UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

Grado en Ingeniería Informática

Impacto del Ruido Gaussiano y el Brillo en Redes Neuronales de Detección de Objetos Pre-entrenadas

Impact of Gaussian Noise and Brightness on Pretrained Object Detection Neural Networks

Realizado por
Juan Antonio Ángel Ruiz

Tutorizado por
Miguel Ángel Molina Cabello
Rafaela Benítez Rochel

Departamento
Lenguajes y Ciencias de la Computación

MÁLAGA, junio de 2023

UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Grado en Ingeniería Informática

Impacto del Ruido Gaussiano y el Brillo en Redes Neuronales de Detección de Objetos Pre-entrenadas

Impact of Gaussian Noise and Brightness on Pretrained Object Detection Neural Network

Realizado por
Juan Antonio Ángel Ruiz

Tutorizado por
Miguel Ángel Molina Cabello
Rafaela Benítez Rochel

Departamento
Lenguaje y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA

Fecha defensa:

Resumen

El Aprendizaje Profundo aplicado al procesamiento de imágenes y vídeos se trata de una actividad cada vez mas presente en la actualidad. Dentro de esta aplicación de lo que hoy día conocemos como Inteligencia Artificial, destacamos las conocidas redes neuronales para la detección y clasificación de objetos en imágenes, lo cual resulta muy útil en una gran amplitud de campos.

Podemos destacar su uso en el sector sanitario por la gran rapidez con la que se puede procesar elevadas cantidades de datos, siendo muy relevante en el análisis de diferentes pruebas médicas. También encontramos aplicaciones interesantes en áreas como la cartografía, sistemas de información geográficos, conducción autónoma, metalurgia ...

El estudio se centra en la problemática existente por el impacto que provoca determinados factores, como el ruido gaussiano y el brillo en el rendimiento de los diferentes modelos de redes neuronales más comunes.

Una gran cantidad de redes neuronales, se entrenan a través de conjuntos de datos que no tienen en cuenta estos factores, los cuales están presentes en casi cualquier dispositivo que recoja y transmita información. El factor del brillo junto con el ruido afectan al rendimiento de forma negativa, especialmente en las redes neuronales especializadas en tareas como la detección de objetos o patrones.

En la realización de los análisis cualitativos y cuantitativos de los resultados observamos como algunas de las versiones estudiadas en función del área de extensión de los objetos en las imágenes adquieren un comportamiento muy parecido. Mientras que otras versiones de YOLO se centran en tareas en las que se necesite rapidez y eficiencia, ciertas versiones del modelo estudiado de Faster R-CNN, son mucho más precisas, aunque más lentas generalmente.

Palabras clave: aprendizaje profundo, inteligencia artificial, detección, clasificación, YOLO, Faster R-CNN, ruido gaussiano, brillo.

Abstract

Deep Learning applied to image and video processing is an increasingly present activity today. Within this application of what we know today as Artificial Intelligence, we highight the well-known neural networks for the detection and classification of objects in images, which is very useful in a wide range of fields.

We can hihglight its use in the health sector due to the great speed with which large amounts of data can be processed, being very relevant in the analysis of different medical tests. We also find interesting aaplications in areas such as cartography, geographic information systems, autonomous driving, metallurgy ...

The study focuses on the existing problems due to the impact caused by certaing factors, such as gaussian noise and brightness, on the performance of the different models of the most common neural networks.

A large number of neural networks are trained through data sets that do not take into account these factors, which are present in almost any device that collects and transmits information. The brightness factor together with noise noise negatively affect performance, especially in neural networks specialized in tasks such as object or pattern detection.

In carrying out the qualitative and quantitative analysis of the results, we observed how some of the versions studied, depending on the area of extension of the objects in the images, acquire a very similar behaviour. While other versions of YOLO focus on tasks where speed and efficiency are needed, certain versions of the Faster R-CNN model studied are much more accurate, although generally slower.

Keywords: deep learning, artificial intelligence, detection, classification, YOLO, Faster R-CNN, COCO, gaussian noise, brightness.

Índice

1 Introducción

1.1. Motivación

El **Ruido Gaussiano** se asocia a la radiación electromagnética [37]. Por lo tanto, podemos encontrarlo en los datos procesados por cualquier dispositivo, cualquiera de ellos necesita de cierta comunicación eléctrica. Es imposible evitar que este tipo de ruidos afecten a los datos que procesamos en los diferentes dispositivos.

Prácticamente cualquier sensor utilizado para el procesamiento de imágenes se ve afectado por dicho ruido, estos sensores son empleados en una gran cantidad de tareas muy presentes en la sociedad. En cuanto a la aplicación de dichos sensores, en las ciencias de la vida encontramos que se aplican en la realización de imágenes dentales, mamografías digitales, exámenes oftalmológicos para averiguar el estado de los ojos, análisis de patologías...

La verdadera problemática surge cuando las imágenes extraídas de estos sensores se encuentran afectados por este tipo de ruido, provocando alteraciones en las conclusiones extraídas del tratamiento de datos realizado. Concretamente, el **Ruido Gaussiano** dificulta el cometido de ciertos modelos de **redes neuronales** en tareas de **Deep Learning (Aprendizaje Profundo)** [29].

El **Deep Learning** como rama de conocimiento de la **Inteligencia Artificial** esta cobrando últimamente mucho protagonismo ya que se ha demostrado que se puede aplicar a una gran cantidad de ámbitos del mundo que nos conscierne. Los principales elementos utilizados en este campo se tratan de las **redes neuronales** [24]. En principio, las vamos a definir como herramientas que nos permiten realizar tareas de una forma mucho menos compleja, con menos coste temporal y una elevada eficacia en los resultados obtenidos. Las **redes neuronales**, generalmente, son usadas para predicciones de sucesos y simulaciones de cierta complejidad, reconocimiento y clasificación de patrones, monitorización de sistemas, etc [24].

De la gran cantidad de modelos de **redes neuronales** que podemos encontrar, las **Redes Neuronales Convolucionales (CNN)** son en las que nos vamos a centrar. El uso de **Redes Neuronales Convolucionales** se enfoca a actividades como la detección de objetos en imágenes y vídeos, el análisis de imágenes médicas, el procesamiento del lenguaje natural, la elaboración de sistemas de recomendación, etc.

Existen una gran cantidad de modelos diferentes de **Redes Neuronales Convolucionales**, cada modelo posee diferentes versiones del mismo, en las que se mejora la capacidad de aprendizaje y precisión de la red en sí, además, cada modelo se ha diseñado para la realización de una tarea concreta entre las mencionadas anteriormente.

Entre las tareas mas influyentes que pueden llegar a realizar las **Redes Neuronales Convolucionales**, encontramos la detección de objetos y patrones en imágenes [29]. Esto se aplica a entornos de diagnóstico médico, donde la detección de objetos o patrones proveniente de tomografías computadas o resonancias magnéticas es muy útil para elaborar un buen diagnóstico. También podemos ver aplicado su uso en otras tareas más punteras como la conducción autónoma, aquí es fundamental reconocer a los peatones, señales viales y otros vehículos. Otra aplicación utilizada se enfoca en la detección de vehículos para analizar el tráfico de una manera más productiva y sencilla, permitiendo determinar los tramos con riesgos de accidente, las áreas más concurridas, etc [24].

En todas las tareas mencionadas anteriormente intervienen sensores electrónicos los cuales a la hora de digitalizar las imágenes, tal y como hemos mencionado en el comienzo, introducen **Ruido Gaussiano** en estos, provocando en mayor o menor medida una alteración en las inferencias o resultados obtenidos por las redes en cuestión [29]. Esto, a su vez, puede dar lugar a la toma de decisiones erróneas en cualquiera de las aplicaciones de los diferentes ámbitos que hemos mencionado.

La utilización de **Redes Neuronales Convolucionales**, se presentan en nuestro día a día cada vez de una forma más frecuente, pero existen muy pocos estudios que analicen el impacto de determinados factores en el funcionamiento y la eficacia de este tipo de redes. Por ello, es necesario continuar con labores de investigación en este sector ya que mejoraríamos el comportamiento de estas **redes neuronales** ante condiciones adversas y conseguiríamos grandes beneficios a nivel académico, además, en muchas otras áreas, como es el caso del sector sanitario o el sector automovilístico.

1.2. Fundamentos del Problema

El problema particular que vamos a estudiar es acerca del **impacto** que determinados factores provocan sobre el rendimiento de redes neuronales que se basan en **modelos convolucionales**. Tal y como venimos comentando, cuando se recoge información de un sensor electrónico para digitalizar una imagen, esta imagen indiscutiblemente, en mayor o menor medida, se encuentra infectada por **ruido gaussiano**. Esto, junto con las condiciones de **luminosidad** reales en las que se ha tomado la imagen, afecta más de lo que podemos creer al funcionamiento de los diferentes modelos de redes neuronales, llegando a obtener resultados por parte de la red erróneos o no lo suficientemente precisos.

1.3. Objetivos

Esta investigación tiene como finalidad estudiar el impacto que puede llegar a provocar la presencia del **Ruido Gaussiano** en el funcionamiento de **Redes Neuronales Convolucionales** como **YOLO** o **Faster R-CNN**. Estas **redes neuronales** se utilizan para detectar y clasificar objetos en imágenes. Precisamente, vamos a estudiar el efecto en el rendimiento de las mencionadas **redes neuronales**, al introducir **Ruido Gaussiano** en diferentes grados a un conjunto de imágenes determinado.

Los modelos de las **redes neuronales** mencionadas ya se encuentran pre-entrenadas, con el fin de poder observar como afecta a su comportamiento, dicho ruido, a la hora de realizar la detección y la clasificación de los diferentes objetos dados en cada una de las imágenes presentes del conjunto de imágenes del que partimos.

Para valorar el comportamiento de ambos modelos de red ante las diferentes afecciones de ruido, se utilizarán la métrica, **Average Precision (Precisión Promedio)** tanto para la red **YOLO** como para **Faster R-CNN**.

Además de estudiar las consecuencias en el comportamiento de ambos modelos de **red neuronal** por el **Ruido Gaussiano**, se estudiará también el efecto que tiene en su funcionamiento diferentes condiciones de **brillo**. Así podremos elaborar unas conclusiones claras sobre el comportamiento y el impacto que tendría utilizar ambos modelos de **red** ante las mencionadas condiciones adversas de brillo y ruido.

1.4. Estructura del documento

El documento de investigación sigue la siguiente estructura:

- En la introducción, ya hemos visto como se ha contextualizado la rama de conocimiento que se trata en el estudio, además de mencionar la motivación y los objetivos del mismo.
- Por consiguiente, se explican y detallan los fundamentos y los conceptos de los que partimos para la correcta comprensión del estudio. Se mencionan los principios y fundamentos que han encauzado a lo que conocemos como **Inteligencia Artificial**, haciendo especial hincapié en dos de sus ramas, **Deep Learning** y **Machine Learning (Aprendizaje Automático)**, para enmarcar de donde surgen las **Redes Neuronales**, explicándose el cometido de estas, el funcionamiento básico y muchas de sus funcionalidades. Destacamos lo modelos de red **YOLO** y **Faster R-CNN**, que se tratan de las **Redes Neuronales Convolucionales (CNN)** que usaremos a posteriori.
- Seguidamente, vamos a mencionar las tecnologías que usaremos para realizar el estudio, los paquetes y librerías software empleadas, así como cualquier tipo de entorno software utilizado.
- Posteriormente, se indicará la metodología de trabajo que se ha llevado a cabo para realizar la investigación y la empleada para llevar a cabo el desarrollo del código pertinente.
- Después de los puntos mencionados, explicaremos brevemente el conjunto de datos del que partimos, además, se mostrarán y describirán los resultados experimentales obtenidos para las diferentes condiciones de **brillo** y **ruido gaussiano** en ambos modelos de red mencionados. Mostrando los resultados cualitativos y cuantitativos obtenidos en los diferentes experimentos.
- Por último, se exponen las conclusiones del estudio y las diferentes posibilidades, en cuanto a lineas de investigación futuras se refiere.
- En cuanto a los requisitos previos para la instalación del proyecto utilizado para los experimentos, así como su instalación y uso, la encontramos en los archivos **requirements.txt** y **README.md** que se adjuntan junto con este estudio.

2

Estado del arte

2.1. Inteligencia Artificial

Antes de comenzar a hablar sobre la **Inteligencia Artificial**, nos conviene definir y entender lo que hoy conocemos como **inteligencia** desde el punto de vista del ser humano. La inteligencia humana siempre ha sido definida como la **capacidad intelectual** de los seres humanos, pero, ¿qué es la **capacidad intelectual** entonces?. Generalmente la definimos como el grado de potencialidad del desarrollo cognoscitivo de una persona, es decir, el grado de habilidad y desarrollo de los procesos de **análisis, síntesis y generalización** de dicha persona. En resumidas cuentas, podemos sintetizarlo como la habilidad de aprender, los seres humanos cuando absorbemos un nuevo conocimiento, el proceso que hemos llevado a cabo para ello es el de **analizar, sintetizar** y por último, **generalizar**.

La **Inteligencia Artificial (IA)** desde el punto de vista informático no es mas que ese intento o aproximación de dotar a una máquina de inteligencia humana con el objetivo de realizar tareas que ya podemos realizar los humanos, pero con mucha más facilidad y rapidez, además de ser capaz de mejorar en función de la información recopilada. Al fin y al cabo, se trata de un **conjunto de teorías y técnicas** que permiten a las máquina realizar tareas que para los humanos requiere de algún tipo de proceso cognitivo o razonamiento [26].

Parece un concepto bastante actual que cada vez escuchamos más en cualquier conversación a lo largo del día, pero lo que es cierto es que ya fue inventado hace unos 65 años. La expresión se acuño poco después de la Segunda Guerra Mundial, cerca de 1956, fue el prominente informático **John McCarthy** en la **Conferencia de Dartmouth** [26], desde entonces ya se empezaba a plantear al mundo la pregunta **"¿Pueden las máquinas pensar?"**.

Hasta ahora, han sido muchos los avances que se han producido en este campo, desarrollándose competencias y habilidades que parecían impensables en su origen. Un ejemplo claro

que la mayoría de personas utilizamos a diario es el correo electrónico, gracias a la **IA**, la mayoría de servidores de correos actuales son capaces de identificar los correos entrantes de tipo SPAM y moverlos automáticamente a la carpeta correspondiente [9].

Dejando atrás el origen y las diferentes concepciones que podemos encontrar acerca de la **Inteligencia Artificial**, destacamos que actualmente encontramos en ella diferentes ramas intrínsecas que cabe mencionar.

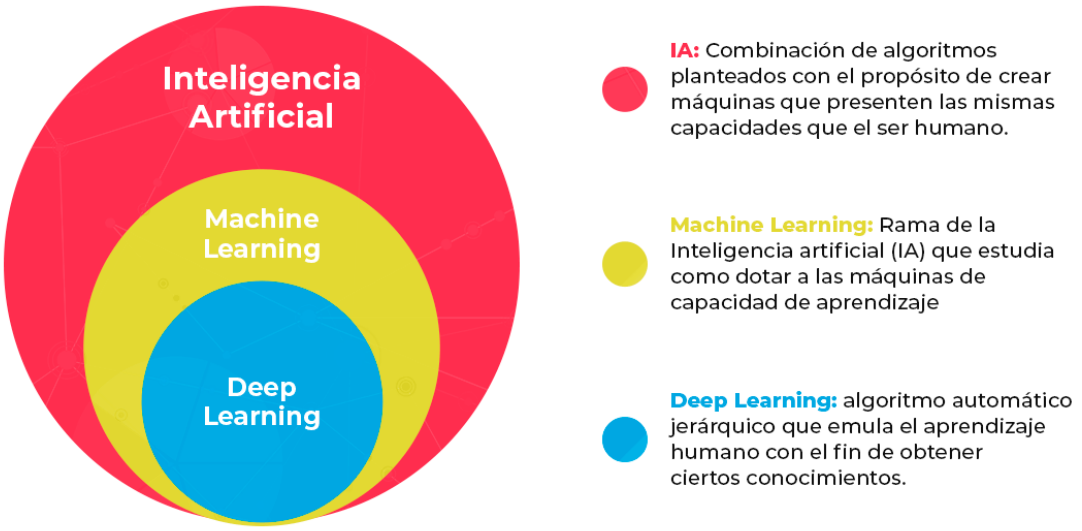


Figura 1: Ramas IA [10]

Como observamos en la figura superior, tanto **Machine Learning** como **Deep Learning** se tratan de ramas o disciplinas que encontramos en la concepción de **Inteligencia Artificial**. Tanto el **Machine Learning** como el **Deep Learning** buscan imitar las capacidades del ser humano en las máquinas.

La disciplina que conocemos como **Machine Learning** se centra en la elaboración y utilización de algoritmos para analizar datos, aprender de esos datos y luego ser capaz de tomar decisiones influenciadas por lo aprendido. En cuanto al **Deep Learning**, se trata de una rama intrínseca del **Machine Learning**, porque la disciplina en sí se enfoca en la producción de algoritmos estructurados en capas, simulando algo parecido a una **red neuronal** humana, la cual podrá aprender y tomar decisiones por sí misma [27].

2.2. Machine Learning y Deep Learning

Lo más común es que se confunda la terminología y el campo de aplicación entre **Machine Learning** y **Deep Learning**.

Como hemos podido observar anteriormente, el **Deep Learning** es parte, como disciplina, del **Machine Learning**. Realmente, el término diferenciador entre ambos es que cuando hablamos de **Machine Learning** se estudia cómo proporcionar capacidad de aprendizaje a una máquina basándose, en general, en algoritmos que sean útiles para encontrar patrones e identificarlos en grandes conjuntos de datos, para luego, poder aprender de ellos. En cambio, al hablar de **Deep Learning**, el concepto se enfoca en implementar algoritmos de aprendizaje mucho mas complejos para aprender patrones específicos y sofisticados de una forma más eficiente, usando **redes neuronales**.

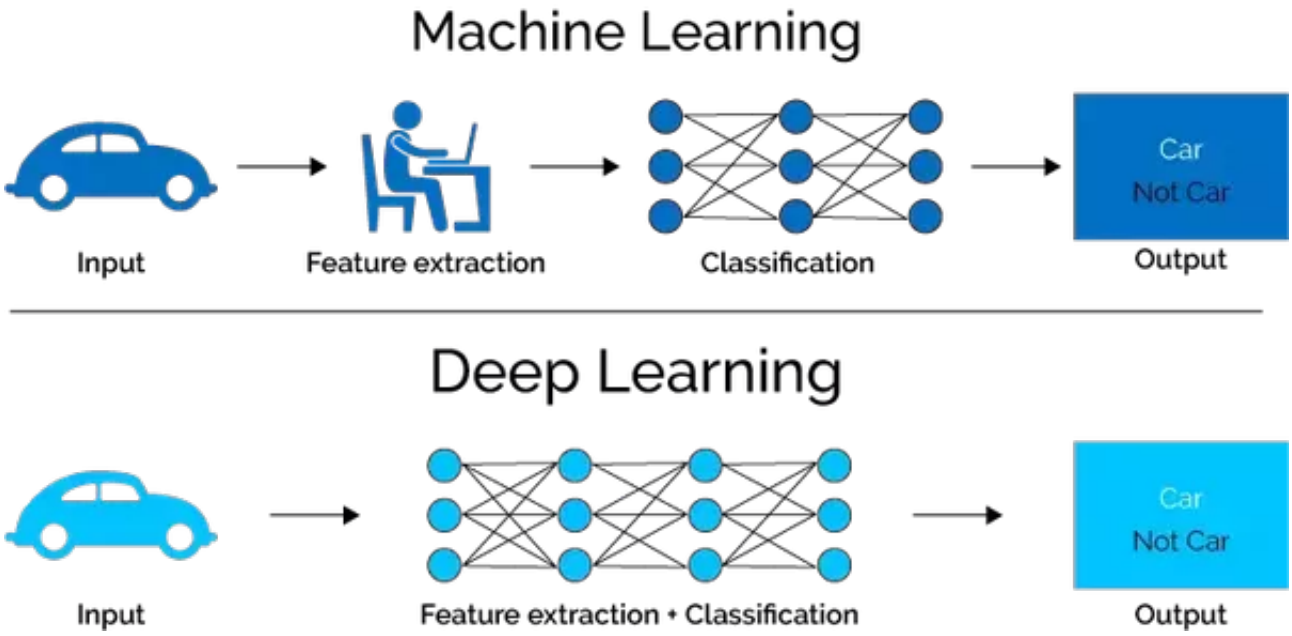


Figura 2: Ejemplo Clasificación de un coche [20]

Como vemos en el ejemplo de la figura, si nuestro objetivo fuese clasificar una imagen de entrada para determinar si se trata de un coche, si hablamos de **Machine Learning** o **Aprendizaje Automático** vemos como se necesitaría de la capacidad humana para extraer las características a tratar por el **Modelo de Clasificación**, que sería el encargado de predecir si se trata la imagen de un coche o se trata de cualquier otra cosa. En cambio, si hablamos de **Deep Learning** o **Aprendizaje Profundo**, el propio **modelo** o **red de clasificación** sería capaz

de aprender de patrones directamente desde la imagen de entrada, ya que el funcionamiento de la red sería más complejo, permitiéndole extraer las características útiles a la imagen de entrada y poder proporcionar una conclusión de la que aprender si se trata de un coche.

2.2.1. Machine Learning

Centrándonos en el **Machine Learning**, sabemos que se tratan de sistemas que son capaces de identificar relaciones en los datos para poder interpretar determinados patrones o comportamientos,consiguiendo **predicciones** lo suficientemente precisas.

A grandes rasgos, la técnica de implementación para el **Machine Learning** se trata de elaborar un **modelo de predicción** donde el error de predicción sea mínimo y así poder generalizar un patrón ya observado. El objetivo al implantar la técnica adecuada es permitir que el **sistema de aprendizaje automático** o **agente** mejore su desempeño con la experiencia a través del uso de datos.

