumentaciones.

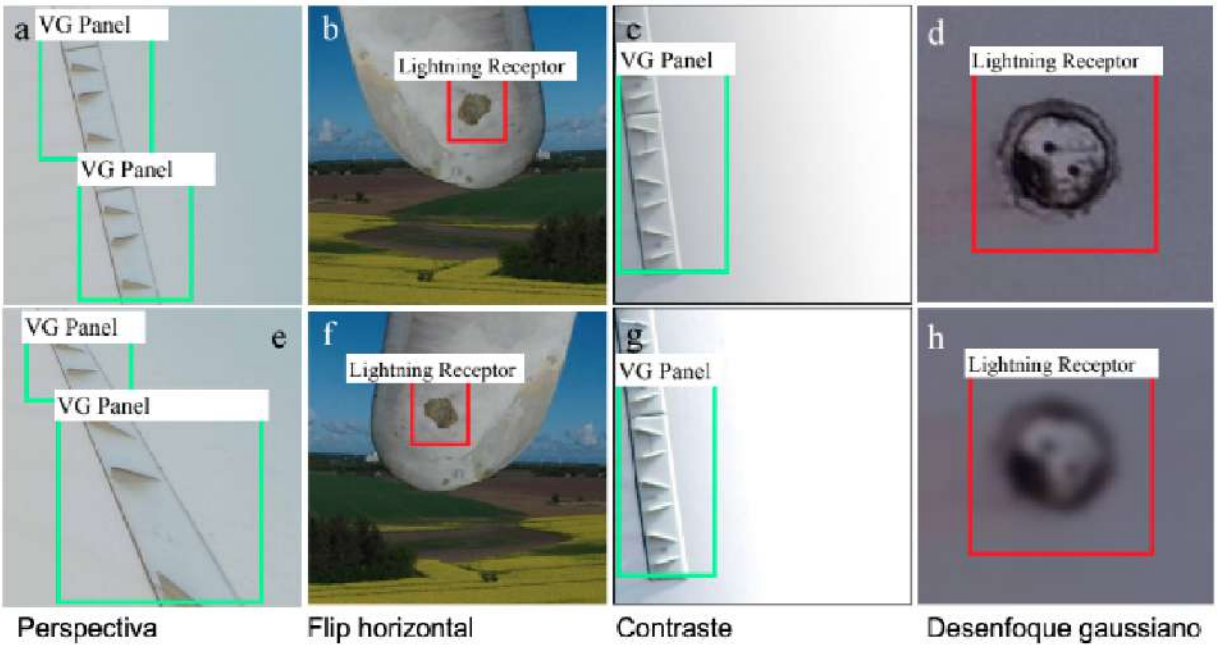


Figura 3.9: Ejemplo de aumentaciones vistas: perspectiva, *flip* horizontal, contraste y desenfoque gaussiano. (Shihavuddin y cols., 2019)

3.7. Modelos

Como se evidencia en la tabla 3.3, la mayoría de los estudios revisados para el reconocimiento de objetos prefieren utilizar redes convolucionales. Esta preferencia se respalda por su eficacia y versatilidad en la detección de patrones visuales complejos, particularmente adecuados para el ámbito del reconocimiento de objetos. Dentro de la categoría de redes convolucionales, se observa que los modelos más recurrentes son YOLO y los basados en arquitecturas R-CNN (de dos etapas), siendo Resnet un *backbone* muy común.

Para el caso del *dataset* de COCO, los modelos con *backbones* más profundos suelen mejorar los resultados. Por ejemplo, YOLOv8n tiene un $mAP_{0.5:0.95}$ de 37.3, YOLOv8s de 44.9 y YOLOv8m de 50.2 para este *dataset* (Ultralytics, 2023). Se puede notar mejoras de más de 0.5 en el mAP. No obstante, no se observa una mejora significativa en el rendimiento al emplear *backbones* más profundos para la detección de daños, si bien esto puede variar dependiendo del contexto y del documento específico. Por ejemplo, el documento (Foster y

cols., 2022), llega a los mejores resultados con el modelo de YOLOv5s, el menos profundo. Esto puede ser atribuido a la poca cantidad de imágenes y la necesidad de largos volúmenes de datos, sobre todo para arquitecturas más profundas. Por otro lado, en (Pizarro Muñoz, 2020) la mejor *Accurracy* obtenida fue de 0.773 con resnet50, mientras que resnet34 obtuvo 0.764. Estas diferencias son ínfimas si se comparan los cambios que se pueden observar en *datasets* como el de COCO. Esto sugiere que los cambios a arquitecturas más profundas no suelen relacionarse directamente con la mejora de las métricas y que los modelos menos profundos suelen ser los más adecuados para este problema.

3.8. Métricas y resultados

Como se puede observar en la tabla 3.3 las métricas más frecuentemente empleadas en la evaluación de los modelos son mAP, *Recall, Precision* y *Accuracy*. Por otro lado, algunos estudios se centran exclusivamente en la clasificación sin considerar la localización de los objetos, como se observa en (Iyer y cols., 2022), (Zou y Cheng, 2022), (Patel y cols., 2021) y (Yang y cols., 2020). Esto reduce las métricas a solo las de clasificación, lo cual puede dificultar la comparación de los resultados con estudios que sí incluyen la localización. Los modelos de clasificación, al no tener que lidiar con la localización, tienden a tener una precisión más alta.

En general, se observa una considerable variación en los valores alcanzados para estas métricas entre los distintos artículos. Esto puede deberse a distintos motivos, como la selección del umbral de confianza, el umbral de IoU, o incluso la calidad del conjunto de datos empleado, entre otros factores. Por ello, se realizará un análisis detallado del rendimiento obtenido en dos de los artículos más relevantes, seleccionados como referencia. Se consideran relevantes dado que trabajan sobre la tarea de detección de objetos, donde documentan diferentes métricas e hiperparámetros. Utilizan varias clasificaciones de daños y entrenan múltiples modelos, lo cual hace al estudio más completo.

En el artículo (Shihavuddin y cols., 2019), se exploran los modelos Inception-V2, ResNet-50, ResNet-101 e Inception-ResNet-V2 como *backbones* de la arquitectura Faster-RCNN, utilizando la métrica mAP para evaluar su desempeño. Se establece un IoU de 0.3 como umbral para considerar una detección como verdadero positivo. El valor más alto registrado fue del 0.811, logrado con Inception-ResNet-V2 al implementar diversas técnicas de aumentación de los datos.

Por otro lado, el estudio en (Foster y cols., 2022) se centra en los modelos Faster R-CNN, YOLOv5s (*small*), YOLOv5m (*medium*) y YOLOv5l (*large*), evaluando su rendimiento a través de las métricas mAP (con IoU de 0.5 y de 0.5 a 0.95), *precision* y *recall*. Entre los modelos evaluados, YOLOv5s demostró tener los valores más altos registrados, estos fueron 0.51 ($mAP_{0.5:0.95}$), 0.79 ($mAP_{0.5}$) y 0.82 (*precision*). Por otro lado, el *recall* más alto fue de 0.82, logrado por YOLOv5l. Es relevante señalar que en este estudio se trabajó sobre el *dataset* de Nordtank. A su vez, se implementaron técnicas de aumentación de los datos durante el entrenamiento de los modelos y se detallan los parámetros utilizados para las transformaciones.

3.9. Conclusiones

En primer lugar, la privacidad de los datasets es una cuestión recurrente. La falta de disponibilidad de datos y la falta de transparencia en cuanto a sus estadísticas dificultan su replicabilidad, así como la comparación entre distintos estudios. Además, la variedad de métricas y parámetros utilizados, como el umbral de IoU para el cálculo de mAP, puede variar significativamente entre investigaciones, lo que complica aún más la comparación de resultados.

Algunos estudios se enfocan solo en clasificar objetos sin localizarlos, lo que reduce las métricas a tan solo las de clasificación, dificultando la comparación de sus resultados con estudios que sí incluyen la localización. A su vez, la falta de documentación sobre las implementaciones también es un problema común. Las implementaciones tienden a ser privadas y no se proporcionan muchos detalles sobre las configuraciones de los modelos utilizados.

Dada la escasez de datos en muchos casos, la mayoría de los estudios utilizan técnicas de aumentación. Las técnicas utilizadas son variadas, pero se nota una tendencia a realizar transformaciones geométricas. Aun así, generalmente no se detalla en profundidad los parámetros utilizados individualmente para cada transformación. Por ejemplo, se indica que se realizan cambios de contraste o saturación, pero no en qué grado, se mencionan cambios de escala pero no en qué proporción, entre otros.

Para el conjunto de datos COCO, los modelos con *backbones* más profundos tienden a dar mejores resultados. Sin embargo, para la detección de daños, modelos menos profundos como YOLOv5s pueden ser más efectivos debido a la falta de datos suficientes. Comparaciones entre diferentes modelos muestran diferencias mínimas en la precisión, sugiriendo que la profundidad del modelo no siempre se traduce en mejoras significativas en este contexto específico.

En resumen, el estado actual de la detección de objetos en el contexto de los daños en palas de aerogeneradores presenta desafíos significativos en términos de privacidad de datos, variedad de métricas y transparencia en las implementaciones. Es esencial adoptar un enfoque más claro y uniforme, que garantice transparencia y la posibilidad de replicar los resultados obtenidos. Aun así, este problema está bajo constante investigación a nivel global, sobre todo en el territorio chino, de donde provienen la mayoría de los estudios. Los documentos relevados datan desde 2018 hasta la actualidad, con un promedio en el año 2021, indicando lo reciente que es esta área de investigación.

Artículo	Técnicas de Pre-Procesamiento	Modelos	Clases	Resultados
(Aird y cols., 2023)	Ajuste de contraste, saturación y corrección de campo plano	*Pixel Intensity Thresholding and Shadow* (PTS) y Mask R-CNN	4 clases de LEE: superficiales (picaduras y rascaduras) y profundos (hueco y delaminación)	Pixel Acc = 65
(C. Zhang y cols., 2022)	Agregar ruido, voltear, escalar, cambiar el contraste y el brillo para simular diferentes ángulos de disparo y condiciones de luz	MobileNetv1-YOLOv4, SE-Net, ECANet y CBAM	Grieta, Picadura, Contaminación, Resquebrajamiento	MobileNetv1-YOLOv4 mAP= 93.7
(Zou y Cheng, 2022)	Ruido aleatoreo, voltear, rotaciones, normalizacion 56x56	Net(inventado), ShuffleNet-V2	Grieta, Desprendimiento	Net (Acc=99.23 y f1=99.38) ShuffleNet-V2 (Acc=99.48 y f1=98.86)
(Lv y cols., 2022)	Volteos al azar, rotaciones y cambios al azar de brillo, contraste y saturación	SSD, EADD(inventado), YOLOv4, DETR	Desprendimiento, Grieta, Corrosión	SSD(mAP=75) EADD(mAP=81.2) YOLOv4(mAP=78.2) DETR(mAP=81.2)
(Iyer y cols., 2022)	Cambios geométricos y de iluminación	ResNet50	Grieta	Recall=96 Precision=85 F1-Score=90
(Foster y cols., 2022)	Cambios en HSV, traslación, escalado, volteos y la técnica de mosaico	Faster R-CNN (con Resnet-101), YOLOv5 Small(S), YOLOv5 Medium(M) y YOLOv5 Large(L)	Suciedad, Daño	Faster R-CNN, $mAP_{0.5}$: 0.7539 Recall: 0.73123 YOLOv5s, $mAP_{0.5}$: 0.7937 Recall: 0.79045 YOLOv5m, $mAP_{0.5}$: 0.7837 Recall: 0.79152 YOLOv5L, $mAP_{0.5}$: 0.7836 Recall: 0.82118
(Nguyen y cols., 2022)	Reducción de definición en imágenes	Mask R-CNN	Daño	Precisión = 40.48 Recall = 94.4
(Guo y cols., 2021)	No especifica	VGG16	Defecto de Fibra, Grieta, Defecto de Recubrimiento, LEE	F1-score=67
(Shihavuddin y cols., 2021)	Cambio en brillo, contraste, matiz, saturación, volteo horizontal y vertical	YOLOv5 y EfficientDet	LEE, VG Dañado, Grieta, Escombro, Derrame de Aceite, LR Dañado	mAP=86.6
(J. Zhang y cols., 2021)	Volteo vertical y horizontal, tonos de gris y rotación de 90, 180 y 270 grados	YOLOv3, YOLOv4 y Mask R-CNN	Grieta, Erosión, Hueco, 'Otro'	IE Mask R-CNN: $mAP_{0.5}$=84.21
(Sarkar y Gunturi, 2021)	No especifica	YOLOv3	VG sano, VG Dañado, Corrosión en LE, Grieta, LR, LR Dañado	Precision=0.96
Patel y cols., 2021	Cambio de ancho y alto, rotaciones de 45 grados, volteos horizontales, acercamientos	CNN, VGG16-RCNN y AlexNet	Área de Daño, Área de Erosión, Área de Borde, Área de Referencia	CNN (Acc=90.7, FN=14.08, FP=14) VGG16-RCNN (Acc=93.8, FN=5.2, FP=90.9)
(Pizarro Muñoz, 2020)	Acercamientos y alejamientos de la imagen y ruido gaussiano	VGG, ResNet y SSD	Burbuja, Fundido, Pelaje Desconchado, Defecto del Pelaje, Grieta alrededor del LR, Grieta alrededor del tornillo SPL, Erosión, Laminado Dañado, Daños por Rayos, LR Dañado, Ruido, Erosión de Pintura, Marcas de Frotamiento	VGG16(mAP=[13.3,32.4]) Resnet34(mAP=69.6, AP=0.764) Resnet50(Acc=0.758, AP=0.773)
(Yang y cols., 2020)	Recortes básicos	AlexNet y ResNet50	Grieta, Agujero, Mezclado, Fondo	AlexNet: Acc=94.19 Resnet50: acc=95.58
(Shihavuddin y cols., 2019)	Disminución de resolución, extracción de parches, transformación de perspectiva, volteo izquierda a derecha, normalización de contraste y ruido Gaussiano	FasterRCNN con distintos extractores de características (Inception-ResNet-V2, Inception-V2, ResNet-101 y ResNet-50)	LEE, VG Sano, VG Dañado, LR	Inception-Resnet-V2: mAP=81.1
(Crous y cols., 2018)	Aumentación por flipeo horizontal, vertical y diagonalmente	Resnet18 classifier y R-CNN	Daño	AP=82

Tabla 3.3: Resumen de la información relevante contenida en los artículos.

Capítulo 4

Experimentación

Contenido

Al inicio de las experimentaciones, solo se contaba con uno de los dos conjuntos de datos utilizados, específicamente el denominado Nordtank, comentado en la sección 3.4. Por consiguiente, las pruebas iniciales de la sección 4.3, donde se comparó el rendimiento de los seis modelos discutidos en la sección 2.8, se realizaron exclusivamente con este conjunto de datos. Posteriormente, en la sección 4.4, se evaluaron los dos mejores modelos seleccionados en la sección anterior utilizando tres conjuntos de datos adicionales creados mediante diferentes técnicas de aumento a partir del conjunto Nordtank.

