Trabajo Fin de Grado
Grado de Ingeniería Electrónica, Robótica y Mecatrónica

Reconocimiento de objetos mediante técnicas de visión por computador y aprendizaje automático

Autor: Manuel Martín Romero
Tutor: Ramón González Carvajal
Rubén Martín Clemente

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Reconocimiento de objetos mediante técnicas de visión por computador y aprendizaje automático

Autor:
Manuel Martín Romero

Tutor:
Ramón González Carvajal
Catedrático
Rubén Martín Clemente
Profesor titular

Dpto. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2022

Trabajo Fin de Grado: Reconocimiento de objetos mediante técnicas de visión por computador y aprendizaje automático

Autor: Manuel Martín Romero

Tutor: Ramón González Carvajal

Rubén Martín Clemente

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2022

El Secretario del Tribunal

Agradecimientos

A mis padres por el sacrificio que realizaron y la confianza que depositaron en mí; y a mi hermana Carmen por escucharme cuando no tenía a quién acudir.

A todos y cada uno de los profesores y docentes que me guiaron y acompañaron durante estos 6 años.

A mis tutores Ramón González Carvajal y Rubén Martín Clemente por su paciencia y la oportunidad que me han brindado.

Manuel Martín Romero

Sevilla, 2022

Resumen

Con la llegada de la denominada Industria 4.0 comienza a haber una demanda por sistemas de visión e inferencia basados en inteligencias artificiales. Gracias a avances en el campo del Deep Learning es posible crear y entrenar modelos de redes neuronales que sean capaces de reconocer en tiempo real la posición o trayectoria de objetos, algo de gran utilidad en la logística de muchas industrias ya que permite obtener datos de interés como por ejemplo el tiempo de empleo de infraestructuras, la capacidad de trabajo de éstas o un historial del movimiento de mercancías, entre muchos otros.

Si bien ya existen diversos algoritmos de tratamiento de imágenes para detección de objetos, su implementación junto a estas redes posibilita la creación de programas de reconocimiento de objetos muy flexibles y generalizados que pueden implementarse a distintas industrias.

En este trabajo en concreto se hará, por una parte, empleo de visión artificial para detección de barcos cargueros y grúas en el Puerto de Sevilla; y, por otra parte, se hará una demostración del empleo de algoritmos y técnicas de tratamiento de imágenes para la detección de movimiento en una fábrica de virolas para aerogeneradores.

Abstract

The Fourth Industrial Revolution has created a demand for the integration of Artificial Intelligence systems. Thanks to advances in the field of Deep Learning, it is possible to train neural network able to recognize the position or trajectory of objects in real time. This has great use in the logistics of many industries since it allows to obtain data of interest such as the time of use of the infrastructures, their working capacity or a registry of the movement of goods, among many others.

Although there are already various image processing algorithms for object detection, their implementation together with these neural networks is what allows the creation of flexible and generalized object recognition programs that can be implemented in different industries.

In this work, on the one hand, computer vision will be used to detect cargoships and cranes from a camera at the Port of Seville; and, on the other hand, there will be shown a demonstration of the use of algorithms for image processing techniques for the detection of movement in a ferrule factory.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

Los avances en computación y semiconductores de las últimas décadas han facilitado el desarrollo e implementación del aprendizaje profundo o Deep Learning. Aunque sus bases ya fueron teorizadas en la época de los 60, es ahora cuando comienzan a ser posible numerosas aplicaciones prácticas relacionadas con este campo: conducción autónoma, reconocimiento facial, procesamiento del lenguaje escrito, reconocimiento de voz, deepfake, entre muchas otras.

Figura 2–1. Muelle automatizado en el puerto de Qingdao, China. [1]

Junto al Internet of Things y el Big Data, se espera que el Deep Learning sea uno de los pilares fundamentales de las próximas tecnologías que revolucionen la industria en el futuro cercano.

1.1 Motivación y objetivos

Dado que el contenido de las materias impartidas en el grado de Ingeniería Electrónica, Robótica y Mecatrónica no indagan con demasiada profundidad en el tema de la Inteligencia Artificial, el interés de realizar este trabajo reside en tener una primera toma de contacto con unos conceptos tan amplios y llamativos como el Machine Learning, la Visión artificial y el Deep Learning.

Más allá de intentar ahondar y tratar de explicar con rigor los conceptos nombrados anteriormente, el objetivo principal de este trabajo será aprender a utilizar herramientas para entrenar modelos de redes neuronales que respondan de manera acorde a lo deseado, y el empleo de técnicas de tratamiento digital de imágenes para realizar diversas funcionalidades.

Por ello, por un lado; se hará énfasis en el entreno de redes neuronales y se creará un software que implemente una red neuronal para identificar barcos cargueros y grúas portuarias; y, por otro lado, se creará un software que tratará de determinar si una grúa móvil que transporta láminas de metal está en movimiento, esto último haciendo empleo de algoritmos y tratamiento digital de imágenes exclusivamente.

1.2 Organización

En este documento se irán exponiendo progresivamente los conocimientos que permiten concluir con los objetivos anunciados en el apartado anterior, por ello la memoria se organizará de la siguiente forma:

- **Capítulo 1: Introducción.** Exposición de los motivos por los que se quiere realizar este trabajo y los objetivos que se desean alcanzar.
- **Capítulo 2: Estado del arte.** Sobre cómo se puede trabajar digitalmente con una imagen y el estado de la Inteligencia Artificial en la actualidad.
- **Capitulo 3: Deep learning para tratamiento de imágenes.** Explicación de qué son las redes neuronales artificiales convolucionales, y como crear y entrenar una para la detección de objetos en imágenes.
- **Capítulo 4: Técnicas de tratamiento de imágenes.** Exposición de diversos tipos de técnicas y métodos para manipular imágenes que se emplean es este trabajo, y una explicación básica de sus fundamentos.
- **Capítulo 5: Método propuesto.** Exposición del plan de trabajo que se llevará a cabo para alcanzar los objetivos marcados.
- **Capítulo 6: Conclusiones y líneas futuras.** Valoración general del trabajo realizado, analizando sus defectos y posibles puntos de mejora. Proposición de pasos a seguir para corregir dichos defectos e implementar las mejoras.

2 ESTADO DEL ARTE

Se expondrá a continuación una introducción sobre cómo se genera una imagen digital a partir de la luz que incide en una cámara, y el estado actual de la inteligencia artificial y algunas aplicaciones recientes.

2.1 Imagen digital como matriz

Es relevate conocer cómo se puede trabajar con imágenes en la computación digital, pues es lo que se hará en apartados posteriores.

2.1.1 Obtención de la imagen digital

El proceso de obtención de una imagen o fotografía digital comienza con la captación de la luz del objeto que se desea fotografiar. Esta luz es filtrada y procesada (usualmente por lentes y filtros polarizados) antes de llegar a lo que se conoce como un sensor CCD (Charged-Coupled device, cada vez más anticuado) o un sensor CMOS (más usados actualmente), un array de fotoreceptores que se encarga de transformar la luz incidente en tensiones eléctricas.

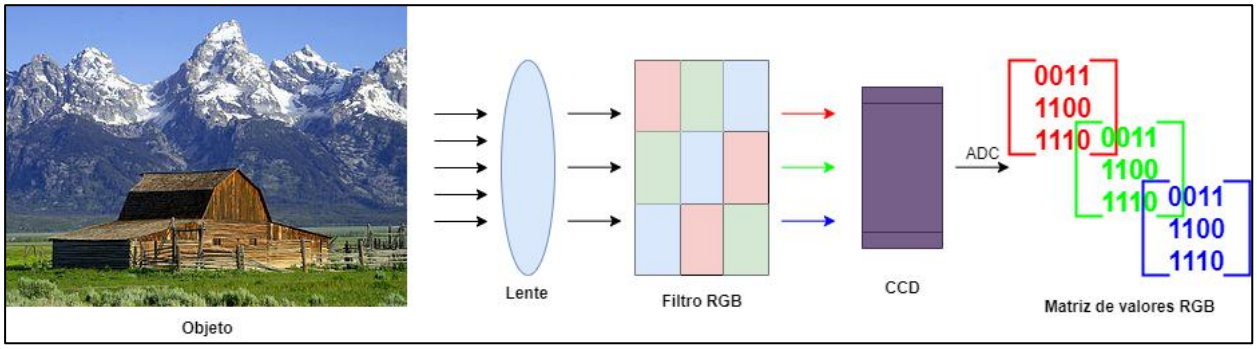


Figura 3–1. Proceso general del funcionamiento de una cámara digital.

Las tensiones generadas en los receptores del sensor se transforman en valores digitales mediante un conversor A/D; posteriormente estos valores son ordenados en una matriz y almacenados en un archivo con formato de imagen. Lo interesante de esto es que permite que se puedan aplicar algoritmos y funciones matemáticas a dichas matrices, lo que permite trabajar con imágenes en computación digital.

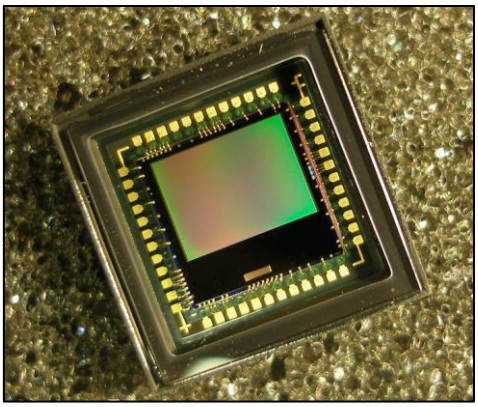


Figura 2–2. Sensor CMOS expuesto. [2]

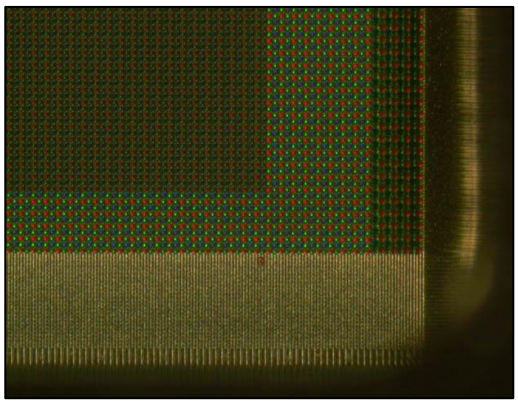


Figura 2–3. Vista de un sensor CMOS en microscopio.

2.1.2 Formación del archivo de imagen

Antes de continuar hay que entender primero qué es un píxel. Un píxel, de picture element, suele conocerse como el elemento más pequeño e indivisible que forma una imagen digital. Puesto que el número de fotoreceptores en el sensor de imagen es finito, también lo será el número de píxeles que componen la imagen digital resultante.

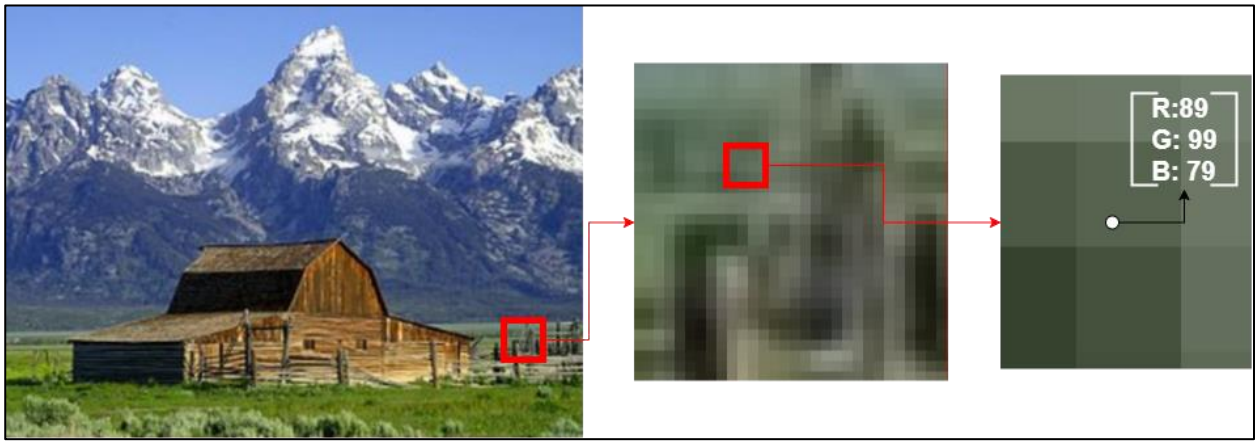


Figura 2–4. Imagen digital ampliada para distinguir sus píxeles.

En un modelo de color RGB el píxel tiene asociado un valor de luminosidad referente al color rojo, verde y azul. Estos valores suelen tener un rango de entre 0 y 255 (referente a que cada color se caracteriza con 8 bits), y es común representar dichos valores en notación hexadecimal en la forma #RRGGBB. Por ejemplo, el negro vendría representado como *#000000*, el blanco como *#FFFFFF*, y el tono grisáceo del píxel marcado en la figura anterior como *#59634F*. Existen otros modelos de color, como el CMYK o el HSV; o el RGBA, que añade un canal extra para caracterizar la transparencia u opacidad del píxel, aunque esto no es relevante por el momento.

Un término interesante relacionado con el píxel es el que se conoce como resolución de imagen, este viene a indicar el número total de píxeles que compone una imagen. A mayor resolución, mayor calidad suele poseer dicha imagen. También se aplica a las pantallas o los propios sensores de imagen de las cámaras, como el número de píxeles que puede mostrar o generar, respectivamente.

El conjunto de bits de colores de cada píxel obtenidos en la fotografía se almacena en un archivo de imagen. Según la manera en la que se desee, o se puedan, almacenar dichos valores hay que atender a lo que se conoce como el formato del archivo. El formato del archivo es una manera estandarizada de organizar y almacenar los datos de la imagen en dicho archivo.

En la actualidad hay una amplia cantidad de formatos de archivos de imagen, como .PNG, .JPEG, .GIF, .BMP, entre muchos otros. Factores a tener en cuenta a la hora de elegir un formato u otro pueden ser la compresibilidad: si se desea que el archivo comprima la información para que ocupe menos memoria en disco a coste de sacrificar fidelidad de imagen, la profundidad de color: número de bits que puede emplear el píxel para definir su color, a mayor profundidad (más bits) mayor el número de colores disponibles; o la transparencia: hay formatos que no permiten trabajar con un canal para la transparencia del pixel mientras que otros sí.

Figura 2–5. Comparación de una misma imagen en formatos JPEG y PNG.

2.2 Inteligencia artificial en la actualidad

Un estudio que lleva a cabo la Universidad de Stanford conocido como 'One Hundred Year Study on IA' [3], o AI100, se encarga de seguir el progreso que están teniendo las tecnologías basadas en inteligencia artificial en nuestro mundo. En el ultimo reporte realizado en el año 2022, sacado de la pagina web aiindex.org, destaca:

· Alrededor de 20.000 publicaciones científicas sobre IA realizadas entre los años 2010 y 2019; siendo la categoría más popular Machine Learning, seguida de Computer Vision y Procesamiento del lenguaje natural. China y Estados Unidos aportan la mayor cantidad de publicaciones en colaboración cross-country sobre IA, más del doble que las publicadas por China y Reino Unido, el segundo par más alto en el ranking.

· La inversión privada en IA en el año 2021 ascendió a la suma de más de 93.5 mil millones de dólares, más del doble que en el año 2020.

· Desde el año 2018 se ha reducido el coste de entrenar sistemas de clasificación de imágenes en un 63.6%, y el tiempo de entrenamiento en un 94.4%. Por otra parte, el precio de hardware como brazos robóticos se ha visto reducido un 46.2% en los últimos 5 años, de 42.000$ en 2017 a 22.600$ en 2021.

· Siendo IA la especialidad más popular en Computer Science; el número de estudiantes que ingresan a grados relacionados con Inteligencia Artificial se ha multiplicado por 16 con respecto al año 2010.

· La tasa de error en detección de objetos (demostrado en LSVRC, the Large-Scale Visual Recognition Challenge) se ha reducido de un 28% en 2010 hasta un 2% en 2017, superando el rendimiento humano.

· En 2019, softwares basados en IA han igualado o mejorado el desempeño humano en juegos como ajedrez, Go, poker, Pac-Man, Quake III, Dota 2, StarCraft II, entre muchos otros.

· En medicina, sistemas basados en IA son capaces de diagnosticar de manera temprana enfermedades como cáncer de piel, cáncer de prostata o retinopatía diabética con mayor eficacia que un doctor humano. También destaca el uso del Deep Learning para resolver el plegado de estructuras proteicas complejas.

También destacan otros avances significativos que ya han tenido lugar:

Conducción autónoma: destacan los vehículos Tesla, como el Tesla Modelo 3 lanzado en 2017, o los prototipos de vehículos de la empresa Waymo que ofrecen servicio de taxi robótico desde 2018 en la ciudad de San Francisco. También destacan drones con capacidad de vuelo autónoma para el reparto de medicamentos en Ruanda, iniciado en 2016. Otros ejemplos de interés son drones y quadcopters capaces de realizar mapeados 3D de interiores de edificios, u organizarse en formación junto con otros drones autónomos para realizar diversas acrobacias.

Movimiento articulado: Spot, el robot de cuatro patas amarillo desarollado por Boston Dynamics con capacidad para adaptar su movilidad al terreno, salió a la venta en el año 2020. También resalta el Atlas, un robot humanoide capaz de caminar, correr, saltar e incluso hacer volteretas.

Reconocimiento de voz: Microsoft demostró en 2017 que su Sistema de Reconocimiento de Conversación alcanzó una tasa de error de palabras del 5.1%, igualando el rendimiento humano en Switchboard task, un experimento en el que se trata de transcribir conversaciones telefónicas. Alrededor de un tercio de las interacciones con ordenadores se realiza ya mediante comandos de voz en lugar de teclado; Alexa, Siri, Cortana y Google son ejemplos de IAs capaces de responder preguntas y realizar tareas comandadas por voz.

3 INTRODUCCIÓN AL DEEP LEARNING

En este capitulo se explicará el funcionamiento básico de una red neuronal y la relación que estas poseen con el Deep learning, así como qué puede entenderse por inteligencia artificial y cómo engloba al machine learning.

3.1 Inteligencia Artificial, Machine Learning y Deep Learning

Inteligencia artificial, machine learning y deep learning son tres conceptos que a menudo son confundidos o aplicados de manera incorrecta. Antes de continuar es necesario diferenciarlos y explicar qué abarca cada uno de ellos.

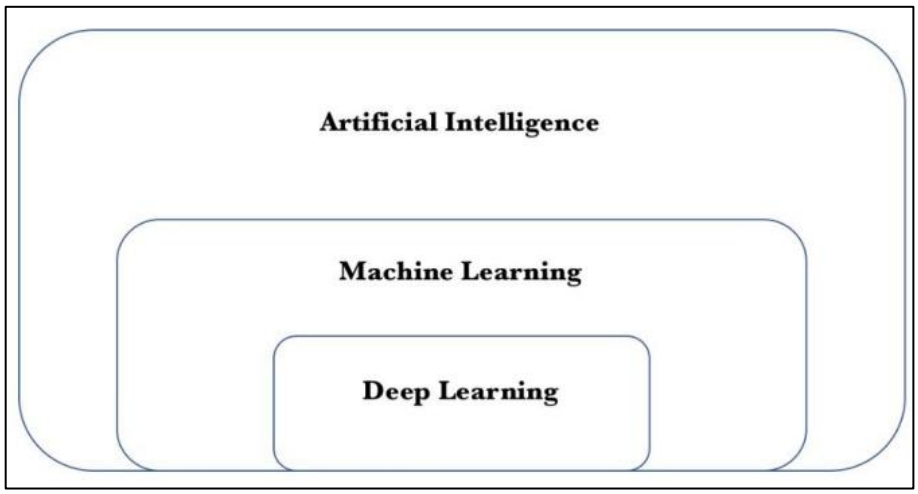


Fig 3–1. Inteligencia Artificial engloba el Machine Learning, y éste al Deep learning.

Inteligencia artificial:

Una definición precisa y detallada pueden darla autores como Stuart Rusell y Peter Norvig en *Artificial Intelligence, a modern aproach* [4]. Pero la inteligencia artificial, IA o AI en inglés, puede definirse de manera muy breve como la inteligencia que muestran las máquinas en contraste con la inteligencia natural humana. En un sentido más amplio, también puede incluirse al esfuerzo de intentar automatizar tareas que requieren cierto grado de abstracción que los humanos realizan naturalmente, como por ejemplo el reconocimiento facial o el entendimiento del lenguaje. Es un campo de estudio científico muy extenso que ocupa de otros campos del conocimiento como la matemática, la computación o incluso la neurociencia.

Machine learning:

El machine learning, traducido al castellano como aprendizaje automatizado, es un subcampo o rama de la inteligencia artificial dedicado a estudiar técnicas para hacer que máquinas o computadoras aprendan por sí mismas a realizar una tarea. Por lo general, lo que se introduce en la máquina son unos datos de los que se espera que la propia máquina encuentre los patrones que necesita identificar para ser capaz de identificar esos mismos patrones, o similares, en otros elementos.

También es un amplio campo de estudio. Existe un elevado número de técnicas que abordan el machine learning, pero se pueden agrupar en tres categorias principales:

· **Aprendizaje supervisado:** los datos utilizados para el entrenamiento incluyen la solución o el patrón que la maquina debe aprender a reconocer de manera explícita; se dicen que portan la etiqueta buscada. En esta categoría se encuentran las redes neuronales, el algoritmo de interés de este trabajo.

· **Aprendizaje no supervisado:** se le entrega a la máquina unos datos sin etiquetas y se espera ella misma sea capaz de aprender a clasificar la información.

· **Aprendizaje reforzado:** se presenta a la máquina un entorno desconocido y ella misma debe aprender qué acción realizar a través de la prueba y error, habiendo un mecanismo que recompense o penalice las conductas que se acerquen o alejen de la solución deseada. Este método de aprendizaje se suele emplear en combinación con otros tipos de entrenamiento, ya puede simular muchos escenarios del mundo real.

Deep learning:

Aunque se profundizará más sobre ellas posteriormente, para entender el concepto del deep learning (aprendizaje profundo en inglés) primero hay que entender qué son las redes neuronales. Las redes neuronales son un caso del machine learning que consiste en crear un modelo de varias capas de neuronas, las cuales realizan una transformación a unos datos de entrada y los entregan como una salida. El término deep (profundidad en inglés) hace referencia a la profundidad o número de estas capas de neuronas.

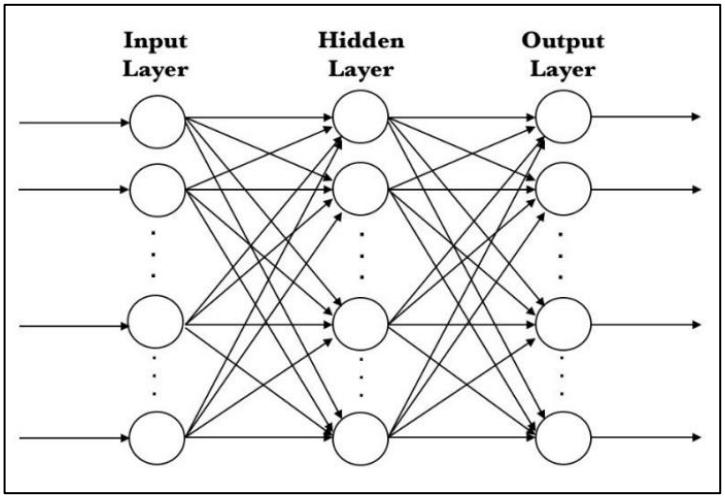


Figura 3–2. Representación gráfica de una red neuronal. [5]

Programar algoritmos para Deep learning no es algo trivial; variables como el número de capas y neuronas, funciones de activación o la forma en que se entrenan, pueden afectar significativamente el resultado del modelo. Aunque existe mucha literatura sobre el Deep Learning y tiene años de investigación, saber ajustar de manera precisa esas variables para obtener el resultado deseado requiere experiencia y técnica.

3.2 Redes neuronales artificiales

Las redes neuronales artificiales, abreviadas como ANNs (del inglés Artificial Neuronal Nertworks), engloban un conjunto de procesos o algoritmos en Machine Learning y son el epicentro del Deep Learning.

3.2.1 Perceptrón simple y multicapa

Una forma de comprender el funcionamiento de una sola neurona en una red neuronal es a partir del perceptrón simple, un algoritmo de aprendizaje supervisado de clasificación binaria. Este será capaz de determinar si un elemento de entrada, representado como un vector de números reales, pertenece, o no, a un grupo o clase.

Consiste en el sumatorio de unos valores de entradas, respresentados por un vector x_i, ponderados según otros valores denominados Pesos sinápticos, w_i. El resultado de esta operación es sumado a su vez a otro valor denominado bias, antes de ser computado por una función de activación.

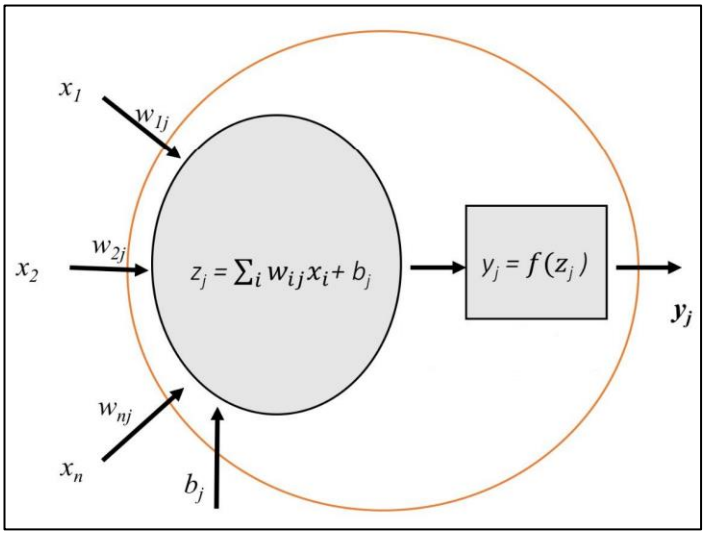


Figura 3–3. Esquema del perceptrón simple.

La forma en la que el perceptrón es capaz de aprender es mediante un proceso iterativo en el que se compara una predicción estimada con la predicción esperada para una entrada dada. En cada iteración se ajustan los valores de los pesos w_i y los bias b de forma que el error cometido vaya decrementando, con la intención de reducirlo tanto como sea posible.