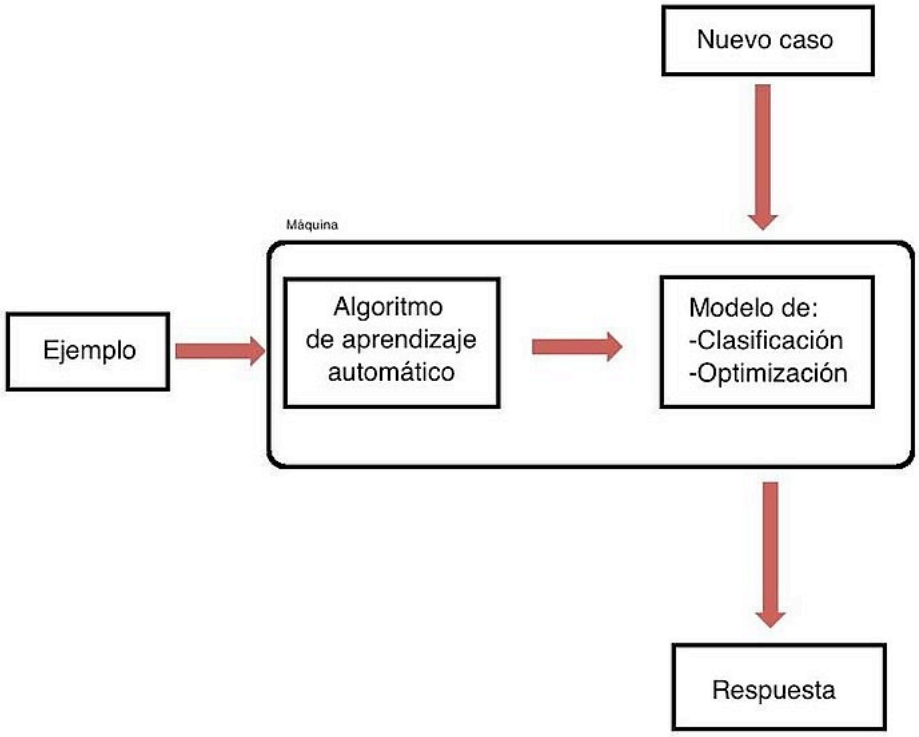


Figura 3: Funcionamiento **Machine Learning**[14]

Para comprender mucho mejor su funcionamiento vamos a dividir en dos componentes fundamentales un sistema que emplee técnicas de **Machine Learning**:

- **Algoritmo de Aprendizaje Automático:** Se trata de un conjunto de instrucciones o reglas que permiten al sistema aprender de los datos sin haber sido programado para ello. El papel que tiene es fundamental, es capaz de desarrollar y modificar modelos, para que estos puedan realizar tareas más específicas como clasificar, predecir, agrupar, etc. En estos algoritmos se suelen utilizar técnicas estadísticas para encontrar mucho más fácilmente relaciones entre los datos.

- **Modelo de Aprendizaje Automático:** Apoyándonos en el elemento anterior definido, se trata de la representación matemática de aquellos patrones y relaciones que han sido inferidas a partir de un conjunto de datos de entrada. Este modelo es generado por un **Algoritmo de Aprendizaje Automático**, el cual es capaz de a través de los datos de entrenamiento ajustar determinados parámetros en el modelo, y así, conseguir que el modelo obtenga predicciones mas certeras sobre nuevos datos. El **Modelo de Aprendizaje Automático** es muy importante para el sistema, ya que será el encargado de generalizar las relaciones obtenidas de los datos para aplicarlas en tareas como la clasificación sobre nuevos datos de entrada.

En cuanto a la forma en cómo aprenden los **sistemas de aprendizaje automático**, existen 4 **tipos de aprendizaje**:

Figura 4: Algoritmos de Machine Learning [28]

- **Aprendizaje Supervisado:** Se trata de un conjunto de técnicas que nos permite realizar predicciones basadas en características o comportamientos analizados en datos ya previamente etiquetados. El **Modelo de Aprendizaje Automático** se entrena para identificar patrones en los datos de entrada para posteriormente relacionarlos con sus etiquetas correspondientes. Con esto, el sistema ya podría hacer predicciones certeras sobre otros nuevos datos [17].

- **Aprendizaje No Supervisado:** Se catalogan como un conjunto de técnicas útiles para comprender datos complejos sin la necesidad de conocimiento previo, es decir, prescindiendo de datos etiquetados. El **Algoritmo de Aprendizaje** identifica los patrones subyacentes en los datos sin la necesidad de que estos ya estuvieran. Al entrenar el **Modelo de Aprendizaje Automático** con los patrones deducidos, conseguimos una representación mas compacta de los datos, identificándolos por grupos, cada cual cumple con unas determinadas características o patrones[17].

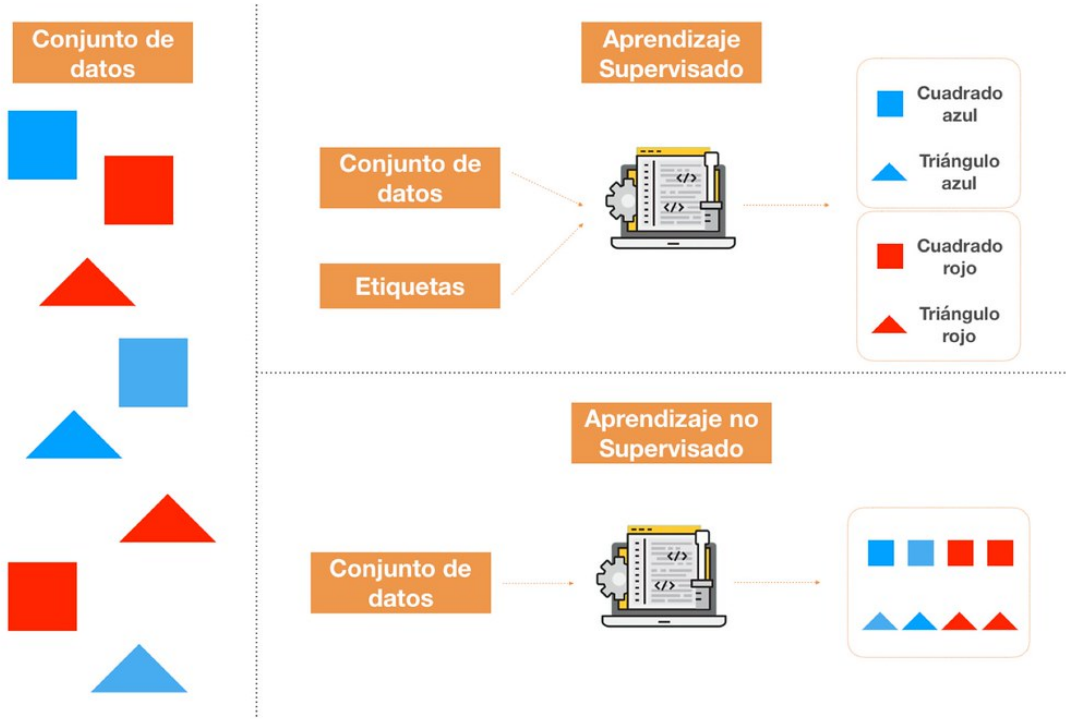


Figura 5: **Aprendizaje Supervisado** y **Aprendizaje No Supervisado** [23]

En la imagen [23], distinguimos la diferencia entre **aprendizaje supervisado** y **aprendizaje no supervisado**. Como vemos el conjunto de datos de entrada del que partimos se trata de una serie de figuras geométricas que varía entre ellas el color y la forma.

Si la técnica de aprendizaje empleada es **Aprendizaje Supervisado**, al **Sistema de Aprendizaje Automático** se le proporcionan el mismo conjunto de datos, además de las etiquetas correspondientes que pretendemos inferir sobre nuevos datos de entrada. En este caso, las etiquetas podrían ser **cuadrado azul**, **cuadrado rojo**, **triángulo azul**, **triángulo rojo**. Cuando el modelo, ya entrenado, realiza la predicción sobre otros datos, es capaz de diferenciar aquellas figuras que son rojas de las que son azules y la forma que las diferencia. En cambio, si la técnica de aprendizaje se trata de **aprendizaje no supervisado**, el conjunto de datos de entrada para el sistema solo consta del conjunto de figuras que vemos, sin etiquetas asociadas. Será el mismo **Algoritmo de Aprendizaje Automático** el que encuentre los patrones en común para clasificar las figuras. En este caso, ha agrupado a los **cuadrados** de forma independiente a los **triángulos**, denotándolos como grupos de datos con características diferentes.

- **Aprendizaje Semi-Supervisado:** El concepto es sencillo de entender, sigue tratándose de una técnica de **Aprendizaje Automático** en la que, como su nombre indica, el conjunto de datos utilizado para el entrenamiento del sistema está **semi-supervisado**, es decir, existe una determinada cantidad de datos etiquetados junto con datos no etiquetados. Se ha demostrado que al utilizarse una pequeña cantidad de datos etiquetados junto con el resto no etiquetados, mejora considerablemente la precisión en las predicciones del modelo [16].

- **Aprendizaje por refuerzo:** Este tipo de aprendizaje se basa en la **experimentación** obtenida de los datos [15]. El sistema es capaz de aprender de su propia experiencia para un entorno determinado. El **Algoritmo de Aprendizaje Automático** elabora unas determinadas interacciones con el entorno, con las cuales es parametrizado el **Modelo de Aprendizaje Automático**. Este último, ejecuta las acciones y retroalimenta al algoritmo positiva o negativamente en función del comportamiento ideal que se busque. Las acciones mencionadas las repetirá y **"reforzará"** según lo positivas o negativas que sean para el fin buscado, hasta encontrar el comportamiento de predicción deseado para el sistema.

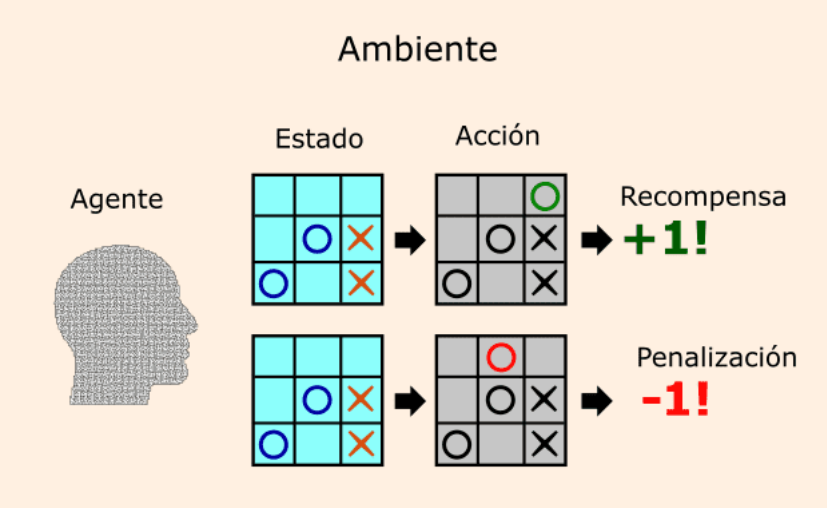


Figura 6: **Aprendizaje Por Refuerzo** [18]

Para comprender mucho mejor como funciona un sistema basado en **Aprendizaje Por Refuerzo** debemos entender los diferentes conceptos que se encuentran involucrados, tal y como vemos en la imagen anterior.

- **Entorno:** Es el ambiente o contexto en el que el **Sistema de Aprendizaje Automático** interactúa, sujeto a una serie de limitaciones.
- **Agente** o **Modelo de Aprendizaje Automático:** Como su nombre indica, se trata del modelo el cual implementa esta técnica de aprendizaje.
- **Estado:** Se trata de uno o varios indicadores en forma de variables que muestran como se encuentra el **entorno** en todo momento.
- **Acciones** o **Interacciones:** Son las decisiones posibles que se pueden realizar en función del **estado** del **entorno**.
- **Recompensas:** Se trata de bonificaciones o ponderaciones positivas que se realizan para determinar cuanto de cerca esta un camino de decisiones tomado por el **agente** sobre el comportamiento que se está buscando.
- **Penalizaciones:** Al igual que las **recompensas**, pero al contrario, se tratan de bonificaciones o ponderaciones negativas para determinar cuanto se aleja un camino de decisiones del comportamiento ideal.

2.2.2. Deep Learning

Como ya hemos mencionado anteriormente, **Deep Learning** en cuanto a su funcionamiento, trata de replicar el cerebro humano, aunque es cierto que aún está bastante lejos de igualar la capacidad de un cerebro humano. Esta disciplina ha avanzado bastante en los últimos años, permitiendo que se elaboren sistemas que son capaces de realizar predicciones con una gran precisión.

El **Deep Learning** se trata una disciplina que figura como subconjunto del **Machine Learning** o **Aprendizaje Automático**. Cuando hablamos de **Deep Learning** o **Aprendizaje Profundo** estamos indirectamente señalando a una **red neuronal**. Estas redes emulan en cierta forma el comportamiento del razonamiento humano, permitiéndoles aprender a partir de grandes cantidades de datos.

Actualmente esta rama del **Machine Learning** ha hecho emerger una buena cantidad de servicios y aplicaciones de **IA**, mejorando la automatización en general en todos los sectores, y en la realización de actividades analíticas y físicas sin la participación del humano.

Las **redes neuronales artificiales** son las estructuras que se emplean como herramientas para conseguir lo mencionado en el párrafo anterior. Una forma resumida de describir una **red neuronal** podría tratarse de una combinación de neuronas. Una **neurona** supone la unidad de procesamiento básica de una **red neuronal**, cada una de las cuales está formada por una serie de entradas de datos, ponderaciones, salidas de datos... Todo en su conjunto es lo que le permite a la red poder reconocer, clasificar y predecir con precisión ciertos objetos dentro de los datos que se le proporcionen.

2.3. Redes Neuronales

Una red neuronal en sentido matemático no es mas que un modelo simple del funcionamiento de nuestro sistema nervioso. Se organizan en **capas**, poseen una serie de **entradas** las cuales recogen la información a tratar, y una serie de **salidas**, donde proporciona la información concluida por la red.

En principio vamos a dejar claro, el funcionamiento y concepto de **neurona**. Lo primero que se nos viene a la cabeza es el concepto biológico de neurona, una unidad de procesamiento de información. Si nos paramos a pensar detenidamente lo que es, se trata al fin y al cabo de una función matemática, con una serie de **entradas** y una o más **salidas**. Internamente esta neurona realiza unos cálculos que vamos a ver más adelante cómo se llegan a obtener.

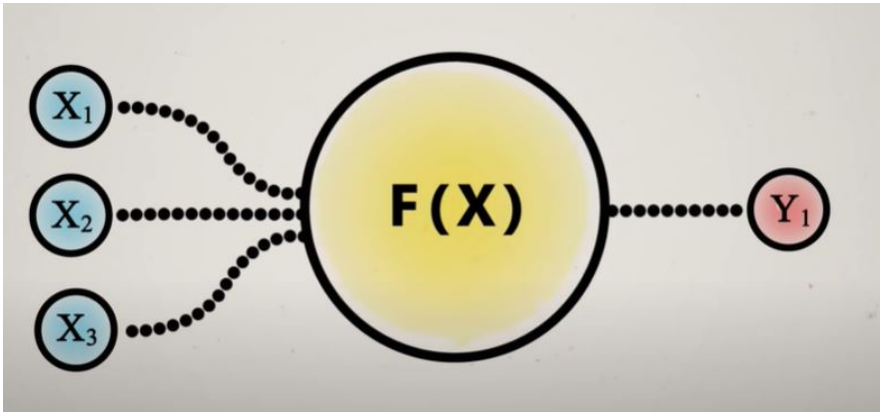


Figura 7: **Neurona Básica** [11]

Para entender mucho mejor, el funcionamiento interno de una neurona, debemos mencionar otro de los elementos más importantes que intervienen en una neurona, los **pesos**. Los **pesos** se tratan de valores numéricos que se asocian a las conexiones entre las neuronas de

una red, con el objetivo de determinar la intensidad con la que afecta cada variable de entrada a la salida de la neurona.

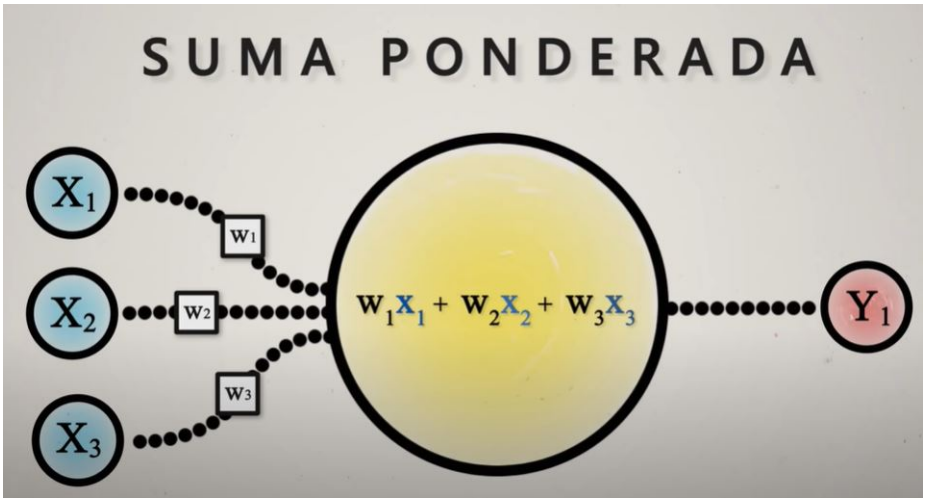


Figura 8: **Suma Ponderada de una Neurona** [11]

Si nos fijamos, realmente una neurona no es más que una forma de expresar una **regresión lineal** para modelar datos.

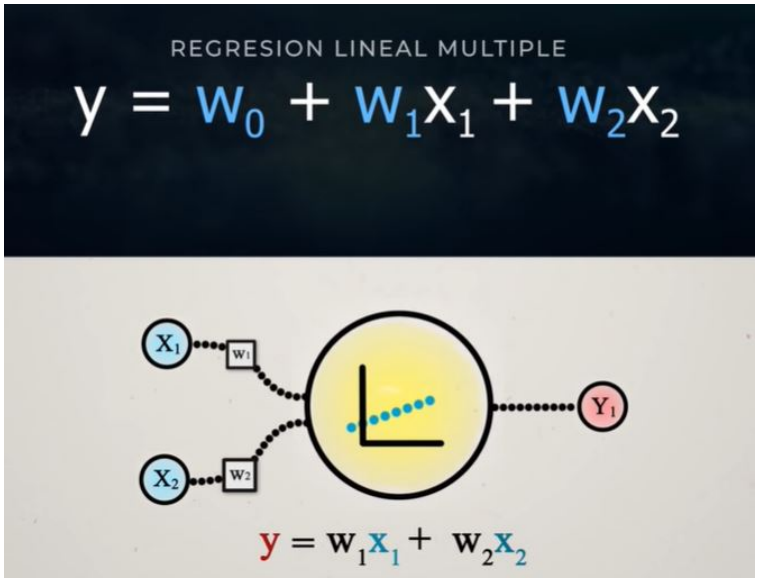


Figura 9: **Neurona - Regresión Lineal** [11]

En toda regresión lineal existe un **término independiente** que nos permite mover verticalmente a la función de regresión. El término independiente se denomina **sesgo** o **bias** en inglés. El **sesgo** se interpreta como una conexión más a la neurona, donde el valor siempre

es 1, pudiendo controlar dicho valor en función de nuestras necesidades. Si unimos a todo lo anterior este nuevo concepto, podemos decir que ahora una neurona podría representar fielmente a un **modelo de regresión lineal**.

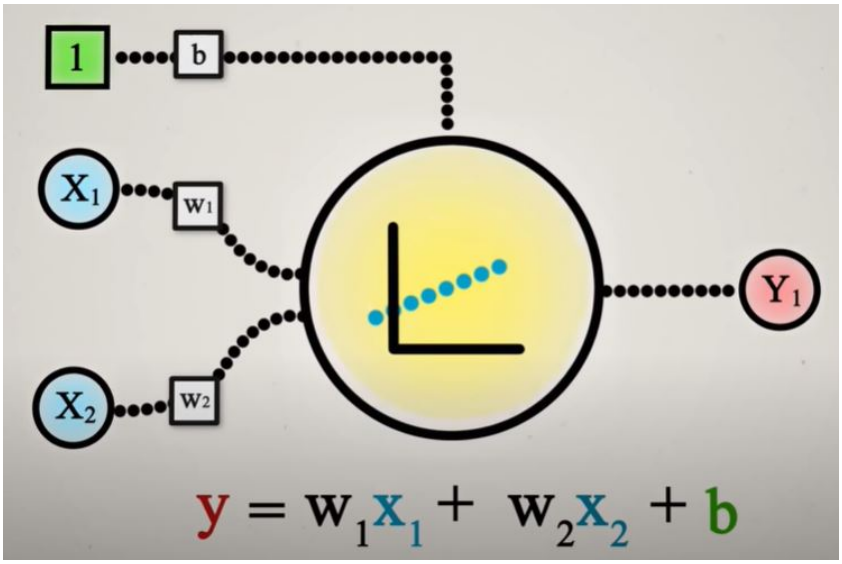


Figura 10: **Neurona - Sesgo** [11]

Un **modelo de regresión lineal** devuelve un valor continuo, si las variables de entrada y la de salida solo cobran un valor binario, ¿cómo podemos convertir esos valores continuos devueltos por el modelo?. Muy sencillo a través de lo que se denomina **umbral**, que en el sentido matemático no es más que el **sesgo** con signo opuesto. De esta forma ya podemos obtener valores discretos de valores continuos.