Más adelante, se accedió a un nuevo conjunto de datos conteniendo imágenes representativas de parques eólicos locales, el cual se comenzó a incorporar en las experimentaciones. Inicialmente, se decidió combinar los conjuntos de datos disponibles con el objetivo de aumentar la variabilidad en el conjunto de entrenamiento, lo que debería permitir una mejor generalización del modelo. Para lograr esto, se redujo el conjunto de instancias de Nordtank exclusivamente a las anotaciones de daños, eliminando las anotaciones de suciedad.

De esta forma, se trataron a todas las clases presentes en el conjunto de DS.UY como a una única clase de daño. Sin embargo, pronto se evidenció que esta estrategia no era adecuada. Los daños en ambos conjuntos de imágenes eran considerablemente diferentes entre sí, lo que provocaba que el modelo no pudiera aprender correctamente las características específicas de cada tipo de daño al agruparlos en una sola clase. También se experimentó utilizando los pesos preentrenados con Nordtank, pero el desempeño fue peor que utilizando los pesos preentrenados con conjuntos de datos como Imagenet y COCO. Lo cual afirma la diferencia en

la distribución de los daños de ambos conjuntos comentada anteriormente. Una característica distintiva de los daños de Nordtank es que los mismos presentan un fondo de pintura roja, que indica que la pala tuvo una erosión. Sin embargo, esto no sucede con los daños de DS.UY, haciendo que los patrones de los daños a encontrar sean totalmente distintos.

Además, como se evidenciará más adelante, se detectaron deficiencias en las anotaciones del conjunto de datos de DS.UY, lo que también afectaba la calidad del entrenamiento y las evaluaciones. Debido a estos problemas, se decidió no documentar estas experimentaciones.

Finalmente, en la sección 4.5, se realizaron experimentaciones utilizando el conjunto de datos de DS.UY con sus cinco clases de daños originales. Se mejoraron las anotaciones y se evaluó el rendimiento de cada clase. Además, se exploraron técnicas con el objetivo de mejorar los resultados, como la adición de imágenes sintéticas y la aplicación de regularización mediante aumento de datos y *dropout*. También se probaron diferentes optimizadores, *backbones* y tamaños de *batch size* para optimizar el desempeño del modelo.

4.1. Plataforma de ejecución

Cluster UY (ClusterUY, 2024) es la plataforma de ejecución utilizada para la experimentación. Es producto de una iniciativa académica para proporcionar poder de cómputo en el ámbito nacional. De entre el hardware que se tiene disponible se le dio un fuerte uso a las unidades de procesamiento gráfico general. Se hizo un uso más frecuente de las unidades de procesamiento gráfico Nvidia Tesla P100-PCIE con 12GB de VRAM por sobre las GPU Nvidia Ampere A100-PCIE con 40GB de VRAM, ya que estas últimas constituyen tan solo 2 de los 29 nodos disponibles para ejecución.