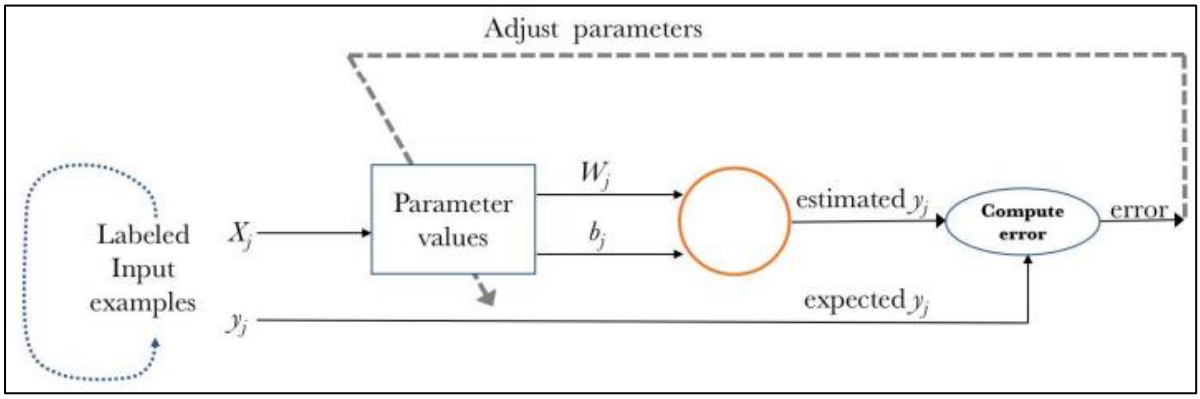


Figura 3–4. Diagrama del proceso de aprendizaje de un perceptrón.

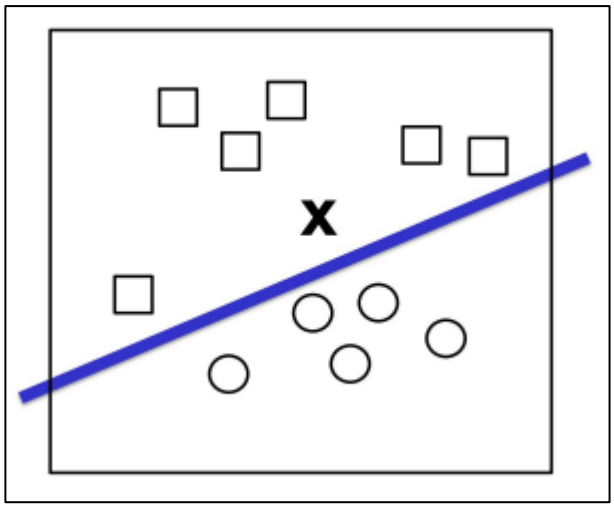


Figura 3–5. Diagrama de funcionamiento de un perceptrón dibujando la separación fronteriza entre dos tipos de objetos (cuadrados y círculos).

La tarea básica que realiza un perceptrón simple es identificar una característica entre dos grupos de datos para clasificar nuevos datos de entrada. En la Figura 3–5 se muestra un espacio 2D donde el perceptrón ha delimitado dos regiones mediante una recta, perteneciendo una de ellas a un grupo de círculos y otra a un grupo de cuadrados. Para un nuevo punto X = (x_1, x_2) el perceptrón será capaz de determinar a que grupo pertenece según su posición relativa con la recta frontera que ha estimado (la recta azul).

Esta recta frontera se puede representar como:

$$y = W'X + b \tag{3–1}$$

siendo W el vector de pesos w_i, X el vector de entradas x_i y b el bias u offset.

Aunque para muchas aplicaciones el perceptrón simple es suficiente, existen problemas que requieren mayor capacidad de abstracción como el reconocimiento facial o la detección de caracteres alfanúmericos, entre muchos otros. Para desempeñar dichas tareas se hace uso de lo que se conoce como el perceptrón multicapa, que será capaz de abstraer características más complejas como rasgos de una cara o la morfología de letras escritas.

Por lo general, el perceptrón multicapa y la gran mayoría de ANNs se compone de una capa de entrada de una o más neuronas, a la que se introducen los datos con los que se va a trabajar. Ésta es seguida de otras capas, llamadas capas ocultas, que pueden tener diferentes números de neuronas. Por último, otra capa de salida sigue tras las capas ocultas, y es la capa de donde se sacarán los resultados de las transformaciones que la red ha realizado sobre los datos de entrada.

En cada capa, cada neurona funciona como un perceptrón simple, siendo su entrada el resultado de las neuronas de la capa anterior. Cuando todas o la mayoría de las neuronas de una capa están conectadas a las neuronas de capas anteriores o posteriores, se dice que es una capa densamente conectada.

3.2.2 Funciones de activación

Para propagar la salida de una neurona hacia capas posteriores se hace uso de las denominadas funciones de activación. Las funciones de activación sirven para introducir no-linealidades en el modelo de la red. Algunas de las funciones de activación más usadas son:

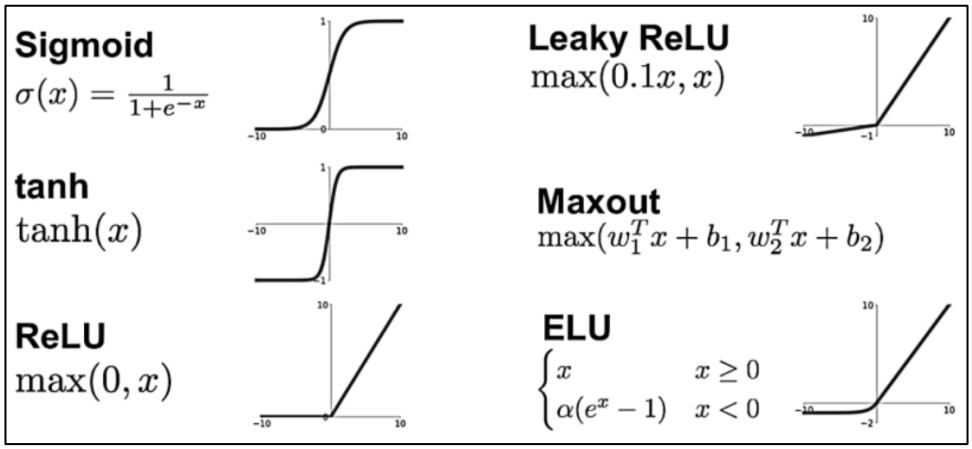


Figura 3–6. Funciones de activación más comunes en machine learning. [6]

No todas las capas de la red tienen que usar la misma función de activación, existen arquitecturas en las que es conveniente que la capa de entrada o salida posea una función de activación no-lineal (sigmoide, tangente hiperbólica, etc) frente a una función lineal a tramos para las capas ocultas (por ejemplo, ReLU).

Dependiendo del tipo problema que se aborde, algunas funciones de activación serán más apropiadas que otras. Por ejemplo, la función sigmoide puede ser útil para predicción de probabilidades; mientras que la función ReLU es muy usada en las capas ocultas de redes convolucionales (tratamiento de imágenes).

Una función de activación muy empleada en problemas de clasificación excluyente es la conocida como función softmax [7]. Esta función se emplea en la capa de salida de la red, interpretándose la salida de sus neuronas como las probabilidades correspondientes a que un input dado pertenezca a dichas clases. Por ejemplo, en una red para reconocimiento de dígitos, las clases serían [0 1 2 3 4 5 6 7 8 9]. Para un input de una imagen del número 3, el output de la red podría ser [0.004 0.006 0.004 0.956 0.003 0.005 0.003 0.004 0.009 0.006]. Esto quiere decir que la probabilidad de que el input entregado se corresponda al número 3 es casi la unidad, mientras que la probabilidad de que se corresponda a cualquier otro número es casi nula. También destaca que la suma de todas las probabilidades es 1.

3.3 Redes neuronales convolucionales

Las redes neuronales convolucionales (CNNs o ConvNets) se trata de un tipo concreto de red neuronal en Deep Learning que ya eran presentes en la década de los 90, pero que han ganado protagonismo en el campo de computer vision en los últimos años debido al buen desempeño que poseen para tareas de reconocimiento de imágenes.

Una diferencia principal respecto a las redes neuronales artificiales tratadas en el apartado anterior es que en las redes convolucionales se asume que las entradas o inputs serán imágenes, luego su estructura estará orientada a tratar con matrices de datos.

La forma en la que una ConvNet es capaz de identificar formas como caras, animales u objetos, es incrementando el nivel de abstracción que la red es capaz de realizar en cada capa, pasando por patrones y características de más simples a más complejos.

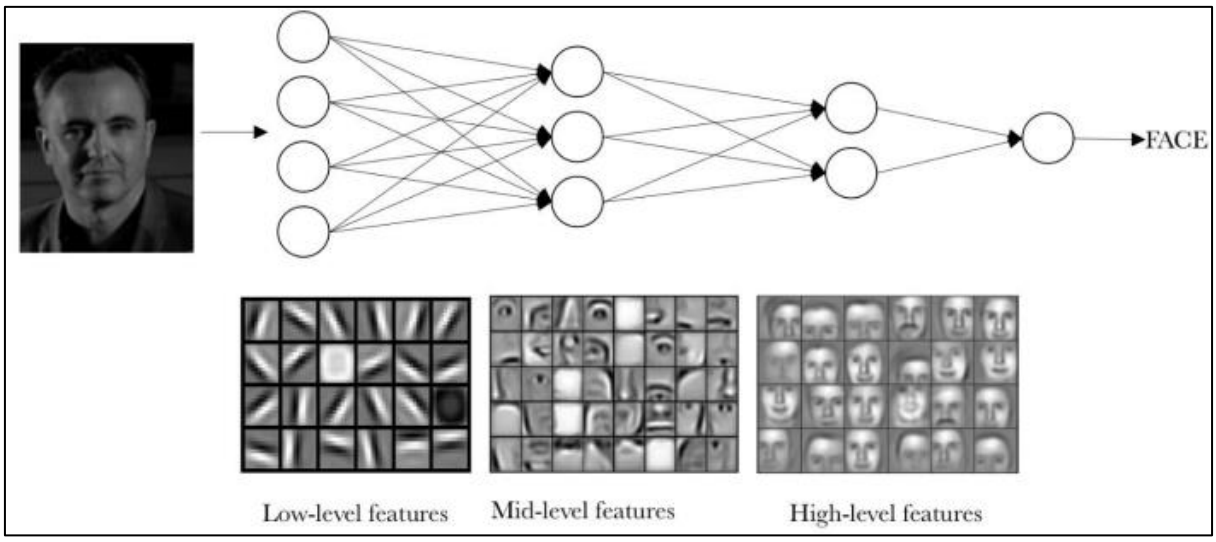


Figura 3–7. Visualización intuitiva de cómo funciona una red neuronal convolucional.

Sin embargo, las redes convolucionales no pueden implementarse de la misma forma que una red neuronal artificial de capas densamente conectadas. Para una entrada de una imagen de tamaño reducido de, por ejemplo, 150x150 píxeles, x3 canales RGB, se da que las neuronas en la capa de entrada deberían trabajar con 67500 parámetros, una cifra demasiado elevada que conllevaría un alto coste computacional. Para abordar este problema se introducen dos operaciones: convolución y pooling.

3.3.1 Convolución

La principal diferencia entre una capa densamente conectada y una capa convolucional (una capa a la que se aplica la operación de convolución), es que la capa densamente conectada aprende patrones globales para todo su input, mientras que la capa convolucional aprende patrones locales en pequeñas ventanas del input total.

La capa convolucional es capaz de detectar características como bordes, líneas, gradientes de color, entre otros; pudiendo extrapolar la detección de esas características a otros puntos cualquiera de la imagen. Otro rasgo importante de las capas convolucionales es que puede aprender jerarquías de patrones preservando relaciones espaciales. De esta forma, las primeras capas aprenden patrones simples como bordes, mientras que capas posteriores aprenden patrones más complejos compuestos por patrones y características aprendidas en capas anteriores; de esta forma, la red convolucional es capaz de aprender a reconocer conceptos de mayor complejidad y abstracción.

Para entender la operación de la convolución se va a exponer el siguiente ejemplo: una imagen de dimensiones 28x28 en escala de grises se introduce como input a una capa. La siguiente capa tras la capa de entrada estará conectada convolucionalmente, esto quiere decir, las neuronas de esta capa oculta tendrán como entrada una ventana de 5x5 píxeles de la capa anterior. Esta ventana de 5x5 va deslizándose de izquierda a derecha y de arriba hacia abajo recorriendo la imagen completa. Por cada posición de la ventana hay una neurona de la siguiente capa oculta que se encargará de procesar la información que le llegue de ese subconjunto de 5x5 píxeles.

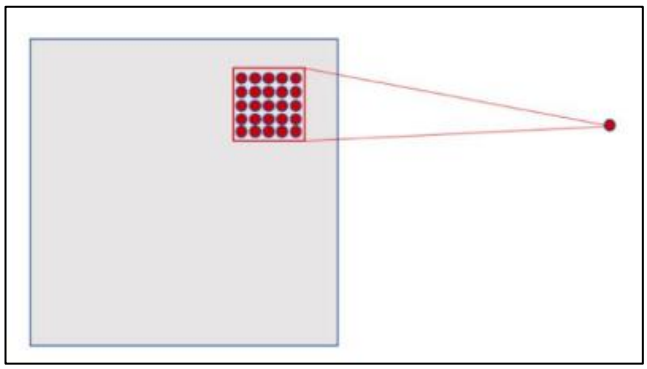


Figura 3–8. Proceso de convolución.

Usando una ventana 5x5 que se mueve sólo una posición en cada desplazamiento, en una capa de 28x28 píxeles, se obtiene una capa oculta de 24x24 neuronas. El tamaño de la ventana o el número de posiciones que se mueve en cada desplazamiento (llamado stride) son parámetros que se pueden configurar, dando lugar a capas convolucionales más reducidas incluso.

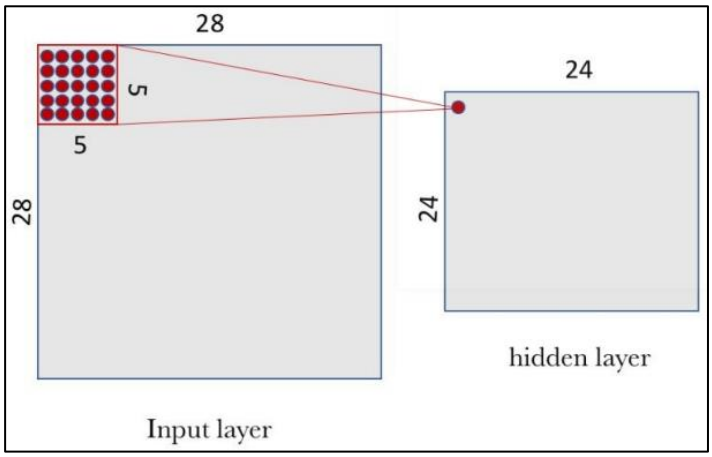


Figura 3–9. Resultado del proceso de convolución.

3.3.2 Pooling

Aparte de las capas convolucionales, las redes neuronales convolucionales acompañan esas capas con capas de pooling. Las capas de pooling, descrito de forma muy simple, simplifican y condensan la información que se les introduce como entrada. Esto permite reducir el número de neuronas y parámetros necesarios en capas posteriores, simplificando el modelo de la red neuronal.

Para realizar la operación de pooling a una capa se define el tamaño una ventana, o kernel, y se divide dicha capa en porciones del tamaño de la ventana. Cada porción se puede condensar de diversas maneras, una de las más comunes se conoce como Max Pooling, que toma la máxima intensidad del píxel o valor de neurona en dicha porción. También existe el Min Pooling, análogo a lo anterior, pero tomando la mínima intensidad, o Average Pooling, que hace una media ponderada de todas las intensidades de la porción.

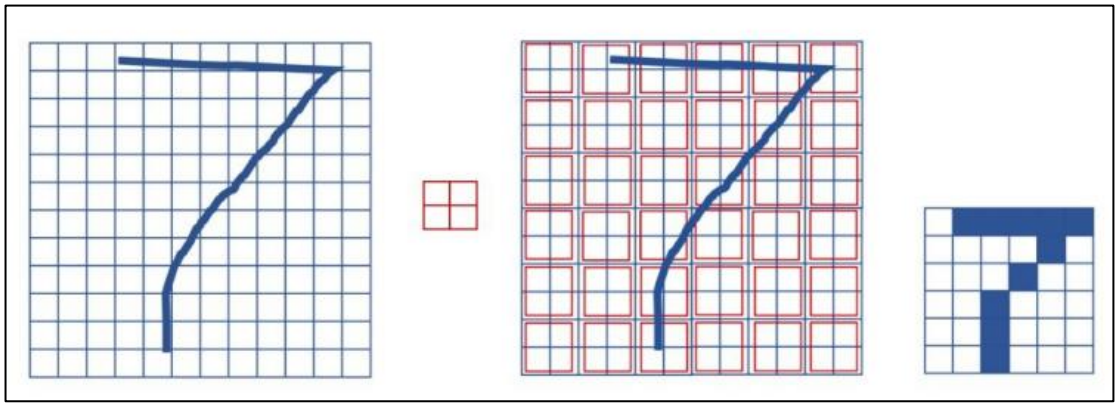


Figura 3–10. Ejemplo de Max Pooling para un kernel 2x2.

Cuanto mayor sea la ventana o kernel empleado, más pequeña será la capa resultante, pudiéndose perder información relevante en el proceso. Sin embargo, si se escoge un tamaño adecuado, se puede reducir el número de neuronas y parámetros del modelo de la red neuronal sin que esto afecte a su desempeño, aliviando la carga computacional.

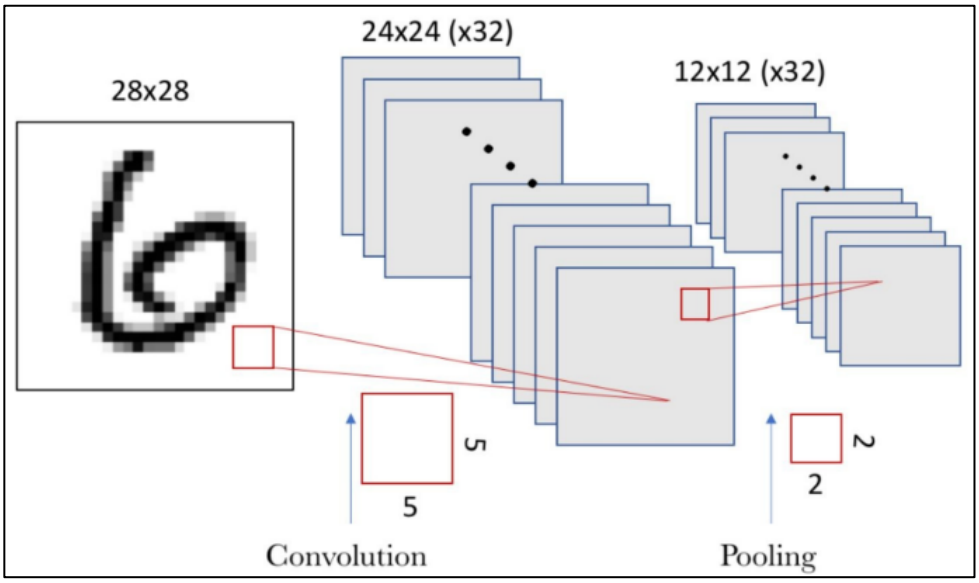


Figura 3–11. Disminución progresiva del tamaño de las capas como resultado de procesos de convolución y pooling.

Para el caso del ejemplo del apartado anterior; partiendo de una imagen de 28x28 píxeles se pasa a unas capas convolucionales de 24x24 neuronas tras una convolución usando una ventana de tamaño 5x5; y, posteriormente, aplicando un kernel de 2x2, se pasa a unas capas de pooling de 12x12. Este proceso puede repetirse varias veces, intercalando capas convolucionales y capas de pooling. Por último, la última capa convolucional o de pooling preceder a una capa densamente conectada, con una función de activación que se adecúe al problema que debe resolver el modelo de red (en el caso del ejemplo de reconocimiento de números, la última capa usaría la función softmax para clasificación).

3.4 Entrenamiento

El proceso de entrenamiento de una red neuronal consiste en alcanzar los pesos que hacen que la red, para una entrada dada, produzca una salida similar a la que se esperaría.

3.4.1 Datasets

Para que el entrenamiento sea efectivo hay que partir de una base de datos (o dataset) adecuada. En problemas de clasificación, el dataset deberá estar catalogado y etiquetado correctamente; además, es recomendable que haya equilibrio en el número de elementos pertenecientes a cada clase. Por lo general, también se sugiere que el dataset sea lo más grande posible, o de un tamaño considerable, ya que datasets pequeños pueden no ser lo suficientemente representativos.

Usualmente no se utiliza el dataset entero para la etapa de entrenamiento, sino que se guarda una parte (un 20% o 30%) exclusivamente para una etapa de evaluación, que sirve para evaluar qué tan bueno es el modelo de red entrenado: si se ha producido over-fitting o under-fitting, si puede ser interesante entrenar durante más épocas, etc.

Una práctica que se realiza para aumentar de manera artificial el tamaño de un dataset es realizando lo que se conoce como Data Augmentation [8]. Consiste en generar nuevos elementos introduciendo pequeñas variaciones en elementos del dataset ya existentes. En datasets de imágenes se suelen introducir rotaciones, translaciones, ruido, desenfoques, volteados o cambios de escala.

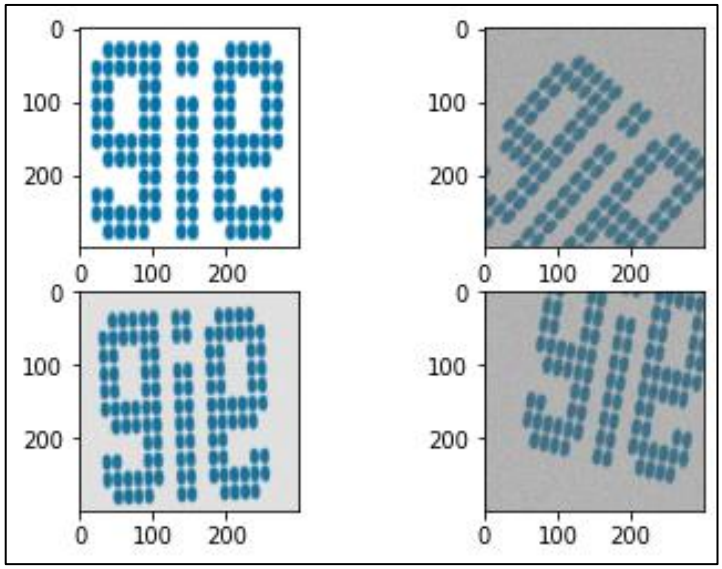


Figura 3–12. Ejemplo de data augmentation.

3.4.2 Hiperparámetros

En redes neuronales, los hiperparámetros son variables de configuración externas al modelo, especificadas por el programador. Éstos definen la topología de la red (número de capas y neuronas, tipo de capas, función de activación de las neuronas…) y el algoritmo de aprendizaje (learning rate, número de épocas, batch size…). Según el tipo de problema que se quiera abordar, estos hiperparámetros deberán haber sido escogidos acorde. En este apartado se explicarán los hiperparámetros relacionados al proceso de aprendizaje:

· **Batch size:** durante el proceso de entrenamiento, no suele ser posible entregar el dataset entero por falta de memoria RAM. En su lugar, el dataset se entrega en trozos más pequeños, o batches. Emplear un tamaño de batch pequeño suele reducir el tiempo de entrenamiento y requiere menos memoria, pero a cambio provoca que las estimaciones del gradiente de pérdida sean menos precisas.

· **Epochs:** referido al número de iteraciones del proceso de entrenamiento. Entrenar durante un gran número de épochs suele resultar en un modelo overfit; mientras que hacerlo durante un número muy pequeño de epochs provoca que el modelo no esté suficientemente entrenado.

· **Learning rate:** referido a cuánto cambian los pesos y sesgos en el proceso de entrenamiento, para tratar de reducer el valor de pérdida. Un paso muy pequeño puede provocar que al modelo le cueste muchas iteraciones para llegar a un mínimo global de pérdida; mientras que un paso muy elevado puede provocar nunca se alcance el mínimo.

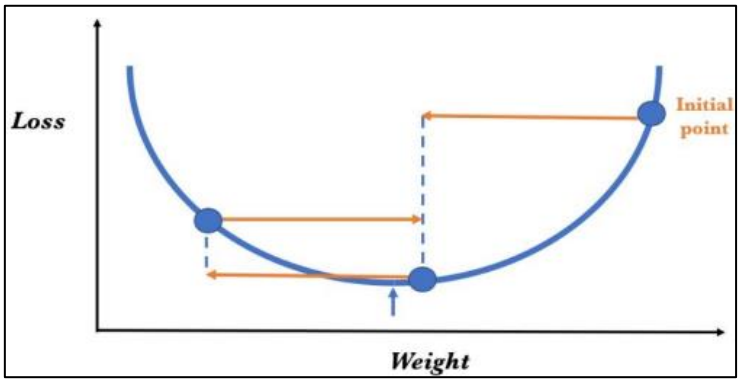


Figura 3–13. Optimizador gradient descent. Un learning rate demasiado grande impide que se alcance el mínimo.

3.4.3 Proceso de aprendizaje

El proceso de aprendizaje de una red neuronal es un proceso iterativo, en el que se recorren las capas hacia delante y hacia atrás, ajustando los pesos y sesgos de las neuronas en cada iteración.

Primero tiene lugar la fase de forward propagation en la que la red es expuesta al dataset de entrenamiento. El input entregado hace que las neuronas en cada capa realicen transformaciones a la información recibida de capas anteriores y la manden a capas posteriores. Cuando la información ha terminado de recorrer todas las capas, la capa de salida mostrará como resultado una predicción basada en la entrada dada.

Posteriormente se calcula la pérdida (loss) o error cometido comparando la predicción generada por la red con el resultado esperado. Idealmente, se desea que esta pérdida o error sea nulo o el mínimo valor posible. Los pesos y sesgos se van ajustando de manera gradual en función al valor de pérdida, hasta lograr una predicción correcta o suficientemente parecida a la esperada; esto se puede cuantificar mediante las métricas de precision (accuracy) durante el entrenamiento.