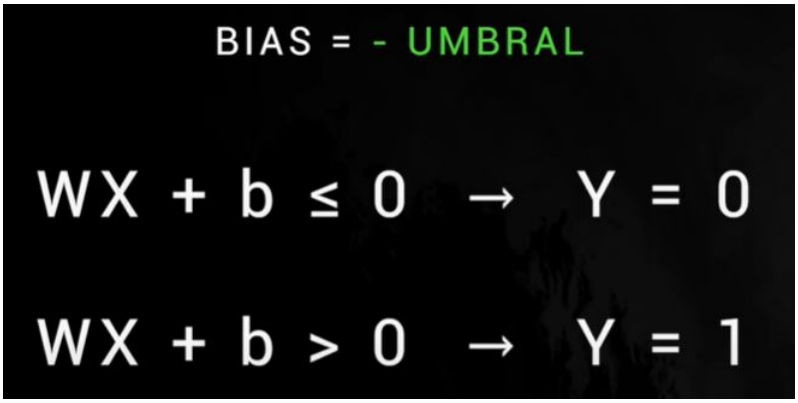


Figura 11: **Sesgo - Umbral** [11]

Una vez introducido conceptualmente el término de **neurona**, la organización de dichas neuronas es un aspecto muy importante en el funcionamiento de una **red neuronal**. Las redes neuronales se dividen en tres capas: **capa de entrada**, una o varias **capas ocultas** y una **capa de salida**. Cada capa tiene un objetivo específico en el procesamiento de la información y en la toma de decisiones de la red.

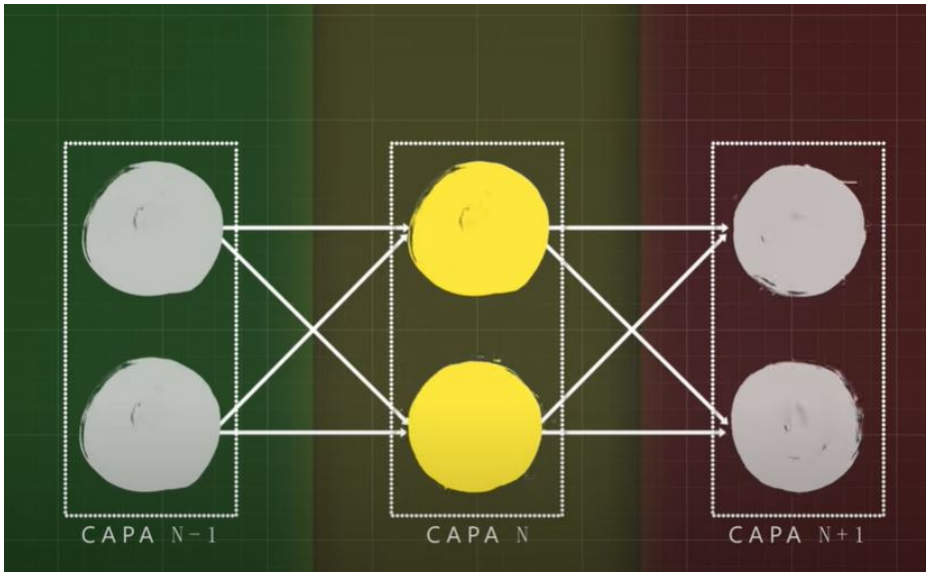


Figura 12: **Red Neuronal - Capas** [12]

- **Capa de entrada:** La capa de entrada recibe los datos de entrada en bruto, pueden tratarse de imágenes, texto o cualquier otra forma de información que se vaya a procesar. En concreto, esta capa no realiza procesamiento alguno, simplemente transmite los datos a la capa siguiente. El número de las neuronas que encontramos en esta capa viene determinado por la dimensión de los datos de entrada.

- **Capas ocultas:** Las capas ocultas se tratan del motor de la red neuronal y se encargan de procesar toda la información recibida por parte de la capa de entrada. En cada una de las **capas ocultas** encontramos una serie de **neuronas**, las cuales se encuentran interconectadas con las neuronas de otra capa oculta. Las **neuronas** que se encuentran en estas capas aplican una **función de activación** a la suma ponderada de las entradas recibidas por parte de la capa anterior. Durante el entrenamiento de la red neuronal estos **pesos** o **ponderaciones** se van ajustando permitiéndole a la red aprender y adaptarse mucho mejor a los datos de entrada.

Además de las mencionadas **ponderaciones**, estas capas, pueden llegar a contener **sesgos**, como hemos mencionado, son valores invariables que se aplican a la suma ponderada de las entradas antes de que se lleve a cabo la **función de activación**. Estos se utilizan para reforzar la red y permitir que haga predicciones mas exactas, al mismo tiempo, mejorando la capacidad de generalización de la red.

- **Capa de salida:** Como su nombre indica y podemos imaginar se trata de la última capa que encontramos en el funcionamiento de una **red neuronal**. La **capa de salida** se encarga de devolver la salida final de la red. En esta capa, las neuronas que encontramos también aplican una **función de activación** a la suma ponderada de las entradas recibidas por las últimas **capas ocultas**. La **función de activación** en esta última capa varía dependiendo del tipo de problema que la red tenga previsto resolver, encontrándose una gran variedad de **funciones de activación**, cada una para un fin diferente.

  Entonces, ¿por qué es necesario colocar las neuronas de forma secuencial alimentando unas a otras?. Esto se realiza para que la red pueda aprender lo que se denomina **conocimiento jerarquizado**.

  Para entender mucho mejor lo que significa **conocimiento jerarquizado**, pensemos en que queremos predecir si vamos a **pasar una noche entretenida** en función de si esa noche **jugamos a videojuegos** y **comemos nachos**.

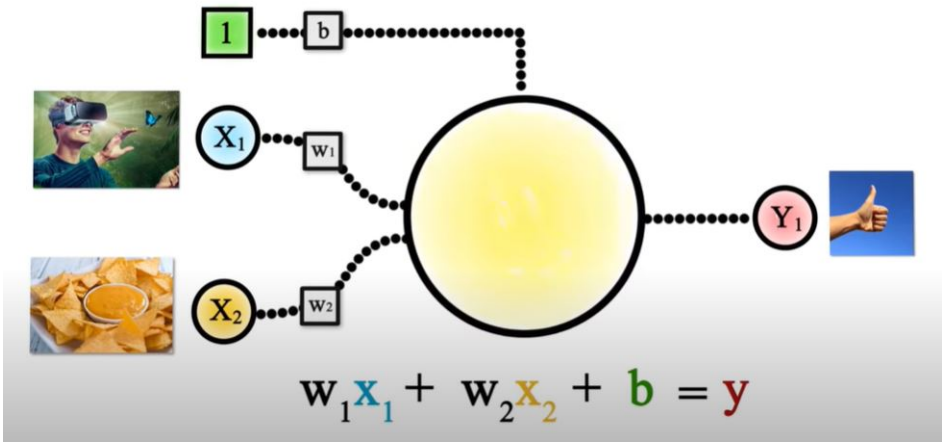


Figura 13: **Ejemplo - 1 Neurona** [12]

Ahora imaginemos que con la salida proporcionada por la neurona anterior queremos saber si **aprobaremos el examen de la semana que viene**. Si añadimos dos **variables de entrada** más, una que se trate de proporcionar información acerca de si **la materia del examen es interesante** y otra que informe sobre la **dificultad del examen**. Podemos modificar los parámetros anteriormente definidos para cada neurona, para especializar cada neurona en un cometido.

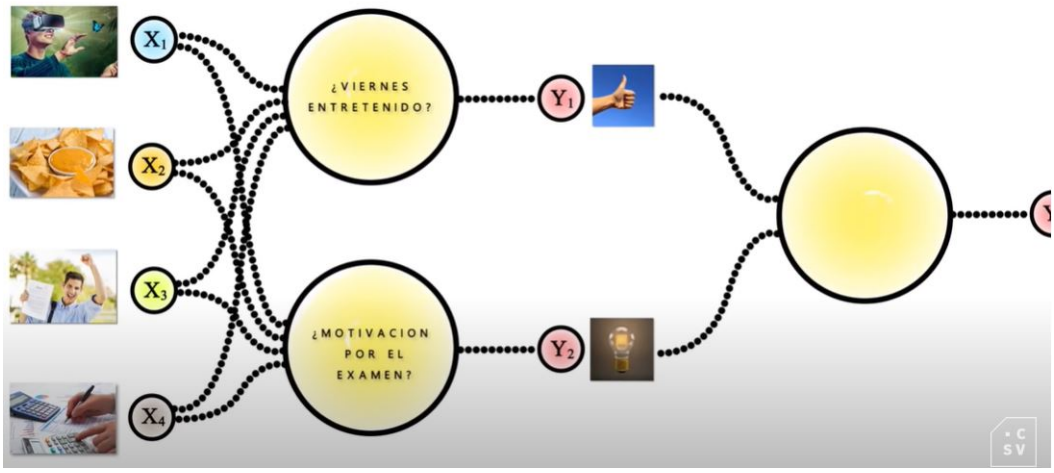


Figura 14: **Ejemplo - 3 Neuronas** [12]

Una de ellas se especializará en determinar si tuvimos un viernes entretenido y otra en determinar la motivación que tenemos de cara al examen. Luego esta información será proporcionada como entrada a la tercera neurona, la cual se encargará determinar como será el **resultado de nuestro examen**.

Con este ejemplo concluimos, que la conexión secuencial de las neuronas de una capa con otra, junto con la configuración de las mismas capas, nos permitirá generar redes neuronales para poder elaborar conocimiento más complejo.

Hasta aquí todo parece funcionar perfectamente, pero si nos paramos a pensar que desdes el punto de vista matemático concatenar neuronas de esta forma no es más que ir concatenando **regresiones lineales**, el efecto del resultado será otra **regresión lineal**. Es decir, esto significa que todas las neuronas conectadas es como si colapsasen en una sola neurona. Esto supone un problema, ya que existirían problemas no lineales que no podríamos llegar a resolver.

Para resolver el problema anterior, se utilizan las **funciones de activación**, que nos permitirán aportar esa no-linealidad a la regresión que realiza cada neurona.

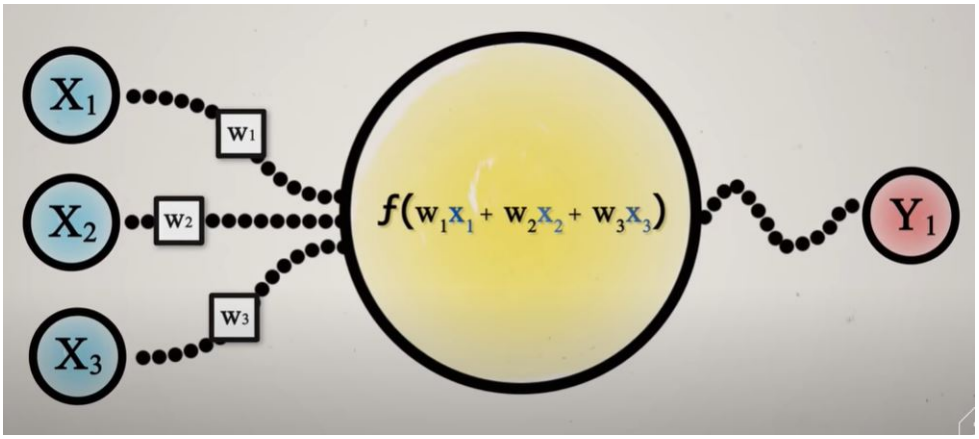


Figura 15: **Neurona - Función de Activación** [12]

De esta forma podemos encadenar de forma efectiva la computación de cada una de estas neuronas y llegar a abordar problemas mucho mas complejos. Dicho de otra forma la función de activación nos permite activar o disparar la neurona en función de la entrada que recibe.

Recordemos que la **función de activación** se aplica a la suma ponderada interna de cada neurona, existiendo muchos tipos de **funciones de activación** (escalonada, sigmoide, ReLU, softmax ...)

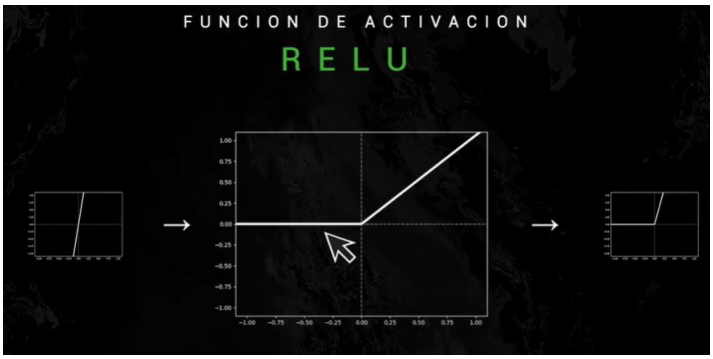


Figura 16: **Función de Activación Escalonada** [12]

En cuanto a la **función ReLU**, simplemente nos permite activar la neurona cuando el resultado de la suma ponderada es mayor que 0 , devolviendo dicho resultado de la **suma ponderada** y desactivarla cuando la suma ponderada es menor que 0. Comportándose

como una función lineal cuando el resultado de la **suma ponderada** es mayor que 0 y como una función constante a 0 cuando el valor de la **suma ponderada es negativo**.

2.3.1. Redes Convolucionales

Una vez ya sabemos como funciona de forma conceptual una **red neuronal**, podemos empezar a hablar sobre el tipo de redes que vamos a tratar en esta investigación. Las **redes neuronales convolucionales** basan su funcionamiento en como funciona la corteza visual del cerebro humano, a la hora de detectar bordes, formas y por consiguiente objetos. Estos conceptos biológicos los aplicamos a la visión por computador con el objetivo de replicar el funcionamiento de nuestro cerebro.

Para entender mucho mejor el concepto de este tipo de red, imaginemos que nos colocamos delante una imagen de una cara, ¿cómo sabemos que es una cara?, ¿por qué lo sabemos?.

Figura 17: **Ejemplo Imagen - Cara** [36]

Sencillamente, sabemos que es una cara porque gracias a nuestro aprendizaje detectamos que hay elementos que pertenecen o se asocian a una cara, como los ojos, la nariz, la boca ...

Si abstraemos más estos elementos y nos preguntamos, ¿por qué se que es un ojo?. Porque un ojo a su vez está conformado por una pupila negra, superficies blancas, pestañas ...

Si seguimos abstrayendo y nos preguntamos el motivo por el cual somos capaces de reconocer lo que es una pupila negra, es debido a que somos capaces de detectar cambios de contraste, patrones circulares, texturas ...

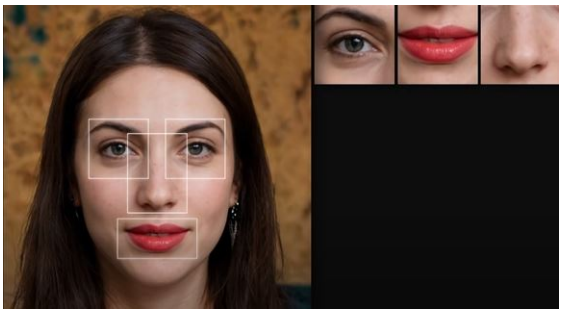


Figura 18: **Ejemplo Imagen - Elementos de una cara** [36]

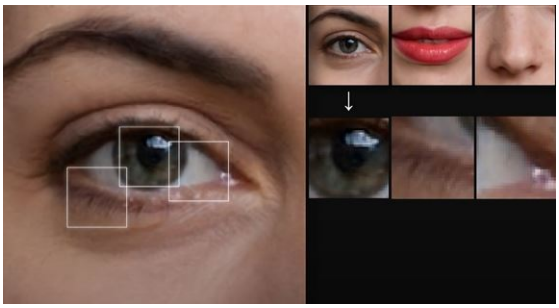


Figura 19: **Ejemplo Imagen - Elementos de un ojo** [36]

Este procedimiento es que realiza nuestro cerebro, es un proceso en cascada donde en principio a la hora de reconocer o detectar una imagen o patrón, primero analiza los elementos generales o más básicos que se van combinando para obtener patrones mas complejos como detectar la cara de una persona en este caso.

Si pasamos a analizar entonces como simula este comportamiento una **red neuronal convolucional**, se basa simplemente en analizar la **estructura espacial** de la imagen. Este análisis de la **estructura espacial** consiste en aprovechar y analizar la localización de un píxel de la imagen junto con la relación que guarda con los píxeles vecinos. A través de esta técnica podemos obtener y determinar esas formas o patrones que va a ayudar a la **red neuronal** a determinar el objeto que esta procesando.

La técnica que se utiliza para operar con estos píxeles y aislar los patrones que nos interesan para determinar el objeto se denomina **convolución**. Cuando hablamos de aplicar una **convolución** a una imagen estamos haciendo referencia a la alteración de los valores de los píxeles de la imagen para obtener otra nueva imagen en la que cada píxel se ha visto alterado por un **filtro** particular.

La operación consiste en alterar el valor de cada píxel multiplicando los píxeles vecinos,

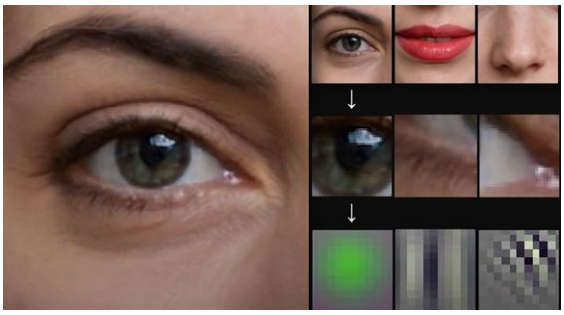


Figura 20: **Ejemplo Imagen - Elementos de una pupila** [36]

Figura 21: **Convolución - Filtro** [36]

incluido el propio píxel, por los valores especificados en el **filtro**. Luego sumamos estas operaciones realizadas y el valor resultante se lo asignamos al píxel procesado.

El procedimiento anterior lo aplicamos desplazando el filtro por cada uno de los píxeles de la imagen, dando lugar a otra imagen diferente. Esto significa que en función de como configuremos los parámetros de nuestro filtro podremos obtener un resultado u otro.

Por ejemplo, en la figura superior, tenemos el ejemplo de un **filtro** configurado para que los valores de los píxeles de la columna izquierda multipliquen en negativo y los de la columna derecha multipliquen en positivo, cuando el filtro se sitúa en una zona donde los colores son iguales, los términos en positivo y negativo se cancelarán como en el primer caso. En cambio, para valores contrarios, como es el segundo caso de la imagen, se llegará a sumar más.

Este **filtro** se activará cuando encuentre diferencias de contraste en una imagen, permitiendo detectar bordes verticales en una imagen.

Como ya podemos imaginar, en función del **filtro** que utilicemos podremos concluir un

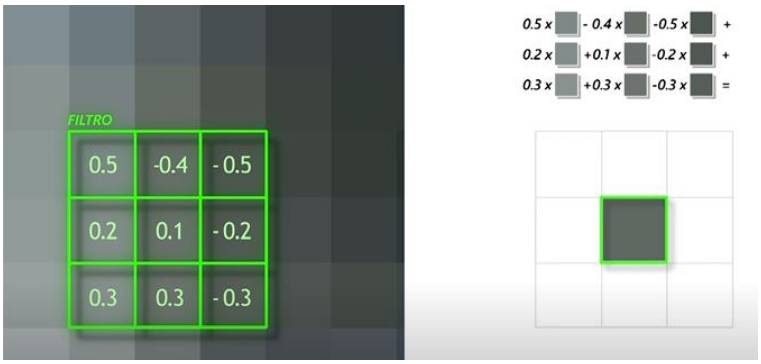


Figura 22: **Convolución - Operación con Filtro** [36]

patrón o característica de la imagen que estemos analizando, por consiguiente, los valores de los diferentes **filtros** de la red son los que ella misma irá modificando poco a poco para entrenarse en el reconocimiento de unas determinadas características. El principal objetivo de la **red neuronal convolucional** será a aprender los mejores valores para cada uno de los filtros que se apliquen a la hora de detectar los diferentes patrones.

Cada una de las imágenes generadas después de aplicar el filtro a la imagen, se le conoce como **mapa de características**, como su propio nombre indica se trata de un mapa, donde se nos indica la zona de la imagen con la característica o patrón buscado por dicho filtro. Cada píxel blanco en el **mapa de características** indica una activación de que el elemento o patrón buscado se encuentra en esa zona de la imagen.

Para concluir, el funcionamiento de un red convolucional, debemos destacar que una vez entra una imagen en la red, se aplica una serie de **convoluciones**, luego se genera un conjunto de **mapas de características** que a su vez se convertirán en la entrada de la siguiente capa de la red, realizando el mismo procedimiento **convolucional** de forma secuencial.

Imaginemos que aplicamos a la imagen original 16 filtro diferentes, cada uno con un objetivo distinto. Después de realizar la **convolución**, obtendremos 16 mapas de características que serán la entrada de la siguiente capa convolucional, tal y como vemos en la figura anterior.

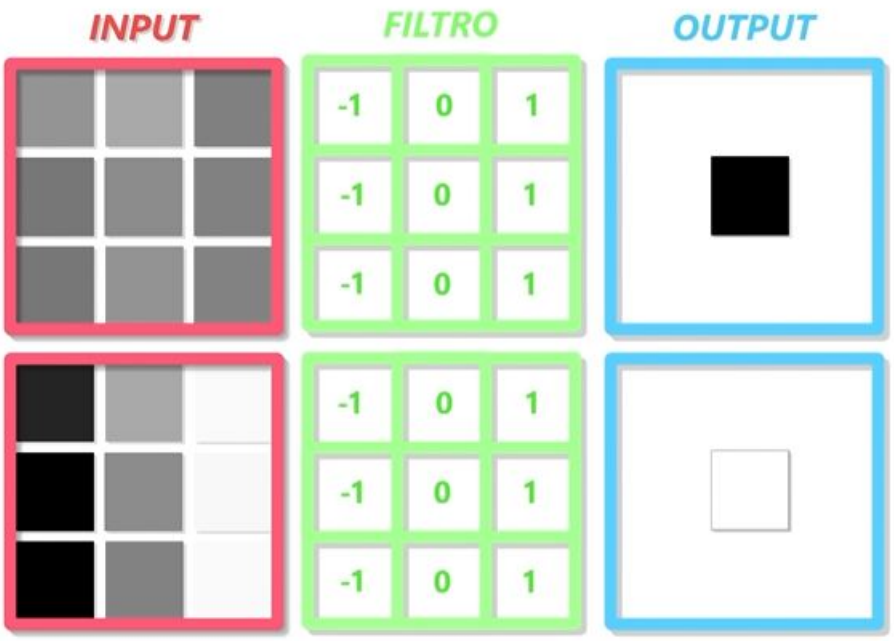


Figura 23: **Convolución - Ejemplo Filtro** [36]

Con estas operaciones, conseguiremos acceder cada vez a mas información espacial acerca de la imagen original. Estos **mapas de características** se utilizan a su vez para formar y deducir formas más complejas a partir de los patrones básicos obtenidos anteriormente, gracias a lo que la red podría detectar y reconocer patrones u objetos en la imagen original de entrada.