El lenguaje utilizado es Python en su versión 3.10.12. Además, para la mayoría de los entrenamientos, se emplea un *framework* que, por lo general, utiliza una librería como PyTorch. Estos *frameworks* suelen tener una interfaz con C++ para mejorar la eficiencia en el manejo de datos y operaciones. En este caso, la versión del compilador GCC usado es 4.8.5 20150623 (Red Hat 4.8.5-28). Ejecutando el comando `nvidia-smi` se puede ver información sobre el estado y el rendimiento de las GPU NVIDIA en el sistema. Se adjunta el resultado del comando en la figura 4.1.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 530.30.02              Driver Version: 530.30.02    CUDA Version: 12.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                  Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf            Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=================================+======================+======================|
|   0  NVIDIA A100-PCIE-40GB           Off| 00000000:37:00.0 Off |                    0 |
| N/A   35C    P0              39W / 250W|      3MiB / 40960MiB |      0%      Default |
|                                         |                      |             Disabled |
+-------------------------------+----------------------+----------------------+
```

Figura 4.1: Información desplegada al ejecutar el comando.

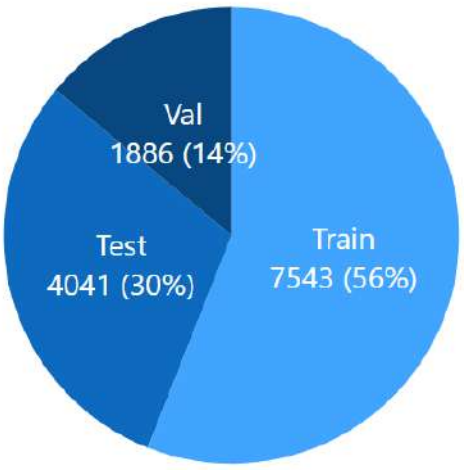
4.2. *Datasets* utilizados

Inicialmente, se optó por utilizar el conjunto de datos de Nordtank para la fase I (sección 4.3) y fase II (sección 4.4). Dado que cuenta con anotaciones que, aunque no son perfectas, están relativamente bien definidas. Además, este conjunto de datos es lo suficientemente extenso como para permitir un entrenamiento efectivo de los modelos.

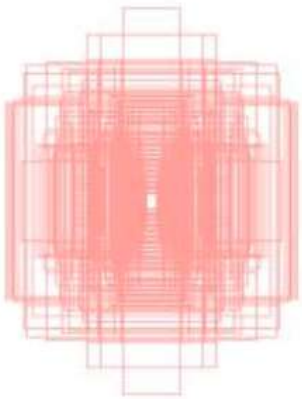
Como se mencionó anteriormente, durante la elaboración del proyecto de grado, se proporcionó un nuevo conjunto de datos DS.UY que incluye imágenes representativas de aerogeneradores locales. A partir de ese momento, se tomó la decisión de entrenar el modelo en la fase III (sección 4.5) y IV (sección 4.6) con este conjunto de datos recién adquirido. Esta elección se fundamenta en la previsión de que en el futuro se utilizará el modelo entrenado para detectar posibles daños en estos aerogeneradores específicos de DS.UY.

4.2.1. Nordtank

El *dataset* de Nordtank se separó en *train*, *val* y *test*, obteniendo la distribución que se muestra en la figura 4.2a. Las proporciones para los conjuntos fue la siguiente: *train* y *val* corresponde al 70 % del total de las imágenes, mientras que test solo el 30 %. Luego, entre el total de *train* y *val*, 80 % corresponde a *train* mientras que el 20 % a *val*. Esto deja una distribución neta de 56 % para *train*, 14 % para *val* y un 30 % para *test*. Por otro lado, a modo de visualización, en la figura 4.2b se pueden observar los distintos cuadros delimitadores de las instancias de daño presentes en el conjunto de datos.



(a) Distribución porcentual de las imágenes de Nordtank en cada conjunto.



(b) Instancias de *bounding boxes* de *Nordtank*.

Figura 4.2: Distribución y *bounding boxes* de Nordtank.

A partir de la distribución de imágenes en los conjuntos *train, val y test*, se puede observar en detalle la distribución de clases en la figura 4.3. Esta representación gráfica proporciona una visión clara de cómo se distribuyen las clases en los diferentes conjuntos de datos. La misma muestra cómo se distribuyen las clases en cada conjunto de datos de manera equitativa, garantizando buena representación de cada clase para los subconjuntos.

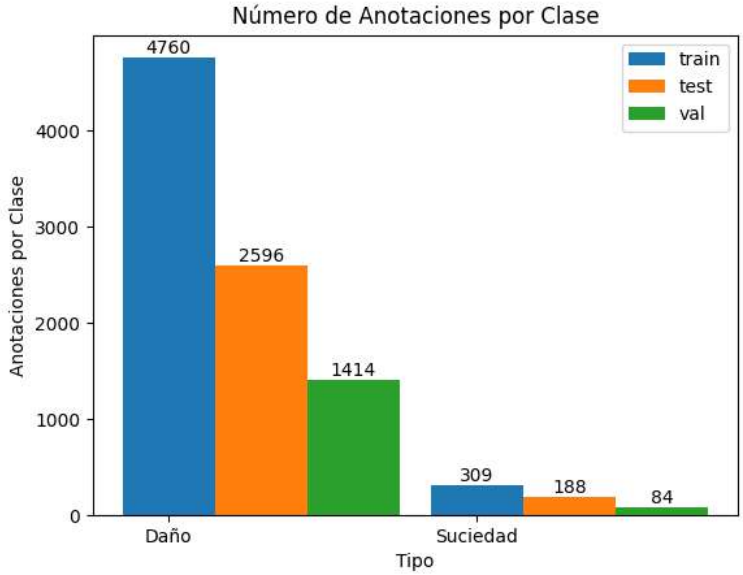
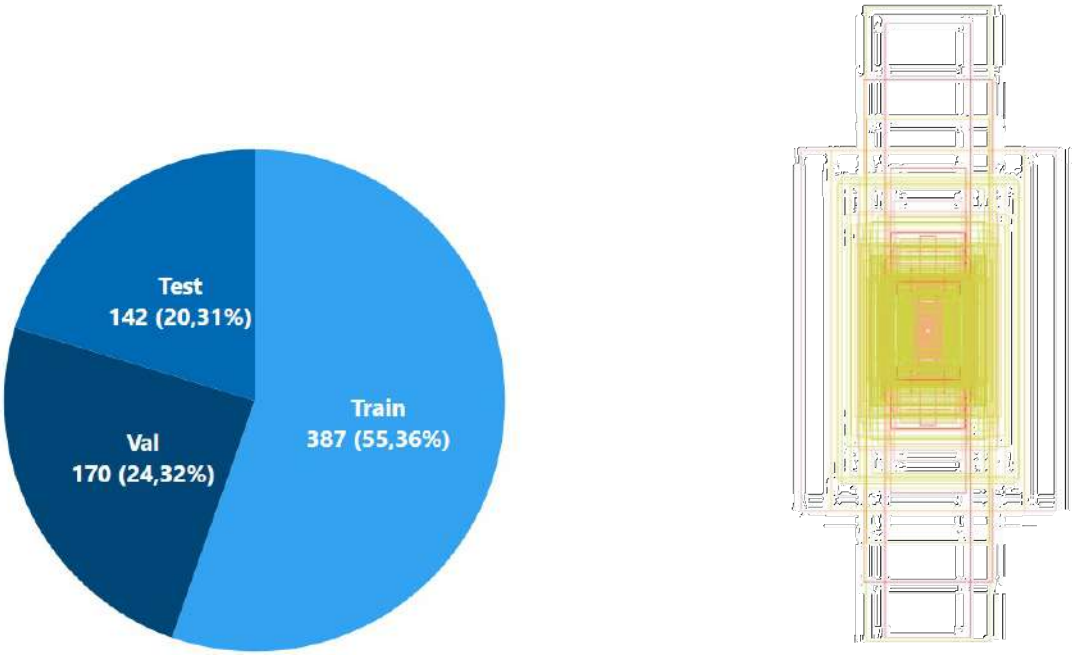


Figura 4.3: Distribución de anotaciones por clase en cada conjunto.

4.2.2. DS.UY

Para este *dataset*, como se detallará más adelante en la sección 4.5, fue esencial realizar una mejora en las anotaciones, ya que la calidad de los datos influye directamente en la efectividad de los modelos entrenados. A su vez, se eligió una nueva distribución para separar *train*, *val* y *test*, diferente a la utilizada en el conjunto de Nordtank. Esta decisión de diseño se tomó por diversas razones. Para empezar el tamaño de los dos *datasets* difieren en demasiadas imágenes: Nordtank consta de 13000 imágenes aproximadamente, mientras que DS.UY únicamente de 699. A su vez, Nordtank presenta solo dos clases, mientras que DS.UY cuenta con cinco clases distintas. Esta variabilidad en la cantidad de clases y tamaño de los conjuntos de datos plantea la necesidad de reconsiderar la distribución de los datos utilizada previamente. Inicialmente, se asignaba 56 % a *train*, el 14 % *val* y el 30 % a *test*. Se espera que esta distribución sea válida para Nordtank, ya que con 13000 imágenes, el conjunto de *val* aún contenía un número suficiente de muestras para cada clase. Sin embargo, esta misma distribución resultó insatisfactoria al aplicarla a DS.UY, donde no se garantizaba una representación adecuada de todas las clases en el conjunto de *val*. Debido a esto, se optó por elegir una distribución que mantuviese la proporción de *train*, pero que buscara equiparar más *val* y *test*. Para ello se eligió la siguiente distribución: *train* y *val* corresponde al 80 % del total de las imágenes, mientras que test solo el 20 %. Esto deja una distribución neta de 56 % para *train*, 24 % para *val* y un 20 % para *test*, manteniendo el porcentaje del conjunto *train*. La distribución porcentual puede ser vista en 4.4a mientras que en la figura 4.5 se puede visualizar la distribución de clases para cada uno de los conjuntos con esta nueva distribución. Como era de esperar, el muestreo de las clases para cada conjunto es aceptable. Por otro lado, la figura 4.4b muestra los cuadros delimitadores de las instancias de daño para el dataset DS.UY, el mismo en comparación con Nordtank tiene una distribución menos uniforme, los cuadros son bastante más alargados en el eje vertical.



(a) Distribución porcentual de las imágenes de DS.UY en cada conjunto.

(b) Instancias de *bounding boxes* de DS.UY.

Figura 4.4: Distribución y *bounding boxes* de DS.UY.

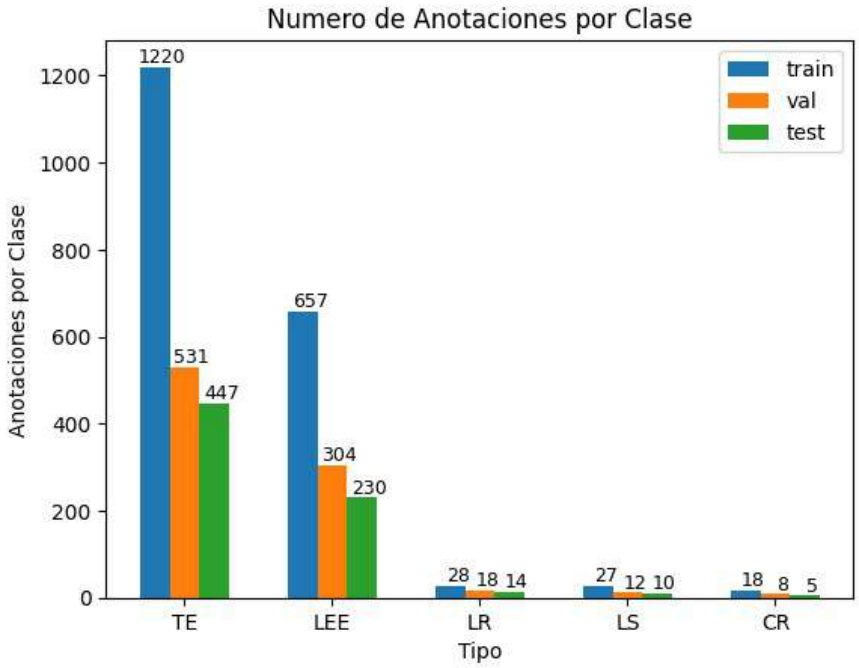


Figura 4.5: Anotaciones por clase luego de distribuir *train, val y test* del *dataset* de DS.UY.

4.3. Fase I: Comparación sobre Nordtank

Para obtener una primera evaluación del rendimiento de los modelos seleccionados y facilitar su comparación, se entrenaron durante 50 épocas sin aumentación de los datos y se mantuvieron los valores por defecto para sus hiperparámetros. Se optó por utilizar un tamaño de lote (*batch size*) de 16 en una GPU A100 para todos los modelos. Esta elección se basó en las restricciones específicas para cada modelo, las cuales fueron determinadas mediante experimentación previa.

En particular, en los modelos como Retina, Faster-RCNN y Detr, no era posible utilizar un tamaño de lote mayor a 16 en una GPU A100. Esta restricción se volvía aún peor al emplear una GPU P100. La limitación radica en que el tamaño de lote determina cuántas imágenes se procesan simultáneamente en cada paso de entrenamiento, y un tamaño de lote más grande requiere más memoria para almacenar las imágenes. Por ende, para aprovechar un tamaño de lote mayor y acelerar el proceso de entrenamiento, se decidió utilizar la GPU A100. Además, tras ejecutar los modelos durante 100 épocas, se observó que después de la época 50, no se apreciaba una diferencia significativa en el desempeño.

Todos los modelos se inicializaron utilizando pesos preentrenados. YOLO y SSD emplearon pesos previamente entrenados en el conjunto de datos COCO, mientras que Faster-RCNN, Retina, EfficientDet y DETR utilizaron pesos preentrenados de Imagenet.

Puede encontrarse información detallada sobre las diferencias de desempeño para cada modelo en la tabla 4.1.

Modelo	Tiempo	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
YOLOv5n	1h 31min	0.492	0.300	0.452
YOLOv8n	1h 35min	0.509	0.316	0.432
SSD	**1h 29min**	0.488	0.314	0.466
Retina	6h 30min	**0.648**	**0.427**	**0.612**
Detr	11h 14min	0.106	0.056	0.540
EfficientDet	5h 50min	0.549	0.363	0.582
Faster-RCNN	6h 59min	0.448	0.253	0.352

Tabla 4.1: Tabla comparativa del desempeño de cada modelo para 50 épocas.

De la tabla 4.1 se pueden resaltar los distintos puntos:

- **Tiempo de entrenamiento:** El modelo más rápido en términos de tiempo de entrenamiento es SSD con 1 hora y 29 minutos, seguido de YOLOv5n con 1 hora y 31 minutos y finalmente YOLOv8n con 1 hora y 35 minutos. Cabe resaltar que estos tres modelos difieren por pocos minutos entre ellos, mientras que el siguiente demora 5 horas y 50 minutos. Por otro lado, Detr es el modelo que requiere más tiempo para entrenar, con 11 horas y 14 minutos.

- **$mAP_{0.5}$**: Los modelos Retina y EfficientDet muestran los mejores resultados con $mAP_{0.5}$ de 0.648 y 0.549 respectivamente, lo que sugiere que tienen un mejor rendimiento en la detección de objetos en comparación con los otros modelos. Por otro lado, el modelo Detr muestra el peor desempeño con un $mAP_{0.5}$ de solo 0.106, lo que indica que tiene dificultades significativas en la detección precisa de objetos en las imágenes.

- **$mAP_{0.5:0.95}$**: Similar al análisis anterior, los modelos Retina y EfficientDet muestran los mejores resultados, con valores de $mAP_{0.5:0.95}$ de 0.427 y 0.363 respectivamente. Esto sugiere que estos modelos son consistentes en la detección de objetos en un rango amplio de umbrales de confianza, desde 0.5 hasta 0.95. Por otro lado, el modelo Detr sigue mostrando el peor desempeño, con un $mAP_{0.5:0.95}$ de solo 0.056, lo que indica que también tiene dificultades en la detección de objetos con altos umbrales de IoU.

- **$Recall_{0.5:0.95}$**: Los modelos Retina y EfficientDet muestran los mejores resultados en términos de *recall*, con valores de 0.612 y 0.582 respectivamente. Esto sugiere que estos modelos son más efectivos para detectar la mayoría de los objetos presentes en las imágenes, incluso a través de diferentes umbrales de confianza. Por otro lado, el modelo Detr sigue mostrando un desempeño inferior en cuanto al *recall*,

con un valor de 0.540. Esto indica que el modelo puede estar dejando pasar una cantidad considerable de objetos que debería detectar.

Si se prioriza la velocidad de entrenamiento, SSD, YOLOv5 y YOLOv8n son buenas opciones. Si se busca el mejor mAP y *recall*, Retina y EfficientDet son modelos destacados, aunque con tiempos de entrenamiento más largos. Detr muestra un rendimiento general inferior en comparación con los otros modelos. A continuación se realiza un análisis más detallado del desempeño de cada uno de los modelos.

Faster-RCNN

Modelo entrenado con el *framework* de Detectron 2 (R. Girshick, Radosavovic, Gkioxari, Dollár, y He, 2019), el mismo consume una gran cantidad de memoria VRAM de la GPU y para ser ejecutado con un *batch size* de 16, fue necesaria la utilización de los nodos con la GPU A100.

Como se mencionó anteriormente y se evidencia en el artículo (Shihavuddin y cols., 2019), donde se entrena la arquitectura con ResNet-50 (R50) y ResNet-101 (R101), entre otros, Faster-RCNN permite emplear distintos *backbones* fácilmente. En teoría, a medida que se aumenta el número de capas en el *backbone*, se esperaría una mayor capacidad para extraer características complejas y aprender representaciones más profundas de los datos. No obstante, en dicho documento, sorprendentemente, no se observan diferencias significativas entre los dos *backbones* mencionados. De hecho, el mejor resultado de mAP obtenido entre estas dos configuraciones fue de 0.776, logrado con ResNet-50 en conjunto con la técnica de aumentación *Pyramid + Patching*.

Se comenzó con la configuración por defecto con un *backbone* (extractor de características) de 50 capas sin congelamiento (*freezing*) en sus pesos, *batch size* de 16 y una tasa de aprendizaje de 0.001, entre otros hiperparámetros.

A pesar de haber comenzado inixcialmente con el *backbone* de 50 capas, también se realizó un entrenamiento con un *backbone* más profundo de 101 capas. Esto resultó en un aumento significativo del $\text{mAP}_{0.5}$ a expensas de un mayor consumo de recursos computacionales, como se puede observar en la Tabla 4.2. El tiempo requerido para este entrenamiento se incrementó aproximadamente en una hora. Además, se utilizó un parámetro por defecto inicialmente no empleado, el cual permite filtrar imágenes sin anotaciones. Esta acción produjo una mejora de 0.1 en el mAP, y también se observó una notable mejora en el *recall*, aumentando en 0.176. Estos resultados sugieren que las imágenes sin anotaciones, especialmente en este modelo y configuración por defecto, no contribuyen significativamente al entrenamiento.

Backbone	Tiempo (50 épocas)	$\text{mAP}_{0.5}$	$\text{mAP}_{0.5:0.95}$	$\text{Recall}_{0.5:0.95}$
Default R50	**6h 54min**	0.448	0.253	0.352
Default R50 (con filtrado)	7h 3min	0.554	0.351	**0.528**
Default R101	8h 20min	**0.609**	**0.393**	0.526

Tabla 4.2: Desempeño de los distintos *backbones* de Faster-RCNN para 50 épocas.

Al considerar que los modelos de dos etapas, a pesar de ser más costosos, suelen ser más precisos que los de una sola etapa, se decidió extender el entrenamiento a 100 épocas. Esto se hizo con la esperanza de que, al igual que en las primeras 50 épocas, el filtrado de imágenes, un hiperparámetro que previamente había demostrado mejorar el rendimiento del *backbone*, pudiera seguir contribuyendo a la mejora del modelo.

Backbone	Tiempo (100 épocas)	$\text{mAP}_{0.5}$	$\text{mAP}_{0.5:0.95}$	$\text{Recall}_{0.5:0.95}$
Default R50 (con filtrado)	**17h 40min**	0.531	**0.348**	0.565
Default R101 (con filtrado)	18h 15min	**0.583**	0.337	**0.558**

Tabla 4.3: Desempeño de los distintos *backbones* de Faster-RCNN para 100 épocas.