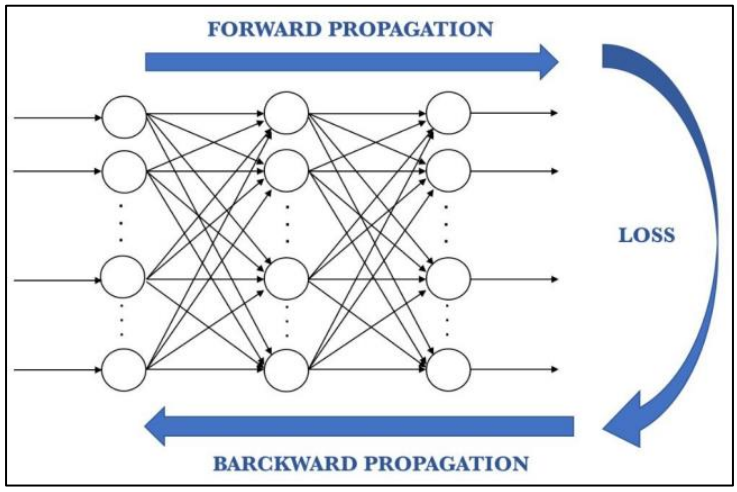


Figura 3–14. Proceso iterativo de aprendizaje.

Una vez calculado el error, esta información se transmite hacia atrás (backward propagation), empezando por la capa de salida. La información de pérdida se transmite a las neuronas en capas anteriores de manera relativa según su contribución a la red. El proceso se repite capa por capa, hasta llegar a la capa de entrada. Una vez que la información ha llegado a todas las neuronas, se ajustan sus pesos y sesgos empleando optimizadores como el gradient descent [9] (ver Fig. 3–13), buscando que la pérdida en cada neurona sea menor en la siguiente iteración.

3.4.4 Resultados del entrenamiento

El proceso de entrenamiento de la red como tal puede alargarse tanto como sea necesario o se desee. En función de los hiperparámetros establecidos y el dataset utilizado, el modelo entrenado puede resultar:

· **No convergente:** el modelo no sirve pues el proceso de entrenamiento no es capaz ajustar los parámetros de forma que minimicen la función de pérdida. Este caso puede darse si el learning rate es demasiado grande, o si el batch size es demasiado pequeño; también si la estructura de red que emplea no es apropiada para el problema que se pretende resolver.

· **Under-fitting:** el modelo es demasiado simple y no es capaz de realizar predicciones acertadas. Este caso suele ocurrir cuando la estructura de la red no posee suficientes capas o se ha entrenado durante muy pocas épocas.

· **Over-fitting:** opuesto al caso anterior, el modelo ha aprendido características muy específicas del dataset de entrenamiento, y no es capaz de generalizar correctamente para otros casos que no provengan de ese dataset. Esto ocurre cuando la arquitectura de la red es demasiado compleja, el dataset es muy reducido o se ha entrenado durante demasiadas épocas. Existen técnicas y métodos para evitar este caso, como dejar de entrenar el modelo cuando la pérdida de entrenamiento y validación empiezan a divergir (Early-stopping), ampliar el dataset de entrenamiento realizando data augmentation, o implementar técnicas como Regularización [10] o Dropout [11].

· **Good fit:** el modelo de red ha aprendido a reconocer las características del dataset de entrenamiento y es capaz de generalizar y producir buenas predicciones para otras entradas.

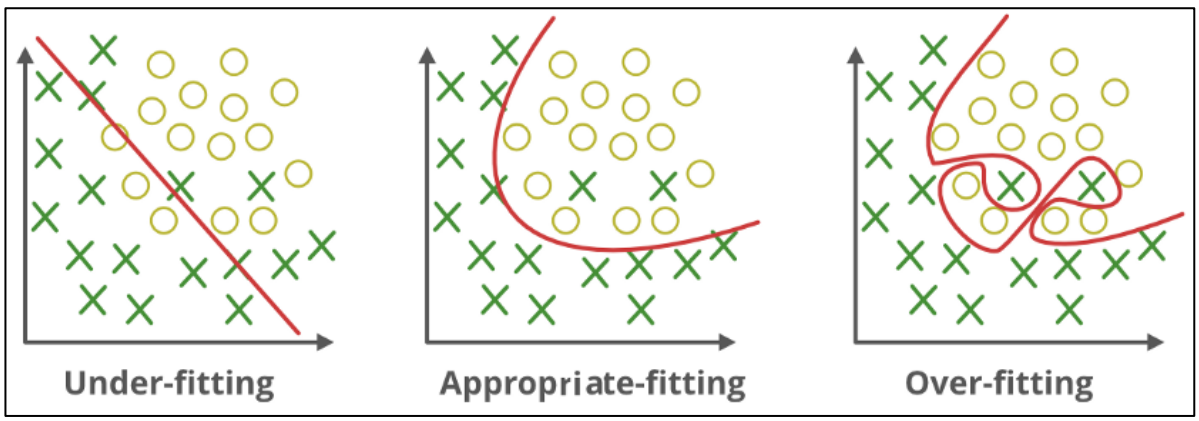


Figura 3–15. Representación de la capacidad de clasificación de un modelo según el resultado del entrenamiento.

4 TÉCNICAS DE TRATAMIENTO DE IMÁGENES

En este capitulo se introducirá la teoría de algunos algoritmos y técnicas para el tratamiento digital de imágenes que se emplearán en apartados posteriores.

En este trabajo se hará uso de la librería OpenCV (open computer vision) [12], que aporta herramientas y utilidades para la manipulación de imágenes y videos.

4.1 Modelo de colores HSV

El modelo de color HSV [13] (del inglés Matiz, Saturación y Valor) define un espacio de color alternativo al modelo RGB. Se puede representar visualmente como un cilindro:

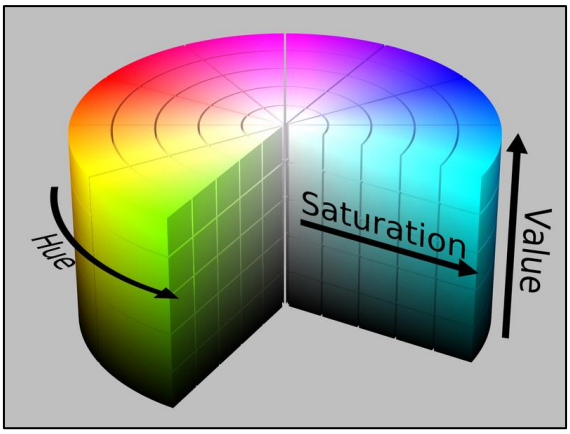


Figura 4-1. Mapa visual del modelo de colores HSV. [14]

La componente de Matiz o Hue representa una dimensión angular abarcando todo el círculo cromático. Los colores primarios rojo, verde y azul se encuentran en Hues de 0º, 120ª y 240ª respectivamente. El resto de los colores se alcanzan como una mezcla lineal de dichos colores primarios.

Las componentes de Saturación y Valor permiten obtener tonalidades de cualquier color mezclado con el negro y el blanco.

Una cualidad destacable del espacio de colores HSV es que permite definir rangos de tonalidades de un color cualquiera variando sólo el parámetro Matiz. Por ejemplo, si se deseara seleccionar los píxeles de una imagen cuyo color esté dentro de un rango de tonalidades amarillas, se puede definir dicho rango como una cuña del cilindro de la figura anterior manteniendo constantes los parámetros Saturación y Valor y variando sólo el parámetro Matiz. En un espacio de color RGB dicho rango de tonalidades amarillas tendría una forma más compleja, que tendría que ser definida variando los 3 parámetros del espacio RGB.

4.2 Conversión a escala de grises

En muchas ocasiones es preferible trabajar con imágenes en una escala de color monocromática (grises), ya que solo requiere un canal frente a los tres canales RGB de una imagen a color. Una forma de convertir imágenes a color a una escala de grises es ponderando los valores RGB de cada píxel, de forma que su suma determine la intensidad posterior de dicho píxel.

Por ejemplo, sea un píxel de tres canales RGB de 8 bits R: 255, G: 242, B: 0 (color amarillo), se pondera la intensidad del píxel en escala de grises de manera que:

$$I = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \qquad (4\text{–}1)$$

$$I = 0.299 \cdot 255 + 0.587 \cdot 242 + 0.114 \cdot 0 = 218$$

A una intensidad I máxima (255), el píxel sería de color blanco, mientras que a una intensidad I nula (0), el píxel sería de color negro. La escala de grises estaría comprendida entre todos los valores intermedios del rango 0-255. Es destacable que el color verde tiene mucho más peso que el rojo y azul. Aunque se puede realizar la ponderación con otras relaciones, en la mayoría de las librerías de tratamiento de imágenes el color dominante es el verde; esto se hace así debido a que el ojo humano es más sensible dicho color.

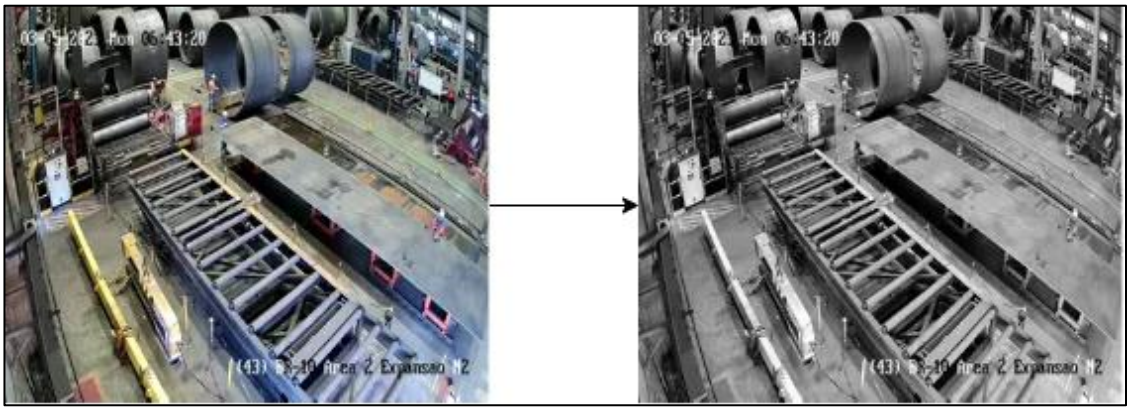


Figura 4–2. Conversión de imagen en espacio RGB a escala de grises.

4.3 Máscaras

Una forma de aislar zonas concretas de una imagen es mediante una máscara. En este trabajo las máscaras que se emplean son variables array imágenes donde la zona de interés se colorea de blanco y el resto de negro. Posteriormente, mediante una función llamada bitwise [15], la máscara se aplica a otra imagen o a un stream de video para aislar o eliminar las zonas resaltadas de la máscara.

Ejemplo: se desea aislar el rail de rodillos de la imagen. Para ello se crea una máscara con una zona coloreada de blanco que superpuesta con la imagen original coincida con la zona de interés.

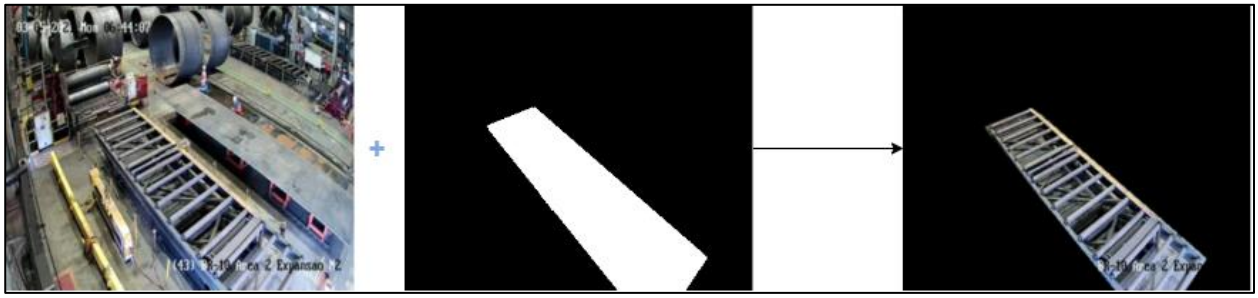


Figura 4–3. Demostración de aplicación de máscara.

En el caso de la figura anterior se emplea la función bitwise_and, donde el color negro equivale a un 0 binario y el blanco a un 1. Píxel por píxel realiza una comparación entre la imagen original y la máscara. En la zona negra el resultado será también un color negro mientras que en la zona blanca el resultado será el color del píxel de la imagen original.

4.4 Filtro gaussiano

Gaussian blur [16], filtrado o suavizado gaussiano en castellano, es un efecto que desenfoca la imagen

convolucionando el mapa de bits de dicha imagen con una función gaussiana bidimensional de la forma:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\pi\sigma^2}} \tag{4–2}$$

Donde x e y reprensentan la coordenada del píxel y σ la desviación estándar. A mayor desviación estándar mayor será el emborronamiento que provoca en la imagen. Dicho emborronamiento ayuda a reducir el detalle de la imagen, lo que facilita la posterior detección de los contornos.

Figura 4–4. Resultado de aplicar desenfoque gaussiano para una desviación estándar de 7 con OpenCV.

4.5 Detección de bordes mediante algoritmo de Canny

Desarrollado por John F. Canny en 1986, el algoritmo de Canny [17] es un proceso de varias etapas que permite detectar bordes y contornos de objetos en una imagen. De manera resumida, se puede dividir este proceso en tres pasos fundamentales: filtrado gaussiano para suavizar la imagen y reducir el ruido, calcular gradientes de intensidad de la imagen en varias direcciones y umbralizar el valor de dichos gradientes para descartar aquellos de poca intensidad o para determinar la dirección dominante del gradiente en un punto.

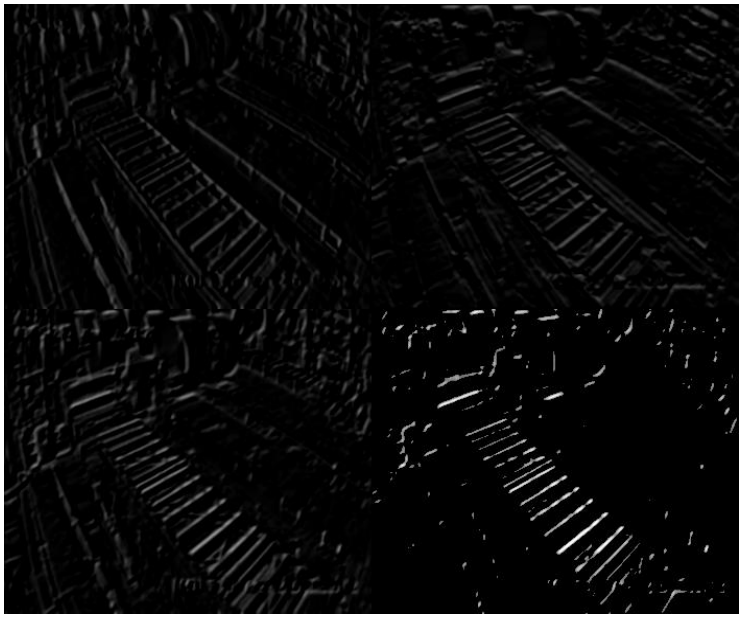


Figura 4–5. En sentido horario: (a) Gradiente horizontal. (b) Gradiente vertical. (c) Suma de gradientes (d) Umbral y amplificación.

En aplicaciones reales se realizan muchos otros pasos intermedios y posteriores que mejoran la precisión del algoritmo, pero el fundamento es similar. Parámetros de peso en el resultado final pueden ser la desviación estándar escogida en el filtro gaussiano y los valores de umbral.

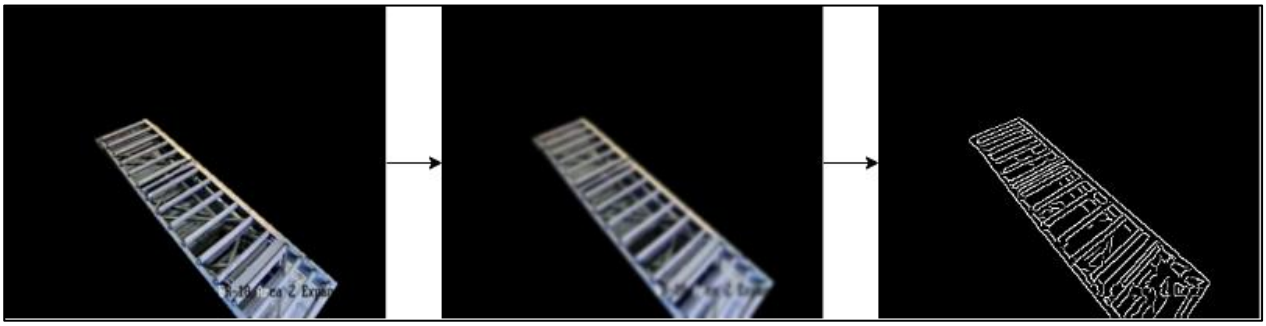


Figura 4–6. Resultado realizar filtrado gaussiano y detección de bordes mediante algoritmo de Canny con OpenCV.

4.6 Detección de rectas mediante transformada de Hough

La transformada de Hough [18] es una técnica que permite encontrar determinadas formas y figuras tales como líneas, círculos o elipses. En este trabajo se va a emplear sólo para detectar líneas rectas, que es el caso más sencillo de esta transformada.

Ya que en coordenadas cartesianas una recta vertical tendría pendiente infinita, lo que puede dificultar computar dicha recta, se utiliza coordenadas polares en su lugar:

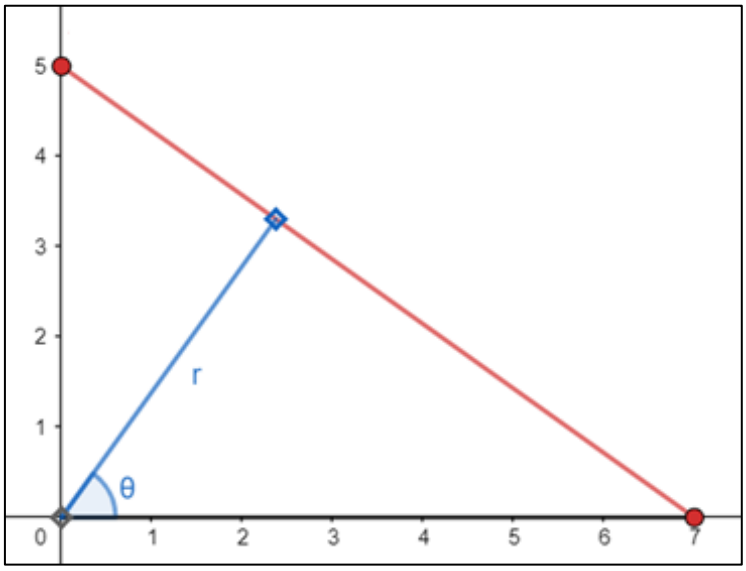


Figura 2–7. Representación de una recta en coordenadas polares.

Donde ***r*** es la longitud de la mediana que es perpendicular a la recta y pasa por el origen de coordenadas, y ***θ*** el ángulo que dicha mediana forma con el eje de abscisas. La recta puede definirse entonces como:

$$r = x \cdot cos\theta + y \cdot sin\theta \tag{4–3}$$

siendo (x, y) las coordenadas de un punto que pertenece a la recta.

Dado un punto (x_0, y_0) cualquiera, puede definirse una familia de rectas que contienen dicho punto como:

$$r = x_0 \cdot cos\theta + y_0 \cdot sin\theta \tag{4–4}$$

Cada par de valores (r, θ) representa una línea que pasa por el punto (x_0, y_0). Dicha familia de líneas equivale una sinusoide si se representa en un plano θ – r; por ejemplo, para un punto (x_0, y_0) = (8, 6):

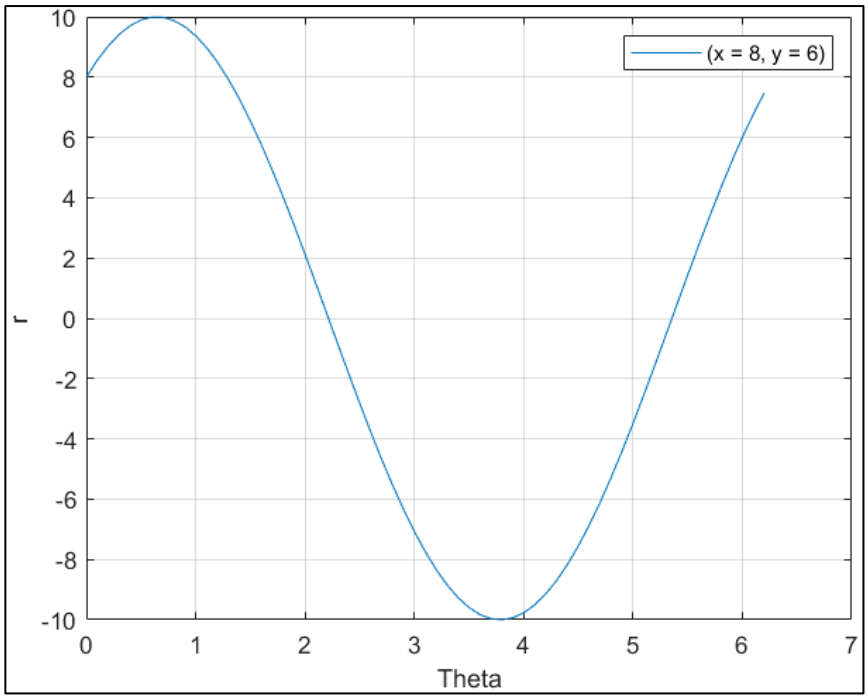
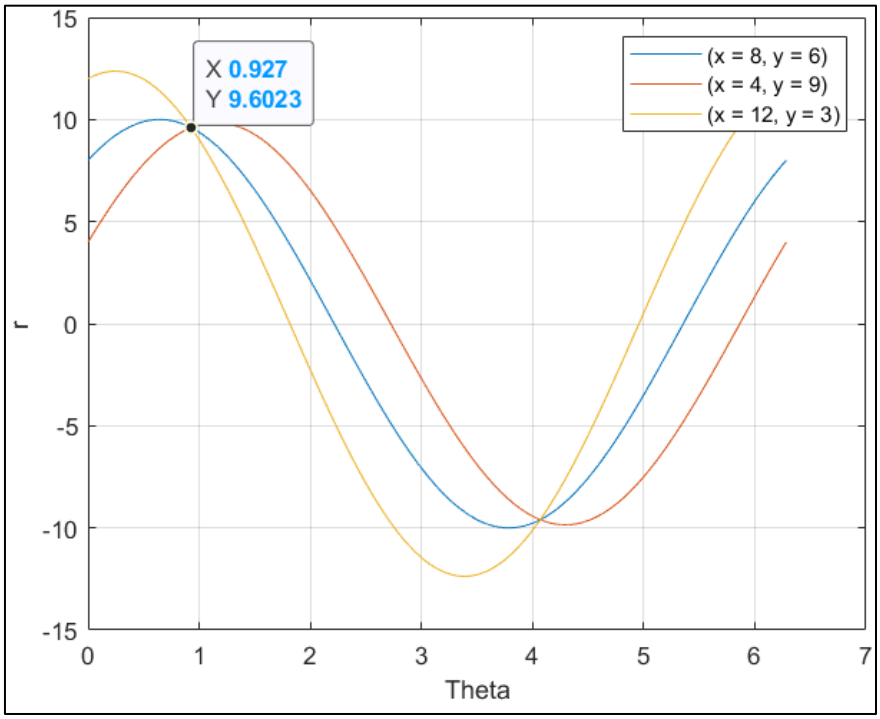


Figura 4–8. Representación θ – r de todas las rectas que pasan por el punto (8, 6).

Realizando el mismo proceso para los puntos (4, 9) y (12, 3):



Figra 4–9. Las sinusoides θ – r de dos o más puntos alineados entre sí comparten un punto de intersección.

No es coincidencia que las tres sinusoides intersequen en un mismo punto. Si se representan los tres

puntos dados anteriormente en un plano cartesiano puede comprobarse que forman parte de una misma recta:

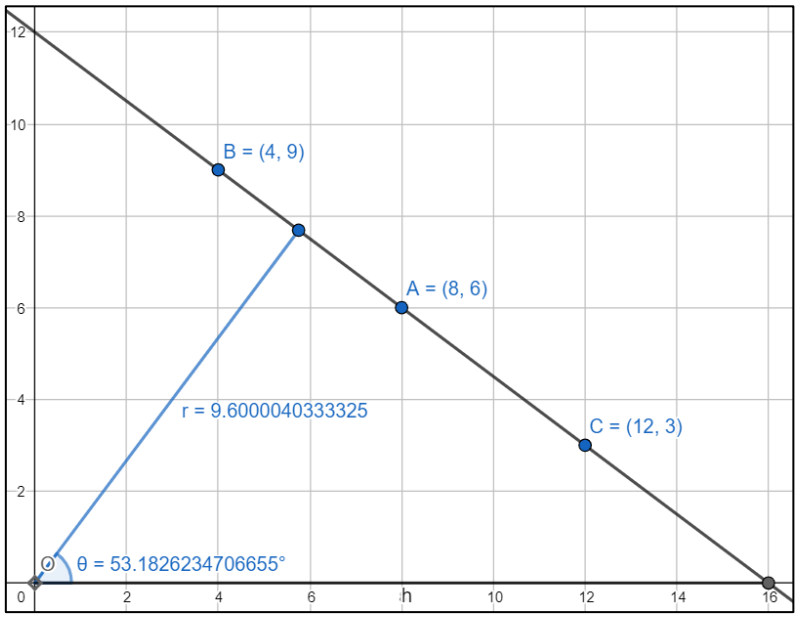


Figura 4–10. Las coordenadas polares de la recta corresponden al valor de intersección de las sinusoides θ – r.

Las coordenadas polares de dicha recta son las del punto de intersección de las sinusoides en el plano θ – r (53º son aproximadamente 0.92 radianes).

Para estimar líneas en una imagen mediante esta técnica lo que se suele hacer es realizar la transformada de Hough a todos los puntos de los bordes detectados previamente en dicha imagen (mediante algoritmo de Canny, por ejemplo). Posteriormente se computan las intersecciones de las sinusoides resultantes; aquellos pares de valores (r, θ) que se repitan múltiples veces serán candidatos a la detección de una recta en la imagen. Umbralizando el número de repeticiones mínimas que deben ocurrir o la tolerancia máxima permitida entre los valores de dichas repeticiones se puede parametrizar el funcionamiento de este algoritmo.