2.3.2. YOLO

YOLO (You Only Look Once) se trata de un algoritmo de **inteligencia artificial** de código abierto usado generalmente para **detección de objetos**. La particularidad de este método de **IA** es que solo hace uso de una única **red neuronal convolucional** para llevar a cabo su objetivo. **YOLO** es capaz de aprender representaciones generales de objetos, dando lugar un error muy bajo a la hora de detectar nuevas entradas, distintas a las que se usaron para el entrenamiento de la red [13].

Cuando hablamos de **detección de objetos**, **YOLO** resuelve este problema basándose principalmente en dos conceptos:

- **Categoría del objeto:** Se trata de la clase de objeto que estamos tratando de detectar en una imagen, por ejemplo, un perro podría ser una categoría presente en un modelo

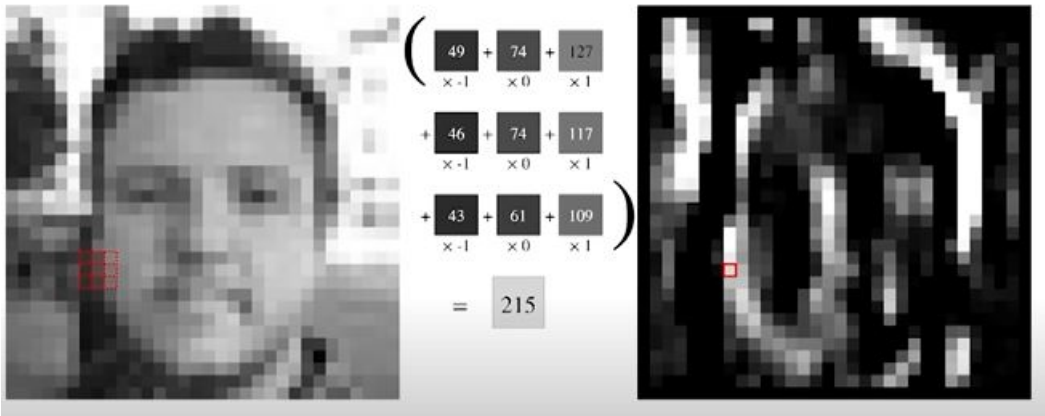


Figura 24: **Convolución - Filtro Bordes Verticales** [36]

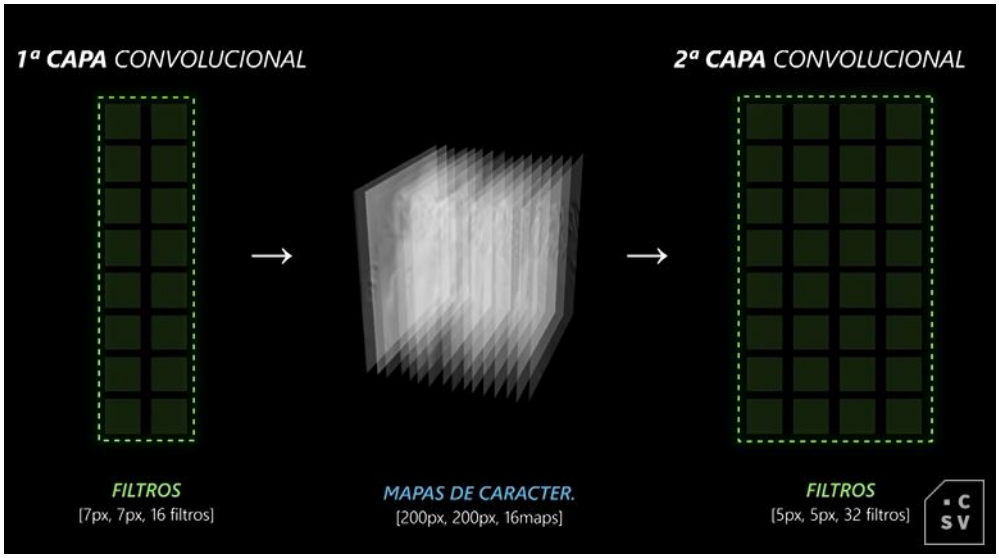


Figura 25: **Proceso entre Capas Convolucionales** [36]

YOLO.

- **Localización del objeto:** Como el propio nombre indica, hace referencia a la posición y región ocupada por dicho objeto en la imagen. En el caso de esta investigación, localizamos los objetos a través de los denominados **bounding boxes (cuadros delimitadores)**. Estos elementos no son más que un rectángulo que dibuja **YOLO** en la región donde predice que esta el objeto. Existen muchas formas de indicar la localización de **bounding box** en la imagen, la que se va a utilizar en esta investigación está definida por un punto (x,y) o coordenada extremo superior izquierda del **bounding box** y además el **ancho (width) y alto(height)** del mencionado rectángulo formado.

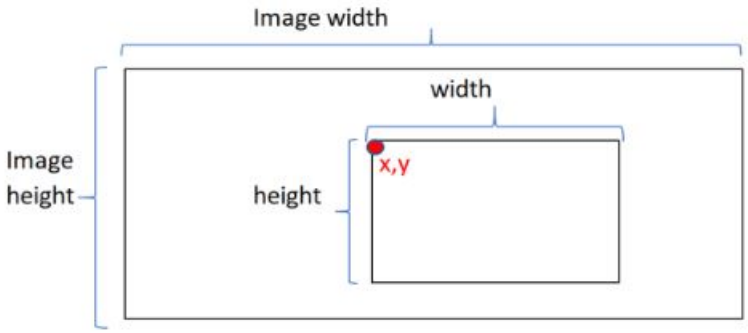


Figura 26: **YOLO - Tipo Bounding Box** [22]

- **Probabilidad de la predicción:** Se trata de la forma que tiene **YOLO** de puntuar las predicciones conseguidas, no es mas que una probabilidad que expresa el acierto de la clase asignada por **YOLO** a un objeto en la detección de una imagen.

Una vez comprendido los conceptos necesarios para entender el funcionamiento de **YOLO**, mostramos un modelo de arquitectura simplificada:

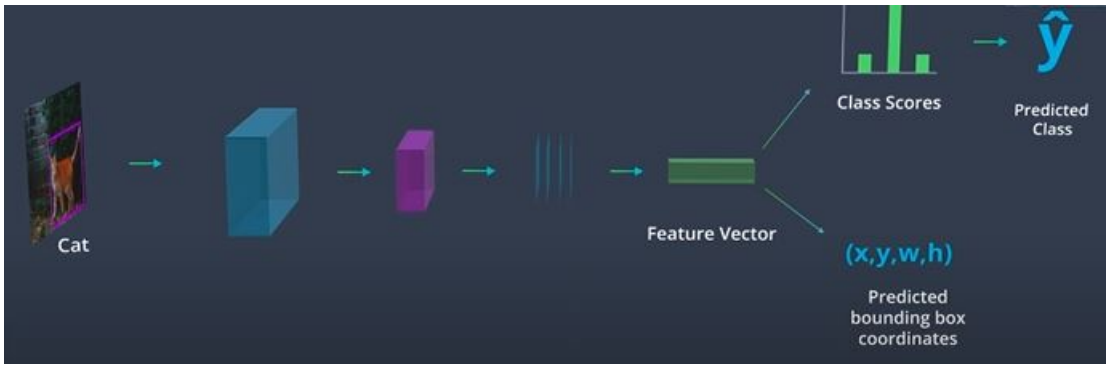


Figura 27: **YOLO - Arquitectura General** [21]

Como vemos en la imagen superior, la imagen de entrada pasa por una **red convolucional**, es decir, se hace pasar por una serie de **filtros**, que como explicamos en apartados anteriores, cada uno de ellos consigue aislar patrones y características en la imagen, formando lo que denominamos **mapa de características**.

Estos **mapas de características** son simplificados en un **feature vector (característica vectorial)**, entonces, ¿Que ocurre con este elemento? El **feature vector** se hace pasar por una capa neuronal que se encarga exclusivamente de asociar probabilidades de que exista cierto objeto (individual), obteniéndose una probabilidad para cada una de las clases especificadas en

el modelo. Finalmente, se escoge la clase con la máxima probabilidad para un objeto detectado en la imagen y se asigna dicha clase a la detección concreta con la probabilidad asociada.

A los **mapas de características** anteriormente mencionados, se le añade otra capa neuronal de salida. Esta capa se encarga a través de **regresiones lineales** de predecir las coordenadas del **bounding box** para la misma detección, consiguiendo las coordenadas necesarias para localizar al objeto detectado en cuestión **(x,y,w,h)**.

El procedimiento anterior tiene sentido, pero para detectar un solo objeto en una imagen. ¿Cómo podemos llegar detectar múltiples objetos en una imagen dada?.

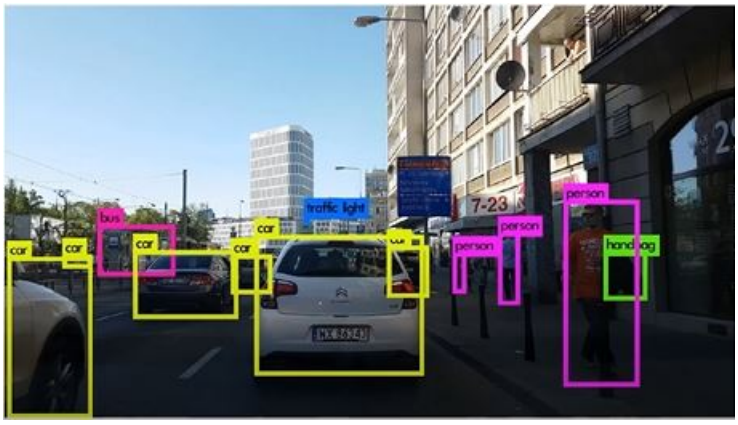


Figura 28: **YOLO - Detección Múltiple de Objetos** [21]

Para solucionar el problema planteado en la cuestión anterior, se recurre a lo que se denomina **Sliding Windows (Ventanas Deslizantes** o **Ventanas Deslizantes**. El procedimiento consiste en crear una ventana mucho más pequeña que la imagen original, la cual se va desplazando a lo largo de la imagen original hasta abarcar la imagen completamente. Cada una de esta ventanas es procesada por **YOLO** detectando el posible objeto en cada una de ellas y realizando el procedimiento descrito anteriormente, lo cual aumenta bastante el coste computacional.

Para reducir el coste computacional que incurría la técnica del **Sliding Window**, se implementa otra, denominada **Region proposals** o **Regiones propuestas**, donde sencillamente lo que se hace es hacer pasar la imagen original previamente por una capa de red convolucional que se encarga de obtener de la imagen aquellas secciones que tienen mas probabilidad de contener un objeto, basándose en cambios de contraste detectados o determinados patrones geométricos. A fin de cuentas, con esto conseguimos que el número de ventanas que va

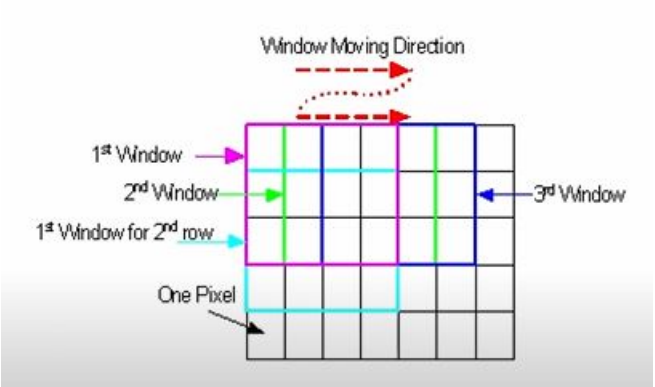


Figura 29: **YOLO - Ventana Deslizante** [21]

a procesar nuestro algoritmo **YOLO** sea mucho menor y el coste computacional se reduzca considerablemente.

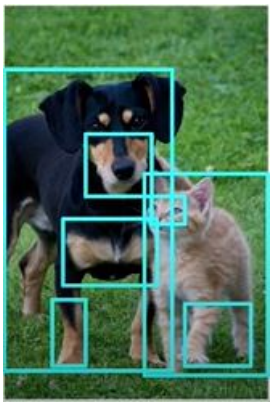


Figura 30: **YOLO - Regiones Propuestas** [21]

2.3.3. Faster R-CNN

Una red **Faster R-CNN (Region-based Convolutional Neural Network)** se trata de un modelo de **red neuronal** que se basa en el concepto de **redes neuronales convolucionales**. Cabe mencionar, que es un modelo evolucionado, de lo que en un origen se conocía como **R-CNN (Region-based Convolutional Neural Network)**. Este tipo de modelos, al igual que **YOLO**, se basan en la extracción de las regiones de la imagen propensas a que incluyan un objeto, pero debido al lento procesamiento que ello conllevaba el modelo inicial, comenzaron a aparecer otros modelos que incluían capas o operaciones adicionales para reducir el tiempo de procesamiento y permitir otras formas de **detección de objetos Fast R-CNN, Faster R-**

CNN, Mask R-CNN.

En el estudio nos vamos a centrar en el funcionamiento de **Faster R-CNN**, como ya explicamos en el modelo **YOLO**, para detectar un objeto necesitamos realizar una predicción en cuanto a qué **tipo o clase de objeto** y luego también necesitamos identificar donde aparece dicho objeto en la imagen. Para ello, se utiliza lo que se denomina como **bounding box**, que es un rectángulo que enmarca cada objeto detectado en la imagen, definiéndose con las coordenadas de la posición x e y de la esquina superior izquierda del rectángulo, además del ancho y el alto.

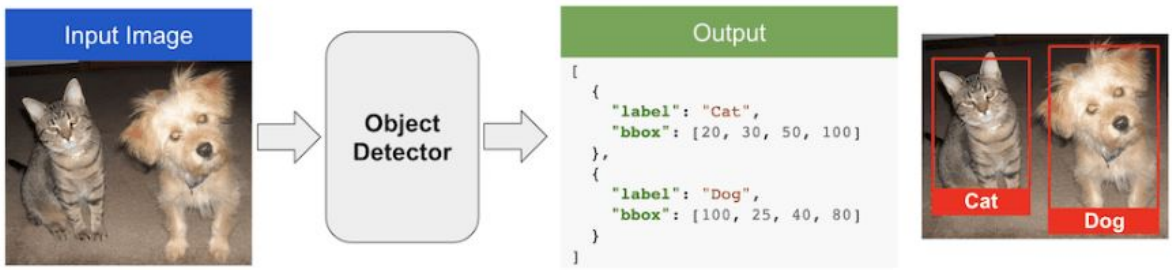


Figura 31: **Faster R-CNN - Detección de Objetos** [35]

Cuando hablamos de **detección de objetos** podemos decir que es un proceso que se describe en dos pasos:

- **1.- Detectar Bounding Boxes:** Lo primero en realizar es marcar aquellos cuadros delimitadores en los que se tenga la suficiente certeza de que contiene un solo objeto.
- **2. - Clasificar Objeto:** Por último, se clasifica el supuesto objeto dentro de cada **bounding box** asociándole una etiqueta.

Si nos adentramos en el funcionamiento de **Faster R-CNN**, primero debemos introducir en el modelo que se basa, **R-CNN**.

Como podemos observar en la imagen el funcionamiento general es bastante parecido a **YOLO**. En primer lugar, identificamos y extraemos las **regiones propuestas** donde es más probable que encontremos objetos. Luego, cada una de estas regiones son procesadas por las diferentes **capas convolucionales** de la red, permitiendo clasificar cada una de estas regiones, identificando la clase de objeto que le corresponde a cada región propuesta.

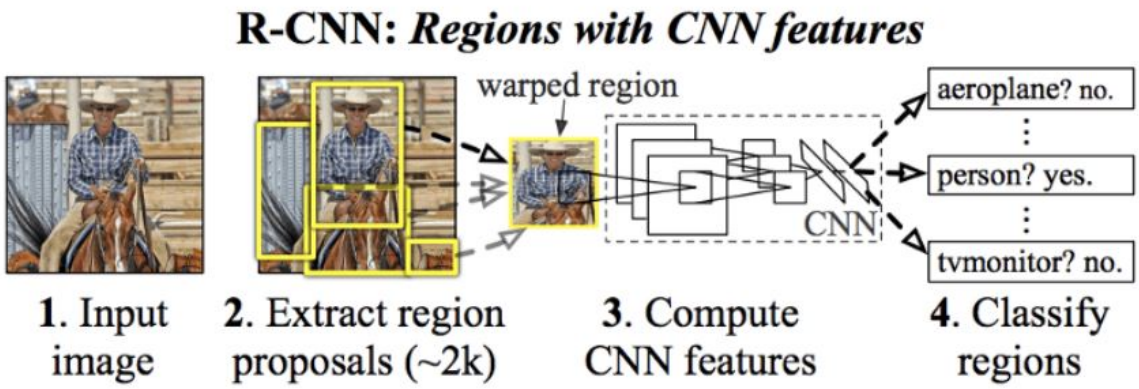


Figura 32: **R-CNN - Esquema Funcionamiento** [35]

Después de **R-CNN** y **Fast R-CNN**, nace **Faster R-CNN** corroborando ser un modelo de **aprendizaje profundo** mucho mas rápido que los anteriores, simplemente por el hecho de que con **Faster R-CNN** el cálculo de propuestas de regiones y clasificación de imágenes pueden utilizar el mismo **mapa de características** y así combatir la carga computacional que suponían modelos anteriores.

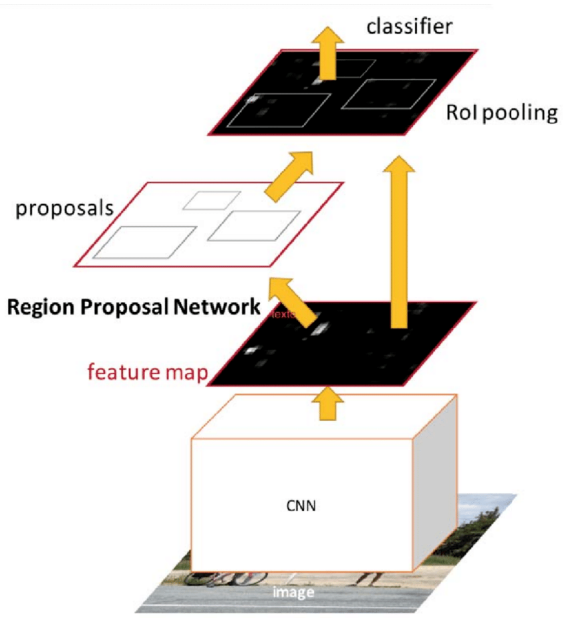


Figura 33: **Faster R-CNN - Esquema Funcionamiento** [35]

Vamos a comentar el proceso de una **Faster R-CNN** desde que entra la imagen hasta que se realiza la clasificación de cada uno de los objetos detectados. Podemos dividir el proceso en tres etapas:

- **1.- Extracción de características:** En la primera etapa se hace uso de una red **CNN** pre-entrenada (VGG o ResNet). Esta red se encarga de obtener todos los **mapas de características** posibles, que se pueden extraer de la imagen original de entrada.

- **2.- Propuesta de Regiones:** Durante la segunda etapa, tal como vemos en la figura anterior, se obtienen las denominadas **regiones propuestas**. Esto consiste en la aplicación de **algoritmos de selección de regiones (Selective Search)** sobre los **mapas de características**, con el objetivo de marcar posibles regiones que pueden llegar a contener objetos en la imagen original.

- **3.- Detección de Objetos:** En la última etapa se hace uso de 2 redes neuronales diferentes, una encargada de **clasificar** los objetos en función de su categoría dentro de cada **región propuesta**, y otra que se encarga de **localizar** a través de regresiones los objetos, ajustando la ubicación y el tamaño de la **región propuesta**, enmarcando el objeto en un **bounding box**.

2.3.4. Diferencia entre YOLO y Faster R-CNN

En principio, ambos modelos de red parecen bastante parecidos, pero realmente guardan una diferencia fundamental en el rendimiento de cada uno de ellos independientemente.

Para encontrar esta diferencia, tenemos que fijarnos en la arquitectura de un modelo **Faster R-CNN** y la arquitectura de un modelo **YOLO**.

La figura que estamos observando muestra la la arquitectura de ambos modelos, mostrando todos sus componentes.

El modelo **Faster R-CNN** utiliza una arquitectura de tres etapas, una primera etapa **Convolutional Network (Red Convolucional)** en la que se extraen los **mapas de características** de la imagen de entrada, una segunda etapa **Region Proporsal Network (Red de Regiones Propuestas)**, encargada de generar **regiones candidatas** que pueden contener objetos en los **mapas de características** anteriores. Además, interviene una tercera etapa **Fast R-CNN Network**, que recibe cada **región propuesta**, la clasifica en las diferentes **categorías de objetos** y predice los **bounding boxes** de los objetos detectados.