Luego de correr nuevamente 100 épocas, los modelos empeoraron su precisión frente a las corridas de 50 épocas, solo mejoraron levemente el *recall*. Esto puede deberse a que, cuando se filtran las imágenes sin anotaciones, el conjunto de entrenamiento queda excesivamente reducido, llevando a que el modelo sobreajuste.

RetinaNet

Al igual que Faster-RCNN este modelo fue ejecutado con el *framework* de Detectron 2. A su vez, las primeras corridas fueron realizadas con la configuración por defecto, sin filtrar imágenes sin anotaciones. Luego, de los hiperparámetros disponibles, se bajó el umbral de IoU de 0.5 a 0.35 para el algoritmo de *Non-Max Suppresion*. Esta decisión se tomó dado que, al ser daños superficiales, la superposición en una misma clase es escasa. Sin embargo, como se puede ver en la tabla 4.4, con este ajuste no se notaron cambios significativos. También se observó que el filtrado de imágenes sin anotaciones no condujo a mejoras significativas en este modelo. Esto puede atribuirse principalmente a la función de costo de RetinaNet para clasificación, conocida como *Focal Loss*. Como se explicó anteriormente en la subsección 2.6.2, esta función de costo aborda el problema de ponderación de instancias negativas (fondo) excesivas en conjuntos de datos de detección de objetos.

Configuración	$\text{mAP}_{0.5}$	$\text{mAP}_{0.5:0.95}$	$\text{Recall}_{0.5:0.95}$	AP-dmg	AP-dirt
Por defecto (nms_iou=0.5)	0.644	0.420	0.611	0.347	0.492
nms_iou=0.35	**0.648**	**0.427**	**0.612**	**0.348**	**0.507**
Por defecto (con filtrado)	0.571	0.375	0.598	0.303	0.447

Tabla 4.4: Desempeño de las experimentaciones con Retina para 50 épocas.

Detr

El entrenamiento se realizó utilizando el modelo implementado en el repositorio de GitHub (woctezuma, 2021). Sin embargo, los resultados obtenidos no cumplieron con las expectativas, ya que el $\text{mAP}_{0.5}$ alcanzó un valor de 0.101 y el *recall* de 0.540.

Es crucial señalar que, si bien el *recall* obtenido es satisfactorio e incluso superior al de otros modelos, su precisión es notablemente inferior en comparación con estos. Sabiendo esto, y debido al excesivo tiempo y recursos requeridos para lograr estos resultados, se determinó que no era viable continuar con la optimización de este modelo, especialmente cuando otros modelos demostraron un rendimiento satisfactorio en menos tiempo.

EfficentDet

Para este modelo se utilizó la implementación de GitHub (zylo117, 2020). El tipo de anotaciones utilizadas para entrenar EfficentDet fueron COCO, por lo cual se tuvieron que transformar las anotaciones originales en formato YOLO a este nuevo formato. Aunque los resultados obtenidos fueron satisfactorios, no fueron suficientes como para considerar este modelo como uno de los dos mejores dentro del marco del proyecto. Modelos seis veces más rápidos llegaron a resultados similares.

YOLO

Se comenzó corriendo el modelo YOLOv5n (Jocher, 2020) con sus hiperparámetros por defecto. Luego se hizo este mismo procedimiento con YOLOv8n (Jocher, Chaurasia, y Qiu, 2023) obteniendo mejores resultados en mAP como se ve en la tabla 4.1, con una ínfima diferencia de tiempo extra. Es por esto que este último que se eligió como modelo a comparar para la siguiente etapa.

SSD

Se comenzó corriendo el modelo SSD (Liu y cols., 2015) con sus hiperparámetros por defecto. El tipo de anotaciones utilizadas para entrenar SSD fueron Pascal VOC XML, por lo cual se tuvieron que transformar las anotaciones originales en formato YOLO a este nuevo formato. Este modelo fue el más rápido, pero únicamente por unos minutos de diferencia respecto a otros dos modelos: YOLOv5 y YOLOv8n. Dado que de los tres modelos más rápidos, SSD presentó el $\text{mAP}_{0.5}$ más bajo, se optó por no seguir con este modelo. Cabe destacar que el cuarto modelo más rápido demora casi 5 horas más.

Conclusiones

Después de llevar a cabo las corridas detalladas anteriormente, se llegó a la conclusión de que YOLOv8n y RetinaNet son las opciones más sólidas para las próximas fases. YOLOv8n destaca por su equilibrio entre velocidad y precisión, manteniendo resultados competitivos en $mAP_{0.5}$ y *recall* mientras ofrece tiempos de entrenamiento similares a modelos más rápidos como SSD. Por otro lado, RetinaNet sobresale por su excepcional precisión en la detección de objetos, mostrando los mejores resultados en términos de $mAP_{0.5}$ y *recall*, aunque esto implica un ligero sacrificio en velocidad. Cabe destacar que la diferencia de velocidad es sumamente amplia, mientras los modelos mas rápidos demoran 1 hora y 30 minuntos aproximadamente, RetinaNet llega a demorar casi 6 horas.

Durante estas evaluaciones, también se consideraron los modelos SSD, Detr, EfficientDet y Faster-RCNN, cada uno con sus propias características distintivas en cuanto a velocidad de entrenamiento y precisión. SSD destacó por su rapidez, aunque sacrificó precisión en comparación con otros modelos. Detr, por otro lado, mostró un rendimiento notablemente inferior, lo que sugiere limitaciones en su aplicabilidad en este contexto específico. A pesar de que EfficientDet demostró resultados satisfactorios, modelos más rápidos alcanzaron resultados similares, lo que lo relegó como una opción. Además, Faster-RCNN mostró ser competitivo en términos de precisión, especialmente al emplear *backbones* más profundos, pero su exigencia computacional lo posicionó como una opción menos práctica.

4.4. Fase II: Aumentaciones sobre Nordtank

El relevamiento de los antecedentes mostró que el uso de aumentaciones es fundamental para mejorar la capacidad de generalización de los modelos. El objetivo de la segunda experimentación es evaluar que tanto el desempeño de los modelos mejora al utilizar imágenes aumentadas. Para el caso de estudio (Shihavuddin y cols., 2019), la mejora de los resultados fue significativa, mostrando un incremento en todas las métricas. En el caso del mejor modelo, Inception-ResNet-V2, se pasó de tener 27.5 a 81.1 de mAP con la mejor combinación de aumentaciones (*Pyramid + patching + regular*).

Para realizar las pruebas se eligieron dos modelos de la tabla 4.1. A Retina por ser el modelo con mayor $mAP_{0.5}$ (además del *recall* más alto) y YOLOv8n por ser el modelo con mayor $mAP_{0.5}$ entre los más rápidos. Se realizaron 3 *datasets* nuevos con aumentaciones como se ve en la tabla 4.5 los cuales son generados a partir del *dataset* de Nordtank.

Para estandarizar los experimentos y asegurar una comparación justa entre ambos modelos, se empleó la librería de Python *imgaug* (Jung y cols., 2020) para la implementación de las aumentaciones (D_1,D_2 y D_3). Sin embargo, el *framework* de YOLOv8 tiene la capacidad de realizar aumentaciones en tiempo de entrenamiento, y también incluye valores por defecto que han sido determinados empíricamente por el equipo de Ultralytics en diversos conjuntos de datos. Es debido a esto que se agrega un caso D_4 (particular de YOLOv8) para evaluar los valores por defecto. Estas aumentaciones por defecto combinan tanto transformaciones afines como cromáticas, pero no se tiene la posibilidad de utilizar transformaciones que agreguen ruido. Sin embargo, es importante destacar que el conjunto de datos D_4 generado por el framework YOLO, aunque útil para evaluar los valores por defecto de aumentación, puede no ofrecer una comparación totalmente justa debido a su combinación de transformaciones geométricas y cromáticas. Aun así, se emplearon con el objetivo de evaluar si se obtendrían mejores resultados. Por otro lado, Detectron 2 el *framework* de RetinaNet y Faster-RCNN, tiene la posibilidad de definir aumentaciones, pero no posee valores por defecto definidos.

La elección de los conjuntos de aumentaciones se basó en la generación de "grupos semánticos". Estos grupos se refieren a categorías de cambios aplicados a las imágenes. Los grupos semánticos seleccionados son los siguientes:

- Transformaciones afines: Todas aquellas transformaciones que influyen sobre la forma de la imagen. Esto incluye transformaciones geométricas como rotación, estiramientos, volteos horizontales o verticales. Estas transformaciones modifican la geometría de la imagen original para obtener diferentes perspectivas o aspectos visuales.

- Transformaciones cromáticas: Todas aquellas transformaciones que cambien el color de la foto, estas incluyen: tono, brillo, saturación y contraste.

- Transformaciones ruidosas: Todas aquellas transformaciones que agreguen algún tipo de ruido. Estas incluyen el ruido gaussiano, desenfoque, etc.

Para el caso de las transformaciones afines, se realizaron aleatoriamente entre dos y cinco transformaciones de las listadas en la tabla 4.5. El cambio de tamaño se realizó de manera uniforme, dejando la imagen entre un 60 % y un 120 % de su tamaño original. Ejemplos pueden ser vistos en la figura 4.6.

Por otro lado, se aplicaron entre una y tres transformaciones cromáticas de forma aleatoria a cada imagen, seleccionadas de las listadas. Las distribuciones para estas alteraciones también fueron uniformes, (0.5, 1.5) para el contraste, (-25,25) para el brillo y (-10, 10) para la saturación. Estos valores se eligieron subjetivamente tratando de mantener la integridad de la imagen. Ejemplos pueden ser vistos en la figura 4.7.

Finalmente, las transformaciones ruidosas también se aplicaron para cada imagen de forma aleatoria entre una y tres de las listadas en la tabla. Al igual que con las cromáticas, los parámetros se seleccionaron subjetivamente para preservar la integridad de las imágenes. El desenfoque gaussiano se realizó con el valor de sigma uniforme entre cero y uno. Los valores por defecto para el ruido gaussiano quedaron sin cambios, lo que agrega ruido que varía uniformemente entre 0 y 127.5 . Para el caso de la lluvia se definió su velocidad en 0.1. Todos estos parámetros son configurables con la librería *imgaug*. Ejemplos pueden ser vistos en la figura 4.8.

Para cada uno de los *datasets* a excepción de D_4 se entrenó con ambos modelos, RetinaNet y YOLOv8n.

Figura 4.6: Ejemplos visuales de transformaciones afines.

Figura 4.7: Ejemplos visuales de transformaciones con cambios cromáticos.

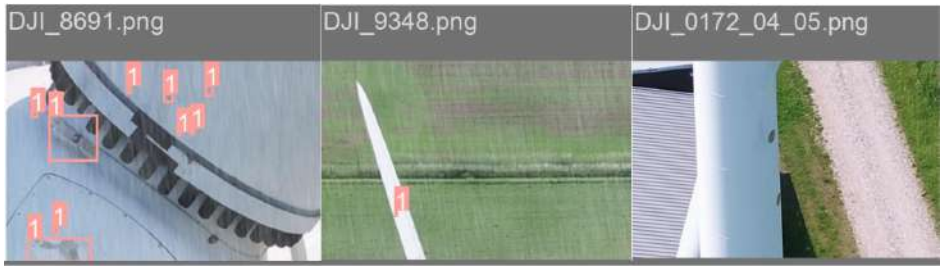


Figura 4.8: Ejemplos visuales de transformaciones con ruido.

Dataset	Tamaño	Aumentaciones
D_0	7543	Ninguna
D_1 (afines)	12541	Rotación 90°, 180° o 270° + Flip Horizontal o Vertical + Cambio de tamaño
D_2 (cromáticas)	12541	Contraste + Brillo + Saturación
D_3 (ruidosas)	12541	Desenfoque Gaussiano + Ruido Gaussiano Aditivo + Lluvia
D_4 (por defecto)	N.A.	Tono + Saturación + Brillo + Traslación + Flip Horizontal + Zoom out + Mosaico

Tabla 4.5: Descripción de las aumentaciones.

Modelo	*Dataset*	GPU	Épocas	Tiempo	Batch Size	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
Retina	D_0	A100	50	6h 30min	16	0.648	0.427	0.611
Retina	D_1	A100	50	10h 55min	16	**0.656**	**0.427**	**0.616**
Retina	D_2	A100	50	15h 5min	16	0.656	0.427	0.616
Retina	D_3	A100	50	15h 2min	16	0.619	0.406	0.569
YOLOv8n	D_0	A100	50	1h 35min	16	0.519	0.309	0.519
YOLOv8n	D_1	P100	50	2h 49min	16	**0.571**	**0.348**	**0.625**
YOLOv8n	D_2	P100	50	5h 27min	16	0.508	0.318	0.550
YOLOv8n	D_3	P100	50	5h 30min	16	0.511	0.309	0.556
YOLOv8n	D_4	A100	50	5h 10min	16	0.655	0.428	0.708

Tabla 4.6: Tabla comparativa del desempeño de cada modelo con aumentaciones.

De la tabla 4.6, se puede observar que los modelos entrenados con D_1 (aumentaciones geométricas) y D_2 (cambios cromáticos) tienen un rendimiento ligeramente superior al modelo entrenado con D_0 (sin aumentaciones). Sin embargo, el modelo entrenado con D_3 (agregando desenfoque y ruido) tiene un rendimiento ligeramente inferior.

Además, podemos observar que el modelo Retina tiene un rendimiento en términos generales mejor que el modelo YOLOv8n en todos los conjuntos de datos aumentados, especialmente en D_1 y D_2, esto es esperable dada la comparación inicial 4.1.

El entrenamiento especial con las aumentaciones por defecto de YOLOv8(D_4) muestra que existe una mejora significativa en el desempeño del modelo, equiparándose con RetinaNet en la mitad de tiempo, incluso superando en la métrica de $Recall_{0.5:0.95}$. Esta última métrica la consideramos aún más valiosa para la detección temprana de daños, este resultado, entre otras consideraciones, determinó la decisión de usar YOLOv8 para futuras experimentaciones utilizando sus propias aumentaciones.

Además de que el entrenamiento de Retina es más lento que el de YOLO, se decidió evaluar el tiempo de inferencia individualmente de cada uno en una misma imagen de Nordtank con el objetivo de tener una comparación más completa. Retina tiene un tiempo de inferencia de aproximadamente 0.7s por imagen. Por otro lado, YOLO, tiene un tiempo de inferencia aproximado de 0.12s por imagen. El propio framework de Ultralytics a su vez provee un desglose en los tiempos, donde indica tiempo de preprocesamiento, inferencia y postprocesamiento. Estas inferencias fueron realizadas sobre las GPU P100.

4.4.1. Conclusiones y evaluación final de Nordtank

Uno de los aspectos a resaltar de estas experimentaciones, es que los cambios cromáticos y sobre todo las transformaciones afines pueden mejorar el rendimiento del modelo. En contraste, la adición de ruido en principio no parece mejorar el desempeño general. Cabe destacar que la cantidad de ruido agregada pudo haber sido excesiva, para el caso de YOLO, existe una ligera mejora en el $Recall_{0.5:0.95}$ con el *dataset* D_3, lo que sugiere que la exploración de una configuración óptima para estas aumentaciones podría ser valiosa.

De la tabla 4.7 se revela el rendimiento del modelo YOLOv8 en el conjunto de entrenamiento con aumentaciones por defecto (D4) para Nordtank, específicamente para las clases “Suciedad” y “Daño”. En el conjunto de *val*, se destaca la efectividad general del modelo, con un $mAP_{0.5}$ de 0.655. Se puede observar una buena capacidad de detección en la clase “Suciedad” con un $mAP_{0.5}$ de 0.706 y un $Recall_{0.5:0.95}$ de 0.821. Este alto $Recall_{0.5:0.95}$ es de especial importancia para realizar un sistema de sugerencia para expertos en palas de aerogeneradores, dado que es capaz de encontrar una alta cantidad de instancias de suciedad,