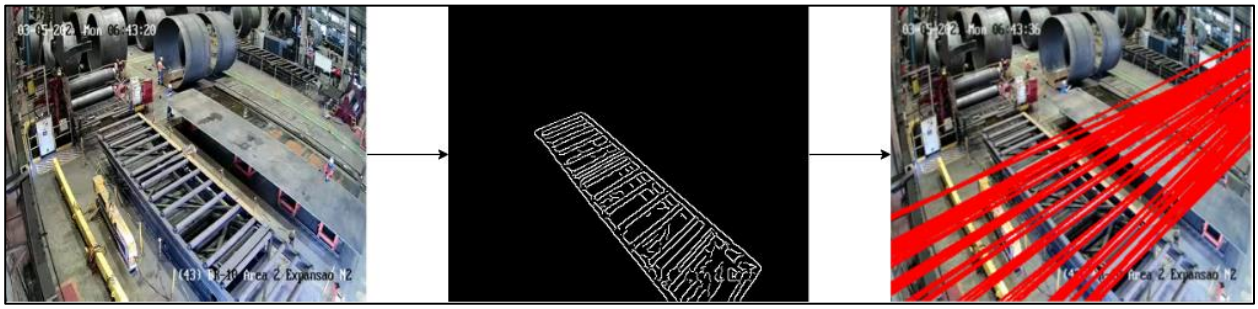


Figura 4–11. Detección de railes horizontales mediante Tfa. de Hough con OpenCV.

Una implementación más eficiente de esta técnica se conoce como Transformada de Hough de línea probabilística [19], que permite estimar los extremos de las líneas detectadas, o parametrizar la distancia permitida entre líneas detectadas.

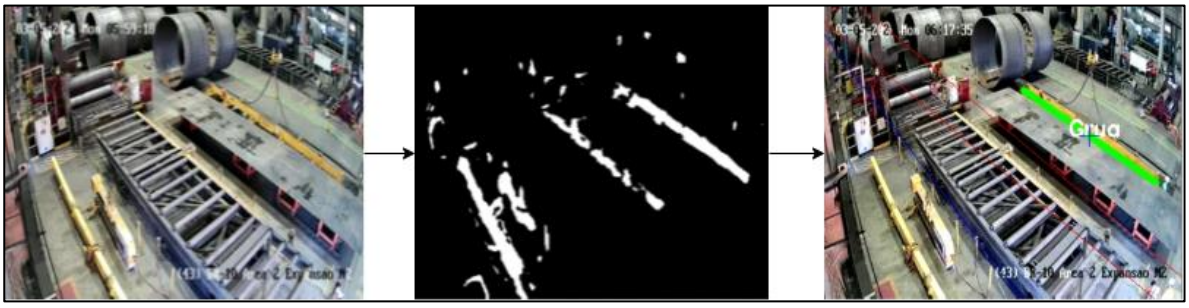


Figura 4–12. Detección de la grúa empleando Tfa. de Hough probabilística con OpenCV.

4.7 Background Substraction

En streams de video donde la cámara esté fija y el fondo sea estático, es posible detectar de manera muy sencilla objetos que entran y se mueven en la escena realizando lo que se conoce como sustracción de fondo [20]. Esto es, se toma una imagen del fondo de la escena y se restan píxel por píxel los valores de sus colores a los de la imagen de frames posteriores del video. Si no se ha producido ningún cambio en el escenario el resultado debería ser que ambas imágenes son iguales, luego su resta es nula. Si un objeto se mueve o entra en escena el resultado de la sustracción no será nulo y la diferencia podrá ser procesada.

En una aplicación práctica real es muy difícil que el fondo de la escena que se esté filmando sea totalmente estático, o que la percepción que la cámara obtiene de ella sea exactamente la misma que obtuvo en frames anteriores. Ligeros movimientos como vibraciones en la cámara o cambios en la ilumación provocan pequeñas pero perceptibles diferencias; por esto, se suele aplicar un umbral al resultado de la resta. Si en el frame de la resta, la intensidad de la diferencia es muy pequeña y no supera el umbral, se ignora. Si, por el contrario, dicha diferencia supera el umbral, pero es muy tenue, puede ser conveniente amplificarla de manera artificial para que sea más fácil trabajar con ella.

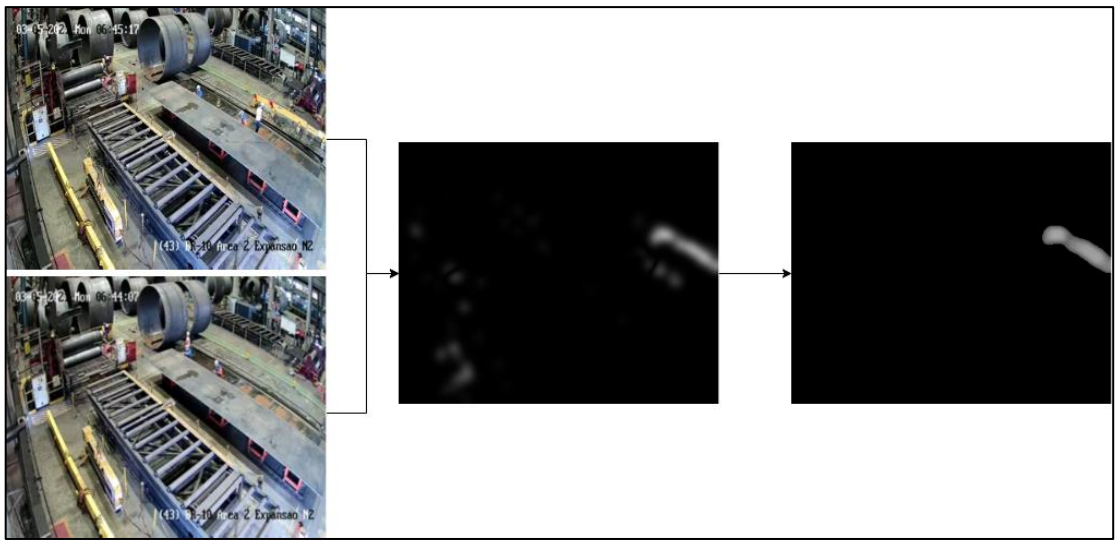


Figura 4–13. Ejemplo de detección de movimiento entre dos frames mediante background substraction.

Para mejorar el funcionamiento de algoritmos basados en este método se puede actualizar periódicamente el frame con el que se realiza la sustracción en lugar de trabajar siempre con el frame que hubiera al inicio del proceso. De esta forma pequeñas diferencias que se van produciendo paulatinamente como cambios en la iluminación no se acumulan pasado un tiempo largo. Esta técnica también se suele utilizar en combinación de otras que permiten estimar una imagen más precisa del background a partir de imagenes en las que éste solo se

muestre de forma parcial.

4.8 Optical Flow

Optical Flow [21], flujo óptico en castellano, se define como como el patrón de movimiento aparante de un objeto entre dos frames consecutivos. En computer vision se suele dar como un vector 2D que indica la cantidad de desplazamiento de un punto y su dirección entre dos frames.

Se debe asumir que el movimiento de un punto entre dos frames es relativamente pequeño y que puntos cercanos se mueven en la misma dirección. Sea I(x, y, t) la intensidad del pixel en un punto en la posición (x, y) para un instante t, que se mueve una distancia (dx, dy) en un tiempo dt:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \tag{4–5}$$

Realizando su aproximación mediante series de Taylor de primer orden y diviendo entre dt:

$$I(x + dx, y + dy, t + dt) - I(x, y, t) = \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt \tag{4–6}$$

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t}\frac{dt}{dt} = 0 \tag{4–7}$$

$\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\ \frac{\partial I}{\partial t}$ son los gradientes de la imagen, que pueden ser calculados, y $\frac{dx}{dt}, \frac{dy}{dt}$ las componentes de velocidad del flujo óptico, V_x y Vy, incógnitas.

Se tiene entonces una sola ecuación para dos incógnitas:

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0 \tag{4–8}$$

Dado que es un problema indeterminado, sólo se puede tratar de estimar una solución. Existen varios métodos y estrategias de estimación; en este trabajo se hará lo que se conoce como Dense Optical Flow, basado el algoritmo descrito por Gunnar Farneback en “Two-Frame Motion Estimation Based on Polynomial Expansion” en 2003. Entender el funcionamiento exacto de dicho algoritmo escapa de esta lectura; sólo es necesario saber que en este método se computa el flujo óptico de todos los píxeles en pantalla o de una red de píxeles equidistantes, lo que conlleva una carga computacional considerable, pero a la vez ofrece resultados más precisos.

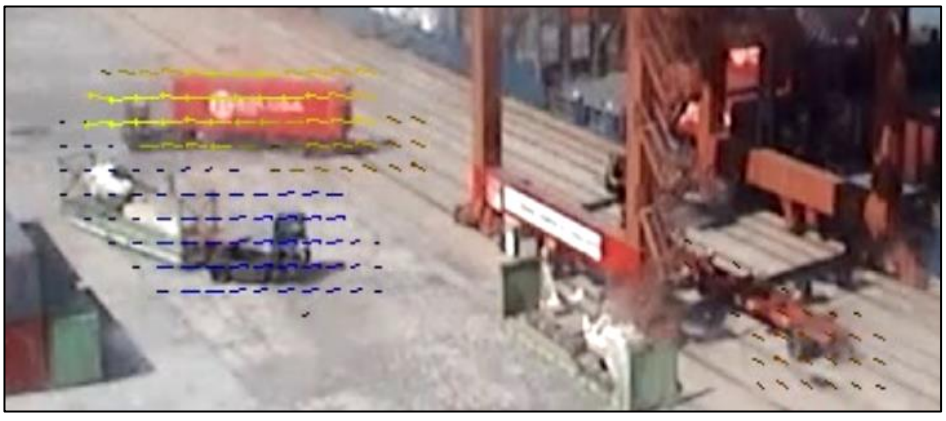


Figura 4–14. Ejemplo de Dense Optical Flow. El color del vector indica su dirección, el módulo su velocidad.

5 MÉTODO PROPUESTO

En este capítulo se enumerarán los objetivos que se desean alcanzar y se propondrá el plan de trabajo para lograrlo. Se hará primero una descripción del material con el que se va a trabajar, y posteriormente se presentará la ruta de trabajo seguida y los resultados.

5.1 Material

5.1.1 Hardware

Trabajar con redes neuronales para imágenes requiere de una potencia computacional para la que suele ser más adecuada una Tarjeta Gráfica (GPU) potente frente a una CPU que se encuentre en el mismo rango de precio. Esto se debe a que las arquitecturas de las GPUs están diseñadas para realizar múltiples cálculos simples de manera simultánea, de manera similar a como funciona una red neuronal biológica. Las CPUs, sin embargo, no están diseñadas para este tipo de tareas, pues pueden realizar cálculos más complejos pero de manera secuencial.

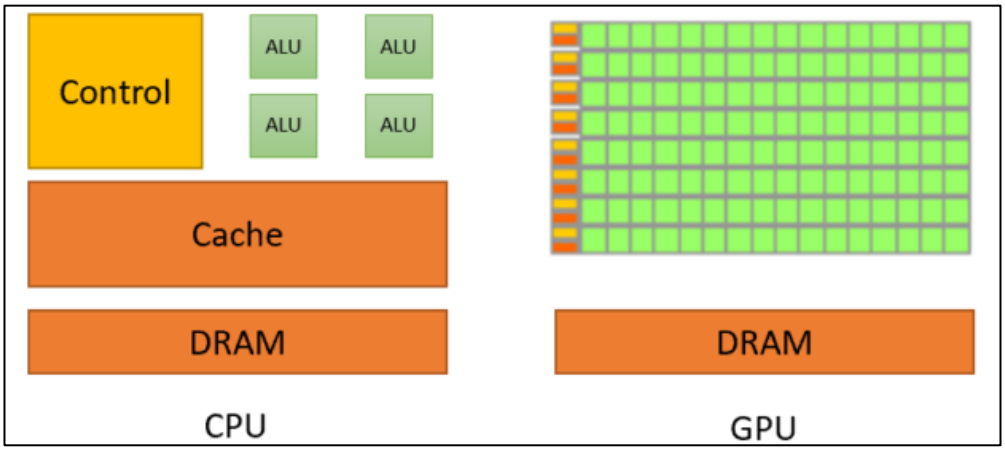


Figura 5–1. Ilustración simplificada de las arquitecturas de una CPU y una GPU. [22]

Es por esto por lo que empresas que se especializaron en el desarrollo de unidades de procesamiento gráfico como Nvidia o AMD han liderado los últimos avances referentes en computación de inteligencia artficial, pues contaban con el material y las tecnologías necesarias para implementar Deep Learning de manera efectiva.

Para este trabajo se hará uso de un equipo HP Pavilion Power 15-CB012NS con las siguientes características:

Tabla 5–1. Características del hardware de trabajo.

OS	Windows 10 Enterprise 64 bits
CPU	Intel Core i7-7700HQ @2.80GHz
RAM	8GB SK Hynix 2400MHz
GPU	NVIDIA GeForce GTX 1050
SSD	128 GB
HDD	1 TB

Aunque no es un equipo que vaya a desempeñar un rendimiento espectacular, es más que suficiente para implementar tanto deep learning como computer vision; siendo la desventaja principal que le tomará un

tiempo relativamente grande la etapa de entrenamiento de las CNNs, así como una carga computacional considerable el realizar inferencias con dichas redes.

Existen alternativas si no se dispone del hardware necesario para trabajar con Deep learning. Empresas como Amazon, Microsoft o Google ofrecen servicios de pago de computación en la nube, como Amazon Web Service, Microsoft Azure y Google Collab. Para este trabajo se consideró que no era necesario contratar ninguno de esos servicios, decidiéndose continuar con el equipo descrito anteriormente

5.1.2 Software

Este trabajo se desarrollará empleando Python como lenguaje de programacion, debido a su versatilidad y las expectativas de crecimiento de número de dispositivos progamados en dicho lenguaje para los próximos años. El IDE empleado será Spyder 3.3.6, una versión obsoleta de Spyder pero que hace posible trabajar con unas librerías concretas que no están disponibles en versiones actuales de Spyder.

Las librerías que se han empleado para este trabajo son:

- **OS:** librería incluida en Spyder que permite manipular direcciones de carpetas, mover archivos de lugar, etc. Suele ser necesaria para poder hacer uso de otras librerías sin que entren en conflicto con el propio Sistema Operativo.
- **numpy 1.19.5:** librería que permite realizar operaciones matemáticas en arrays multidimensionales.
- **matplotlib 3.3.4:** librería para graficar datos o visualizar imágenes.
- **imutils 0.5.4:** incluye funciones para el tratamiento de imágenes.
- **Keras 2.2.4:** librería para crear y entrenar ANNs, enfocada al deep learning. Se complementa con otras librerías como TensorFlow.
- **TensorFlow 1.15.3:** librería creada para computación numerica de alta velocidad enfocada al machine learning.
- **opencv-python 4.5.2.52:** incluye una amplia variedad de funciones y utilidades para computer vision y procesamiento de imágenes.
- **Mask R-CNN:** framework para segmentación y encuadre de instancias de objetos [23]. Basado en los modelos de red ResNet 101 y FPN; se hará uso del modelo pre-entrenado para el dataset Microsoft COCO.

5.2 Reconocimiento de barcos cargueros y grúas portuarias

Para esta parte del trabajo se plantea desarrollar un software que sea capaz de identificar en tiempo real la situación de un muelle del Puerto de Sevilla. Se propone entrenar una red neuronal para localizar barcos cargueros y grúas portuarias. Esta red funcionará apoyada de un algoritmo que estabilice la detección de los objetos mencionados. Se desea conseguir ser capaz de identificar y señalar correctamente la posición de barcos y grúas.

5.2.1 Creación de la base de datos de imágenes

Para entrenar la red neuronal es preciso disponer de una gran cantidad de imágenes en la que aparezca objeto que se desea detectar. Se descargan imágenes de barcos cargueros y grúas portuarias sacadas de páginas como unsplash.com [24]. En total se han recopilado 277 imágenes de barcos cargueros y 65 de grúas portuarias; es una cantidad bastante pequeña, pero mediante data augmentation se ampliará el número de imágenes x10.

Estos archivos vienen sin etiquetar; es decir, no incluyen información sobre qué objeto se muestra o dónde se encuentra dentro de la imagen. Como se va a realizar aprendizaje supervisado, es necesario añadir manualmente esta información para cada una de las imágenes descargadas. Para esta tarea se hará uso del software LabelImg [25], una aplicación para etiquetar objetos dentro de una imagen definiendo su posición. El resultado es un archivo .xml con las coordenadas del rectángulo que encuadra dicho objeto y su clase (tipo de objeto que es).

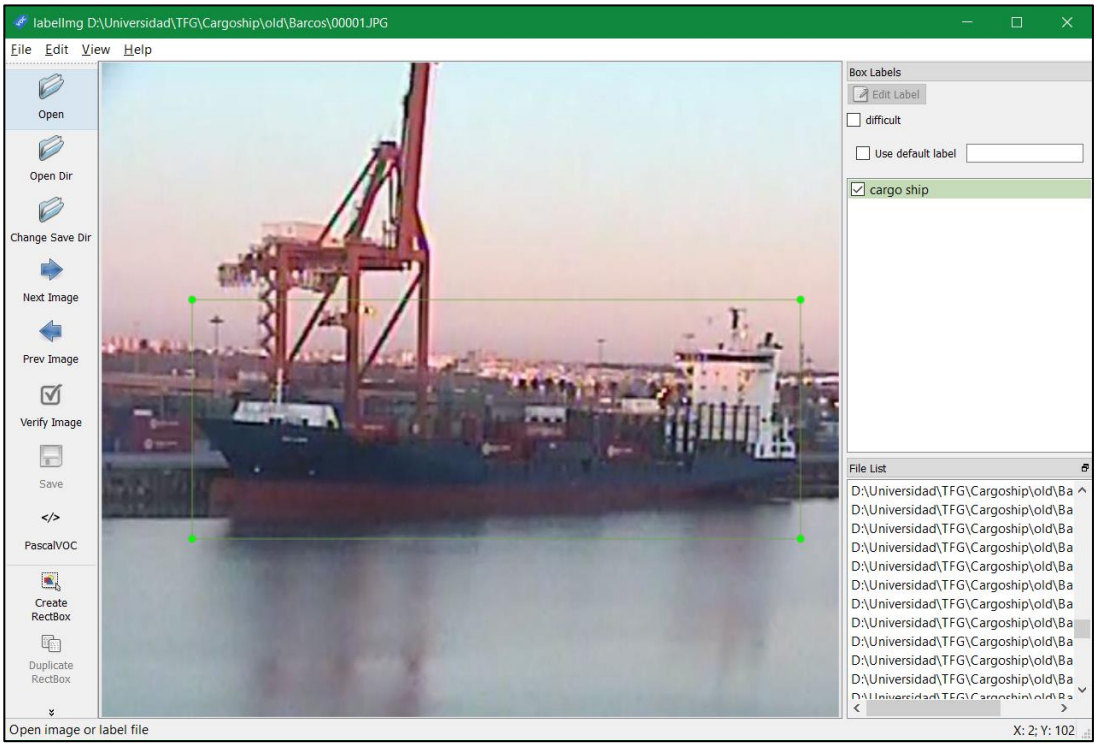


Figura 5–2. Asignación de etiquetas para las imágenes empleando LabelImg.

En función del número de objetos encuadrados y sus clases se tendrá un archivo .xml de la forma:

```
<annotation>
	<folder>Barcos</folder>
	<filename>00222.JPG</filename>
	<path>D:\Universidad\TFG\Fotos\Barcos\00222.JPG</path>
	<source>
		<database>Unknown</database>
	</source>
```

```
        <size>
                <width>750</width>
                <height>665</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>cargo ship</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>92</xmin>
                        <ymin>241</ymin>
                        <xmax>711</xmax>
                        <ymax>484</ymax>
                </bndbox>
        </object>
</annotation>
```

Código 5–1. Información de etiqueta contenida en el archivo .xml

Cuando se entrene la red neuronal habrá que pasarle tanto el archivo de la imagen como la información correspondiente de su etiqueta sacada del archivo .xml generado (en este caso las coordenadas del recuadro dentro de la llave 'bndbox').

```
# Extrae marcos de los archivos de anotación
def extract_boxes(self, filename):
    # Carga y analiza el archivo
    tree = ElementTree.parse(filename)
    # Obtiene la raiz del documento
    root = tree.getroot()
    # Extrae cada caja de marcos
    boxes = list()
    for box in root.findall('.//bndbox'):
        xmin = int(box.find('xmin').text)
        ymin = int(box.find('ymin').text)
        xmax = int(box.find('xmax').text)
        ymax = int(box.find('ymax').text)
        coors = [xmin, ymin, xmax, ymax]
        boxes.append(coors)
    # Extrae dimensiones de la imagen
    width = int(root.find('.//size/width').text)
    height = int(root.find('.//size/height').text)

    return boxes, width, height
```

Código 5–2. Función que extrae las coordenadas de los recuadros a partir de los archivos .xml

Para aumentar la base de datos de manera artificial se realiza lo que se conoce como data augmentation: introducir pequeños cambios como rotaciones, desplazamientos, cambios de color o añadido de ruido en la imagen. Para emplear esta técnica se va a hacer uso de una librería para data augmentation obtenida de Github [26]. Esta librería, a parte de realizar modificaciones de forma aleatoria en la imagen, también modifica su archivo .xml asociado de forma que las etiquetas estén posicionadas correctamente en la imagen modificada. El código empleado aparece en el anexo de data augmentation.

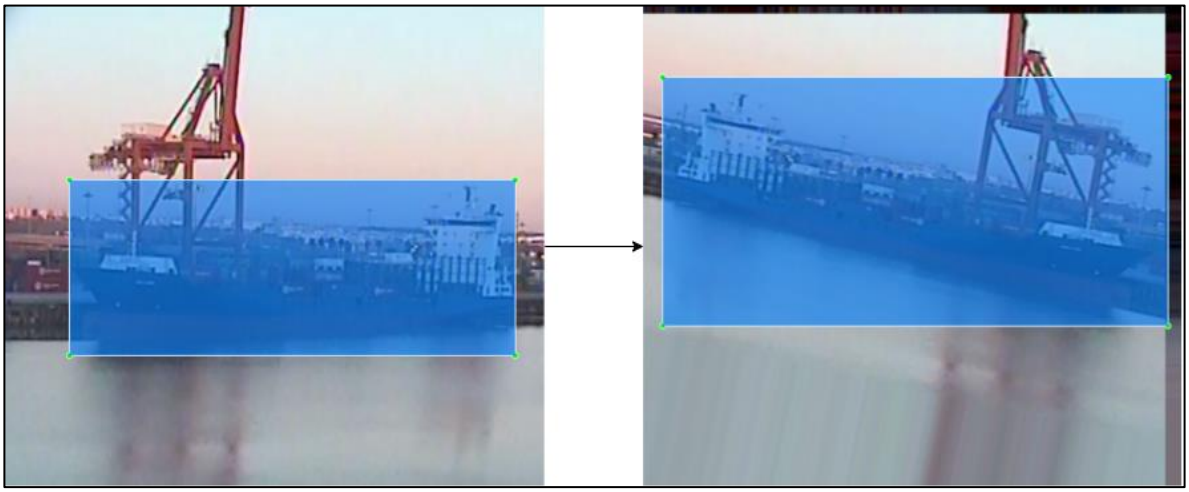


Figura 5–3. Ejemplo de realizar data augmentation.

Tras realizar data augmentation se pasa de tener 277 imágenes de barcos a tener alrededor de **2700**, y de 65 de grúas a tener alrededor **600**.

Por último, se divide la base de datos entre dataset destinado a entrenamiento y dataset destinado a validación. Como regla general, se suele destinar un 80% del total de imágenes al dataset de entrenamiento y el 20% restante al dataset de validación.

```
def load_dataset(self, dataset_dir, is_train=True):
    # Define una clase
    self.add_class("dataset", 1, "Cargoship")
    # Define localización de las imágenes + anotaciones
    images_dir = dataset_dir + '/Barcos/'
    annotations_dir = dataset_dir + '/Barcos-Annot/'
    # Encuentra todas las imágemes y asigna una ID por el nombre (número)
    for filename in listdir(images_dir):
        image_id = filename[:-4]
        # Salta id erronea/vacía
        if (int(image_id) >= 1 and int(image_id) <= 20) or (int(image_id) >= 2090 and
int(image_id) <= 2099) or (int(image_id) >= 2490 and int(image_id) <= 2499) or (int(image_id) >=
2540 and int(image_id) <= 2549):
            continue
        # Sólo incluye hasta un número para el set de entrenamiento
        if is_train and int(image_id) >= 2184:
            continue
        # Salta hasta un número para el set de validación
        if not is_train and int(image_id) < 2184:
            continue
        img_path = images_dir + filename
        ann_path = annotations_dir + image_id + '.xml'
        # Añade imagen al dataset

        self.add_image('dataset', image_id=image_id, path=img_path, annotation=ann_path)
```

Código 5–3. Creación del dataset de entrenamiento y validación de barcos.

El resto del código sobre la creación de los datasets de imágenes y las funciones asociadas a estos se encuentran en los anexos TrainBarcos.py y TrainGruas.py.

5.2.2 Entrenamiento y validación del modelo

En lugar de crear un modelo de red neuronal desde cero, se va a tomar un modelo ya pre-entrenado para detección de objetos mediante un dataset genérico, y se va a re-entrenar para detectar los objetos de interés de este trabajo con el dataset creado anteriormente; este proceso se conoce como ‘transfer learning’. Se va a trabajar con el modelo pre-entrenado incluido en Mask R-CNN, a partir del dataset de imágenes Microsoft

COCO [27].