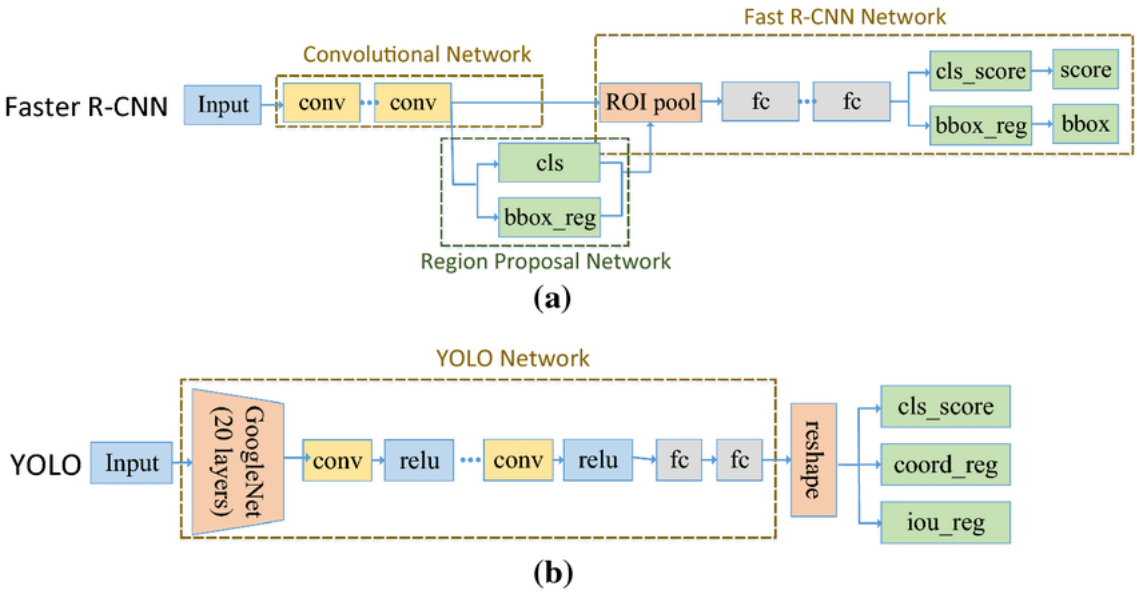


Figura 34: **Faster R-CNN vs YOLO**[1]

En cambio, el modelo **YOLO** tiene un enfoque mas integral para la **detección de objetos**, considerando la imagen completa de una sola vez, dividiendo la imagen en ventanas, seguidamente prediciendo los **bounding boxes** y las clases de los objetos en celda de la cuadrícula de una sola pasada.

Podemos concluir que la principal diferencia radica en que **Faster R-CNN** tiene 3 redes neuronales diferentes, cada una con un propósito, lo cual ralentiza el procesado incurriendo a un coste computacional significativo. Mientras que **YOLO**, se basa en una única red neuronal, lo que hace que el procesado sea mucho más rápido que el de **Faster R-CNN** pero a la vez menos precisa.

2.4. Ruido Gaussiano

El **Ruido Gaussiano** se trata de un ruido particular con cierta aleatoriedad debido a una gran variedad de factores como puede ser la interferencia electromagnética, los errores de medición, el ruido térmico en los sensores de la cámara ...

Su principal característica es que sigue una **distribución de probabilidad normal**, la intensidad del mismo cambia de forma aleatoria siguiendo una distribución de **Campana de Gauss**.

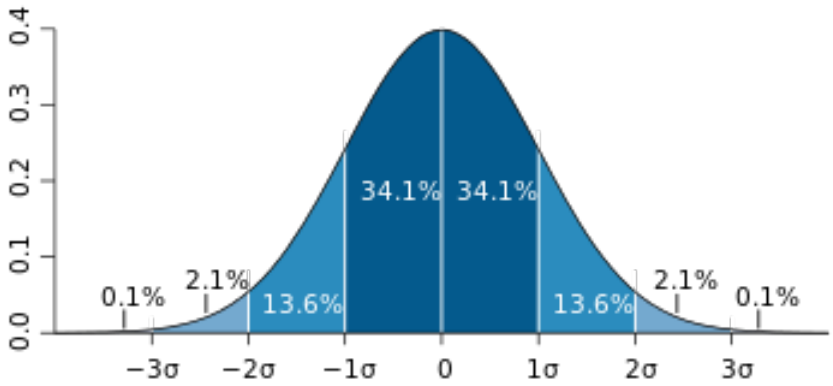


Figura 35: Ruido Gaussiano - Campana de Gauss [19]

Este ruido afecta, generando un impacto significativo, a imágenes que se procesan en diferentes sectores como la videovigilancia, medicina, industria y fotografía principalmente. El efecto que provoca en las imágenes es: distorsión de la imagen, pérdida de detalle y disminución de la calidad general de la misma.

En el caso del sector de la salud, el **ruido gaussiano** puede aparecer en imágenes de diagnóstico médico como resonancias magnéticas o tomografías computadas, lo cual dificulta el propio diagnóstico de la enfermedad o el trastorno que se esté estudiando. En la industria, este tipo de ruido en las imágenes procesadas, puede llegar a causar errores graves de medición en los sistemas de inspección de calidad, dando lugar a la producción de productos defectuosos.

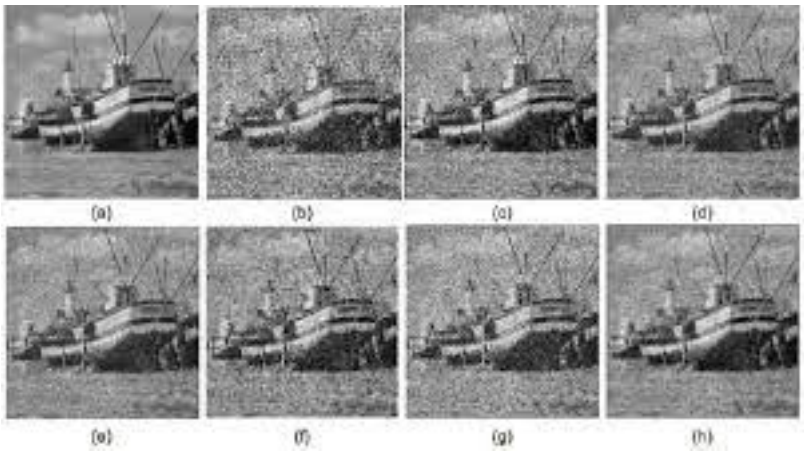


Figura 36: **Ruido Gaussiano - Comparativa** [2]

En las imágenes que podemos ver, vemos como el granulado y la pérdida de calidad de la imagen se produce a medida que aumentamos la desviación típica (sigma) de la **distribución normal** asociada a este **ruido gaussiano**.

¿Cómo afecta todo esto a la **Inteligencia Artificial**?. El impacto que genera los vamos a estudiar más adelante en los experimentos pertinentes. Lo que es seguro, es que genera un impacto suficiente como para afectar en la predicción de **modelos de redes neuronales convolucionales** durante la detección de objetos, aplicado a los sectores anteriormente nombrados.

3
Tecnología

En esta sección comentamos y describimos brevemente las funcionalidades de las bibliotecas, frameworks y APIs empleadas para llevar a cabo el estudio.

3.1. Pytorch

Pytorch se trata de un **framework (estructura)** de **aprendizaje profundo open-source** basado en Python, concretamente en la biblioteca **Torch**. Este **framework** entre sus funciones principales permite crear y entrenar **modelos de aprendizaje profundo**, gracias a que proporciona una forma de trabajar con **tensores** muy fácil de entender, permitiendo poder realizar operaciones matemáticas con datos multidimensionales (texto, imágenes y sonido) [32].

Cuando hablamos de un **tensor**, se hace referencia a una array multidimensional o matriz multidimensional. Se trata de la unidad fundamental de datos de **Pytorch** para la construcción y entrenamiento de **modelos de aprendizaje profundo.**

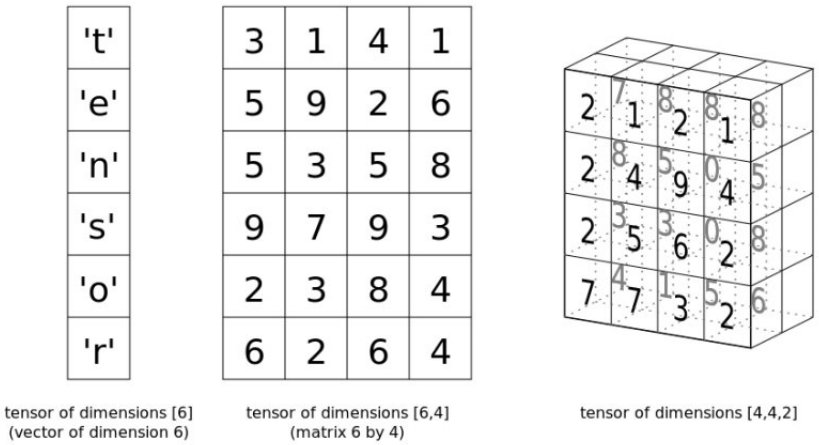


Figura 37: **Ejemplo de Tensores** [33]

Una característica más a mencionar sobre **Pytorch** es que nos permite realizar cálculo en GPU, con el objetivo de utilizar los tensores (datos) para que sean procesados en dicha GPU por un **modelo de aprendizaje profundo**, necesitándose una buena capacidad de cómputo.

Pytorch proporciona una gran variedad de bibliotecas y módulos que facilitan la construcción y entrenamiento de los diferentes **modelos de aprendizaje profundo** a los que podamos optar, en función del cometido que tengan cada uno de estos modelos. Entre variedad de herramientas que proporciona el **framework** (**torchvision, torchtext, torchaudio, torchsummary, ignite)** destacamos **torchvision** que es el módulo del que vamos a hacer especial uso durante el estudio [34].

Torchvision es una biblioteca cuyo objetivo es facilitar la manipulación de datos referidos a la visión artificial por computadora, enfocada en imágenes y vídeos. Proporciona utilidades muy útiles que enumeramos a continuación:

- **Cargar datos:** Proporciona herramientas para cargar una gran cantidad de conjuntos de datos muy comunes en la visión artificial, incluso conjunto de datos personalizados.

- **Transformar datos:** Permite que hagamos transformaciones de datos para preprocesar imágenes y vídeos, como transformaciones de cambio de tamaño, normalización, rotación, etc.

- **Utilizar Modelos pre-entrenados:** Incluye una buena gama de modelos cuyos pesos ya se encuentra pre-entrenados para tareas como la clasificación y segmentación de imágenes, detección de objetos, etc.

En concreto, durante el estudio, para utilizar el **modelo de aprendizaje profundo YOLO** y las diferentes versiones del mismo hacemos uso de la **versión 1.13.1 de Pytorch**, que se trata de una versión estable, además esta versión ha sido compilada con soporte para **CUDA**, con la **versión 11.7**. **CUDA** se trata de una plataforma de cómputo paralelo de **NVIDIA** utilizada para acelerar cálculos en GPU. Para la utilización de los **modelos de aprendizaje profundo Faster R-CNN** también se ha usado esta versión ya que resulta totalmente compatible.

3.2. Numpy

Numpy se trata de una biblioteca de Python cuya finalidad principal es trabajar de forma eficiente con matrices y arrays numéricos, utilizándose normalmente para el análisis de datos y la computación científica[3].

Como hemos mencionado anteriormente, uno de los puntos fuertes de manejar matrices y arrays numéricas con **Numpy** es que las estructuras de datos que proporciona están optimizadas para realizar operaciones matemáticas donde intervengan grandes cantidades de datos con mucha más rapidez que utilizando las estructuras básicas de Python.

Por ello en este estudio, se hace uso de las estructuras y utilidades que porporciona **Numpy** deliberadamente, consiguiendo así reducir el costo computacional que conlleva manejar los datos correspondientes al conjunto de datos de imágenes y facilitar las operaciones durante la **detección de objetos.** Algunas de las utilidades principales de las que se hace uso de forma interna en los **modelos de aprendizaje profundo** utilizados en el estudios son: **estructuras de datos para matrices n-dimensional (n-dimensiones)** y **funciones para realizar operaciones matemáticas en matrices (aritméticas básicas, trigonométricas, exponenciales)**[30].

3.3. YOLO Ultralytics

YOLO ultralytics es una implementación mejorada del **modelo de detección de objetos YOLO**, comentado en secciones anteriores.

Ultralytics [4] se trata de una empresa especializada en la elaboración y desarrollo de técnicas que mejoran el campo de las herramientas de **visión por computadora** en la detección de objetos en imágenes y en tiempo real.

A través de **YOLO ultralytics** ofrece **YOLO V8** y **YOLO V5**, ambos se tratan de **frameworks** para los que implementa la metodología de **YOLO** en un **modelo de aprendizaje profundo** incluyendo mejoras en la precisión y eficiencia, además de ofrecernos una API para facilitar la visualización de los resultados obtenidos en la **detección de objetos** en imágenes y vídeos.

En nuestro estudio haremos uso de **YOLO V8**, se trata de un modelo de última generación, basado en el éxito de las versiones anteriores de **YOLO**, en el que se introducen nuevas carac-

terísticas y mejoras para producir una mejora en el rendimiento. Permite tareas de **detección** y **seguimiento** de objetos, **segmentación de imágenes** y **clasificación** de imágenes, incluso algunas de estas en tiempo real.

Dentro de las tareas mencionadas que nos permite realizar, nos centraremos durante el estudio en la predicción para la **detección de objetos** en imágenes. La descripción de las utilidades que proporciona y como podemos usarlas para las detecciones se pueden dividir en:

- **Fuentes**: Acepta varias fuentes de entrada, entre ellas **imágenes, URL, imágenes PIL, OpenCV, matrices numpy, tensores, archivos CSV, videos, directorios, videos de Youtube y streamings**.

source	model(arg)	type	notes
image	`'im.jpg'`	`str`, `Path`	
URL	`'https://ultralytics.com/images/bus.jpg'`	`str`	
screenshot	`'screen'`	`str`	
PIL	`Image.open('im.jpg')`	`PIL.Image`	HWC, RGB
OpenCV	`cv2.imread('im.jpg')[:,:,::-1]`	`np.ndarray`	HWC, BGR to RGB
numpy	`np.zeros((640,1280,3))`	`np.ndarray`	HWC
torch	`torch.zeros(16,3,320,640)`	`torch.Tensor`	BCHW, RGB
CSV	`'sources.csv'`	`str`, `Path`	RTSP, RTMP, HTTP
video ✅	`'vid.mp4'`	`str`, `Path`	
directory ✅	`'path/'`	`str`, `Path`	
glob ✅	`'path/*.jpg'`	`str`	Use `*` operator
YouTube ✅	`'https://youtu.be/Zgi9g1ksQHc'`	`str`	
stream ✅	`'rtsp://example.com/media.mp4'`	`str`	RTSP, RTMP, HTTP

Figura 38: **Fuentes de YOLO Ultralytics [31]**

- **Argumentos:** Los argumentos aceptados por la función encargada de predecir (**model.predict**), podemos observarlos en la siguiente figura.

Key	Value	Description
source	'ultralytics/assets'	source directory for images or videos
conf	0.25	object confidence threshold for detection
iou	0.7	intersection over union (IoU) threshold for NMS
half	False	use half precision (FP16)
device	None	device to run on, i.e. cuda device=0/1/2/3 or device=cpu
show	False	show results if possible
save	False	save images with results
save_txt	False	save results as .txt file
save_conf	False	save results with confidence scores
save_crop	False	save cropped images with results

Figura 39: **Argumentos YOLOV8 (1)** [31]

hide_labels	False	hide labels
hide_conf	False	hide confidence scores
max_det	300	maximum number of detections per image
vid_stride	False	video frame-rate stride
line_thickness	3	bounding box thickness (pixels)
visualize	False	visualize model features
augment	False	apply image augmentation to prediction sources
agnostic_nms	False	class-agnostic NMS
retina_masks	False	use high-resolution segmentation masks
classes	None	filter results by class, i.e. class=0, or class=[0,2,3]
boxes	True	Show boxes in segmentation predictions

Figura 40: **Argumentos YOLOV8 (2)** [31]

- **Formatos de Vídeo e Imagen**: Descripción de los posibles formatos de imagen y vídeo que podemos utilizar con esta versión.

Image Suffixes	Example Predict Command	Reference
.bmp	yolo predict source=image.bmp	Microsoft BMP File Format
.dng	yolo predict source=image.dng	Adobe DNG
.jpeg	yolo predict source=image.jpeg	JPEG
.jpg	yolo predict source=image.jpg	JPEG
.mpo	yolo predict source=image.mpo	Multi Picture Object
.png	yolo predict source=image.png	Portable Network Graphics
.tif	yolo predict source=image.tif	Tag Image File Format
.tiff	yolo predict source=image.tiff	Tag Image File Format
.webp	yolo predict source=image.webp	WebP
.pfm	yolo predict source=image.pfm	Portable FloatMap

Figura 41: **Formatos de Imagen YOLOV8** [31]

Video Suffixes	Example Predict Command	Reference
.asf	yolo predict source=video.asf	Advanced Systems Format
.avi	yolo predict source=video.avi	Audio Video Interleave
.gif	yolo predict source=video.gif	Graphics Interchange Format
.m4v	yolo predict source=video.m4v	MPEG-4 Part 14
.mkv	yolo predict source=video.mkv	Matroska
.mov	yolo predict source=video.mov	QuickTime File Format
.mp4	yolo predict source=video.mp4	MPEG-4 Part 14 - Wikipedia
.mpeg	yolo predict source=video.mpeg	MPEG-1 Part 2
.mpg	yolo predict source=video.mpg	MPEG-1 Part 2
.ts	yolo predict source=video.ts	MPEG Transport Stream
.wmv	yolo predict source=video.wmv	Windows Media Video
.webm	yolo predict source=video.webm	WebM Project

Figura 42: **Formatos de Vídeo YOLOV8** [31]

3.4. Faster R-CNN Pytorch

Como ya mencionamos anteriormente, **Pytorch** se trata de un **framework open-source** que proporciona herramientas y utilidades en el ámbito del **aprendizaje profundo**.

Faster R-CNN de **Pytorch** [5] proporciona gama de implementaciones eficientes y sencillas de usar del algoritmo **Faster R-CNN**, enfocadas en tareas de **detección de objetos**, **clasificación de imágenes**, **segmentación semántica y de instancia** principalmente.

Entre las posibilidades que nos ofrece podemos destacar las siguientes:

- **Transformación y aumento de imágenes: Pytorch** permite realizar transformaciones para modificar las imágenes de entrada de diversas formas, como normalizar, recortar, cambiar el brillo, rotar, redimensionar, entre otras. Estas herramientas son útiles

para preparar y mejorar los conjunto de datos que empleemos en el entrenamiento y evaluación de modelos **Faster R-CNN**

- **Datapoints:** Proporciona estructuras de datos eficientes y flexibles para almacenar y acceder a información de datos individuales. Los **datapoints** contienen detalles sobre imágenes, anotaciones de objetos y otro metadatos importantes. Con la utilización de esta utilidad conseguimos una mejor organización y manipulación de datos.

- **Modelos y pesos pre-entrenados: Pytorch** pone a disposición una enorme variedad de **modelos pre-entrenados** de **Faster R-CNN**, donde se incluyen los pesos ya entrenados en conjuntos de datos relevantes, como en nuestro caso es **COCO**. Los pesos pre-entrenados los podemos utilizar directamente o como base para entrenar el modelo y ajustarlo a otros conjuntos de datos.

- **Conjuntos de Datos:** También incluye funciones y herramientas para cargar y gestionar conjuntos de datos de detección de objetos.

- **Lectura/escritura de imágenes y vídeos:** Incluye funciones que nos permiten leer y escribir imágenes y vídeos desde y hacia diferentes formatos.

En nuestro caso, nos vamos a centrar en los **Modelos y pesos pre-entrenados**, ya que haremos uso de dos modelos **Faster R-CNN** con pesos pre-entrenados.

- **Faster R-CNN RESNET50 FPN V2:** Se trata de una mejora del modelo clásico **Faster R-CNN**, con una **ResNet-50-FPN backbone**.

- **Faster R-CNN MOBILENET V3 LARGE FPN:** Se trata de un modelo de Faster R-CNN de alta resolución con una **MobileNetV3-Large-FPN backbone**.

ResNet50 FPN V2 se trata de una arquitectura mas compleja de **backbone**, ya que tiene mas capacidad de representación precisión pero necesita de mas recursos computacionales, en cambio, **MobileNetV3 Large** es una arquitectura mas ligera y eficiente en recursos, lo que le otorga mas rapidez al procesamiento del modelo a cambio de una ligera reducción en la precisión.

3.5. COCO API

COCO API es una biblioteca de Python llevada a cabo por el proyecto **COCO (Common Objects in Context - Objetos Comunes en Contexto)** [6], donde podemos encontrar una serie de funcionalidades que nos permite trabajar de una forma mucho más sencilla y cómoda con el conjunto de datos **COCO**.

- **Carga y manipulación de datos:** Nos permite cargar y acceder a las imágenes y sus correspondientes anotaciones. Podemos hacer uso de una interfaz a través de la cual accedemos a los archivos de anotaciones para realizar consultas sobre las detecciones de objetos en determinadas imágenes.

- **Visualización de resultados: COCO API** ofrece herramientas para visualizar las anotaciones de objetos en las imágenes, permitiendo dibujar los cuadros delimitadores para las detecciones, las etiquetas del objeto detectado, y así, poder verificar los resultados de detección **(Fifity One)**.

- **Evaluación de modelos de detección:** Ofrece una serie de métricas y funciones que nos permite evaluar los resultados obtenidos por nuestro modelo en cuestión obteniendo el rendimiento del mismo para el conjunto de datos **COCO**. Entre las métricas que proporciona, destacamos el **promedio de precisión de objetos (mAP)** y **la precisión media (AP)** a diferentes umbrales de **IoU (Intersección Sobre Unión)**.

- **Generación de resultados sobre las métricas: COCO API** nos permite generar resultados acerca de las métricas obtenidas de los resultados de las **detecciones o segmentaciones de objetos**. Esto nos permite comparar diferentes modelos de redes bajo ciertas condiciones controladas y así ponerlos a prueba.

- **Soporte para diferentes tipos de anotaciones:** Además de admitir anotaciones de detección de objetos, también permite la obtención de métricas a partir de las anotaciones sobre los resultados de la **segmentación** y **keypoints (puntos clave)**.