minimizando la presencia de falsos positivos. Sin embargo, la clase "Daño" muestra un rendimiento más modesto, con un $mAP_{0.5}$ de 0.603 y un $Recall_{0.5:0.95}$ de 0.526.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
Todas	0.655	0.428	0.674
Suciedad	0.706	0.526	0.821
Daño	0.603	0.331	0.526

Tabla 4.7: Desempeño en *val* para YOLOv8 con entrenamiento en el dataset D4.

Estas tendencias se confirman en el conjunto de *test*, como se puede ver en la tabla y 4.8, subrayando la consistencia del modelo. Esto apunta a una eficiente detección de suciedad, mientras que la detección de daño podría beneficiarse de ajustes adicionales para mejorar su desempeño. Los resultados muestran que para trabajos futuros se deberían realizar optimizaciones de los hiperparámetros con un énfasis particular en la clase "Daño", dado que estos presentan mayor riesgo que la suciedad en cuanto a ruptura, llevando a mayores costos de reparación. Cabe destacar que este es un resultado inesperado, debido a que la cantidad de instancias de suciedad es considerablemente menor, abarcando los daños un 93.79 % del total de instancias, como se puede ver en la figura 3.3b (sección 3.4).

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
Todas	0.645	0.408	0.668
Suciedad	0.682	0.487	0.797
Daño	0.609	0.329	0.538

Tabla 4.8: Desempeño en *test* para YOLOv8 con entrenamiento en el dataset D4.

Como una última apreciación, a partir de las siguientes fases solamente se trabajará sobre el *dataset* de DS.UY, el más valioso en el contexto de este proyecto.

4.5. Fase III: Entrenamiento de YOLOv8 con DS.UY

Se decidió seleccionar el *framework* de YOLOv8 en lugar de las otras opciones disponibles para continuar con la tercera etapa de experimentación por varias razones. En primer lugar, YOLOv8n se destaca por su equilibrio entre eficiencia y eficacia, generando resultados satisfactorios en menos tiempo en comparación con la mayoría y utilizando menos cantidad de memoria VRAM. Gracias a su eficiente uso de memoria es posible entrenarse con un *batch size* de 16 en cualquier GPU P100 disponible en el clúster, en lugar de estar limitados únicamente a las dos GPU A100. Esto aumenta la flexibilidad y escalabilidad de las experimentaciones.

Además, desde una perspectiva de facilidad de uso del *framework* y obtención de métricas, YOLOv8 se presenta como un paquete completo. Ofrece una gran cantidad de las características y herramientas necesarias para abordar los diversos desafíos del proyecto, lo que ahorra tiempo y esfuerzo en la implementación de funcionalidades adicionales.

También es importante destacar que cuando se probó con la configuración por defecto, incluyendo sus aumentaciones, YOLOv8 mostró un buen mAP y el mayor $\text{Recall}_{0.5:0.95}$ de todos los entrenamientos. Esto proporciona una base sólida para iniciar las investigaciones y desarrollos, acelerando el progreso inicial.

Por último, pero no menos importante, a lo largo de su evolución, YOLO ha mostrado un progreso constante, alcanzando su última versión, YOLOv8, que es ampliamente utilizada y respaldada por una comunidad sólida. Esta mejora continua da confianza en su capacidad para seguir adaptándose a las necesidades cambiantes de este y futuros proyectos.

En esta etapa del proyecto el *dataset* DS.UY comentado en la sección 3.4 fue proporcionado para ser utilizado. Esto es de especial importancia, dado que es de mayor interés realizar un análisis de los datos representativos de la realidad local. Dada esta situación, se optó por discontinuar los experimentos con el conjunto de datos extranjero proveniente de Dinamarca (Nordtank). Cabe destacar que ambos conjuntos de datos tienen una distribución fuertemente distinta, desde los tipos de daños vistos hasta la definición y formato de captura de imágenes.

4.5.1. Refinamiento de Anotaciones

En esta sección, se detalla el proceso de refinamiento de las anotaciones del *dataset* DS.UY, una tarea que fue necesaria para mejorar la calidad de las predicciones.

Con el objetivo de verificar la calidad del *dataset* inicialmente proporcionado, se realizó una corrida especial, donde se entrenó y validó con el mismo conjunto de datos. Al estar entrenando y validando sobre el mismo conjunto, se espera ver un $\text{mAP}_{0.5}$ alto, implicando que es capaz de aprender sobre lo que entrenó.

Después de llevar a cabo el entrenamiento del modelo YOLOv8n con el conjunto de datos de *train* del *dataset* inicial, durante 100 épocas, se validó sobre el mismo conjunto de datos y se encontró un modesto rendimiento: un 86.26 % de $\text{mAP}_{0.5}$ y un recall del 81.65 %. Las 100 épocas fueron suficientes para llegar a un sobreajuste y que la validación comenzara a aumentar.

Estos resultados indicaron la existencia de problemas en el aprendizaje del modelo, y se comenzó a identificar deficiencias en las anotaciones. A partir de dicho análisis, se encontraron algunas irregularidades, entre ellas, la presencia de imágenes duplicadas con nombres distintos, anotaciones faltantes e incluso algunas confusiones entre clases. Algunas deficiencias de estas anotaciones se pueden ver evidenciadas en la figura 4.9.

Este descubrimiento llevó a la conclusión de que una mejora cualitativa de las anotaciones era esencial para el éxito del modelo.

Tomando en cuenta lo anterior, y considerando que la cantidad de imágenes era relativamente manejable, se implementaron mejoras sustanciales en las anotaciones del mismo. Se eliminaron imágenes incorrectamente anotadas o duplicadas y se corrigieron confusiones entre clases en las anotaciones. Es importante mencionar que este proceso está sujeto a error, dado que estas correcciones no están hechas por expertos. Por otro lado, otro problema recurrente encontrado en la anotación es la falta de estandarización sobre daños con difícil localización. Por ejemplo, la figura 4.10 presenta daños del tipo TE, el mismo es difícil de localizar, podría definirse como una sola instancia o varias. Aquí no es claro dónde comienza una instancia y dónde termina otra. Esto puede resultar problemático porque si el modelo predice varias instancias para un daño que en la anotación se define como una sola (o viceversa), es probable que las predicciones no se consideren como "verdaderas positivas" al calcular el IoU si no se supera un umbral mínimo predefinido.

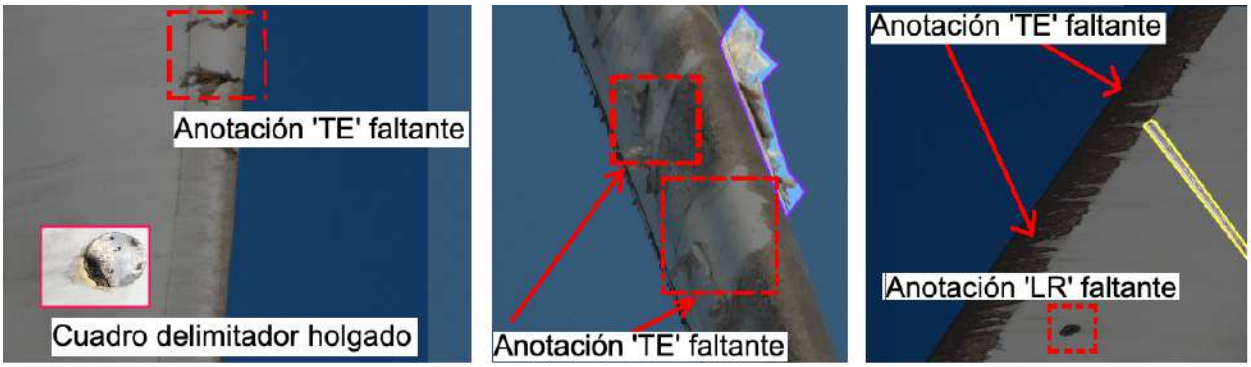


Figura 4.9: Ejemplos de anotaciones faltantes o mejorables.

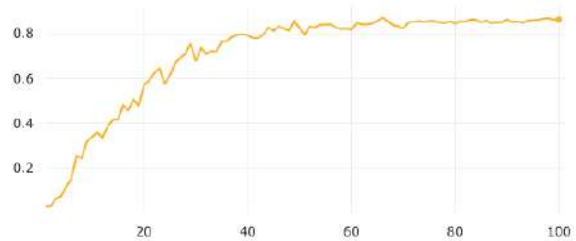
Figura 4.10: Daño difícil de localizar.

Estos tipos de daños difíciles de localizar podrían verse beneficiados con un tipo de anotación poligonal, y realizando la tarea de segmentación. Por otro lado, al momento de evaluar los modelos se podría utilizar un umbral de IoU más bajo con el propósito de ser más permisivos con la localización. Un documento que realiza este cambio es (Shihavuddin y cols., 2019), el cual toma un umbral de 0.3, distinto al estándar generalmente utilizado 0.5 y 0.5:0.95.

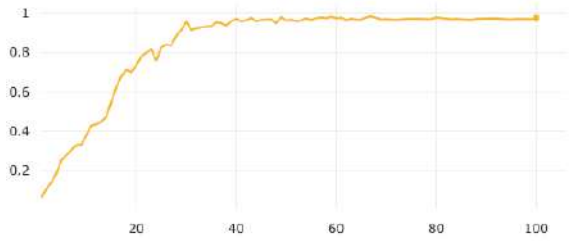
A pesar de persistir esta problemática de localización, con el propósito de medir si existe una mejora sustancial en el refinamiento, se realizó el mismo proceso de entrenamiento y validación con el conjunto refinado de datos.

Este proceso resultó ser crucial para mejorar la calidad del conjunto de datos. Anteriormente, la medida de $\text{mAP}_{0.5}$ se estancaba 86.26 %. Tras estas mejoras, se observa una notable transformación en los resultados, logrando un $\text{mAP}_{0.5}$ del 96 % y un recall del 88 % como se evidencia en la figura 4.11a y 4.11b.

Es importante destacar que ambas configuraciones fueron entrenadas durante tan solo 100 épocas, resaltando así la diferencia que unas anotaciones precisas pueden tener en el rendimiento del modelo de detección de objetos.



(a) $\text{mAP}_{0.5}$ del *dataset* DS.UY inicial.



(b) $\text{mAP}_{0.5}$ del *dataset* DS.UY refinado.

Figura 4.11: Gráfica de $\text{mAP}_{0.5}$ vs épocas.

4.5.2. Análisis del desempeño de cada clase

Posteriormente, habiendo confirmado que la calidad de los datos era aceptable, con el último modelo entrenado sobre el conjunto refinado se realizó la validación, pero esta vez utilizando el conjunto de validación previamente separado. Esto permitió evaluar el rendimiento inicial del modelo para cada una de las clases en instancias no vistas previamente. En la tabla 4.9, se puede observar cómo las clases con un mayor número de instancias, como LEE y TE, exhiben un rendimiento superior en comparación con aquellas clases que tienen un número reducido de instancias, como CR y LS. Además, se observa que a pesar de contar con pocas instancias, la clase LR muestra un buen rendimiento. Este fenómeno podría explicarse por la facilidad con la que se puede identificar esta clase, dada la escasa variabilidad de la misma. Por ejemplo, la forma del daño que ocurre en el receptor de rayos (LR) es siempre circular, a diferencia de los daños en otras clases que presentan formas más variables.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
Todas	0.328	0.163	0.357
CR	0.008	0.002	0
LEE	0.256	0.100	0.329
LR	0.611	0.311	0.750
LS	0.174	0.095	0.111
TE	0.589	0.312	0.596

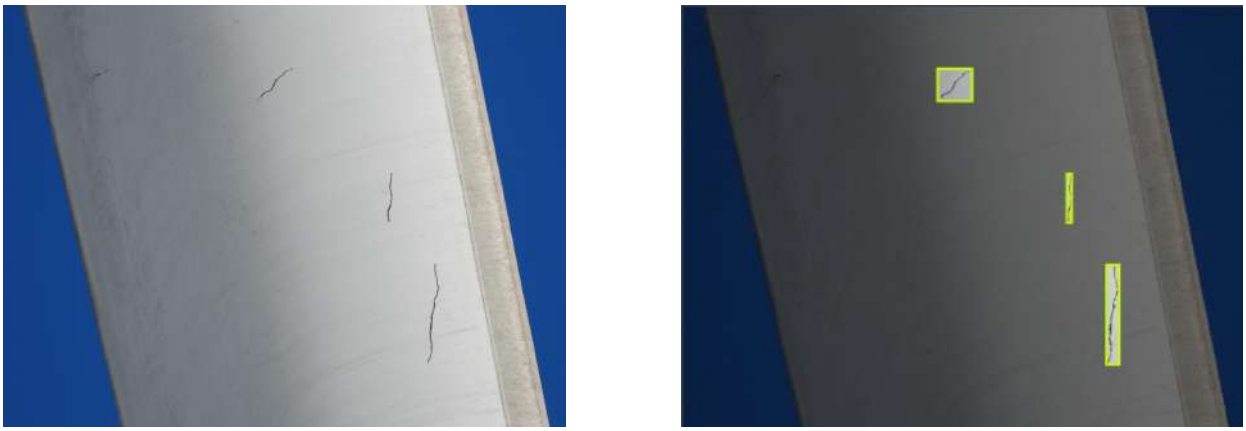
Tabla 4.9: Desempeño para cada clase corriendo 100 épocas.

A partir de los resultados obtenidos, se optó por aumentar la cantidad de instancias de la clase CR con imágenes sintéticas. Es relevante señalar que el proveedor de las imágenes de DS.UY enfatizó que, entre todas las clases, la clase CR tiende a ser la más grave, lo que la convierte en un aspecto crucial para la detección temprana. Con el objetivo de no opacar la presencia de las instancias reales, se crearon únicamente 13 nuevas imágenes que se sumaron a las otras 24 que había originalmente. Para crearlas se utilizó un *dataset* de Roboflow que contiene *cracks* segmentados en hormigón (Raipur, 2023), un ejemplo de estos cracks se puede ver en la imagen 4.12. Se aislaron los segmentos de crack y se los colocó aleatoriamente sobre la superficie de palas sanas del *dataset* original de DS.UY, un ejemplo se puede observar en la imagen 4.13a con sus respectivas anotaciones mostradas en la imagen 4.13b.

(a) *Crack* en hormigón.

(b) Respectiva segmentación.

Figura 4.12: *Crack* en hormigón y su segmentación.



(a) Pala con *cracks* sintéticos.

(b) Respectivas anotaciones.

Figura 4.13: Pala con *cracks* sintéticos y sus anotaciones

Como se puede apreciar en la tabla 4.10 el desempeño del modelo con el aumento de instancias de CR a partir de imágenes sintéticas no tuvo un cambio significativo. Esto se puede deber a la diferencia visual entre los daños originales de *crack* y los agregados artificialmente. Un ejemplo de *crack* real se puede ver en la imagen 4.14a con su respectiva anotación en 4.14b. Como el desempeño del modelo agregando las imágenes sintéticas no mejoró con respecto al *dataset* original para la clase CR, se decidió no utilizarlas para las siguientes experimentaciones.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
Todas	0.333	0.167	0.322
CR	0.006	0.004	0
LEE	0.239	0.0942	0.200
LR	0.352	0.200	0.417
LS	0.487	0.248	0.500
TE	0.580	0.289	0.494

Tabla 4.10: Desempeño para cada clase con la adición de imágenes sintéticas.

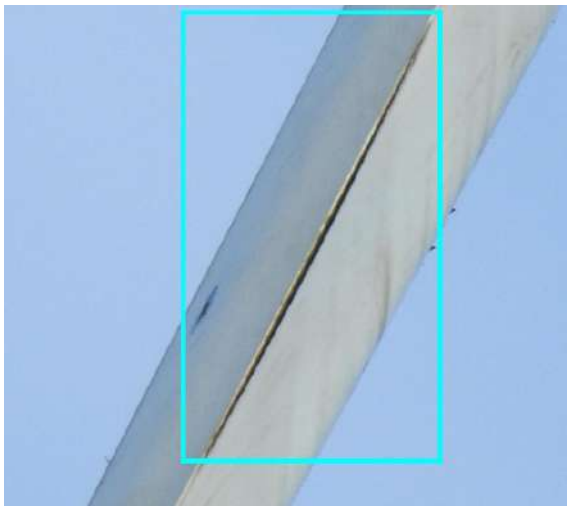
4.5.3. Aumentación de los datos

En esta nueva etapa se decidieron utilizar las aumentaciones por defecto de YOLOv8 que constan de cambios de tono, saturación, brillo, volteo horizontal, *zoom out* y mosaico. A su vez, se decidió agregar volteos verticales con probabilidad de 0.5 y rotaciones aleatorias entre $-90°$ y $90°$. Estas decisiones se tomaron teniendo en cuenta los resultados de la subsección 2.4.5 y la revisión de la literatura, donde se pudo evidenciar las mejoras que este tipo de aumentaciones provocaban en el desempeño de los modelos. Un ejemplo de esto se puede ver en el artículo (J. Zhang y cols., 2021). Asimismo, se decidió aumentar la cantidad de épocas de entrenamiento a 1000 debido a que la aumentación de los datos conlleva que el modelo tome más tiempo en sobreajustarse. Se puede observar en la tabla 4.11 cómo este aumento en los datos ha mejorado el desempeño del modelo en todas sus métricas. Por este motivo, se decidió continuar con esta configuración para las próximas experimentaciones.

4.5.4. Cambio del optimizador

En esta nueva experimentación, se optó por cambiar el optimizador del modelo a SGD. Esta decisión se basa en la observación de resultados prometedores en el artículo (Foster y cols., 2022), donde se evidencia su eficacia para YOLOv5, una versión anterior que comparte similitudes con YOLOv8. El optimizador por

(a) Pala con *crack* real.



(b) Respectiva anotación.

Figura 4.14: Pala con *crack* real y su anotación

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.396	0.177	0.439
CR	0.071	0.00896	0
LEE	0.347	0.139	0.414
LR	0.552	0.218	0.667
LS	0.407	0.212	0.493
TE	0.605	0.307	0.620

Tabla 4.11: Desempeño para cada clase corriendo 1000 épocas con aumentaciones.

defecto es uno automático que comienza utilizando AdamW para las primeras 10000 iteraciones y luego cambia a SGD para las iteraciones restantes. Según el creador de YOLOv8, Glenn Jocher, esta decisión está respaldada por experimentación extensiva y resultados empíricos. La elección se fundamenta en la observación de que el uso inicial de AdamW puede acelerar la convergencia, especialmente en las etapas tempranas de entrenamiento. Sin embargo, después de un cierto número de iteraciones (en este caso, 10000), cambiar a SGD puede ofrecer un ajuste fino más efectivo y un rendimiento global superior. Como se puede apreciar en la tabla 4.12, el rendimiento del modelo se mantiene muy similar al observado en la tabla 4.5, lo cual era de esperar dado que se ha empleado el mismo optimizador durante la mayoría de las iteraciones. Este hecho, junto con el deterioro en el rendimiento de la clase CR, que es la más difícil de reconocer, ha llevado a la decisión de emplear el optimizador automático por defecto en los próximos experimentos.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.389	0.183	0.442