A partir de los datasets de entrenamiento y validación definidos en el punto anterior se entrenan dos modelos de redes neuronales, uno para la detección de barcos y otro para la detección de grúas. En ambos modelos se entrenan durante 5 épocas, consistiendo cada época de 150 steps.

```
# ENTRENAMIENTO
# Ignora warnings de librerías obsoletas
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)

# Define la configuración de predicción
class CargoshipConfig(Config):
        NAME = "cargoship_cfg"
        # Num. de clases (background + barco)
        NUM_CLASSES = 1 + 1
        # Numero de pasos por época
        STEPS_PER_EPOCH = 150


# Prepara dataset de entrenamiento
train_set = CargoshipDataset()
train_set.load_dataset('Cargoship', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))

# Prepara dataset de validación
test_set = CargoshipDataset()
test_set.load_dataset('Cargoship', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))

# Prepara la configuración
config = CargoshipConfig()
config.display()

# Define el modelo: entrenamiento
model = MaskRCNN(mode='training', model_dir='./', config=config)

# Carga los pesos del modelo prototipo (coco) y excluye las capas de salida
model.load_weights('mask_rcnn_coco.h5', by_name=True, exclude=["mrcnn_class_logits",
"mrcnn_bbox_fc",  "mrcnn_bbox", "mrcnn_mask"])

# Entrena el modelo
# NOTA: Puede tardar del orden de 10 horas. Variar el número de épocas y el número de pasos por
época
# puede reducir/aumentar este tiempo a coste de sobreentrenar la red. Es más recomendable
aumentar el dataset antes de
# modificar los parámetros de entrenamiento

model.train(train_set, test_set, learning_rate=config.LEARNING_RATE, epochs=5, layers='heads')
```

Código 5–4. Entrenamiento de red para detección de barcos. Análogo a la de detección de grúas.

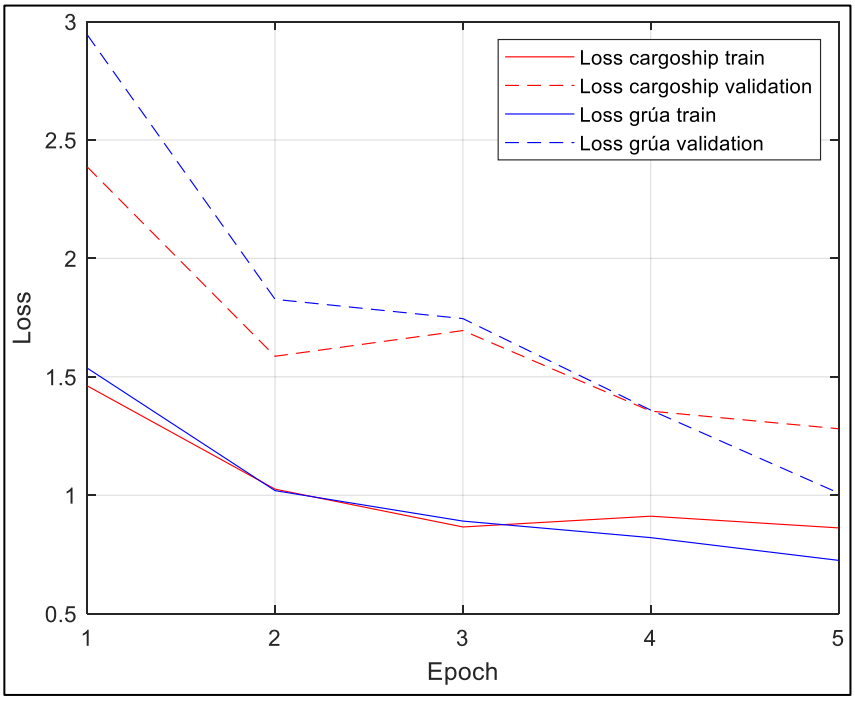


Figura 5–4. Curvas de pérdida de entrenamiento y validación para barcos y grúas.

Las gráficas de pérdida de entrenamiento en ambos casos no terminan de ser horizontal, por lo que puede ser interesante entrenar durante más épocas; sin embargo, se va a continuar el trabajo con los modelos de 5 épocas debido a la gran cantidad de tiempo que requiere realizar el entrenamiento (alrededor de 18 y 10 horas para los datasets de barcos y grúas, respectivamente), se deja como una posibilidad de mejora del trabajo.

5.2.3 Simulación de detección en tiempo real

Para probar cómo funcionaría la red neuronal entrenada ante un stream de video en directo de una cámara apuntando al Puerto de Sevilla, se emplea un video de una grabación de seguridad del sitio en el que aparecen dos grúas portuarías y dos barcos cargueros atracados en un muelle del puerto.

Figura 5–6. Frame del vídeo con el que se va a trabajar.

Realizando inferencias de la red sobre frames del video se obtiene una primera aproximación al problema. Aunque es capaz de detectar correctamente los dos barcos y las dos grúas del video, también genera falsos positivos y solapa la detección de barcos y grúas. Para solventar este tipo de problemas se realizará un post-procesado a las detecciones de la red.

Figura 5–7. Inferencias generadas por la red neuronal.

En un primer acercamiento, este post-procesado se va a centrar en solventar los siguientes problemas:

- **Falsos positivos:** se filtrarán las detecciones de la red para descartar aquellas con una predicción menor al 90%.
- **Solapamiento grúa-barco:** se computará el centro de cada rectángulo, y se comprobará si está dentro del rectángulo de otro objeto. Si es el caso, se hará una corrección manual al tamaño de los rectángulos, de forma que disminuya el solapamiento (se va a considerar que las grúas siempre deben de estar más arriba que los barcos).

Figura 5–8. Resultado de aplicar el primer post-procesado.

Tras aplicar el post-procesado descrito anteriormente, se obtiene una mejora en la detección de los objetos. Sin embargo, existen frames como el de la Fig. 5–8 en los que el rectángulo de uno de los barcos engloba al del otro barco, también existen frames en los que detecta a los dos barcos como uno solo, dejando de detectar al segundo (falsos negativos). Por último, aunque se ha corregido la mayor parte del solapamiento entre barco y grúa, sigue habiendo un solapamiento significativo. Se intenta corregir estos errores mediante post-procesado:

- **Englobamiento de barcos:** similar a lo que se hizo con el solapamiento grúa-barco, se comprueba si el centro del rectángulo de un barco está dentro del rectángulo de otro barco. Si es el caso, se corrige manualmente el tamaño de los rectángulos en el eje horizontal.
- **Falsos negativos:** se comprueba si el número de detecciones con respecto al frame anterior ha cambiado. Si ha decrementado (se detectan menos objetos) se ignorará la nueva detección y se seguirán mostrando las del frame anterior. Si este número persiste durante 3 periodos, entonces se dejará de considerar un falso negativo y se mostrarán las nuevas detecciones.
- **Falsos positivos:** aunque tras filtrar las detecciones según el nivel de predicción, pueden existir casos en los que se den falsas detecciones esporádicas. Esto se corregirá de la misma manera que los falsos negativos, salvo que en este caso se comprueba si el número de detecciones ha incrementado.
- **Solapamiento grúa-barco:** se comprueba si el area de los rectángulos solapan. Si es el caso, se corrige manualmente la posición de los bordes verticales a una cota intermedia.
- **Solapamiento barco-barco:** para evitar que se de el caso de solapamiento grúa-barco de la figura, se realiza la misma operación del punto anterior horizontalmente entre rectángulos de barcos.

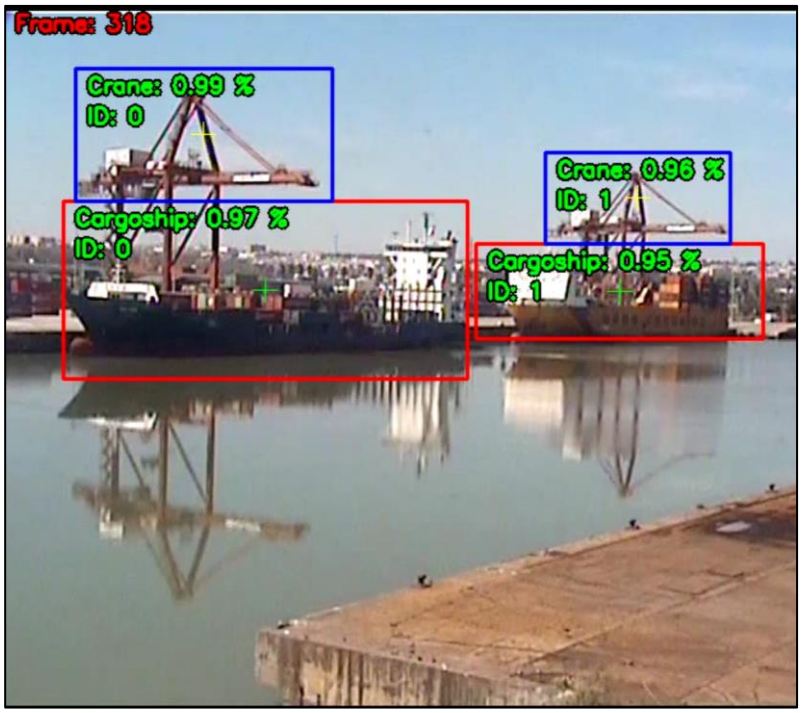


Figura 5–9. Resultado de aplicar el post-procesado completo.

Tras aplicar el post-procesado descrito la detección de las grúas y barcos se realiza de forma estable durante todos los frames del video, pudiendo extrapolarse a un stream en directo.

Para comprobar el correcto funcionamiento del algoritmo de detección y post-procesado es necesario ver el vídeo en movimiento donde se esté aplicando el programa.

El código empleado esta sección no se mostrará aquí debido a su extensa longitud; se añadirá como anexo al final del documento.

5.3 Detección de objetos en fábrica de virolas

En esta parte del trabajo lo que se pretende conseguir es implementar computer vision en una fábrica industrial de virolas para determinar el estado de la maquinaria y la posición de una grúa suspendida.

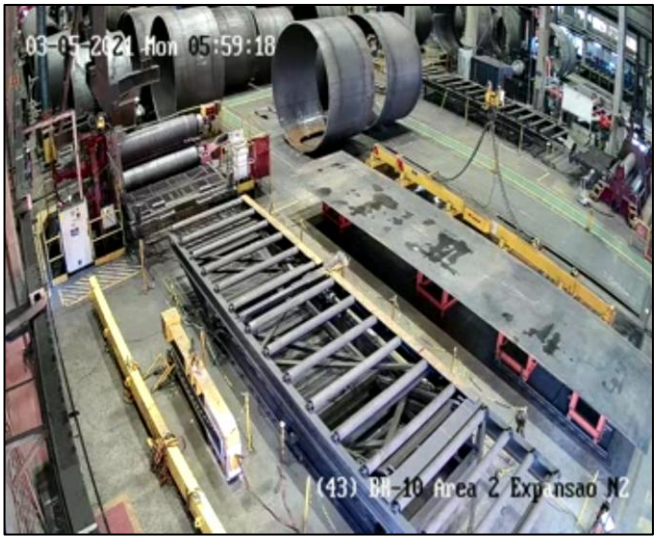


Figura 5–10. Frame del video con el que se va a trabajar.

5.3.1 Detección de la grúa

Antes de comenzar, es necesario suponer que en el stream de vídeo sólo podrá aparecer una única grúa, y que esta será de color amarillo. Como se va a detectar la grúa por su color, es necesario ignorar todas las partes de la escena que puedan confundirse con la grúa: todo aquello de color amarillo que sea estático. Para eso, se crea una máscara para ignorar dichas zonas:

Figura 5–11. Máscara que elimina partes amarillas de la escena.

El siguiente paso es aquellos píxeles cuyos colores no estén dento de un rango de tonalidades amarillas. Para definir este rango de color se hace empleo del espacio de colores HSV, como se vio en el capítulo 4. El rango HSV escogido es ([15, 50, 120], [35, 255, 255]):

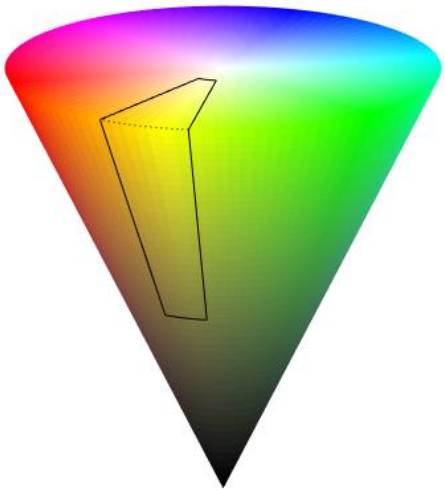


Figura 5–12. Rango de colores del filtro definido en espacio HSV.

De las figuras resultantes del proceso anterior, se determinan sus bordes mediante algoritmo de Canny.

Figura 5–13. En sentido horario: (a) Imagen del stream de video. (b) Aplicación de una máscara y filtrado Gaussiano. (c) Filtrado por color en espacio HSV. (d) Detección de bordes mediante algoritmo de Canny.

Mediante la transformada de Hough probabilística se detectan las líneas de mayor longitud, que corresponden a la grúa que se quiere reconocer. La función incluida la librería opencv permite parametrizar la longitud mínima que debe tener la línea (en píxeles) y el tamaño máximo de una discontinuidad en la propia línea, si es que la hubiera. Para este caso se parametriza una longitud mínima de 45px y un salto máximo de 35px.

Como se está aplicando sobre los bordes del contorno de la grúa, lo normal es que detecte dos líneas en lugar de solo una. Esto se puede corregir simplemente ignorando la línea más corta y quedándose con la de mayor longitud:

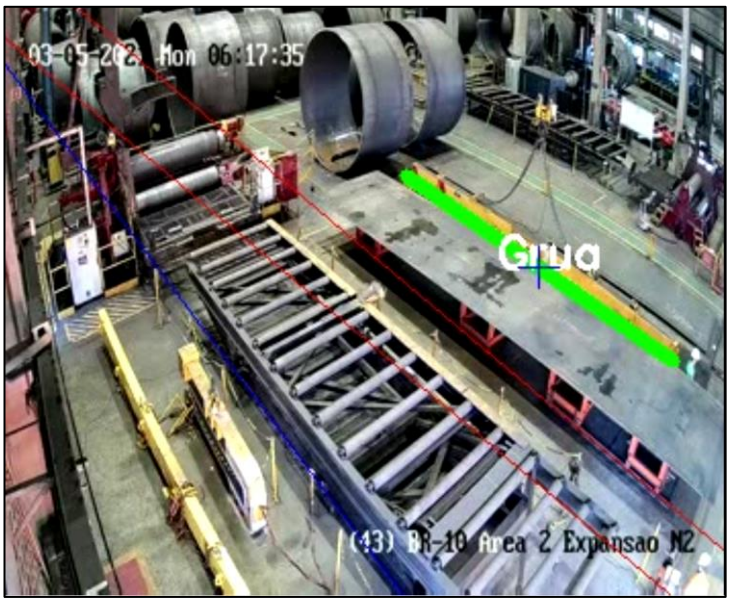


Figura 5–14. Detección de posición de la grúa mediante Tfa. de Hough.

Por último, se añade un algoritmo para mejorar la "continuidad" de la detección: evitar en lo posible falsos positivos y negativos, promediando la posición de la grúa con frames anteriores y dibujando la grúa en su última posición detectada durante algunos frames si el programa ha dejado de detectarla. Para comprobar estas características es necesario ver un vídeo en movimiento donde se esté aplicando el programa.

```
# Detecta cosas amarillas (grúa)
# Aplica máscara para ignorar objetos estáticos amarillos, luego aplica un filtrado gaussiano
frameYeLi = cv2.bitwise_and(frame,frame, mask = maskAntiAmarillos)
frameYeLi = cv2.GaussianBlur(frameYeLi, (7, 7), 0)

# Conversión a espacio de colores HSV
hsv = cv2.cvtColor(frameYeLi, cv2.COLOR_BGR2HSV)

low_yellow = numpy.array([15, 50, 120]) # Rango para colores amarillos
up_yellow = numpy.array([35, 255, 255])
mask_yellow = cv2.inRange(hsv, low_yellow, up_yellow)      # Filtrado por color. Ignora lo que
no sea amarillo
mask_HSVGrua = cv2.GaussianBlur(mask_yellow, (21, 21), 0)  # Filtrado gaussiano, facilita los
siguientes procesos

# Detecta los bordes de las figuras resultantes
edges = cv2.Canny(mask_HSVGrua, 50, 100)

# Estima líneas rectas de los bordes anteriores, longitud mínima: 45px, máxima discontinuidad:
35px
linesGrua = cv2.HoughLinesP(edges, 1, numpy.pi/180, 45, maxLineGap=35)

px1, px2, py1, py2 = [0, 0, 0, 0]
    if linesGrua is not None:
        for line in linesGrua:
            x1, y1, x2, y2 = line[0]
            longitud = math.sqrt(abs(x1-x2)^2+abs(y1-y2)^2)
            # Calcula longitudes, dibuja la más larga
            if (longitud > varDist):
                varDist = longitud
                px1 = x1
                px2 = x2
                py1 = y1
                py2 = y2
```

```
    if not((px1, py1) == (0,0) or (px2, py2) == (0,0)): # No tiene un punto nulo (en el origen)
        cv2.line(output, (px1, py1), (px2, py2), (0, 255, 0), 5)
        xc = abs(int((px2-px1)/2))+px1
        yc = abs(int((py2-py1)/2))+py1
        string = "Grua"
        cv2.putText(output, string, (xc-20, yc), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 4)
        cv2.putText(output, string, (xc-20, yc), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
        cv2.drawMarker(output, (xc, yc), (255, 0, 0))

        countFramesGruaOLD = 0
        if flagGuardaPosGrua == 0:
            PosGruaOLD = [px1, py1, px2, py2]   # Posición de la grúa en el frame anterior
            flagGuardaPosGrua = 1
    elif countFramesGruaOLD < 120 and not(PosGruaOLD == [0,0,0,0]):   # En caso de que no haya
grua que dibujar, dibuja la que había en el frame anterior
        [px1, py1, px2, py2] = PosGruaOLD
        countFramesGruaOLD += 1
        cv2.line(output, (px1, py1), (px2, py2), (0, 255, 0), 5)
        xc = abs(int((px2-px1)/2))+px1
        yc = abs(int((py2-py1)/2))+py1
        string = "Grua"
        cv2.putText(output, string, (xc-20, yc), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 4)
        cv2.putText(output, string, (xc-20, yc), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
        cv2.drawMarker(output, (xc, yc), (255, 0, 0))
        gruaEnPantalla = 1
    else:   # No detecta grúa en la pantalla
        PosGruaOLD = [0,0,0,0]
        countFramesGruaOLD = 0
        gruaEnPantalla = 0
```

Codigo 5–5. Algoritmo de detección de la grúa.

5.3.2 Detección del estado de la máquina dobladora

En este apartado lo que se pretende detectar es si la máquina dobladora puede ser alimentada con planchas de metal. Se debe detectar si el carril de rodillos está libre u ocupado para saber si se puede colocar una nueva plancha de metal sobre éste. La estrategia para seguir será intentar detectar el número de rodillos en pantalla, si se detecta una cantidad alta es que el carril está libre; si hay una plancha de metal encima y no se detectan rodillos es que el carril está en uso.

Se comienza aplicando una máscara para ignorar todas las zonas de la imagen que no correspondan al carril de rodillos:

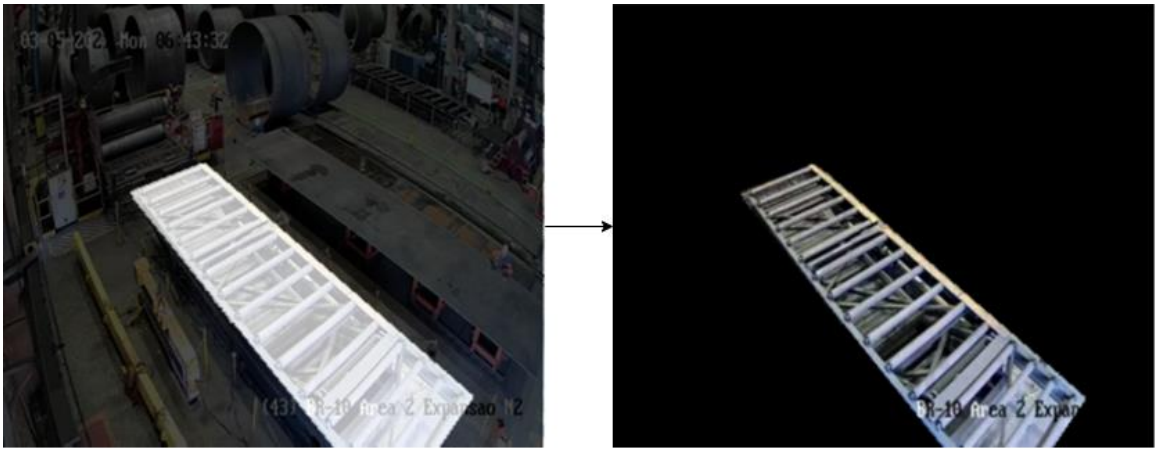


Figura 5–15. Máscara que marca la ROI.

Tras aplicar un filtrado gaussiano, se detectan los bordes de los rodillos mediante algoritmo de Canny:

Figura 5–16. Detección de bordes de los railes mediante algoritmo de Canny.

Para detectar rectas en la imagen se utiliza la Transformada clásica de Hough, que en la librería OpenCV viene dada en la función HoughLines. A diferencia de la función usada para detectar la grúa, esta dibuja rectas infinitas, sin embargo, se emplea esta función en concreto ya que facilita distinguir entre rectas según su ángulo o pendiente; las rectas asociadas a los rodillos tienen una inclinación de entre 43º y 72º.

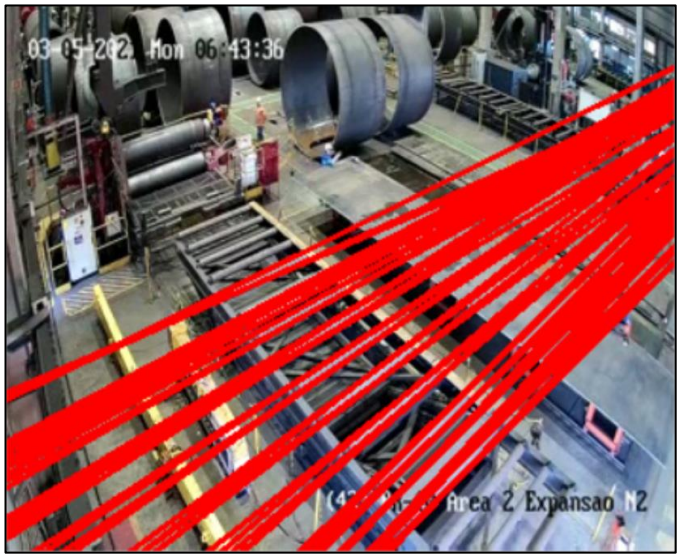


Figura 5–17. Detección de las rectas de los railes.

No es impresdincible que las rectas sean del tamaño de los rodillos. Por último, basta con contar el número de líneas detectadas para determinar el estado del carril. Para este caso, se propuesto que si el numero de rectas detectadas es superior a 40, entonces el carril está libre, y si es inferior a 15, es que el carril está ocupado:

Figura 5–18. Si algo obstruye la visibilidad de los rodillos se considera que el rail está siendo utilizado.

```
# Aplica máscara que se queda con la zona de los railes de la dobladora
res = cv2.bitwise_and(frame,frame,mask = maskRailes)
cv2.imshow("Mascara railes", res)
frameBlur = cv2.GaussianBlur(res, (9, 9), 0)

edges = cv2.Canny(frameBlur, 50, 150)
cv2.imshow("Edges railes", edges)

linesRailes = cv2.HoughLines(edges, 1, numpy.pi/360, 45)

# Railes dobladora
frameRailes = frame.copy()
countLines = 0
for line in linesRailes:    # Cuenta total de lineas detectadas en los rieles de la dobladora
    rho,theta = line[0]
    if theta > 0.75 and theta < 1.25:
        a = numpy.cos(theta)
        b = numpy.sin(theta)
        x0 = a*rho
        y0 = b*rho
        x1 = int(x0 + 1000*(-b))
        y1 = int(y0 + 1000*(a))
        x2 = int(x0 - 1000*(-b))
        y2 = int(y0 - 1000*(a))
        cv2.line(frameRailes,(x1,y1),(x2,y2),(0,0,255),1)    # Dibuja línea en frame

        countLines += 1

if railesOcupados == 0 and countLines < 15: # Determina el estado del raíl y lo notifica en
pantalla
    railesOcupados = 1
elif railesOcupados == 1 and countLines > 40:
    railesOcupados = 0

if railesOcupados == 0:
    string = "Rail: Libre"
    color = (255, 254, 51)
else:
    string = "Rail: Ocupado"
    color = (255, 0, 0)

cv2.putText(output, string, (20, frame_height-20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 4)
cv2.putText(output, string, (20, frame_height-20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)
```

Código 5–6. Algoritmo de comprobación del estado del rail de la máquina dobladora.

5.3.3 Detección del estado de la grúa

Determinar si la grúa está en movimiento o parada no es tan sencillo como comparar si, en la detección de esta en dos frames diferentes, se ha cambiado de posición. En un stream de vídeo continúo puede darse el caso en el que factores externos provoquen ligeros cambios en la percepción de la grúa, aún estando estática, haciendo que su detección no sea idéntica a la de frames anteriores.

Para tener mayor fiabilidad a la hora de intentar determinar el estado de la grúa se va a hacer uso de lo que se conoce como Background Substraction, explicado en el capítulo anterior.

Para comenzar, se va a tomar como imagen de fondo un frame donde no aparece la grúa. No siempre es posible conseguir un modelo del background 'límpio'; para este trabajo no es crucial que en la imagen de fondo no aparezca la grúa, pero facilita el funcionamiento del software que se va a implementar.

Figura 5–19. (a) Imagen de background utilizada. (b) Imagen de un frame posterior.

Como lo que se pretende distinguir es de color amarillo, se aplica de nuevo un filtrado por color en espacio HSV como se hizo en el punto 5.3.1:

Figura 5–20. Frames de la Fig. 5–19 filtrados por color amarillo.

La diferencia entre los dos frames se conoce como frame delta o frame de diferencia; el resultado de sustraer el background a la imagen de un frame dado. A menudo suele ser conveniente umbralizar el resultado obtenido para depurar zonas del que muestran una diferencia tenue que suelen corresponderse a ligeros

cambios de iluminación en el escenario.