Todas estas funcionalidades nos permite trabajar y evaluar diferentes tipos de tareas de **visión por computador** utilizando el conjunto de datos **COCO** [6], del cual hablaremos en capítulos más adelante.

4
Metodología

4.1. Metodología del Estudio y Desarrollo

En este apartado se exponen ambas metodologías llevadas a cabo a la hora de llevar a cabo el estudio. Nos encontramos, dos metodologías diferentes dedicadas a partes distinta del estudio en sí. El método científico es una metodología que se utiliza en el ámbito de investigación científica para realizar estudios sistemáticos y obtener conocimiento confiable y que se pueda verificar para procesos específicos. Mientas que el método incremental se basa en la realización gradual y progresiva de funcionalidades necesarias para el estudio, el código del estudio se ha realizado mediante versiones funcionales a lo largo del proceso de desarrollo.

4.1.1. Método Científico

El método científico en su descripción común de las etapas de las que se compone de la **Observación**, **Planteamiento de Preguntas**, **Análisis y Recopilación de datos**, **Formulación de una Hipótesis**, **Evaluación e Interpretación de resultados**

- **Observación:** Entre las actividades realizadas se han escogido librerías y bibliotecas de modelos de detección de objetos, con el objetivo de entender su correcto funcionamiento, concretamente **YOLO** de **Ultralytics** y **Faster R-CNN** de **Pytorch**.

- **Planteamiento de Preguntas:** Una vez comprendido el funcionamiento básico a través de la ejecución de los diferentes modelos, se han planteado preguntas como: ¿El ruido gaussiano y el brillo afectaran significativamente de forma negativa para la detección de objetos en imágenes?, ¿Un algoritmo **YOLO** como se comporta ante una determinada escala de inserción de ruido gaussiano o reducción de brillo?, ¿En que aplicaciones destacarán **YOLO** y **Faster R-CNN** en la detección de objetos? ...

- **Análisis y Recopilación de datos:** Una vez planteadas las preguntas, se utilizaron foros, informes, artículos científicos y otras fuentes para determinar cómo realizar la experimentación en la detección de objetos sobre un conjunto de imágenes en diferentes condiciones de brillo y ruido.

- **Formulación de una Hipótesis:** Una vez analizados los datos a manejar por ambos modelos de detección de objetos, se plantea que **YOLO** parece ser una tecnología más rápida, mientras que **Faster R-CNN** elabora tareas de detección más precisas.

- **Evaluación e Interpretación de los Resultados:** Una vez realizados los experimentos, se confirman las hipótesis y además se obtienen otras en cuanto a el rendimiento en la detección de objetos de diferentes áreas de extensión.

4.1.2. Método Incremental

La metodología incremental se basa en la división de un proyecto en etapas, en lugar de llevarlo a cabo como si se trata de una sola tarea. Cada etapa que se realiza completa una o varias funcionalidades de lo que se pretende desarrollar.

Este se lleva a cabo en iteraciones, cada iteración recoge el análisis, diseño, implementación pruebas de una iteración, estas actividades se repiten en cada iteración hasta completar el objetivo en sus totalidad.

- **Iteración 1.** Durante la primera iteración se ha elaborado las funcionalidades de detección de objetos en una serie de imágenes de forma automática con **YOLO**. El módulo elaborado, se dedica a obtener los resultados de las detecciones realizadas en un archivo de texto, junto con los objetos delimitados en la imagen de forma automatizada para un conjunto de imágenes.

- **Iteración 2.** Para poder generar los experimentos, sobre las imágenes pertenecientes a **COCO**, se ideó un código capaz de insertar ruido y reducir el brillo en las imágenes seleccionadas para el experimento. Simplemente la funcionalidad consistía en insertar un determinado nivel de ruido gaussiano en una imagen determinada junto con otra funcionalidad para reducir el nivel de brillo en cierto factor.

- **Iteración 3.** Durante la tercera iteración se implementa la funcionalidad que nos permite obtener la evaluación de los resultados de las predicciones con **YOLO**, utilizando como métrica **Average Precision (AP)** y **mean Average Precision (mAP)**. Este módulo de código nos permite para un experimento concreto con un determinado nivel de ruido y brillo, obtener una evaluación de los resultados en términos de **Precisión promedio** para la versión del modelo de la red con la que se está experimentando.

  Iteración 4. Siendo una de las últimas iteraciones, en la cuarta iteración se elabora otro código capaz de obtener los **heatmaps (mapas de calor)** para poder evaluar los resultados basados en el **Average Precision (AP)** para cada versión estudiada de **YOLO** y **Faster R-CNN**. Estos **heatmaps** se han utilizado para confirmar las hipótesis planteadas durante el estudio y descubrir alguna conclusión acerca del funcionamiento de los modelos estudiados.

  Iteración 5. Por último, se ha producido la incorporación de un módulo de código dedicado a la detección automática de las imágenes escogidas para los experimentos con **YOLO**. Este módulo de igual forma que el elaborado en la **iteración 1 y 2** es capaz de escribir los resultados, para luego posteriormente hacer uso del resto de funcionalidades elaboradas en las iteraciones anteriores y conseguir de igual forma los **heatmaps** donde podemos observar el rendimiento de cada versión de **Faster R-CNN** dentro de cada modelo elegido.

5 Resultados

5.1. Métodos

Para realizar los diferentes experimentos y así extraer conclusiones se ha hecho uso de una serie de **modelos de redes neuronales** destinados a la **detección de objetos**, cuyos pesos ya se encontraban pre-entrenados con anterioridad. Entre ellos, encontramos a **YOLO V5u** de la biblioteca de **Ultralytics**, que se trata de una versión mejorada con respecto a la original, haciendo uso de las versiones **YOLO V5nu**, **YOLO V5mu** y **YOLO V5xu**, desde la versión mas ligera hasta la versión mas pesada. Además también se ha experimentado con **YOLO V8** y las siguientes versiones **YOLO V8n**, **YOLO V8m** y **YOLOV8x** [4].

Otro de los modelos que se ha evaluado se trata de **Faster R-CNN** de la biblioteca **Pytorch** a través del paquete **Torchvision**. En concreto, se ha hecho uso de **Faster R-CNN ResNet50 FPN V2** [7], una versión mejorada en cuanto a precisión sobre el modelo original, ambos pre-entrenados con un conjunto de imágenes denominado **ResNet**. Otro de los modelos estudiados, enfocado en una **detección de objetos** más ligera, es **Faster R-CNN Mobilenet V3 Large FPN** [8], pre-entrenado con el conjunto de imágenes **Mobilenet**.

Todos los experimentos han sido realizados en una computadora personal con arquitectura de 64 bits con un procesador Intel Core i7-8750H a una frecuencia de 2.20 GHz, 16 GB de RAM y una GPU NVIDIA Geforce GTX 1060.

5.2. Conjunto de Datos

El conjunto de datos referente a imágenes en el que se basa todo nuestro experimento se trata de **COCO** [6], un **dataset (conjunto de datos)** muy ampliamente utilizado en el campo de la visión por computador.

COCO abarca imágenes que han sido seleccionadas en sus condiciones naturales con un

total de aproximadamente 300.000 imágenes de las cuales más de 200.000 imágenes se encuentran etiquetadas, encontrando 80 categorías de objetos diferentes. Podemos encontrar 2 grandes conjuntos de datos seleccionados para tareas de **detección**, **segmentación** y **keypoints**, uno de ellos es el conjunto de imágenes seleccionado en 2014, segregado en un conjunto **entrenamiento**, un conjunto de **validación** y un conjunto de **testeo**, con sus correspondientes anotaciones cada uno de estos subconjuntos. Hay que tener en cuenta que para el conjunto de testeo no se han publicado las anotaciones pertinentes.

El conjunto que vamos a utilizar se trata del **conjunto de imágenes de 2017**, concretamente el **conjunto de validación** unas 5.000 imágenes que ocupan 1GB aproximadamente, con sus correspondientes anotaciones para evaluar nuestros experimentos.

5.3. Métricas

Cuando hablamos de las métricas, hacemos referencia a las herramientas que nos permiten evaluar como de buenos son nuestros modelos en la **detección de objetos**.

Los resultados obtenidos en los diferentes experimentos se basan en el **AP (Average Precision)**, se trata de una métrica que se usa bastante para evaluar **modelos de detección de objetos** (**Faster R-CNN, YOLO, SSD, etc)**. En nuestro caso nos es bastante útil ya que una de las métricas que nos permite obtener **COCO** a la hora de evaluar resultados de detecciones es el **AP** y el **mAP**.

El **AP** mide la precisión del modelo calculando el valor de precisión promedio para el valor de recuperación de 0 a 1, basándose en un concepto denominado **IoU (Intersección sobre Unión)**

Para comprender mejor como se hace al cálculo anterior del **AP**, necesitamos entender que es el **IoU**. El **IoU** se trata de una medida que evalúa la superposición entre el área de predicción y el área de anotación verdadera, es decir, cuánto se superpone el límite que ha predicho nuestro modelo con el límite del objeto real en la imagen.

Otro de los conceptos que debemos comentar hace referencia a la **Precisión** en sí, la **Precisión** simplemente mide qué tan precisas son las predicciones que hace nuestro modelo en base al **IoU** obtenido de nuestra detección, es decir, el porcentaje de predicciones sobre detecciones de objetos que son correctas, dicho de otra forma, porcentaje de detecciones cuyos **IoU**'s sean del 50 % (0.5) o superior. Este umbral de **IoU** podemos ir variándolo según como

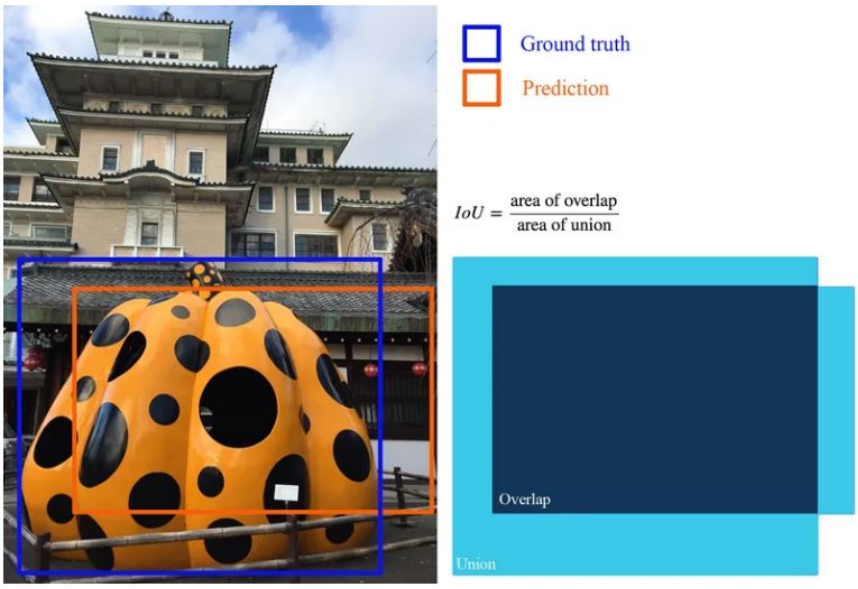


Figura 43: **IoU - Detección de Objetos** [25]

de estricto queremos que sea la evaluación sobre las detecciones de nuestro modelo, pues todas las detecciones que posean un **IoU** inferior al del umbral establecido se descartarán como predicciones verdaderas.

Nuestro código de evaluación, disgrega entre casos **verdaderos positivos (TP)**, **verdaderos negativos (TN)**, **falsos positivos (FP)** y **falsos negativos (FN)**.

$$Precision = \frac{TP}{TP + FP}$$

Figura 44: **Fórmula de Precisión 1** [25]

$$Precision = \frac{TP}{\text{total positive results}}$$

Figura 45: **Fórmula de Precisión 2** [25]

Teniendo en cuenta lo anterior, la forma en la que se considera una detección como un **verdadero positivo** o **detección correcta** es cuando el **IoU** sobre esa detección supera o iguala el umbral establecido del **IoU**. En caso de que sea inferior, se establece como un caso **falso positivo** o **detección incorrecta**.

Luego, para cada categoría de objeto, se calcula la precisión a diferentes umbrales de **IoU** como hemos mencionado y se promedia en base a la cantidad de umbrales, obteniendo lo que denominamos **AP**.

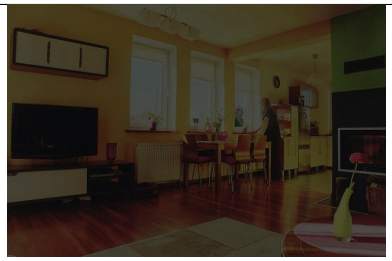
En general, cuánto mayor sea el **AP** para una categoría de objeto mayor rendimiento obtendrá el modelo en la detección de ese tipo de objetos.

Por otro lado, **mAP (mean Average Precision)** se trata del promedio de los **AP** de todos los tipos de objetos del conjunto de datos, proporcionándonos una medida de rendimiento general sobre el modelo para todas las clases de objetos.

5.4. Resultados Cualitativos

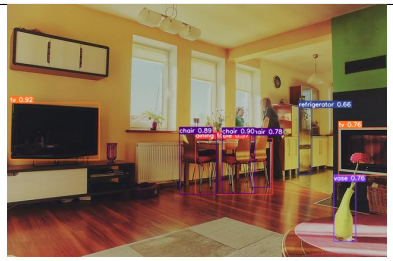
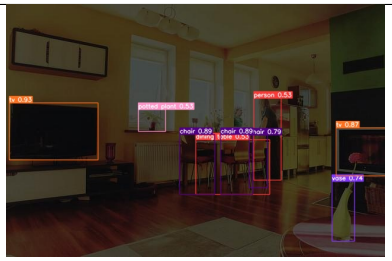
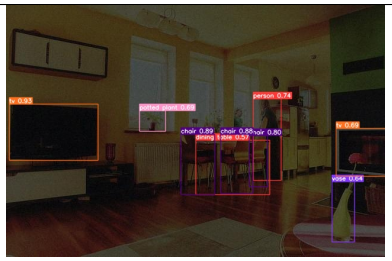
En esta sección, se muestra la comparación visual entre la alteración de la imagen al haber insertado **ruido gaussiano** y la modificación del brillo con respecto a la imagen en crudo, proporcionada por **COCO**. Esto se realiza para comprender como afecta al rendimiento la inserción del **ruido gaussiano** y la modificación del brillo de la imagen original. Las imágenes expuestas en la siguiente tabla corresponde con una habitación donde encontramos multitud de objetos a simple vista. Para los experimentos aislados, vamos a hacer uso del **modelo de detección de objetos YOLO V8** para detectar y etiquetar los objetos que aparecen. Entre los objetos que podemos encontrar en el primer experimento a mostrar, entre las 80 categorías de objetos proporcionada por **COCO**, aparecen **5 sillas**, **2 personas**, **2 televisiones**, **1 reloj de pared**, **4 floreros**, **1 microondas**, **1 frigorífico**, **2 libros**, **1 maceta** y **1 mesa**. En el segundo experimento, encontramos menos objetos a identificar, tan solo **1 sombrilla**, **1 camioneta**. **1 persona** y **1 coche**, justamente se ha elegido este experimento con muchos menos objetos a detectar para observar el comportamiento del modelo en cuestión cuando la cantidad de objetos a localizar es menor.

La imagen original, en ambos experimentos se encuentra en la celda de la esquina superior izquierda y el resto de imágenes no son mas que la misma pero con modificaciones en el **brillo** y la inserción de **ruido gaussiano**. Como podemos observar, para estas seis imágenes a medida que aumentamos el **ruido gaussiano** insertado, se produce un efecto de granulado blanco en la imagen haciendo cada vez menos claro los objetos y perdiéndose información, que como ya veremos más adelante afectará a las detecciones realizadas sobre la misma.

Noise	Original brightness ($b = 1{,}0$)	Modified brightness ($b = 0{,}4$)
Raw ($\sigma_g = 0{,}0$)		
Gaussian ($\sigma_g = 13{,}5$)		
Gaussian ($\sigma_g = 22{,}5$)		

Cuadro 1: Imagen 139 sin la aplicación de ruido gaussiano y modificación del brillo, y con inserciones de diferentes grados de ruido gaussiano (0.0, 13.5, 22.5). Además se incluye la imagen para los valores de ruido anteriores con el nivel de brillo modificado (1.0, 0.4).

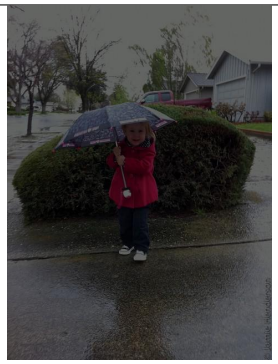
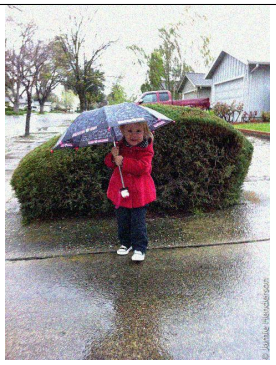
En la siguiente tabla podemos observar que después de realizar nuestro modelo **YOLO** las detecciones de los objetos sobre las imágenes tratadas y el número de predicciones se ve afectado por las circunstancias de ruido y brillo. Cada **categoría o clase** de objeto es delimitado con un color particular para esa **categoría**. Si observamos las imágenes, la imagen en crudo, es decir, la que se encuentra en la esquina superior izquierda, es la que mejores puntuaciones ha asignado a los objetos detectados y mas detecciones ha conseguido generalmente. Luego, si nos fijamos en la imagen de la esquina inferior derecha, la que se encuentra en peores condiciones, según las expuestas, tiene un número de detecciones inferior y la puntuación asignada a cada objeto es inferior.

Noise	Original brightness ($b = 1{,}0$)	Modified brightness ($b = 0{,}4$)
Raw ($\sigma_g = 0{,}0$)		
Gaussian ($\sigma_g = 13{,}5$)		
Gaussian ($\sigma_g = 22{,}5$)		

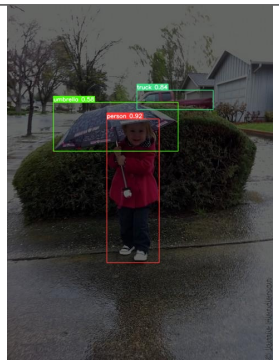
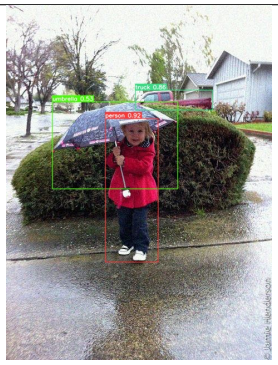
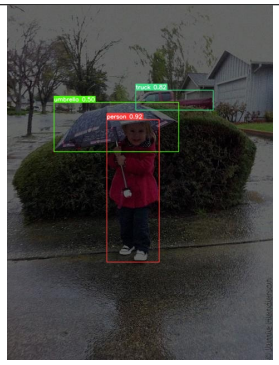
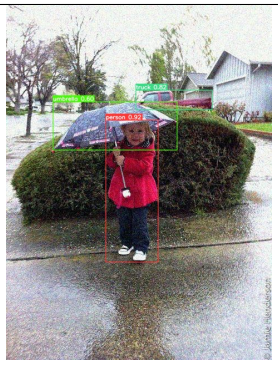
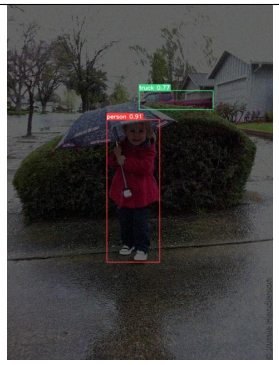
Cuadro 2: Imagen 139 con la detección de los diferentes objetos, teniendo en cuenta las mismas condiciones de brillo y ruido aplicado.

En cuanto al segundo experimento, mostramos las imágenes con las diferentes afectaciones de ruido y brillo, tal y como en el primero. La principal diferencia que encontramos, es que se trata de una imagen donde los posibles objetos a detectar, dentro de las 80 categorías establecidas en **COCO**, son muchos menos que en el primer experimento.

Podemos ver que en la tabla de detecciones (Cuadro 4) más abajo, cuando combinamos valores altos de **ruido gaussiano** con valores reducidos de brillo, algunos objetos, como la sombrilla, no los llega a detectar. Incluso, vemos como el **ruido gaussiano** en aquellas imágenes de la tabla más afectadas, interviene en la localización del objeto a detectar, asignándole más área de la que ocupa realmente en la imagen. Generalmente los resultados obtenidos, en cuanto a probabilidad de precisión por objeto detectado, son mucho mejores que en el primer experimento donde teníamos muchos mas objetos a detectar.

Noise	Original brightness ($b = 1{,}0$)	Modified brightness ($b = 0{,}4$)
Raw ($\sigma_g = 0{,}0$)		
Gaussian ($\sigma_g = 13{,}5$)		
Gaussian ($\sigma_g = 22{,}5$)		

Cuadro 3: Imagen 7088 con la detección de los diferentes objetos, teniendo en cuenta las mismas condiciones de brillo y ruido aplicado.

Noise	Original brightness ($b = 1{,}0$)	Modified brightness ($b = 0{,}4$)
Raw ($\sigma_g = 0{,}0$)		
Gaussian ($\sigma_g = 13{,}5$)		
Gaussian ($\sigma_g = 22{,}5$)		

Cuadro 4: Imagen 7088 con la detección de los diferentes objetos, teniendo en cuenta las mismas condiciones de brillo y ruido aplicado.

En la siguiente sección veremos como afecta el **ruido gaussiano** y la modificación del brillo a la imagen anterior en términos de probabilidad de predicción. En principio por lo que hemos podido observar visualmente en la tabla anterior parece que al combinar un elevado **ruido gaussiano** con una condición de **brillo** modificada podría afectar mucho más a la predicción que si la imagen solo estuviera afectada por **ruido gaussiano** o tuviese un **brillo** inferior al original. Otro factor que interviene significativamente en el rendimiento aparente del modelo es la cantidad de objetos candidatos a detectar que se encuentren en la imagen.