CR	0.00437	0.00175	0
LEE	0.333	0.126	0.418
LR	0.562	0.258	0.750
LS	0.430	0.237	0.431
TE	0.617	0.294	0.612

Tabla 4.12: Desempeño para cada clase corriendo 1000 épocas con SGD.

4.5.5. Reducción de aumentaciones

En esta nueva experimentación se decidió quitar las rotaciones entre $-90°$ y $90°$ porque se consideraron agresivas al visualizarlas aplicadas en el conjunto de train, como se puede observar en la imagen 4.15. Por un lado, cuando hay muchas anotaciones que quedan parcialmente fuera de la imagen, puede perderse información importante. Además, las rotaciones que no son múltiplos exactos de $90°$ suelen tener una pérdida de información en sus *bounding boxes*, ya que es imposible determinar con precisión los límites reales del objeto con este tipo de anotación. Para abordar este problema, podría considerarse la anotación de los daños en formato poligonal, aunque estos se traduzcan finalmente a cuadros delimitadores justo antes de introducirlos en la red, un ejemplo de esto se ilustra en la figura 4.16.

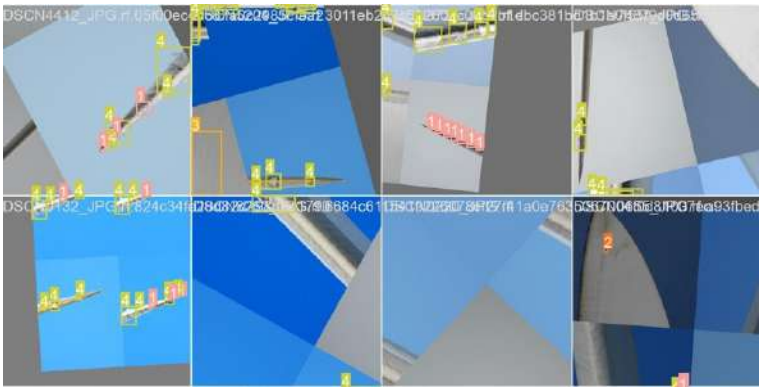


Figura 4.15: Ejemplos de rotaciones aplicadas en el conjunto de *train*.

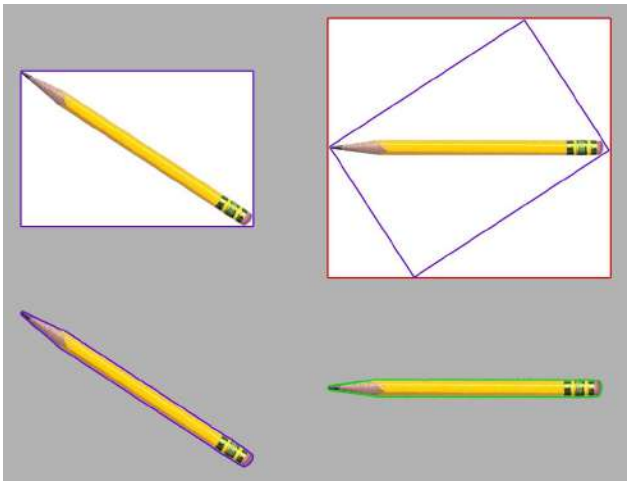


Figura 4.16: Ejemplo de rotaciones para cada formato de anotación.
(Dwyer, 2022)

Con el objetivo de confirmar este análisis, se ejecutó exactamente la misma configuración anterior de aumentaciones pero quitando las rotaciones. Los resultados pueden ser vistos en la tabla 4.13. Esto demuestra como efectivamente este tipo de aumentaciones no estaban ayudando al modelo, sino empeorando su desempeño general.

4.5.6. Regularización con *dropout*

Para paliar el sobreajuste que conlleva quitar las rotaciones, se optó por aplicar *dropout* al 50 % de las neuronas en cada iteración. Esto significa que durante cada iteración del entrenamiento, algunas neuronas no

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.410	0.226	0.419
CR	0.0152	0.00455	0
LEE	0.374	0.175	0.414
LR	0.574	0.320	0.667
LS	0.457	0.298	0.389
TE	0.629	0.330	0.625

Tabla 4.13: Desempeño para cada clase corriendo 1000 épocas aumentando sin rotaciones.

contribuirán a la propagación hacia adelante ni hacia atrás de la red. Este proceso fomenta que la red aprenda a confiar en diversas combinaciones de características, lo que aumenta su robustez y reduce la probabilidad de sobreajuste. En la tabla 4.14 se puede observar cómo este cambio ayudó significativamente al modelo. Por este motivo se seguirá utilizando esta configuración en las siguientes experimentaciones.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.448	0.232	0.481
CR	0.182	0.0771	0
LEE	0.393	0.174	0.398
LR	0.582	0.306	0.917
LS	0.445	0.259	0.444
TE	0.638	0.343	0.648

Tabla 4.14: Desempeño para cada clase corriendo 1000 épocas con *dropout*.

4.5.7. Incremento del tamaño de *batch size*

En esta nueva experimentación se decidió modificar el tamaño de *batch size*. Un tamaño de lote más grande permite que el modelo explore una mayor cantidad de datos durante cada iteración de entrenamiento. Esto puede ayudar al modelo a encontrar una solución que generalice mejor para datos nuevos. A su vez, la documentación de YOLOv8 recomienda utilizar el tamaño de lote más grande posible que permita el hardware disponible. Este enfoque se fundamenta en la mejora de las estadísticas de *batch normalization* a tener un tamaño de batch más grande(ver subsección 2.2.1 para más detalles), lo que contribuye a mejorar la estabilidad y eficacia del entrenamiento de la red neuronal (Ultralytics, 2023). Actualmente, el tamaño de *batch size* utilizado era de 16, el cual es el más grande que permite la GPU P100 para este modelo con una resolución de 1280 píxeles. Para poder aumentar este *batch size* a 32 se tuvo que disminuir la resolución a 960 píxeles. Como se puede observar en los resultados obtenidos en la tabla 4.15, el modelo mantiene un desempeño muy similar e incluso llega a empeorar levemente para $mAP_{0.5}$ y $Recall_{0.5:0.95}$. Este comportamiento se puede deber a la disminución en la resolución de las imágenes, dado que se pierden detalles que podrían ser relevantes para el aprendizaje, además de la existencia de muchas instancias de objetos pequeños.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.437	0.239	0.38
CR	0.136	0.0931	0
LEE	0.332	0.143	0.336
LR	0.608	0.374	0.5
LS	0.512	0.267	0.505
TE	0.599	0.317	0.558

Tabla 4.15: Desempeño para cada clase corriendo 1000 épocas con aumento de *batch size*.

4.5.8. Cambio del *backbone*

En esta fase de experimentación, se optó por emplear el modelo YOLOv8s, el cual presenta una estructura más profunda en comparación con YOLOv8n. Este último utiliza un total de 3.2 millones de parámetros, mientras que YOLOv8s utiliza 11.2 millones, casi cuatro veces más. Esta diferencia le permite a YOLOv8s capturar una gama más amplia de características y patrones en los datos. Debido al incremento en la complejidad del modelo, se hizo necesario reducir la resolución de las imágenes a 1080 píxeles para poder aprovechar la capacidad de procesamiento de la GPU P100.

Como se había observado en la sección 4.3, el modelo Faster-RCNN se benefició al emplear un *backbone* de mayor profundidad. Sin embargo, al analizar los resultados presentados en la tabla 4.16, se observa que el rendimiento del modelo se mantuvo bastante similar e incluso empeoró en algunos casos. Esta situación podría atribuirse a la disminución en la resolución de las imágenes, similar a lo observado al aumentar el tamaño del *batch size* en la sección anterior.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.435	0.23	0.371
CR	0.0132	0.00483	0
LEE	0.38	0.153	0.241
LR	0.675	0.395	0.667
LS	0.453	0.248	0.388
TE	0.654	0.351	0.561

Tabla 4.16: Desempeño para cada clase corriendo 1000 épocas con cambio de *backbone*.

4.5.9. Conclusiones

Durante la Fase III del desarrollo del modelo YOLOv8, se realizaron una serie de ajustes y cambios con el objetivo de mejorar su rendimiento. En primer lugar, se llevó a cabo un refinamiento de las anotaciones del conjunto de datos, lo que resultó en una mejora significativa en la precisión y calidad de los datos. Esta mejora en la calidad de los datos contribuyó a un aprendizaje más efectivo por parte del modelo, destacando la importancia de contar con datos y anotaciones de alta calidad en el entrenamiento de modelos de aprendizaje automático.

Asimismo, se realizó un análisis detallado del rendimiento del modelo en cada clase particular con el objetivo de identificar áreas específicas de mejora. Se encontró que la clase CR era la más difícil de detectar, y también la clase que tenía menos instancias en el conjunto de datos. Por otro lado, se observó que la clase LS, a pesar de también poseer pocas instancias, era más fácil de detectar. Para abordar el bajo rendimiento en la detección de la clase CR, se agregaron instancias sintéticas a los datos de entrenamiento. Sin embargo, estas instancias resultaron no aportar mejoras al rendimiento del modelo. Una posible explicación podría ser que las imágenes provienen de superficies de concreto en lugar de una superficie más parecida a la de una pala de aerogenerador, que sería más relevante para este contexto. Este desajuste en la fuente de las imágenes podría estar afectando negativamente la capacidad del modelo para generalizar y adaptarse de manera efectiva a las condiciones específicas que se busca simular.

Se exploró uno de los algoritmos de optimización recomendados en otros estudios revisados; sin embargo, los resultados obtenidos fueron similares o ligeramente inferiores. Además, se redujo la cantidad de aumentaciones aplicadas a los datos de entrenamiento, particularmente las rotaciones, las cuales estaban causando una degradación en las anotaciones. Este cambio contribuyó a una mejora general en el rendimiento del modelo.

Finalmente, se exploraron otros tamaños de *batch size* y profundidad del modelo, a costa de una peor definición de procesamiento. Estos cambios dieron resultados similares o incluso peores, lo que sugiere que es mejor dedicar los recursos de la VRAM a la definición de entrada por sobre el aumento del *batch size*. Sin embargo, esto no necesariamente significa que modelos más profundos o *batch size* más grandes sean peores, sino que, sugiere que encontrar un equilibrio adecuado entre estos factores puede llevar a un mejor rendimiento general del modelo.

4.6. Fase IV: evaluación final de DS.UY

El propósito de esta etapa final es evaluar el rendimiento del modelo YOLOv8 utilizando la configuración que demostró el mejor desempeño. Esta experimentación se llevará a cabo en el conjunto de *test* reservado al inicio de las experimentaciones. Como se ha deducido a lo largo de las secciones anteriores, la mejor configuración fue aquella que empleaba el modelo YOLOv8n junto con técnicas de aumento de datos como tono, saturación, brillo, volteo horizontal y vertical, *zoom out* y mosaico. Además, se utilizó un tamaño de lote (*batch size*) de 16, un optimizador automático que combina AdamW y SGD, y una tasa de abandono (*dropout*) de 0.5.

En la tabla 4.18, se presenta el rendimiento del modelo en el conjunto de *test*, utilizando los mejores pesos obtenidos durante la validación tras 1000 épocas de entrenamiento, cuyas métricas están detalladas en la tabla 4.17. Se observa que el rendimiento del modelo es ligeramente inferior, lo cual era de esperarse, ya que nunca había sido expuesto a las imágenes de *test*. Sin embargo, el rendimiento general para todas las clases se mantiene, indicando una capacidad adecuada de generalización del modelo.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.448	0.232	0.481
CR	0.182	0.0771	0
LEE	0.393	0.174	0.398
LR	0.582	0.306	0.917
LS	0.445	0.259	0.444
TE	0.638	0.343	0.648

Tabla 4.17: Desempeño en *val* para cada clase con la mejor configuración.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.392	0.193	0.4
CR	0.0219	0.0102	0
LEE	0.377	0.174	0.376
LR	0.508	0.265	0.6
LS	0.375	0.16	0.357
TE	0.678	0.355	0.669

Tabla 4.18: Desempeño en *test* para cada clase con la mejor configuración.

Después de completar esta evaluación, se tomó la decisión de combinar los conjuntos de *train* y *val* para entrenar el modelo utilizando la configuración previamente mencionada a lo largo de 1000 épocas. De esta manera, el modelo tendrá un conjunto de imágenes más amplio para aprender y captar los patrones necesarios para detectar daños en el conjunto de *test*. Los resultados presentados en la tabla 4.19 muestran una mejora significativa en el rendimiento del modelo con la inclusión de las imágenes del conjunto de *val*, lo que evidencia la importancia de contar con una mayor cantidad de datos para un aprendizaje más efectivo. A su vez, en la figura 4.17 se puede observar cómo en la matriz de confusión no hay casi confusiones entre clases, sino que la mayor complicación está en distinguir fondo de daño.

Clase	$mAP_{0.5}$	$mAP_{0.5:0.95}$	$Recall_{0.5:0.95}$
all	0.495	0.284	0.484
CR	0.0282	0.0108	0.2
LEE	0.386	0.203	0.43
LR	0.811	0.497	0.771
LS	0.57	0.331	0.346
TE	0.681	0.376	0.673

Tabla 4.19: Desempeño en *test* para cada clase entrenando con imágenes de *train* y *val* juntas.

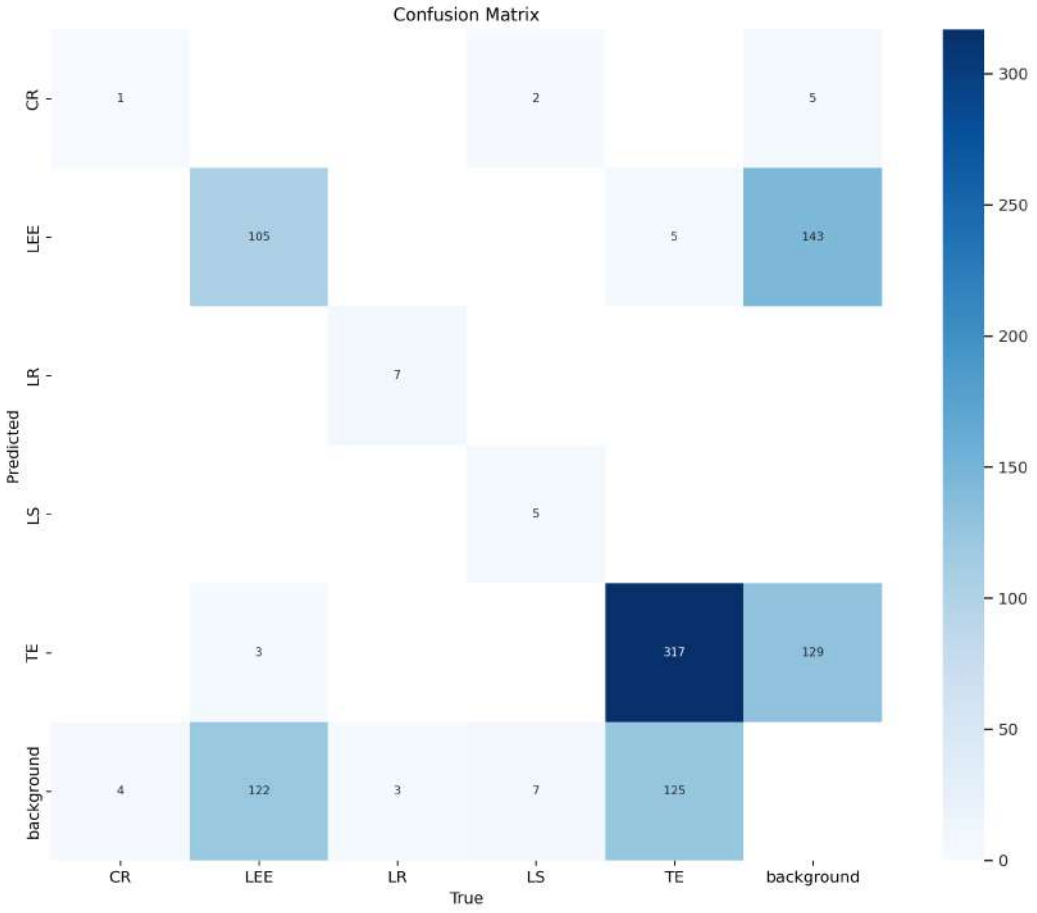


Figura 4.17: Matriz de confusión para la última experimentación juntando *train* y *val*.

Dado los resultados obtenidos hasta el momento, las modestas métricas sugieren que, aunque la intervención humana para la detección de daños aún es necesaria, la asistencia proporcionada por los modelos puede ser altamente beneficiosa para acelerar el proceso. Como es respaldado por diversos artículos (Iyer y cols., 2022), (Shihavuddin y cols., 2021), (Foster y cols., 2022).

Incluso, en (Nguyen y cols., 2022), se propone un proceso de control de calidad que integra anotaciones y reportes automáticos con la asistencia humana.

A pesar de que no fue cuantificado específicamente el tiempo requerido de anotación humana por imagen en este contexto, se estima que está en el rango de aproximadamente 30 segundos por imagen. Sin embargo, es importante tener en cuenta que este tiempo puede variar significativamente dependiendo de varios factores, como la complejidad y la definición de la imagen. Por ejemplo, imágenes de mayor resolución o con daños más sutiles podrían requerir un tiempo de análisis y anotación más prolongado.

Comparado con el artículo, donde se registra un tiempo de alrededor de 12 segundos por imagen, la estimación de 30 segundos representa una diferencia notable. La diferencia en los tiempos reportados podría atribuirse a variaciones en la metodología utilizada, como las características específicas de las imágenes evaluadas, la complejidad de los daños detectados o los criterios de evaluación del tiempo.

Sin embargo, es importante destacar que el ahorro de tiempo por anotación de imagen reportado en este último artículo es de alrededor de medio segundo por imagen, lo cual, si bien representa una mejora en la eficiencia del proceso, puede considerarse relativamente modesto en comparación con el potencial total de ahorro de tiempo que podría lograrse.

Para cuantificar de manera más precisa el esfuerzo ahorrado por el uso de la herramienta o modelo desarrollado en esta tesis, es crucial considerar el contexto en el que se aplicaría. Por ejemplo, si el modelo logra un recall de 1, lo que significa que identifica correctamente todos los daños en las imágenes, el ahorro de tiempo podría ser notable. Supongamos que se tiene un conjunto de imágenes considerablemente grande y se aplica el modelo sobre todas ellas, si el modelo logra reducir la cantidad de imágenes que un trabajador humano necesita revisar a la mitad, entonces se estaría ahorrando una cantidad significativa de tiempo y recursos. Este ahorro de tiempo no solo se reflejaría en la fase de detección de daños, sino también en la planificación y ejecución de acciones correctivas posteriores.

En (He, Girshick, y Dollár, 2018) se explora el entrenamiento de distintos modelos de detección de objetos con diferentes cantidades de datos de entrenamiento. En el estudio se muestra que ciertos modelos entrenados desde pesos preentrenados, pero con significativamente menos datos (10 % del tamaño de COCO), aún pueden rendir de manera comparable a los modelos entrenados con COCO. Esto sugiere que, entrenar modelos con al menos 10 % del tamaño del *dataset* de COCO (aproximadamente 10000 imágenes) puede lograr resultados buenos. Esto puede verse reflejado en los valores de testeo luego de juntar *val* y *train*, sugiriendo que una mayor cantidad de datos de entrenamiento puede repercutir positivamente en él desempeñó del modelo.

A su vez, para obtener el mejor rendimiento con YOLO, el propio Ultralytics recomienda utilizar un conjunto de datos grande y bien etiquetado con al menos 1500 imágenes por clase y 10000 instancias etiquetadas (Ultralytics, 2024). Es importante que estas imágenes sean variadas y representen el entorno real. Las etiquetas deben ser consistentes, precisas y verificadas. Además, es buena práctica agregar hasta un 10 % de imágenes de fondo para reducir los falsos positivos.

Capítulo 5

Conclusiones y Trabajo Futuro

El proyecto se enfocó en la automatización de la detección de daños superficiales en las palas de los aerogeneradores, utilizando modelos de aprendizaje automático aplicados a imágenes. Este es un tema de gran relevancia en la industria de la energía eólica y ha generado un importante esfuerzo de investigación a nivel global, con especial énfasis en China. Sin embargo, aún quedan desafíos por resolver en este campo, incluyendo la variabilidad y falta de transparencia en los conjuntos de datos utilizados en diferentes estudios y la implementación práctica de estas soluciones en la industria. Es crucial continuar investigando para avanzar en la detección de daños en palas de aerogeneradores y hacer que estas soluciones sean más accesibles y efectivas para la industria. Este proyecto buscaba no solo mejorar la eficacia en la detección de fallos, sino también colaborar con la industria nacional, al proporcionar sugerencias de posibles daños durante las inspecciones. De esta manera, se esperaba reducir la carga de trabajo de los inspectores, quienes actualmente deben revisar manualmente cada imagen en busca de anomalías.

Los resultados experimentales obtenidos a partir del entrenamiento realizado sobre DS.UY sugieren que, aunque los modelos de detección de objetos pueden ser valiosos como herramientas de asistencia para los inspectores, no pueden asumir completamente la responsabilidad de la detección de daños. Aun así, considerando que los niveles de las métricas alcanzadas al momento ($\text{mAP}_{0.5} = 0.495$, $Recall_{0.5:0.95} = 0.484$) fueron valorados positivamente y respaldados por la opinión de técnicos expertos en el área, se entiende que el modelo de reconocimiento desarrollado es valioso para realizar la identificación inicial de fallas.

Además, se observó que el tamaño del *backbone* de los modelos no siempre se correlaciona directamente con una mejora significativa en la capacidad de detección de daños. Si bien un *backbone* más grande puede ser útil para la detección de una amplia variedad de objetos, se encontró que para el caso específico de la detección de daños en palas de aerogeneradores, que se reduce a unas pocas clases, un *backbone* más simple puede ser igualmente efectivo. Por otro lado, se dio la misma situación con el aumento de *batch size*. Por lo tanto, antes de utilizar extractores de características muy profundos o tamaños de lote muy grandes, se recomienda tratar de destinar los recursos de memoria a la definición con la que se procesan las imágenes.

En cuanto a las técnicas de aumento de datos, se confirmó que estas son esenciales para mejorar la variabilidad de los conjuntos de datos y permitir la generalización de los modelos. Sin embargo, determinar la configuración óptima de las aumentaciones no es una tarea trivial, y se encontró que las aumentaciones geométricas o afines, como las rotaciones y las transformaciones de escala, fueron las más efectivas en este contexto. No obstante, se advierte que estas aumentaciones deben aplicarse con precaución, ya que durante las experimentaciones se observó que las rotaciones que no son múltiplo de 90° pueden distorsionar las anotaciones y reducir la precisión del modelo.

Además, se notó una diferencia significativa en el rendimiento del modelo para diferentes umbrales de IoU ($\text{mAP}_{0.5}$ y $\text{mAP}_{0.5:0.95}$). Esto sugiere que el modelo tiene dificultades para localizar con precisión los daños, especialmente en casos donde las instancias de daño están muy próximas o superpuestas, como en el caso de los daños LEE y TE ejemplificados en la figura 4.10. Asimismo, se encontró que algunas anotaciones de daños, como las correspondientes a LR, eran demasiado holgadas, lo que también afectaba la precisión de la detección, como se puede ver en la figura 4.9. El refinamiento de estas anotaciones fue fundamental para obtener mejores resultados como se vio en la subsección 4.5.1.

Mirando en perspectiva se identificaron varias direcciones de trabajo a futuro. Se sugiere ampliar y mejorar

las anotaciones en el conjunto de datos con el objetivo de estandarizar la localización de los daños, especialmente en casos complicados como TE y LEE. Además, se propone explorar el uso del modelo YOLOv8-obb, que es compatible con anotaciones de cuadros delimitadores orientados. Estas *oriented bounding boxes* (OBBs) ofrecen la ventaja de mantener la eficiencia al agregar tan solo un escalar adicional a la representación de las cajas delimitadoras. Esto hace que el modelo sea más eficiente en términos de recursos computacionales en comparación con el modelo de segmentación y además podría contribuir positivamente al problema de la localización. Esto se debe a que el modelo permite anotaciones más precisas y las mismas no se verían degradadas por rotaciones que no sean múltiplos 90 o 180 grados, como se dio en la experimentación. Esta orientación de cuadros sería valiosa, sobre todo con el conjunto de datos DS.UY el cual tiene muchas anotaciones con *bounding boxes* alargadas. También durante el desarrollo del proyecto, se exploró la posibilidad de utilizar la biblioteca Raytune para ajustar los hiperparámetros de los modelos (Liaw y cols., 2022). Sin embargo, este enfoque se vio limitado debido al consumo excesivo de memoria VRAM, incluso al intentar optimizar tan solo dos hiperparámetros simultáneamente. Esta fue la razón por la que se optó por un ajuste manual de los mismos. A futuro sería interesante reconsiderar esta herramienta, dada su aparente utilidad y potencial para encontrar una combinación de hiperparámetros óptimos. Una posible forma de mitigar el problema de la VRAM es con el uso de múltiples GPU en paralelo. Esta estrategia no solo aceleraría el tiempo de entrenamiento, sino que también posibilitaría aumentar el *batch size* y otros hiperparámetros que estuvieron restringidos previamente, mejorando así la escalabilidad y, muy posiblemente, el desempeño del modelo. Sin embargo, debido al tiempo que requeriría implementar esta solución, y dada la falta de experiencia, se decidió no llevarla a cabo.

Otras posibles líneas de investigación incluyen continuar la exploración de modelos como RetinaNet, que mostraron un buen rendimiento inicial y aún no se han probado con el conjunto de datos de DS.UY, así como investigar el uso de modelos de segmentación como Mask-RCNN o inclusive el modelo YOLOv8-seg para mejorar la precisión y versatilidad en la detección de daños, especialmente en términos de localización precisa.

Para expandir aún más nuestra investigación en el futuro, una sugerencia es la implementación de tracking e inferencia en línea para mejorar la confianza del modelo en tiempo real. Esto implica obtener más imágenes con daños sobre las cuales el modelo tenga "dudas". Actualmente, esta tesis se centra en la detección de objetos en imágenes de alta definición previamente capturadas. Sin embargo, en el futuro, se podrían realizar inferencias en tiempo real utilizando drones, permitiendo la captura de secuencias de video en lugar de imágenes estáticas. Con esta capacidad, los drones podrían seguir y acercarse a las áreas identificadas como potencialmente dañadas, capturando imágenes más detalladas y diversas para mejorar la precisión y confiabilidad del modelo de detección de objetos. Esta extensión proporcionaría una visión más completa y actualizada de las condiciones de las palas de los aerogeneradores, contribuyendo así a una gestión más eficiente y proactiva del mantenimiento. Actualmente, el *framework* de YOLO cuenta con la posibilidad de realizar seguimiento (tracking), lo cual sería recomendable explorar.

A su vez, estas mismas tecnologías y metodologías, podría ser usadas no solo para la detección de daños en palas de aerogeneradores, como es el caso de esta tesis, sino que puede aplicarse a una variedad de problemas. Desde la localización de aerogeneradores en imágenes áreas, como la detección de daños en otras superficies (líneas de alta tensión, por ejemplo), e incluso para realizar conteos. En este último punto, a modo de ejemplo, se podría detectar la posición de las palas y contar los giros que completa en un molino. Esta última aplicación podría utilizarse para estimar la eficiencia de un molino en términos de generación de energía eléctrica. Si dos molinos completan la misma cantidad de vueltas, pero uno genera más energía que el otro, esto podría indicar que el primero es más óptimo en términos de generación de energía, o que el segundo podría tener alguna falla interna que le lleva a producir menos energía.

Referencias

Aird, J. A., Barthelmie, R. J., y Pryor, S. C. (2023). Automated quantification of wind turbine blade leading edge erosion from field images. *Energies*, *16*(6). Descargado de https://www.mdpi.com/1996-1073/16/6/2820 doi: 10.3390/en16062820

Analyticsvidhya. (2020). *Foto de analyticsvidhya.* Descargado 2024-02-06, de https://cdn.analyticsvidhya.com/wp-content/uploads/2020/07/graphic4.jpg

ANII. (2024). *Anii.* Descargado 2024-02-09, de https://www.anii.org.uy/

Bahaghighat, M., Xin, Q., Motamedi, S. A., Zanjireh, M. M., y Vacavant, A. (2020). Estimation of wind turbine angular velocity remotely found on video mining and convolutional neural network. *Applied Sciences*, *10*(10). Descargado de https://www.mdpi.com/2076-3417/10/10/3544 doi: 10.3390/app10103544

Benavides, F. (2024). *Proyectos de investigación de facundo benavídez.* Descargado 2024-02-06, de https://www.fing.edu.uy/~fbenavid/

Bolya, D., Zhou, C., Xiao, F., y Lee, Y. J. (2019, October). Yolact: Real-time instance segmentation. En *Proceedings of the ieee/cvf international conference on computer vision (iccv).*

Buslaev, A., Iglovikov, V. I., Khvedchenya, E., Parinov, A., Druzhinin, M., y Kalinin, A. A. (2020). Albumentations: Fast and flexible image augmentations. *Information*, *11*(2). Descargado de https://www.mdpi.com/2078-2489/11/2/125 doi: 10.3390/info11020125

Canetti, R. (2024). *Proyectos de investigación de rafael canetti.* Descargado 2024-02-06, de https://www.researchgate.net/profile/Rafael-Canetti

Carbajal, G. (2024). *Proyectos de investigación de guillermo carbajal.* Descargado 2024-02-06, de https://www.researchgate.net/profile/Guillermo-Carbajal-2

Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., y Zagoruyko, S. (2020). End-to-end object detection with transformers.

ClusterUY. (2024). *Centro nacional de supercomputación.* Descargado 2024-02-06, de https://cluster.uy/

Crous, M., Intelligentie, B. O. K., y Visser, A. (2018). Combining weakly and strongly supervised segmentation methods for wind turbine damage annotation. *Computer Science*.

Decreto no. 403/009. (2009). http://www.ursea.gub.uy/web/mnormativo2.nsf/EFE20024A58C326E8325794C003FE146/$file/N%C2%BA%20403-009.pdf?OpenElement. (Accessed: 2009-01-01)

DTU. (2024). *Universidad técnica de dinamarca.* Descargado 2024-02-06, de https://www.dtu.dk/english/about

Du, Y., Zhou, S., Jing, X., Peng, Y., Wu, H., y Kwok, N. (2020). Damage detection techniques for wind turbine blades: A review. *Mechanical Systems and Signal Processing*, *141*, 106445. Descargado de https://www.sciencedirect.com/science/article/pii/S0888327019306661 doi: https://doi.org/10.1016/j.ymssp.2019.106445

Dwyer, B. (2022). *Rotations.* Descargado 2024-02-20, de https://blog.roboflow.com/content/images/size/w1000/2022/07/pencil-rotate.jpg

Energy5. (2023). *Un futuro más verde: Análisis comparativo de la energía eólica y los combustibles fósiles.* Descargado 2024-02-06, de https://energy5.com/es/un-futuro-ms-verde-anlisis-comparativo-de-la-energa-elica-y-los-combustibles-fsiles

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., y Zisserman, A. (2010). The pascal visual

object classes (voc) challenge. *International journal of computer vision*, *88*, 303–338. Descargado de https://web.archive.org/web/*/http://host.robots.ox.ac.uk/pascal/VOC/

Feng, Z.-H., Kittler, J., Awais, M., Huber, P., y Wu, X.-J. (2018, 06). Wing loss for robust facial landmark localisation with convolutional neural networks.. doi: 10.1109/CVPR.2018.00238

Foster, A., Best, O., Gianni, M., Khan, A., Collins, K., y Sharma, S. (2022). Drone footage wind turbine surface damage detection. En *2022 ieee 14th image, video, and multidimensional signal processing workshop (ivmsp)* (p. 1-5). doi: 10.1109/IVMSP54334.2022.9816220

Fu, M., Wang, Q., Wang, J., Sun, L., Ma, Z., Zhang, C., ... Li, W. (2023). Deep cnn-based materials location and recognition for industrial multi-crane visual sorting system in 5g network.

Garcia, F. (2018). *ClasificaciÓn de fallas en rodamientos de mÁquinas rotativas utilizando aprendizaje de mÁquinas* (Tesis Doctoral). doi: 10.13140/RG.2.2.33614.41284

Géron, A. (2022). *Hands-on machine learning with scikit-learn, keras, and tensorflow, 3rd edition* (3rd ed.). O'Reilly Media, Inc. (ISBN: 9781098125974)

Girshick, R., Radosavovic, I., Gkioxari, G., Dollár, P., y He, K. (2019). *Detectron2: Open-source object detection and segmentation.* Descargado de https://github.com/facebookresearch/detectron2

Girshick, R. B. (2015). Fast R-CNN. *CoRR*, *abs/1504.08083*. Descargado de http://arxiv.org/abs/1504.08083

Girshick, R. B., Donahue, J., Darrell, T., y Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. En *Proceedings of the ieee conference on computer vision and pattern recognition (cvpr)*.

Google scholar. (2024). https://scholar.google.com/.

Gu, J., Liu, G., y Li, M. (2022). Damage detection for rotating blades using digital image correlation with an ac-surf matching algorithm. *Sensors*, *22*(21). Descargado de https://www.mdpi.com/1424-8220/22/21/8110 doi: 10.3390/s22218110

Guissous, A. E. (2019, 11). *Skin lesion classification using deep neural network.*

Guo, J., Liu, C., Cao, J., y Jiang, D. (2021). Damage identification of wind turbine blades with deep convolutional neural networks. *Renewable Energy*, *174*, 122-133. Descargado de https://www.sciencedirect.com/science/article/pii/S0960148121005589 doi: https://doi.org/10.1016/j.renene.2021.04.040

He, K., Girshick, R., y Dollár, P. (2018). *Rethinking imagenet pre-training.*

He, K., Girshick, R. B., y Dollár, P. (2018). Rethinking imagenet pre-training. *CoRR*, *abs/1811.08883*. Descargado de http://arxiv.org/abs/1811.08883

He, K., Gkioxari, G., Dollár, P., y Girshick, R. (2017). Mask r-cnn. En *2017 ieee international conference on computer vision (iccv)* (p. 2980-2988). doi: 10.1109/ICCV.2017.322

He, K., Zhang, X., Ren, S., y Sun, J. (2015). *Deep residual learning for image recognition.*

He, K., Zhang, X., Ren, S., y Sun, J. (2016). Deep residual learning for image recognition. En *2016 ieee conference on computer vision and pattern recognition (cvpr)* (p. 770-778). doi: 10.1109/CVPR.2016.90

IEA. (2023). *Latin america energy outlook 2023.* Descargado de https://www.iea.org/reports/latin-america-energy-outlook-2023 (License: CC BY 4.0)

ImageNet. (2009). *ImageNet.* http://www.image-net.org.

Ioffe, S., y Szegedy, C. (2015). *Batch normalization: Accelerating deep network training by reducing internal covariate shift.*

Iyer, A., Nguyen, L. Q., y Khushu, S. (2022). Learning to identify cracks on wind turbine blade surfaces using drone-based inspection images. *ArXiv*, *abs/2207.11186*. doi: arXiv:2207.11186

Jocher, G. (2020, mayo). *YOLOv5 by Ultralytics.* Descargado de https://github.com/ultralytics/yolov5 doi: 10.5281/zenodo.3908559

Jocher, G., Chaurasia, A., y Qiu, J. (2023, enero). *YOLO by Ultralytics.* Descargado de https://github.com/ultralytics/ultralytics

Jung, A. B., Wada, K., Crall, J., Tanaka, S., Graving, J., Reinders, C., ... others (2020). *imgaug.* https://github.com/aleju/imgaug. (Online; accessed 01-Feb-2020)

Kabir, S., Aslansefat, K., Gope, P., Campean, F., y Papadopoulos, Y. (2022). Online dynamic reliability evaluation of wind turbines based on drone-assisted monitoring. *arXiv preprint arXiv:2211.13258*.

KaKoYu007. (2022). *Wind-turbine-blade.* https://github.com/KaKoYu007/Wind-turbine-blade.

Krizhevsky, A., Sutskever, I., y Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. En F. Pereira, C. Burges, L. Bottou, y K. Weinberger (Eds.), *Advances in neural information*

processing systems (Vol. 25). Curran Associates, Inc. Descargado de https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Liaw, R., Nishihara, R., ... Liang, E. (2022). *Ray tune.* Descargado de https://docs.ray.io/ (Accessed: February 21, 2024)

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., y Belongie, S. (2017). Feature pyramid networks for object detection. En *2017 ieee conference on computer vision and pattern recognition (cvpr)* (p. 936-944). doi: 10.1109/CVPR.2017.106

Lin, T.-Y., Goyal, P., Girshick, R., He, K., y Dollár, P. (2018). *Focal loss for dense object detection.*

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... Zitnick, C. L. (2014). *Microsoft coco: Common objects in context.* http://cocodataset.org/. Descargado de https://arxiv.org/pdf/1405.0312.pdf

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C., y Berg, A. C. (2015). SSD: single shot multibox detector. *CoRR*, *abs/1512.02325*. Descargado de http://arxiv.org/abs/1512.02325

Lv, L., Yao, Z., Wang, E., Ren, X., Pang, R., Wang, H., ... Wu, H. (2022). Efficient and accurate damage detector for wind turbine blade images. *IEEE Access*, *10*, 123378-123386. doi: 10.1109/ACCESS.2022.3224446

Moreno, S., Peña, M., Toledo, A., Treviño, R., y Ponce, H. (2018). A new vision-based method using deep learning for damage inspection in wind turbine blades. En *2018 15th international conference on electrical engineering, computing science and automatic control (cce)* (p. 1-5). doi: 10.1109/ICEEE.2018.8533924

Nguyen, L., Iyer, A., y Khushu, S. (2022). An automated system for detecting visual damages of wind turbine blades. *arXiv preprint arXiv:2205.10954*.

Nikolov, I., Nielsen, M., Garnæs, J., y Madsen, C. (2020). *Wind turbine blade surfaces dataset.* Mendeley Data. doi: 10.17632/jrmm82m4mv.2

Okoli, C. (2015, noviembre). A Guide to Conducting a Standalone Systematic Literature Review. *Communications of the Association for Information Systems*, *37*. Descargado de https://hal.science/hal-01574600

Patel, J., Sharma, L., y Dhiman, H. S. (2021). Wind turbine blade surface damage detection based on aerial imagery and vgg16-rcnn framework. *arXiv preprint arXiv:2108.08636*.

Peng, L., y Liu, J. (2018). Detection and analysis of large-scale wt blade surface cracks based on uav-taken images. *IET Image Processing*, *12*(11), 2059–2064.

Pexels. (2017). *Foto de white windmill.* Descargado 2024-02-06, de https://www.pexels.com/photo/agriculture-alternative-energy-clouds-countryside-414837/

Pizarro Muñoz, J. S. (2020). Wind turbine blade damage identification using deep learning algorithms.

Pyimagesearch. (2016). *Segmentación.* Descargado 2024-02-06, de https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

Raipur, I. N. (2023, oct). *Crack_segmentation dataset* [Open Source Dataset]. https://universe.roboflow.com/iiit-naya-raipur-ajytz/crack_segmentation-kxa7r. Roboflow. Descargado de https://universe.roboflow.com/iiit-naya-raipur-ajytz/crack_segmentation-kxa7r (visited on 2024-02-18)

Reddy, A., Indragandhi, V., Ravi, L., y Subramaniyaswamy, V. (2019). Detection of cracks and damage in wind turbine blades using artificial intelligence-based image analytics. *Measurement*, *147*, 106823. Descargado de https://www.sciencedirect.com/science/article/pii/S0263224119306803 doi: https://doi.org/10.1016/j.measurement.2019.07.051

Redmon, J., Divvala, S., Girshick, R., y Farhadi, A. (2016). *You only look once: Unified, real-time object detection.*

Ren, S., He, K., Girshick, R., y Sun, J. (2016). *Faster r-cnn: Towards real-time object detection with region proposal networks.*

Rezatofighi, S. H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I. D., y Savarese, S. (2019). Generalized intersection over union: A metric and A loss for bounding box regression. *CoRR*, *abs/1902.09630*. Descargado de http://arxiv.org/abs/1902.09630

Sarkar, D., y Gunturi, S. K. (2021). Wind turbine blade structural state evaluation by hybrid object detector relying on deep learning models. *Journal of Ambient Intelligence and Humanized Computing*, *12*, 8535–8548.

Sazli, M. (2006, 01). A brief review of feed-forward neural networks. *Communications Faculty Of Science University of Ankara*, *50*, 11-17. doi: 10.1501/commua1-2_0000000026

SETScholars. (2023). *Understanding overfitting in machine learning: A comprehensive guide.* Descargado 2024-02-06, de https://setscholars.net/understanding-overfitting-in-machine-learning-a-comprehensive-guide/

Shihavuddin, A., Chen, X., Fedorov, V., Nymark Christensen, A., Andre Brogaard Riis, N., Branner, K., ... Reinhold Paulsen, R. (2019). Wind turbine surface damage detection by deep learning aided drone inspection analysis. *Energies*, *12*(4). Descargado de https://www.mdpi.com/1996-1073/12/4/676 doi: 10.3390/en12040676

Shihavuddin, A., Rashid, M. R. A., Maruf, M. H., Hasan, M. A., ul Haq, M. A., Ashique, R. H., y Mansur, A. A. (2021). Image based surface damage detection of renewable energy installations using a unified deep learning approach. *Energy Reports*, *7*, 4566-4576. Descargado de https://www.sciencedirect.com/science/article/pii/S2352484721005102 doi: https://doi.org/10.1016/j.egyr.2021.07.045

Siddhardhan. (2022). *6.5. overfitting in machine learning — causes for overfitting and its prevention.* Descargado 2024-02-06, de https://www.youtube.com/watch?v=gy8kXdd6K-o

Song, X., Xing, Z., Jia, Y., Song, X., Cai, C., Zhang, Y., ... Li, Q. (2022). Review on the damage and fault diagnosis of wind turbine blades in the germination stage. *Energies*, *15*(20). Descargado de https://www.mdpi.com/1996-1073/15/20/7492 doi: 10.3390/en15207492

Szeliski, R. (2022). *Computer vision: Algorithms and applications* (2nd ed.). Springer.

Tan, M., Pang, R., y Le, Q. V. (2020). Efficientdet: Scalable and efficient object detection. *CoRR*, *abs/1911.09070*. Descargado de http://arxiv.org/abs/1911.09070

Timbó. (2024). *Timbó.* Descargado 2024-02-06, de https://foco.timbo.org.uy/home

Ultralytics. (2023). *Segmentación.* Descargado 2024-02-06, de https://docs.ultralytics.com/es/tasks/segment/

Ultralytics. (2024). *Tips for best training results.* Descargado de https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/

UruguayXXI. (2019). *Uruguay lider en energías renovables.* Descargado 2024-02-06, de https://www.uruguayxxi.gub.uy/en/news/article/uruguay-lider-en-energias-renovables/

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., andŁukasz Kaiser, A. N. G., y Polosukhin, I. (2017). *Attention is all you need.* Descargado de https://arxiv.org/pdf/1706.03762.pdf

Verri, W. (2021). *Cómo uruguay se convirtió en el segundo país del mundo en energía eólica.* Descargado 2024-02-06, de https://www.walterverri.com.uy/como-uruguay-se-convirtio-en-el-segundo-pais-del-mundo-en-energia-eolica/

Wang, L., y Zhang, Z. (2017). Automatic detection of wind turbine blade surface cracks based on uav-taken images. *IEEE Transactions on Industrial Electronics*, *64*(9), 7293-7303. doi: 10.1109/TIE.2017.2682037

Wang, L., Zhang, Z., Xu, J., y Liu, R. (2018). Wind turbine blade breakage monitoring with deep autoencoders. *IEEE Transactions on Smart Grid*, *9*(4), 2824-2833. doi: 10.1109/TSG.2016.2621135

woctezuma. (2021). *Finetune detr.* https://github.com/woctezuma/finetune-detr.git. GitHub.

Wolniak, R., y Skotnicka-Zasadzień, B. (2023). Development of wind energy in eu countries as an alternative resource to fossil fuels in the years 2016-2022. *Resources*, *12*(8). Descargado de https://www.mdpi.com/2079-9276/12/8/96 doi: 10.3390/resources12080096

Yang, P., Dong, C., Zhao, X., y Chen, X. (2020). The surface damage identifications of wind turbine blades based on resnet50 algorithm. En *2020 39th chinese control conference (ccc)* (p. 6340-6344). doi: 10.23919/CCC50068.2020.9189408

Yu, Y., Cao, H., Liu, S., Yang, S., y Bai, R. (2017). Image-based damage recognition of wind turbine blades. En *2017 2nd international conference on advanced robotics and mechatronics (icarm)* (p. 161-166). doi: 10.1109/ICARM.2017.8273153

Zhang, C., Yang, T., y Yang, J. (2022). Image recognition of wind turbine blade defects using attention-based mobilenetv1-yolov4 and transfer learning. *Sensors*, *22*(16). Descargado de https://www.mdpi.com/1424-8220/22/16/6009 doi: 10.3390/s22166009

Zhang, J., Cosma, G., y Watkins, J. (2021). Image enhanced mask r-cnn: A deep learning pipeline with new evaluation measures for wind turbine blade defect detection and classification. *Journal of Imaging*, *7*(3). Descargado de https://www.mdpi.com/2313-433X/7/3/46 doi: 10.3390/jimaging7030046

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R., y Ren, D. (2019). *Distance-iou loss: Faster and better learning for bounding box regression.*

Zheng, Z., Wang, P., Ren, D., Liu, W., Ye, R., Hu, Q., y Zuo, W. (2021). *Enhancing geometric factors in model learning and inference for object detection and instance segmentation.*

Zhu, Y., Liu, X., Li, S., Wan, Y., y Cai, Q. (2022). Wind turbine blade defect detection based on acoustic features and small sample size. *Machines*, *10*(12). Descargado de https://www.mdpi.com/2075-1702/10/12/1184 doi: 10.3390/machines10121184

Zou, L., y Cheng, H. (2022). Research on wind turbine blade surface damage identification based on improved convolution neural network. *Applied Sciences*, *12*(18). Descargado de https://www.mdpi.com/2076-3417/12/18/9338 doi: 10.3390/app12189338

Zou, L., Wang, Y., Bi, J., y Sun, Y. (2022). Damage detection in wind turbine blades based on an improved broad learning system model. *Applied Sciences*, *12*(10). Descargado de https://www.mdpi.com/2076-3417/12/10/5164 doi: 10.3390/app12105164

zylo117. (2020). *Yet another efficientdet pytorch.* https://github.com/zylo117/Yet-Another-EfficientDet-Pytorch.git. GitHub.