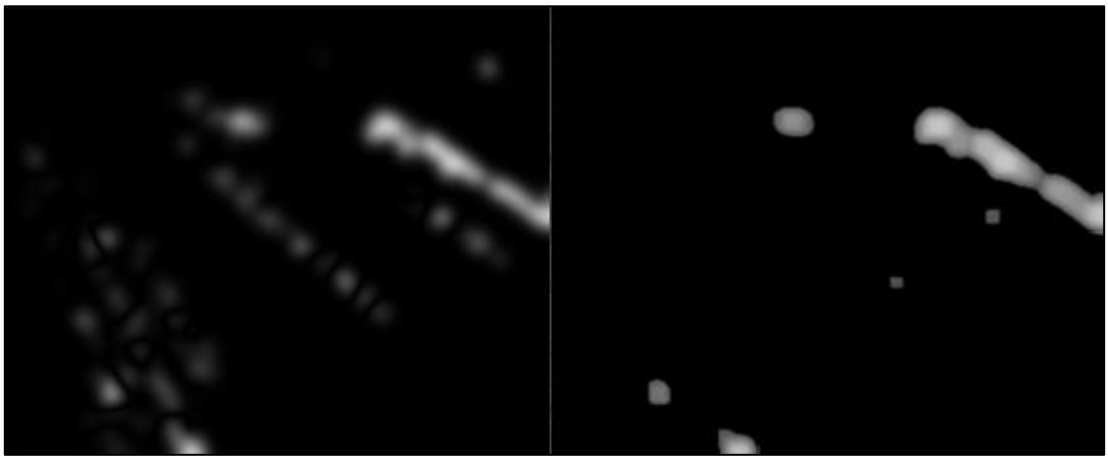


Figura 5–21. (a) Frame delta. (b) Frame delta umbralizado.

Aquellas zonas de intensidad inferior a cierto umbral serán ignoradas mientras que las zonas donde haya una intensidad de píxel alta se amplifican de manera artificial para facilitar su tratamiento posterior.

Lo que se pretende ahora es ser capaz de identificar sólo la forma alargada que se corresponde con la grúa en movimiento, e ignorar el resto de los puntos y figuras que puedan dar lugar a falsos positivos. Mediante la función findContours de la librería open-cv y grab_contours de imutils, se obtiene un rectángulo aproximado que enmarca la formas que aparecen en el frame delta.

Comparando el área de dichos rectángulos se pueden descartar si por ejemplo dicha área es inferior a 1000 píxeles cuadrados. Si el área de dicho rectángulo supera el umbral marcado, se comprueba si se ha detectado la grúa como se describió en el apartado 5.3.1; de ser así se comprueba si línea inferida mediante la Transformada de Hough está dentro o interseca con el rectángulo del objeto presuntamente en movimiento. Si se da el caso, se considera que la grúa está moviéndose.

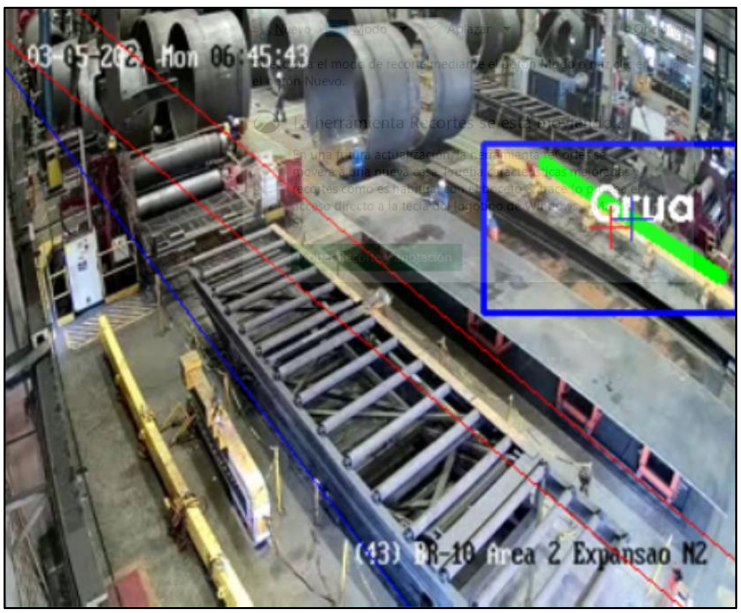


Figura 5–22. Se encuadra el contorno de las formas resultantes de umbralizar el frame delta.

En caso de que no se detecte movimiento mediante background substraction ni se haya detectado la grúa mediante Tfa. De Hough en el apartado 5.3.1, se considerará que la grúa estará fuera de pantalla.

Por último, si el rectángulo que encuadra el resultado del frame delta umbralizado no varía su posición ni tamaño durante un número consecutivo de frames, se considerará que la grúa ha dejado de moverse y está quieta. Se actualiza la imagen de background a la de ese instante para que el resultado del background substraction sea nulo.

Figura 5–23. En sentido horario: (a) La grúa está fuera de cámara. (b y c) La grúa entrar en pantalla, moviéndose. (d) La grúa se detiene y se posa en el suelo.

```
# Si en un frame no se ha detectado la grúa se dibujará la del frame anterior
# Si pasado un número de frames sigue sin detectarla, dejará de dibujar la grúa antigua
(posiblemente haya salido de la pantalla)
countFramesGruaOLD = 0
PosGruaOLD = [0, 0, 0, 0]    # Posición de la grúa en el frame anterior
flagGuardaPosGrua = 0

# Inicializar primer frame del video/stream
firstFrame = None
min_area = 1000 # Área mínima de detección de movimiento
areaMvtOld = 0  # Área de objeto en movimiento para frames anteriores
centroAreaMvtOld = [0,0]
flagObjStill = 0    # Servirá para considerar que un objeto en movimiento ha pasado a estar
quieto

gruaEnPantalla = 0
gruaMoviendose = 0

# Mascara nula (todo negro)
mascaraVacia = cv2.imread('mascaraVacia.png',0)


# Detecta cosas en movimiento (Grúa amarilla)
# IMPORTANTE: el primer frame debe ser un fondo donde no aparezca nada de lo que se pretende
```

```
detectar
mask_detection = cv2.erode(mask_HSVGrua, None, iterations=2)     # Máscara detección cosas
amarillas
mask_detection = cv2.dilate(mask_detection, None, iterations=2)
cv2.imshow("Mov. Detection grua", mask_detection)

gray = cv2.GaussianBlur(mask_detection, (21, 21), 0)

# Si no existe frame inicial, inicializar
if firstFrame is None:
    firstFrame = gray
    frameBackground = gray

if flagObjStill > 150:  # Nuevo background si el objeto ha estado quieto mucho tiempo
    frameBackground = gray
    flagObjStill = 0
    gruaMoviendose = 0

# Diferencia absoluta entre frame actual y frame inicial
frameDelta = cv2.absdiff(frameBackground, gray)
# Threeshold
thresh = cv2.threshold(frameDelta, 75, 255, cv2.THRESH_TOZERO)[1]
thresh = cv2.dilate(thresh, None, iterations=2)

# Localiza 'contornos' en frame delta
cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
                        cv2.CHAIN_APPROX_SIMPLE)
cnts grua = imutils.grab contours(cnts)
cv2.imshow("Thresh grua", thresh)
cv2.imshow("Frame Delta grua", frameDelta)


# Dibuja contornos de objetos en movimiento (Grúa Amarilla)
(x, y, w, h) = (frame_width,frame_height,0,0)
for c in cnts_grua:
    # Si el contorno es muy pequeño, ignorar
    if cv2.contourArea(c) < min_area:
        continue
    # Dibuja rectangulo que encuadre el contorno

    #(x, y, w, h) += cv2.boundingRect(c) # En caso de que haya varios rectángulos, hacer una
suma de ambos
    if (x > cv2.boundingRect(c)[0]): x = cv2.boundingRect(c)[0]
    if (y > cv2.boundingRect(c)[1]): y = cv2.boundingRect(c)[1]
    w += cv2.boundingRect(c)[2]
    h += cv2.boundingRect(c)[3]

# Determina centro y área del rectángulo, y su variación con respecto al detectado en el frame
anterior
centro = [int(x + w/2), int(y + h/2)]
dCentro = abs(centroAreaMvtOld[0] - centro[0]) + abs(centroAreaMvtOld[1] - centro[1])
area = h*w
dArea = abs(area - areaMvtOld)

if dArea < 400 and dCentro < 4 and areaMvtOld > 0: # Compara si el área y la posición del rect.
ha cambiado respecto frames anteriores
    flagObjStill += 1
else:
    flagObjStill = 0

xcg = abs(int((px2-px1)/2))+px1 # Centro grúa
ycg = abs(int((py2-py1)/2))+py1

if area > 0 and puntoDentroRectangulo((xcg, ycg), (x, y, x+w, y+h)):    # Si la recta detectada
de la grúa está dentro del rectángulo de movimiento, la grúa está moviendose
    gruaMoviendose = 1

areaMvtOld = area
centroAreaMvtOld = centro
#print(flagObjStill, dArea, dCentro)

cv2.rectangle(output, (x, y), (x + w, y + h), (255, 0, 0), 2)
xc = abs(int(w/2))+x
yc = abs(int(h/2))+y
cv2.drawMarker(output, (xc, yc), (0, 0, 255))

# Actualiza estado detección grúa
```

```
if gruaEnPantalla == 1 and gruaMoviendose == 1:
    string = "Grua: en movimiento"
    color = (255, 0, 0)
    frameBackground = firstFrame
elif gruaEnPantalla == 1 and gruaMoviendose == 0:
    string = "Grua: parada"
    color = (255, 254, 51)
else:
    string = "Grua: fuera de pantalla"
    color = (51, 87, 255)
cv2.putText(output, string, (frame_width-200, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0), 4)

cv2.putText(output, string, (frame_width-200, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
```

Código 5–7. Algoritmo de comprobación del estado de la grúa.

6 CONCLUSIONES, LÍNEAS FUTURAS

En este trabajo se ha realizado un primer estudio sobre técnicas de computer vision y machine learning, tratando de entender parte de sus fundamentos teóricos. Se han planteado dos aplicaciones prácticas de reconocimiento e identificación de objetos en imágenes y se ha mostrado la metodología de trabajo seguida para realizarlas.

Por un lado, se ha abordado la aplicación relacionada con el puerto de carga mediante Deep learning, creando una base de datos de imágenes propia y entrenando un modelo pre-entrenado para el reconocimiento visual de objetos. Sin embargo, los modelos de red que se han entrenado no son lo suficientemente precisos, requiriendo de un algoritmo de post-procesado importante. Idealmente se desearía que los modelos hubieran bastado para realizar la tarea; esto podría conseguirse entrenando durante un mayor número de épocas, mejorando la arquitectura de la red y entrenando con una base de datos de imágenes más grande.

Se ha comprobado que trabajar con redes neuronales, sobre todo entrenarlas, requiere recursos y material capacitado (una buena tarjeta gráfica, una CPU capacitada, etc). En el caso de este trabajo, se ha planteado la opción de recurrir a servicios privados de computación en la nube que permitan entrenar modelos de red con mayor velocidad, aunque se prefirió continuar con el equipo descrito.

Dado que el enfoque de la aplicación del puerto comercial es obtener registros logísticos del mismo, una posible dirección en la que continuar el trabajo sería la de contabilizar el tiempo que los barcos pasan atracados en el muelle, la cantidad de contenedores cargados/descargos por las grúas, así como el tiempo que éstas están en funcionamiento, e ir almacenando los datos en un servidor o base de datos.

Otra posibilidad sería la de aplicar Deep learning para ser capaz de reconocer el nombre de los barcos, lo que permitiría hacer comprobaciones en línea de los mismos: rastrear su origen o destino, el tipo de carga que llevan, la ruta de viaje que realizan, etc. Esto seguiría estando ligado a la idea de recabar datos logísticos, lo que facilitaría a la administración del puerto llevar registros, realizar comprobaciones y tomar decisiones.

Por otro lado, el problema de detección en la fábrica de virolas se ha tratado de resolver meramente utilizando algoritmos y técnicas clásicas de computer vision. Funcionalidades como localizar la posición de la grúa, trackear su movimiento o determinar el estado de la instalación se han implementado mezclando técnicas básicas de visión artificial con algoritmos simples de código.

Otras funcionalidades que se podrían implementar pueden ser determinar el estado de funcionamiento de la máquina dobladora que aparece en el vídeo. También puede ser interesante la identificación de trabajadores en la planta, comprobar su posición y detectar infracciones de acceso o determinar posibles riesgos laborales. Desde un enfoque logístico, poder realizar comprobaciones en la capacidad de almacenamiento de la planta, contar el número de virolas en stock o contabilizar el tiempo que éstas pasan estacionadas es otra manera de continuar el trabajo.

Para acabar, se ha concluido que la cámara con la que se graba y/o trasmite la escena y las imágenes tiene un peso importante en proyectos de este tipo. Factores como la resolución y calidad de la cámara, su orientación o la luminosidad del entorno que enfoca, pueden ser problemas si no se tienen en cuenta. En este trabajo, se ha encontrado que la humedad y la orientación del sol podían hacer que la cámara que grababa al puerto estuviese inutilizable en determinados momentos. También se podría pensar que tener control sobre las condiciones del entorno con el que se va a trabajar facilitarían la implementación de las funcionalidades exigidas (pintar de un color reconocible objetos que se quieran identificar, poder obtener un modelo de background libre de obstáculos o poder cambiar la dirección de la cámara a zonas que quieran observarse, por ejemplo).

REFERENCIAS

[1] CHINADAILY, «Automated port terminal in Qingdao handles 790,000 TEUs within first year», [En línea]. Disponible: http://www.chinadaily.com.cn/m/qingdao/2018-05/19/content_36270082.htm

[2] CMOS Image sensor. Wikipedia Commons. https://commons.wikimedia.org/wiki/File:Matrixw.jpg Filya1, 18 March 2009

[3] Universidad de Stanford, 2021. One Hundred Year Study on Artificial Intelligence (AI100). https://ai100.stanford.edu/

[4] Russel S. y Norvig P. (2009). Artificial Intelligence: A Modern Approach. Prentice Hall.

[5] Artificial neural networks and Deep Learning en Torres J. (2018) First contact with Deep Learning, Practical introduction with Keras.

[6] Dominguez Pavón, Sara. (2019). Identificación del modelo de cámara mediante Redes Neuronales Convolucionales. Sevilla, ETSI. (p. 14).

[7] Wikipedia, (2018). Softmax function [en línea]. Available at: https://en.wikipedia.org/wiki/Softmax_function

[8] A. Mikolajczyk y M. Grochowski, "Data augmentation for improving deep learning in image classification problem", 2018 International Interdisciplinary PhD Workshop (IIPhDW).

[9] N. Cui, "Applying Gradient Descent in Convolutional Neural Networks", Electrical and Computer Engineering, University of Massachusetts Lowell, 2018.

[10] Moore, R. C. and DeNero, J. (2011). L1 and L2 regularization for multiclass hinge loss models. In Symposium on Machine Learning in Speech and Natural Language Processing.

[11] Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014a). Dropout: A simple way to prevent neural networks from overfitting. JMLR, 15, 1929–1958.

[12] See https://docs.opencv.org/4.6.0/

[13] Ihaka R, Murrell P, Hornik K, Fisher JC, Stauffer R, Wilke CO, McWhite CD, Zeileis A. Color Spaces: S4 Classes and Utilities. R Project: colorspace package (Article).

[14] The HSV color model mapped to a cylinder. Wikipedia Commons. https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder_saturation_gray.png SharkD, 22 March 2010

[15] Bitwise logical operations. https://docs.opencv.org/4.6.0/d2/d18/group__core__hal__interface__logical.html

[16] Shapiro, L. G. & Stockman, G. C: "Computer Vision", page 137, 150. Prentice Hall, 2001

[17] Canny, J. 1986. A computational approach to edge detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 8(6), pp. 679–698.

[18] Risse, T. 1989. Hough transform for line recognition: Complexity of evidence accumulation and cluster detection. Computer Vision, Graphics, and Image Processing, 46(3), pp. 327–345.

[19] RS Stephens, Probabilistic approach to the Hough transform, Image and Vision Computing, Volume 9, Issue 1, The first BMVC 1990, February 1991, Pages 66-71.

[20] Piccardi, M. (2004). "Background subtraction techniques: A review" (PDF). 2004 IEEE International Conference on Systems, Man and Cybernetics. pp. 3099–3104.

[21] Fleet, David J.; Weiss, Yair (2006). "Optical Flow Estimation" (PDF). In Paragios, Nikos; Chen, Yunmei; Faugeras, Olivier D. (eds.). Handbook of Mathematical Models in Computer Vision. Springer. pp. 237–257.

[22] CUDA Refresher: Reviewing the Origins of GPU Computing. [En línea]. Disponible: https://developer.nvidia.com/blog/cuda-refresher-reviewing-the-origins-of-gpu-computing/

[23] See https://github.com/matterport/Mask_RCNN/releases/tag/v2.0

[24] See https://unsplash.com/

[25] See https://github.com/tzutalin/labelImg

[26] See https://github.com/mukopikmin/bounding-box-augmentation

[27] See https://cocodataset.org/#home

BIBLIOGRAFÍA

Stuart J. Russel and Peter Norvig. (2021). Artificial Intelligence: A Modern Approach, Global Edition [4 ed.]. *Editorial Pearson*

Torres J. (2018). First contact with Deep Learning, Practical introduction with Keras. jorditorres.org

Dominguez Pavón, Sara. (2019). Identificación del modelo de cámara mediante Redes Neuronales Convolucionales. Sevilla, ETSI.

Kenneth D. (2014). A Practical Introduction to Computer Vision with OpenCV. Editorial Wiley

ANEXOS

TrainBarcos.py

```
# Creación de datasets de imágenes

from xml.etree import ElementTree
from os import listdir
from numpy import zeros
from numpy import asarray
from numpy import expand_dims
from numpy import mean
from mrcnn.utils import Dataset
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import compute_ap
from mrcnn.model import load_image_gt
from mrcnn.model import mold_image


# Clase que define y carga el dataset de los barcos
class CargoshipDataset(Dataset):
	def load_dataset(self, dataset_dir, is_train=True):
		# Define una clase
		self.add_class("dataset", 1, "Cargoship")
		# Define localización de las imágenes + anotaciones
		images_dir = dataset_dir + '/Barcos/'
		annotations_dir = dataset_dir + '/Barcos-Annot/'
		# Encuentra todas las imágemes y asigna una ID por el nombre (número)
		for filename in listdir(images_dir):
			image_id = filename[:-4]
			# Salta ids erroneas/vacías
			if (int(image_id) >= 1 and int(image_id) <= 20) or (int(image_id) >= 2090
and int(image_id) <= 2099) or (int(image_id) >= 2490 and int(image_id) <= 2499) or
(int(image_id) >= 2540 and int(image_id) <= 2549):
				continue
			# Sólo incluye hasta un número para el set de entrenamiento
			if is_train and int(image_id) >= 2184:
				continue
			# Salta hasta un número para el set de validación
			if not is_train and int(image_id) < 2184:
				continue
			img_path = images_dir + filename
			ann_path = annotations_dir + image_id + '.xml'
			# Añade imagen al dataset
			self.add_image('dataset', image_id=image_id, path=img_path,
annotation=ann_path)

	# Extrae marcos de los archivos de anotación
	def extract_boxes(self, filename):
		# Carga y analiza el archivo
		tree = ElementTree.parse(filename)
		# Obtiene la raiz del documento
		root = tree.getroot()
		# Extrae cada caja de marcos
		boxes = list()
		for box in root.findall('.//bndbox'):
			xmin = int(box.find('xmin').text)
			ymin = int(box.find('ymin').text)
			xmax = int(box.find('xmax').text)
			ymax = int(box.find('ymax').text)
			coors = [xmin, ymin, xmax, ymax]
			boxes.append(coors)
		# Extrae dimensiones de la imagen
		width = int(root.find('.//size/width').text)
		height = int(root.find('.//size/height').text)
		return boxes, width, height

	# Carga la máscara de una imagen
	def load_mask(self, image_id):
		# Obtiene los detalles de la imagen
		info = self.image_info[image_id]
		# Define el archivo donde se encuentran las anotaciones
```

```
                path = info['annotation']
                boxes, w, h = self.extract_boxes(path)
                # Crea un array para todas las máscaras
                masks = zeros([h, w, len(boxes)], dtype='uint8')
                # Crea las máscaras
                class_ids = list()
                for i in range(len(boxes)):
                        box = boxes[i]
                        row_s, row_e = box[1], box[3]
                        col_s, col_e = box[0], box[2]
                        masks[row_s:row_e, col_s:col_e, i] = 1
                        class_ids.append(self.class_names.index('Cargoship'))
                return masks, asarray(class_ids, dtype='int32')

        # Retorna la dirección de una imagen
        def image_reference(self, image_id):
                info = self.image_info[image_id]
                return info['path']

# Set de entrenamiento
train_set = CargoshipDataset()
train_set.load_dataset('Cargoship', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))

# Set de validación
test_set = CargoshipDataset()
test_set.load_dataset('Cargoship', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))


# ENTRENAMIENTO
# Ignora warnings de librerías obsoletas
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)

# Define la configuración de predicción
class CargoshipConfig(Config):
        NAME = "cargoship_cfg"
        # Num. de clases (background + barco)
        NUM_CLASSES = 1 + 1
        # Numero de pasos por época
        STEPS_PER_EPOCH = 150

# Prepara dataset de entrenamiento
train_set = CargoshipDataset()
train_set.load_dataset('Cargoship', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))
# Prepara dataset de validación
test_set = CargoshipDataset()
test_set.load_dataset('Cargoship', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
# Prepara la configuración
config = CargoshipConfig()
config.display()
model = MaskRCNN(mode='training', model_dir='./', config=config)
# Carga los pesos del modelo pre-entrenado (coco)
model.load_weights('mask_rcnn_coco.h5', by_name=True, exclude=["mrcnn_class_logits",
"mrcnn_bbox_fc",  "mrcnn_bbox", "mrcnn_mask"])
# Entrena el modelo
# NOTA: Puede tardar del orden de 10 horas. Variar el número de épocas y el número de pasos por
época
# puede reducir/aumentar este tiempo a coste de sobreentrenar la red. Es más recomendable
aumentar el dataset antes de
# modificar los parámetros de entrenamiento
model.train(train_set, test_set, learning_rate=config.LEARNING_RATE, epochs=5, layers='heads')


#EVALUACIÓN
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import Dataset
```

```
import matplotlib.pyplot as plt

# Calcula el mAP para un dataset
def evaluate_model(dataset, model, cfg):
        APs = list()
        for image_id in dataset.image_ids:
                # Carga imagen, marcos y máscara para cada ID de imagen
                image, image_meta, gt_class_id, gt_bbox, gt_mask = load_image_gt(dataset, cfg,
image_id, use_mini_mask=False)
                # Convierte valores de píxeles
                scaled_image = mold_image(image, cfg)
                # Convierte imagen a una muestra
                sample = expand_dims(scaled_image, 0)
                # Realiza predicción
                yhat = model.detect(sample, verbose=0)
                # Extrae resultados de la primer muestra
                r = yhat[0]
                # Calcula y almacena estadísticas
                AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"],
r["class_ids"], r["scores"], r['masks'])
                APs.append(AP)
        # Calcula la media de la Average Precision para todas las imágenes
        mAP = mean(APs)
        return mAP, APs

# Define la configuración de predicción
class PredictionConfig(Config):
        NAME = "cargoship cfg"
        # Num. de clases (fondo + barco)
        NUM_CLASSES = 1 + 1
        # Simplifica configuración de GPU
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1

# Prepara el dataset de entretamiento
train_set = CargoshipDataset()
train_set.load_dataset('Cargoship', is_train=True)
train_set.prepare()
print('Train: %d' % len(train set.image_ids))
# Prepara el dataset de evaluación
test_set = CargoshipDataset()
test_set.load_dataset('Cargoship', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
# Define configuración y modelo
cfg = PredictionConfig()
model = MaskRCNN(mode='inference', model_dir='./', config=cfg)
# Carga los pesos del modelo entrenado
model.load_weights('mask_rcnn_cargoship_cfg_0005.h5', by_name=True)
# Evalua modelo para dataset de entrenamiento
[train mAP, train APh] = evaluate model(train_set, model, cfg)
print("Train mAP: %.3f" % train_mAP)
# Evalua modelo para dataset de validación
[test_mAP, test_APh] = evaluate_model(test_set, model, cfg)
print("Test mAP: %.3f" % test_mAP)

plt.figure(0)
plt.plot(train_APh,'r')
plt.plot(test_APh,'g')
plt.xticks(np.arange(0, 11, 2.0))
plt.rcParams['figure.figsize'] = (8, 6)
plt.xlabel("Num of Epochs")
plt.ylabel("Accuracy")
plt.title("Training Accuracy vs Validation Accuracy")
plt.legend(['train','validation'])

plt.figure(1)
plt.plot(snn.history['loss'],'r')
plt.plot(snn.history['val_loss'],'g')
plt.xticks(np.arange(0, 11, 2.0))
plt.rcParams['figure.figsize'] = (8, 6)
plt.xlabel("Num of Epochs")
plt.ylabel("Loss")
plt.title("Training Loss vs Validation Loss")
plt.legend(['train','validation'])

plt.show()
```

```
# DETECCIÓN DE NUEVAS FOTOS
from os import listdir
from xml.etree import ElementTree
from numpy import zeros
from numpy import asarray
from numpy import expand_dims
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.model import mold_image
from mrcnn.utils import Dataset

# Muestra un número de imágenes con la predicción del modelo
def plot_actual_vs_predicted(dataset, model, cfg, n_images=2):
    for i in range(n_images):
                # Carga imágen y máscara
        image = dataset.load_image(i)
        mask, _ = dataset.load_mask(i)
                # Convierte valores de píxeles
        scaled_image = mold_image(image, cfg)
                # Convierte imágen en muestra
        sample = expand_dims(scaled_image, 0)
                # Realiza predicción (inferencia)
        yhat = model.detect(sample, verbose=0)[0]
                # Crea subplot
        pyplot.subplot(n_images, 2, i*2+1)
                # Muestra imagen original
        pyplot.imshow(image)
        pyplot.title('Actual')
        # Muestra marcos originales

        for j in range(mask.shape[2]):
            pyplot.imshow(mask[:, :, j], cmap='gray', alpha=0.3)

                # Crea subplot
        pyplot.subplot(n images, 2, i*2+2)
        # Muestra imagen original
        pyplot.imshow(image)
        pyplot.title('Predicted')
        ax = pyplot.gca()
                # Muestra todos los marcos inferidos por la red
        for box in yhat['rois']:
            y1, x1, y2, x2 = box
                        # Calcula ancho y alto de cada marco, crea una forma rectangular
            width, height = x2 - x1, y2 - y1
            rect = Rectangle((x1, y1), width, height, fill=False, color='red')
            ax.add_patch(rect)
       # Mostrar figura
    pyplot.show()

# Prepara dataset entrenamiento
train_set = CargoshipDataset()
train_set.load_dataset('Cargoship', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))
# Prepara dataset validación
test_set = CargoshipDataset()
test_set.load_dataset('Cargoship', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
# Define config. y modelo
cfg = PredictionConfig()
model = MaskRCNN(mode='inference', model_dir='./', config=cfg)
# Carga pesos del modelo
model_path = 'mask_rcnn_cargoship_cfg_0005.h5'
model.load_weights('mask_rcnn_cargoship_cfg_0005.h5', by_name=True)
model.load_weights(model_path, by_name=True)
# Muestra predicciones para dataset entrenamiento
plot_actual_vs_predicted(train_set, model, cfg)
# Muestra predicciones para dataset validación

plot_actual_vs_predicted(test_set, model, cfg)
```

TrainGruas.py

```
# Creación de datasets de imágenes con marcos de objetos
from xml.etree import ElementTree

# Función para extraer marcos de objetos de archivos de notas
def extract_boxes(filename):
	# Carga y analiza el archivo
	tree = ElementTree.parse(filename)
	# Obtiene la raiz
	root = tree.getroot()
	# Extrae los límites del marco
	boxes = list()
	for box in root.findall('.//bndbox'):
		xmin = int(box.find('xmin').text)
		ymin = int(box.find('ymin').text)
		xmax = int(box.find('xmax').text)
		ymax = int(box.find('ymax').text)
		coors = [xmin, ymin, xmax, ymax]
		boxes.append(coors)
	# Extrae las dimensiones de la imagen
	width = int(root.find('.//size/width').text)
	height = int(root.find('.//size/height').text)
	return boxes, width, height

from os import listdir
from numpy import zeros
from numpy import asarray
from numpy import expand_dims
from numpy import mean
from mrcnn.utils import Dataset
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import compute_ap
from mrcnn.model import load_image_gt
from mrcnn.model import mold_image



# Clase que define y carga el dataset de las grúas
class CraneDataset(Dataset):
	def load_dataset(self, dataset_dir, is_train=True):
		# Define una clase
		self.add_class("dataset", 1, "Crane")
		# Define localización de las imágenes + anotaciones
		images_dir = dataset_dir + '/Gruas/'
		annotations_dir = dataset_dir + '/Gruas-Annot/'
		# Encuentra todas las imágemes y asigna una ID por el nombre (número)
		for filename in listdir(images_dir):
			image_id = filename[:-4]
			# Salta id erronea
			if (int(image_id) >= 180 and int(image_id) <= 189) or (int(image_id) >=
200 and int(image_id) <= 219) or (int(image_id) >= 520 and int(image_id) <= 529):
				continue
			# Sólo incluye hasta un número para el set de entrenamiento
			if is_train and int(image_id) >= 487:
				continue
			# Salta hasta un número para el set de validación
			if not is_train and int(image_id) < 487:
				continue
			img_path = images_dir + filename
			ann_path = annotations_dir + image_id + '.xml'
			# Añade imagen al dataset
			self.add_image('dataset', image_id=image_id, path=img_path,
annotation=ann_path)

	# Extrae marcos de los archivos de anotación
	def extract_boxes(self, filename):
		# Carga y analiza el archivo
		tree = ElementTree.parse(filename)
		# Obtiene la raiz del documento
		root = tree.getroot()
		# Extrae cada caja de marcos
		boxes = list()
		for box in root.findall('.//bndbox'):
```

```
                        xmin = int(box.find('xmin').text)
                        ymin = int(box.find('ymin').text)
                        xmax = int(box.find('xmax').text)
                        ymax = int(box.find('ymax').text)
                        coors = [xmin, ymin, xmax, ymax]
                        boxes.append(coors)
                # Extrae dimensiones de la imagen
                width = int(root.find('.//size/width').text)
                height = int(root.find('.//size/height').text)
                return boxes, width, height

        # Carga la máscara de una imagen
        def load_mask(self, image_id):
                # Obtiene los detalles de la imagen
                info = self.image_info[image_id]
                # Define el archivo donde se encuentran las anotaciones
                path = info['annotation']
                boxes, w, h = self.extract_boxes(path)
                # Crea un array para todas las máscaras
                masks = zeros([h, w, len(boxes)], dtype='uint8')
                # Crea las máscaras
                class ids = list()
                for i in range(len(boxes)):
                        box = boxes[i]
                        row_s, row_e = box[1], box[3]
                        col_s, col_e = box[0], box[2]
                        masks[row_s:row_e, col_s:col_e, i] = 1
                        class ids.append(self.class names.index('Crane'))
                return masks, asarray(class_ids, dtype='int32')

        # Retorna la dirección de una imagen
        def image_reference(self, image_id):
                info = self.image info[image_id]
                return info['path']

# Set de entrenamiento
train_set = CraneDataset()
train_set.load_dataset('Crane', is_train=True)
train set.prepare()
print('Train: %d' % len(train_set.image_ids))

# Set de validación
test_set = CraneDataset()
test_set.load_dataset('Crane', is_train=False)
test set.prepare()
print('Test: %d' % len(test_set.image_ids))


# ENTRENAMIENTO
# Ignora warnings de librerías obsoletas
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)
# Define la configuración de predicción
class CraneConfig(Config):
        NAME = "crane_cfg"
        # Num. de clases (background + grúa)
        NUM_CLASSES = 1 + 1
        # Numero de pasos por época
        STEPS_PER_EPOCH = 150


# Prepara dataset de entrenamiento
train_set = CraneDataset()
train_set.load_dataset('Crane', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))
# Prepara dataset de validación
test_set = CraneDataset()
test_set.load_dataset('Crane', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
# Prepara la configuración
config = CraneConfig()
config.display()
model = MaskRCNN(mode='training', model_dir='./', config=config)
```

```
# Carga los pesos del modelo prototipo (coco) y excluye las capas de salida
model.load_weights('mask_rcnn_coco.h5', by_name=True, exclude=["mrcnn_class_logits",
"mrcnn_bbox_fc",  "mrcnn_bbox", "mrcnn_mask"])
# Entrena el modelo
# NOTA: Puede tardar del orden de 10 horas. Variar el número de épocas y el número de pasos por
época
# puede reducir/aumentar este tiempo a coste de sobreentrenar la red. Es más recomendable
aumentar el dataset antes de
# modificar los parámetros de entrenamiento
model.train(train_set, test_set, learning_rate=config.LEARNING_RATE, epochs=5, layers='heads')


#EVALUACIÓN
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import Dataset
import matplotlib.pyplot as plt

# Calcula el mAP para un dataset
def evaluate_model(dataset, model, cfg):
	APs = list()
	for image_id in dataset.image_ids:
		# Carga imagen, marcos y máscara para cada ID de imagen
		image, image_meta, gt_class_id, gt_bbox, gt_mask = load_image_gt(dataset, cfg,
image_id, use_mini_mask=False)
		# Convierte valores de píxeles
		scaled_image = mold_image(image, cfg)
		# Convierte imagen a una muestra
		sample = expand_dims(scaled_image, 0)
		# Realiza predicción
		yhat = model.detect(sample, verbose=0)
		# Extrae resultados de la primer muestra
		r = yhat[0]
		# Calcula y almacena estadísticas
		AP, _, _, _ = compute_ap(gt_bbox, gt_class_id, gt_mask, r["rois"],
r["class_ids"], r["scores"], r['masks'])
		APs.append(AP)
	# Calcula la media de la Average Precision para todas las imágenes
	mAP = mean(APs)
	return mAP, APs

# Define la configuración de predicción
class PredictionConfig(Config):
	NAME = "crane_cfg"
	# Num. de clases (fondo + grúa)
	NUM_CLASSES = 1 + 1
	# Simplifica configuración de GPU
	GPU_COUNT = 1
	IMAGES_PER_GPU = 1

# Prepara el dataset de entretamiento
train_set = CraneDataset()
train_set.load_dataset('Crane', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))
# Prepara el dataset de evaluación
test_set = CraneDataset()
test_set.load_dataset('Crane', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
# Define configuración y modelo
cfg = PredictionConfig()
model = MaskRCNN(mode='inference', model_dir='./', config=cfg)
# Carga los pesos del modelo entrenado
model.load_weights('mask_rcnn_crane_cfg_0005.h5', by_name=True)
# Evalua modelo para dataset de entrenamiento
[train_mAP, train_APh] = evaluate_model(train_set, model, cfg)
print("Train mAP: %.3f" % train_mAP)
# Evalua modelo para dataset de validación
[test_mAP, test_APh] = evaluate_model(test_set, model, cfg)
print("Test mAP: %.3f" % test_mAP)


# DETECCIÓN DE NUEVAS FOTOS
from os import listdir
from xml.etree import ElementTree
from numpy import zeros
```

```
from numpy import asarray
from numpy import expand_dims
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.model import mold_image
from mrcnn.utils import Dataset

# Muestra un número de imágenes con la predicción del modelo
def plot_actual_vs_predicted(dataset, model, cfg, n_images=2):
    for i in range(n_images):
                # Carga imágen y máscara
        image = dataset.load_image(i)
        mask, _ = dataset.load_mask(i)
                # Convierte valores de píxeles
        scaled_image = mold_image(image, cfg)
                # Convierte imágen en muestra
        sample = expand_dims(scaled_image, 0)
                # Realiza predicción (inferencia)
        yhat = model.detect(sample, verbose=0)[0]
                # Crea subplot
        pyplot.subplot(n_images, 2, i*2+1)
                # Muestra imagen original
        pyplot.imshow(image)
        pyplot.title('Actual')
        # Muestra marcos originales

        for j in range(mask.shape[2]):
            pyplot.imshow(mask[:, :, j], cmap='gray', alpha=0.3)

                # Crea subplot
        pyplot.subplot(n images, 2, i*2+2)
        # Muestra imagen original
        pyplot.imshow(image)
        pyplot.title('Predicted')
        ax = pyplot.gca()
                # Muestra todos los marcos inferidos por la red
        for box in yhat['rois']:
            y1, x1, y2, x2 = box
                        # Calcula ancho y alto de cada marco, crea una forma rectangular
            width, height = x2 - x1, y2 - y1
            rect = Rectangle((x1, y1), width, height, fill=False, color='red')
            ax.add_patch(rect)
        # Mostrar figura
    pyplot.show()

# Prepara dataset entrenamiento
train_set = CraneDataset()
train_set.load_dataset('Crane', is_train=True)
train_set.prepare()
print('Train: %d' % len(train_set.image_ids))
# Prepara dataset validación
test_set = CraneDataset()
test_set.load_dataset('Crane', is_train=False)
test_set.prepare()
print('Test: %d' % len(test_set.image_ids))
# Define config. y modelo
cfg = PredictionConfig()
model = MaskRCNN(mode='inference', model_dir='./', config=cfg)
# Carga pesos del modelo
model_path = 'mask_rcnn_crane_cfg_0005.h5'
model.load_weights('mask_rcnn_crane_cfg_0005.h5', by_name=True)
model.load_weights(model_path, by_name=True)
# Muestra predicciones para dataset entrenamiento
plot_actual_vs_predicted(train_set, model, cfg)
# Muestra predicciones para dataset validación

plot_actual_vs_predicted(test_set, model, cfg)
```

InferenciaPuerto.py

```
from os import listdir
from numpy import zeros
from numpy import asarray
from numpy import expand_dims
from numpy import mean
from mrcnn.utils import Dataset
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import compute_ap
from mrcnn.model import load_image_gt
from mrcnn.model import mold_image
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from mrcnn.config import Config

# Define la configuración de predicción para barcos
class PredictionConfigBarcos(Config):
        NAME = "cargoship_cfg"
        # Num. de clases (fondo + barco)
        NUM_CLASSES = 1 + 1
        # Simplifica configuración de GPU
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1

# Define la configuración de predicción para grúas
class PredictionConfigGruas(Config):
        NAME = "crane_cfg"
        # Num. de clases (fondo + barco)
        NUM_CLASSES = 1 + 1
        # Simplifica configuración de GPU
        GPU_COUNT = 1
        IMAGES_PER_GPU = 1


# Realiza una inferencia y retorna un rectángulo que encuadra el objeto detectado
def inferenciaFrame(image, model, cfg):
                # Convierte valores de píxeles
        scaled_image = mold_image(image, cfg)
                # Convierte imágen en muestra
        sample = expand_dims(scaled_image, 0)
                # Realiza predicción (inferencia)
        yhat = model.detect(sample, verbose=0)[0]

        boxes = numpy.zeros((10,4)) # Es dificil que vaya a detectar más de 10 objetos de cada
clase
                # Muestra todos los marcos inferidos por la red
        i = 0
        for box in yhat['rois']:
            boxes[i] = box
            i = i+1

        estimacion = numpy.zeros((10,1))
        i = 0
        for box in yhat['scores']:
            estimacion[i] = box
            i = i+1

        # Devuelve 5 parámetros: [x1, y1, x2, y2] [score]
        return boxes, estimacion

# Determina si un rectángulo no existe (uno de sus lados es nulo)
def detectaLadoNulo(rectangulo):
    x1 = int(rectangulo[0])
    y1 = int(rectangulo[1])
    x2 = int(rectangulo[2])
    y2 = int(rectangulo[3])
    if (x1 - x2) == 0 or (y1 - y2) == 0:
        return True
    else:
        return False
```

```
# Determina el centro del cuadro de un objeto detectado
def obtieneCentroObjeto(rectangulo):
    x1 = int(rectangulo[0])
    y1 = int(rectangulo[1])
    x2 = int(rectangulo[2])
    y2 = int(rectangulo[3])
    xc = int(abs(x1-x2)/2 + x1)
    yc = int(abs(y1-y2)/2 + y1)
    centro = [xc, yc]
    return centro

# Determina si un punto está dentro de un rectángulo dado
def puntoDentroRectangulo(punto, rectangulo):
    xp = int(punto[0])
    yp = int(punto[1])
    x1 = int(rectangulo[0])
    y1 = int(rectangulo[1])
    x2 = int(rectangulo[2])
    y2 = int(rectangulo[3])
    if (xp > x1) & (xp < x2) & (yp > y1) & (yp < y2):
        return True
    else:
        return False

# Detecta colisión entre marcos Grua-Barco
def detectaColisionBarcoGrua(frameBarco, frameGrua):
    centroBarco = obtieneCentroObjeto(frameBarco)
    centroGrua = obtieneCentroObjeto(frameGrua)

    if (puntoDentroRectangulo(centroBarco, frameGrua) & puntoDentroRectangulo(centroGrua,
frameBarco)):
        # Tanto el centro del barco está dentro del marco de la grúa y el viceversa (hay que
corregir ambos marcos)
        return 3
    elif (puntoDentroRectangulo(centroBarco, frameGrua)):
        # Sólo el centro del barco está dentro del marco de la grúa
        return 2
    elif (puntoDentroRectangulo(centroGrua, frameBarco)):
        # Sólo el centro de la grúa está dentro del marco del barco
        return 1
    else:
        # No hay centros dentro de marcos ajenos
        return 0

# Detecta colisión entre marcos Grua-Barco
def detectaColisionBarcoBarco(frameBarco1, frameBarco2):
    centroBarco1 = obtieneCentroObjeto(frameBarco1)
    centroBarco2 = obtieneCentroObjeto(frameBarco2)

    if (puntoDentroRectangulo(centroBarco1, frameBarco2) & puntoDentroRectangulo(centroBarco2,
frameBarco1)):
        # Ambos centros en mismos marcos
        return 3
    elif (puntoDentroRectangulo(centroBarco1, frameBarco2)):
        # Sólo centro del primer barco dentro de otro marco
        return 2
    elif (puntoDentroRectangulo(centroBarco2, frameBarco1)):
        # Sólo centro del segundo barco dentro de otro marco
        return 1
    else:
        # No hay centros dentro de marcos ajenos
        return 0

# Detecta overlap entre marcos
def detectaOverlap(frame1, frame2):
    if (frame1[0]>=frame2[2]) or (frame1[2]<=frame2[0]) or (frame1[3]<=frame2[1]) or
(frame1[1]>=frame2[3]):
        return False
    else:
        return True

# En la primera inferencia, guarda los datos obtenidos en una lista
def creaTablaGruas(rectGruasPresent):
    i = 0; ID = 0
    while i < rectGruasPresent.shape[0]:
        x1 = int(rectGruasPresent[i][0])
        y1 = int(rectGruasPresent[i][1])
```

```
        x2 = int(rectGruasPresent[i][2])
        y2 = int(rectGruasPresent[i][3])
        if(x1+x2+y1+y2 == 0):    # Campos vacíos (no existe marco)
            i += 1; continue

        area = (x2-x1)*(y2-y1)
        tablaGruas[ID,:] = [ID, x1, y2, x1, y2, area]
        ID += 1
        i += 1

def creaTablaBarcos(rectBarcosPresent):
    i = 0; ID = 0
    while i < rectBarcosPresent.shape[0]:
        x1 = int(rectBarcosPresent[i][0])
        y1 = int(rectBarcosPresent[i][1])
        x2 = int(rectBarcosPresent[i][2])
        y2 = int(rectBarcosPresent[i][3])
        if(x1+x2+y1+y2 == 0):    # Campos vacíos (no existe marco)
            i += 1; continue

        area = (x2-x1)*(y2-y1)
        tablaBarcos[ID,:] = [ID, x1, y2, x1, y2, area]
        ID += 1
        i += 1

def comparaTablaGruas(rectGruasPresent):
    i = 0; ID = 0; num1 = 0; num2 = 0
    while i < rectGruasPresent.shape[0]:
        x1 = int(rectGruasPresent[i][0])
        y1 = int(rectGruasPresent[i][1])
        x2 = int(rectGruasPresent[i][2])
        y2 = int(rectGruasPresent[i][3])
        if(x1+x2+y1+y2 == 0):    # Campos vacíos (no existe marco)
            i += 1; continue
        num1 += 1
        i +=1
    while ID < tablaGruas.shape[0]:
        x1 = int(tablaGruas[ID][1])
        y1 = int(tablaGruas[ID][2])
        x2 = int(tablaGruas[ID][3])
        y2 = int(tablaGruas[ID][4])
        if(x1+x2+y1+y2 == 0):    # Campos vacíos (no existe marco)
            ID += 1; continue
        num2 += 1
        ID += 1
    if(num1 != num2): return 1  # No coincide el número de grúas registradas con las inferidas
    else: return 0

def comparaTablaBarcos(rectBarcosPresent):
    i = 0; ID = 0; num1 = 0; num2 = 0
    while i < rectBarcosPresent.shape[0]:
        x1 = int(rectBarcosPresent[i][0])
        y1 = int(rectBarcosPresent[i][1])
        x2 = int(rectBarcosPresent[i][2])
        y2 = int(rectBarcosPresent[i][3])
        if(x1+x2+y1+y2 == 0):    # Campos vacíos (no existe marco)
            i += 1; continue
        num1 += 1
        i +=1
    while ID < tablaBarcos.shape[0]:
        x1 = int(tablaBarcos[ID][1])
        y1 = int(tablaBarcos[ID][2])
        x2 = int(tablaBarcos[ID][3])
        y2 = int(tablaBarcos[ID][4])
        if(x1+x2+y1+y2 == 0):    # Campos vacíos (no existe marco)
            ID += 1; continue
        num2 += 1
        ID += 1
    if(num1 != num2): return 1  # No coincide el número de grúas registradas con las inferidas
    else: return 0

def actualizaTablaGruas(rectGruasPresent):
    i = 0; ID = 0
    while i < rectGruasPresent.shape[0]:
        x1 = int(rectGruasPresent[i][0])
        y1 = int(rectGruasPresent[i][1])
        x2 = int(rectGruasPresent[i][2])
```

```
        y2 = int(rectGruasPresent[i][3])
        if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
            i += 1; continue

        area = (x2-x1)*(y2-y1)
        tablaGruas[ID,:] = [ID, x1, y1, x2, y2, area]
        ID += 1
        i += 1

def actualizaTablaBarcos(rectBarcosPresent):
    i = 0; ID = 0
    while i < rectBarcosPresent.shape[0]:
        x1 = int(rectBarcosPresent[i][0])
        y1 = int(rectBarcosPresent[i][1])
        x2 = int(rectBarcosPresent[i][2])
        y2 = int(rectBarcosPresent[i][3])
        if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
            i += 1; continue

        area = (x2-x1)*(y2-y1)
        tablaBarcos[ID,:] = [ID, x1, y1, x2, y2, area]
        ID += 1
        i += 1

def copiaTablaGruas():
    ID = 0
    rectGruasPresent = zeros((5,4), dtype = int) # Reinicia la matriz de grúas inferidas
    while ID < tablaGruas.shape[0]:
        x1 = int(tablaGruas[ID][1])
        y1 = int(tablaGruas[ID][2])
        x2 = int(tablaGruas[ID][3])
        y2 = int(tablaGruas[ID][4])
        if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
            ID += 1; continue
        rectGruasPresent[ID] = [x1, y1, x2, y2]
        ID += 1

def copiaTablaBarcos():
    ID = 0
    rectBarcosPresent = zeros((5,4), dtype = int) # Reinicia la matriz de barcos inferidos
    while ID < tablaGruas.shape[0]:
        x1 = int(tablaBarcos[ID][1])
        y1 = int(tablaBarcos[ID][2])
        x2 = int(tablaBarcos[ID][3])
        y2 = int(tablaBarcos[ID][4])
        if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
            ID += 1; continue
        rectBarcosPresent[ID] = [x1, y1, x2, y2]
        ID += 1


# Define config. y modelo barcos
cfgBarcos = PredictionConfigBarcos()
modelBarcos = MaskRCNN(mode='inference', model_dir='./', config=cfgBarcos)
# Carga pesos del modelo
model_path = 'mask_rcnn_cargoship_cfg_0005.h5'
modelBarcos.load_weights('mask_rcnn_cargoship_cfg_0005.h5', by_name=True)
modelBarcos.load_weights(model_path, by_name=True)

# Define config. y modelo grúas
cfgGruas = PredictionConfigGruas()
modelGruas = MaskRCNN(mode='inference', model_dir='./', config=cfgGruas)
# Carga pesos del modelo
model_path = 'mask_rcnn_crane_cfg_0005.h5'
modelGruas.load_weights('mask_rcnn_crane_cfg_0005.h5', by_name=True)
modelGruas.load_weights(model_path, by_name=True)


import imutils, cv2, numpy

# Toma frames de un archivo de video o de una cámara
cap = cv2.VideoCapture('videoBaliza.mp4')

# Comprueba si no se ha podido acceder a la cámara
if (cap.isOpened()== False):
  print("Error opening video stream or file")
```

```
# Resolución del video/cámara
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

# Genera un archivo de vídeo
out = cv2.VideoWriter('inferenciaPuerto.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 25,
(frame_width,frame_height))

# Variable que almacena los recuadros de inferencias de frames anteriores
rectBarcosPresent = zeros((5,4))
rectGruasPresent = zeros((5,4), dtype = int)
countFramesInferencia = 0

# Variable que almacena el número de frames que se han mostrado en total
countFrames = 0

primeraInferencia = 1

# Tabla que registra posiciones de marcos
tablaGruas = zeros((5,6), dtype=int)  # 10 filas: ID, x1, y1, x2, y2, área
tablaBarcos = zeros((5,6), dtype=int)  # 10 filas: ID, x1, y1, x2, y2, área

flagNumeroObjetosGruas = 0   # Sirve para evalúar si ha habido un cambio en el número de cosas
inferidas
flagNumeroObjetosBarcos = 0

outComparacionGruas = 0     # Inicializa variable
outComparacionBarcos = 0

# Lee frames hasta que acaba video o se pulsa Q
while(cap.isOpened()):
  # Captura frame a frame
  ret, frame = cap.read()
  if ret == True:
    output = frame.copy()

    # Realiza inferencias y obtiene recuadros resultante de inferencias cada cierto número de
frames
    # Solo realiza inferencias cada un número determinado de frames para agilizar el proceso
    if countFramesInferencia == 0:
        rectBarcos, scoreBarcos = inferenciaFrame(frame, modelBarcos, cfgBarcos)
        rectGruas, scoreGruas = inferenciaFrame(frame, modelGruas, cfgGruas)
        rectBarcosPresent = rectBarcos
        rectGruasPresent = rectGruas
        scoreBarcosPresent = scoreBarcos
        scoreGruasPresent = scoreGruas

        # POST-PROCESADO
        count = 0
        # Tachado de falsos positivos por baja probabilidad
        while count < rectBarcosPresent.shape[0]:
            if float(scoreBarcosPresent[count]) < 0.90:    # Para una estimación menor del 90%
considerar como falso positivo
                rectBarcosPresent[count] = 0
                count += 1
                continue
            if detectaLadoNulo(rectBarcosPresent[count]):    # No existe el rectángulo o tiene
un lado nulo
                rectBarcosPresent[count] = 0
                count += 1
                continue
            count += 1

        count = 0
        while count < rectGruasPresent.shape[0]:
            if float(scoreGruasPresent[count]) < 0.90:    # Para una estimación menor del 90%
considerar como falso positivo
                rectGruasPresent[count] = 0
                count += 1
                continue
            if detectaLadoNulo(rectGruasPresent[count]):
                rectGruasPresent[count] = 0
                count += 1
                continue
            count += 1

        count = 0
```

```
        # Corrección ante falsos negativos/positivos espontáneos
        if(primeraInferencia):
            creaTablaGruas(rectGruasPresent)
            creaTablaBarcos(rectBarcosPresent)
        else:
            outComparacionGruas = comparaTablaGruas(rectGruasPresent)
            outComparacionBarcos = comparaTablaBarcos(rectBarcosPresent)

            if(outComparacionGruas == 1 or outComparacionBarcos == 1):
                if(flagNumeroObjetosGruas < 3):  # Si no ha ocurrido 3 veces seguidas,
considerarlo como un fallo
                    # No coincide el número de grúas inferidas con las registradas
anteriormente, retiene los marcos de la inferencia anterior
                    ID = 0
                    rectGruasPresent = zeros((5,4), dtype = int) # Reinicia la matriz de grúas
inferidas
                    while ID < tablaGruas.shape[0]:
                        x1 = int(tablaGruas[ID][1])
                        y1 = int(tablaGruas[ID][2])
                        x2 = int(tablaGruas[ID][3])
                        y2 = int(tablaGruas[ID][4])
                        if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
                            ID += 1; continue
                        rectGruasPresent[ID] = [x1, y1, x2, y2]
                        ID += 1
                    flagNumeroObjetosGruas += 1
                else:   # Si se repite 3 veces la discrepancia del número de objetos inferidos,
aceptarlo
                    # Continúa con normalidad
                    i = 0; ID = 0   # Actualiza tabla grúas
                    while i < rectGruasPresent.shape[0]:
                        x1 = int(rectGruasPresent[i][0])
                        y1 = int(rectGruasPresent[i][1])
                        x2 = int(rectGruasPresent[i][2])
                        y2 = int(rectGruasPresent[i][3])
                        if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
                            i += 1; continue

                        area = (x2-x1)*(y2-y1)
                        tablaGruas[ID,:] = [ID, x1, y1, x2, y2, area]
                        ID += 1
                        i += 1
                    flagNumeroObjetosGruas = 0


                if(flagNumeroObjetosBarcos < 3):  # Si no ha ocurrido 3 veces seguidas,
considerarlo como un fallo
                    # No coincide el número de grúas inferidas con las registradas
anteriormente, retiene los marcos de la inferencia anterior
                    ID = 0
                    rectBarcosPresent = zeros((5,4), dtype = int) # Reinicia la matriz de barcos
inferidos
                    while ID < tablaBarcos.shape[0]:
                        x1 = int(tablaBarcos[ID][1])
                        y1 = int(tablaBarcos[ID][2])
                        x2 = int(tablaBarcos[ID][3])
                        y2 = int(tablaBarcos[ID][4])
                        if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
                            ID += 1; continue
                        rectBarcosPresent[ID] = [x1, y1, x2, y2]
                        ID += 1
                    flagNumeroObjetosBarcos += 1
                    print('debug1')
                    print(flagNumeroObjetosBarcos)
                else:   # Si se repite 3 veces la discrepancia del número de objetos inferidos,
aceptarlo
                    # Continúa con normalidad
                    i = 0; ID = 0   # Actualiza tabla barcos
                    while i < rectBarcosPresent.shape[0]:
                        x1 = int(rectBarcosPresent[i][0])
                        y1 = int(rectBarcosPresent[i][1])
                        x2 = int(rectBarcosPresent[i][2])
                        y2 = int(rectBarcosPresent[i][3])
                        if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
                            i += 1; continue
```

```
                        area = (x2-x1)*(y2-y1)
                        tablaBarcos[ID,:] = [ID, x1, y1, x2, y2, area]
                        ID += 1
                        i += 1
                    flagNumeroObjetosBarcos = 0

        # Correción de overlaping barco-grua
        while count < rectBarcosPresent.shape[0]:   # Comprueba todas las combinaciones posibles
barco-barco
            if float(scoreBarcosPresent[count]) < 0.90:    # Para una estimación menor del 90%
considerar como falso positivo
                count += 1
                continue
            if detectaLadoNulo(rectBarcosPresent[count]):    # No existe el rectángulo o tiene
un lado nulo
                count += 1
                continue

            i = 0

            while i < rectGruasPresent.shape[0]:
                if float(scoreGruasPresent[i]) < 0.90:    # Para una estimación menor del 90%
considerar como falso positivo
                    i += 1
                    continue
                if detectaLadoNulo(rectGruasPresent[i]):    # No existe el rectángulo o tiene un
lado nulo
                    i += 1
                    continue
                if detectaColisionBarcoGrua(rectBarcosPresent[count], rectGruasPresent[i]) == 0:
# No hay colisiones
                    i += 1
                    continue

                n = detectaColisionBarcoGrua(rectBarcosPresent[count], rectGruasPresent[i])
                centroBarco = obtieneCentroObjeto(rectBarcosPresent[count])
                centroGrua = obtieneCentroObjeto(rectGruasPresent[i])
                if n == 3:      # Ambos centros dentro de los mismos marcos
                    rectBarcosPresent[count][0] = centroGrua[0]
                    rectGruasPresent[i][2] = centroBarco[0]
                elif n == 2:    # Sólo centro del barco está en el marco de la grúa
                    rectBarcosPresent[count][0] = centroGrua[0]
                    rectGruasPresent[i][2] = centroBarco[0]
                elif n == 1:    # Sólo centro de la grúa está en el marco del barco
                    rectBarcosPresent[count][0] = centroGrua[0]
                    rectGruasPresent[i][2] = centroBarco[0]

                if(detectaOverlap(rectBarcosPresent[count], rectGruasPresent[i])):  # Corrige
overlap barco-grua (vertical)
                    newCoor = (rectBarcosPresent[count][0] + rectGruasPresent[i][2])/2
                    rectBarcosPresent[count][0] = newCoor
                    rectGruasPresent[i][2] = newCoor
                i += 1
            count += 1

        count = 0
        # Corrección de overlaping barco-barco
        while count < rectBarcosPresent.shape[0]:   # Comprueba todas las combinaciones posibles
barco-barco
            if float(scoreBarcosPresent[count]) < 0.90:    # Para una estimación menor del 90%
considerar como falso positivo
                count += 1
                continue
            if detectaLadoNulo(rectBarcosPresent[count]):    # No existe el rectángulo o tiene
un lado nulo
                count += 1
                continue

            i = 0

            while i < rectBarcosPresent.shape[0]:
                if i == count:    # No comparar un barco consigo mismo
                    i += 1
                    continue
                if float(scoreBarcosPresent[i]) < 0.90:    # Para una estimación menor del 90%
considerar como falso positivo
                    i += 1
```

```
                    continue
                if detectaLadoNulo(rectBarcosPresent[i]):    # No existe el rectángulo o tiene
un lado nulo
                    i += 1
                    continue
                if detectaColisionBarcoGrua(rectBarcosPresent[count], rectBarcosPresent[i]) ==
0:  # No hay colisiones
                    i += 1
                    continue

                n = detectaColisionBarcoBarco(rectBarcosPresent[count], rectBarcosPresent[i])
                centroBarco1 = obtieneCentroObjeto(rectBarcosPresent[count])
                centroBarco2 = obtieneCentroObjeto(rectBarcosPresent[i])
                if n == 3:      # Ambos centros dentro de los mismos marcos
                    pass
                elif n == 2:
                    distanciaCentros = abs(centroBarco1[1] - centroBarco2[1])
                    if(centroBarco1[1] < centroBarco2[1]):  # Si está a la izquierda
                        rectBarcosPresent[i][1] = rectBarcosPresent[i][1] + distanciaCentros/1.5
                    elif(centroBarco1[1] > centroBarco2[1]):    # Si está a la derecha
                        rectBarcosPresent[i][3] = rectBarcosPresent[i][3] - distanciaCentros/1.5
                    #pass
                elif n == 1:
                    distanciaCentros = abs(centroBarco1[1] - centroBarco2[1])
                    if(centroBarco1[1] < centroBarco2[1]):  # Si está a la izquierda
                        rectBarcosPresent[count][3] = rectBarcosPresent[count][3] -
distanciaCentros/1.5
                    elif(centroBarco1[1] > centroBarco2[1]):    # Si está a la derecha
                        rectBarcosPresent[count][3] = rectBarcosPresent[count][3] +
distanciaCentros/1.5

                if(detectaOverlap(rectBarcosPresent[count], rectBarcosPresent[i])):  # Corrige
overlap barco-barco (horitonzal)
                    if(centroBarco1[1] < centroBarco2[1]):  # Si está a la izquierda
                        newCoor = (rectBarcosPresent[i][1] + rectBarcosPresent[count][3])/2
                        rectBarcosPresent[i][1] = newCoor
                        rectBarcosPresent[count][3] = newCoor
                    elif(centroBarco1[1] > centroBarco2[1]):    # Si está a la derecha
                        newCoor = (rectBarcosPresent[i][3] + rectBarcosPresent[count][1])/2
                        rectBarcosPresent[i][3] = newCoor
                        rectBarcosPresent[count][1] = newCoor
                i += 1
            count += 1

        if(outComparacionGruas == 0 and outComparacionBarcos == 0): # Actualiza tablas de marcos
procesados
            flagNumeroObjetosGruas = 0
            flagNumeroObjetosBarcos = 0
            i = 0; ID = 0   # Actualiza tabla grúas
            while i < rectGruasPresent.shape[0]:
                x1 = int(rectGruasPresent[i][0])
                y1 = int(rectGruasPresent[i][1])
                x2 = int(rectGruasPresent[i][2])
                y2 = int(rectGruasPresent[i][3])
                if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
                    i += 1; continue

                area = (x2-x1)*(y2-y1)
                tablaGruas[ID,:] = [ID, x1, y1, x2, y2, area]
                ID += 1
                i += 1
            i = 0; ID = 0   # Actualiza tabla barcos
            while i < rectBarcosPresent.shape[0]:
                x1 = int(rectBarcosPresent[i][0])
                y1 = int(rectBarcosPresent[i][1])
                x2 = int(rectBarcosPresent[i][2])
                y2 = int(rectBarcosPresent[i][3])
                if(x1+x2+y1+y2 == 0):   # Campos vacíos (no existe marco)
                    i += 1; continue
                area = (x2-x1)*(y2-y1)
                tablaBarcos[ID,:] = [ID, x1, y1, x2, y2, area]
                ID += 1
                i += 1

        print("Frame:", countFrames)
        print(rectBarcosPresent)
        print(tablaBarcos)
```

```
    if(primeraInferencia):
        primeraInferencia = 0

    countFramesInferencia += 1

    if countFramesInferencia >= 60:
        countFramesInferencia = 0


    # Dibuja recuadros de inferencias de barcos detectados
    count = 0
    while count < rectBarcosPresent.shape[0]:
        if detectaLadoNulo(rectBarcosPresent[count]):    # No existe el rectángulo o tiene un
lado nulo
            rectBarcosPresent[count] = 0
            count += 1
            continue
        x1 = int(rectBarcosPresent[count][0])
        y1 = int(rectBarcosPresent[count][1])
        x2 = int(rectBarcosPresent[count][2])
        y2 = int(rectBarcosPresent[count][3])
        score = scoreBarcosPresent[count]
        [xc, yc] = obtieneCentroObjeto(rectBarcosPresent[count]);
        cv2.rectangle(output, (y1, x1), (y2, x2), (0, 0, 255), 2)
        cv2.drawMarker(output, (yc, xc), (0, 255, 0))
        string = "Cargoship: %.2f %%" % score
        cv2.putText(output, string, (y1+10, x1+20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 5)
        cv2.putText(output, string, (y1+10, x1+20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0),
2)
        string = "ID: %d" % count
        cv2.putText(output, string, (y1+10, x1+45), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 5)
        cv2.putText(output, string, (y1+10, x1+45), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0),
2)
        cv2.imshow("Video + inferencia", output)
        count += 1

    # Dibuja recuadros de inferencias de grúas detectadas
    count = 0
    while count < rectGruasPresent.shape[0]:
        if detectaLadoNulo(rectGruasPresent[count]):
            rectGruasPresent[count] = 0
            count += 1
            continue
        x1 = int(rectGruasPresent[count][0])
        y1 = int(rectGruasPresent[count][1])
        x2 = int(rectGruasPresent[count][2])
        y2 = int(rectGruasPresent[count][3])
        score = scoreGruasPresent[count]
        [xc, yc] = obtieneCentroObjeto(rectGruasPresent[count]);
        cv2.rectangle(output, (y1, x1), (y2, x2), (255, 0, 0), 2)
        cv2.drawMarker(output, (yc, xc), (0, 255, 255))
        string = "Crane: %.2f %%" % score
        cv2.putText(output, string, (y1+10, x1+20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 5)
        cv2.putText(output, string, (y1+10, x1+20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0),
2)
        string = "ID: %d" % count
        cv2.putText(output, string, (y1+10, x1+45), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 5)
        cv2.putText(output, string, (y1+10, x1+45), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0),
2)
        cv2.imshow("Video + inferencia", output)
        count += 1


    string = "Frame: %d" % countFrames
    cv2.putText(output, string, (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 5)
    cv2.putText(output, string, (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
    cv2.imshow("Video + inferencia", output)
    # Escribe nuevo frame en video
    out.write(output)

    # Rompe el bucle si pulsa Q
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    # Termina el proceso una vez se procesen 1500 frames - ELIMINAR PARA UN PROCESADO CONTINUO
    if countFrames >= 1500:
        break
    countFrames += 1
```

```
  # Si no puede obtener más frames, rompe el bucle
  else:
    break

# Libera objetos de captura de vídeo
cap.release()
out.release()

# Cierra todas las ventanas generadas

cv2.destroyAllWindows()
```

GRI.py

```
import math
import time
from os import listdir
from numpy import zeros
from numpy import asarray
from numpy import expand_dims
from numpy import mean
from mrcnn.utils import Dataset
from mrcnn.config import Config
from mrcnn.model import MaskRCNN
from mrcnn.utils import compute_ap
from mrcnn.model import load_image_gt
from mrcnn.model import mold_image
from matplotlib import pyplot
from matplotlib.patches import Rectangle
from mrcnn.config import Config
import imutils, cv2, numpy

# Determina si un punto está dentro de un rectángulo dado
def puntoDentroRectangulo(punto, rectangulo):
    xp = int(punto[0])
    yp = int(punto[1])
    x1 = int(rectangulo[0])
    y1 = int(rectangulo[1])
    x2 = int(rectangulo[2])
    y2 = int(rectangulo[3])
    if (xp > x1) & (xp < x2) & (yp > y1) & (yp < y2):
        return True
    else:
        return False

# Crea videocaptura de un archivo de video o cámara

cap = cv2.VideoCapture('videoGRIgrua3.mp4')

# Comprueba si la cámara es accesible
if (cap.isOpened()== False):
  print("Error opening video stream or file")

# Resolución del video
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))

# Define archivo de vídeo que se va a generar
out = cv2.VideoWriter('outputGRI.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 25,
(frame_width,frame_height))

# Variable que almacena los recuadros de inferencias de frames anteriores
rectVirolasPresent = zeros((5,4))
countFramesInferencia = 0

# Variable que almacena el número de frames que se han mostrado en total
countFrames = 0

# Si en un frame no se ha detectado la grúa se dibujará la del frame anterior
# Si pasado un número de frames sigue sin detectarla, dejará de dibujar la grúa antigua
(posiblemente haya salido de la pantalla)
countFramesGruaOLD = 0
PosGruaOLD = [0, 0, 0, 0]    # Posición de la grúa en el frame anterior
flagGuardaPosGrua = 0
```

```
# Inicializar primer frame del video/stream
firstFrame = None
min_area = 1000 # Área mínima de detección de movimiento
areaMvtOld = 0  # Área de objeto en movimiento para frames anteriores
centroAreaMvtOld = [0,0]
flagObjStill = 0    # Servirá para considerar que un objeto en movimiento ha pasado a estar
quieto

gruaEnPantalla = 0
gruaMoviendose = 0

# Máscara anti-amarillos, facilita detección de grua, elimina falsos positivos
maskAntiAmarillos = cv2.imread('mascaraAmarillos.png',0)
# Carga máscara para detectar ROI railes y dobladora
maskRailes = cv2.imread('mascaraRailes.png',0)
maskDobladora = cv2.imread('mascaraDobladora.png',0)
# Mascara nula (todo negro)
mascaraVacia = cv2.imread('mascaraVacia.png',0)


countLines = 0    # Variable que almacena el número de rectas detectadas en los rieles
railesOcupados = 0  # 0: Rail libre (detecta rectas) 1: Rail ocupado (no detecta rectas)

# Vars. dobladora
firstFrameDobladora = None

# Lee frames hasta que acaba video o se pulsa Q
while(cap.isOpened()):
  # Captura frame a frame
  ret, frame = cap.read()
  if ret == True:
    output = frame.copy()


    # Solo aplica el algoritmo cada cierto intervalo de frames para aliviar la carga
    if countFramesInferencia == 0:
        # Detecta cosas amarillas (grúa)
        frameYeLi = cv2.bitwise and(frame,frame, mask = maskAntiAmarillos)
        cv2.imshow("bitwise amarillo", frameYeLi)
        frameYeLi = cv2.GaussianBlur(frameYeLi, (7, 7), 0)
        hsv = cv2.cvtColor(frameYeLi, cv2.COLOR_BGR2HSV)

        low_yellow = numpy.array([15, 50, 120]) # Máscara para colores amarillos
        up_yellow = numpy.array([35, 255, 255])
        mask_yellow = cv2.inRange(hsv, low_yellow, up_yellow)
        mask_HSVGrua = cv2.GaussianBlur(mask_yellow, (21, 21), 0)
        cv2.imshow("HSV grua", mask_HSVGrua)

        edges = cv2.Canny(mask_HSVGrua, 50, 100)
        linesGrua = cv2.HoughLinesP(edges, 1, numpy.pi/180, 45, maxLineGap=35)
        cv2.imshow("Edges grua", edges)
        flagGuardaPosGrua = 0

        # Aplica máscara que se queda con la zona de los railes de la dobladora
        res = cv2.bitwise_and(frame,frame,mask = maskRailes)
        cv2.imshow("Mascara railes", res)
        frameBlur = cv2.GaussianBlur(res, (9, 9), 0)

        edges = cv2.Canny(frameBlur, 50, 150)
        cv2.imshow("Edges railes", edges)

        linesRailes = cv2.HoughLines(edges, 1, numpy.pi/360, 45)

        # Aplica máscara que se queda con lo que ocurre alrededor de la máquina dobladora
        frameBlur = cv2.GaussianBlur(frame, (9, 9), 0)
        cv2.imshow("Gaussian blur dobladora", frameBlur)

        mask_HSVDobladora = cv2.cvtColor(frameBlur, cv2.COLOR_BGR2HSV)
        low_gray = numpy.array([0, 0, 50]) # Máscara para colores grises
        up_gray = numpy.array([179, 125, 125])
        mask_gray = cv2.inRange(mask_HSVDobladora, low_gray, up_gray)
        mask_HSVDobladora = cv2.GaussianBlur(mask_gray, (9, 9), 0)
        mask_HSVDobladora = cv2.bitwise_and(mask_HSVDobladora,mask_HSVDobladora,mask =
maskDobladora)
        cv2.imshow("HSV dobladora", mask_HSVDobladora)
```

```
        # Detecta cosas en movimiento (Grúa amarilla)
        # IMPORTANTE: el primer frame debe ser un fondo donde no aparezca nada de lo que se
pretende detectar
        mask_detection = cv2.erode(mask_HSVGrua, None, iterations=2)    # Máscara detección
cosas amarillas
        mask_detection = cv2.dilate(mask_detection, None, iterations=2)
        cv2.imshow("Mov. Detection grua", mask_detection)
        gray = cv2.GaussianBlur(mask_detection, (21, 21), 0)
        # Inicializar firstFrame si no existe
        if firstFrame is None:
            firstFrame = gray
            frameBackground = gray

        if flagObjStill > 150:  # Nuevo background si el objeto ha estado quieto mucho tiempo
            frameBackground = gray
            flagObjStill = 0
            gruaMoviendose = 0
        # Calcula diferencia entre frame actual y el de background
        frameDelta = cv2.absdiff(frameBackground, gray)
        thresh = cv2.threshold(frameDelta, 75, 255, cv2.THRESH_TOZERO)[1]
        # Dilata las zonas del resultado anterior, encuentra contornos
        thresh = cv2.dilate(thresh, None, iterations=2)
        cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)
        cnts_grua = imutils.grab_contours(cnts)
        cv2.imshow("Thresh grua", thresh)
        cv2.imshow("Frame Delta grua", frameDelta)

    countFramesInferencia += 1
    if countFramesInferencia >= 30: # 30
        countFramesInferencia = 0

    # Dibuja grúa amarilla detectada mediante Tfa. Hough
    varDist = 0
    px1, px2, py1, py2 = [0, 0, 0, 0]
    if linesGrua is not None:
        for line in linesGrua:
            x1, y1, x2, y2 = line[0]
            longitud = math.sqrt(abs(x1-x2)^2+abs(y1-y2)^2)
            # Calcula longitudes, dibuja la más larga
            if (longitud > varDist):
                varDist = longitud
                px1 = x1
                px2 = x2
                py1 = y1
                py2 = y2
                if not(PosGruaOLD == [0,0,0,0]):    # Hacer promedio entre posición nueva y la
anterior
                    [ox1, oy1, ox2, oy2] = PosGruaOLD
                    px1 = int((px1+ox1)/2)
                    px2 = int((px2+ox2)/2)
                    py1 = int((py1+oy1)/2)
                    py2 = int((py2+oy2)/2)

    if not((px1, py1) == (0,0) or (px2, py2) == (0,0)): # No tiene un punto nulo (en el origen)
        cv2.line(output, (px1, py1), (px2, py2), (0, 255, 0), 5)
        xc = abs(int((px2-px1)/2))+px1
        yc = abs(int((py2-py1)/2))+py1
        string = "Grua"
        cv2.putText(output, string, (xc-20, yc), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 4)
        cv2.putText(output, string, (xc-20, yc), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
        cv2.drawMarker(output, (xc, yc), (255, 0, 0))
        gruaEnPantalla = 1

        countFramesGruaOLD = 0
        if flagGuardaPosGrua == 0:
            PosGruaOLD = [px1, py1, px2, py2]   # Posición de la grúa en el frame anterior
            flagGuardaPosGrua = 1
    elif countFramesGruaOLD < 120 and not(PosGruaOLD == [0,0,0,0]):   # En caso de que no haya
grua que dibujar, dibuja la que había en el frame anterior
        [px1, py1, px2, py2] = PosGruaOLD
        countFramesGruaOLD += 1
        cv2.line(output, (px1, py1), (px2, py2), (0, 255, 0), 5)
        xc = abs(int((px2-px1)/2))+px1
        yc = abs(int((py2-py1)/2))+py1
        string = "Grua"
```

```
        cv2.putText(output, string, (xc-20, yc), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 4)
        cv2.putText(output, string, (xc-20, yc), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255),
2)
        cv2.drawMarker(output, (xc, yc), (255, 0, 0))
        gruaEnPantalla = 1
    else:   # No detecta grúa en la pantalla
        PosGruaOLD = [0,0,0,0]
        countFramesGruaOLD = 0
        gruaEnPantalla = 0

    # Dibuja contornos de objetos en movimiento (Grúa Amarilla)
    (x, y, w, h) = (frame_width,frame_height,0,0)
    for c in cnts_grua:
            # Si el contorno es muy pequeño, ignorar
        if cv2.contourArea(c) < min_area:
            continue
        if (x > cv2.boundingRect(c)[0]): x = cv2.boundingRect(c)[0]
        if (y > cv2.boundingRect(c)[1]): y = cv2.boundingRect(c)[1]
        w += cv2.boundingRect(c)[2]
        h += cv2.boundingRect(c)[3]
    centro = [int(x + w/2), int(y + h/2)]
    dCentro = abs(centroAreaMvtOld[0] - centro[0]) + abs(centroAreaMvtOld[1] - centro[1])
    area = h*w
    dArea = abs(area - areaMvtOld)

    if dArea < 400 and dCentro < 4 and areaMvtOld > 0: # Compara si el área y la posición del
rect. ha cambiado respecto frames anteriores
        flagObjStill += 1
    else:
        flagObjStill = 0

    xcg = abs(int((px2-px1)/2))+px1 # Centro grúa
    ycg = abs(int((py2-py1)/2))+py1

    if area > 0 and puntoDentroRectangulo((xcg, ycg), (x, y, x+w, y+h)):    # Si la recta
detectada de la grúa está dentro del rectángulo de movimiento, considerar que se mueve
        gruaMoviendose = 1

    areaMvtOld = area
    centroAreaMvtOld = centro

    cv2.rectangle(output, (x, y), (x + w, y + h), (255, 0, 0), 2)
    xc = abs(int(w/2))+x
    yc = abs(int(h/2))+y
    cv2.drawMarker(output, (xc, yc), (0, 0, 255))

    # Actualiza estado detección grúa
    if gruaEnPantalla == 1 and gruaMoviendose == 1:
        string = "Grua: en movimiento"
        color = (255, 0, 0)
        frameBackground = firstFrame
    elif gruaEnPantalla == 1 and gruaMoviendose == 0:
        string = "Grua: parada"
        color = (255, 254, 51)
    else:
        string = "Grua: fuera de pantalla"
        color = (51, 87, 255)
    cv2.putText(output, string, (frame_width-200, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 0),
4)
    cv2.putText(output, string, (frame_width-200, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    # Railes dobladora
    frameRailes = frame.copy()
    countLines = 0
    for line in linesRailes:    # Cuenta total de lineas detectadas en los rieles de la
dobladora
        rho,theta = line[0]
        if theta > 0.75 and theta < 1.25:
            a = numpy.cos(theta)
            b = numpy.sin(theta)
            x0 = a*rho
            y0 = b*rho
            x1 = int(x0 + 1000*(-b))
            y1 = int(y0 + 1000*(a))
            x2 = int(x0 - 1000*(-b))
            y2 = int(y0 - 1000*(a))
            cv2.line(frameRailes,(x1,y1),(x2,y2),(0,0,255),2)
```

```
            countLines += 1

    if railesOcupados == 0 and countLines < 15: # Según el número de líneas detectadas,
considerar el raíl libre u ocupado
        railesOcupados = 1
    elif railesOcupados == 1 and countLines > 40:
        railesOcupados = 0

    if railesOcupados == 0:
        string = "Rail: Libre"
        color = (255, 254, 51)
    else:
        string = "Rail: Ocupado"
        color = (255, 0, 0)

    cv2.putText(output, string, (20, frame_height-20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0),
4)
    cv2.putText(output, string, (20, frame_height-20), cv2.FONT_HERSHEY_SIMPLEX, 0.7, color, 2)

    cv2.imshow("Lineas railes", frameRailes)
    cv2.imshow("Video + inferencias", output)

    # Escribe nuevo frame en video
    out.write(output)

    # Termina bucle si pulsa Q
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    countFrames += 1

  else: # Termina bucle si no puede obtener más frames
    break

# Libera objetos de captura de video
cap.release()
out.release()

# Cierra todas las ventanas generadas

cv2.destroyAllWindows()
```