5.5. Resultados Cuantitativos

Ante las diferencias entre **ruido gaussiano** y **brillo**, **AP** es la métrica que nos va a proporcionar cómo de efectiva ha sido la predicción en comparación con las diferentes modificaciones insertadas en la imagen 139 y la imagen 7088 de las tablas del apartado anterior.

Como hemos mencionado en apartados anteriores, **AP** es una métrica que se utiliza bastante en el campo de la **detección de objetos** para medir la precisión de un modelo al detectar y localizar objetos en una imagen. En el cálculo de la misma, se considera tanto la **precisión de la detección** (cuántos objetos detectados son relevantes) como la **precisión de la localización** (cuánta precisión hay en la posición y el tamaño de los objetos detectados).

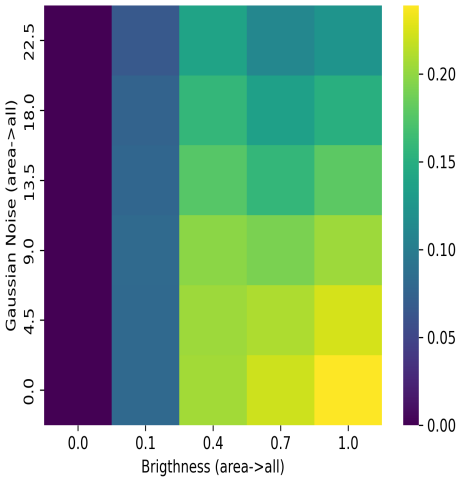
Image	Noise	B	AP
139	$\sigma_g = 0{,}0$	1.0	**0.333**
	$\sigma_g = 0{,}0$	0.4	**0.297**
	$\sigma_g = 13{,}5$	1.0	**0.333**
	$\sigma_g = 13{,}5$	0.4	**0.294**
	$\sigma_g = 22{,}5$	1.0	**0.149**
	$\sigma_g = 22{,}5$	0.4	**0.154**
7088	$\sigma_g = 0{,}0$	1.0	**0.675**
	$\sigma_g = 0{,}0$	0.4	**0.675**
	$\sigma_g = 13{,}5$	1.0	**0.5**
	$\sigma_g = 13{,}5$	0.4	**0.7**
	$\sigma_g = 22{,}5$	1.0	**0.45**
	$\sigma_g = 22{,}5$	0.4	**0.675**

Cuadro 5: Predicciones sobre los objetos de la imagen 139 y 7088 (COCO) realizados por **YOLO V8**. Cada columna que podemos ver, expone el valor del ruido gaussiano, el valor del brillo alterado y la métrica **AP** para cada combinación posible en cada imagen.

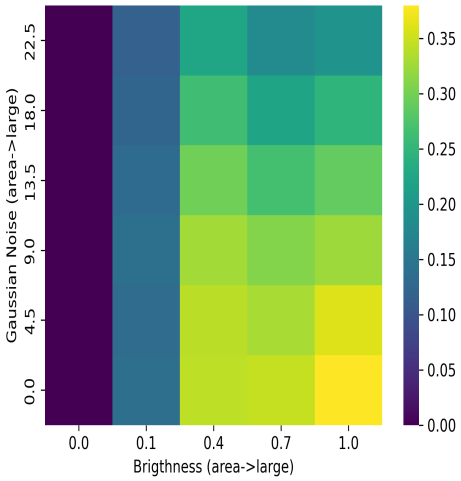
Si observamos la tabla anterior (Cuadro 5), podemos observar como el **AP (Average Precision)** para los objetos detectados se reduce a medida que aumentamos el **ruido gaussiano** en la imagen y reducimos las condiciones de **brillo** de la imagen. Particularmente, se confirma

lo mencionado en el apartado anterior, para la imagen 7088, al existir menos objetos a detectar el **AP (Average Precision)** es mucho mayor.

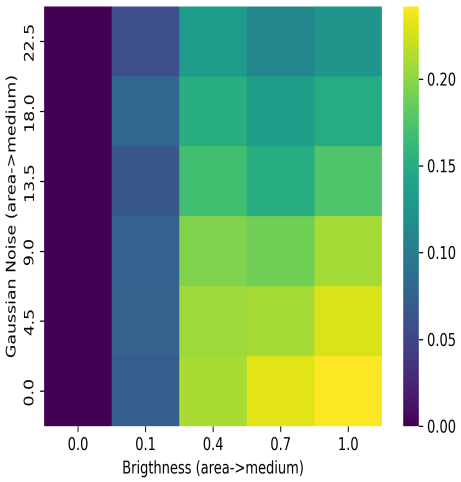
A continuación vamos a mostrar los resultados de los experimentos llevados a cabo para el modelo **YOLO** y **Faster R-CNN**, incluyendo las correspondientes versiones mencionadas anteriormente. Los valores de **ruido gaussiano** utilizados para los experimentos son: [0.0, 4.5, 9.0, 13.5, 18.0, 22.5] y los valores de **brillo**: [0.0, 0.1, 0.4, 0.7, 1.0]. Los resultados se expondrán en **heatmaps**, para cada versión de cada uno de los modelos se dedica un **heatmap** donde se muestran los diferentes valores **AP** obtenidos para cada una de las combinaciones entre los valores de ruido y brillo mencionados. Cabe destacar que por cada **heatmap** se ha elaborado una escala de colores, se muestra a la derecha de cada **heatmap**, asociándose dicha escala por intervalos a los valores de **AP** obtenidos para el **heatmap** en cuestión. Para el experimento se han utilizado un total de 1000 imágenes del conjunto de datos **COCO**, concretamente los 1000 primeras imágenes enumeradas por id en el conjunto de datos.



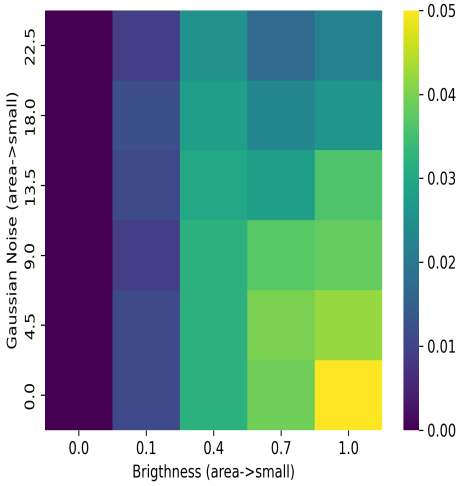
(a) Área de Objeto(Todas)



(b) Área de Objeto(Grande)

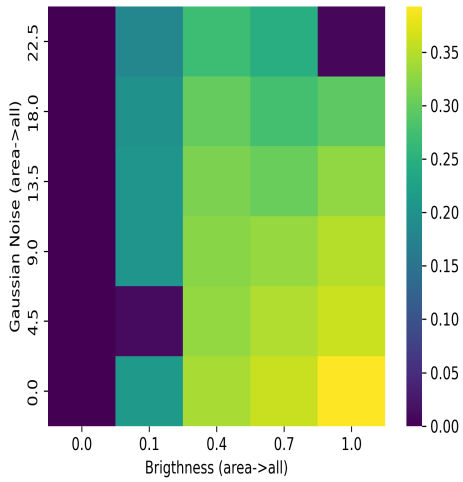


(c) Área de Objeto(Mediano)

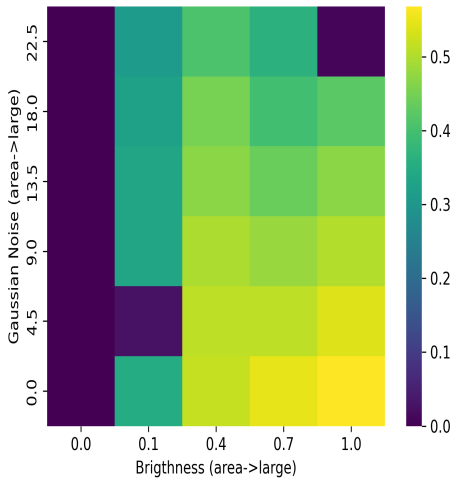


(d) Área de Objeto(Pequeño)

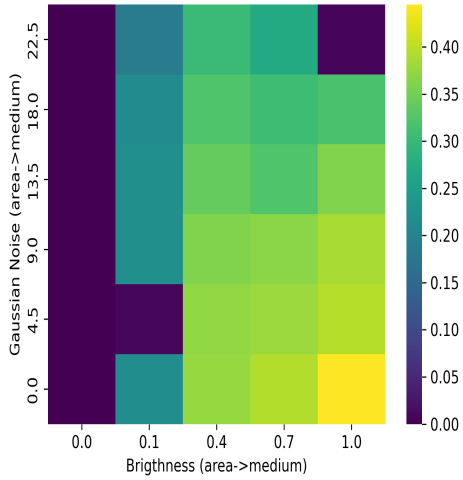
Figura 46: Resultados YOLOv5nu



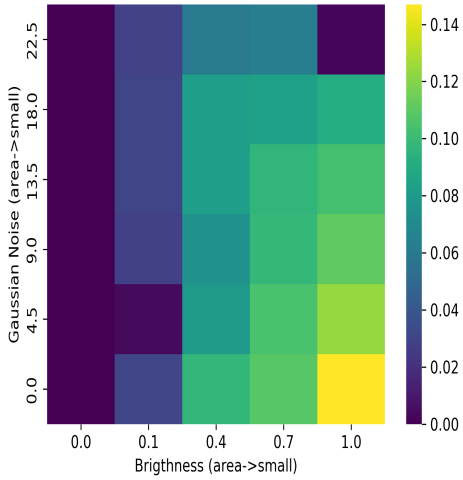
(a) Área de Objeto(Todas)



(b) Área de Objeto(Grande)

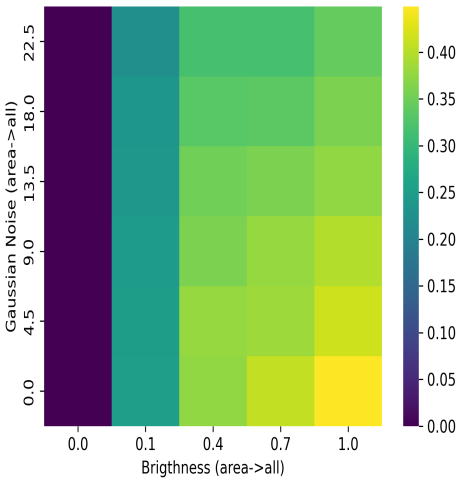


(c) Área de Objeto(Mediano)

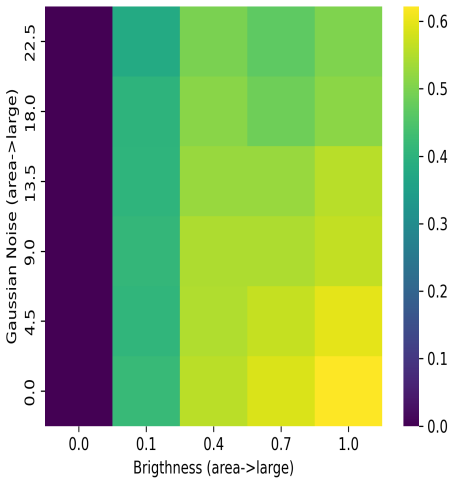


(d) Área de Objeto(Pequeño)

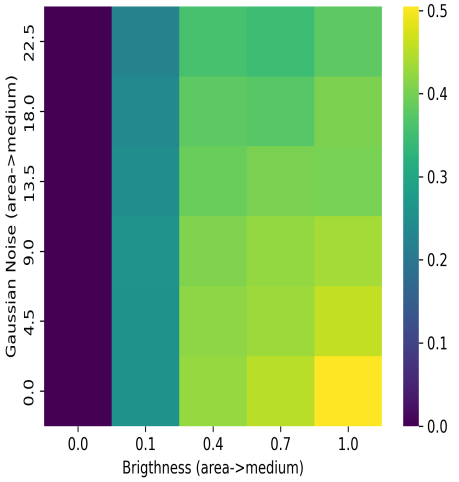
Figura 47: Resultados YOLOv5mu



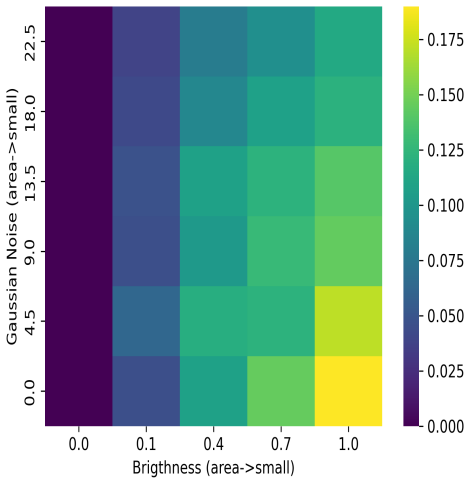
(a) Área de Objeto(Todas)



(b) Área de Objeto(Grande)

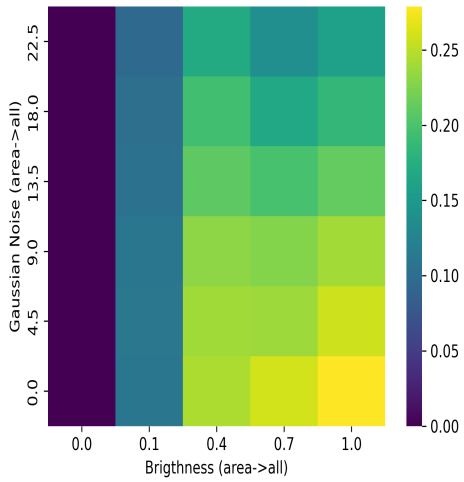


(c) Área de Objeto(Mediano)

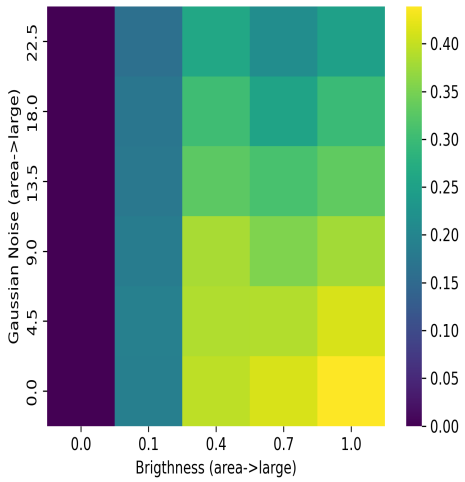


(d) Área de Objeto(Pequeño)

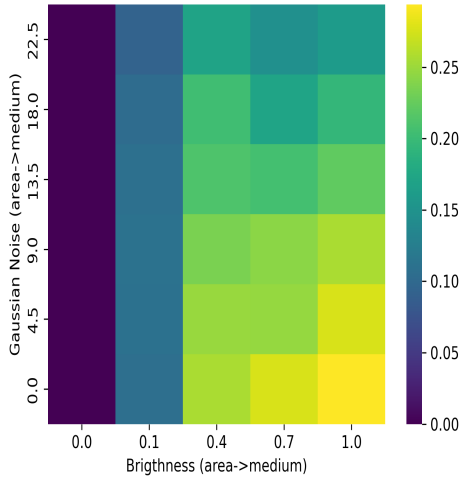
Figura 48: Resultados YOLOv5xu



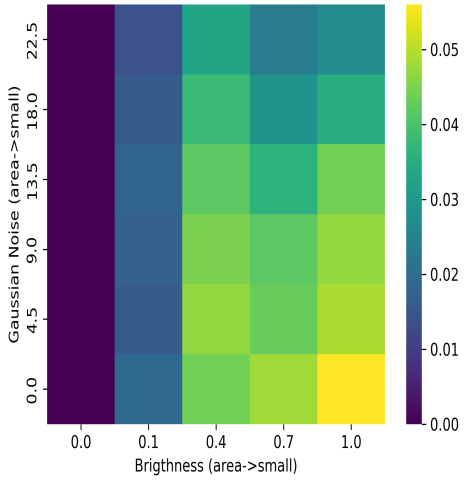
(a) Área de Objeto(Todas)



(b) Área de Objeto(Grande)

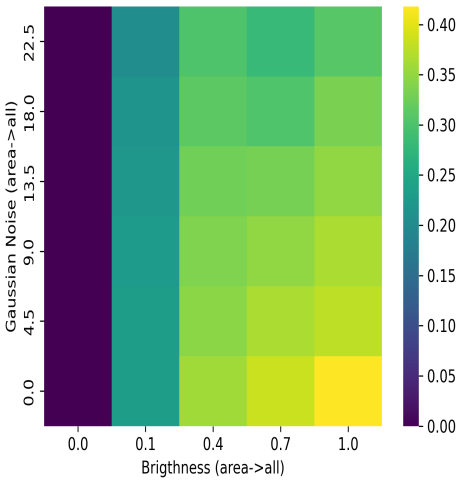


(c) Área de Objeto(Mediano)

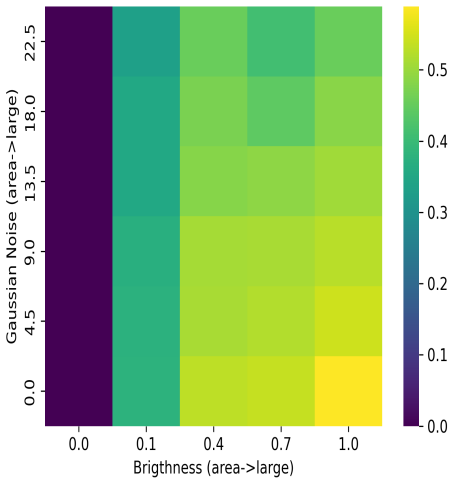


(d) Área de Objeto(Pequeño

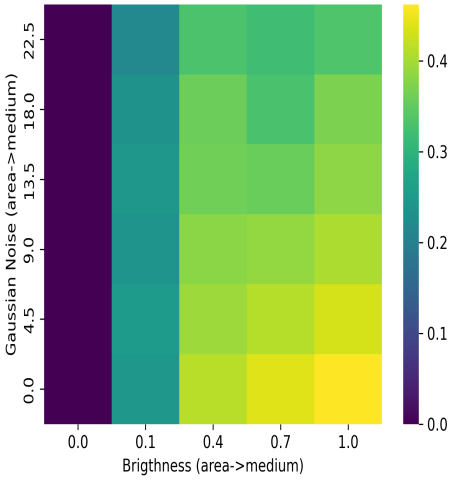
Figura 49: Resultados YOLOv8n



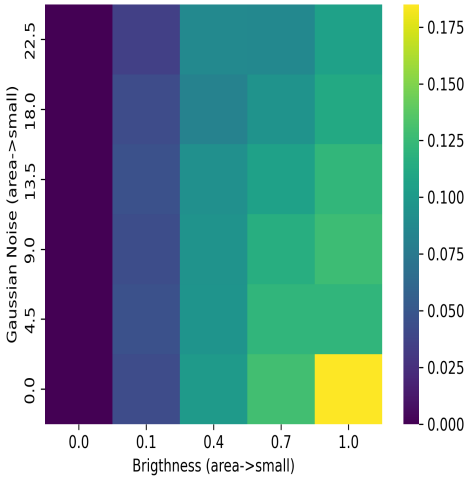
(a) Área de Objeto(Todas)



(b) Área de Objeto(Grande)

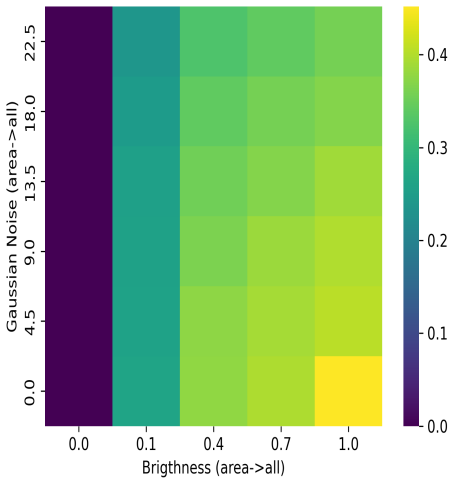


(c) Área de Objeto(Mediano)

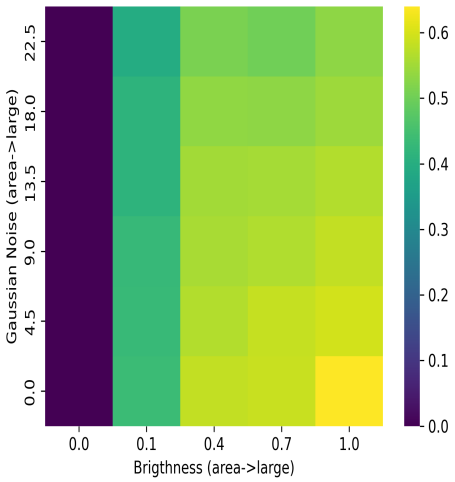


(d) Área de Objeto(Pequeño)

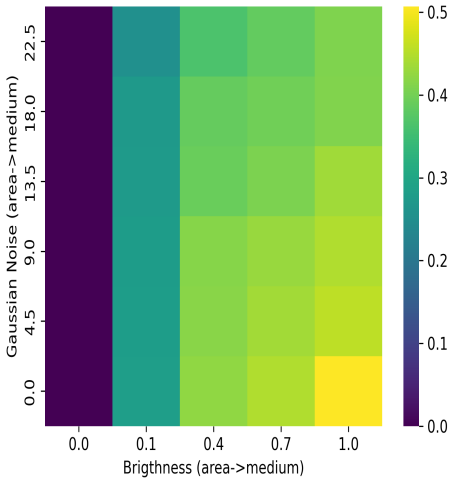
Figura 50: Resultados YOLOv8m



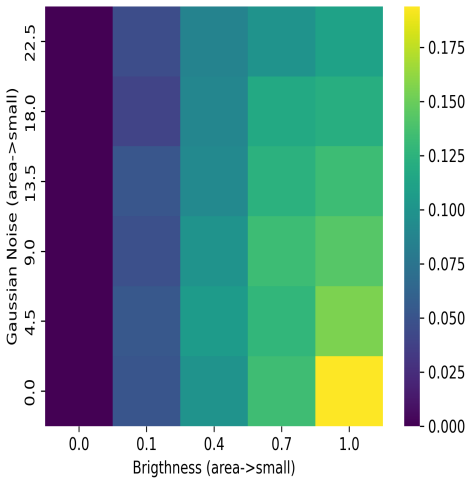
(a) Área de Objeto(Todas)



(b) Área de Objeto(Grande)

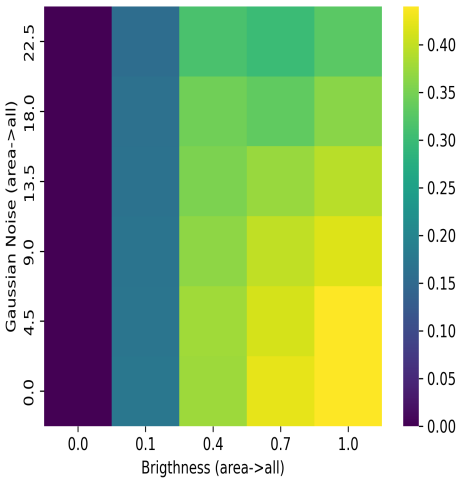


(c) Área de Objeto(Mediano)

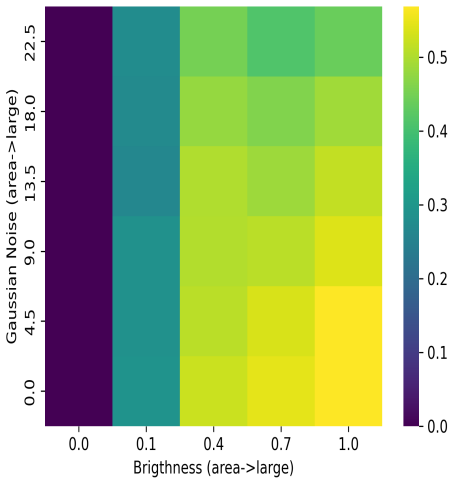


(d) Área de Objeto(Pequeño)

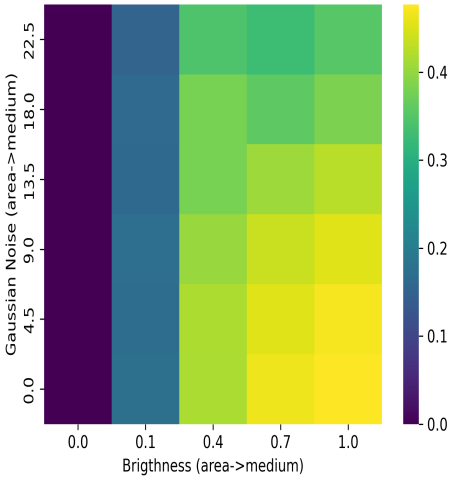
Figura 51: Resultados YOLOv8x



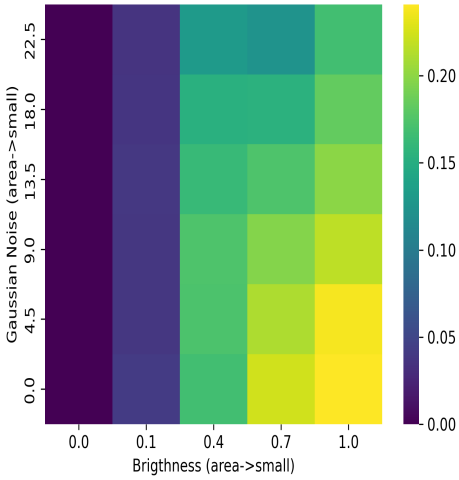
(a) Área de Objeto(Todas)



(b) Área de Objeto(Grande)

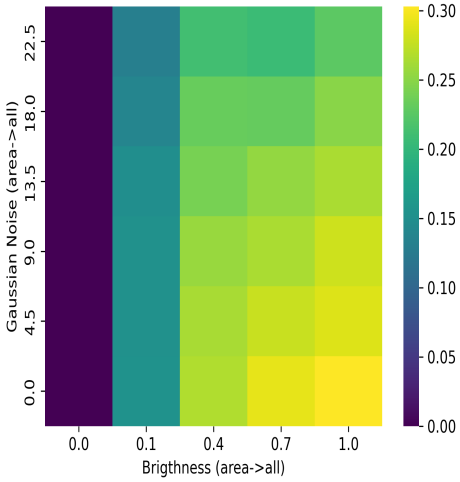


(c) Área de Objeto(Mediano)

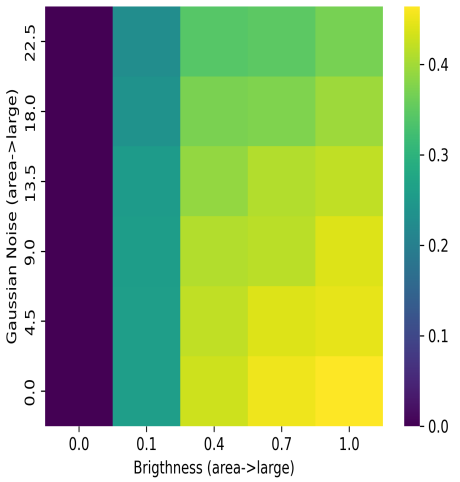


(d) Área de Objeto(Pequeño)

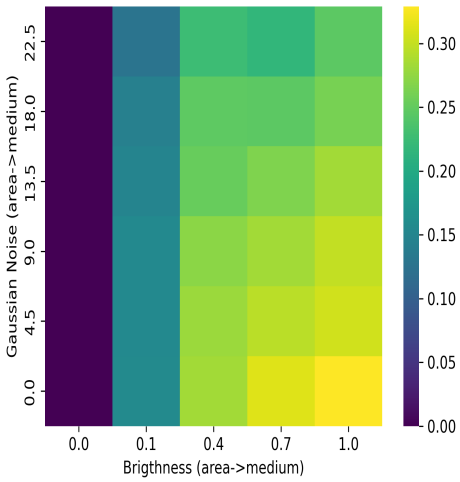
Figura 52: Resultados Faster R-CNN ResNet50 fpn v2



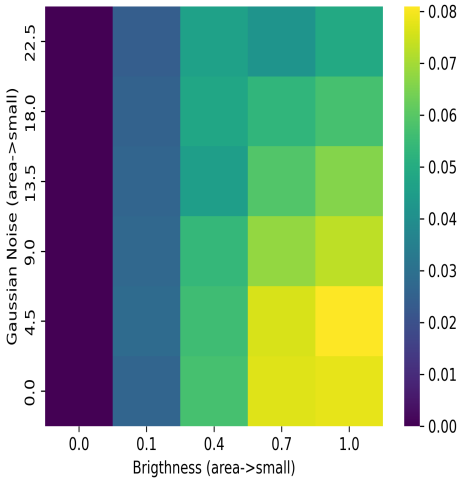
(a) Área de Objeto(Todas)



(b) Área de Objeto(Grande)



(c) Área de Objeto(Mediano)



(d) Área de Objeto(Pequeño)

Figura 53: Resultados Faster R-CNN MobileNet v3 large

Como vemos, en los **heatmaps** mostrados, se hace una segregación entre objetos que consideramos con un área **pequeña (menor a 32x32 píxeles)**, **mediana (32x32 - 96x96 píxeles**, **grande (mayor a 96x96 píxeles)** y otro **heatmap** que incluye a todas las áreas de objetos. Esto se realiza para cada versión de cada modelo estudiado.

Vamos a detenernos en hacer un análisis acerca del efecto provocado por las diferentes afecciones de **ruido gaussiano** y **brillo** en las diferentes detecciones realizadas por **YOLO** y **Faster R-CNN**.

En el análisis cualitativo que realizábamos acerca de los resultados obtenidos con una sola imagen ya podíamos observar como para valores altos de **ruido gaussiano** el **AP** obtenido de las detecciones por **YOLO** se veía reducido considerablemente.

Al realizar un análisis más exhaustivo con una amplitud de valores de ruido y brillo mayor, junto con la segregación realizada por tamaño de objeto, obtenemos una serie de **heatmaps** por cada versión de modelo de detección que nos ilustra de una forma más sencilla como se comporta dicha versión del modelo ante las condiciones pertinentes. Por ello, vamos a realizar un análisis de estos **heatmaps** para cada versión de modelo.

- Para el modelo **YOLO v5nu**, tratándose de la versión más ligera, generalmente en la detección de objetos de cualquier área alcanza valores de **AP** de 0.25 en sus mejores condiciones de brillo y ruido. Destacamos su rendimiento en la detección de objetos de área extensa , pues alcanza hasta 0.4 en cuanto a **AP**, teniendo su peor rendimiento en la detección de objetos de área reducida que no llega a alcanzar los 0.05 de **AP**. A medida que aumentamos a versiones más pesadas de **YOLO**, como la versión **YOLO v5mu**, rinde generalmente con un **AP** promedio de 0.40 en sus mejores condiciones , pero en esta versión se consigue una mejor precisión sobre los objetos de área extensa, alcanzando hasta 0.6 de **AP** . Con la versión pesada de este modelo, **YOLO v5xu** , debería rendir más que el resto de versiones de forma considerable, pero los valores promedios que alcanzan en cualquier área de objeto son bastante similares a los del modelo intermedio, **YOLO v5mu**, anteriormente comentado. Concluyendo con esta versión de **YOLO**, parece ser que la versión que mejor rinde en relación costo computacional y **AP** obtenido, es la versión **YOLO v5mu**, siendo su mejor fruto de detección aquellos objetos con un área de extensión grande, alcanzando valores de **AP** de hasta 0.6.

- Pasando a comentar los resultados obtenidos con **YOLO v8**, podemos comentar que a grandes rasgos se obtiene mejor rendimiento en general que con la versión 5, pero no existe una diferencia notoria. Podemos destacar cualquiera de las versiones de **YOLO v8** que se han estudiado en la detección de objetos de área extensa, donde encontramos a **YOLO v8x** llegando a alcanzar valores de **AP** de hasta 0.7.

- Dejando atrás **YOLO**, comentamos los resultados obtenidos por **Faster R-CNN**, comenzando con **Faster R-CNN ResNet**. Este de partida y generalmente consigue un rendimiento muy similar a **YOLO v5mu**, alcanzando valores de precisión promedio de 0.45 y tiene su máximo esplendor también en la detección de objetos de áreas extensas con un **AP** máximo de 0.6. Tenemos que destacar que este modelo **R-CNN ResNet** tiene mayor precisión promedio en la detección de objetos de área reducida que todos los modelos de **YOLO** expuestos anteriormente, llegando alcanzar un **AP** máximo de 0.25.

- En cuanto a **Faster R-CNN MobileNet** es un modelo muy parecido en rendimiento con las versiones intermedias y más pesadas de **YOLO**, alcanzando un promedio de precisión máximo de 0.30 y particularmente siendo mucho menos preciso en la detección de objetos de área reducida que todos los modelos anteriores, siendo su **AP** máximo de 0.08.

6

Conclusiones y Líneas Futuras

6.1. Conclusiones

Como conclusión final, podemos comentar que las imágenes con **ruido gaussiano** parecen que no alteran en gran medida el rendimiento de estos modelos de detección de objetos cuando se presenta como valores bajos o intermedios, pero si los valores de ruido son altos, reduce considerablemente la precisión promedio en la detección de objetos de dichos modelos. La presencia de **ruido gaussiano** también afecta a la localización del objeto, pues como observamos en el análisis cualitativo, el área del cuadro delimitador de ciertos objetos se puede ver afectada por este ruido, incluyendo regiones de la imagen que no pertenecen al objeto que se esté detectando. En cuanto a la modificación del brillo, si el nivel de brillo que posee la imagen es muy pequeño o moderadamente bajo, afecta más significativamente al rendimiento de las detecciones de objetos, reduciendo considerablemente tanto la precisión asociada a los objetos que se detectan como a la localización de los mismos dentro de la imagen.

Se debe matizar, que al generalizar los resultados de los experimentos realizados, encontramos que todos los modelos estudiados **YOLO V5, YOLOV8, Faster R-CNN RESNET50 FPN V2, Faster R-CNN MOBILENET LARGE FPN V3** particularmente tienen un elevada precisión promedio **(AP)** en detectar objetos con un área (píxeles) considerada como amplia. Destacamos, como caso contrario, la pobre precisión promedio que se obtiene al realizar la detección con estos modelos para objetos cuya área (píxeles) sea pequeña. Además, del rendimiento en cuanto a **AP (Precisión Promedio)** se refiere, es ganador el modelo de detección de objetos mejorado **Faster R-CNN RESNET50 FPN V2**, encontrándose ligeramente cercano **YOLOV8** y **Faster R-CNN MOBILENET LARGE FPN V3**. Aunque los modelos de detección

YOLO son mucho más rápidos y eficientes, pensados para aquellas aplicaciones que requieren una baja latencia, como el análisis y la detección de objetos en vídeos en tiempo real, vehículos autónomos, sistemas de vigilancia en tiempo real, aplicaciones móviles con detección de objetos, control de calidad en líneas de producción, detección de objetos en deportes, detección de objetos en entornos médicos, detección de peatones y vehículos en sistemas de seguridad vial, monitoreo de objetos en logística y almacenamiento ...

Faster R-CNN RESNET50 FPN se utiliza en aplicaciones donde la precisión es una prioridad y la velocidad es un requisito secundario, se puede hacer uso de este modelo para detección de objetos en imágenes de alta resolución, análisis de imágenes médicas, inspección de calidad en manufactura, reconocimiento de objetos en aplicaciones de realidad aumentada, detección de objetos en agricultura de precisión, reconocimiento de objetos en sistemas de asistencia para personas con discapacidades visuales, detección de objetos en seguridad industrial, detección de objetos en análisis de vídeo forense, detección en sistemas de detección y seguimiento de animales ...

6.2. Líneas Futuras

Del estudio hemos podido concluir para qué fines y bajo qué condiciones se comportarían particularmente bien determinados modelos de detección de objetos, unos enfocados más bien en la precisión descuidando la eficiencia y la rapidez, mientras que otros se desenvuelven bastante bien en cuanto a eficiencia y rapidez pero con una precisión reducida.

Este estudio se podría continuar por diversas ramas, una de ellas es prolongar el estudio aplicado al impacto negativo de condiciones de ruido en aplicaciones industriales, como la inspección de calidad, el reconocimiento de objetos o la visión artificial. Si hacemos referencia a la inspección de calidad, además del **ruido gaussiano**, existen otros muchos ruidos y anomalías que dificultarían la detección de defectos o anomalías en productos o componentes.

Otro de los campos afectados, se trata de la sanidad, donde pueden existir consecuencias negativas en casos de uso como la detección de enfermedades o anomalías médicas, como la detección de tumores o lesiones. Las variaciones que introduciría este tipo de ruido y otros conllevarían a una pérdida de detalles importantes en las imágenes médicas dando lugar a errores y alteraciones en la precisión del diagnóstico y el tratamiento. Existiendo otras muchas actividades que podrían verse comprometidas donde actúan este tipo de modelos de detección

o segmentación, como en el análisis de imágenes histológicas o microscópicas o el seguimiento de órganos y estructuras en imágenes de intervenciones en tiempo real.

Por último, otros de los sectores de estudio podría ser las consecuencias de dichos ruidos en tareas de detección y segmentación relacionadas con la agricultura, como la detección de plagas y enfermedades en los cultivos, afectando a la detección precisa de síntomas de plagas o enfermedades en las plantas. También en otras muchas actividades como la clasificación y conteo de frutas, verduras y otros productos agrícolas.

Para abordar estos desafíos, lo común es aplicar técnicas de preprocesamiento de imágenes como la mejora del contraste, el filtrado de ruido o la corrección de iluminación o normalización de las imágenes, consiguiendo reducir de esta forma el impacto de anomalías como el **ruido gaussiano**, ya estudiado, así como otros ruidos y anomalías que también podemos encontrar.

Finalmente, comentar que este estudio se ha realizado con el objetivo de utilizarlo como base para elaborar un artículo científico donde complementaremos los experimentos realizados en este experimento. Al realizar nuevos experimentos complementarios y obtener los nuevos resultados de interés, se enviará para que sea publicado en una revista de alto impacto o en un congreso internacional de alto prestigio, dependiendo de la relevancia de las conclusiones que se obtengan.

Bibliografía

[1] URL: https://www.researchgate.net/figure/Overview-of-the-network-structure-of-a-faster-R-CNN-and-b-YOLO.

[2] URL: https://scielo.org.mx/scielo.php.

[3] URL: https://numpy.org/.

[4] URL: https://ultralytics.com/.

[5] URL: https://pytorch.org/vision/main/models/faster_rcnn.html.

[6] URL: https://cocodataset.org/#detection-eval.

[7] URL: https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn_v2.html#torchvision.models.detection.fasterrcnn_resnet50_fpn_v2.

[8] URL: https://pytorch.org/vision/main/models/generated/torchvision.models.detection.fasterrcnn_mobilenet_v3_large_fpn.html#torchvision.models.detection.fasterrcnn_mobilenet_v3_large_fpn.

[9] *¿Cuáles son los tipos de inteligencia artificial que existen?* URL: https://www.inesdi.com/blog/tipos-de-inteligencia-artificial/.

[10] *¿Qué es el machine learning?: Aprendizaje Automático: UCM*. Jun. de 2021. URL: https://www.masterdatascienceucm.com/que-es-machine-learning/.

[11] *¿Qué es una red neuronal? parte 1 : La Neurona | DotCSV*. Mar. de 2018. URL: https://www.youtube.com/watch?v=MRIv2IwFTPg&list=PL-Ogd76BhmcB9OjPucsnc2-piEE96jJDQ.

[12] *¿Qué es una red neuronal? parte 2 : La Red | DotCSV*. Mayo de 2018. URL: https://www.youtube.com/watch?v=uwbHOpp9xkc&list=PL-Ogd76BhmcB9OjPucsnc2-piEE96jJDQ&index=2.

[13] *Algoritmo you only look once (YOLO)*. Nov. de 2020. URL: https://es.wikipedia.org/wiki/Algoritmo_You_Only_Look_Once_(YOLO).

[14] *Aprendizaje Automático*. Mar. de 2023. URL: https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico#Resumen.

[15] *Aprendizaje por refuerzo y optimización - expertos EN IIC*. Feb. de 2021. URL: https://www.iic.uam.es/inteligencia-artificial/aprendizaje-por-refuerzo/.

[16] *Aprendizaje Semisupervisado*. Nov. de 2022. URL: https://es.wikipedia.org/wiki/Aprendizaje_semisupervisado.

[17] *Aprendizaje Supervisado y no supervisado: Blog ue*. Sep. de 2022. URL: https://universidadeuropea.com/blog/aprendizaje-supervisado-no-supervisado/.

[18] Blog de CEUPE. *Aprendizaje por refuerzo: Concepto, características Y ejemplo*. Abr. de 2022. URL: https://www.ceupe.com/blog/aprendizaje-por-refuerzo.html.

[19] *Distribución de Probabilidad*. Mar. de 2023. URL: https://es.wikipedia.org/wiki/Distribuci%C3%B3n_de_probabilidad.

[20] *El Aprendizaje Profundo (deep learning) en la Optimización de Estructuras*. URL: https://victoryepes.blogs.upv.es/2020/09/15/el-aprendizaje-profundo-deep-learning-en-la-optimizacion-de-estructuras/.

[21] *Explicación de Yolo Para detección de objetos*. Dic. de 2021. URL: https://www.youtube.com/watch?v=3h1_-AaVjWY.

[22] *February learnings–CNNS*. Ago. de 2021. URL: https://prabhjotkaurgosal.com/weekly-learnings/weekly-learning-blogs/.

[23] Ligdi Gonzalez. *Diferencia Entre Aprendizaje Supervisado y no supervisado*. Ago. de 2020. URL: https://aprendeia.com/diferencia-entre-aprendizaje-supervisado-y-no-supervisado/.

[24] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[25] Jonathan Hui. *Map (mean average precision) for object detection*. Abr. de 2019. URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173.

[26] *Inteligencia artificial*. Mar. de 2023. URL: https://es.wikipedia.org/wiki/Inteligencia_artificial.

[27] *Las diferencias entre deep learning Y machine learning*. Feb. de 2020. URL: https://www.jasminsoftware.es/blog/deep-learning/.

[28] *Machine learning: ¿Qué es el aprendizaje automático y cómo funciona?* URL: https://www.algotive.ai/es-mx/blog/machine-learning-que-es-el-aprendizaje-autom%C3%A1tico-y-c%C3%A1mo-funciona.

[29] Miguel Ángel Molina-Cabello et al. "The Effect of Noise and Brightness on Convolutional Deep Neural Networks". En: *ICPR International Workshops and Challenges* (2020).

[30] *NumPy*. Ene. de 2023. URL: https://es.wikipedia.org/wiki/NumPy.

[31] *Predict*. URL: https://docs.ultralytics.com/modes/predict/.

[32] *Pytorch*. Dic. de 2022. URL: https://es.wikipedia.org/wiki/PyTorch.

[33] *Pytorch*. URL: https://proyectoidis.org/pytorch/.

[34] *Pytorch*. URL: https://pytorch.org/.

[35] *Que es r-CNN, fast R-CNN, Faster R-CNN Y mask r-CNN - explicación en español*. Feb. de 2021. URL: https://www.youtube.com/watch?v=C-kBqPIZGo8.

[36] *Redes neuronales convolucionales, ¿Cómo funcionan?* Nov. de 2020. URL: https://www.youtube.com/watch?v=V8j1oENVz00&t=4s.

[37] Wikipedia. *Ruido Gaussiano*. 2022. URL: https://es.wikipedia.org/wiki/Ruido_gaussiano (visitado 15-02-2023).

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga