Universidad Nacional de Mar del Plata
Facultad de Ingeniería
Departamento de Electrónica y Computación

Detección de barcos en imágenes SAR con modelos basados en CNN para el control marítimo y protección del medio ambiente

Proyecto Final Ingeniería en Computación

Bozzalla Bondio, Joaquín Matías
bozzallajoaquin@gmail.com

Silva, Juan Jose
juanjosesilva97@gmail.com

Directora: Dra. Leticia M. Seijas

Codirector: Ing. Jorge Marquez

El presente trabajo de Proyecto Final fue realizado en el Laboratorio de Comunicaciones del Departamento de Electrónica y Computación, ICyTE, Facultad de Ingeniería, Universidad Nacional de Mar del Plata.

Mar del Plata, 2022

Universidad Nacional de Mar del Plata
Facultad de Ingeniería
Departamento de Electrónica y Computación

Detección de barcos en imágenes SAR con modelos basados en CNN para el control marítimo y protección del medio ambiente

Proyecto Final Ingeniería en Computación

Bozzalla Bondio, Joaquín Matías
bozzallajoaquin@gmail.com

Silva, Juan Jose
juanjosesilva97@gmail.com

Directora: Dra. Leticia M. Seijas

Codirector: Ing. Jorge Marquez

El presente trabajo de Proyecto Final fue realizado en el Laboratorio de Comunicaciones del Departamento de Electrónica y Computación, ICyTE, Facultad de Ingeniería, Universidad Nacional de Mar del Plata.

Mar del Plata, 2022

Resumen

Los satélites de Radar de Apertura Sintética poseen cada vez más protagonismo en el campo de la observación de la Tierra y la vigilancia marítima. Dada la gran cantidad de datos generados por las plataformas satelitales se requiere el uso de técnicas avanzadas para la extracción de información útil de ellos. La detección de objetos es una disciplina del aprendizaje profundo que reconoce y localiza patrones dentro de una imagen. Este trabajo propone un modelo con arquitectura YOLO v4 entrenado con el conjunto de datos HRSID utilizando Transfer Learning que obtiene un desempeño superior comparado con resultados presentes en la literatura. Por último se pone a prueba el modelo ante escenas capturadas con los satélites Sentinel 1 y SAOCOM 1 que no estuvieron presentes en el entrenamiento.

Palabras Clave: Detección de barcos, Aprendizaje profundo, YOLO v4, SAR, HRSID.

Abstract

Synthetic Aperture Radar satellites are increasingly playing a leading role in the field of Earth observation and maritime surveillance. Given the large amount of data generated by satellite platforms, the use of advanced techniques is required to extract useful information from them. Object detection is a Deep Learning discipline that recognizes and locates patterns within an image. This work proposes a model with YOLO v4 architecture trained with the HRSID dataset using Transfer Learning, which obtains superior performance compared to results found in the literature. Finally, the model is tested against scenes captured with the Sentinel 1 and SAOCOM 1 satellites that were not present in the training.

Keywords: Ship detection, Deep Learning, YOLO v4, SAR, HRSID.

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

Con el desarrollo de plataformas satelitales cada vez más económicas y sensores cada vez más avanzados, el área de la observación de la Tierra posee un gran potencial de explotación acompañando el actual auge de la industria espacial. Debido a la gran cantidad de datos recolectados por estos sensores surge la necesidad de utilizar técnicas modernas para la extracción de conocimiento útil de ellos.

La inteligencia artificial [1], o AI, surge del afán del ser humano en reproducir su comportamiento inteligente mediante computadoras. Uno de estos comportamientos es la capacidad de aprender, creándose así una de las disciplinas más importantes de la AI: el aprendizaje automático. Algunas técnicas de esta disciplina permiten detectar patrones en datos mediante modelos entrenados con datos conocidos que luego generalizan el conocimiento obtenido a otros casos nuevos no conocidos por el modelo. De esta forma se logra aprender de experiencias pasadas y generalizar ese conocimiento para aplicarlo a experiencias futuras.

El reciente auge del Deep Learning (DL) ha mejorado el estado del arte de muchos campos de la inteligencia artificial como la detección de objetos o el reconocimiento de patrones, entre otros. En particular, las Redes Neuronales Convolucionales (CNN), un tipo específico de DL, han obtenido gran aceptación debido a los buenos resultados alcanzados y la facilidad en su entrenamiento frente a otros tipos de DL.

Utilizado desde hace varias décadas pero tomando un papel cada vez más relevante, la tecnología de radar de apertura sintética se posiciona como una de las más prometedoras en el campo de la observación terrestre en los próximos años. Un radar de apertura sintética (SAR) emite un pulso electromagnético modulado en frecuencia, este se refleja en la interfaz entre aire y superficie terrestre, y luego esa energía vuelve a la antena que emitió el pulso. De esa manera se obtiene información de las características de la superficie terrestre que se visualizan en forma similar a un mapa.

Desde hace varios años, las imágenes SAR son utilizadas para la detección de barcos [2] siendo de gran utilidad para el control marítimo y protección del medio ambiente. Al utilizar técnicas de Deep Learning se abre un mundo de posibilidades para aprovechar las características de las imágenes SAR. A su vez, estas mismas características presentan un desafío debido a que los modelos de CNN están orientados a imágenes del tipo óptico. Dentro del área del control marítimo esta tecnología puede ser utilizada para detectar barcos ilegales, administración de puertos, búsqueda y rescate, vigilancia de zonas económicas exclusivas y control de pesca. Además, es de vital importancia para el manejo y protección de los recursos naturales.

En este trabajo se estudia el estado del arte de la detección de barcos en imágenes SAR y se propone un modelo de Deep Learning con arquitectura YOLO v4 entrenado con Transfer Learning que supera los resultados de la literatura sobre el dataset HRSID. Además se pone a prueba el modelo propuesto al utilizarlo en escenas capturadas por los satélites Sentinel 1 y SAOCOM 1.

Este trabajo se organiza de la siguiente manera: el Capítulo 2 describe el concepto de sensado remoto y la tecnología SAR. El Capítulo 3 explica qué es la inteligencia artificial haciendo hincapié en el Deep Learning. En el Capítulo 4 se habla sobre el reconocimiento de objetos dentro del mundo de la visión computacional. El Capítulo 5 detalla los hiperparámetros existentes y las técnicas disponibles para optimizarlos. Las distintas plataformas de hardware para entrenar modelos haciendo foco en el Cloud Computing son descritas en el Capítulo 6. En el Capítulo 7 se describen los conjuntos de datos que fueron utilizados y cómo fueron creados. En el Capítulo 8 se explica cómo fue entrenado el modelo propuesto y se analizan los resultados obtenidos. Por último el Capítulo 9 presenta las conclusiones y los trabajos futuros.

Capítulo 2

Sensado remoto

2.1. Introducción

En un sentido amplio, el sensado remoto es la acción de realizar una medición a la distancia. Esta definición, aplicada a este trabajo en particular, se puede reformular como la obtención de información de la superficie de la Tierra desde la distancia y sin estar en contacto con el objetivo [3]. El sensado puede realizarse de forma activa o pasiva dependiendo si la fuente de energía es el mismo sensor o si por el contrario proviene de otra fuente como el Sol.

La utilización de los datos de los sensores requiere un análisis y una interpretación para convertir los datos en información que pueda utilizarse para resolver problemas prácticos, como la ubicación de derrames de petróleo o controlar la deforestación de bosques. Los mismos datos de los sensores pueden analizarse desde distintas perspectivas para obtener diferentes interpretaciones. Así, una misma imagen puede interpretarse para proporcionar información sobre los suelos, el uso de la tierra o la hidrología, por ejemplo.

El proceso de sensado remoto implica la interacción entre la radiación incidente y el objeto de interés y está caracterizado por la participación de los siguientes siete elementos [3]:

(a) Fuente de energía o iluminación: Debe existir una fuente de energía que ilumine o provea energía electromagnética al objetivo. En los sensores pasivos, que veremos después en más detalle, es el Sol quien provee energía y los sensores activos proveen su propia energía.

(b) Radiación y atmósfera: Cuando la energía viaja desde la fuente al objetivo, o viceversa, entra en contacto e interactúa con la atmósfera. Esta interacción puede provocar dos efectos conocidos como dispersión (scattering) y absorción. La dispersión se da cuando al atravesar la atmósfera las ondas de luz se desvían de su camino original. Este fenómeno no ocurre en las ondas de radio debido a que su longitud es mayor. La absorción, como su nombre lo indica, se da cuando la atmósfera absorbe parte de la energía transmitida.

(c) Interacción con el objetivo: Cuando la energía atraviesa la atmósfera y llega al objetivo, interactúa con él dependiendo de las características de ambos. La interacción puede darse de tres formas: absorción, transmisión y reflexión. Toda la energía interactúa de una o varía de estas formas. La proporción de cada una depende de la

longitud de onda y del material del objetivo. La absorción se produce cuando la radiación es absorbida por el objetivo, mientras que la transmisión se produce cuando la radiación atraviesa el objetivo. La reflexión se produce cuando la radiación rebota en el objetivo y se redirige. El porcentaje de radiación que se refleja en la superficie terrestre se denomina albedo, las superficies claras, como la nieve por ejemplo, tienen valores de albedo muy alto por lo que la mayoría de la radiación incidente se refleja, en cambio, las superficies oscuras como el asfalto o los océanos, tienen un albedo muy bajo reflejando un porcentaje pequeño de la radiación incidente [4].

(d) Registro de la energía por el sensor: Después de que la energía haya sido retrodispersada por, o emitida por el objetivo, necesitamos un sensor remoto, es decir que no esté en contacto con el objetivo, para recoger y registrar la radiación electromagnética.

(e) Transmisión, recepción y procesamiento: La energía registrada por el sensor puede ser transmitida a una estación de recepción donde los datos se procesan en una imagen. Otra posibilidad, es que sean almacenados para su posterior descarga y procesamiento en tierra en los casos donde la plataforma sea un avión o dron.

(f) Interpretación y análisis: Se utilizan algoritmos y procesamiento específico sobre las imágenes adquiridas para extraer información sobre el objetivo que fue iluminado.

(g) Aplicación: El elemento final del proceso se logra cuando se aplica la información que se ha podido extraer de las imágenes sobre el objetivo para comprenderlo mejor, revelar alguna información nueva o ayudar a resolver un problema concreto.

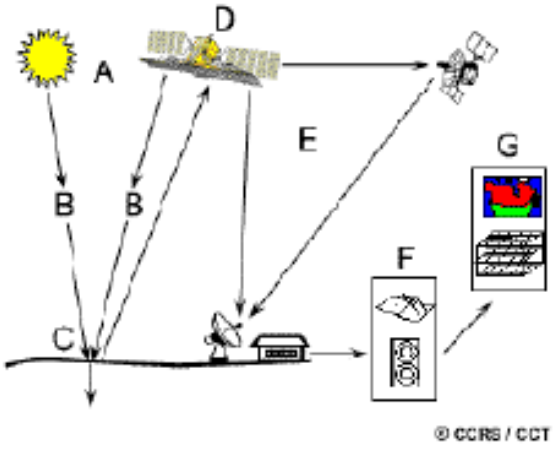


Fig. 2.1: Proceso del sensado remoto

2.2. Historia del sensado remoto

Al consistir en la obtención y análisis de imágenes, el sensado remoto tiene sus orígenes en conjunto con el nacimiento de la fotografía en el siglo XIX. La adquisición de las primeras fotografías aéreas fue realizada en globos aerostáticos simplemente por curiosidad y no con fines prácticos.

Durante la primera guerra mundial la adquisición de fotografías aéreas tomadas desde aeroplanos fue de gran utilidad en el reconocimiento y vigilancia militar. La complejidad del campo del sensado remoto comenzó a aumentar debido a su gran potencial. Hubo grandes avances en la tecnología de las cámaras y se comenzó a utilizar la fotogrametría, ciencia que consiste en realizar mediciones precisas en fotografías. La llegada de la segunda guerra mundial potenció el rápido crecimiento del área en el campo militar siendo el período de Guerra Fría uno de los más destacados. En este período se desarrollaron los aviones U2 y la serie de satélites CORONA, ambos programas enfocados al espionaje por parte de los Estados Unidos.

En la década del 50 comenzó a utilizarse el sensado remoto con aplicaciones civiles como por ejemplo para cartografía. Uno de los grandes hitos fue la puesta en órbita del primer satélite meteorológico TIROS-1 en 1960. Posteriormente, en 1972, fue lanzado el Landsat-1 siendo el primer satélite destinado a la observación de la tierra. Landsat proporcionó por primera vez en la historia imágenes satelitales de forma sistemática y repetitiva. A finales de 1990 ya se tenían sistemas con gran resolución que, junto con el desarrollo de sistemas de información geográfica, abriendo un sin fin de posibilidades de aplicaciones comerciales. La creación de mapas de infraestructura urbana, asistencia a la agricultura o el mapeo de inundaciones son algunos ejemplos.

2.3. Características de las plataformas

Para que un sensor pueda recolectar información sobre la energía reflejada o emitida por una superficie debe estar ubicado en una plataforma estable alejada del objetivo. Existen variadas plataformas donde los sensores pueden ser ubicados, por ejemplo, en el suelo, en un avión o en un satélite.

Los sensores en el suelo pueden estar ubicados en escaleras, edificios o grúas y su uso está orientado a cubrir con mucho detalle pequeñas porciones de la superficie. En cambio, los sensores ubicados en aviones o helicópteros permiten observar regiones más grandes a un mayor costo. Por último, los sensores ubicados en satélites, que giran en órbitas alrededor de la Tierra, pueden obtener información de toda la superficie de la Tierra en forma repetitiva cada cierto intervalo de tiempo.

Las características de cada una de las plataformas se deberán tener en cuenta al momento de elegir cuál se utilizará. El costo, el área de la superficie de interés y el intervalo de tiempo entre muestras son factores que influyen en la decisión. En este trabajo se abordará principalmente los sensores ubicados en satélites, debido a su cobertura mundial y la disponibilidad de los datos, siendo hoy en día, la plataforma más utilizada para el sensado remoto.

Actualmente, de acuerdo a la Union of Concerned Scientists (UCS) [5], hay más de 3300 satélites en órbita alrededor de la Tierra, y más de 900 están orientados a la observación del planeta.

Rango del espectro electromagnético

La energía electromagnética viaja en forma de onda a través de la atmósfera y del vacío del espacio. Estas ondas pueden ser de longitudes y frecuencias muy variadas. El diagrama del espectro electromagnético es una buena herramienta visual que ayuda a entender las diferentes frecuencias y donde las podemos encontrar en la naturaleza o en las tecnologías que nos rodean. El ojo humano es capaz de percibir solo un pequeño rango de longitudes

de ondas comprendidas entre los 380 nm y los 750 nm. Si se quiere detectar otros tipos de longitudes entonces se necesita instrumentación especializada.

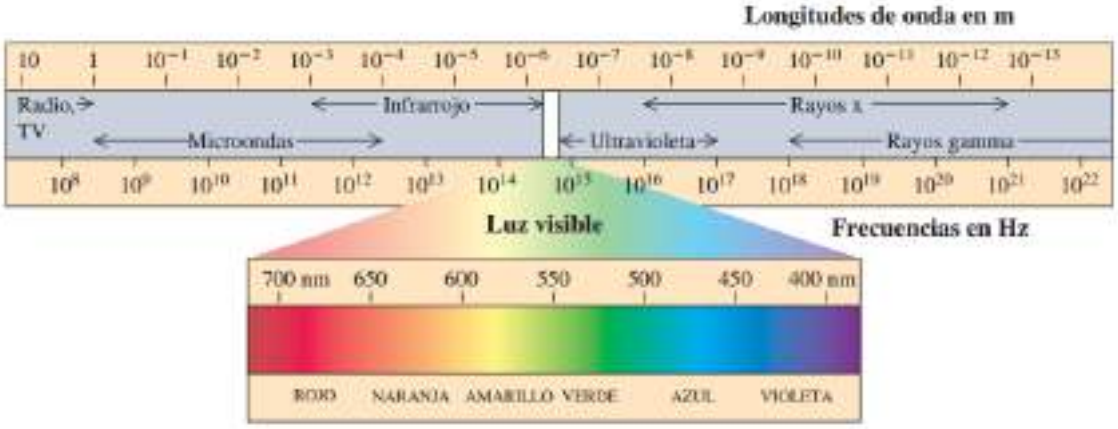


Fig. 2.2: Rango del espectro electromagnético

Fuente de energía

Los sensores se pueden clasificar en pasivos o activos en función de donde proviene la energía que utilizan.

Los sensores pasivos son los que utilizan como fuente de energía al Sol y capturan la energía que es reflejada por la superficie de la tierra o absorbida y luego reemitida como infrarrojo térmico. Estos sensores, al necesitar al Sol como fuente de iluminación solo pueden operar cuando este está disponible, la energía reflejada sólo tiene lugar durante el día. Otra limitación de estos sensores es que, debido a las longitudes de ondas que utilizan, pueden no penetrar densas capas de nubes imposibilitando la detección, es decir, son sensibles al estado meteorológico de la región.

En contraposición, los sensores activos son aquellos que emiten su propia energía para la iluminación del área que van a sensar. La mayoría de los sensores activos operan en la banda de microondas del espectro electromagnético, lo que les da la capacidad de penetrar en la atmósfera en la mayoría de las condiciones, independientemente de la hora del día, la estación del año o el estado meteorológico. Sin embargo, los sistemas activos requieren la generación de una cantidad bastante grande de energía para iluminar adecuadamente los objetivos. Algunos ejemplos de sensores activos son los radares de apertura sintética (SAR) o los sensores de medición y detección de luz (LiDAR).

Órbita

El camino o trayectoria que sigue un satélite se denomina órbita. La órbita elegida para cada satélite está relacionada con la capacidad de los sensores y con el objetivo que se quiere observar. Las órbitas pueden variar en altitud (altura sobre la superficie de la Tierra), orientación y rotación relativa a la Tierra.

Los satélites ubicados a gran altitud, aproximadamente a 36.000 kilómetros, que giran a la misma velocidad que la tierra, por lo que observan siempre la misma superficie terrestre, se denominan geoestacionarios. Estas órbitas permiten a los satélites observar

Fig. 2.3: Sensores pasivos y activos

y obtener información de forma continua sobre áreas específicas de la tierra. Los satélites meteorológicos como el GOES y de comunicaciones como ARSAT suelen tener este tipo de órbitas.

La mayoría de los satélites de observación de la Tierra siguen una órbita polar, es decir, con una inclinación de aproximadamente 90 grados sobre el ecuador. Esta inclinación en conjunto con el movimiento de rotación de la Tierra permite al satélite cubrir toda la superficie de la Tierra cada cierto periodo de tiempo. Muchas de las órbitas de estos satélites son también sincrónicas con el sol, de modo que cubren cada zona del mundo a la misma hora del día. En cualquier latitud, la posición del sol en el cielo cuando el satélite pasa por encima será la misma dentro de la misma estación. Esto garantiza unas condiciones de iluminación constantes cuando se adquieren imágenes en una estación concreta a lo largo de años, o en una zona concreta a lo largo de días. En estas órbitas polares el satélite viaja del polo sur al polo norte en un lado de la Tierra y del polo norte al polo sur del otro lado, estas mitades de las órbitas se conocen como pasajes ascendentes y descendentes, respectivamente. Si esta órbita es además sincrónica al sol el paso descendente ocurre en el lado iluminado por el sol y el paso ascendente en el lado sombreado. Los sensores pasivos, que utilizan al sol como fuente de energía solo pueden tomar imágenes en el momento que el sol está iluminando la Tierra, generalmente esto ocurre en el pasaje descendente y en algunos momentos del pasaje ascendente. Además estos sensores pueden detectar el infrarrojo térmico en el pasaje ascendente, mientras que los sensores activos toman imágenes en ambos pasajes.

Resoluciones

La resolución tiene un rol muy importante en cómo la información obtenida por un satélite puede ser utilizada. La resolución puede variar dependiendo la órbita del satélite y el diseño del sensor. Hay cuatro tipos de resoluciones que podemos considerar [6]: radiométrica, espacial, espectral y temporal.

La **resolución radiométrica** es la cantidad de información almacenada en un píxel, es decir, el número de bits que representan ese píxel. Cuanto mayor sea la resolución radiométrica, más valores diferentes se pueden almacenar, haciendo posible que se pueda discriminar mejor entre pequeñas diferencias de energía.

La **resolución espacial** está definida por el tamaño de cada píxel en una imagen y el área de la superficie de la Tierra que representa. Cuanto menor sea el valor de la resolución espacial, mayor será el detalle que muestre la imagen, pudiendo distinguir objetos más

pequeños como casas o autos.

La **resolución espectral** es la capacidad de un sensor para percibir más bandas y más estrechas. Cuanto más estrecha sea la gama de longitudes de onda de una banda determinada, más fina será la resolución espectral y cuando más bandas incluya un sensor, más características de esa porción de superficie obtendrá.

Por último, la **resolución temporal** es el tiempo que le toma a un satélite completar una órbita y visitar nuevamente el mismo área de observación. La resolución temporal depende de la órbita, las características del sensor y del ancho del swath. Para los satélites geoestacionarios, la resolución temporal es el tiempo que demora en tomar una nueva imagen, entre 30 segundos y 1 minuto, en cambio, para los satélites con órbitas polares la resolución puede variar entre 1 y 16 días.

Al momento de construir un sensor se debe hacer un equilibrio entre las resoluciones en función de la aplicación que van a tener los datos obtenidos, esto se debe a que no se puede construir un sensor con la mejor de todas las resoluciones ya que no son compatibles. Por ejemplo, para obtener una gran resolución espacial, se requiere un swath más estrecho, por lo que se necesita mayor tiempo para una revisita aumentando la resolución temporal. Otro factor importante a tener en consideración, particularmente en los sensores SAR, es el ancho de banda, que se abordará en más detalles en las siguientes secciones. Teniendo en cuenta cuál será la aplicación del sensor se debe elegir las características que más se adecuen.

2.4. Sensado remoto activo mediante radar

Un radar, término que proviene del inglés radio detection and ranging, es un sensor electromagnético que sirve para detectar y localizar objetos a una distancia considerable. Esto se logra transmitiendo pulsos electromagnéticos hacia los objetivos y observando los ecos que retornan de ellos, es decir, las señales reflejadas [7]. Estos objetivos pueden ser de cualquier tipo, ya sea aviones y embarcaciones o hasta aves, insectos y cuerpos astronómicos.

El desarrollo de sistemas de radar tomó gran impulso durante las décadas de 1930 y 1940 con fines totalmente militares. Actualmente, además de usarse en el sector militar, tiene muchas aplicaciones civiles. El control de tráfico aéreo, meteorología, navegación de embarcaciones y observación planetaria son algunos ejemplos de su aplicación.

Un sistema de radar consiste en un transmisor que produce señales electromagnéticas y las irradia al espacio por medio de una antena. Al alcanzar un objeto, la señal es reflejada en varias direcciones. Este reflejo es recibido por la antena del radar que está conectada a un receptor donde la señal es procesada. La distancia al objeto se determina midiendo el tiempo que toma la señal en viajar desde el radar hacia el objeto y volver. Por lo general, los sistemas de radar no transmiten y reciben al mismo tiempo, sino que lo hacen de forma alternada, esto hace que el uso de una sola antena sea suficiente.

2.5. Radar de apertura sintética

2.5.1. Introducción

Al hablar de sensado remoto, la tecnología de radar de apertura sintética, o SAR, es una forma completamente diferente de generar una imagen del terreno al iluminarlo activamente en vez de utilizar la luz del sol como en las imágenes ópticas. Esta tecnología

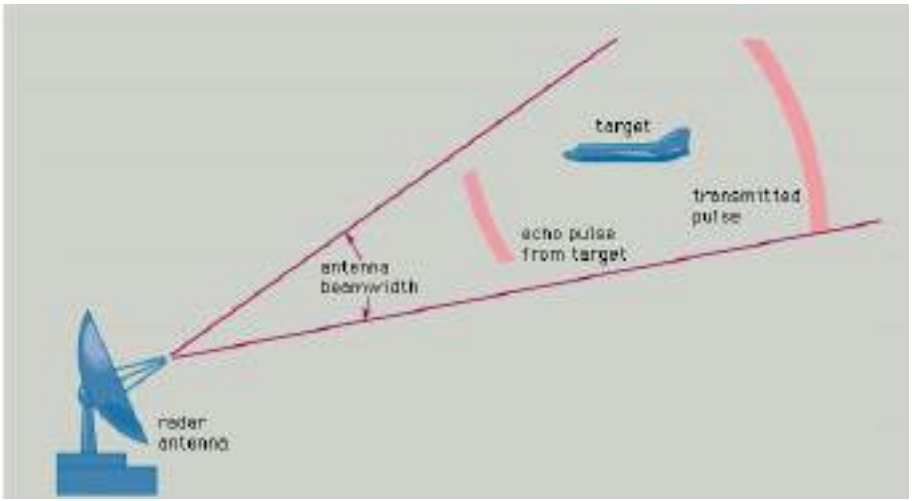


Fig. 2.4: Esquema del funcionamiento de un radar

tiene la capacidad de proveer imágenes de alta resolución espacial independientemente de la luz solar, la cobertura de nubes y de las condiciones climáticas.

Los sistemas SAR son un radar de pulsos instalado en una plataforma que se mueve hacia adelante. El radar transmite pulsos electromagnéticos de alta potencia y recibe, de forma secuencial, los ecos de la señal dispersada. La amplitud y fase de la señal dispersada por un objetivo depende de sus propiedades físicas, como la geometría o la aspereza, y de las propiedades eléctricas, como la permeabilidad.

Los sistemas más simples de radar proveen un mapa en dos dimensiones de la reflectividad en un área. Los objetivos o targets de los cuales se recibe alta dispersión son identificados como puntos brillantes, mientras que las superficies lisas son vistas como zonas negras.

La dirección en la cual se mueve la plataforma se llama **azimuth**, y la dirección perpendicular a la dirección de vuelo se llama **slant range**. Además, el **Swath** width determina la extensión de suelo de la escena, es decir, el ancho del barrido del sensor. Por otro lado, el largo de la escena depende de la duración de la toma de datos, cuanto mayor sea la duración de la toma de datos más extensa será la escena. Otro parámetro para destacar es el tiempo de iluminación Till por el cual un target es observado.

Los primeros sistemas de radar llamados SLAR no utilizaban los principios de apertura sintética. Esto hacía que la resolución conseguida en la dirección de azimuth no sea muy buena. La resolución azimuth está dada por la separación más pequeña que puede ser detectada entre dos targets.

El problema de la resolución de azimuth fue resuelta con el uso de un radar coherente junto con los principios de Doppler beam sharpening. Posteriormente, este concepto se extendió a lo que hoy conocemos como apertura sintética.

2.5.2. Principios básicos

Un sistema SAR está montado en una plataforma en movimiento. Los pulsos electromagnéticos son transmitidos de forma secuencial y el eco de la dispersión es obtenido por la antena. El hecho de que la transmisión y recepción sea secuencial hace que sucedan cuando la plataforma está en diferentes posiciones ya que está en movimiento. Esto, sumado a una

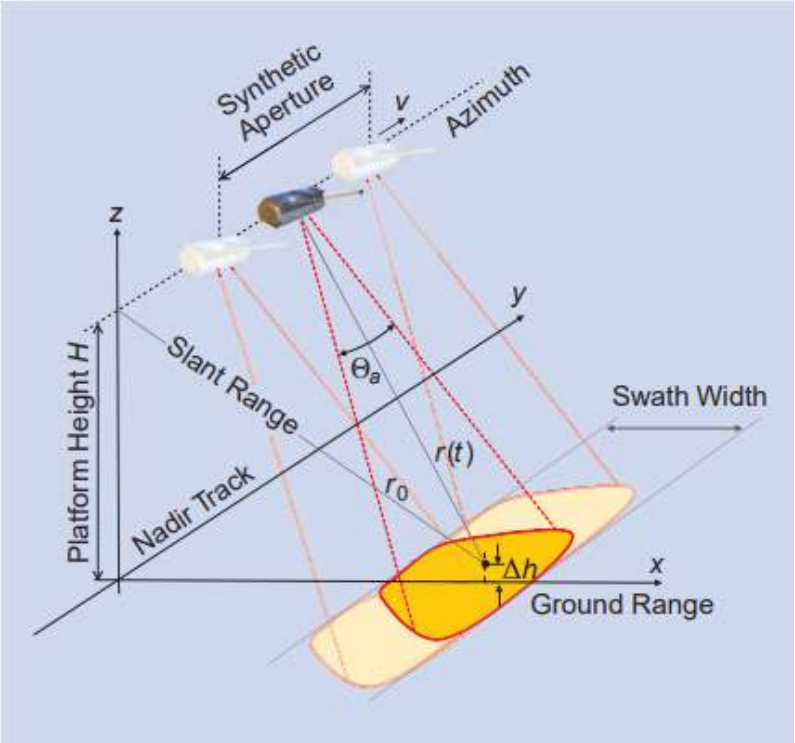


Fig. 2.5: Dirección Azimuth, Slant Range y Swath de un satélite

combinación coherente de las señales recibidas permiten la construcción de una apertura virtual mucho más larga que el largo de la antena real. A partir de este concepto es que nace el término de **apertura sintética** que le da el nombre a la tecnología.

Una técnica relacionada a SAR es el uso arreglos de antenas, o también llamado phased array, que consiste en utilizar varias antenas distribuidas en una o dos dimensiones, es decir, colocando las antenas formando una línea o varias, con el objetivo de lograr que el haz de la señal sea más estrecho. Todos los elementos del arreglo reciben o transmiten de forma simultánea, y las señales son tratadas individualmente por cada antena del arreglo. Al utilizar esta tecnología, es posible cambiar la dirección y el patrón de radiación de las señales transmitidas sin tener que mover las antenas. Esto se logra transmitiendo por todas las antenas con la misma frecuencia, pero controlando el desfase entre ellas. El desfase debe ser tal que las señales interfieran constructivamente en la dirección deseada y destructivamente en las demás direcciones.

Si bien los arreglos de antenas tienen varias aplicaciones, en el caso de radar se utilizan para iluminar un blanco de forma más precisa y para conocer con exactitud la dirección de las señales recibidas. Haciendo una analogía con un sistema SAR, se puede pensar a un satélite SAR que viaja a velocidad constante V y que realiza un ciclo de transmisión/recepción cada cierta cantidad de tiempo T, como un arreglo de antenas con una distancia entre sí de $V \cdot T$.

Los sensores SAR utilizan pulsos modulados en frecuencia, también llamadas señales **chirp**. La amplitud de la onda transmitida es constante durante la duración del pulso t, mientras que la frecuencia a cada instante varía linealmente de acuerdo con $f = k \cdot t$, donde k es el chirp rate. Luego, el radar "escucha" y almacena los ecos durante un tiempo que se denomina **echo windows**.

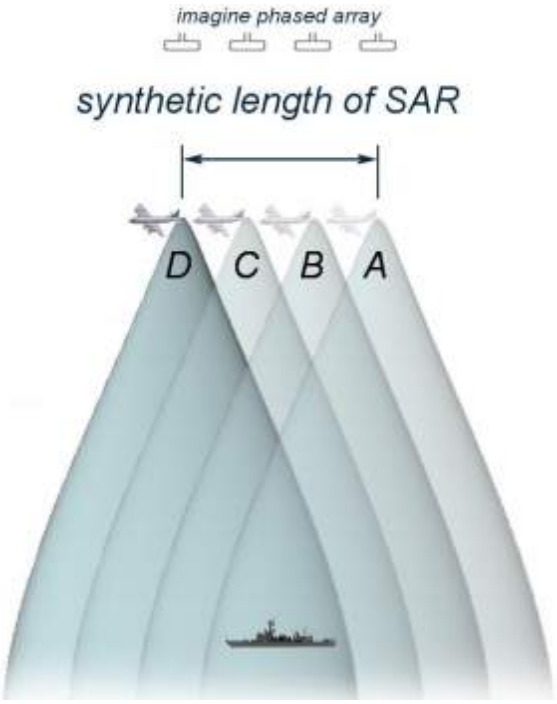


Fig. 2.6: Sistema SAR visto como un array de antenas

La transmisión y recepción es repetida cada PRI segundos, donde el intervalo de repetición del pulso (PRI) es la inversa de la frecuencia de repetición del pulso $PRI = \frac{1}{PRF}$.

Al tratar con imágenes de radar entran en juego dos resoluciones. Por un lado, está la resolución slant range δ_r, es decir, la resolución en la dirección perpendicular a la plataforma, que es inversamente proporcional al ancho de banda del sistema según la ecuación $\delta_r = \frac{Co}{2BR}$ donde Co es la velocidad de la luz. En cambio, la resolución azimuth δ_a depende de la construcción de la apertura sintética. Esta última es la dirección en que se mueve la plataforma. Se desea que la apertura sintética sea lo más larga posible ya que resulta en un ancho de pulso más focalizado. Esto se consigue con una antena más corta.

2.5.3. Formato de los datos crudos

La señal recibida forma una matriz de números complejos de dos dimensiones, donde cada número complejo contiene información de la amplitud y fase de la señal. La primera dimensión corresponde a la dirección range. Una línea de range consiste en los valores complejos de la señal luego de ser amplificada, modulada a banda base, digitalizada y almacenada en memoria. El radar obtiene una de estas líneas cada vez que viaja una distancia $v \cdot PRI$ para así formar la segunda dimensión de la matriz conocida como azimuth o slow-time.

Se suelen utilizar los términos fast-time cuando se habla de la dirección de rango ya que en ese sentido la adquisición es a la velocidad de las ondas electromagnéticas. En contrapartida, a la dirección de azimuth se la suele llamar slow-time, debido a que se sensa solo a la velocidad en que viaja la plataforma.

2.5.4. Creación de imágenes

La visualización de los datos crudos de SAR no aporta ninguna información entendible por un humano. Para llegar a ese punto hay que aplicar técnicas de procesamiento de señales.

De un modo simplificado, el procesamiento de los datos [8] puede ser entendido como dos operaciones, una para cada dirección. En la siguiente figura se muestran los pasos básicos.

1. El primer paso tiene como objetivo comprimir la señal chirp transmitida en un pulso corto. Se realiza una multiplicación en frecuencia en lugar de una convolución en el tiempo debido a que el costo computacional es mucho menor. Así, cada línea del range es multiplicada en el dominio de la frecuencia por el conjugado complejo del espectro del chirp transmitido. El resultado es una imagen de range comprimida que muestra información sobre la distancia relativa entre el radar y cualquier punto del terreno.

2. La compresión azimuth sigue el mismo razonamiento, es decir, la señal se multiplica en la frecuencia por el conjugado complejo de la respuesta esperada de un objetivo puntual del suelo.

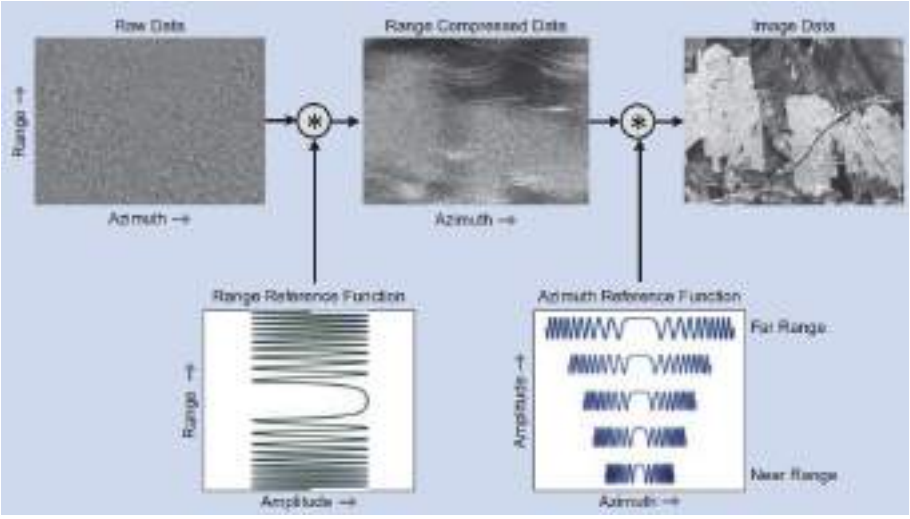


Fig. 2.7: Procesamiento de los datos SAR

Las imágenes SAR son comúnmente mostradas como los valores de intensidad para cada píxel, que corresponden con el nivel de reflectividad de ese punto en tierra. Para lograr esto se necesitan dos pasos adicionales: calibración y geocodificación. La **calibración** sirve para asegurar que los valores de intensidad de cada píxel se pueden relacionar directamente con la retrodispersión del radar de la escena. Por otro lado, la **geocodificación** sirve para determinar qué punto del planeta tiene asociado cada píxel de la imagen.

2.5.5. Particularidades de las imágenes SAR

Las imágenes SAR están geométricamente distorsionadas debido a que el radar solo mide la proyección de una escena tridimensional en las coordenadas de slant-range y

azimuth. Esto causa ciertos efectos como las sombras, el foreshortening, el layover y corner reflector.

Sombras o shadowing: Es causado por objetos que bloquean el paso directo de los pulsos electromagnéticos. En las imágenes SAR las sombras aparecen como zonas completamente negras ya que no hay ninguna información que provenga desde ahí.

Foreshortening: Debido a que el radar está inclinado (side-looking), la dispersión que retorna a la antena está determinada por cuán lejos está el target de la plataforma en el plano de slant. Esto causa ciertas distorsiones geométricas en las imágenes. Si el terreno no es completamente plano, sino que hay cierta topografía, los picos de los targets no aparecerán en la posición geográfica correcta, sino que se verán más próximos al nadir. Esto se debe a que la señal que rebota en el pico, al estar más cerca del radar, llega antes que la señal proveniente de la parte baja del target ya que, si se asume que el terreno es plano, entonces esa señal debería provenir de un punto más cercano al nadir.

Layover: Es un caso extremo de foreshortening donde el pico está tan cerca del satélite que el punto geográfico donde se muestra está por delante de la parte más baja del target, haciendo que el objeto se vea invertido.

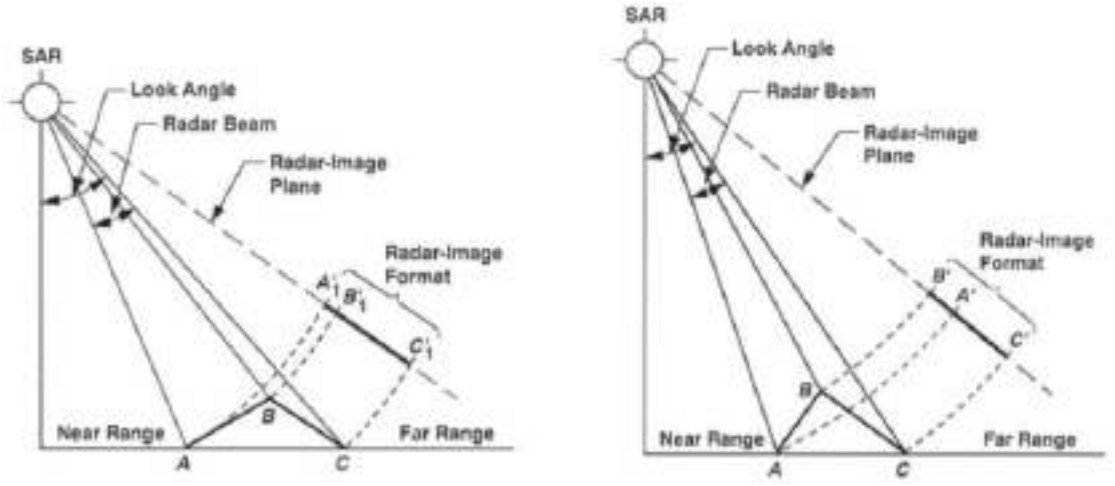


Fig. 2.8: Efectos de Foreshortening y Layover

Corner Reflector: Este efecto ocurre cuando la energía incide contra dos cuerpos de metal colocados a 90° entre sí. De esta manera prácticamente toda la energía incidente será reflejada y devuelta en la dirección que provino. Este fenómeno puede detectarse con facilidad en una imagen SAR al observar puntos muy brillantes. Es común su ocurrencia en barcos o edificios.

Otro efecto que se produce en las imágenes SAR es el **speckle**, que es causado por la presencia de muchos elementos de dispersión dentro de una misma celda de resolución con una distribución aleatoria. La consecuencia es que la intensidad y fase de una imagen dejan de ser determinísticas. Si bien se suele referirse al speckle como ruido, este fenómeno no se puede reducir incrementando la potencia de la señal transmitida ya que tiene carácter multiplicativo. Para mitigarlo se utiliza una técnica llamada multi-look que consiste en promediar la intensidad de capturas independientes de la misma escena, reduciendo el ruido, pero también la resolución espacial de la imagen. Algo interesante es que el speckle tiene menor importancia cuanto mayor sea la resolución del sistema, ya que las celdas de resolución son más pequeñas.

2.5.6. Resolución de azimuth y swath

Mejorar la resolución de azimuth significa un mayor ancho de banda Doppler del eco recibido y consecuentemente un mayor muestreo, que en este caso incrementa el PRF. Esto reduce el tamaño de la ventana de eco, es decir el tiempo disponible para recibirlos, produciendo que se reduzca el ancho del swath. Por lo tanto, la resolución de azimuth y el ancho de swath se contradicen y no se puede mejorar ambas al mismo tiempo.

2.5.7. Modos de operación

Los sistemas SAR actuales son capaces de operar en diferentes modos al controlar los patrones de radiación de la antena. En una antena plana esto se logra dividiéndola en sub-aperturas y controlando la fase y amplitud de cada una. En otras palabras, la misma antena del sistema SAR es un arreglo de antenas.

Fig. 2.9: Elementos radiantes del SAOCOM

El modo más común es el **Strip-map**. En este modo el patrón está ligado a un solo swath barrido de forma continua. Si se requiere un swath más ancho, se puede operar en el modo **ScanSAR** donde se utilizan múltiples sub-swaths los cuales son iluminados por muchos pulsos, pero por un periodo de tiempo menor. La desventaja de ScanSAR es que la resolución de azimuth se ve degradada. Si lo que se requiere es una mayor resolución de azimuth se puede utilizar el modo **Spotlight** en el cual los pulsos son dirigidos a un punto fijo para iluminar solo esa región aumentando la longitud de la apertura sintética. La desventaja es que, al no tener un barrido de swath continuo, se logra sólo adquirir una región pequeña.

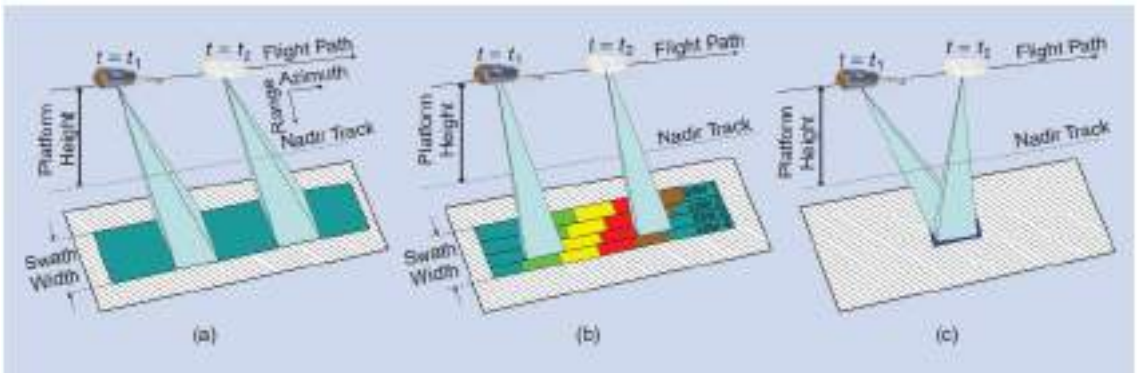


Fig. 2.10: Modos de operación: (a) Stripmap; (b) ScanSAR; (c) Spotlight

2.5.8. Polarización y scattering

La polarización hace referencia a la orientación del plano del campo eléctrico. Esta orientación puede tener cualquier ángulo, pero normalmente los sistemas SAR utilizan polarización lineal. La polarización horizontal es indicada con la letra H mientras que la vertical con la letra V. Los sistemas pueden transmitir y recibir en H, en V, o en ambas. De esta forma, las señales transmitidas en vertical y recibidas en horizontal se denotan VH, y del mismo modo obtenemos las combinaciones HV, VV y HH. Examinar la señal en sus diferentes polarizaciones nos da mucha información sobre la estructura de los objetos del terreno. Diferentes tipos de dispersión son más sensibles a un tipo de polarización que a otra.

Las superficies rugosas como por ejemplo el suelo desnudo o las carreteras son más sensibles a la dispersión VV, es decir, cuando se transmiten ondas electromagnéticas con polarización vertical las señales obtenidas de la dispersión de estos targets serán mayormente verticales. Por otro lado, la dispersión de volumen es más sensible a las polarizaciones cruzadas, es decir, VH y HV. Un ejemplo de este tipo de dispersión puede ser un bosque repleto de árboles. Por último, la dispersión por doble rebote, provocada por edificios o troncos de árboles, es más sensible a la polarización HH.

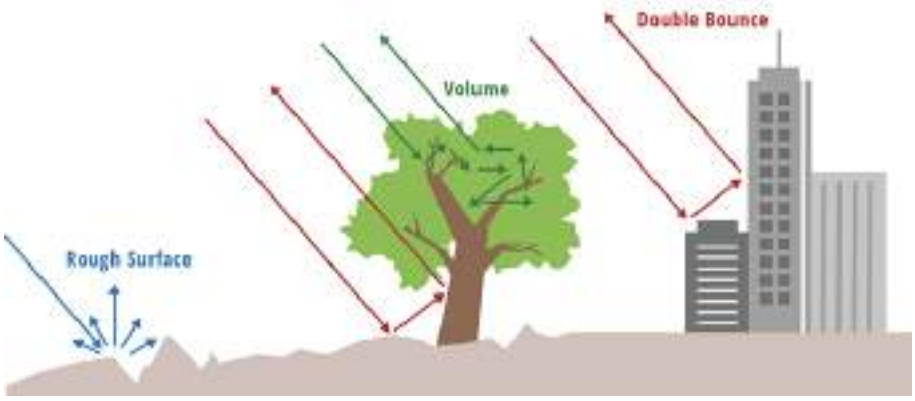


Fig. 2.11: Efecto de scattering sobre diferentes superficies

La longitud de onda es un factor muy importante a la hora de estudiar la dispersión de un objeto. La penetración de la señal varía en función de la longitud de onda y del target iluminado. Un ejemplo muy común son los bosques. Si se utiliza una señal de banda C solo penetrará en la capa superior de los árboles. Esto es debido a que la señal de banda C posee una longitud de onda de entre 3,5 cm y 7,5 cm, tamaño muy similar a las hojas de los árboles. Si por el contrario se hace uso de la banda L, con longitud de onda de entre 15 y 30 cm, la penetración será mucho mayor, pudiendo alcanzar e incluso penetrar el suelo.

2.6. Satélites existentes y futuros

El primer satélite con tecnología SAR fue el SeaSat [9], lanzado por la NASA en 1978. Pionero por ser el primero en su tipo con aplicaciones civiles, recolectó más información sobre los océanos en sus 110 días de operación que en los 100 años previos mediante el uso de embarcaciones. Siendo una misión experimental, demostró la viabilidad del uso de la tecnología radar en el estudio y observación del planeta.

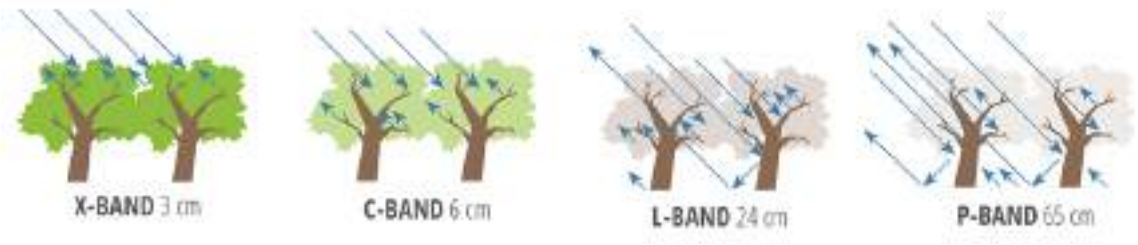


Fig. 2.12: Penetración de la onda en diferentes bandas

Entre las instituciones que operan satélites con instrumentos SAR, la Agencia Espacial Canadiense (CSA) y la Agencia Espacial Europea (ESA) [10] se destacan por su gran trayectoria en el área.

La CSA operó entre 1996 y 2013 el RADARSAT-1 [11], ampliamente utilizado tanto en el sector científico como en el comercial. Debido al éxito del primer satélite, la agencia canadiense puso en órbita el RADARSAT-2 en 2007 con grandes mejoras con respecto a su predecesor. La misión más actual posee el nombre de RADARSAT constellation, constituida por tres satélites de menor tamaño que permitirán una mejora en la resolución temporal de los datos.

Por su parte, la ESA llevó a cabo las misiones ERS-1 y ERS-2 en la década de 1990. Actualmente, la agencia posee una constelación de dos satélites SAR llamada Sentinel-1. Esta misión forma parte del programa Copernicus, actualmente el más ambicioso del mundo en cuanto a observación de la tierra. Además, tanto el Centro Espacial Alemán (DLR) como la Agencia Espacial Italiana (ASI) cuentan con sus propios programas. Los satélites TerraSAR-X y TanDEM-X son operados por Alemania mientras que Italia cuenta con una constelación llamada Cosmo-SkyMed. Esta última está formada por cuatro satélites y, si bien su principal uso es militar, también tiene uso civil.

En la actualidad existen varias misiones planificadas para poner en órbita nuevos satélites de este tipo debido a las ventajas que posee el uso de SAR en el sensado remoto y a los grandes avances en la tecnología. En 2023 está programado el lanzamiento de la misión NISAR [12] llevada adelante en conjunto por la NASA y la Agencia India de Investigación Espacial (ISRO, por sus siglas en inglés). Entre sus aplicaciones más destacadas está el estudio de la dinámica del suelo y el estudio de los océanos y cuerpos de agua.

En el marco del Plan Nacional Espacial [13], la Comisión Nacional de Actividades Espaciales (CONAE) [14] de Argentina lleva a cabo la misión SAOCOM en colaboración con la ASI. Ambas agencias espaciales integran el SIASGE, Sistema Ítalo Argentino de Satélites para la Gestión de Emergencias. El objetivo principal es la medición de la humedad del suelo y aplicaciones en emergencias, tales como detección de derrames de hidrocarburos en el mar y seguimiento de la cobertura de agua durante inundaciones. La misión consiste en dos constelaciones, SAOCOM 1 y SAOCOM 2, cada una integrada por dos satélites. En la actualidad están en órbita ambos satélites de la primera constelación.

2.7. Aplicaciones

El sensado remoto tiene una gran variedad de aplicaciones [15] que abarcan campos muy diversos. A continuación, se nombran algunas de las principales aplicaciones de las imágenes SAR.

Monitoreo de agricultura: La agricultura, que es uno de los principales componentes de la economía de muchos países, puede ser monitoreado en forma satelital y obtener información muy valiosa como las condiciones del terreno, el estado del crecimiento de los cultivos, estimaciones del rendimiento, el momento de cosecha o las condiciones de helada, entre otros.

Monitoreo de inundaciones: Las inundaciones afectan aproximadamente al 75 % de todas las personas afectadas por desastres naturales. Al año se producen alrededor de 150 inundaciones graves en todo el mundo. Al igual que las sequías, las inundaciones son eventos que, en cierta medida, siguen un ciclo natural predecible. El sensado remoto puede ser utilizado para proveer información y ser un indicador de las regiones con altas probabilidades de sufrir este fenómeno.

Monitoreo de bosques: Los bosques son uno de los recursos más valiosos que tiene la Tierra, además de funcionar como reservorio de dióxido de carbono, más de la mitad de la biodiversidad de la Tierra vive en ellos y nos proveen de madera, combustible, refugio, entre otros. A partir de las imágenes satelitales se puede controlar el crecimiento y la deforestación, monitorear la diversidad y encontrar focos de incendios.

Uso de la tierra: Otra aplicación importante del sensado remoto es el control del uso de la tierra, el sensado remoto permite obtener información global en forma rápida de toda la superficie de la Tierra y poder monitorear el manejo de los recursos naturales, la expansión de los centros urbanos, control de la vida silvestre, etc.

Monitoreo de océanos: Los océanos cubren aproximadamente el 70 % de la superficie de la Tierra y contienen el 97 % del agua disponible. Son una importante fuente de alimentos para las personas, regulan la temperatura de la Tierra y son el hábitat de una gran biodiversidad. Algunas de las aplicaciones del sensado remoto en los océanos son el estudio de las características de las olas y el viento en la superficie, la detección de derrames de petróleo, la clasificación de la línea de costa, la detección de islas de basura y la detección de barcos.

Utilizando información de distintos tipos de satélites de radar SAR nuestro trabajo se encuentra dentro de la categoría monitoreo de océanos, donde se desarrollarán modelos de inteligencia artificial particularmente con el objetivo de detectar barcos.

Capítulo 3

Deep Learning

3.1. Introducción

Desde los inicios de las computadoras, las personas se preguntaban si tales máquinas en algún momento se convertirían en inteligentes. Para construir tal sistema se requiere el entendimiento de la funcionalidad de nuestro sistema cognitivo. Hoy en día, la inteligencia artificial (AI) es una de las ramas de las ciencias computacionales con mayor impacto en la vida de las personas y con gran potencial para el futuro. La inteligencia artificial es la simulación de los procesos de la inteligencia humana mediante sistemas computacionales. Entre estos procesos se encuentran el aprendizaje, el razonamiento y la autocorrección.

3.2. Historia

Inicios del estudio de la mente humana

El asociacionismo es una teoría que establece que la mente es un conjunto de elementos conceptuales organizados mediante asociaciones entre ellos. Inspirado en Platón, Aristóteles estudió el proceso del recuerdo y estableció las cuatro leyes del asociacionismo [16].

- **Contigüidad:** Los eventos con proximidad espacial o temporal tienden a estar asociados.
- **Frecuencia:** La cantidad de ocurrencias de dos eventos es proporcional a la fuerza de la asociación entre estos dos eventos.
- **Similaridad:** Un pensamiento sobre un evento tiende a desencadenar el pensamiento de un evento similar.
- **Contraste:** Un pensamiento sobre un evento tiende a desencadenar el pensamiento de un evento opuesto.

Aristóteles describió la implementación de estas cuatro leyes como sentido común. Por ejemplo, el tacto, el olor o el sabor de una manzana, mediante el sentido común, naturalmente nos lleva al concepto de una manzana. Hoy en día, muchas de estas leyes sirven como conceptos fundamentales para el aprendizaje automático.

Inicios de la computación

Entre los siglos XIX y XX se desarrollaron los pilares de la computación moderna. En 1836, el matemático Charles Babbage y la matemática Augusta Ada Byron, ambos de la Universidad de Cambridge, inventaron el primer diseño de una máquina programable. En la década de 1940, el matemático John Von Neumann de la Universidad de Princeton propuso una arquitectura de computadora con almacenamiento de los programas, es decir que los programas y los datos a ser procesados se mantienen guardados en una memoria.

Hacia el término inteligencia artificial

El término inteligencia artificial es ampliamente utilizado desde la conferencia de verano de Dartmouth College en 1956. Patrocinada por la Agencia de Proyectos de Investigación Avanzados de Defensa (DARPA), la propuesta de la conferencia era trabajar sobre la conjetura de que cada aspecto del aprendizaje y cada característica de la inteligencia podían ser tan precisamente descritos que se podían crear máquinas que las simularan.

En la conferencia de Dartmouth College, los expertos en la materia predijeron que una inteligencia artificial equivalente a la humana estaba muy cerca de ser lograda. Estas declaraciones atrajeron el apoyo de gobiernos y empresas. En esos momentos, grandes avances sobre las bases del campo de la AI fueron desarrollados. En 1950, Newell y Simon publicaron el algoritmo General Problem Solver (GPS) que estaba lejos de solucionar problemas complejos pero que sentó los fundamentos para el desarrollo de arquitecturas más complejas. En esos tiempos, McCarthy desarrolló Lisp, un lenguaje de programación orientado a inteligencia artificial que aún es usado hoy en día. A mediados de la década de 1960, el profesor Joseph Weizenbaum del MIT desarrolló ELIZA, un procesador de lenguaje natural básico que sentó las bases para los chatbots de hoy.

Invierno de la AI

Los avances en el campo de la inteligencia artificial estuvieron muy lejos de las grandes expectativas expuestas en Dartmouth College debido mayormente a la gran complejidad del problema y a las limitaciones en memoria y procesamiento de las computadoras de la época. Por estas razones comenzó un período conocido como el invierno de la AI caracterizado por la falta de apoyo por gobiernos y empresas, y por el desfinanciamiento de los proyectos de investigación.

Resurgimiento

El incremento en la potencia computacional junto con una explosión en la cantidad de datos disponibles condujo al renacimiento de la inteligencia artificial a finales de la década del 90. Grandes avances en campos muy variados como es el procesamiento del lenguaje natural (NLP), la visión computacional y la robótica, permitieron desarrollar aplicaciones útiles de esta tecnología. Hoy en día, está presente en un sin fin de productos y servicios tanto de consumo masivo como en el sector empresarial y gobiernos. Salud, educación, finanzas, derecho y la industria manufacturera son ejemplos de áreas en que la AI es ampliamente utilizada.

Por mucho tiempo se intentó conseguir una inteligencia artificial de propósito general o *General AI* que concentre todas las características de la inteligencia humana en un mismo sistema. Lograr esto es una tarea extremadamente compleja. En la actualidad,

todos los sistemas de AI son llamados *Narrow AI* o inteligencia artificial específica. En este tipo de sistemas solo se exhibe una característica en particular de la inteligencia humana especializándose solo en una tarea. Esto dio lugar a la creación de sistemas expertos que asisten a profesionales u operarios en su labor diaria, siendo el objetivo ayudar a las personas en vez de reemplazarlas completamente.

3.3. Machine Learning

El campo de machine learning (ML) o aprendizaje automático es una rama de la inteligencia artificial que se centra en la pregunta de cómo construir un programa informático que pueda mejorar con la experiencia. Su fundamento es la idea de que los sistemas son capaces de aprender de los datos y a través de la experiencia ganada en el entrenamiento mejorar su rendimiento. Mitchell (1997) lo define como "Un programa de computadora se dice que aprende de una experiencia E con respecto de alguna tarea T midiendo la performance P, si la performance P en la tarea T mejora con la experiencia E" [17].

El funcionamiento del cerebro humano, su capacidad para reconocer patrones y aprender en base a la experiencia fue una de las principales inspiraciones de la inteligencia artificial. Podemos considerar al cerebro como un gran sistema complejo, no lineal y que trabaja procesando información en paralelo que está compuesto por millones de componentes estructurales, conocidos como neuronas. Inspirados en este funcionamiento y su componente elemental, surge el concepto de **artificial neural network (ANN)** o redes neuronales artificiales.

3.3.1. Red Neuronal Artificial

Una red neuronal artificial, en su forma más general, es un sistema diseñado para imitar la forma en que el cerebro realiza una tarea y asemejar su proceso de aprendizaje. La red suele implementarse utilizando componentes electrónicos o se simula por software.

Las redes neuronales artificiales son ampliamente utilizadas en una gran variedad de áreas de la industria, entre ellas se encuentra, por ejemplo, el procesamiento de imágenes y reconocimiento de patrones que permite reconocer caracteres, animales o caras, entre otras cosas, en una imagen, o la detección automática de enfermedades como Parkinson o Alzheimer. Esta tecnología también es aplicada en muchos aspectos de nuestra vida cotidiana, desde cuando hacemos una búsqueda en un buscador web, las recomendaciones en tiendas online o las publicidades que nos muestran hasta en la cámara de nuestros celulares cuando sacamos una foto.

La implementación más sencilla de una ANN es lo que se conoce como **perceptrón simple** [18] que consiste del modelo matemático de una simple neurona artificial. En la figura 3.1 podemos observar los 3 elementos básicos del modelo:

1. Una serie de pesos sinápticos w_{k1}, w_{k2}, $\ldots . w_{km}$ que afectan a las entradas x_1, x_2, $\ldots x_m$ del modelo y el bias o umbral de entrada b_k.

2. Una función de suma donde todas las entradas son multiplicadas con su respectivo peso y sumadas junto al bias obteniendo el valor v_k.

3. Una función de activación ϕ que determina el valor de salida y_k de la neurona.

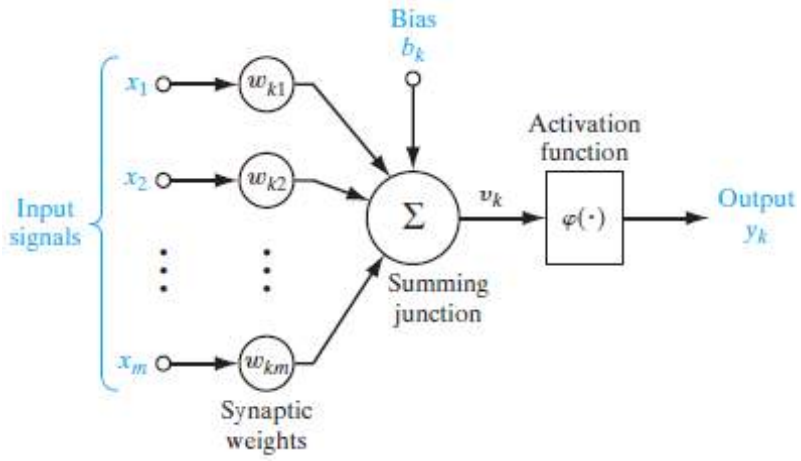


Fig. 3.1: Modelo no lineal de una neurona

Debido a su sencillez, el perceptrón simple tiene la limitación de que solo puede resolver problemas de clasificación de patrones que sean linealmente separables. Por este motivo, para resolver problemas más complejos que no sean linealmente separables, se utiliza una ANN denominada **perceptrón multicapa (MLP)** [18]. El MLP está formado por múltiples perceptrones simples interconectados de manera ordenada y separados en capas. Todos los MLP cuentan con al menos una capa oculta entre los nodos de entrada y salida de la red, que se encarga de la detección de características de los datos. Esto le permite, en comparación con el perceptrón simple, aprender de datos más complejos, pudiendo reconocer y clasificar patrones de formas muy variadas.

El entrenamiento de una ANN es la etapa más importante de su desarrollo ya que en esta etapa los parámetros libres del sistema, es decir, los pesos sinápticos wki y los biases bk, se adaptan en función de los datos de entrada de la red. En función del algoritmo de aprendizaje implementado la red obtiene conocimiento de los datos de entrada y lo almacena en los parámetros.

3.3.2. Clasificación de los algoritmos

Los algoritmos de aprendizaje se pueden clasificar en tres grandes categorías, aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

- El **aprendizaje supervisado** tiene como principal característica que los datos utilizados para el entrenamiento están compuestos por un par de valores entrada-salida. Cada dato de entrada tiene asociado una etiqueta o target de salida proporcionado por un supervisor o experto en la temática que le indica al algoritmo la salida esperada. Los parámetros de la red se ajustan bajo la influencia combinada del vector de entrenamiento y la señal de error. La señal de error se define como la diferencia entre la respuesta deseada o target y la respuesta real de la red. Este ajuste se lleva a cabo de forma iterativa, paso a paso, con el objetivo de que la red aprenda de los datos. El conocimiento del entorno del que dispone el supervisor se transfiere a la red neuronal a través del entrenamiento y se almacena en pesos sinápticos fijos. Esta categoría de entrenamiento es la que se usa generalmente en los modelos de clasificación como el que implementaremos en este trabajo. En la figura 3.2 se muestra un diagrama de

bloques que ilustra esta forma de aprendizaje.

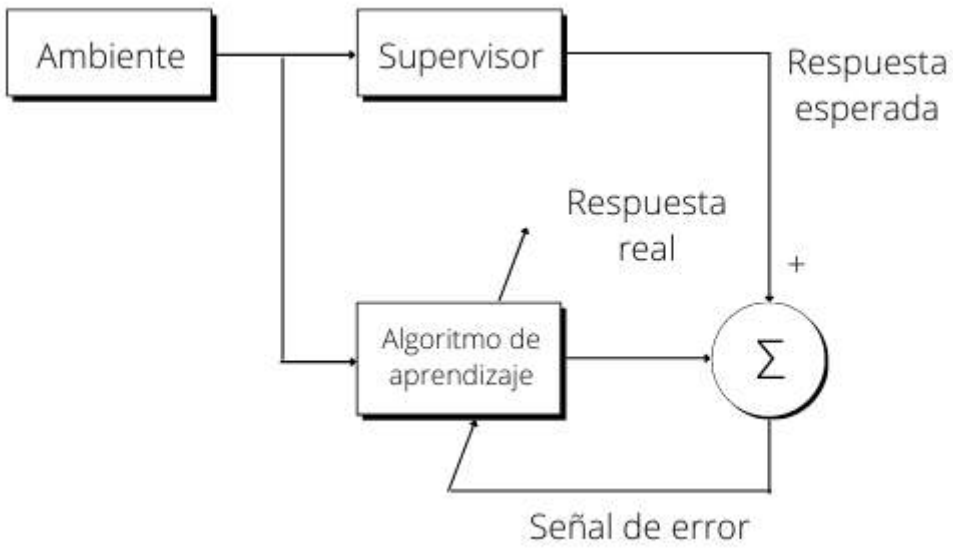


Fig. 3.2: Diagrama de bloques del aprendizaje supervisado

- El **aprendizaje no supervisado** se caracteriza porque los datos utilizados para el entrenamiento de la red no tienen un valor de salida asociado, es el sistema quien aprende características de la red y la utiliza como experiencia para las etapas posteriores. El diagrama de bloques que representa esta forma de aprendizaje se observa en la figura 3.3. En este tipo de aprendizaje no existe un supervisor o "crítico" que supervise el proceso de aprendizaje, es por esto que se define una medida de la calidad de la tarea que realiza la red y los parámetros libres son ajustados en base a esta medida.

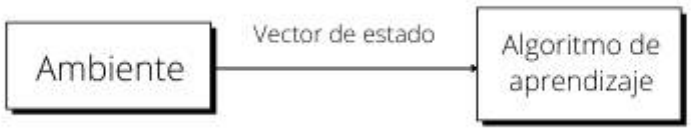


Fig. 3.3: Diagrama de bloques del aprendizaje no supervisado

- En la categoría de **Aprendizaje por refuerzo o *Reinforcement learning*** el aprendizaje de un mapeo de entrada-salida se realiza a través de la interacción continua con el entorno. El sistema está diseñado para aprender bajo refuerzo retardado, lo que significa que el sistema observa una secuencia temporal de estímulos también recibidos del entorno, que finalmente dan lugar a la generación de la señal heurística de refuerzo. La Figura 3.4 muestra el diagrama de bloques donde un crítico o critic convierte una señal de refuerzo primaria recibida del entorno en una señal de refuerzo de mayor calidad denominada señal de refuerzo heurística.

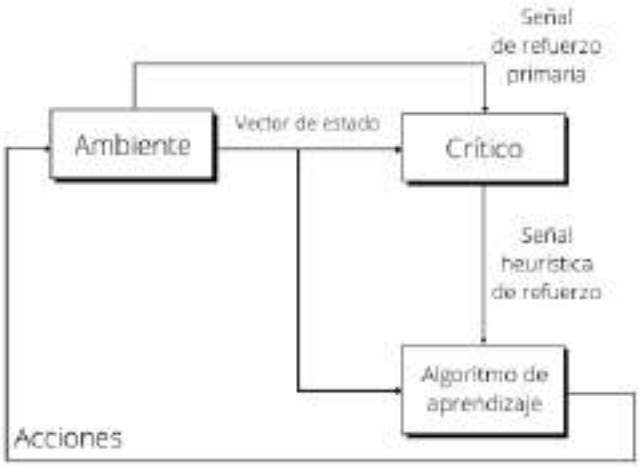


Fig. 3.4: Diagrama de bloques del aprendizaje por refuerzo

3.4. Deep Learning

Luego de ver las potencialidades del *Machine Learning* comenzó un auge por conseguir redes neuronales cada vez más potentes. Al incrementar la cantidad de capas de los MLP la complejidad de los posibles problemas a resolver también aumenta. Esto es debido a que se extiende la capacidad de extraer características relevantes. Las capas más tempranas extraerán características básicas de los datos y a medida que se avanza entre las capas las características y patrones se vuelven más complejos, siendo las capas finales las encargadas de detectar patrones de más alto nivel. En otras palabras, se logra obtener representaciones de datos con múltiples niveles de abstracción.

No hay dudas de que el hecho de construir redes neuronales cada vez más profundas impulsó las capacidades del *Machine Learning* de forma exponencial dando lugar al aprendizaje profundo o *Deep Learning (DL)*.

Si bien al aumentar la cantidad de capas también aumenta la versatilidad de la red, hacerlo trae varias dificultades a enfrentar en el momento de entrenar este tipo de redes, como por ejemplo, el problema del desvanecimiento del gradiente.

El DL ha mejorado el estado del arte de muchos campos de la inteligencia artificial como la detección de objetos, el reconocimiento de patrones o la traducción automática. Además, abrió las puertas a la solución de problemas más complejos, y como resultado se extendió el dominio del DL a campos más modernos como la visión computacional, el reconocimiento de rostros o el modelado del lenguaje natural.

3.5. Redes Neuronales Convolucionales

Una Red Neuronal Convolucional o CNN [19] es un tipo particular de Deep Learning que es más fácil de entrenar y generaliza mejor que las redes totalmente conectadas o *Fully Connected*. El nombre CNN indica que la red utiliza una operación matemática llamada convolución en alguna de sus capas, en lugar de una multiplicación matricial.

La convolución es una operación de dos funciones con argumento real y se representa

con un asterisco:

$$S(t) = (x * w)(t) \tag{3.1}$$

El primer argumento (x) es comúnmente la entrada y el segundo (w) el núcleo o *kernel*, la salida se denomina mapa de características (feature map). En aplicaciones de DL, la entrada es usualmente un arreglo multidimensional de datos y el kernel un arreglo multidimensional de parámetros, estos arreglos se denominan Tensores.

La operación de convolución se aprovecha de tres ideas que ayudan a mejorar el rendimiento del sistema de DL: (1) Interacciones escasas (sparse interactions); (2) Parámetros compartidos (parameter sharing); y (3) Representaciones equivalentes (equivariant representations) [1].

- Las interacciones escasas se logran haciendo el kernel mucho menor que la entrada. Por ejemplo, al procesar una imagen, la entrada tiene cientos o millones de píxeles, pero se pueden detectar bordes o características importantes con kernels de decenas o cientos de píxeles. De esta manera se reduce la cantidad de parámetros a almacenar y también se reducen la cantidad de operaciones a realizar.

- Los parámetros compartidos se refieren al uso del mismo parámetro para más de una función en el modelo. En las CNN, cada miembro del kernel es usado en cada posición de la entrada, en lugar de aprender un conjunto de parámetros para cada posición, se aprende solo uno. Esto reduce los requerimientos de almacenamiento del sistema.

- La forma de compartir los parámetros hace que la capa tenga una propiedad llamada equivarianza a la traslación, es decir, si la entrada cambia, la salida cambia de la misma manera.

Una red convolucional es un perceptrón multicapa diseñado específicamente para reconocer objetos bidimensionales con un alto grado de invariabilidad a la distorsión [18]. Las CNN utilizan el método de aprendizaje supervisado y la estructura de la red está compuesta por [18]:

- Extracción de características: cada neurona toma su entrada sináptica de un campo receptivo local de la capa anterior, forzándola a extraer rasgos locales. Una vez que la característica fue extraída, su posición absoluta no es relevante, sino, su posición relativa a otros rasgos.

- Mapeo de características: cada capa de la red está compuesta por múltiples mapas de características. A estos mapas se le realiza una convolución con un kernel de tamaño pequeño y posteriormente se pasa a través de una función de activación.

- Submuestreo: cada capa convolucional es seguida por una capa de pooling que realiza el promedio local y el submuestreo, con lo que se reduce la resolución del mapa de características. Esta operación tiene el efecto de reducir la sensibilidad de la salida del mapa de características a los desplazamientos y otras formas de distorsión.

Todos los pesos en todas las capas de una CNN son aprendidos en el entrenamiento de la red. Durante el entrenamiento, el método de retropropagación del error (backpropagation error) es el utilizado para ajustar los valores de los parámetros.

Debido al uso alternado de capas de convolución y de submuestreo se obtiene un efecto "bipiramidal", es decir, en cada capa, de convolución o muestreo, el número de mapas de características aumenta mientras que la resolución espacial se reduce comparado con la capa anterior.

Una red neuronal convolucional típica se puede dividir en cuatro capas principales, que son comunes a todas las arquitecturas conocidas:

- Capa de convolución: en esta capa se realiza la operación de convolución entre la matriz de entrada y el kernel, obteniendo el mapa de características.
- Capa no lineal: se utiliza una función de activación no-lineal, como por ejemplo la función Sigmoide o ReLU.
- Capa de pooling: la capa de pooling se utiliza para unir características similares en solo una y de esta forma reducir la dimensionalidad del mapa de características. A su vez, ayuda a volver a la representación invariante a las pequeñas traslaciones y otras formas de distorsión en la entrada. Existen dos modos de operación para el pooling, promedio o máximo.
- Capa totalmente conectada: la salida de la última capa convolucional o capa de pooling es convertida en un vector que será la entrada de una serie de capas totalmente conectadas que clasifica las salidas en las clases.

3.6. ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

ImageNet Large Scale Visual Recognition Challenge [20] fue un desafío llevado a cabo entre 2010 y 2017 de clasificación y detección de objetos. El desafío consistía en una competencia anual en el que múltiples y muy importantes instituciones presentaban la arquitectura de la red neuronal que habían diseñado. Cada año un nuevo dataset de entrenamiento y testeo era presentado a la comunidad y debía ser utilizado para el concurso. Estos datasets estaban conformados por un subconjunto de datos de las más de 14 millones de imágenes etiquetadas que componen ImageNet, distribuidos entre 1000 categorías.

Al momento de probar los resultados de una red se obtiene la predicción como la probabilidad de que en la imagen exista cada una de las 1000 categorías posibles y aunque el objetivo principal es conseguir el menor porcentaje de error en la clasificación de las imágenes, en el desafío se evalúan dos tipos de errores.

El error llamado top-1 indica si la probabilidad más alta corresponde con la salida esperada de la red, por otro lado, se evalúa si la salida esperada está dentro de las cinco probabilidades más altas, conocido como error top-5. En ambos casos el puntaje se calcula como la cantidad de aciertos sobre la cantidad total de evaluaciones.

Las arquitecturas que fueron presentadas en el concurso y que fueron entrenadas con ImageNet son ampliamente utilizadas dentro de la comunidad en un sinfín de áreas, algunas de las cuales serán presentadas a continuación.

3.7. Arquitecturas

3.7.1. LeNet

Esta arquitectura de CNN fue una de las primeras en desarrollarse y, debido a que demostró la viabilidad de la tecnología, sentó las bases del Deep Learning. La arquitectura

es sencilla con respecto a las más modernas. Cuenta con seis capas, dos convolucionales, dos capas de submuestreo y, por último, dos capas totalmente conectadas como se puede observar en la figura 3.5.

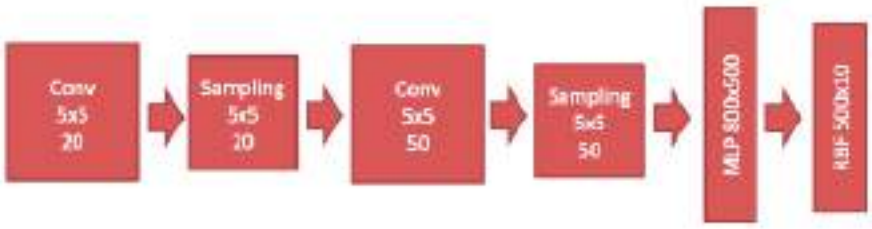


Fig. 3.5: Arquitectura LeNet

LeNet [21] es conocida por su habilidad de clasificar correctamente imágenes con dígitos con variaciones en la rotación, escala, e incluso dígitos de diferentes grosores. Además, con la introducción de esta arquitectura, LeCun et al. [22] también introdujo el dataset MNIST, que luego se convirtió en un benchmark estándar para el reconocimiento de dígitos.

3.7.2. AlexNet

Sin bien LeNet marcó el comienzo de la era de las redes neuronales convolucionales, con la aparición de AlexNet [23], inventada por Krizhevsky et al., se comenzó a utilizar las CNNs para problemas más complejos. AlexNet fue la primera red que tuvo una buena performance en el dataset ImageNet, históricamente conocido por la variedad de sus datos y la dificultad de obtener buenos resultados. Esto condujo a los grupos de investigación a una competencia en el desarrollo de arquitecturas de CNNs utilizando como benchmark ImageNet.

A continuación, se describe la arquitectura de la red y algunas de las características presentadas en el artículo original.

Arquitectura

La arquitectura de la red está compuesta por ocho capas, cinco de ellas convolucionales y tres fully connected o totalmente conectadas. La salida de la última capa totalmente conectada alimenta una capa softmax de 1000 salidas que produce una distribución entre las 1000 categorías de ILSVRC. La arquitectura se puede observar en la figura 3.6.

La primera capa convolucional filtra la imagen de entrada de $224 \times 224 \times 3$ con 96 filtros de tamaño $11 \times 11 \times 3$ con un stride de 4 píxeles. La segunda capa convolucional toma como entrada la salida de la primera capa convolucional, después de aplicar normalización y pooling, y la filtra con 256 filtros de tamaño $5 \times 5 \times 48$. La tercera, la cuarta y la quinta capa convolucional están conectadas entre sí sin ninguna capa de pooling o normalización intermedia. La tercera capa convolucional tiene 384 filtros de tamaño $3 \times 3 \times 256$ conectados a las salidas de la segunda capa convolucional. La cuarta capa convolucional tiene 384 filtros de tamaño $3 \times 3 \times 192$, y la quinta capa convolucional tiene 256 filtros de tamaño $3 \times 3 \times 192$. Las capas totalmente conectadas tienen 4096 neuronas cada una.

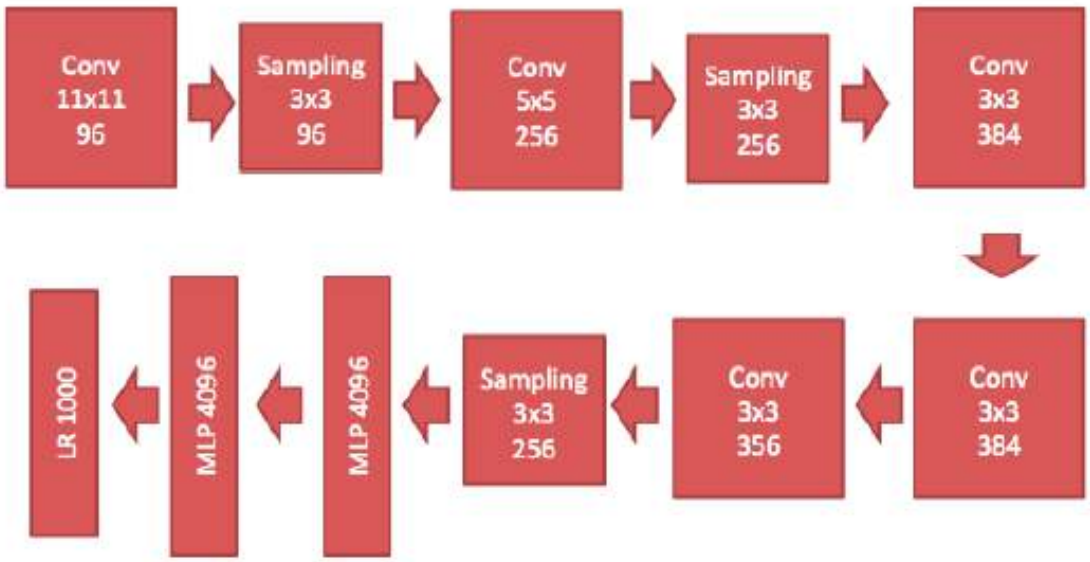


Fig. 3.6: Arquitectura AlexNet

ReLU

Una de las características más destacadas de AlexNet fue la utilización de una función de activación no lineal llamada ReLU (Rectified Linear Unit) definida como $f(x) = max(0, x)$, graficada en la figura 3.7, en lugar de las funciones utilizadas hasta el momento como la función tangente hiperbólica $f(x) = tanh(x)$ o la función sigmoidea $f(x) = (1 + e^{-x})^{-1}$.

Las funciones tangente hiperbólica y sigmoidea, al restringir el rango de valores posibles de la salida de una neurona, generan un efecto de saturación del gradiente y, debido a esto, al realizar la retropropagación del error para el ajuste de los pesos, no se producen cambios significativos en los parámetros generando un entrenamiento más lento. La función ReLU, al no estar limitada a un valor máximo de salida, evita el problema de saturación del gradiente y, por lo tanto, el estancamiento del entrenamiento.

Las redes neuronales convolucionales que utilizan la función de activación ReLU son considerablemente más rápidas en términos de velocidad de entrenamiento al compararlas con aquellas que utilizan *tanh* u otras funciones de activación similares.

Pooling superpuesto

Las capas de pooling realizan una operación de submuestreo en una grilla de $z \times z$ a partir de un punto central, estando cada una de estas grillas separadas por s pixeles. Si se establece s = z, se obtiene el pooling tradicional que se emplea comúnmente en las CNN, en cambio, si se estable $s < z$ obtenemos un pooling superpuesto. En AlexNet se utiliza este modelo de pooling superpuesto con s = 2 y z = 3. Este esquema reduce las tasas de error del top-1 y del top-5 en un 0,4 % y un 0,3 %, respectivamente, en comparación con el esquema no superpuesto s = 2, z = 2, que produce una salida de dimensiones equivalentes. En general, se observa que durante el entrenamiento los modelos con pooling superpuesto ayudan a disminuir el overfitting.

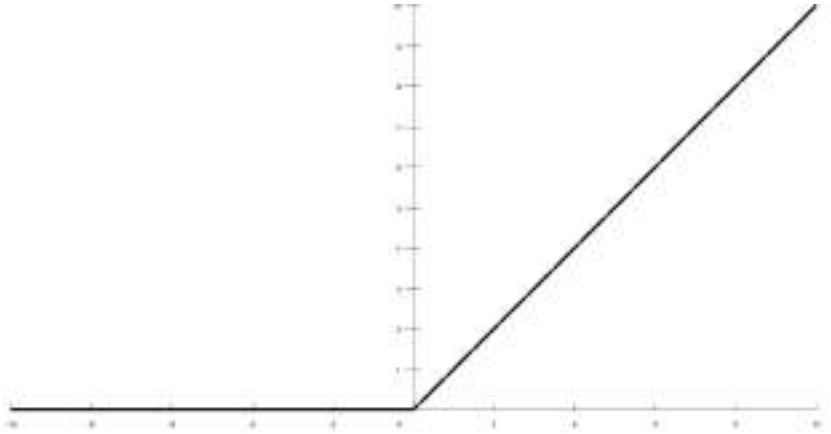


Fig. 3.7: Función de activación ReLU

Data Augmentation

El método más conocido y utilizado para reducir el overfitting es realizar transformaciones en el conjunto de datos. Los autores realizaron dos tipos de data augmentation durante el entrenamiento.

El primero consiste en generar traslaciones o reflexiones horizontales sobre una imagen. Para ello se realizan recortes aleatorios de 224×224 a partir de las imágenes de 256×256 que componen el conjunto de datos. Lo que se hizo fue tomar cinco recortes de 224×224, los cuatro fragmentos de las esquinas y el fragmento central, así como sus reflexiones horizontales. De esta manera se obtuvieron diez imágenes a partir de cada una.

La segunda forma reside en alterar las intensidades de los canales RGB, aumentando los componentes principales en un valor aleatorio. Este esquema reduce la tasa de error top-1 en más de un 1 % basándose en que la identidad de un objeto es invariable a los cambios de intensidad y color de la iluminación.

Dropout

La técnica de dropout consiste en poner en 0 el valor de salida de cada neurona oculta con una probabilidad de 0,5. De esta manera, esta neurona tampoco contribuye a la retropropagación del error. Esta técnica reduce las dependencias complejas de las neuronas, ya que una neurona no puede confiar en la presencia de otra neurona concreta.

En la arquitectura AlexNet se utiliza dropout en las dos primeras capas totalmente conectadas. Sin el dropout, la red muestra un sobreajuste sustancial. La utilización de dropout duplica aproximadamente el número de iteraciones necesarias para converger.

Entrenamiento

El modelo fue entrenado utilizando el descenso por gradiente estocástico con un tamaño de lote de 128, momentum de 0,9 y un decaimiento del peso de 0,0005.

Se inicializaron los pesos en cada capa a partir de una distribución gaussiana de media cero con desviación estándar de 0,01. Se inicializaron los biases de las neuronas en las capas convolucionales segunda, cuarta y quinta, así como en las capas ocultas totalmente

conectadas, con 1. Esta inicialización acelera las primeras etapas de aprendizaje al proporcionar a las ReLU entradas positivas. Además,los bias de las neuronas en las capas restantes se iniciaron con 0.

Para su implementación se utilizó la misma tasa de aprendizaje para todas las capas, que fue ajustada manualmente durante el entrenamiento. La regla que se siguió fue dividir la tasa de aprendizaje por 10 cuando el error de validación no mejora con respecto a la tasa actual. La tasa de aprendizaje fue inicializada en 0,01 y reducida tres veces.

Se entrenó la red durante aproximadamente 90 épocas a través de 1,2 millones de imágenes, lo que llevó entre cinco y seis días en dos GPU NVIDIA GTX 580 de 3 GB.

3.7.3. You Only Look Once (YOLO)

YOLO fue desarrollada por Redmon, Divvala, Girshick y Farhadi [24] de la Universidad de Washington, Allen Institute for AI y Facebook AI Research. El objetivo era reducir la detección de objetos en una imagen a un problema más simple que pudiera ser optimizado y entrenado más fácilmente, esto se logró enmarcando la detección como un problema de regresión. Con este sistema solo se mira una vez la imagen (you only look once) para predecir qué objetos están presentes.

En YOLO, una simple red convolucional predice simultáneamente múltiples bounding boxes y las probabilidades de clase para cada box. Este modelo tiene algunos beneficios frente a los métodos clásicos de detección de objetos.

En primer lugar, YOLO es extremadamente rápido, dado que la detección se redujo a un problema de regresión, permitiendo incluso realizar detecciones en tiempo real. En segundo lugar, YOLO razona globalmente sobre la imagen cuando hace predicciones. A diferencia de las técnicas basadas en ventanas deslizantes y en propuestas de regiones, YOLO ve toda la imagen durante el tiempo de entrenamiento y de test, por lo que codifica implícitamente la información contextual sobre las clases, es decir, utiliza características de toda la imagen para predecir cada bounding box y predice el bounding box para todas las clases de manera simultánea.

La red divide la imagen en una cuadrícula de $S \times S$, si el centro de un objeto cae dentro de una celda, esta celda es responsable de detectar ese objeto. Cada celda de la cuadrícula predice B bounding boxes y un valor de confianza de cada box. Estas puntuaciones reflejan el grado de confianza que tiene el modelo en que el box contiene un objeto y también la precisión que cree que tiene el box que predice. Si no existe un objeto en esa celda, el valor de confianza debería ser 0. Cada celda predice C clases con sus probabilidades. El resultado de las predicciones es un tensor de $S \times S \times (B * 5 + C)$.

Arquitectura

La arquitectura está inspirada en GoogLeNet [25]. La red está compuesta por 24 capas convolucionales seguidas por 2 capas totalmente conectadas. En lugar de los módulos Inception utilizados en GoogLeNet, se utilizan capas de reducción 1×1 seguidas por capas convolucionales de 3×3.

Las capas convolucionales alternadas 1×1 reducen el espacio de características de las capas anteriores. A su vez, las capas convolucionales están pre-entrenadas con ImageNet a la mitad de la resolución, es decir, con imagen de entrada de 224×224. Luego se duplica la resolución para la detección.

Entrenamiento

La red fue entrenada durante 135 épocas en los datasets de entrenamiento y validación de PASCAL VOC 2007 y 2012. Para el testeo también fueron utilizados los datasets de 2007 y 2012. A lo largo del entrenamiento se utilizó un tamaño de lote de 64, un momentum de 0,9 y un weight decay de 0,0005. La tasa de aprendizaje fue variando a lo largo del entrenamiento. Para las primeras épocas, se aumentó lentamente la tasa de aprendizaje de 10^{-3} a $10^{-}2$. Si se empieza con una tasa de aprendizaje alta, el modelo suele diverger debido a la inestabilidad de los gradientes. Se siguió entrenando con 10^{-2} durante 75 épocas, luego con 10^{-3} durante 30 épocas, y finalmente con 10^{-4} durante 30 épocas.

Para evitar el overfitting se utilizó dropout y data augmentation. Una capa de dropout con una tasa de 0,5 después de la primera capa conectada evita la coadaptación entre capas [26]. Para aumentar los datos, se realizaron escalamientos y traslaciones aleatorias de hasta el 20 % del tamaño de la imagen original. También se ajustó aleatoriamente la exposición y la saturación de la imagen hasta un factor de 1,5 en el espacio de color HSV.

Nuevas versiones

Con el paso de los años nuevas versiones mejoradas de YOLO han sido desarrolladas y presentadas a la comunidad por parte de sus creadores. Hoy en día existen al menos 5 versiones de YOLO. Las versiones de YOLO y su funcionamiento seran desarrollas con mas detalle en la sección 4.2.4.

3.7.4. VGG

A partir de los grandes resultados obtenidos por las redes convolucionales en el reconocimiento de imágenes y videos a gran escala y motivados por el ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) [20], Simonyan y Zisserman [27] decidieron mejorar la arquitectura AlexNet abordando la profundidad. Dejando fijos otros parámetros de la arquitectura y aumentado la profundidad de la red añadieron más capas convolucionales, lo cual es factible debido al uso de filtros de convolución muy pequeños de tamaño 3×3 en todas las capas.

Arquitectura

La entrada de la red es una imagen RGB con un tamaño fijo de 224×224 píxeles. El preprocesamiento que se realiza es calcular el valor medio RGB de cada pixel. La imagen pasa a través de una pila de capas convolucionales, con filtros muy pequeños de 3×3. La pila de capas convolucionales está seguida por tres capas Fully-Connected (FC), las primeras dos de 4096 neuronas cada una y la tercera realiza la clasificación ILSVRC y por lo tanto contiene 1000 neuronas, una para cada clase. La capa final es una softmax.

Los autores proponen cinco variantes diferentes a la configuración y profundidad de la red, como se ve en la tabla 3.1 fueron nombradas con las letras (A – E). Todas las variantes siguen el mismo diseño general, pero cambian en la profundidad, desde 11 capas con pesos en la red A hasta 19 en la red E. El número de canales parte de 64 en la primera capa y aumenta en un factor de 2 después de cada capa de max-pooling, hasta llegar a 512.

En las arquitecturas propuestas siempre se utilizan capas convolucionales de 3×3, aun cuando se coloque una pila de capas y sabiendo que una pila de dos capas de convolución de 3×3, sin pooling espacial entre ellas, tienen un campo receptivo efectivo de 5×5 y

análogamente con 3 capas un campo receptivo efectivo de 7×7. Esto se debe a que se incorporan tres capas de ReLU en lugar de una sola, lo que hace que la función de decisión sea más discriminativa y, en segundo lugar, se disminuye el número de parámetros de la red. Si se asume que cada capa de convolución de 3×3 tiene C canales, una pila de 3 capas está parametrizada por $3 \times (3^2 \times C^2) = 27C^2$ parámetros, al mismo tiempo, una única capa de 7×7 requería $7^2 \times C^2 = 49C^2$, es decir, un 81 % más de parámetros.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 x 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 conv1-256	conv3-256 conv3-256 conv3-256	conv3-256 conv3-256 conv3-256 conv3-256
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 conv1-512	conv3-512 conv3-512 conv3-512	conv3-512 conv3-512 conv3-512 conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Tab. 3.1: Arquitectura VGG

Entrenamiento

Simonyan y Zisserman siguieron el procedimiento de entrenamiento que realizó Krizhevsky et al. (2012) en AlexNet. El tamaño del lote se fijó en 256, el momentum en 0,9 y se aplicó dropout con un valor de 0,5. La tasa de aprendizaje fue fijada inicialmente en 10^{-2} y luego disminuida en un factor de 10 cuando la precisión de la validación dejaba de mejorar. En total la tasa de aprendizaje se redujo tres veces y el aprendizaje se detuvo después de

370.000 iteraciones, es decir, 74 épocas. Comparado con lo realizado por Krizhevsky, a pesar de contar con un mayor número de parámetros y ser más profunda, el entrenamiento requirió menos épocas para converger debido a la regularización implícita impuesta por una mayor profundidad sumado a menores tamaños de filtros de convolución y la pre-inicialización de algunas capas.

La inicialización de los pesos de la red es importante, a causa de que la mala inicialización puede estancar el entrenamiento debido a la inestabilidad del gradiente en las redes profundas. Para solucionar este problema, se comenzó entrenando la configuración A (ver Tabla 3.1) con pesos aleatorios. A partir de este entrenamiento, cuando se quiso entrenar las arquitecturas más profundas se inicializaron los pesos de las primeras cuatro capas y de las últimas tres capas totalmente conectadas con los valores de los pesos obtenidos por la configuración A.

El tamaño de las imágenes de entrada se estableció en 224×224 por lo que las imágenes fueron recortadas aleatoriamente a este tamaño. Además, para aumentar el conjunto de imágenes se realizaron rotaciones horizontales aleatorias y cambios RGB aleatorios.

El entrenamiento fue llevado a cabo en un sistema equipado con cuatro GPU NVIDIA Titan Black y cada configuración tomó dos o tres semanas dependiendo de la arquitectura.

3.7.5. ResNet

Siguiendo los pasos de VGG, Resnet [28] explora estructuras más profundas, pero con capas más simples. Entrenar redes de varias capas es una tarea difícil, incrementar solamente el número de capas conduce a peores resultados. Se ha demostrado que simplemente aumentar la profundidad de la red provoca un problema de degradación. A medida que se agregan capas, la precisión aumenta hasta llegar a un punto de saturación. Luego, si se sigue agregando capas, la precisión comienza a disminuir.

Para resolver el problema de degradación, Kaiming et al. (2015) presentó las redes residuales o ResNet, con las que, por primera vez, fue posible entrenar redes de gran profundidad. En el artículo original, Kaiming evalúa arquitecturas de hasta 152 capas, ocho veces más que VGG. Con ResNet, el grupo obtuvo el primer puesto en la competencia de clasificación ILSVRC 2015.

El avance que introduce ResNet se denomina Residual Block o bloque residual. La idea detrás de los bloques residuales es que, si ya se cuenta con una red A que funciona bien, entonces se puede crear una red B más profunda que A, y hacer que la primera parte de B sea exactamente igual a A. La última parte de B implementa un mapeo de identidad, de modo que, como mínimo, puede tener la misma performance que A, pero no peor. El funcionamiento de un bloque residual se puede observar en la figura 3.8.

Un mapeo de identidad es cuando la salida de una red es igual a la entrada. Lo que aprende ResNet es la función residual $F(x) = H(x) - x$, donde si $F(x) = 0$ entonces se logra un mapeo de identidad. Dicho esto, la entrada x a una capa puede pasarse a un componente dos capas más adelante, por un lado, siguiendo el camino tradicional el cual involucra capas convolucionales y transformaciones ReLU, y por el otro, yendo directamente sin pasar por ese camino. Como resultado, la entrada al componente dos capas más adelante es $F(x) + x = H(x)$ en vez de simplemente $F(x)$.

Para lograr la arquitectura final de ResNet parten de una red plana como base. Esta red está inspirada en VGG y las convoluciones suelen tener filtros de 3×3. Luego, a la red plana se le agregan las conexiones o identity shortcuts consiguiendo de este modo una red ResNet.

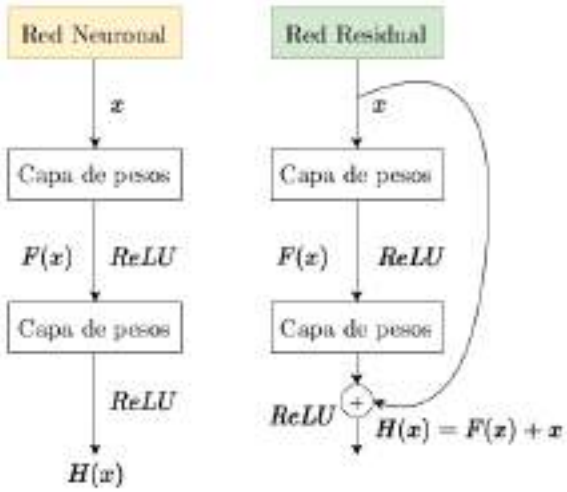


Fig. 3.8: Bloque residual

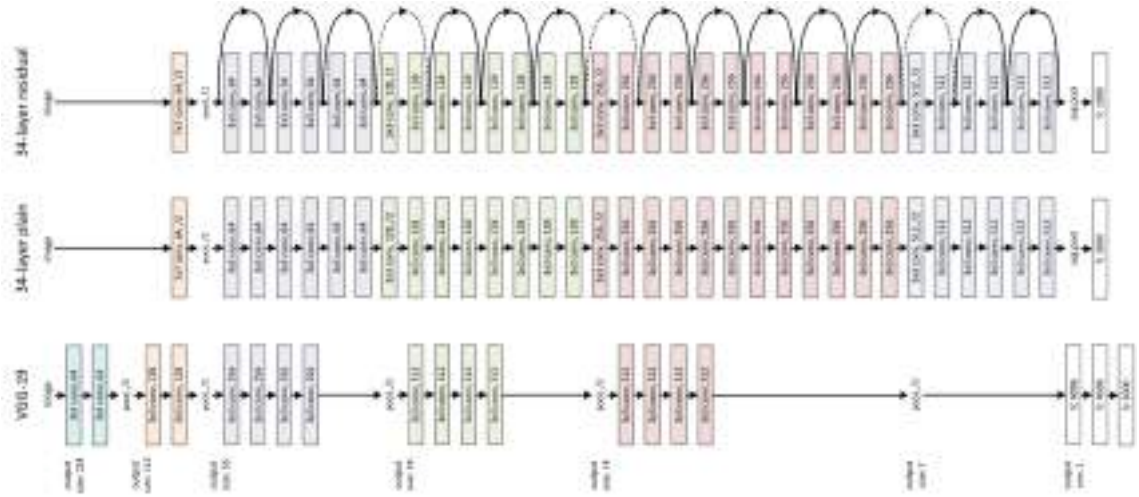


Fig. 3.9: Red neuronal con bloques residuales

3.8. Transfer learning

Transfer learning, o transferencia del aprendizaje [29], es una metodología de Deep Learning en la que un modelo que se ha entrenado para una tarea se utiliza como punto de partida para un modelo que realiza otra tarea similar. Suele ser mucho más fácil y rápido actualizar y volver a entrenar una red con transferencia del aprendizaje que entrenarla desde cero. Esta metodología se utiliza con frecuencia en aplicaciones de detección de objetos, reconocimiento de imágenes y reconocimiento de voz, entre otras.

La transferencia del aprendizaje es una técnica de uso habitual ya que:

- Permite entrenar modelos con menos datos etiquetados reutilizando modelos de uso habitual que ya se han entrenado con conjuntos de datos de gran tamaño.
- Ayuda a reducir el tiempo de entrenamiento y los recursos informáticos. Los pesos no se aprenden desde cero, dado que el modelo previamente entrenado ya ha aprendido los pesos a partir de los aprendizajes anteriores.

- Permite aprovechar arquitecturas de modelos desarrolladas por la comunidad de investigadores de Deep Learning, incluidas arquitecturas de uso habitual como GoogLeNet y ResNet.

La arquitectura de una red convolucional está dividida en capas que se encargan de aprender diferentes características de los datos. Las capas iniciales aprenden características de alto nivel y a medida que se avanza a capas más profundas las características aprendidas se tornan más complejas. Finalmente estas capas convolucionales se conectan con capas totalmente conectadas para conseguir la clasificación final.

La idea del transfer learning es tomar los pesos de modelos previamente entrenados y adaptar sus pesos al nuevo conjunto de datos. Para realizar esto hay dos posibilidades. Una es reemplazar las capas totalmente conectadas del clasificador con uno nuevo y entrenar solo los pesos de estas capas con los nuevos datos. La otra posibilidad es, además de reemplazar el clasificador, congelar los pesos de algunas de las capas convolucionales del modelo y entrenar aquellos pesos de las capas sin congelar. Esta última técnica es conocida como fine-tuning.

Por lo general, las primeras capas de los modelos suelen congelarse ya que son las encargadas de aprender características muy generales que suelen compartirse entre varios problemas, siendo la detección de bordes un ejemplo de esto.

Capítulo 4

Reconocimiento de objetos

4.1. Introducción

El reconocimiento de objetos [30] es un término general que describe un conjunto de tareas de visión computacional enfocadas a identificar objetos en imágenes. Entre esas tareas encontramos la clasificación de imágenes, que involucra la predicción de un solo objeto presente en una imagen. Otra tarea es la localización de objetos, en la cual se identifican las coordenadas de uno o más objetos dentro de una imagen junto con el bounding box de cada uno de ellos, es decir, junto con un rectángulo que contiene al objeto mismo. Por último, la detección de objetos es la combinación ambas tareas y por lo tanto involucra la localización e identificación de uno o más objetos en una imagen.

Existe una cuarta tarea dentro de la visión computacional que se llama segmentación de objetos o segmentación semántica. En este tipo de tarea en vez de identificar a un objeto junto con el bounding box que lo rodea se identifica cada píxel que conforma el objeto. En la figura 4.1 pueden observarse las cuatro tareas principales del reconocimiento de objetos.

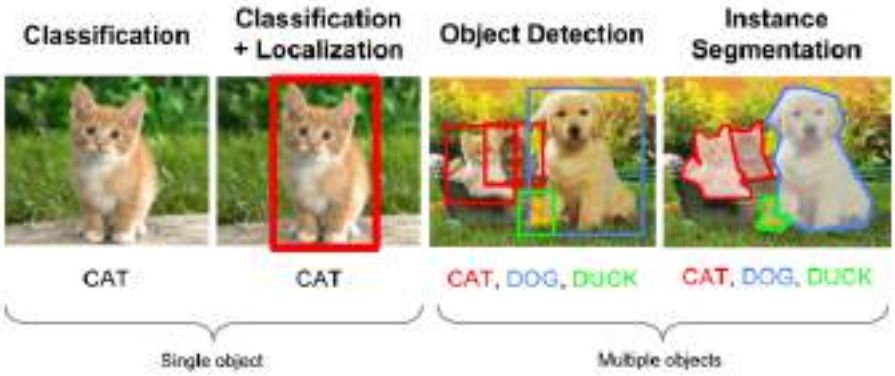


Fig. 4.1: Tareas de visión computacional

Gran parte de las innovaciones en el área del reconocimiento de objetos proviene de participaciones en la competencia ILSVRC. En esta competencia se establecieron diferentes desafíos para los distintos tipos de problemas con la intención de fomentar mejoras en cada una de las tareas de forma separada para así poder aprovecharlas de manera más horizontal. En la competencia de ILSVRC de 2015 se presentaron tres categorías. El primer

desafío fue el de clasificación de imágenes, donde los algoritmos producen una lista de categorías de objetos presentes en la imagen. El segundo desafío fue la localización de un solo objeto. En este caso los algoritmos producen una lista de categorías de objetos junto con los bounding box, pero solo se realiza una detección por categoría. Por último, el tercer desafío fue el de detección de objetos, donde los algoritmos producen una lista de objetos presentes en la imagen junto con los bounding boxes respectivos para cada instancia detectada, es decir que en este caso puede haber más de un objeto de la misma categoría.

La performance de un modelo de clasificación de imágenes es evaluada usando el error medio sobre las clases predichas. Por otro lado, la performance de un modelo de localización de un solo objeto es evaluada usando la distancia entre el bounding box esperado y el predicho para cada categoría de objeto. Por último, el desempeño de los modelos de detección de objetos es medido utilizando la precisión y el recall. Estas métricas serán explicadas más adelante en el sección 5.5.

4.2. Modelos para el reconocimiento de objetos

Las redes neuronales profundas son reconocidas por la capacidad de procesar información visual impactando de gran manera en muchos problemas de visión computacional. Entre las aplicaciones que pueden tener las redes neuronales se encuentra la detección y localización de objetos en imágenes. Estas tareas son ampliamente utilizadas en diferentes dominios, incluyendo la conducción autónoma, la videovigilancia y la medicina.

Entre las arquitecturas de redes neuronales para la detección de objetos podemos encontrar dos familias, por un lado las R-CNN y por el otro YOLO, desarrolladas a continuación.

4.2.1. R-CNN

El término R-CNN proviene del inglés Regions with CNN Features, o Region-based Convolutional Neural Network y fue desarrollado por Ross Girshick, et al [31]. En este conjunto de arquitecturas encontramos a R-CNN, Fast R-CNN y Faster R-CNN, que fueron diseñadas para realizar la localización y reconocimiento de objetos de forma separada.

La primera de ellas, R-CNN, fue presentada en 2014 por Ross Girshirk, et al. de la UC Berkeley en un paper titulado “Rich feature hierarchies for accurate object detection and semantic segmentation” [31]. Fue uno de los primeros métodos en demostrar una aplicación exitosa de las redes convolucionales al problema de la localización, detección y segmentación de objetos. El enfoque consiguió los resultados de estado del arte en el dataset VOC-2012 y también en la categoría de 200 clases del ILSVRC-2013.

La arquitectura se compone de tres módulos. El primer módulo es llamado region proposal, en el que se generan y extraen posibles regiones, es decir, bounding boxes candidatas. En el segundo módulo, llamado feature extractor, se extraen las características de cada región candidata usando redes convolucionales profundas. Por último, el tercer módulo es un clasificador, que se encarga de clasificar las características en una de las posibles clases.

El extractor de características usando en R-CNN es AlexNet, ganador del ILSVRC-2012 en la competición de clasificación de imágenes.

R-CNN es un método relativamente simple para solucionar el problema de la localización y reconocimiento de objetos, sin embargo tiene algunas desventajas. El método es

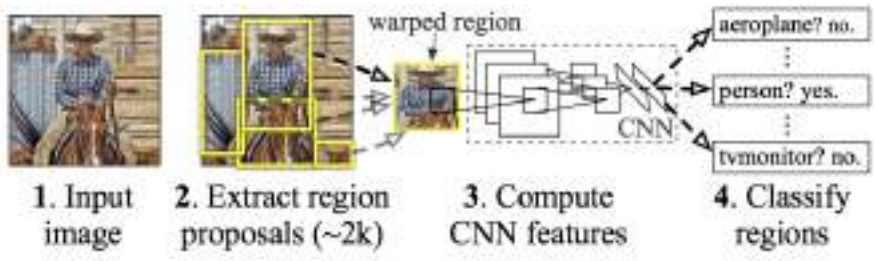


Fig. 4.2: Red convolucional basada en regiones

lento ya que requiere que cada región candidata pase a través del extractor de características. Esto es un problema ya que, como se describe en el artículo original, en las pruebas se obtuvieron aproximadamente 2.000 regiones posibles por imagen [31].

4.2.2. Fast R-CNN

Dado el éxito de R-CNN, su autor, ya trabajando en Microsoft Research, propuso una extensión para solucionar algunos problemas de velocidad de la arquitectura R-CNN.

En el artículo Fast R-CNN [32] enumera algunas limitaciones de R-CNN, que pueden ser resumidas como:

- El entrenamiento involucra varios niveles, que deben ser tratados por separado,
- El entrenamiento es costoso tanto en tiempo como en espacio ya que involucra un gran número de posibles regiones por cada imagen,
- La detección de objetos es lenta debido a la cantidad de posibles regiones por imagen.

Fast R-CNN es presentada como un solo modelo donde la salida son las regiones y las respectivas clasificaciones.

La arquitectura toma de entrada a una fotografía junto con un conjunto de posibles regiones que pasan a través de una red convolucional. Una CNN pre-entrenada, por ejemplo VGG-16, es utilizada para la extracción de características. El final de la CNN es una capa llamada Region of Interest Pooling Layer, o RoI Pooling, que extrae las características específicas para cada una de las regiones candidatas.

La salida de la CNN es interpretada por una capa fully connected y luego el modelo es bifurcado en dos, uno para realizar la predicción de la clase a través de una capa softmax, y el otro con una salida lineal que representa el bounding box. Este proceso se repite una vez por cada región de interés de la imagen.

Este modelo es significativamente más rápido para entrenar y para realizar predicciones, pero aún así todavía requiere un conjunto de regiones candidatas junto con cada imagen.

4.2.3. Faster R-CNN

En el año 2016, la arquitectura del modelo fue mejorada tanto en la velocidad del entrenamiento como en la detección. Estas mejoras fueron realizadas por Shaoqing Ren, et al. en Microsoft Research [33].

Esta arquitectura obtuvo el primer puesto en las competencias ILSVRC-2015 y MS COCO-2015 en las tareas de reconocimiento y detección de objetos.

La arquitectura fue diseñada para plantear y refinar la propuesta de regiones como parte del proceso de entrenamiento. Esas regiones luego serían usadas con la arquitectura Fast R-CNN para concluir con el diseño final del modelo. Esta mejora reduce el número de regiones candidatas y acelera el tiempo de inferencia. El módulo que trabaja junto con la arquitectura Fast R-CNN es llamada Region Proposal Network o RPN.

Faster R-CNN es la arquitectura más desarrollada de la familia R-CNN, obteniendo resultados muy buenos, cercanos al estado del arte en la tarea de reconocimiento de objetos. Una extensión de esta arquitectura que añade soporte para la segmentación de imágenes fue presentada en el año 2017 en el artículo "Mask R-CNN" [34].

4.2.4. YOLO

El modelo YOLO fue presentado en el 2015 por Joseph Redmon, et al. en un artículo titulado "You Only Look Once: Unified, Real-Time Object Detection"[24]. Ross Girshick, autor de R-CNN, fue colaborador en el desarrollo de este modelo.

La principal característica de YOLO es la integración de los procesos de detección y clasificación en una sola red, a diferencia de R-CNN donde se realizan por separado. Está mejora es la que le da el nombre a la red "You Only Look Once". La técnica tiene una precisión de predicción menor pero da la oportunidad de utilizarla en aplicaciones que necesiten inferencia en tiempo real. En su modelo básico puede procesar imágenes a 45 fotogramas por segundo y en su versión más pequeña alcanza los 150 fotogramas por segundo.

El modelo funciona dividiendo la imagen de entrada en una cuadrícula de celdas, donde cada celda es responsable de predecir una región, bounding box, si el centro de la región cae dentro de esa celda. Cada región está compuesta por las coordenadas x e y, la altura y el ancho y la confianza de la predicción.

Luego de su presentación, YOLO tuvo un gran éxito debido no solo a su precisión, sino también a su velocidad, incrementando considerablemente la cantidad de aplicaciones que la detección de objetos puede tener. Un ejemplo claro es la conducción autónoma, donde la detección en tiempo real es indispensable. Al ver las posibilidades de esta tecnología, muchos gobiernos y empresas se vieron interesados en su desarrollo. Por este motivo varias versiones de YOLO han salido a la luz. En este trabajo describiremos hasta la versión número cuatro, que es la versión que utilizamos en nuestro modelo final.

4.2.4.1. Primera versión de YOLO

Las redes R-CNN para detección de objetos tienen un desarrollo complejo y al estar conformadas por múltiples componentes son lentas y difíciles de optimizar. YOLO presenta un nuevo enfoque al considerar la detección de objetos como un problema de regresión lineal.

Una única red neuronal convolucional predice en simultáneo múltiples regiones y la probabilidad de cada clase para esa región. Tener un modelo unificado tiene varias ventajas sobre los métodos tradicionales de detección de objetos.

- YOLO es muy rápido. Debido a la simplificación de la red en un único componente, permite obtener velocidades de 45 FPS y hasta 150 FPS en su versión reducida, per-

mitiendo su utilización en aplicaciones en tiempo real. Además, alcanza una precisión de más del doble de otros sistemas de tiempo real.

- A diferencia de las técnicas de ventana deslizante o propuesta de regiones, YOLO considera toda la imagen durante el entrenamiento, aprendiendo del contexto en el que se encuentra un objeto.
- YOLO aprende representaciones de los objetos de forma generalizable, lo que permite detectar objetos aun cuando es aplicado en nuevos dominios.

Detección

YOLO utiliza características de toda la imagen para predecir cada bounding box. Por otra parte, predice todos los bounding boxes de todas las clases en una imagen simultáneamente. Esto significa que razona globalmente toda la imagen y todos los objetos en ella.

El funcionamiento del sistema consiste en dividir la imagen de entrada en una cuadrícula de $S \times S$. Si el centro de un objeto cae dentro de una celda, esta celda es responsable de detectar el objeto.

A su vez, cada celda predice B bounding boxes y la confianza asociada a esa región. La confianza representa cuán seguro está el modelo de que ese bounding box contiene un objeto y también la precisión que tiene la región que predice, y esta definida por la función 4.1.

$$Pr(object) \cdot IOU(truth/pred) \tag{4.1}$$

Cada bounding box está compuesto por cinco predicciones: x, y, w, h y la confianza. El punto (x, y) representa el centro de la región, w y h representan el ancho y el alto de la región relativos al tamaño de la imagen respectivamente.

Las celdas también predicen C probabilidades de clase condicionales, la probabilidad esta definida por la función 4.2.

$$Pr(Class|Object) \tag{4.2}$$

El tensor resultante tendrá la siguiente forma: $S \times S \times (B \cdot 5 + C)$.

Dependiendo del tamaño de las celdas y de los objetos de la imagen puede ocurrir que múltiples celdas detecten el mismo objeto, para afrontar este problema la técnica de non-maximal suppression es utilizada para que una única celda y un único bounding box sea responsable de la detección.

Arquitectura

La arquitectura de YOLO, inspirada en GoogLeNet [25], está compuesta por 24 capas convolucionales encargadas de extraer características de la imagen seguidas por dos capas totalmente conectadas que predicen las probabilidades de salida. Los módulos "Inception" utilizados en GoogLeNet fueron modificados por una capa de 1×1 de reducción seguida de una capa convolucional de 3×3. El diagrama de la arquitectura puede observarse en la figura 4.3.

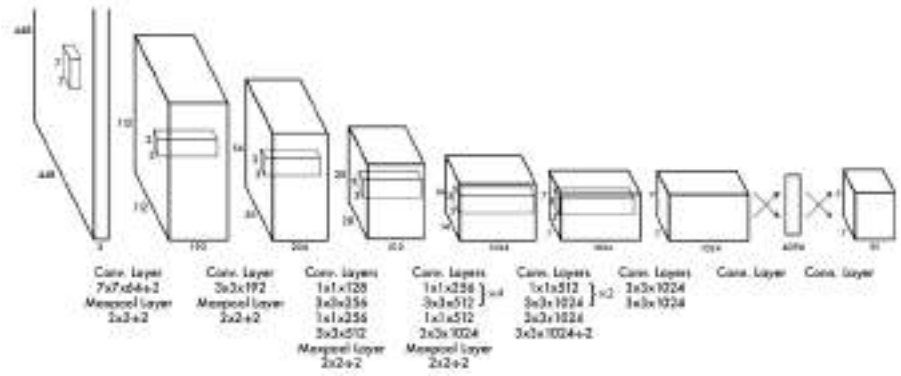


Fig. 4.3: Arquitectura YOLO

Entrenamiento

El modelo ha sido pre-entrenado con el dataset ImageNet de 1000 clases. Para esto, fueron ajustados los parámetros de las primeras 20 capas convolucionales, seguido por una capa de pooling y una totalmente conectada. Este proceso se llevó a cabo durante toda una semana para luego obtener una exactitud top-5 del 88 % sobre el dataset de validación ImageNet 2012. Tanto para el entrenamiento como para la inferencia utilizaron el framework Darknet [35].

Durante el entrenamiento se optimiza la siguiente función de pérdida [24]:

$$\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&\quad + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&\quad + \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
&\quad + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 \\
&\quad + \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \qquad (4.3)
\end{aligned}$$

Donde 1_i^{obj} establece si un objeto aparece en la celda i y 1_{ij}^{obj} establece que el bounding box número j en la celda i es el responsable de la predicción. Se utiliza la suma del error cuadrático porque es fácil de optimizar y funciona bien. Los valores x e y son las coordenadas del bounding box, w es el ancho del bounding box y h es el alto, C es la confianza, y $p_i(c)$ es la probabilidad de una clase.

En cada imagen puede haber muchas celdas que no contengan ningún objeto. Esto hace que el valor de confianza de esas celdas caiga a cero provocando inestabilidad en el modelo. Para solucionar este problema se incrementa la función de pérdida de las predicciones de coordenadas de los bounding boxes y se decrementa la función para aquellos boxes que no

contienen objetos. Para esto se utilizan los parámetros $\lambda_{coord} = 5$ y $\lambda_{noobj} = 0,5$.

Limitaciones
En está primera versión de YOLO el tamaño de los bounding boxes está restringido por el modelo. Cada celda puede predecir dos regiones y sólo contener una clase. Esto limita el número de objetos cercanos que el modelo puede predecir, siendo un problema para imágenes con muchos objetos pequeños, como una bandada de pájaros.

La función de pérdida utilizada penaliza de igual manera los errores en los objetos grandes que en los objetos chicos. Un error pequeño en un objeto de gran tamaño es insignificante pero el mismo error en un objeto muy pequeño tiene un efecto mucho mayor en el IoU. El concepto IoU se explicara en la sección 5.5. El error principal de está red son las localizaciones incorrectas de bounding boxes.

4.2.4.2. Segunda versión de YOLO

La segunda versión de YOLO se llama YOLO9000 o YOLOv2 y fue publicada en diciembre de 2016 por Joseph Redmon y Ali Farhadi [36]. Los objetivos de está versión son desarrollar una red mejor, más rápida y más robusta.

El análisis de los errores de YOLO en comparación con Fast R-CNN muestra que YOLO tiene un porcentaje muy alto de errores de localización. Además, YOLO tiene un valor de recall bajo comparado con otros métodos basados en la propuesta de regiones. En está versión se centraron en mejorar el recall y la localización manteniendo la precisión en la clasificación.

Mejoras de YOLO v2 con respecto a YOLO

Batch Normalization
La normalización por lotes disminuye el cambio en el valor de las capas ocultas y, por lo tanto, mejora la estabilidad de la red neuronal. Al añadir normalización por lotes en todas las capas convolucionales de la arquitectura el mAP 5.5.5 mejoró en un 2 %. También ayuda a regularizar el modelo y se reduce el sobreajuste.

Clasificador de mayor resolución
La versión original de YOLO tiene un tamaño de entrada de 224 x 224 para clasificación y se aumentó a 448 x 448 en YOLO v2. El aumento del tamaño de entrada de la imagen mejora el mAP hasta un 4 %.

Anchor boxes
Uno de los cambios más notables que pueden verse en YOLO v2 es la introducción de anchor boxes. Se eliminan las capas totalmente conectadas de YOLO y se utilizan anchor boxes para predecir los bounding boxes. También se reduce la red para operar con imágenes de 416 x 416 en lugar de 448 x 448 ya que un número impar de celdas ayuda en la detección de objetos grandes al tener una única celda central. Este cambio no produce un cambio positivo en el valor del mAP pero sí una mejora en el valor de recall.

En la versión anterior el tamaño de las regiones están definidas “a mano” y aunque la red puede aprender a ajustar los tamaños de forma adecuada, si se eligen mejores valores iniciales es más sencillo para la red predecir buenas detecciones. Por este motivo en YOLO

v2 se utilizó el algoritmo de k-means en los bounding boxes del conjunto de entrenamiento para encontrar buenas predicciones iniciales.

Objetos pequeños
Uno de los principales problemas a resolver en la primera versión de YOLO es la detección de objetos pequeños en una imagen. Esto se ha resuelto en la segunda versión al dividir la imagen en cuadrículas de 13×13, que son más pequeñas en comparación con su versión anterior. Esto permite identificar o localizar los objetos más pequeños en la imagen sin perder eficacia con los objetos más grandes.

Entrenamiento multiescala
La versión original de YOLO utiliza un único tamaño de entrada fijo para el entrenamiento. Debido a que el modelo sólo utiliza capas convolucionales y de pooling, puede cambiar de tamaño de entrada sobre la marcha. Con el objetivo de lograr un modelo más robusto se decidió flexibilizar el tamaño de entrada cambiando aleatoriamente el tamaño por un múltiplo de 32 entre 320 y 608 cada cierta cantidad de iteraciones. De está manera la opción más pequeña es de 320×320 y la más grande de 608×608. Esto forzó a la red a aprender a predecir correctamente en varias dimensiones diferentes con alta precisión.

Darknet 19
YOLO v2 utiliza una arquitectura llamada Darknet 19 conformada por 19 capas convolucionales, 5 capas de max-pooling y una capa softmax para la clasificación. Darknet es un framework de redes neuronales escrito en el lenguaje de programación C y utilizando CUDA. Es muy rápido en la detección de objetos, lo que es muy importante para la predicción en tiempo real. La arquitectura se muestra en la tabla 4.1.

Comparado con la versión anterior, YOLO v2 obtiene mejoras significativas en precisión, rapidez y robustez. Esta nueva versión de la red es capaz de detectar y clasificar objetos en múltiples configuraciones y dimensiones. También experimenta una gran mejora en la detección de objetos pequeños.

4.2.4.3. Tercera versión de YOLO

Tiempo más tarde, en 2018, los autores de las dos versiones anteriores de YOLO, Joseph Redmon et al., presentaron la tercera versión del modelo [37]. En este caso presentan algunas mejoras, detalladas a continuación, que fueron realizando luego de probar con distintas ideas en el modelo, dejando solo aquellos cambios con los que obtuvieron resultados satisfactorios.

Predicción de los bounding boxes
Al igual que la versión anterior, YOLO v3 utiliza el sistema de anchor boxes para la predicción de los bounding boxes. En esta nueva versión se introduce un nuevo valor en las predicciones de cada bounding box llamado objectness. Este valor debe ser 1 si el bounding box predicho se superpone completamente con un objeto verdadero más que cualquier otro bounding box.

Type	Filters	Size/Stride	Output
Convolutional	32	3 x 3	224 x 224
Maxpool		2 x 2/2	112 x 112
Convolutional	64	3 x 3	112 x 112
Maxpool		2 x 2/2	56 x 56
Convolutional	128	3 x 3	56 x 56
Convolutional	64	1 x 1	56 x 56
Convolutional	128	3 x 3	56 x 56
Maxpool		2 x 2/2	28 x 28
Convolutional	256	3 x 3	28 x 28
Convolutional	128	1 x 1	28 x 28
Convolutional	256	3 x 3	28 x 28
Maxpool		2 x 2/2	14 x 14
Convolutional	512	3 x 3	14 x 14
Convolutional	256	1 x 1	14 x 14
Convolutional	512	3 x 3	14 x 14
Convolutional	256	1 x 1	14 x 14
Convolutional	512	3 x 3	14 x 14
Maxpool		2 x 2/2	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	512	1 x 1	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	512	1 x 1	7 x 7
Convolutional	1024	3 x 3	7 x 7
Convolutional	1000	1 x 1	7 x 7
Avgpool		Global	1000
Softmax			

Tab. 4.1: Arquitectura Darknet-19

Predicción de clases

Al realizar pruebas, los autores llegaron a la conclusión de que usar softmax no era necesario para obtener una buena performance. La utilización de una capa softmax impone la suposición de que cada bounding box contiene exactamente solo una clase que, en dominios complejos, puede no ser el caso. Por este motivo, decidieron prescindir de la capa softmax y en su lugar utilizar un clasificador logístico independiente. De esta forma, el modelo es capaz de identificar bounding boxes con múltiples etiquetas, haciendo que se pueda aplicar en dominios más complejos.

Predicción en múltiples escalas

YOLO v3 realiza predicciones en tres escalas diferentes. El modelo extrae características de esas tres escalas usando un concepto similar a FPN o feature pyramid network [38].

El modelo predice un tensor de tres dimensiones que contiene las bounding boxes, el valor de objectness y las clases predichas. Luego de esta predicción, se toma el mapa de características de dos capas anteriores y se realiza un upsampling por 2x. El resultado se concatena con un mapa de características de una capa más temprana del modelo. De

esta forma se obtiene valiosa información semántica de la capa aumentada y también información con más detalles del mapa de características seleccionado. Luego se agregan más capas convolucionales para procesar la combinación de los mapas de características y eventualmente realizar una predicción del doble de tamaño que la predicción anterior. Este mismo diseño es repetido una vez más para predecir los bounding boxes en la escala final.

La primera de las tres predicciones se lleva a cabo en la capa 82, en las primeras 81 capas la imagen es muestreada en sentido descendente, de forma que la capa 81 tiene un stride de 32. Si consideramos la imagen de entrada con un tamaño de 416×416, el mapa de características resultante tiene un tamaño de 13×13. Está escala, al contener el mapa de características más chico, se encarga de la predicción de los objetos de mayor tamaño.

El mapa de características de la capa 79 es concatenado con el mapa de características de la capa 61 y luego de aplicarse una serie de capas convolucionales, en la capa 94 se realiza la segunda predicción. En está ocasión el mapa de características resultante tiene un tamaño de 26×26, por lo tanto, es el encargado de detectar los objetos de tamaño mediano.

Siguiendo un proceso similar al anterior, la tercera y última predicción se realiza en la capa 106 después de concatenar el mapa de características de la capa 91, posterior al paso por múltiples capas convolucionales, con el mapa de características de la capa 36. Nuevamente, luego de pasar por algunas capas convolucionales, se obtiene el mapa de características de 52×52. La última predicción es la encargada de detectar los objetos más pequeños presentes en la imagen.

Extractor de características

En esta versión de YOLO se presenta una nueva red de extracción de características. Surge de un enfoque híbrido entre la red Darknet-19 utilizada en YOLO v2 y las ideas detrás de las redes residuales. Esta nueva red utiliza sucesivas capas convolucionales 3×3 y 1×1 pero con algunos shortcut connections. Además, esta nueva red es significativamente más larga ya que cuenta con 53 capas convolucionales, obteniendo el nombre de Darknet-53.

Esta red es mucho más poderosa que Darknet-19, manteniendo una eficiencia superior a otras redes como ResNet-101 o ResNet-152 como se puede ver en la tabla 4.3:

Se puede observar que Darknet-53 obtiene la mayor cantidad de operaciones de punto flotante por segundo. Esto quiere decir que la red hace un buen uso de la GPU, haciéndola más eficiente y por lo tanto más rápida.

Resultados

Los resultados presentados en el artículo [37] fueron calculados sobre el dataset MS COCO [39] y muestran una mejora significativa en el average precision del modelo frente a su predecesor YOLO v2. Si bien los mejores resultados los obtiene RetinaNet, hay que tener en cuenta que demora casi cuatro veces más en realizar una inferencia.

4.2.4.4. Cuarta versión de YOLO

YOLO v4 [40] fue la primera versión de YOLO en la cual el autor principal de las versiones anteriores, Joseph Redmon, no formó parte de su desarrollo. Fue presentada en abril de 2020 por Alexey Bochkovskiy, Chien-Yao Wang y Hong-Yuan Mark Liao.

	Type	Filters	Size/Stride	Output
	Convolutional	32	3 x 3	256 x 256
	Convolutional	64	3 x 3 / 2	128 x 128
1x	Convolutional	32	1 x 1	
	Convolutional	64	3 x 3	
	Residual			128 x 128
	Convolutional	128	3 x 3 / 2	64 x 64
2x	Convolutional	64	1 x 1	
	Convolutional	128	3 x 3	
	Residual			64 x 64
	Convolutional	256	3 x 3 / 2	32 x 32
8x	Convolutional	128	1 x 1	
	Convolutional	256	3 x 3	
	Residual			32 x 32
	Convolutional	512	3 x 3 / 2	16 x 16
8x	Convolutional	256	1 x 1	
	Convolutional	512	3 x 3	
	Residual			16 x 16
	Convolutional	1024	3 x 3 / 2	8 x 8
4x	Convolutional	512	1 x 1	
	Convolutional	1024	3 x 3	
	Residual			8 x 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Tab. 4.2: Arquitectura Darknet-53

Backbone	**Top-1**	**Top-5**	**Bn Ops**	**BFLOP/s**	FPS
Darknet-19	74.1	91.8	7.29	1246	**171**
ResNet-101	77.1	93.7	19.7	1039	53
ResNet-152	**77.6**	**93.8**	29.4	1090	37
Darknet-53	77.2	**93.8**	18.7	**1457**	78

Tab. 4.3: Comparación entre redes neuronales

La gran mayoría de las redes neuronales modernas no funcionan en tiempo real y/o requieren de múltiples GPUs durante su entrenamiento con un tamaño de mini lote muy grande. El objetivo de este desarrollo fue diseñar una red capaz de detectar objetos en tiempo real que pueda ser entrenada y utilizada en una GPU convencional. El funcionamiento de los detectores de objetos en tiempo real en unidades de procesamiento gráfico (GPU) convencionales permite su uso masivo a un bajo costo, permitiendo a cualquiera entrenar y conseguir resultados de alta calidad.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD513	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 x 608	DarkNet-53	33.0	57.9	34.4	18.3	35.4	41.9

Tab. 4.4: Comparación de YOLO frente a otras arquitecturas

Los detectores de objetos modernos, como YOLO, suelen tener dos componentes principales. El primer componente es el backbone, cuya responsabilidad es la extracción de características de la entrada. El segundo componente principal es el head que se encarga de predecir las clases y los bounding boxes de los objetos. Además, en los últimos años se comenzó a utilizar un tercer componente entre el backbone y el head llamado neck. Esta última pieza posee capas que son utilizadas para obtener mapas de características de diferentes etapas del backbone persiguiendo el objetivo de agregar más información y mejorar la precisión de la red.

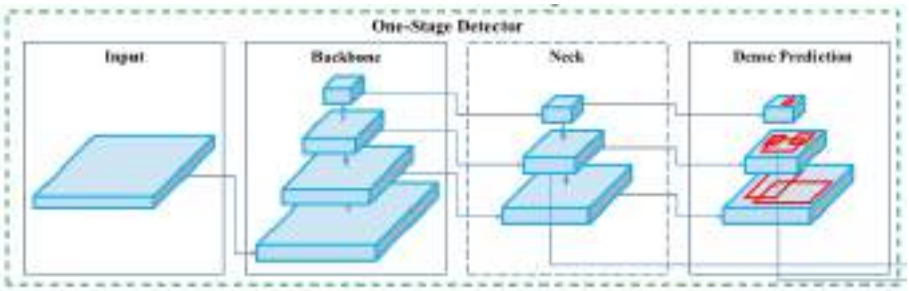


Fig. 4.4: Componentes de un detector de objetos

Existen varios modelos para utilizar como backbone de un detector de objetos. Los más conocidos son VGG, ResNet, ResNeXt y DenseNet. Por lo general, estos modelos son preentrenados, por ejemplo, con el dataset ImageNet. Con respecto al head, algunos de los detectores de una etapa más conocidos son RPN, YOLO, SSD y RetinaNet. Por último, si se decide incluir un neck, este componente suele incluir Feature Pyramid Network (FPN) y Path Aggregation Network (PAN).

Arquitectura

Para la selección de los componentes de la arquitectura se plantearon como objetivo encontrar el balance óptimo entre la resolución de entrada de la red, la cantidad de capas convolucionales, el número de parámetros de la red y la cantidad de outputs de cada capa, es decir, la cantidad de filtros a utilizar.

Como backbone se propusieron dos modelos: CSPResNext50 y CSPDarknet53. La gran cantidad de estudios que llevaron a cabo demostraron que CSPResNext50 es considerablemente mejor en la clasificación de objetos en el dataset ILSVRC2012 (ImageNet). Por otra parte, CSPDarknet53 se destaca en la detección de objetos en el dataset MS COCO.

Otro de los objetivos fue seleccionar los bloques adicionales para incrementar la cantidad de campos receptivos y elegir el mejor método de agregación de parámetros de los diferentes niveles del backbone. Un modelo que es óptimo para clasificación puede no serlo para detección de objetos. La detección requiere:

- Un mayor tamaño de la entrada a la red para detectar múltiples objetos de pequeño tamaño.
- Más capas, para que más campos receptivos cubran la entrada de la red aumentada en tamaño.
- Más parámetros, para tener una mayor capacidad de detectar múltiples objetos de diferentes tamaños en una sola imagen.

Teniendo en cuenta las numerosas pruebas que llevaron a cabo, sumado a que, como se muestra en la tabla, posee la mayor cantidad de parámetros y de campos receptivos, los autores decidieron utilizar CSPDarknet53 como backbone de YOLO v4.

La influencia de campos receptivos de diferentes tamaños se puede resumir como:

- Hasta el tamaño del objeto, permite ver al objeto completo.
- Hasta el tamaño de la red, permite ver el contexto alrededor del objeto.
- Excediendo el tamaño de la red, incrementa el número de conexiones entre un punto de la imagen y la activación final.

Además, al igual que en YOLO v3, se agregó un bloque SPP ya que incrementa significativamente los campos receptivos, separando las características más significativas del contexto sin causar una reducción en la velocidad operativa de la red. Por otro lado, para la agregación de parámetros de diferentes niveles del backbone se utiliza PANet, en contraposición a FPN utilizada en YOLO v3.

Como la capa SPP original presentada por He et al. [41] no puede ser utilizada en YOLO debido a que genera una salida para capas totalmente conectadas, en la tercera versión de la arquitectura [37] se implementó una variante de SPP donde se utilizan capas de maxpool con kernels de diferentes tamaños pero siempre con stride 1. De esta forma la salida de todas las capas de maxpool tendrán el mismo tamaño y se pueden concatenar para servir de entrada a la siguiente red convolucional totalmente conectada. Esta capa se añadió con la intención de mejorar la detección de objetos a diferentes escalas.

Finalmente, la arquitectura final de YOLO v4 queda de la siguiente manera:

- Backbone: CSPDarknet53
- Neck: SPP y PANet
- Head: YOLO v3 head

Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	38 (M)	41.2 %	62.8 %	44.3 %	20.4 %	44.4 %	56.0 %
YOLOv4	CSPDarknet-53	512	31 (M)	**43.0 %**	**64.9 %**	**46.5 %**	**24.3 %**	**46.1 %**	**55.2 %**
YOLOv4	CSPDarknet-53	608	23 (M)	43.5 %	65.7 %	47.3 %	26.7 %	46.7 %	53.3 %
Learning Rich Features at High-Speed for Single-Shot Object Detection									
LRF	VGG-16	300	76.9 (M)	32.0 %	51.5 %	33.8 %	12.6 %	34.9 %	47.0 %
LRF	ResNet-101	300	52.6 (M)	34.3 %	54.1 %	36.6 %	13.2 %	38.2 %	50.7 %
LRF	VGG-16	512	38.5 (M)	36.2 %	56.6 %	38.7 %	19.0 %	39.9 %	48.8 %
LRF	ResNet-101	512	31.3 (M)	37.3 %	58.5 %	39.7 %	19.7 %	42.8 %	50.1 %
Receptive Field Block Net for Accurate and Fast Object Detection									
RFBNet	VGG-16	300	66.7 (M)	30.3 %	49.3 %	31.8 %	11.8 %	31.9 %	45.9 %
RFBNet	VGG-16	512	33.3 (M)	33.8 %	54.2 %	35.9 %	16.2 %	37.1 %	47.4 %
RFBNet-E	VGG-16	512	30.3 (M)	34.4 %	55.7 %	36.4 %	17.6 %	37.0 %	47.6 %
YOLOv3: An incremental improvement									
YOLOv3	Darknet-53	320	45 (M)	28.2 %	51.5 %	29.7 %	11.9 %	30.6 %	43.4 %
YOLOv3	Darknet-53	416	35 (M)	31.0 %	55.3 %	32.3 %	15.2 %	33.2 %	42.8 %
YOLOv3	Darknet-53	608	20 (M)	33.0 %	57.9 %	34.4 %	18.3 %	35.4 %	41.9 %
YOLOv3-SPP	Darknet-53	608	20 (M)	36.2 %	60.6 %	38.2 %	20.6 %	37.4 %	46.1 %

Tab. 4.5: Comparación entre la tercera y cuarta versión de YOLO [40, Tabla 8]

Comparación con YOLO v3

La siguiente comparación entre YOLO v4 y YOLO v3 se realizó sobre el dataset MS COCO [39]. Para la inferencia se utilizó una GPU del tipo Maxwell, pudiendo ser tanto la GTX Titan X como una Tesla M40.

Como se puede observar en la Tabla 4.5, YOLO v4 supera tanto en velocidad como en precisión a su modelo predecesor YOLO v3.

Capítulo 5

Búsqueda de hiperparámetros

5.1. Introducción

Al momento de entrenar una red neuronal se busca encontrar los valores que mejor se ajusten a los datos de entrenamiento, es decir, aquellos para los cuales los datos de entrada con los que se entrena la red provocan la salida correcta sin perder la capacidad de generalización o, en otras palabras, responder correctamente frente a nuevas entradas. Estos valores, también llamados pesos si hablamos de redes neuronales, son los parámetros del modelo, y es lo que distingue dos modelos que poseen la misma arquitectura. En otras palabras, dos tareas diferentes se pueden resolver con la misma arquitectura de red neuronal, la diferencia entre estos dos modelos estará en los parámetros de cada uno.

Por otro lado podemos encontrar los hiperparámetros, que son las variables que rigen el proceso de entrenamiento de la red y no están relacionados con los datos de entrenamiento. Un ejemplo de esto puede ser la configuración de la red neuronal, como el padding y el stride, o la cantidad de capas ocultas a utilizar.

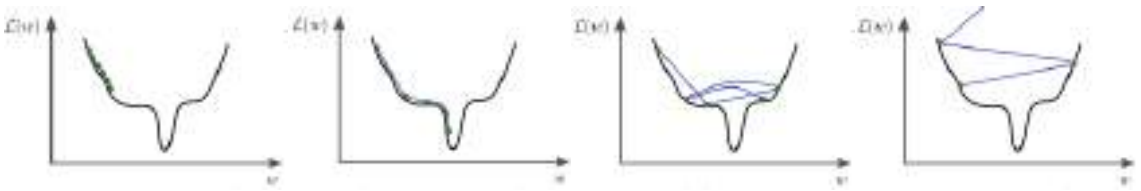
Los parámetros de un modelo se ajustan durante el proceso de entrenamiento. Después de ingresar un conjunto de datos al modelo, se comparan los resultados obtenidos con los valores esperados para ese conjunto y en función del error resultante se optimizan los parámetros. En cambio, los hiperparámetros se configuran antes de comenzar un entrenamiento. Sus valores pueden elegirse a partir de la experiencia de los desarrolladores y/o utilizando técnicas para encontrar los mejores hiperparámetros para un problema específico.

5.2. Hiperparámetros

Existe una gran cantidad de hiperparámetros dependiendo del tipo de modelo con el que estemos trabajando. En este trabajo nos centraremos en las redes neuronales.

Podemos clasificar los hiperparámetros en dos grupos [42]: aquellos que definen la estructura y arquitectura de la red, y aquellos que determinan el comportamiento durante el entrenamiento. Dentro del primer grupo podemos agrupar al número de capas ocultas, tamaño y tipo de filtro, función de activación, stride o padding, dropout, entre otros. El segundo grupo engloba la tasa de aprendizaje, momentum, tamaño de lote, número de épocas y weight decay.

En este trabajo utilizaremos Transfer Learning 3 a partir de una arquitectura preentrenada, por lo tanto, no realizaremos cambios en los hiperparámetros del primer gru-



(a) (b) (c) (d)

Fig. 5.1: Efecto de la tasa de aprendizaje en la búsqueda de un mínimo. (a) Tasa de aprendizaje muy pequeña. (b) Tasa de aprendizaje ideal. (c) Tasa de aprendizaje grande. (d) Tasa de aprendizaje muy grande.

po. A continuación entraremos más en detalle en los hiperparámetros más relevantes del segundo conjunto.

5.2.1. Tasa de aprendizaje

La tasa de aprendizaje [18] es el hiperparámetro que determina cuánto afecta el error obtenido en una iteración a los pesos y biases de cada una de las neuronas del modelo en la dirección opuesta del gradiente local de error.

El ajuste aplicado a la neurona w que conecta la capa j con la i se define como:

$$\Delta w_{ji}(n) = \eta \cdot \delta_j(n) \cdot y_i(n) \tag{5.1}$$

Siendo $\Delta w_{ji}(n)$ el ajuste del peso a actualizar, η la tasa de aprendizaje, $\delta_j(n)$ el gradiente local y $y_i(n)$ la salida de la neurona i, o lo que es lo mismo, la entrada de la neurona j.

Elegir la tasa de aprendizaje ideal para un modelo es un desafío complejo. Una tasa de aprendizaje muy pequeña se traslada a cambios muy pequeños en los pesos de la arquitectura haciendo muy lenta la convergencia al mínimo o incluso estancarse en un mínimo local sin poder salir de él.

En el otro extremo, una tasa de aprendizaje demasiado grande produce cambios muy abruptos. Un paso grande puede ser útil al inicio del entrenamiento cuando se quiere ir rápido pero tiene la desventaja de que puede dar saltos aleatorios sin alcanzar el mínimo o también, en el peor de los casos, volver inestable a la red.

Por lo general este hiperparámetro no se mantiene constante a lo largo de todo el proceso de entrenamiento sino que una buena solución suele ser arrancar desde un valor más grande y a medida que las épocas de entrenamiento transcurren ir disminuyendo el valor. De esta manera se obtiene un entrenamiento más rápido al inicio y a medida que nos vamos acercando a mejores resultados los ajustes a los pesos son menores. En la Figura 5.1 podemos ver graficados distintos valores de la tasa de aprendizaje y cómo varía su comportamiento.

5.2.2. Momentum

El método del descenso por gradiente es un algoritmo de optimización que consiste en calcular el gradiente de la función a optimizar y moverse en la dirección del mayor gradiente ya que este valor representa el camino hacia un mínimo o máximo, repitiendo estos dos pasos hasta que se cumpla cierta condición que indique haber encontrado un

mínimo. De esta forma, se garantiza que en cada paso el valor de la función a minimizar es cada vez más pequeño.

El problema es que la dirección puede variar significativamente en algunos puntos de la función, y en otros mantenerse constante. Esto, en combinación con la velocidad de aprendizaje puede tener efectos indeseables sobre el sistema. Es por eso que, con el objetivo de mantener una dirección consistente durante el entrenamiento, se utiliza el Momentum [43].

Con la técnica del Momentum, para actualizar los pesos se toma una combinación del nuevo gradiente calculado junto con los calculados en pasos anteriores, es decir, la nueva dirección tiene componentes de direcciones pasadas. Por ejemplo, si se tiene un momentum de 0,9, se toma 90 % de los gradientes previos más 10 % del nuevo gradiente, y con eso se actualizan los pesos. De esta forma se consigue tender al mínimo de forma más consistente a través de cada iteración.

5.2.3. Tamaño de lote

El tamaño de lote [44] es un hiperparámetro que define el número de muestras que pasan a través de la arquitectura antes de ajustar nuevamente los pesos sinápticos de la misma. Existen principalmente 3 diferentes alternativas de entrenamiento:

- **Entrenamiento por Descenso por Gradiente Estocástico (SGD)**: En este caso se utiliza un tamaño de lote de una sola muestra, es decir, ingresa una muestra a la red, se realiza la predicción, se calcula el error y la retropropagación del mismo. Se actualizan los pesos de la red después de cada una de las muestras del conjunto de datos de entrenamiento.

- **Entrenamiento por lote**: El tamaño de lote utilizado en esta estrategia es igual al tamaño del conjunto de datos de entrenamiento, en otras palabras, los pesos de la red son ajustados cuando todas las muestras del conjunto pasaron por la arquitectura. El entrenamiento por lote minimiza el error total.

- **Entrenamiento por mini lotes**: En el entrenamiento por mini lotes se divide el conjunto de datos en subconjuntos del mismo tamaño y después de pasar cada uno de estos subconjuntos se actualizan los pesos sinápticos. Es una solución intermedia entre el entrenamiento por lote y el entrenamiento por SGD. Suele utilizarse potencias de 2 como valores recomendados de tamaño de mini lotes, como por ejemplo 2, 4, 8, 16, 32, 64, 128 o 256.

Un tamaño de lote muy grande toma en cuenta el error total, pero requiere de mucha memoria y costo computacional lo cual puede ser un problema, por otro lado, un tamaño pequeño introduce más fluctuaciones en el error. Los valores estándares suelen ser 32 o 64 muestras.

5.2.4. Número de épocas

El método de descenso por gradiente es un algoritmo de optimización iterativo para encontrar la mejor función, o pesos de la red neuronal, que modele los datos de entrenamiento. Que sea iterativo significa que se necesita evaluar la función reiteradas veces hasta encontrar el óptimo. Se dice que se entrenó durante una época cuando todos los datos

de entrenamiento pasaron por la red sólo una vez. Esto es importante ya que, al estar utilizando el método de descenso por gradiente, se necesita entrenar a la red neuronal por más de una época para obtener buenos resultados.

Un entrenamiento pobre en la cantidad de épocas provoca que el modelo resultante tenga subajuste o underfitting, es decir, que todavía tenga mucho margen para seguir entrenando. En el otro extremo, cuando la cantidad de épocas es excesiva se produce el fenómeno contrario, el overfitting. En este caso, se entrenó la red neuronal por tantas épocas que el modelo resultante modela a la perfección los datos de entrenamiento perdiendo una de las características más buscadas a la hora de entrenar redes neuronales, la generalización del aprendizaje. Por último, cuando se entrena con la cantidad de épocas correctas, se dice que el modelo final es óptimo, logrando modelar los datos al mismo tiempo que puede generalizar nuevos casos.

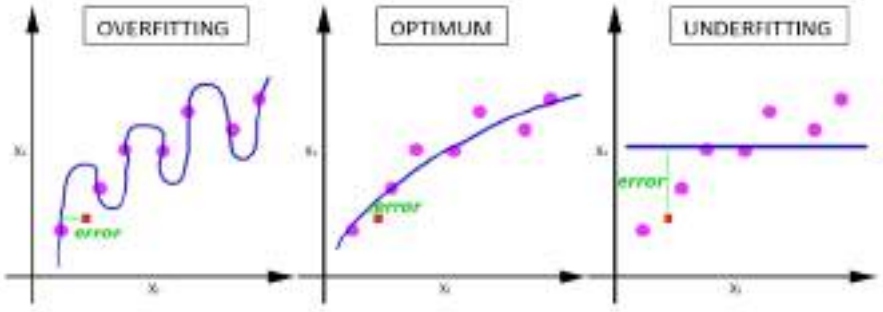


Fig. 5.2: Comparación visual entre los fenómenos overfitting y underfitting

Un concepto relacionado al de época es el de batches. Debido a que actualizar todos los pesos de la red luego de pasar por ella cada uno de los datos de entrenamiento es muy costoso computacionalmente, se divide el set de entrenamiento en subconjuntos llamados batches. Los pesos son actualizados una vez evaluada la red con cada uno de los datos de un batch, es decir, al final de cada iteración. La cantidad de datos en cada uno de los batches se la conoce como batch size.

5.2.5. Weight Decay

La regularización es una técnica que ayuda a prevenir el overfitting o reducir la varianza en la red como se observa en la imagen 5.3. Esto lo hace penalizando la complejidad, ya que esta complejidad hace que nuestro modelo no generalice bien el aprendizaje aún cuando tiene buena performance con los datos de entrenamiento. Por lo tanto, con la regularización lo que se intenta conseguir es una relación de compromiso entre ajustar correctamente los datos de entrenamiento y generalizar bien a datos nunca vistos.

Una primera aproximación para regularizar una red es sumar un término al error calculado en el entrenamiento con el objetivo de penalizar los pesos grandes. De esta forma se busca que los pesos sean lo más pequeños posible, evitando que algunos tengan más protagonismo que otros y reduciendo la complejidad de la red. Una de las técnicas de regularización más conocidas se denomina L2 regularization [1] o weight decay.

Está pequeña penalización a la función de costo del modelo hace que los pesos y biases no se ajusten excesivamente ante un dato particular aislado sino que tiendan a ajustarse para el mayor conjunto de datos, reduciendo el overfitting y permitiendo al modelo tener buenos resultados ante datos nuevos no utilizados durante el proceso de entrenamiento.

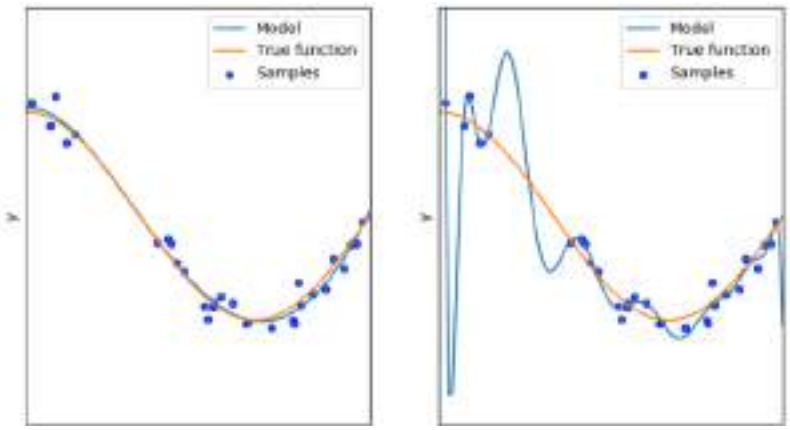


Fig. 5.3: A la izquierda se observa un modelo óptimo mientras que el modelo de la derecha posee gran varianza y por lo tanto mayor complejidad.

5.3. Optimizadores

Para entrenar una red neuronal se debe definir una función de pérdida, costo o error para poder medir la diferencia entre la predicción de nuestro modelo y lo que se quiere predecir. Lo que se busca es un conjunto de pesos para que la red neuronal pueda hacer predicciones precisas, es decir, que el resultado de la función de pérdida sea lo más chica posible.

Con esta técnica, se calcula el gradiente de la función de pérdida con respecto a los pesos de la red que queremos optimizar. Luego, los pesos son actualizados en la dirección negativa del gradiente ya que en ese sentido se intenta encontrar el mínimo de la función de costo.

Al aplicar el descenso por gradiente a los pesos de forma iterativa, eventualmente llegaremos a un conjunto de pesos óptimos que minimizan la función de pérdida y permitan a la red neuronal realizar mejores predicciones.

Este método es la base para algoritmos de optimización más complejos que disminuyen algunos problemas que se presentan al utilizarlo. A continuación se describirán algunos de los algoritmos más conocidos y utilizados.

5.3.1. Descenso por gradiente estocástico en mini-lotes

Calcular el gradiente para cada dato de entrenamiento es muy costoso. Con el intento de mitigar este problema surgió el método del descenso por gradiente estocástico o SGD, por sus siglas en inglés, en mini-lotes. En este caso, se toma un subconjunto o mini-lotes aleatorio del conjunto de entrenamiento y se calcula el gradiente solo a ese mini-lote. De esta forma, se introduce aleatoriedad al algoritmo dificultando el estancamiento y se incrementa la velocidad de entrenamiento mediante el uso de mini-lotes.

5.3.2. Adam

Entre la gran cantidad de optimizadores que fueron desarrollados como alternativa al clásico algoritmo de descenso por gradiente estocástico, uno de los más utilizados hoy en día es el optimizador Adam [45] cuyo nombre deriva del inglés adaptive moment estimation.

Fue presentado por Diederik Kingman de OpenAI y Jimmy Ba de la Universidad de Toronto en 2015. Entre sus ventajas los autores destacan la simpleza de su implementación y la eficiencia.

Una de las grandes diferencias entre el algoritmo SGD y Adam es la forma en que manejan la tasa de aprendizaje. SGD mantiene una sola tasa de aprendizaje para la actualización de todos los pesos. En cambio, Adam mantiene un valor para cada peso de la red que es actualizado por separado.

5.4. Búsqueda de hiperparámetros

Como dijimos anteriormente, los hiperparámetros no se pueden obtener de los datos, por lo tanto, hay que elegirlos al comenzar el entrenamiento. El ajuste o búsqueda de hiperparámetros es el proceso de encontrar la mejor combinación de hiperparámetros que tenga los mejores rendimientos en la red neuronal a utilizar. La cantidad de distintos hiperparámetros de una red varía mucho, y puede llegar a ser muy grande dependiendo de la arquitectura. A su vez, cada hiperparámetro puede tomar valores muy diversos.

En nuestro caso particular, al estar utilizando una red pre-entrenada, los hiperparámetros asociados a la arquitectura están definidos y optimizados, por lo que nos centraremos en los hiperparámetros que están vinculados con el entrenamiento.

Definir que hiperparámetros se van a optimizar, y cuales dejar constantes, es una difícil tarea en la cual se debe tener en cuenta la técnica a utilizar, los recursos disponibles y cuales hiperparámetros van a tener un mayor impacto en el modelo. No existe ningún dogma que especifique qué hiperparámetros son más relevantes para cada arquitectura o aplicación, es por esto que tanto los hiperparámetros elegidos como los valores que pueden tomar fueron seleccionados en base a ser los más utilizados en la literatura y el conocimiento y la experiencia de los participantes.

Se decidió utilizar los hiperparámetros batch size, learning rate, momentum y weight decay.

5.4.1. Exploración de hiperparámetros

Existen diferentes técnicas para la exploración de los hiperparámetros. Entre las más conocidas se encuentran: optimización bayesiana, búsqueda por grilla, búsqueda aleatoria y Hyperband. En este trabajo analizaremos en más profundidad la búsqueda por grilla y la optimización bayesiana.

5.4.1.1. Búsqueda por grilla

La búsqueda por grilla [44] es uno de los algoritmos de búsqueda de hiperparámetros más conocidos y utilizados en la actualidad. Su funcionamiento consiste en aplicar la fuerza bruta para encontrar la combinación de valores que mejores resultados obtenga para el modelo empleado. Se debe definir un espacio de valores para cada uno de los hiperparámetros que se quiere optimizar y se ejecutan todas las combinaciones posibles de estos. Debido a la independencia de cada una de las evaluaciones con respecto a las otras es posible paralelizar la búsqueda por grilla permitiendo reducir el tiempo de su realización. La búsqueda por grilla tiene la "maldición de la dimensionalidad", es decir, su utilización está limitada a un grupo reducido de hiperparámetros y de valores para cada uno de ellos.

5.4.1.2. Búsqueda aleatoria

La búsqueda aleatoria [44], al igual que la búsqueda por grilla, parte de un conjunto de valores para cada hiperparámetro, pero, en este algoritmo, solo algunas combinaciones aleatorias de estos valores son evaluadas. Esto hace que el tiempo requerido para su implementación sea mucho menor. Este algoritmo también sufre con la maldición de la dimensionalidad.

5.4.1.3. Optimización bayesiana

Una de las principales desventajas de la búsqueda por grilla y aleatoria es que no tienen en cuenta los resultados de las evaluaciones anteriores, por lo cual, si el espacio de búsqueda está mal delimitado se van a producir muchas evaluaciones innecesarias. Una alternativa para este problema es la optimización bayesiana [46].

El funcionamiento de la optimización bayesiana consiste en crear un modelo probabilístico para una función objetivo $f(x)$ previamente definida, como pueden ser la precisión o RMSE, y utilizarlo para tomar decisiones sobre el siguiente conjunto de hiperparámetros x que va a utilizar para evaluar la función. A partir de todos los valores obtenidos anteriormente para $f(x)$, no solo el último, junto con el gradiente local y la matriz Hessiana define el próximo valor. Este algoritmo puede encontrar el mínimo de funciones muy complejas con muchas menos evaluaciones que los métodos convencionales a costa de un mayor costo computacional en cada iteración.

Este algoritmo tiene la desventaja de que no puede ser paralelizable ya que todas las evaluaciones realizadas tienen influencia en la decisión del nuevo subconjunto de hiperparámetros con el que se va a evaluar el modelo.

5.4.2. Validación Cruzada

La Validación Cruzada (Cross Validation) es un método estadístico para evaluar, comparar y determinar la habilidad de predicción de modelos de ML sobre un subconjunto de datos que no fueron utilizados para el entrenamiento.

La idea principal detrás de la validación cruzada es dividir los datos, una o más veces, en dos subconjuntos, uno de ellos se denomina de entrenamiento que, como su nombre lo indica, se utiliza para entrenar el modelo y el otro subconjunto se denomina de validación y es usado para evaluar el modelo, medir su habilidad de generalización y obtener el error.

La validación cruzada ayuda a reducir el sobreajuste porque los datos utilizados para el entrenamiento son independientes de los datos utilizados para la validación, siempre que se cumpla que los datos son independientes e idénticamente distribuidos.

En la mayoría de las estrategias de validación se divide y evalúa el modelo múltiples veces con diferentes particiones del mismo conjunto de datos y se obtiene una calificación global de todas las evaluaciones, como puede ser el promedio, para reducir la aleatoriedad de una única partición.

Si bien existen muchas estrategias de validación cruzada en el área del ML, como por ejemplo k-fold, Monte Carlo y Leave One Out, la más utilizada es k-fold.

5.4.2.1. K-fold cross validation

La validación por k-fold [47] consiste en dividir el conjunto de datos en k particiones de igual tamaño, siendo la unión de todas las partes el conjunto completo de datos y la

intersección entre dos particiones igual a vacío. Es decir, todas las muestras del dataset están en una y solo una de las k particiones.

El algoritmo consiste de k iteraciones y en cada una de las iteraciones una de las particiones es usada como conjunto de validación y las k-1 particiones restantes como conjunto de entrenamiento. El error total se calcula como el promedio de los errores parciales de cada una de las iteraciones.

Es importante que la partición se realice una única vez y se utilicen las mismas para cada una de las iteraciones. De esta manera, una única muestra va a formar parte del conjunto de validación una única vez y k-1 veces va a encontrarse dentro del conjunto de entrenamiento.

El valor de k utilizado debe ser elegido cuidadosamente porque una mala elección puede resultar en un valor erróneo de la habilidad del modelo, como una alta varianza o un alto sesgo. Cuanto más grande sea el valor de k, la diferencia en tamaño entre el conjunto de entrenamiento y los subconjuntos de muestreo es menor y cuanto menor es esta diferencia el sesgo se vuelve más pequeño. Una de las tácticas más utilizadas para definir el valor de k es elegir un valor que resulte en particiones lo suficientemente grandes para ser estadísticamente representativas del conjunto total de datos. Otra opción es elegir un k=10 que es un valor que se ha encontrado mediante experimentación que generalmente resulta en una estimación del modelo con bajo sesgo y modesta varianza. Otro valor muy utilizado y recomendado es k=5 que al igual que k=10 se ha demostrado empíricamente que no producen un sesgo excesivamente grande ni una varianza muy elevada.

El procedimiento general de k-fold es:

1. Dividir el dataset en k grupos
2. Para cada iteración
   a) Se selecciona un grupo como conjunto de validación.
   b) Se seleccionan los k-1 grupos restantes como conjunto de entrenamiento.
   c) Se entrena al modelo con el conjunto de entrenamiento y se evalúa con el de validación.
   d) Se almacena el valor obtenido para la iteración.
3. Se obtiene el valor general promediando los resultados de todas las iteraciones.

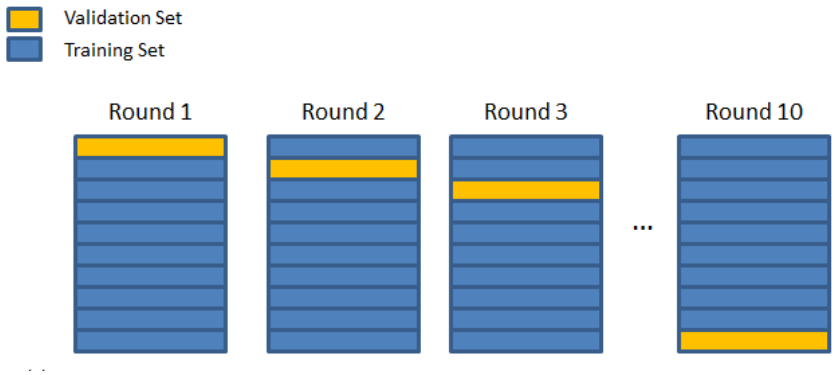


Fig. 5.4: Representación gráfica del algoritmo k-fold

K-fold es un método popular porque es sencillo de entender y porque generalmente da como resultado una estimación menos sesgada de la habilidad del modelo que otros métodos, como una simple división de entrenamiento y validación.

5.4.2.2. Monte Carlo Cross Validation

La validación cruzada de Monte Carlo (MCCV) [47] es otra de las técnicas más difundidas. A diferencia de la validación cruzada de k-fold, la técnica de MCCV consiste en dividir el conjunto de datos en los subconjuntos de entrenamiento y validación de manera aleatoria y con la posibilidad de que existan repeticiones en distintas iteraciones. En está estrategia, al igual que en k-fold, una muestra del conjunto va a estar contenida solo dentro de uno de los subconjuntos, pero, en contraparte a lo que ocurre en otras técnicas, la misma muestra puede pertenecer al subconjunto de validación en más de una iteración.

En Monte Carlo el porcentaje del conjunto de datos que pertenece a los subconjuntos es decisión de los desarrolladores. Algunos porcentajes comunes son 70-30, 75-25 o 80-20. A su vez, el número de iteraciones que se van a evaluar también debe ser definido por el equipo. Cabe destacar que cuanto mayor sea el número de iteraciones evaluadas menor será la incertidumbre en la estimación obtenida. Se deberá seleccionar un valor que sea un balance entre la incertidumbre permitida, el costo computacional y tiempo que se quiera emplear.

La principal ventaja de este método frente a k-fold es que no hay una relación entre el porcentaje de datos utilizados para el entrenamiento y la cantidad de iteraciones, esto se debe a que la división se hace con repetición, permitiendo a una muestra pertenecer al subconjunto de validación en más de una iteración.

La desventaja es que existe la posibilidad de que una muestra no forme parte en ninguna de las iteraciones del subconjunto de validación, mientras que otras muestras pueden ser parte de este conjunto más de una vez.

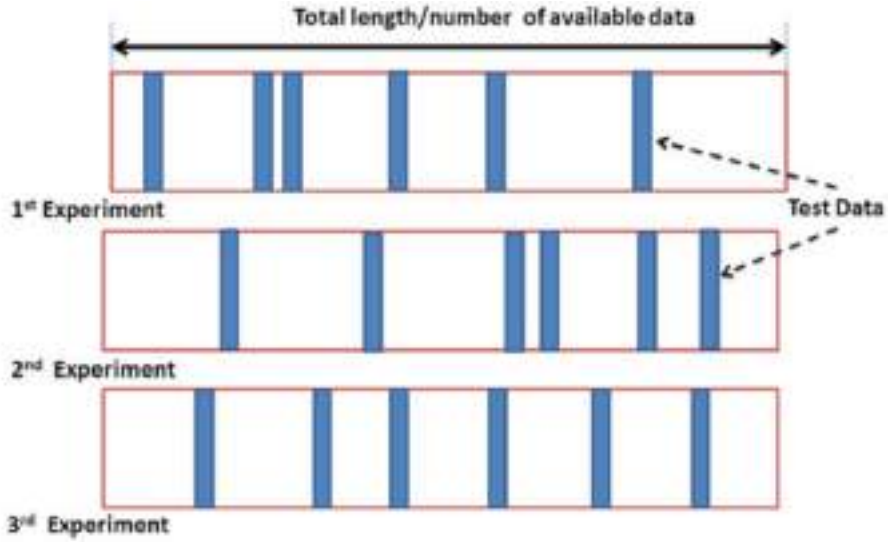


Fig. 3.7 Data splitting in the random sub-sampling approach

Fig. 5.5: Subdivisión de los datos de forma aleatoria

5.5. Métricas

Al buscar cuál es la mejor combinación de hiperparámetros que se adapte a nuestro problema debemos probar varias veces hasta llegar a una conclusión. Para comparar cada una de estas pruebas debemos tener en claro cómo vamos a medir la performance de nuestro modelo. Para hacer esto tenemos un abanico de métricas tanto para problemas de clasificación como de regresión. A continuación describiremos métricas para problemas de clasificación que es lo que nos incumbe en este trabajo.

5.5.1. Matriz de confusión

La matriz de confusión para un problema de n clases es una matriz de dimensión $n \times n$ donde en una de las entradas se colocan las clases reales y en la otra las clases predichas por el modelo. De esta forma, la diagonal principal contiene todas las predicciones correctas del modelo mientras que todo lo demás son los errores.

Fig. 5.6: Matriz de confusión binaria con las clases reales en las columnas y las clases predichas en las filas

Esta herramienta es muy útil y ampliamente utilizada ya que permite analizar el comportamiento del modelo de una forma rápida y visual, pudiéndose detectar no solo los aciertos del modelo sino también con que clases es que se confunde.

Según en qué parte de la matriz de confusión esté una predicción, se la puede clasificar en las siguientes cuatro opciones:

- **Verdaderos positivos**: El valor real es positivo y la predicción dió positivo.
- **Verdaderos negativos**: El valor real es negativo y la predicción dió negativo.
- **Falsos positivos**: El valor real es negativo y la predicción dió positivo.
- **Falsos negativos**: El valor real es positivo y la predicción dió negativo.

Como se puede observar, los dos primeros casos representan los aciertos del modelo mientras que los últimos dos los fracasos.

5.5.2. Precisión

La precisión mide el porcentaje de predicciones correctas realizadas por el modelo. Es decir, de todas las predicciones positivas, cuántas realmente lo eran.

$$Precision = \frac{TP}{TP + FP} \tag{5.2}$$

5.5.3. Recall

La métrica de recall mide qué tan bueno es el modelo prediciendo los positivos. En otras palabras, qué cantidad de positivos el modelo logró detectar correctamente.

$$Recall = \frac{TP}{TP + FN} \tag{5.3}$$

5.5.4. F1-score

El valor F1 se utiliza para combinar las medidas de precisión y recall en un solo valor. Esto es práctico porque hace más fácil la comparación del rendimiento combinado de ambas métricas entre varias soluciones.

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.4}$$

5.5.5. mAP

Dada la importancia del recall y de la precisión, se puede graficar ambas métricas en una curva precisión-recall para poder encontrar más fácilmente el punto óptimo como se muestra en la imagen 5.7.

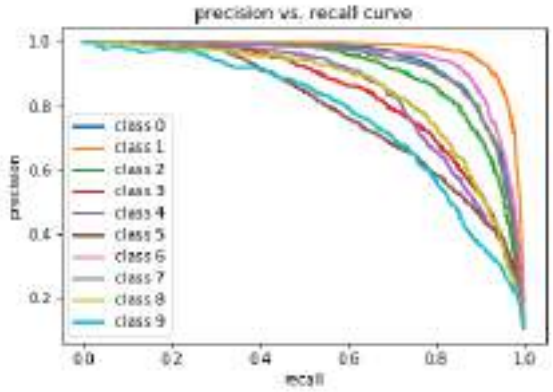


Fig. 5.7: Curva precision-recall para varias clases de un modelo

Se conoce como average precision (AP), o precisión media, al área bajo la curva precisión-recall. Debido a que tanto la precisión como el recall siempre son valores entre cero y uno, el AP también estará entre estos dos valores. Cuando se tiene un modelo con más de una clase, se calcula el AP para cada una de ellas y luego se realiza el promedio entre todas, para así obtener lo que se conoce como mAP o mean average precision.

5.5.6. IoU

En el contexto de detección de objetos en imágenes, una métrica popular es la intersección sobre la unión, o más conocida por sus siglas en inglés IoU. Está métrica mide la superposición entre dos áreas o bounding boxes. Se utiliza para medir cuánto del bounding box predicho se superpone con el real.

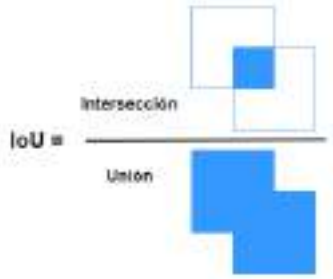


Fig. 5.8: Intersección sobre la unión

Capítulo 6

Hardware para Deep Learning

6.1. Introducción

Históricamente, el progreso en las investigaciones en redes neuronales y Deep Learning han sido influenciadas por la disponibilidad de hardware y herramientas de software [48].

La inteligencia artificial moderna es impulsada por el Deep Learning, que tuvo sus orígenes en experimentos con redes neuronales electrónicas en la década de 1950. El Deep Learning se basa en cuatro ideas: (1) Las funciones complejas pueden ser construidas de manera eficiente ensamblando bloques funcionales simples en grafos multicapas; (2) La función deseada puede ser aprendida ajustando los parámetros a partir de ejemplos; (3) El procedimiento de aprendizaje es minimizar una función objetivo mediante un método basado en el gradiente; (4) El gradiente puede ser calculado de forma eficiente y automática utilizando el algoritmo de backpropagation.

Desde la década de 1950 hasta los ochentas, los modelos de redes neuronales utilizaron neuronas binarias: el modelo de neurona presentado por McCulloch-Pitts calcula la suma ponderada de sus entradas y luego pasa el resultado a través de la función signo. Esto es así porque la operación de multiplicación era extremadamente costosa en tiempo, ya sea implementada en circuitos analógicos, en circuitos digitales o por medio de software. Siendo las entradas binarias, la operación de suma era más que suficiente para calcular la suma de los pesos de la neurona.

El perceptrón original de 1957 fue construido en una computadora analógica dedicada donde los pesos eran implementados mediante potenciómetros motorizados. Por su parte, Adaline, el modelo rival, estaba implementado con memristores electroquímicos y utilizaban relés para representar los pesos. A partir de ese momento se perdió el interés por las redes neuronales por casi dos décadas, hasta que a mediados de la década de 1980 surgió una segunda ola de interés, destacándose el trabajo en redes Hopfield, máquinas de Boltzmann y la popularización del algoritmo de backpropagation.

Una de las razones por la cual el algoritmo de backpropagation no apareció hasta finales de 1980 es porque requiere que las neuronas utilicen funciones de activación continuas no lineales, como por ejemplo la función sigmoide. Este tipo de funciones no fueron viables hasta que el desempeño de las computadoras utilizadas alcanzaron la cantidad de un millón de operaciones de punto flotante por segundo.

A mediados de la década de 1990 el Deep Learning perdió popularidad. Esto fue debido, entre otras cosas, al bajo desempeño de las computadoras del momento, la reducida cantidad de aplicaciones para la cual reunir los datasets necesarios era efectivo y el rechazo

de muchas instituciones de investigación a abrir el código fuente de sus desarrollos.

A partir de 2013 hubo un resurgimiento en la popularidad de las redes neuronales. Esto fue causado, principalmente, por la disponibilidad de grandes datasets con muchas muestras y categorías, por el acceso a procesadores gráficos de propósito general (GPGPU) de bajo costo y con performance del orden de los TFLOPS, y por la aparición de librerías de código abierto para el desarrollo de software.

Antes del surgimiento de las GPGPUs, hubo intentos de construir hardware dedicado para las arquitecturas de redes neuronales, pero ninguno de esos intentos tuvo éxito. Esto es debido a la falta de flexibilidad de las soluciones y porque diseñar el hardware para un tipo en particular de red neuronal no demostró utilidad alguna. Por estos motivos es que las GPGPUs tienen tanto éxito: tienen amplia disponibilidad, son de uso general y se pueden programar.

Debido al éxito de la operación de convolución en el análisis de imágenes, Bell Labs desarrolló un chip llamado Net32k con 256 neuronas, cada una con 128 pesos ternarios, capaz de realizar 320×10^9 operaciones sinápticas por segundo. Casi en simultáneo, fue desarrollado otro chip llamado ANNA (Analog Neural-Network Accelerator) específicamente para correr redes neuronales, contenía 64 neuronas con 64 pesos cada una. El chip corría a 20Mhz y era capaz de realizar 4×10^9 operaciones por segundo. Al igual que Net32K, ANNA utilizaba un circuito de registro de desplazamiento para minimizar el acceso a memoria externa, permitiéndole correr arquitecturas con 130.000 conexiones en alrededor de 1 ms, es decir, 500 veces más rápido que la mejor computadora del momento.

Ambos chips, Net32k y ANNA, estuvieron muy cerca de utilizarse en aplicaciones comerciales, pero eventualmente fueron reemplazados por implementaciones en software sobre DSPs de punto flotante. Esto refleja lo difícil de llevar al mercado arquitecturas e implementaciones en hardware. Luego de dos años, las FPGAs lograron ser lo suficientemente potentes como para igualar el desempeño de ANNA.

En la década 1990 se produjo un segundo invierno para las redes neuronales que duró casi una década. La razón está ligada al hardware, al software y a los datos disponibles. Las redes neuronales requieren grandes cantidades de datos para ser entrenadas, pero, en la era pre-internet solo había disponibilidad de datasets de calidad para algunas tareas como el reconocimiento de voz o de manuscritos. Además, el hardware del momento era muy limitado. Las computadoras con mejor desempeño eran capaces de realizar alrededor de 10 MFLOPS. De esta manera, entrenar una red neuronal para el reconocimiento de manuscritos podía llegar a demorar semanas. Por último, el software utilizado para los desarrollos debía ser construido desde cero, ya que no existían herramientas como Matlab o Python, ni tampoco la gran cantidad de librerías de código abierto que existen hoy, como por ejemplo TensorFlow o Pytorch, ya que en ese momento no era común para las compañías liberar el código de sus desarrollos.

En el año 2003, Geoffrey Hinton de la Universidad de Toronto, Yoshua Bengio de la Universidad de Montréal y Yann LeCun de la Universidad de Nueva York llevaron a cabo una serie de proyectos para revivir el interés de la comunidad en las redes neuronales. Para el año 2009, grupos de investigación en Microsoft, Google y IBM consiguieron grandes logros en la reducción de errores en el reconocimiento de voz al reemplazar modelos basados en Gaussian Mixture Models por redes neuronales profundas. Al cabo de unos meses, esos sistemas ya eran usados comercialmente.

Al mismo tiempo, las redes convolucionales estaban produciendo resultados récord en las tareas de reconocimiento en imágenes, como la segmentación semántica y la detección

de peatones, pero los resultados solían ser ignorados por la comunidad. Entonces, a finales de 2012, Alex Krizhevsky, trabajando en los laboratorios de Geoffrey Hinton, desarrolló una implementación eficiente de redes convolucionales en GPUs. Si bien no fue el primero en hacerlo, su implementación era eficiente incluso para redes muy profundas, lo que le permitió al equipo ganar la competencia de ImageNet, reduciendo el error top-5 de 25 % a 16 %. Ese momento fue un gran logro y tuvo gran repercusión en la comunidad. Luego de dos años, el equipo liberó el código fuente para que cualquiera pueda usar el modelo de redes convolucionales. Esto demuestra que la disponibilidad de hardware de gran desempeño y software de código abierto es crítico para impulsar la adopción y el desarrollo de nuevas técnicas.

6.2. Requerimientos de Hardware

Existen cuatro casos de uso con diferentes requerimientos de hardware: (1) La investigación y desarrollo de Deep Learning; (2) El entrenamiento offline de modelos para producción; (3) La inferencia en servidores; (4) La inferencia en dispositivos móviles y sistemas embebidos.

Investigación y desarrollo de Deep Learning

Para este caso, se requieren computadoras de alta performance del tipo HPC (High Performance Computer), con múltiples GPUs y precisión de 32-bits en las operaciones de punto flotante. Las redes de comunicación deben ser de banda ancha y baja latencia para permitir la paralelización en el entrenamiento de grandes modelos con grandes datasets. En este caso el desempeño y la flexibilidad del sistema son más importantes que el precio y el consumo de energía.

Entrenamiento offline de modelos para producción

Una vez que la arquitectura de un modelo ha sido probada, se debe optimizar reentrenandola con nuevos datos. Este tipo de tareas es posible hacerlas en hardware especializado pero con precisión reducida, como por ejemplo GPUs de NVIDIA o las TPU de Google.

Inferencia en servidores

Gran parte de la carga de trabajo de los sistemas de Deep Learning es realizada en centros de datos. En empresas como Facebook, gran cantidad de datos pasan a través de redes convolucionales para el procesamiento de imágenes, video y voz, así como para la traducción de idiomas. Cada foto pasa a través de redes convolucionales a los dos segundos de ser subida por un usuario. Una de las redes sirve para etiquetar las imágenes y filtrar contenido no deseado como por ejemplo escenas de violencia. Otra red realiza el reconocimiento de caracteres y de rostros en la imagen, además de crear una descripción automática para las personas no videntes, entre otras tareas. Por otra parte, en los servicios de comunicación, como las videollamadas, redes convolucionales son utilizadas a gran escala para realizar tareas de reconocimiento, subtitulado de los discursos y traducción de idiomas, todo en tiempo real y con baja latencia. Para todos estos casos de uso, el consumo de energía y el costo de los sistemas es muy importante.

Inferencia en dispositivos móviles y sistemas embebidos

El éxito de los smartphones sumado a la aparición de nuevos dispositivos con tecnología de realidad aumentada y realidad virtual generan una gran demanda de aceleradores de inferencia de bajo consumo energético. Las tareas en tiempo real requieren que los sistemas de Deep Learning corran en los dispositivos para así evitar la latencia que conlleva enviar los datos a servidores. Algunas de estas tareas son la reconstrucción 3D para realidad aumentada, el seguimiento y reconocimiento de objetos, traductores de idiomas en tiempo real y asistentes virtuales.

6.3. Tipos de procesadores

Hoy en día, muchas de las aplicaciones están alojadas en servicios en la nube por compañías como Amazon, Google o Microsoft. Estas compañías corren sus servicios en grandes centros de datos equipados con miles de servidores, cada uno con varios procesadores, ya sean de propósito general o dedicados. El entrenamiento y la inferencia de los modelos de Deep Learning requiere el procesamiento de gran cantidad de datos. Es por eso que muchas empresas dotan sus servidores con hardware más capaz que simples CPUs, como pueden ser las GPUs, e incluso algunas desarrollan sus propios procesadores [49]. Ejemplos de esto último pueden ser Microsoft con la utilización de FPGAs y Google con el desarrollo de los TPU (Tensor Processing Unit).

CPU

Actualmente, la gran mayoría de los modelos de Deep Learning trabajan sobre GPUs, y el uso de CPUs se restringe solo para el preprocesamiento de los datos. Las CPUs son extremadamente flexibles en cuanto a su programación y versatilidad. Intel es la empresa predominante en el sector de las CPUs sobre otras como AMD, IBM y ARM. Los procesadores Intel Xeon y Xeon Scalable son algunos ejemplos de CPUs especializadas para centros de datos.

GPU

Los procesadores gráficos son los más utilizados para trabajar con Machine Learning y redes neuronales. Están especialmente diseñados para realizar procesos en paralelo y con un gran ancho de banda para el acceso a memoria.

Las redes neuronales requieren ejecutar grandes cantidades de multiplicaciones y convoluciones sobre matrices, operaciones para las cuales las GPUs proveen una forma eficiente en cuanto a consumo energético y costos. Este tipo de procesadores son recomendados especialmente para el entrenamiento de los modelos. Compañías como Nvidia, que acapara la mayor porción del mercado de GPUs, posee entre sus productos líneas como la Titan RTX que en su diseño hacen foco en el desarrollo de inteligencia artificial o los sistemas DGX especializados en centros de datos de gran escala.

Además, Nvidia posee un conjunto de herramientas que facilitan el desarrollo de sistemas de computación en paralelo llamada CUDA.

FPGA

Las FPGA (Field Programmable Gate Arrays) son un tipo de hardware que puede ser programado y configurado mediante el uso de un lenguaje de descripción de hardware (HDL) como por ejemplo VHDL o Verilog. Compañías como Microsoft y Baidu llevan la delantera en el desarrollo de este tipo de soluciones. Si bien las FPGAs no brindan el mejor desempeño en el procesamiento de operaciones de punto flotante, la relación entre performance y consumo de energía sí es superior a las GPUs. Además la versatilidad de las FPGAs al poder ser reconfiguradas, las hacen una solución muy atractiva para las empresas que buscan construir centros de datos de gran escala.

ASIC

Las ASICs son la opción con mejor desempeño pero al mismo tiempo la menos flexible. Poseen un diseño totalmente dedicado a una función en específico. Esto hace que la performance sea mucho más alta que con otro tipo de procesadores. La principal desventaja que tienen es que requiere una gran inversión inicial ya que el costo del proceso de manufactura es muy elevado. Además, son diseñadas para un propósito en específico, lo que le quita flexibilidad en comparación con las CPUs y GPUs que son de propósito general. Este tipo de procesadores pueden ser diseñados tanto para el entrenamiento como para la inferencia. El mejor ejemplo de ASICs son las TPUs desarrolladas por Google. Otros casos son los productos Intel Nervana e Intel Movidius.

TPU

En 2015, Google sacó a la luz un tipo de procesador especializado en Machine Learning y aplicaciones de inteligencia artificial llamados Unidades de Procesamiento Tensorial (TPU). El principal foco fue en la tarea de inferencia, donde las TPU originales trabajaban con un tamaño de dato de 8 bits, y soporta código escrito con la librería TensorFlow, lo que hace muy sencillo su uso. El público general tiene acceso a estos procesadores a través de los servicios de Google Cloud.

6.4. Servicios en la nube

Gracias a la computación en la nube, un modelo centrado en el producto se transformó en un modelo centrado en el servicio, global y distribuido [50] lo que llevó a un cambio de paradigma donde antes se veía a la computación como un producto y ahora como un servicio. Desde su aparición masiva en 2007, la computación en la nube ha cambiado la manera en la cual los servicios de IT se inventan, desarrollan, escalan, actualizan, mantienen y pagan.

Los servicios en la nube permiten a las personas y organizaciones acceder a recursos de computación bajo demanda, en cualquier momento y desde cualquier dispositivo. Además, se reduce la barrera de entrada a la computación de alto rendimiento, permitiendo a organizaciones aprovechar la potencia informática que no tienen el capital suficiente para adquirir.

La computación en la nube también proporciona la infraestructura que ha impulsado las principales tendencias digitales, como la computación móvil, el internet de las cosas (IoT), big data y la inteligencia artificial. De esta forma, se ha convertido en un elemento

crítico para casi todos los aspectos de la vida cotidiana y continuará transformando el mundo en el que vivimos en múltiples formas y maneras.

La computación en la nube tiene sus bases en el avance de múltiples tecnologías. En el área del hardware, la virtualización y los procesadores de múltiples núcleos son dos claros ejemplos. Además, los avances en la tecnología de internet permitió el surgimiento de los servicios web y la arquitectura orientada a servicios. Por último, la computación distribuida abrió muchas puertas con tecnologías como los clusters o la computación en malla. Por otra parte, las mejoras en la administración de sistemas, por ejemplo la automatización de los centros de datos, abrió la posibilidad de escalar los sistemas de computación en la nube de forma radical [51].

La idea de ofrecer cómputo como servicio a través de la red se remonta a finales de la década de 1960 y se convirtió en una fuerza impulsora del desarrollo temprano de internet. A finales de la década de 1990, la industria de las telecomunicaciones experimentó una gran inversión que, junto con la creación de nuevas organizaciones, impulsaron un gran aumento de las redes de fibra óptica a nivel mundial. Estos cambios redujeron drásticamente la latencia de la red y los costes relacionados. Las mejoras fueron acompañadas con la aparición de tecnologías y técnicas para coordinar la provisión a gran escala y bajo demanda de recursos informáticos, lograda, por ejemplo, a partir de las innovaciones en torno a la computación en red, la computación de servicios y la virtualización del hardware. Hoy en día, la computación en red se utiliza especialmente para resolver problemas de investigación que requieren muchos recursos.

Además de los avances en redes, tecnologías y técnicas, organizaciones como Alphabet (la empresa matriz de Google), Amazon y Microsoft crearon grandes instalaciones para transferir la capacidad de cómputo de las computadoras individuales y la infraestructura IT privada a grandes centros de datos externos y públicos a los que se puede acceder a través de internet. Los servicios provistos por estos centros de datos posteriormente se denominaron computación en la nube.

La evolución de las arquitecturas distribuidas y orientadas a los servicios para proporcionar recursos de cómputo a demanda a través de internet y las grandes mejoras de las infraestructura tecnológica, incluidas las redes y los centros de datos, han permitido finalmente el paso a la nube. La nube provee computación mediante grandes conjuntos de recursos informáticos automatizados y escalables, y aplicaciones relacionadas.

6.4.1. Cloud computing y sus características principales

De acuerdo con la definición propuesta por el NIST (National Institute of Standards and Technology), la computación en la nube es un modelo que otorga acceso ubicuo, conveniente y bajo demanda a recursos computacionales de uso compartido que pueden ser provistos rápidamente en cualquier momento y en cualquier lugar con acceso a Internet o a una red. Entre los recursos de computación que se pueden acceder se encuentran el acceso a redes, servidores, almacenamiento y aplicaciones. Los servicios en la nube pueden ser implementados con mínimo esfuerzo administrativo, poca interacción con el proveedor del servicio en la nube y ser configurados de acuerdo a las necesidades del usuario.

A continuación se describirán algunas de las características más importantes de los servicios de computación en la nube.

Modelo basado en servicios

Todas las ofertas de los proveedores pueden expresarse como un servicio basados en un contrato en donde se define su función y los parámetros de calidad que posee.

Acceso bajo demanda

Los usuarios pueden acceder a los servicios de computación de forma independiente e inmediata. Esto se logra gracias a la automatización de los centros de datos, reduciendo drásticamente la interacción humana en la configuración.

Acceso ubicuo

Los servicios de computación son provistos sobre la red de Internet mediante el uso de estándares de comunicación ampliamente aceptados por la comunidad. Esto permite su uso desde distintos dispositivos y plataformas, como por ejemplo dispositivos móviles, tabletas o laptops.

Multi-usuario

Los recursos físicos y virtuales son asignados y reasignados dinámicamente bajo demanda a diferentes usuarios. Esto permite que los recursos se utilicen de forma más eficiente haciendo que el uso de computación en la nube sea económicamente beneficioso ya que se reducen los costos.

Elasticidad

Muchas de las organizaciones planifican los requerimientos de recursos computacionales teniendo en cuenta la capacidad alcanzada en los momentos de carga máxima. Esto causa que gran parte del tiempo tengan sus recursos sin utilizar en su totalidad. En contraparte, los servicios en la nube poseen la flexibilidad y adaptabilidad necesaria, en muchos casos de forma autónoma, para que los recursos utilizados o reservados por una organización coincidan solo con la demanda a través del tiempo. Por lo tanto, una organización consume solo los recursos que necesita en cada momento.

Pago por uso

En los servicios en la nube, los consumos son medidos y facturados dependiendo el tiempo que un cliente utilice un recurso, o utilizando alguna medida de capacidad. Este modelo de facturación se conoce como pago por uso. Al implementar este modelo, la computación en la nube elimina las barreras de entrada impuestas por la gran inversión inicial que se requiere para montar los mismos recursos de forma on premise, es decir, de forma local en una organización.

6.4.2. Modelos de computación en la nube

Los servicios de computación en la nube suelen clasificarse en tres modelos organizados jerárquicamente dependiendo del nivel de abstracción de las capacidades provistas [50]. Los tres modelos son Infraestructura como servicio (IaaS), Plataforma como servicio (PaaS), y Aplicación como servicio (SaaS).

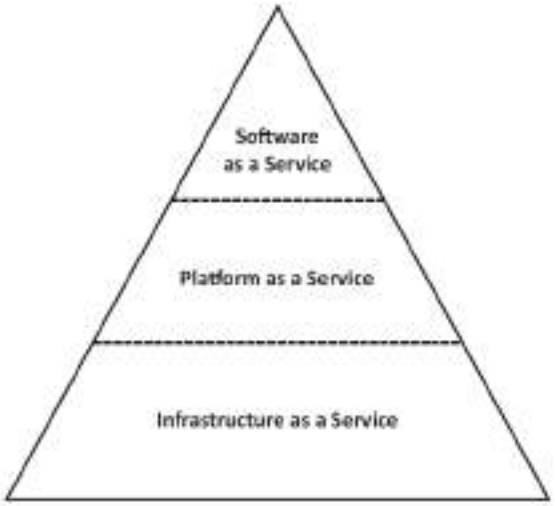


Fig. 6.1: Organización jerárquica de los modelos de servicio

6.4.2.1. Infraestructura como servicio (IaaS)

A través de la infraestructura como servicio, los usuarios pueden acceder a capacidad de procesamiento, almacenamiento, redes y otros recursos computacionales fundamentales. Se pueden utilizar estos recursos implementando y corriendo software administrado por el mismo usuario, como un sistema operativo y/o aplicaciones. El proveedor típicamente provee estos servicios dividiendo su gran infraestructura física en recursos virtuales más pequeños los cuales son accedidos por los usuarios. Los clientes no administran ni controlan la infraestructura donde subyacen los servicios, pero si controlan los sistemas operativos, el almacenamiento y las aplicaciones.

Los usuarios poseen privilegios suficientes para realizar varias tareas sobre las máquinas virtuales, como por ejemplo prenderlas y apagarlas, instalar software, configurar permisos de acceso y configurar las reglas de los firewalls.

6.4.2.2. Plataforma como servicio (PaaS)

En este caso, los usuarios pueden implementar aplicaciones adquiridas a terceros o desarrolladas por ellos mismos utilizando lenguajes de programación, librerías, servicios y herramientas soportadas por el proveedor del servicio. Los usuarios no controlan la infraestructura donde corren estas aplicaciones, pero si controlan el despliegue del software y las configuraciones de los ambientes donde se ejecutan.

PaaS ofrece un entorno altamente integrado donde los desarrolladores pueden diseñar, desarrollar, probar y desplegar aplicaciones sin el costo ni la complejidad de comprar y administrar el hardware y software donde se ejecutan.

6.4.2.3. Aplicación como servicio (SaaS)

En el modelo de aplicación como servicio los usuarios utilizan aplicaciones creadas por los mismos proveedores. Frecuentemente pueden ser accedidas desde diferentes dispositivos a través de una interfaz de usuario como puede ser un navegador web o una aplicación de escritorio. El usuario no administra ni controla la infraestructura en la nube

y está totalmente desligado del mantenimiento del software. Debido a esto, los usuarios están migrando rápidamente de programas instalados en computadoras de forma local a aplicaciones en línea con las mismas funcionalidades pero sin tener que lidiar con su mantenimiento. Ejemplos de SaaS pueden ser las aplicaciones de correo electrónico, procesadores de texto y hojas de cálculo.

6.4.3. Tecnologías esenciales para Cloud Computing

La computación en la nube utiliza un conjunto de tecnologías claves para lograr las características de los servicios que brinda.

Redes de banda ancha e Internet

Internet permite a los usuarios el acceso remoto a los recursos alojados en los centros de datos del proveedor. La necesidad del cloud computing de acceso a internet genera una dependencia del servicio a Internet o a una red privada.

Centros de datos

Los centros de datos contienen múltiples tecnologías y componentes conectados entre sí. Estos centros están diseñados y construidos con hardware estandarizado y con arquitectura modular haciendo que sean altamente escalables al mismo tiempo que reduce los costos de inversión.

La automatización de los centros de datos es cada día más importante, pudiéndose mejorar tareas repetitivas de administración, monitoreo y mantenimiento en los centros de operaciones que, al no ser llevadas a cabo de forma manual por operadores humanos, se incrementa la eficiencia y se reducen los costos.

Tecnología de virtualización

La virtualización utiliza el software para imitar las características del hardware y crear un sistema informático virtual. Entre los recursos que se pueden virtualizar podemos encontrar servidores, almacenamiento, redes y procesamiento. Esta tecnología oculta las características físicas de los recursos y, en cambio, presenta una abstracción del hardware mediante una máquina virtual emulada. Un hipervisor, conocido también como monitor de máquinas virtuales, administra todos los recursos virtualizados en un servidor host y posee la capacidad de coordinación del sistema.

Las máquinas virtuales se comportan como un sistema independiente, pero a diferencia de los sistemas físicos pueden ser iniciados, apagados, y configurados bajo demanda. Además, estos entornos pueden ser mantenidos y replicados de forma sencilla.

Contenedores

La tecnología de contenedores es un método de empaquetar una aplicación junto con sus dependencias para que pueda ser ejecutada de forma aislada de otros procesos independientemente del ambiente en el que esté corriendo. Puede verse como un concepto similar pero más liviano que la virtualización. Los contenedores consumen menos recursos que las máquinas virtuales, presentándose como una gran alternativa para mejorar la interoperabilidad y administración de las aplicaciones en la nube.

Los contenedores ofrecen un mecanismo de empaquetado en el cual se puede abstraer la aplicación y sus dependencias del ambiente en el que corren. Este desacoplamiento permite que el despliegue de aplicaciones basadas en contenedores sea muy sencillo, independientemente de que el ambiente sea un servidor privado o un servicio de computación en la nube.

Además, esta tecnología genera que la división de responsabilidades en un equipo de trabajo sea clara. Mientras los desarrolladores se enfocan en la lógica de las aplicaciones, el equipo de operaciones se encarga del despliegue y la administración de los contenedores, sin tener que preocuparse por los detalles de la aplicación.

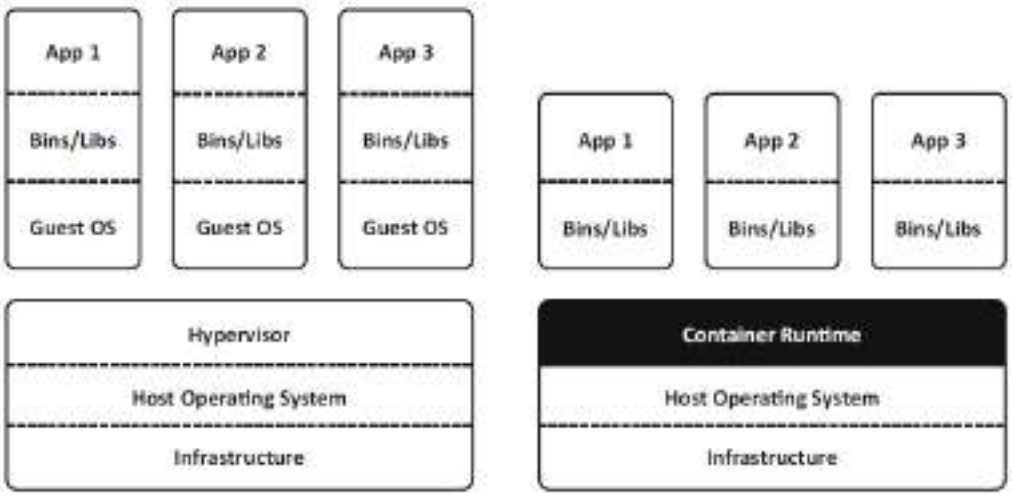


Fig. 6.2: Comparación entre las tecnologías de máquinas virtuales y contenedores

6.4.4. Beneficios de la computación en la nube

La computación en la nube le permite a individuos y organizaciones acceder a recursos de computación bajo demanda, desde cualquier dispositivo y en cualquier momento. Debido a sus características, presenta diversos beneficios y oportunidades. De forma notable, se presenta como un mecanismo de transformación que sirve como herramienta para el desarrollo de servicios y modelos de negocios innovadores, beneficiando a individuos, organizaciones y sociedades. A continuación, se describen algunas de las características que más peso tienen en la toma de decisiones de organizaciones para migrar o comenzar a utilizar tecnologías en la nube.

Baja barrera de entrada

Uno de los beneficios más grandes del cloud computing es la baja barrera de entrada a grandes centros de datos que antes eran inaccesibles para muchas organizaciones debido a la gran inversión inicial necesaria. Por esta razón, esta tecnología le permite a pequeñas y medianas organizaciones entrar en nuevos mercados de forma rápida.

Pago por uso

Debido al modelo de pago por uso, los usuarios pueden optimizar el uso de recursos de IT transformando costos fijos en costos variables. Esto afecta al flujo de caja de los usuarios

logrando flujos más pequeños y estables en vez de grandes flujos de dinero consecuencia de la compra de equipamiento, el pago de licencias, el mantenimiento y las actualizaciones.

Ahorro de costos

Debido a la especialización de los proveedores de servicios y a la economía de escala, se consigue que las plataformas sean muy eficientes en cuanto a costos. Este beneficio también recae sobre los usuarios, que acceden a los recursos de forma más económica.

Flexibilidad y elasticidad

Usando servicios en la nube se consigue un mayor grado de flexibilidad debido a la utilización de recursos de IT altamente escalables y provistos bajo demanda. Los usuarios pueden rápidamente aumentar o reducir la cantidad de recursos utilizados y solo pagar por lo que realmente usan. Esta flexibilidad hace que las organizaciones puedan adaptarse fácilmente y dar respuesta a la volatilidad de sus negocios.

Time-to-Market reducido

Como los servicios son brindados por proveedores externos, ofreciendo acceso casi inmediato e ilimitado a recursos de hardware sin requerir una gran inversión inicial, los usuario pueden desplegar sus productos al mercado velozmente.

6.4.5. Desafíos y riesgos

Aunque existen muchos beneficios de la computación en la nube, no se libra de tener sus propios riesgos y desafíos a enfrentar. Está tecnología, a diferencia de los demás productos y servicios, introduce el sentimiento de incertidumbre en la relación entre el proveedor del servicio y el usuario. A continuación se describirán algunos de los puntos a tener en consideración al momento de evaluar el uso de tecnologías en la nube.

Recursos compartidos

Aunque alquilar solo una porción virtualizada de un centro de datos hace que los usuario puedan acceder a precios increíblemente bajos, la seguridad y privacidad de estos sistemas deben tomarse en consideración. Lo más recomendable es realizar los controles de seguridad apropiados dependiendo la capa de servicio a consumir. Entre estos controles puede estar la utilización de sistemas de detección de intrusos y el empleo de firewalls.

Pérdida de control

Los modelos de computación tradicionales le han permitido a los usuarios gran capacidad de control sobre los recursos. La tecnología en la nube implica una pérdida de poder sobre la administración de los sistemas, así como también sobre el control operacional y de seguridad.

Exclusividad sobre proveedores

Al usar tecnologías cloud existe cierta incertidumbre sobre qué tan fácil es migrar de un proveedor a otro sin incurrir en pérdidas económicas o sociales. Estas pérdidas se definen como el costo de transferencia, y se refiere al costo asociado con el proceso de cambiar entre dos proveedores.

Servicio continuo limitado

Otra preocupación suele ser la incertidumbre sobre si un servicio o funcionalidad en particular permanecerá disponible en el mercado en el mediano y largo plazo.

6.4.6. Proveedores de servicios

Actualmente, existe una gran cantidad de proveedores de servicios en la nube, siendo Amazon Web Services, Microsoft Azure y Google Cloud Platform los que dominan el mercado. Si bien estos tres proveedores acaparan cerca del 60 % del mercado, existen otras empresas que poseen un rol importante como IBM, Salesforce, Alibaba o Baidu.

6.4.6.1. Amazon Web Services

Amazon Web Services (AWS) es una plataforma de servicios en la nube que ofrece más de 200 servicios integrales de centros de datos a nivel global, destinados a una amplia variedad de tecnologías, sectores y casos de uso. [52]

Dentro del catálogo de productos ofrecidos por AWS se encuentra Amazon SageMaker, una plataforma especializada para crear, diseñar, entrenar y ajustar modelos de Machine Learning. Amazon SageMaker soporta los frameworks, lenguajes de programación y herramientas líderes en el aprendizaje automático, entre los que se encuentran, Jupyter Notebooks, TensorFlow, PyTorch, Apache MXNet, Python, R, entre otros. Estos frameworks son automáticamente configurados y optimizados para obtener un alto rendimiento.

Amazon SageMaker Studio es el ambiente de desarrollo integrado (IDE) provisto por Amazon que proporciona una interfaz visual basada en la web donde poder realizar todos los pasos del desarrollo de aprendizaje automático. SageMaker Studio ofrece acceso, control y visibilidad completa en cada paso necesario para crear, entrenar e implementar modelos. Permite cargar rápidamente los datos, crear nuevos notebooks, entrenar y ajustar los modelos, avanzar y retroceder entre los pasos para ajustar experimentos. [53]

SageMaker ofrece un nivel de prueba gratuito que permite utilizar 250 horas en los notebooks de Studio de la instancia ml.t3.medium o ml.t2.medium, ambas compuestas por 2 CPU virtuales y 4 GB de memoria, y 50 horas de las instancias m4.xlarge o m5.xlarge, compuestas por 4 CPU virtuales y 16 GB de memoria, por mes, durante los primeros dos meses.

Dentro de la capa con costo hay una gran cantidad de instancias diferentes con distintas cantidades de CPUs virtuales y memoria que pueden ser contratadas en función de las necesidades del usuario. Los precios varían desde 0,05 USD la hora en la instancia más simple, hasta más de 7 USD la hora. [54]

6.4.6.2. Microsoft Azure

Microsoft Azure es el conjunto de más de 200 servicios en la nube ofrecidos por la empresa Microsoft. Con Azure es posible almacenar información y crear, administrar e implementar aplicaciones en la nube. Proporciona software, plataforma e infraestructura como servicio, y es compatible con muchos lenguajes, herramientas y frameworks de programación diferentes, incluidos software y sistemas de Microsoft y de terceros.

Entre la gran variedad de servicios que posee, se encuentra Azure AI [55], un portafolio de servicios enfocados en la inteligencia artificial y en la ciencia de datos. Mediante llamadas a su API se puede acceder a modelos de visión computacional, reconocimiento de voz, traducción de idiomas y modelos para la toma de decisiones. Además, permite crear modelos de aprendizaje automático con herramientas como Jupyter Notebooks, Visual Studio Code y frameworks de código abierto como TensorFlow y PyTorch.

Uno de los servicios más destacados es Machine Learning Studio [56], un ambiente de desarrollo integrado para crear y desplegar flujos de trabajo de Machine Learning. Este servicio se ofrece en dos tipos: gratuito y estándar. La mayor diferencia es que en el gratuito la duración de un experimento puede ser de hasta una hora, mientras que en el estándar puede durar hasta siete días.

6.4.6.3. Google Cloud Platform

Otro jugador importante en el área del cloud computing es Google con su conjunto de servicios ofrecidos bajo el nombre de Google Cloud Platform (GCP). Esta plataforma incluye herramientas para el almacenamiento, computación y desarrollo de aplicaciones.

Al igual que sus competidores, Google Cloud Platform posee servicios orientados al mundo de la inteligencia artificial. Entre estos servicios se destaca Vertex AI [57], una plataforma unificada de aprendizaje automático que permite crear, desplegar y escalar modelos de forma eficiente.

Otra herramienta ofrecida por Google es Google Colab [58]. En este caso no es un servicio dentro de GCP sino una aplicación como servicio (SaaS) ofrecida por la división Google Research. Aun así, corre sobre la infraestructura de GCP.

Google Colab es un servicio alojado de Jupyter notebook, en otras palabras, permite escribir código en el lenguaje de programación Python directamente en un navegador web y ejecutarlo en los servidores de Google. Este servicio es de gran utilidad en el desarrollo de modelos de aprendizaje automático ya que le permite a los desarrolladores acceder a recursos de computación potentes de forma sencilla.

Google Colab posee tres niveles de suscripción. En la versión gratuita, que no requiere suscripción, el acceso a GPUs y TPUs está muy restringido y los límites de uso son muy estrictos. Si bien Google asigna los recursos dependiendo de la demanda del momento, en la capa gratuita es común encontrar una GPU Nvidia Tesla K80 y 16 GB de RAM. Si se requiere más poder de cómputo, el segundo nivel de suscripción se denomina Colab Pro. Esta versión ofrece acceso a GPUs Nvidia Tesla T4 y P100, considerablemente más poderosas que la encontrada en la versión gratuita, con 32 GB de RAM. Además, los tiempos de ejecución son mucho más extensos, ofreciendo instancias por 24 horas en comparación con las 12 horas máximas de la capa gratuita. Por último, podemos encontrar la suscripción más cara llamada Colab Pro+. En este caso los usuarios podrán acceder a 52 GB de RAM y un acceso prioritario a los recursos. A continuación se muestra la Tabla 6.1 comparando las tres suscripciones:

	Colab gratis	**Colab Pro**	**Colab Pro+**
Prioridad de asignación de recursos	Baja	Alta	Muy Alta
GPU	K80	K80, T4 y P100	K80, T4 y P100
RAM	16 GB	32 GB	52 GB
Tiempo de ejecución	12 horas	24 horas	24 horas
Ejecución en segundo plano	No	No	Sí
Precio	US$ 0	US$ 10	US$ 50

Tab. 6.1: Características y precios de las distintas versiones de Google Colab

Capítulo 7

Detección de barcos y conjuntos de datos

7.1. Problematica detección de barcos

La detección de barcos es un problema que ha tomado relevancia en los últimos años debido a su gran utilidad para el control del tráfico, la vigilancia marítima y el control ambiental y de pesca. En Argentina la Prefectura Naval Argentina (PNA) tiene dentro de sus roles el control ambiental marítimo, fluvial y lacustre, garantizar la seguridad en la navegación, la protección marítima y ambiental y el monitoreo de las 200 millas náuticas de Mar Argentino a fin de detectar buques realizando actividades ilegales dentro de la Zona Económica Exclusiva de la Argentina (ZEEA).

La detección de barcos a partir de imágenes satelitales en conjunto con los sistemas tradicionales de posicionamiento AIS o LRIT y otros servicios propietarios, como el Sistema Guardacostas utilizado por la PNA, son de gran importancia para el cumplimiento de las leyes y reglamentaciones del Mar Argentino.

En los últimos años el rápido desarrollo de la tecnología de la obtención de imágenes por satélite y la mejora exponencial en los tiempos y resultados obtenidos por las redes neuronales convolucionales llevaron a la detección de barcos en imágenes satelitales utilizando aprendizaje profundo a ser una importante aplicación dentro del área del reconocimiento de objetos. Si bien hace más de 20 años se trabaja en esta área [59, 60], recién en los últimos 5-6 años tomó gran relevancia y se presentaron los primeros trabajos en utilizar aprendizaje profundo para la detección de barcos [61, 62, 63, 64].

7.2. Conjuntos de datos

Un dataset es una colección de datos generalmente almacenada en una tabla de una base de datos única donde cada columna de la tabla representa una variable o característica y cada fila un elemento o instancia del conjunto de datos.

En nuestro caso particular, los conjuntos de datos que utilizaremos están compuestos por imágenes, por lo tanto, tendremos tensores de tamaño H x W x (R, G, B), siendo H la altura de la imagen, W el ancho y (R, G, B) los tres canales de colores que tomarán valores entre 0 y 255. Debido a que estamos utilizando imágenes en blanco y negro, el valor de la componente en los tres canales de colores será el mismo.

El conjunto de datos es fundamental y necesario en cualquier proyecto de aprendizaje automático, a partir de ellos es que el modelo aprende durante la etapa de entrenamiento. La precisión que obtenga el modelo va a estar relacionada con las características del

Satelite	Modo de adquisición	Swath (km)	Angulo incidencia (°)	Polarizacion	Resolución (m)	Lugar	Imágenes SAR (num)	Imágenes 800 x 800 px (num)
Sentinel-1B	S3-SM	80	27.6 ∼ 34.8	HH	3	Puerto Houston	40	2269
Sentinel-1B	S3-SM	80	27.6 ∼ 34.8	HV	3	Puerto San Pablo	20	842
Sentinel-1B	S3-SM	80	27.6 ∼ 34.8	HH	3	Puerto San Pablo	21	909
TerraSAR-X	SM	30	20 ∼ 45	VV	3	Puerto Barcelona	23	513
Sentinel-1B	S3-SM	80	27.6 ∼ 34.8	VV	3	Puerto Chittagong	18	593
TerraSAR-X	ST	4	20 ∼ 60	HH	0.5	Presa de Asuán	2	32
TerraSAR-X	ST	4	20 ∼ 60	HH	0.5	Shanghai	2	25
TanDEM	HS	10	20 ∼ 55	HH	1	Canal de Panama	1	30
TerraSAR-X	HS	10	20 ∼ 55	VV	1	Puerto Visakhapatnam	1	54
TerraSAR-X	SM	30	20 ∼ 45	HH	3	Puerto Singapur	4	127
TerraSAR-X	SM	30	20 ∼ 45	HH	3	Estrecho de Gibraltar	2	90
TerraSAR-X	SM	30	20 ∼ 45	VV	3	Puerto de Azufre	1	7
TerraSAR-X	SM	30	20 ∼ 45	VV	3	Bahía de Plenty	1	4
Desconocido	-	-	-	-	-	-	1	109

Tab. 7.1: Información detallada de las imágenes SAR que componen HRSID

conjunto de datos que se utilice, por este motivo es que los conjuntos deben contar con algunas características deseables como presentar situaciones variadas y complejas, y estar equilibrados.

Al momento de entrenar una red neuronal convolucional se requiere de un conjunto de datos lo suficientemente grande y balanceado, con imágenes en diferentes contextos y características, para poder entrenar la red correctamente y validar su precisión en diferentes contextos, aun en aquellos que no estaban incluidos en el conjunto de datos. En el caso particular de las imágenes SAR, el alto costo de su obtención, la dificultad en la definición de los bounding boxes y el ser una aplicación novedosa que está creciendo en los últimos años, condujo a la dificultad de encontrar una base de datos grande, confiable y de buena calidad.

Existen muy pocos conjuntos de imágenes de libre acceso para la comunidad, a continuación se describirán los conjuntos de datos utilizados en este trabajo.

7.3. High Resolution SAR Images Dataset - HRSID

El conjunto de datos High Resolution SAR Images Dataset [65] fue publicado en 2020, contiene 5604 imágenes SAR con resolución entre 0,5 metros y 5 metros de 800 × 800 píxeles cada una. El dataset fue generado a partir de 136 imágenes SAR panorámicas de los satélites Sentinel 1-B, TerraSAR-X y TanDEM-X que operan en las bandas L y X del espectro electromagnético. Estas imágenes fueron obtenidas con diferentes modos de operación y polarizaciones. Para el satélite Sentinel 1-B se utilizó el modo S3 StripMap, para TerraSAR-X los modos Staring SpotLight (ST), High Resolution SpotLight (HS) y StripMap (SM), y para TanDEM-X el modo HS. Además de las 136 imágenes detalladas, el conjunto de datos contiene una imagen 137, de la cual no se proporciona información. El detalle de las 137 imágenes se puede observar en la tabla 7.1.

En general, la co-polarización tiene un valor más alto de coeficientes de retrodispersión en barcos y objetos en el océano que la polarización cruzada. Los barcos en las imágenes SAR de polarización cruzada son más brillantes respecto al fondo que las imágenes SAR co-polarizadas [66]. Cuando el mar se encuentra en calma las imágenes co-polarizadas presentan un mar relativamente más oscuro que en las imágenes de polarización cruzada debido a la reflexión especular del mar. En HRSID se seleccionaron 116 imágenes SAR co-polarizadas y 20 imágenes de polarización cruzada.

Las 5604 imágenes del conjunto contienen 16951 barcos etiquetados con el formato

Microsoft Common Objects in Context (MS COCO) [39]. En promedio cada imagen cuenta con 3,02 barcos. El 54,4 % de los barcos son pequeños con un bounding box menor a 32x32 píxeles, el 43,5 % son medianos con bounding box entre 32x32 y 96x96 píxeles, y el 2,1 % son grandes con bounding box mayor a 96x96 px.

HRSID está dividido en los conjuntos de entrenamiento, con un 65 % de las imágenes, y conjunto de prueba, con un 35 % de las imágenes. El área del bounding boxes y la relación de aspecto de los mismos fueron dos de las variables consideradas al momento de realizar la división del dataset. Además de la división en entrenamiento y prueba, otra división es presentada por los desarrolladores. Teniendo en cuenta el contexto las imágenes fueron divididas en dos grupos, mar adentro y en la costa. Un 18,4 % de las imágenes son obtenidas en la costa y el 81,6 % restante son imágenes mar adentro. Las imágenes mar adentro, donde no hay interferencia de otros objetos, son más fáciles de detectar correctamente para los modelos.

Tamaño barco	**Tamaño bounding box (px)**	**Barcos (num)**	**% del total**
Pequeño	bbox <32 x 32	9442	54,4 %
Mediano	32 x 32 <bbox <96 x 96	7388	43,5 %
Grande	96 x 96 <bbox	321	2,1 %

Tab. 7.2: Distribución del tamaño de los bounding boxes en HRSID

7.4. SAR Ship Detection Dataset (SSDD)

El conjunto de datos SAR Ship Detection Dataset [61] publicado en 2017 contiene 1160 imágenes SAR con resoluciones variables entre 1 y 15 metros y tamaño entre 190 y 500 píxeles en alto y ancho. Los satélites utilizados fueron RadarSat-2, TerraSAR-X y Sentinel-1 en los cuatro tipos de polarización HH, HV, VV y VH. Las 1160 imágenes contienen en total 2456 barcos, obteniendo un promedio de 2,12 barcos por imagen.

Este dataset tuvo una gran aceptación y fue ampliamente utilizado en diferentes arquitecturas y modelos, pero SSDD tiene múltiples problemas, como la baja cantidad de imágenes, lo que dificulta la capacidad de generalizar del modelo, o la repetición de escenas y errores en la anotación.

A pesar de estos errores fue muy utilizado y esto se debe a que fue el primer dataset público de imágenes SAR de barcos. Durante aproximadamente un año y medio fue el único dataset disponible, hasta la publicación de SAR-Ship-Dataset [67] en marzo del 2019, por lo que sentó las bases del Deep Learning aplicado a la detección de barcos en imágenes SAR.

7.5. Nuestro dataset

Para poner a prueba el modelo entrenado en un entorno real decidimos crear nuestros propios datasets. Para esto, accedimos a imágenes provenientes de los satélites Sentinel 1-A y 1-B, y del satélite SAOCOM 1A. Luego le aplicamos un conjunto de transformaciones a los datos para obtener las imágenes finales que el modelo de detección de barcos admite como entrada.

Las imágenes de nuestro dataset son de las siguientes ubicaciones: Puerto de New Orleans, Canal de Suez, Puerto de Chittagong en Bangladesh, y Puerto de Mar del Plata en Argentina.

Para obtener las imágenes finales listas para alimentar el modelo se realizan tres pasos. Primero se descargan los archivos desde la plataforma en línea del programa Copernicus perteneciente a la ESA o desde el portal de la CONAE dependiendo si los datos son de los satélites Sentinel o del SAOCOM respectivamente. Luego, obtenemos subdivisiones de la escena en formato geotiff. Por último, transformamos la imagen geotiff en una imagen en escala de grises y en el formato correspondiente compatible con el modelo. A continuación detallaremos cada uno de los pasos.

7.5.1. Acceso a datos

A través del portal Copernicus Open Access Hub [68], la Agencia Espacial Europea provee de acceso libre y gratuito a los productos de los satélites Sentinel 1, Sentinel 2, Sentinel 3 y Sentinel 5P. En nuestro caso, solo los productos del Sentinel 1 son de interés. Por su parte, la NASA también provee datos de libre acceso desde centros de almacenamiento distribuidos en distintos puntos de Estados Unidos. Uno de estos centros es el Alaska Satellite Facility (ASK) [69], especializado en almacenar datos SAR. A través de la plataforma Vertex del ASK, se tiene acceso a una interfaz de usuario, similar al portal de la ESA, para buscar y descargar datos. La razón por la que también usamos Vertex del ASK es porque en el portal de Copernicus los datos dejan de estar disponibles al cabo de unos meses. En cambio, en ASK los archivos perduran por más tiempo.

Todas las escenas de las ubicaciones nombradas anteriormente se obtuvieron en la plataforma Copernicus Open Access Hub, salvo las imágenes del Canal de Suez que tuvimos que acceder a través del ASK. Esto es porque buscamos específicamente capturas de finales de marzo del 2021, momento en que se produjo el bloqueo al canal por parte del barco Ever Given, desencadenando un gran embotellamiento de barcos en la zona. Con respecto a las tomas de SAOCOM, todas fueron accedidas desde el portal de la CONAE.

Los productos de la misión Sentinel 1 siguen una convención en el nombre de las carpetas que almacenan la información. Este nombre debe estar compuesto por caracteres alfanuméricos y en mayúsculas, y siempre tienen la extensión “SAFE”. Utilizando un guión bajo como separación, el nombre posee la siguiente información:

- El identificador de la misión (MMM) nos dice que satélite realizó la captura, puede ser S1A o S1B.
- El modo o Beam (BB) indica el modo de captura. Puede ser S1-S6 para el caso de Strip Mode IW, EW o WV.
- El tipo de producto (TTT) puede ser RAW, SLC, GRD o OCN.
- Resolución (R) puede ser F (full resolution), H (high resolution) o M (medium resolution). Este parámetro solo aplica para productos del tipo GRD.
- El nivel de procesamiento (L) puede tomar los valores 0, 1 o 2.
- La clase de producto (F) puede ser Standard (S) o Annotation (A).
- Polarización (PP) puede tomar los valores:

- SH para polarización simple HH
- SV para polarización simple VV
- DH para polarización dual HH+HV
- DV para polarización dual VV+VH

- La fecha y hora del comienzo y la finalización del producto se representa con 14 dígitos separados por una T.
- El número de órbita absoluto (OOOOOO) al comienzo del producto.
- El identificador del data-take de la misión (DDDDDD).
- El identificador único del producto (CCCC).

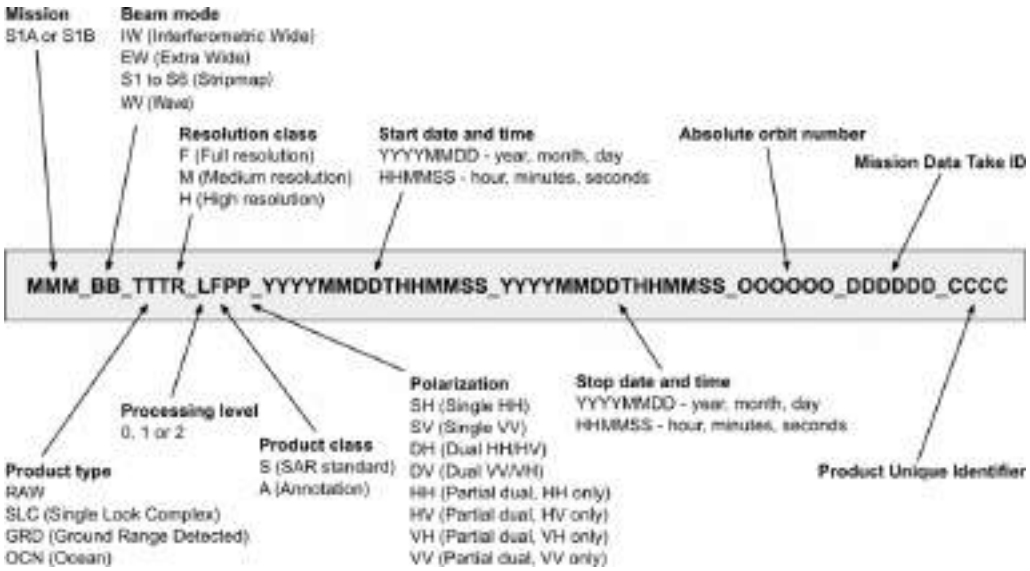


Fig. 7.1: Estructura del nombre de los productos de la misión Sentinel 1.

Entre el contenido de esta carpeta se encuentra un archivo manifest con metadata del producto en formato XML y subcarpetas con los datos del sensado, las anotaciones, vistas previas y archivos de soporte.

7.5.2. Subdivisión de la escena

Si bien no todas las capturas son iguales, el tamaño de una imagen proveniente del Sentinel 1 es en torno a los 15.000 x 25.000 píxeles. Estas magnitudes distan bastante de los 608 x 608 ideales para el modelo de detección de objetos. Por lo tanto, debemos dividirlas en imágenes más pequeñas. Además, el producto original contiene información de la intensidad y amplitud de la señal sensada pero como entrada al modelo debemos proporcionar una imagen en escala de grises, por esto, también se requiere preprocesamiento para obtener la imagen final.

Para llevar a cabo este procesamiento utilizamos el software libre SNAP [70]. Desarrollado por la ESA, contiene todas las herramientas necesarias para trabajar con imágenes

satelitales, especialmente las generadas por satélites Sentinel. Este software está implementado en el lenguaje de programación Java y se puede utilizar a través de su API. Si se desea utilizar Python como lenguaje de desarrollo, SNAP provee una librería de código abierto llamada Snappy que realiza la interfaz con la API en Java. Por último, si no se quiere trabajar por código, se puede utilizar la interfaz de usuario (GUI) que trae incorporada.

En nuestro caso utilizamos la librería Snappy para realizar los siguientes pasos [71]:

1. Subdividir la imagen total en imágenes de 608 píxeles de alto y ancho.
2. Remover el ruido térmico de la escena.
3. Realizar una calibración radiométrica al producto.
4. Realizar correcciones del terreno.

Luego, las imágenes resultantes son almacenadas en formato GeoTIFF.

7.5.3. Creación de la imagen final

Una vez obtenidos los archivos en formato GeoTIFF debemos transformarlos en imágenes que sirvan de entrada para el modelo de detección de barcos. Por lo tanto, necesitamos que tengan formato TIFF con tres canales (RGB) y donde el tipo de dato sea entero de 8 bits. Para realizar este procesamiento utilizamos la librería rasterio [72] en Python, especializada en el tratamiento de archivos en formato GeoTIFF.

Con el objetivo de obtener un solo canal en escala de grises se suman los canales de las dos polarizaciones presentes en la imagen GeoTIFF, y se divide el resultado por dos para obtener el promedio entre ambas. Las dos polarizaciones son VV y VH.

El tipo de datos en la imagen GeoTIFF es float de 32 bits. Para distribuir los valores entre toda la escala de grises que posibilita esta resolución, optamos por realizar una ecualización de histograma. De esta forma obtenemos una imagen uniforme en cuanto a los niveles de grises. Por último, se normalizan los valores de cada píxel entre los valores 0 y 255.

Hecho esto, llegamos a tener un canal con la información de las polarizaciones VV y VH, y con valores enteros de 8 bits. Para finalizar, creamos una imagen de formato TIFF donde el canal resultante es utilizado para los tres canales RGB. Esto se hace con el objetivo de tener una imagen RGB pero en escala de grises.

Capítulo 8

Implementación de los detectores

Se decidió utilizar la arquitectura YOLO v4 pre-entrenada con ImageNet mencionada en el capítulo 4 debido a que consigue muy buenos resultados al mismo tiempo que se destaca por su eficiencia computacional.

Al partir de una red pre-entrenada, las capas más profundas que son las encargadas de detectar colores, bordes o formas de los objetos presentes se mantienen invariantes ya que son de utilidad para todas las aplicaciones de detección. Solamente las capas finales son entrenadas nuevamente.

Antes de comenzar el entrenamiento final de la arquitectura se realizaron diferentes búsquedas de hiperparámetros sobre la base de datos HRSID descriptas en la subsección 8.2. Esto es con el objetivo de encontrar el mejor conjunto de hiperparámetros que nos permita obtener los mejores resultados en esta aplicación.

8.1. Cloud computing

Uno de los entornos de desarrollo más comunes en la comunidad científica es Jupyter Notebooks. Este entorno permite crear documentos de Jupyter Notebook [73] con extensión .ipynb, los cuales contienen una lista ordenada de celdas de entrada/salida con código de programación. Además pueden incluir texto, ecuaciones matemáticas y gráficos. Estas características lo hacen idóneo para el trabajo colaborativo y de investigación ya que permite comunicar ideas de forma simple al mismo tiempo que se cuenta con la potencia y versatilidad de un lenguaje de programación completo. El proyecto Jupyter es de código abierto y soporta varios lenguajes de programación, entre ellos, Julia, Python y R.

Si bien el proyecto Jupyter posee su propio editor de documentos que corre de forma local, el hecho de que sea de código abierto permite que otros editores también puedan ejecutar y manipular los archivos con extensión .ipynb. Al ser un proyecto tan popular, muchos entornos de desarrollo y editores de texto han adoptado este tipo de archivos. Por ser muy utilizado para tareas de investigación han surgido varios servicios de computación en la nube que brindan entornos de desarrollo con Notebooks. De esta manera, se puede editar el documento desde una aplicación web y consumir capacidad de cómputo como servicio. Esto es de gran utilidad cuando no se cuenta con el hardware necesario para ejecutar el código con el que se está trabajando. Por otra parte, al estar almacenado en la nube, trabajar en equipo se realiza de forma muy natural.

Entre los servicios de Notebooks en la nube nombrados en el capítulo 6 para este trabajo elegimos la plataforma Google Colab. [58] Esto es debido a que, al contar con

una capa gratuita, podemos realizar pruebas y experimentar libremente y, solo cuando realmente necesitamos gran poder de cómputo, utilizar la versión paga por un período de tiempo limitado. De esta forma reducimos al mínimo los costos de computación en la nube al mismo tiempo que utilizamos una herramienta completa y potente que nos permite trabajar de forma colaborativa.

8.2. Hiperparámetros

La primera decisión que se tomó en esta etapa fue definir qué hiperparámetros ajustar y cuáles mantener constantes. Un error en la elección o la utilización de valores incorrectos nos puede llevar a una red mal entrenada que se traduce en malos resultados.

Al hacer uso de una red pre-entrenada no se modificaron los hiperparámetros relacionados a la arquitectura de la red, mencionados en el capítulo 5, por lo tanto se buscó optimizar los valores de los hiperparámetros asociados al entrenamiento del modelo.

Los hiperparámetros seleccionados fueron: tasa de aprendizaje, momentum, batch size y weight decay.

8.2.1. Primera búsqueda - Optimización Bayesiana

Dado el abanico de posibles algoritmos para la búsqueda de hiperparámetros detallados en el capítulo 5, se decidió comenzar probando si era viable la utilización de Bayesian Optimization ya que, de ser viable, con este algoritmo obtendríamos un conjunto de hiperparámetros mejor y en menos tiempo que utilizando los otros métodos.

Para este algoritmo se decidió realizar 30 pruebas en total sobre el conjunto de datos HRSID. De este dataset se tomó un 25 % del conjunto de entrenamiento para el subconjunto de validación y el 75 % restante se utilizó para el entrenamiento. Se entrenó la red YOLO v4 pre-entrenada utilizando transfer learning durante 1.000 iteraciones.

La tasa de aprendizaje, tamaño de lote y momentum fueron los hiperparámetros que variaron en cada una de las pruebas. El conjunto de valores que podía tomar cada uno de ellos están definido de la siguiente manera:

- Tamaño de lote: [16, 32]
- Momentum: Valor continuo entre 0,9 y 0,999
- Tasa de aprendizaje: Valor continuo entre 0,001 y 0,1

Además se utilizó la inversa de la precisión media (AP) como función objetivo del algoritmo.

Comparando la optimización bayesiana con la búsqueda por grilla, el primer algoritmo tiene la ventaja de necesitar menor cantidad de evaluaciones para llegar al mismo resultado pero requiere mayor costo computacional por prueba. Por otro lado, la búsqueda por grilla tiene la ventaja de poder ejecutarse de manera discontinua ya que todas las pruebas son independientes de las anteriores, muy diferente a lo que ocurre en la optimización bayesiana en la que una prueba depende del resultado de todas las anteriores y por lo tanto todas las evaluaciones deben realizarse de manera continua.

El requerimiento de muchas horas continuas de entrenamiento fue un problema para nosotros, nos encontrábamos utilizando la versión gratuita de Google Colab que nos proporcionaba recursos y sesiones de tiempo limitadas, al mismo tiempo que el entorno

puede quitar los recursos de acuerdo a la demanda sin previo aviso. En este punto, con la experiencia de este intento decidimos realizar una segunda búsqueda de hiperparámetros, está vez utilizando búsqueda por grilla y Google Colab Pro.

8.2.2. Segunda búsqueda - Búsqueda por grillas

En está búsqueda utilizamos nuevamente la base de datos HRSID con una partición de 65 % para entrenamiento y 35 % para prueba propuesta por los autores de la base de datos. A su vez, al subconjunto de entrenamiento se lo dividió en entrenamiento y validación con porcentajes de 75 % y 25 % respectivamente. En conclusión, se usaron 2.732 imágenes para el entrenamiento, 910 imágenes para validación y 1.962 para testeo.

Para llevar a cabo la búsqueda por grilla se definieron las siguientes posibilidades para cada hiperparámetro dando como resultado 72 combinaciones posibles:

- Tamaño de lote: [32; 64]
- Momentum: [0,9; 0,93; 0,95; 0,97]
- Tasa de aprendizaje: [0,001; 0,005; 0,01]
- Weight decay: [0,0001; 0,0005; 0,001]

Antes de llevar a cabo la búsqueda por grilla se realizó un entrenamiento utilizando una de las combinaciones al azar. El modelo se entrenó durante 6.000 iteraciones con tamaño de lote 64, tasa de aprendizaje 0,01, momentum 0,96 y weight decay 0,0005. Durante el entrenamiento se graficó la variación de la pérdida y el mAP del modelo en función de las iteraciones.

Analizando el gráfico y el tiempo empleado en el entrenamiento se definió como 2.500 el número de iteraciones que utilizaremos en la búsqueda. Como se puede observar en la figura 8.1, el mAP obtenido en las iteraciones 2.400-2.500 no obtiene una mejora significativa hasta alrededor de las 5.000 iteraciones, lo que nos supondría duplicar el tiempo total necesario para llevar a cabo el entrenamiento. Por este motivo se decidió utilizar 2.500 iteraciones ya que nos permitiria seleccionar un conjunto de valores de hiperparámetros con un buen rendimiento, mientras que ahorrabamos tiempo y poder de computo.

Los mejores 16 resultados obtenidos de las 72 pruebas realizadas se muestran en la Tabla 8.1.

Durante el entrenamiento de un modelo existe el riesgo de provocar un sobreajuste de los parámetros al conjunto de entrenamiento. Al momento de evaluar la red con una nueva imagen que no fue utilizada para el entrenamiento ni para la validación, es decir, con el conjunto de test, es posible obtener resultados que no son los esperados. Esto se debe a un sobreentrenamiento (overfitting) y el modelo pierde la capacidad de generalización ante nuevos datos de entrada.

Un recurso para evitar este problema es la utilización de alguna técnica de validación cruzada. En este trabajo utilizaremos la técnica de K-Fold Cross Validation o Validación Cruzada de K iteraciones, mencionada en el capítulo 5.

Utilizamos k = 10, en cada una de las diez iteraciones nueve folds serán utilizados como conjunto de entrenamiento y el restante como conjunto de validación. Debido a que el costo computacional y temporal de realizar diez nuevas iteraciones para cada una de las combinaciones evaluadas previamente solo aplicaremos está técnica a los dos mejores

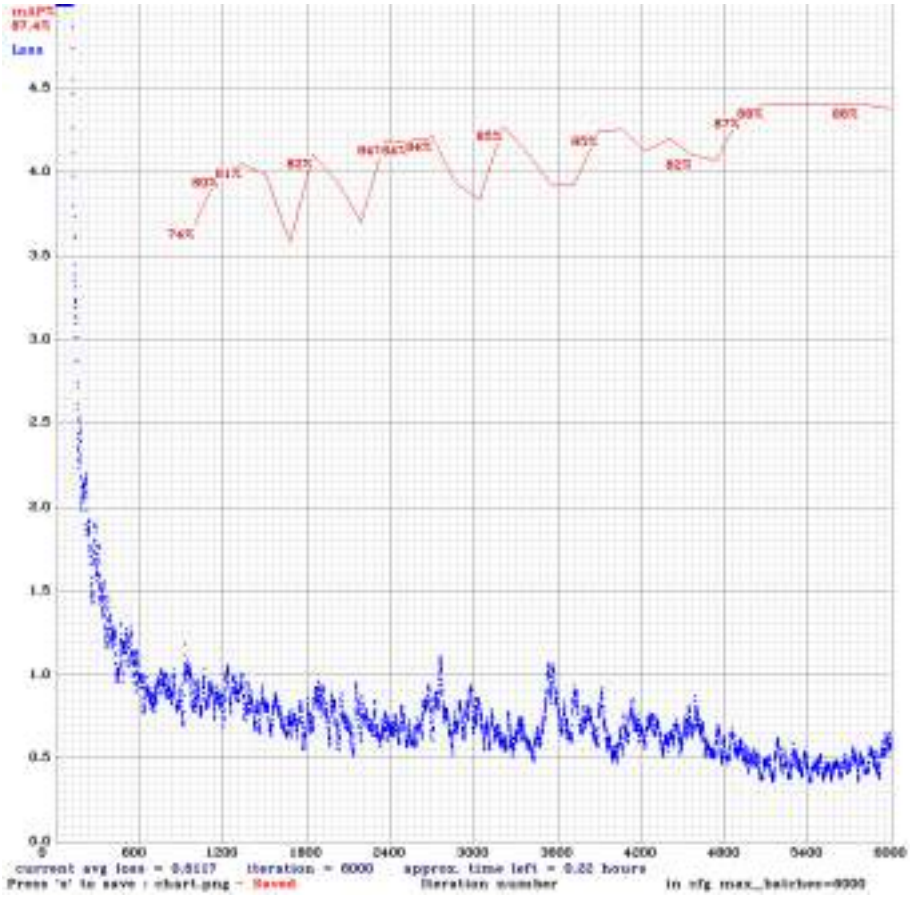


Fig. 8.1: mAP a distintas cantidades de iteraciones

conjuntos obtenidos en la búsqueda por grilla. El valor de mAP reportado es el obtenido de promediar los diez resultados, cada uno con un conjunto de validación diferente.

A partir de evaluar los resultados obtenidos se definieron los siguientes hiperparámetros óptimos para nuestro problema con los que se llevará a cabo el entrenamiento completo de la red.

- Tasa de aprendizaje = 0,01
- Momentum = 0,9
- Tamaño de lote = 64
- Weight Decay = 0,001

8.3. Entrenamiento final

Una vez obtenido el conjunto óptimo de hiperparámetros a utilizar se debía realizar un entrenamiento durante una mayor cantidad de iteraciones con estos valores para obtener los pesos finales de todas las capas de la arquitectura. El entrenamiento, como se mencionó en la sección 8.1, se llevó a cabo en Google Colab, utilizando la técnica de transfer

Nº	mAP	Batch Size	Learning rate	Momentum	Decay
1	0,874741	64	0,01	0,9	0,001
2	0,872719	64	0,01	0,9	0,0005
3	0,8715	64	0,01	0,93	0,0005
4	0,870343	64	0,005	0,95	0,001
5	0,870027	64	0,005	0,9	0,0001
6	0,869974	64	0,01	0,9	0,0001
7	0,869502	64	0,005	0,95	0,0005
8	0,869445	64	0,005	0,93	0,0001
9	0,867443	64	0,005	0,9	0,0005
10	0,865288	64	0,01	0,93	0,001
11	0,864563	64	0,01	0,93	0,0001
12	0,864541	64	0,005	0,93	0,0005
13	0,863141	64	0,005	0,9	0,001
14	0,862964	64	0,01	0,95	0,001
15	0,862784	64	0,005	0,93	0,001
16	0,861994	64	0,01	0,95	0,0005

Tab. 8.1: Resultados obtenidos en la búsqueda por grilla.

Nº	mAP	Batch Size	Learning rate	Momentum	Decay
1	87,39 %	64	0,01	0,9	0,001
2	86,91 %	64	0,01	0,9	0,0005

Tab. 8.2: Resultados de la validación cruzada.

learning y los hiperparámetros óptimos. Además, se definió en 16.000 el número máximo de iteraciones.

Con el objetivo de obtener los mejores resultados se plantearon y desarrollaron diferentes variantes sobre la misma arquitectura YOLO v4. Todas las variantes se entrenaron con el mismo conjunto de datos, los mismos hiperparámetros y durante la misma cantidad de iteraciones. Realizamos dos tipos de experimentos. Por un lado probamos aumentar el tamaño de las imágenes de entrada del modelo durante el entrenamiento, y por el otro añadimos conexiones extras al bloque SPP de la arquitectura. Al aumentar el tamaño de las imágenes de entrada el modelo tendrá más información en su entrenamiento a costa de un mayor uso de memoria y procesamiento. Con respecto a añadir conexiones al bloque SPP, como en las imágenes de entrenamiento existen barcos de distintas dimensiones, conexiones de otros tamaños podría llegar a impactar en una mejora en la detección a distintas escalas. A continuación describiremos cada uno de estos experimentos.

8.3.1. Aumento de la resolución de la entrada

En este caso se plantearon dos variantes de la arquitectura con la única diferencia en el tamaño de entrada de las imágenes al modelo, 416 en el primer caso y 608 en el segundo. Los hiperparámetros utilizados fueron: momentum 0,95; tasa de aprendizaje 0,01; Weight Decay 0,001; y tamaño de lote de 64 muestras.

	YOLO v4 - input 416 (mAP)	YOLO v4 - input 608 (mAP)
train	84,08 %	94,86 %
valid	88,72 %	91,02 %
test	89,14 %	91,17 %
offshore	98,11 %	98,30 %
inshore	82,54 %	85,32 %

Tab. 8.3: Comparación del mAP(0.5) de YOLO v4 con distintos tamaños de entrada sobre los diferentes datasets.

Como se puede observar en la tabla 8.3 aumentar el tamaño de la entrada mejora todos los resultados, siendo el obtenido sobre el dataset de test el que más implicancias tiene incrementándose en un 2 %. Además es interesante destacar la mejora del desempeño sobre el dataset inshore compuesto por imágenes con tierra firme en ellas.

8.3.2. Agregado de conexiones SPP

La arquitectura YOLO v4 original posee un bloque SPP con tres capas de maxpool, con tamaños de kernel 5, 9 y 13 respectivamente, sumado a una conexión directa desde la entrada pudiéndose entender como otra capa más con tamaño de kernel 1.

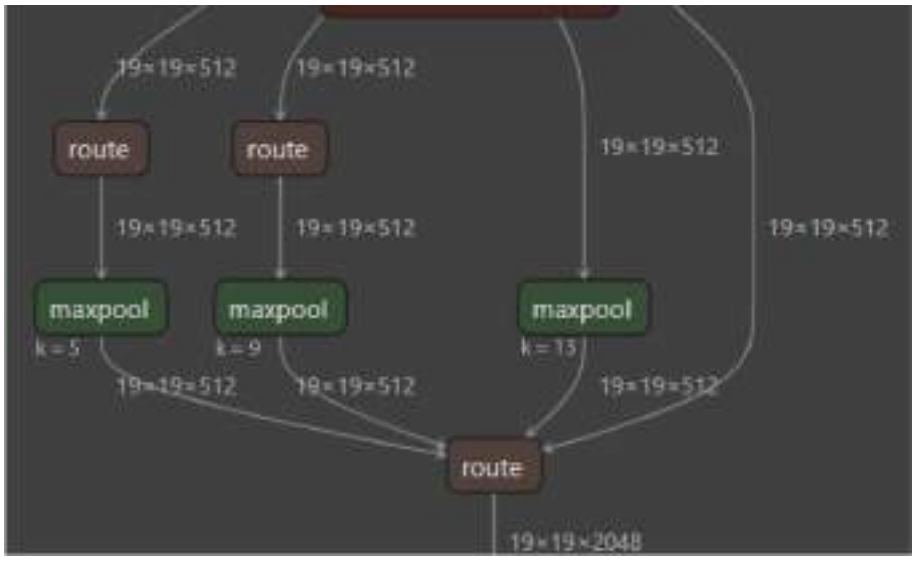


Fig. 8.2: Esquema del bloque SPP de YOLO v4. De izquierda a derecha están las capas de maxpool con kernel de 5, 9 y 13, y una cuarta conexión directa a la derecha.

Como parte del experimento se hicieron dos variantes de la arquitectura original, una

de ellas con una capa maxpool extra con tamaño de kernel 3 y la otra variante con un tamaño de kernel 21. Ambas variantes se entrenaron con un tamaño de entrada de 608.

	Input 608 (mAP)	**Input 608 y SPP 3 (mAP)**	**Input 608 y SPP 21 (mAP)**
train	94,86 %	95,96 %	95,68 %
valid	91,02 %	91,18 %	90,81 %
test	91,17 %	90,88 %	90,37 %
offshore	98,30 %	98,48 %	98,35 %
inshore	85,32 %	86,55 %	85,94 %

Tab. 8.4: Comparación del mAP(0.5) de YOLO v4 con distintos bloques SPP sobre los diferentes datasets.

En la tabla 8.4 se observa que el agregado de capas al bloque SPP de la arquitectura no produce grandes cambios en los resultados obtenidos. Se puede ver que la arquitectura sin agregados sigue siendo la que mejor desempeño tiene sobre el dataset de test. Aún así se puede observar que la variante con una capa maxpool con kernel de 3x3 mejora levemente los resultados sobre los conjuntos de datos offshore e inshore.

Al no poder sacar una conclusión concreta analizando el mAP decidimos analizar la cantidad de predicciones satisfactorias y no satisfactorias de cada modelo.

	Input 608	**Input 608 y SPP 3**	**Input 608 y SPP 21**
TP	5.212	5.301	5.255
FP	801	959	854
FN	710	621	667
avg IoU	73,15 %	71,12 %	72,76 %

Tab. 8.5: Cantidad de barcos detectados y no detectados por cada modelo.

Si bien la variante con una capa extra de 3x3 en el bloque SPP obtiene la mayor cantidad de verdaderos positivos, también posee la mayor cantidad de falsos positivos al mismo tiempo que reduce en 2 % el IoU promedio.

Por otro lado, analizando las inferencias sobre el dataset de test con el modelo sin modificar el bloque SPP llegamos a las siguientes conclusiones:

- La mayoría de los falsos positivos se dividen en:
  - Barcos incompletos en el borde de la imagen
  - Imágenes de muy baja calidad
  - Puntos blancos en la imagen que no son barcos
- Los falsos negativos se suelen dar cuando dos barcos están muy pegados entre sí, detectando uno en vez de ambos.

Con toda esta información llegamos a la conclusión de que las modificaciones realizadas sobre la capa SPP no aportaron grandes beneficios y queda como trabajo futuro seguir explorando en esta línea.

8.3.3. Elección del modelo final

Teniendo en cuenta las conclusiones detalladas en la sección 8.3.2 y 8.3.1 se puede decir que la arquitectura YOLO v4 con entrada de 608 es la que mejor se adapta a nuestro problema.

Se llevó a cabo una validación cruzada k-fold con k = 10 y 8.000 iteraciones en cada entrenamiento para asegurarnos que los resultados no estén sesgados. El resultado promedio del mAP sobre el conjunto de tests arrojó un 91,7 %. De la misma manera, se calculo el promedio de los resultados para las divisiones de offshore e inshore, obteniendo los valores de mAP 98,63 % y 87,64 % respectivamente.

El modelo final está entrenado con el dataset HRSID por 16.000 iteraciones con los hiperparámetros:

- Tasa de aprendizaje: 0,01
- Momentum: 0,9
- Tamaño de lote: 64
- Weight Decay: 0,001

8.4. Comparación con la literatura

A continuación se muestran algunos de los resultados obtenidos por diferentes arquitecturas e implementaciones sobre el conjunto de datos HRSID.

Los resultados se encuentran detallados en el trabajo "HRSID: A High-Resolution SAR Images Dataset for Ship Detection and Instance Segmentation" [65].

En conclusión, observando la tabla 8.6 se puede resaltar una mejora de aproximadamente entre un 2 % y 5 % con respecto a otros modelos. Cabe destacar que el conjunto de datos fue publicado recientemente por lo cual el número de trabajos que lo utilizaron es limitado, siendo complicado encontrar resultados con los que comparar.

La detección de barcos en escenarios de alta mar es menos desafiante para los modelos que la detección en cercanía a la costa donde hay más interferencia y objetos en la imagen que obstaculizan la correcta detección. HRSID está compuesto en un 81,6 % por imágenes mar adentro (offshore) y el restante 18,4 % por escenarios costeros.

En la tabla 8.7 se encuentran los valores obtenidos en diferentes modelos y en la implementación propia en la cual se observa cuanto más desafiante es para la detección la complejidad del escenario donde se encuentra el barco, llegando en algunos casos a una reducción del mAP de más del 20 %.

Nuestra propuesta obtuvo resultados muy satisfactorios en el subconjunto de test y en escenarios complejos de costa, logrando una mejora significativa frente a otros modelos propuestos en la bibliografía.

8.5. Resultados de la detección de barcos

En esta sección se analizará el desempeño de nuestro modelo ante diferentes escenarios destacando sus bondades y detectando los puntos de mejora.

El entrenamiento del modelo se realizó utilizando el dataset HRSID por lo tanto los primeros análisis se realizaron sobre el subconjunto de test del mismo dataset. Luego se

Modelo	Backbone	mAP 0.5
Faster R-CNN	ResNet-50 + FPN	86,7 %
	ResNet-101 + FPN	86,7 %
Cascade R-CNN	ResNet-50 + FPN	87,7 %
	ResNet-101 + FPN	87,9 %
RetinaNet	ResNet-50 + FPN	84,7 %
	ResNet-101 + FPN	84,8 %
Mask R-CNN	ResNet-50 + FPN	88,0 %
	ResNet-101 + FPN	88,1 %
Mask Scoring R-CNN	ResNet-50 + FPN	87,6 %
	ResNet-101 + FPN	88,6 %
Cascade Mask R-CNN	ResNet-50 + FPN	88,5 %
	ResNet-101 + FPN	88,8 %
Hybrid Task Cascade	ResNet-50 + FPN	87,7 %
	ResNet-101 + FPN	87,7 %
HRSDNet	HRFPN-W32	88,4 %
	HRFPN-W40	89,3 %
Implementación propia YOLO v4	CSPDarknet53 + SPP + PANet	91,7 %

Tab. 8.6: Resultados obtenidos por diferentes arquitecturas sobre HRSID.

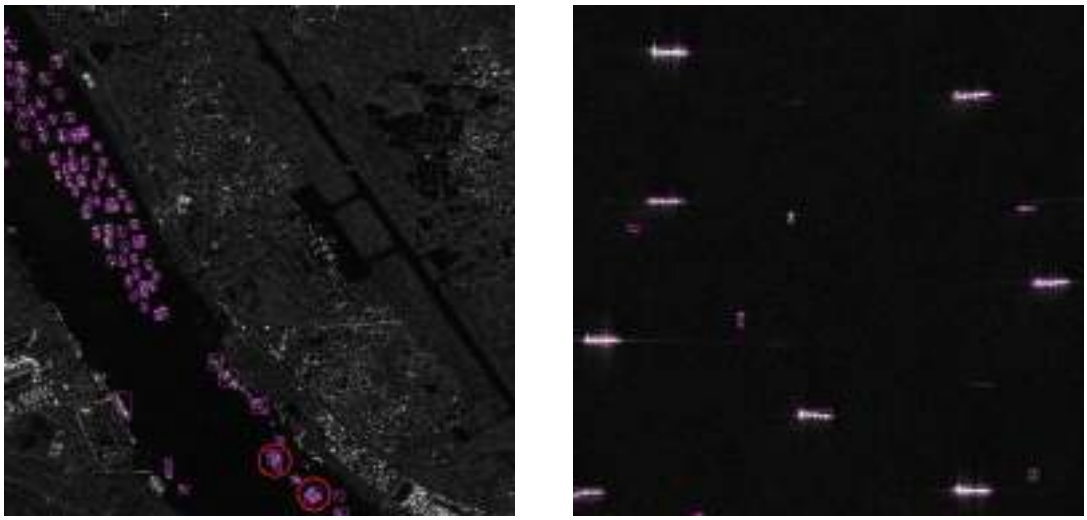
analiza el desempeño sobre imágenes obtenidas de los satélites SAOCOM y Sentinel 1. Al hacer esto se ve qué tan bien se adapta el modelo a imágenes que provienen de fuentes con las que no fue entrenado, en otras palabras, se pone a prueba la capacidad de generalizar el conocimiento.

8.5.1. Inferencias sobre HRSID

Dado que el conjunto de datos HRSID está etiquetado, es decir que una persona experta se tomó el trabajo de analizar la imagen y colocar bounding boxes en los barcos, tenemos datos reales con los que comparar.

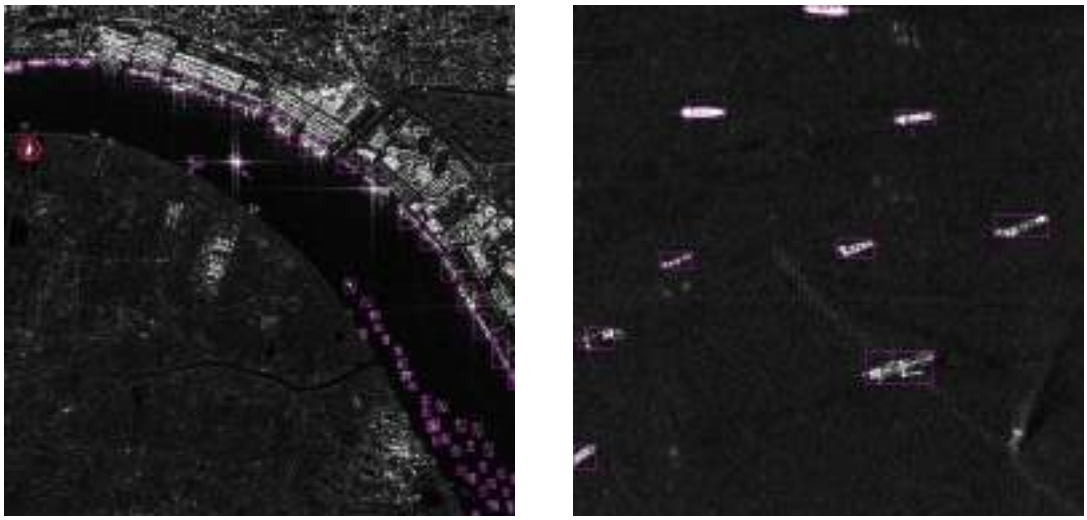
Aciertos

En las imágenes 8.3a, 8.3b, 8.4a y 8.4b se puede ver la efectividad del modelo al detectar barcos de diferentes dimensiones ya sea cuando están totalmente rodeados por agua como cuando hay tierra firme en la imagen. Aún así hay algunos puntos a mencionar. En la parte inferior de la imagen 8.3a hay barcos que están muy cerca entre sí y en estos casos el modelo dibuja tres bounding boxes. Por otro lado, en la imagen número 8.4a hay un falso positivo que está indicado con un círculo rojo.



(a) (b)

Fig. 8.3: Aciertos sobre el conjunto de datos HRSID. (a) (b) Resultado de la detección de barcos en imagen del Puerto Chittagong, Bangladesh. En (a) se resalta con rojo dos errores al detectar múltiples barcos.



(a) (b)

Fig. 8.4: Aciertos sobre el conjunto de datos HRSID. (a) Resultado de la detección de barcos en imagen del Puerto Chittagong, Bangladesh, se resalta con rojo falso positivo. (b) Resultado de la detección de barcos en imagen de la Bahía de Plenty, Nueva Zelanda.

Modelo	Escenario	mAP 0.5
Faster R-CNN	Costa	78,3 %
	Mar adentro	98,0 %
Cascade R-CNN	Costa	79,6 %
	Mar adentro	98,0 %
RetinaNet	Costa	69,0 %
	Mar adentro	98,6 %
Mask R-CNN	Costa	79,0 %
	Mar adentro	98,8 %
Mask Scoring R-CNN	Costa	78,6 %
	Mar adentro	98,0 %
Cascade Mask R-CNN	Costa	80,0 %
	Mar adentro	98,9 %
Hybrid Task Cascade	Costa	82,7 %
	Mar adentro	99,0 %
HRSDNet	Costa	81,3 %
	Mar adentro	96,0 %
Implementación propia YOLO v4	Costa	87,6 %
	Mar adentro	98,6 %

Tab. 8.7: Resultados obtenidos por diferentes arquitecturas sobre HRSID dividido en mar adentro y costa.

Errores

Al analizar las detecciones de nuestro modelo, comparando con los bounding boxes reales del conjunto de datos, se llegó a la conclusión de que la mayoría de los errores se encuentran dentro de los siguientes tres grupos:

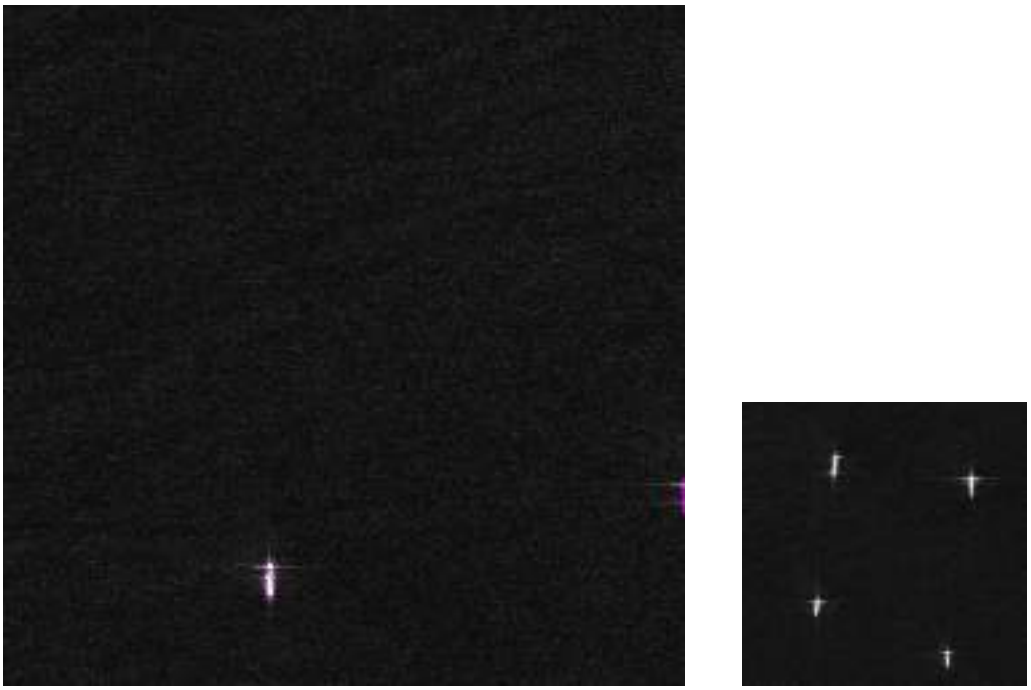
- Barcos incompletos en el borde de la imagen o conjuntos de píxeles que no son barcos.
- Imágenes de mala calidad que dificultan la detección.
- Barcos pegados en los que el modelo los detecta como uno.

En la imagen 8.5a se puede observar un ejemplo de un falso positivo en el borde derecho de la imagen, mínimamente aparecen los primeros píxeles de un barco y el modelo lo detecta como uno, al buscar la imagen contigua (8.5b) observamos que efectivamente se trataba de un barco que está incompleto.

En la imagen 8.6a tenemos dos de los casos de error detallados. En primer lugar hay un falso positivo en la zona central de la imagen. Un conjunto de píxeles blancos fue

detectado como barco. Si bien el tamaño es muy pequeño, la forma del conjunto de píxeles y la cercanía a la costa podría llevarnos a pensar que se trata de un barco costero. En segundo lugar, en el borde derecho se encuentran dos barcos muy cercanos (se puede observar con más detalle en 8.6b) y el modelo los unifica en una sola detección.

Dentro del conjunto de datos HRSID hay un subconjunto de imágenes obtenidas con los satélites TanDEM y TerraSAR-X con resoluciones de 0,5 y 1 metro, estas imágenes de muy alta resolución impactan en el rendimiento del detector de manera negativa. La figura 8.7 muestra un ejemplo de estas imágenes. En la imagen de la izquierda se observan los bounding boxes generados por el modelo y en la imagen de la derecha los bounding boxes reales.

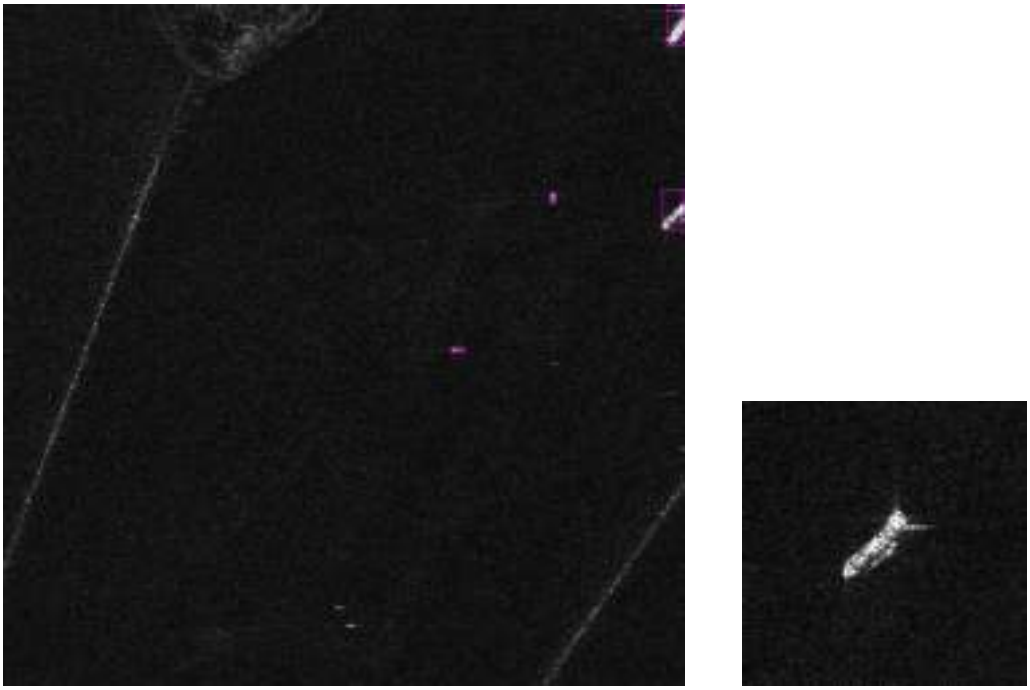


(a) (b)

Fig. 8.5: Falso positivo sobre HRSID. (a) Falso positivo en el borde derecho de la imagen. (b) Imagen aledaña a (a) en la que se observa que el falso positivo se trataba de un barco que esta incompleto.

Conclusión

Luego de analizar los principales aciertos y errores de nuestro desarrollo se puede concluir que los resultados son satisfactorios, obteniendo valores por encima de la literatura. Se consigue observar que nuestra implementación detecta barcos de distintos tamaños y bajo diferentes contextos con gran eficiencia. En contraparte también se debe nombrar que los barcos en los bordes de las imágenes, barcos muy cercanos, el ruido o imágenes con muy alta resolución son un problema que se debe mitigar para mejorar la eficacia.



(a) (b)

Fig. 8.6: Ejemplo de errores sobre HRSID. (a) Falso positivo en la zona central de la imagen y falso negativo en el borde derecho al unificar en una única detección dos barcos. (b) Zoom a los barcos unificados en donde se observa que efectivamente eran dos.

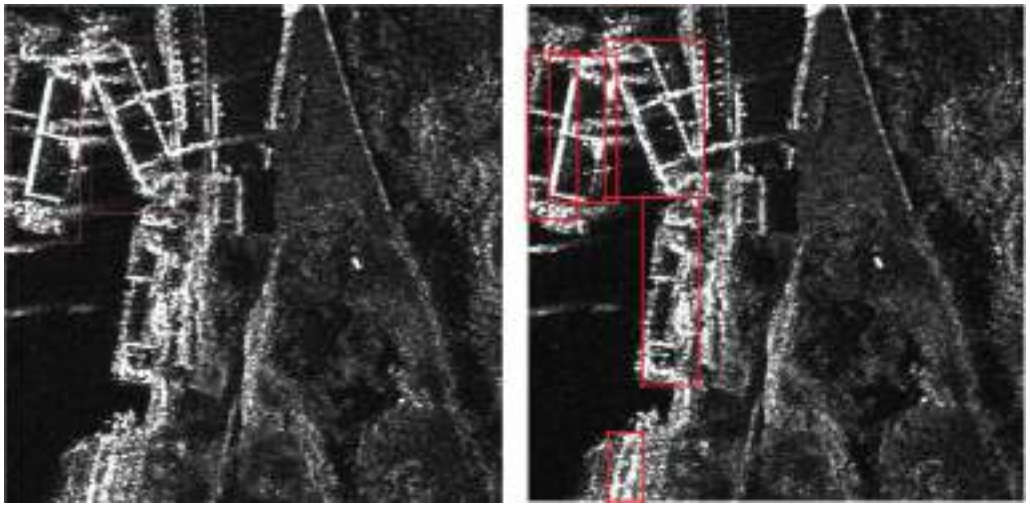


Fig. 8.7: Errores sobre imágenes obtenidas de los satélites TanDEM y TerraSAR-X. (a) Resultado de las detecciones del modelo. (b) Bounding boxes reales.

8.5.2. Detección en imágenes del satélite SAOCOM

Al no tener ground truth para las imágenes del SAOCOM y Sentinel analizamos los resultados de forma cualitativa obteniendo capturas solo de zonas marítimas y apoyándonos en nuestra experiencia al haber visto gran cantidad de imágenes SAR con barcos en el transcurso de este trabajo.

Aciertos

En la figura 8.8 se muestran algunos ejemplos de detecciones realizadas sobre imágenes obtenidas del satélite SAOCOM 1-A, en las que se observa el funcionamiento en nuevos y complejos escenarios. En las figuras 8.8a, 8.8b y 8.8c se observa múltiples aciertos en la detección de barcos en imágenes del canal de Suez en contextos con grandes cantidades de barcos o pequeñas porciones de tierra. Las figuras 8.8d y 8.8e presentan imágenes de mayor complejidad, debido a la presencia de ruido, en las que el modelo detecta correctamente los barcos presentes.

Errores

En la imagen 8.9a se puede observar dos potenciales falsos negativos mientras que en la imagen 8.9b hay un falso positivo en tierra firme.

La imagen número 8.9c corresponde a la costa de Mar del Plata. Se puede observar que detecta cuatro barcos en el puerto. Sabiendo que las detecciones son en el área de la Escollera Sur y que también suele haber barcos en la Base Naval llegamos a la conclusión de que puede haber barcos en la imagen que el modelo no esté detectando. Aún así, la cantidad de ruido en la imagen es superior a otras tomas del mismo satélite, siendo este un factor que dificulta la detección en esta toma en particular y marcando como trabajo a futuro mejorar el preprocesamiento de estas imágenes.

Por último analizamos un error en la zona del Canal de Suez, donde se detectan dos objetos que no tienen aspecto a barco.

8.5.3. Detección en imágenes del satélite Sentinel

Al igual que con las imágenes del satélite SAOCOM, los ejemplos de detecciones mostrados a continuación no presentan una bounding box esperado, sino que a partir de nuestro conocimiento y experiencia fueron clasificadas en aciertos y errores.

Aciertos

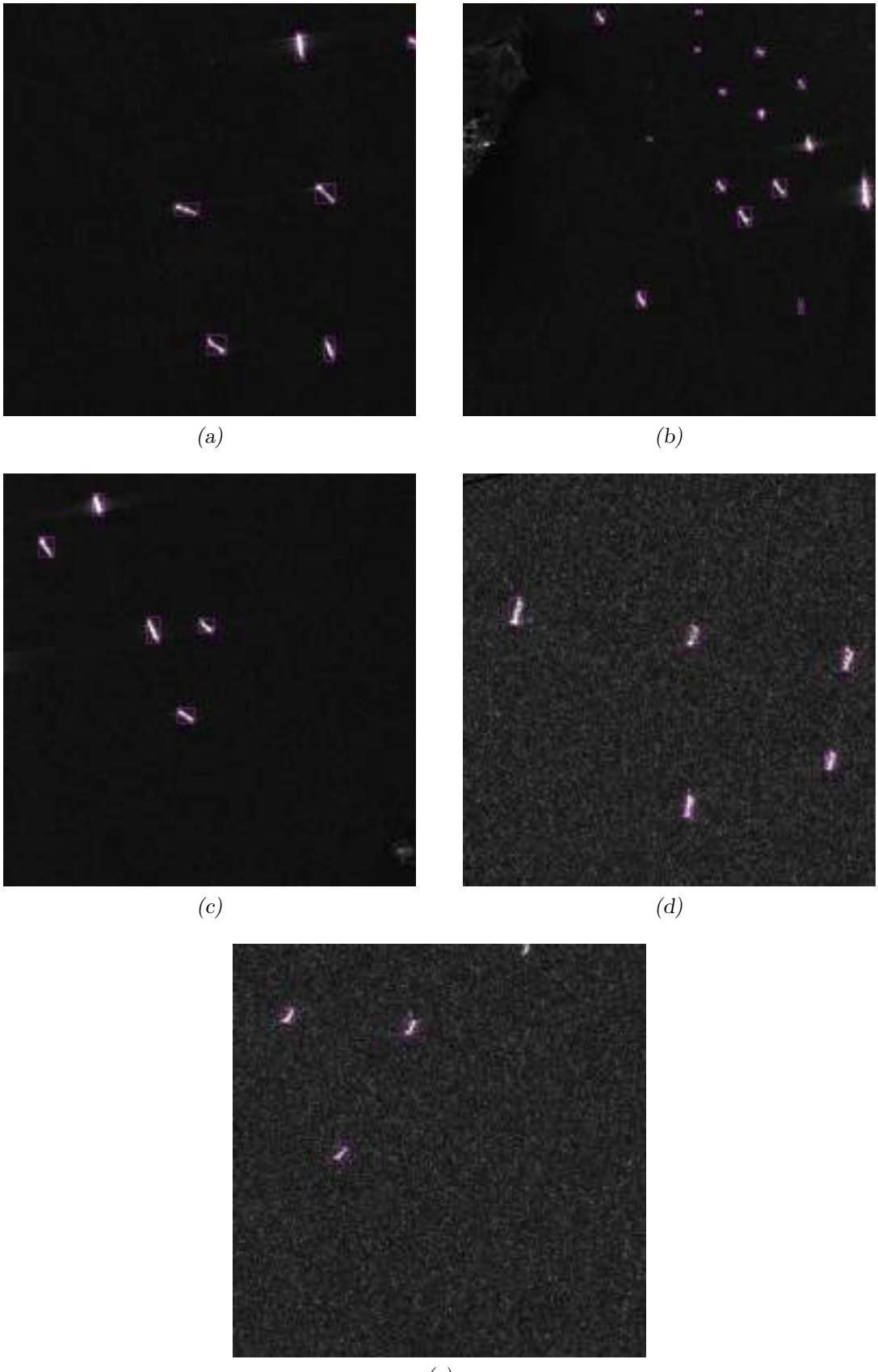
En la figura 8.10 se exhiben ejemplos de detecciones acertadas sobre imágenes obtenidas en distintas ubicaciones por el satélite Sentinel 1. En las figuras 8.10a y 8.10b se observa la capacidad del modelo de detectar correctamente barcos un complejo escenario como lo es el Rio Mississippi. Por otra parte, las figuras 8.10c, 8.10d y 8.10e, obtenidas en el canal de Suez, muestran la habilidad del modelo para detectar múltiples cantidades de barcos de muy pequeño tamaño en una sola imagen.

Errores

En la imagen 8.11a se puede notar que es una zona con muchos potenciales barcos y aunque el modelo fue capaz de detectar la gran mayoría de las embarcaciones, existen posibles falsos negativos.

En la imagen 8.11b, tomada del canal de Suez, posibles barcos en la orilla de la escollera no fueron detectados.

Por último, en la imagen 8.11c, correspondiente al Río Mississippi, se detectó correctamente el barco que está atravesando el río pero en la parte superior de la imagen hubo falsos positivos con pequeñas porciones de tierra.

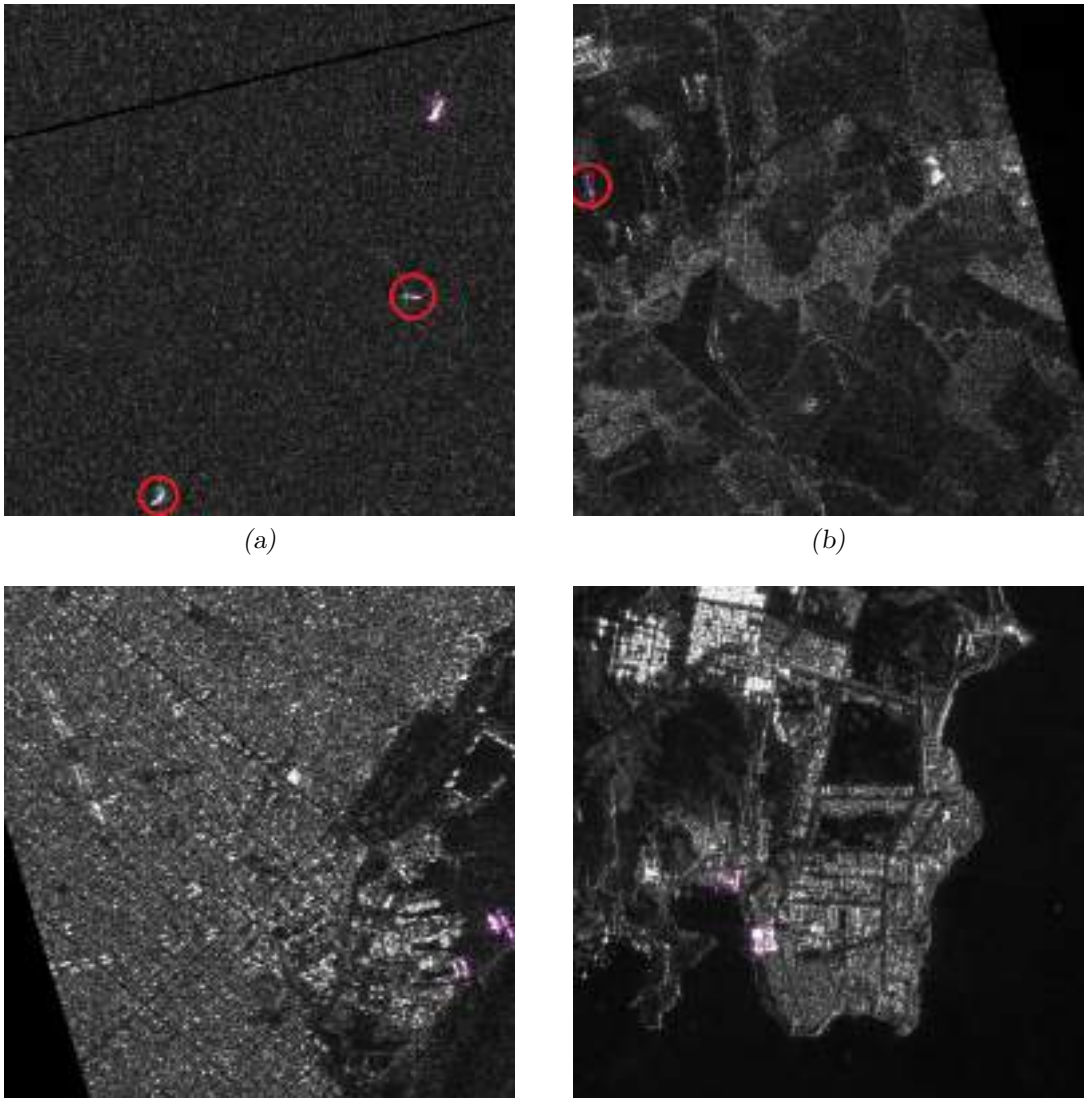


(a) *(b)*

(c) *(d)*

(e)

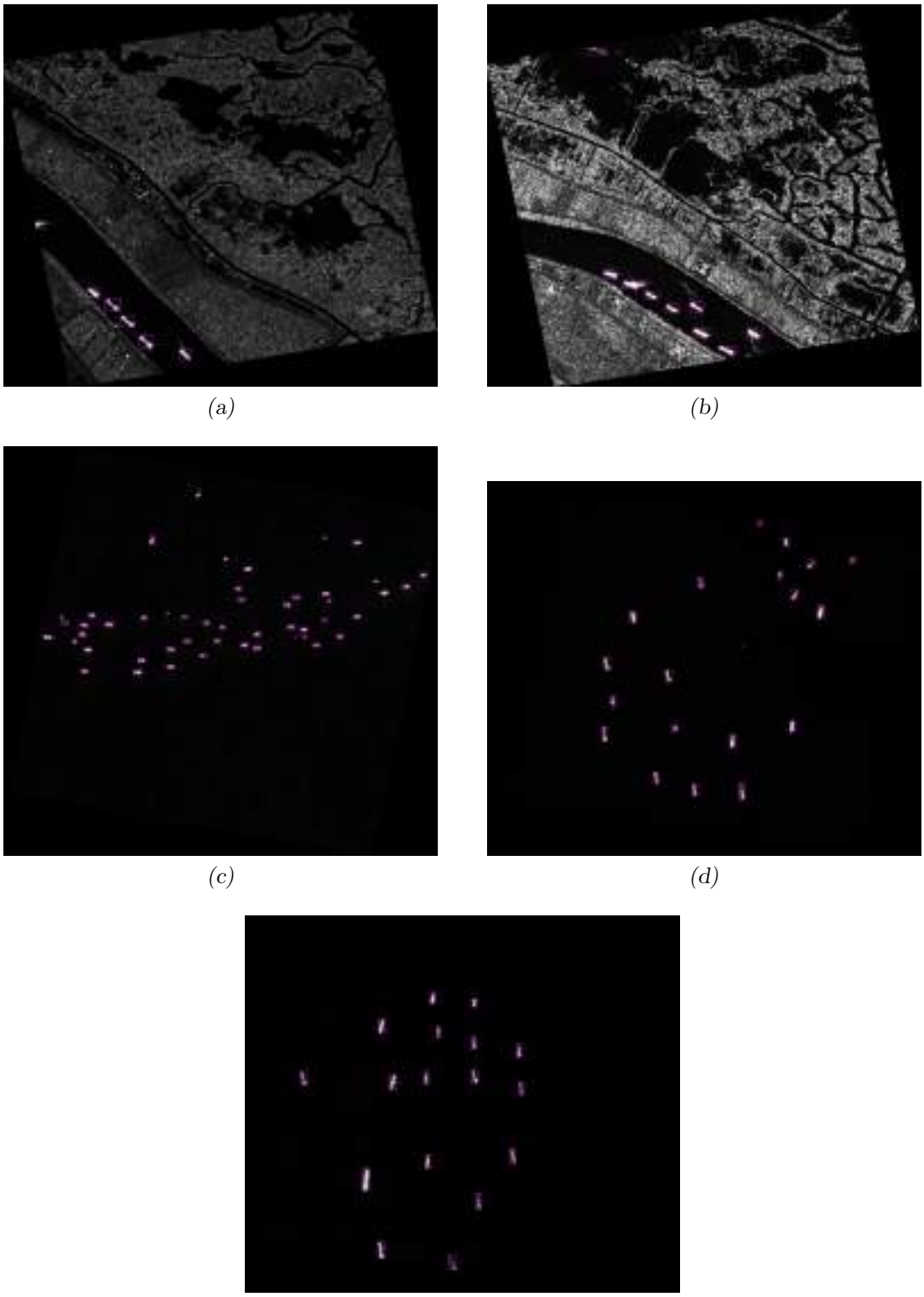
Fig. 8.8: SAOCOM Product®- CONAE - 2022. All Rights Reserved - (a) (b) (c) Imágenes del canal de Suez tomada con el SAOCOM 1-A en modo Stripmap (d) (e) Imagen del Río de la Plata tomada con el SAOCOM 1-A en modo Stripmap



(a) *(b)*

(c) *(d)*

Fig. 8.9: SAOCOM Product®- CONAE - 2022. All Rights Reserved - (a) (b) Imagen del Río de la Plata tomada con el SAOCOM 1-A en modo Stripmap. (c) Imagen de Mar del Plata tomada con el SAOCOM 1-A en modo Stripmap (d) Imagen del canal de Suez tomada con el SAOCOM 1-A en modo Stripmap

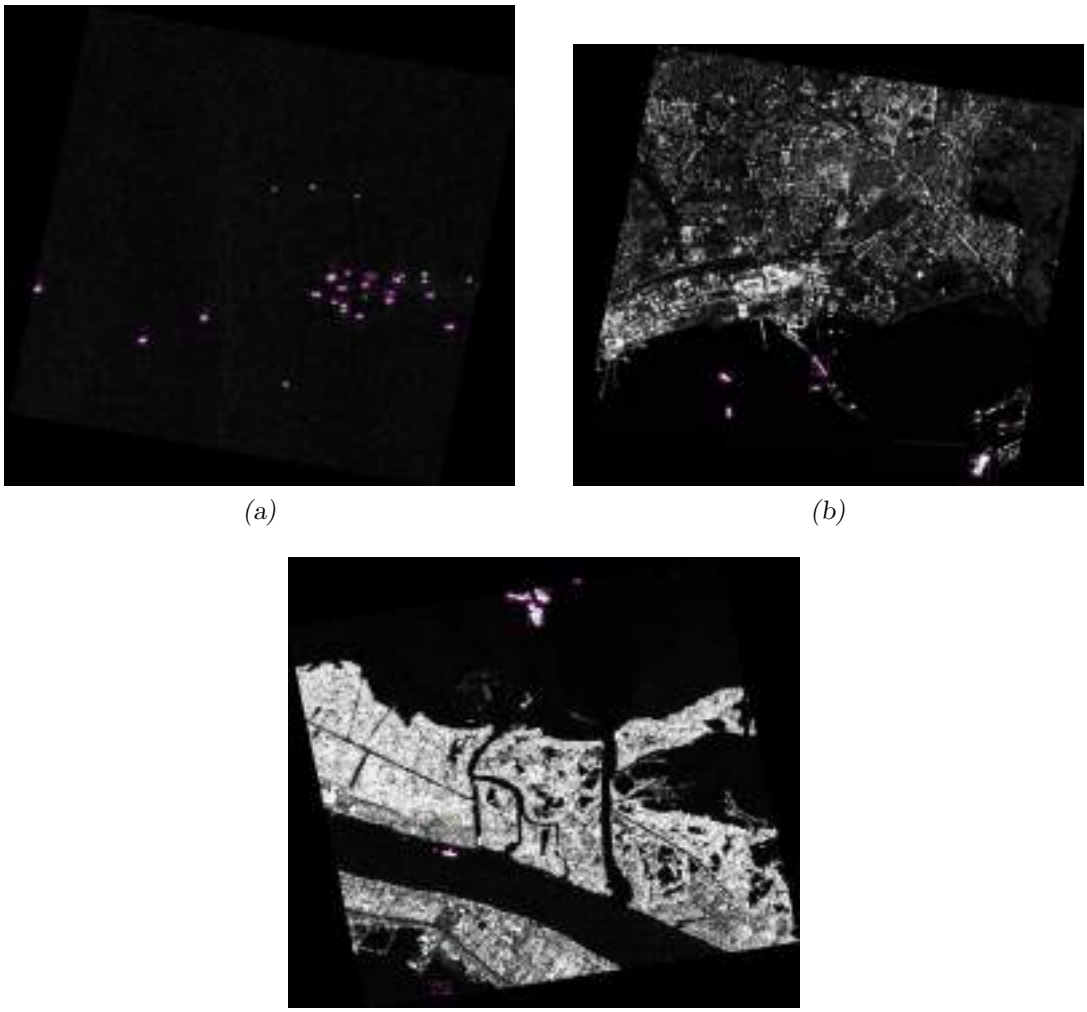


(a) *(b)*

(c) *(d)*

(e)

Fig. 8.10: (a) (b) Imagen del Río mississippi - New Orleans tomada con el Sentinel 1 (c) Imagen del Bangladesh tomada con el Sentinel 1 (d) (e) Imagen del canal de Suez tomada con el Sentinel 1



(a) *(b)*

(c)

Fig. 8.11: (a) Imagen del Bangladesh tomada con el Sentinel 1 (b) Imagen del canal de Suez tomada con el Sentinel 1 (c) Imagen del Río mississippi - New Orleans tomada con el Sentinel 1

Para finalizar podemos decir que el modelo logra generalizar el conocimiento a nuevas capturas de los satélites SAOCOM 1 y Sentinel 1 con puntos de mejora en el ruido de las imágenes y en la detección de barcos cercanos a la costa.

Capítulo 9

Conclusión

En este trabajo final se ha llevado a cabo una investigación sobre el estado del arte y los problemas de la detección de barcos a partir de imágenes satelitales obtenidas mediante el uso de radares de apertura sintética y su tratamiento y abordaje utilizando técnicas de aprendizaje profundo basadas en redes neuronales convolucionales, con el objetivo de reproducir y mejorar resultados de la literatura. Además, se analizaron diferentes plataformas en la nube para el entrenamiento e inferencia de modelos.

Se presentaron implementaciones y variantes de arquitecturas basadas en una red convolucional muy reciente como lo es YOLO v4, presentada en el año 2020, para el problema de la detección de barcos en imágenes SAR. Se utilizó la base de datos HRSID para los experimentos realizados. HRSID, presentada en el año 2020, es una de las pocas bases de datos de público acceso de imágenes SAR de barcos. Este conjunto de datos está compuesto por una gran cantidad de imágenes en diferentes contextos, ya sea mar adentro o escenarios cercanos a la costa y barcos de diferentes tamaños. A su vez está conformado por imágenes adquiridas por distintos satélites, con diferentes resoluciones y características.

Se utilizó la técnica de Transfer Learning junto con otras estrategias y lineamientos utilizados en la literatura durante el desarrollo del trabajo lo que nos permitió alcanzar resultados satisfactorios, mejorando los valores obtenidos por otros trabajos de referencia. El modelo propuesto obtuvo un mAP 0.5 de 91,7 % en el subconjunto de test de HRSID y valores de 87,64 % y 98,63 % para los subconjuntos de costa y mar adentro respectivamente. Estos valores mejoran los resultados obtenidos por varios de los trabajos publicados recientemente.

En conjunto con la Univerdad Nacional de Mar del Plata se obtuvo una licencia de investigación de la CONAE que nos permitió acceder a los datos de los satélites SAOCOM 1A, lanzado en 2018, y SAOCOM 1B, lanzado en 2020. También se obtuvo acceso a los datos provistos por los satélites Sentinel 1A y 1B, lanzados en 2014 y 2016 respectivamente, de la ESA. Se elaboró un conjunto de datos propio a partir de los satélites Sentinel y SAOCOM con el objetivo de poner a prueba el modelo en entornos reales, nuevos y de interés. Se llevo a cabo un procesamiento estándar a los datos crudos provistos por los satélites Sentinel 1 y SAOCOM 1 para que puedan ser utilizados en nuestro modelo. De este conjunto de datos se obtuvieron gratos resultados, valorando la capacidad de generalización del modelo y la posibilidad de aplicarlo a problemáticas locales.

A futuro se pueden generar conjuntos de datos propios para el entrenamiento del modelo incluyendo capturas del SAOCOM 1. Además, se podría crear un producto que cubra todos los aspectos de la detección, desde la descarga de los datos hasta la presentación de

las detecciones en un mapa de forma automática. Otro punto de mejora podría ser la utilización de otras técnicas de preprocesamiento de los datos crudos para obtener imágenes con menor ruido o mejores características. El uso de otros modelos o seguir trabajando en la mejora del modelo propuesto podría ser considerado.

Se ha realizado un trabajo con resultados satisfactorios en la detección de barcos en imágenes SAR, mejorando los valores obtenidos por varios de los trabajos recientemente publicados, con la capacidad de procesar imágenes en tiempo real. Se evalúo su funcionamiento en nuevos escenarios obteniendo gratos resultados, demostrando la capacidad de generalización del modelo.

Bibliografía

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[2] C. P. Schwegmann, W. Kleynhans, and B. P. Salmon, "Ship detection in south african oceans using sar, cfar and a haar-like feature classifier," 2014.

[3] "Tutorial: Fundamentals of remote sensing," February 29, 2016 2016.

[4] "Albedo." https://www.mendoza.conicet.gov.ar/portal/enciclopedia/terminos/Albedo.htm. Accedido 11-03-2022.

[5] "Ucs satellite database." https://www.ucsusa.org/resources/satellite-database. Accedido 11-03-2022.

[6] "What is remote sensing?." https://earthdata.nasa.gov/learn/backgrounders/remote-sensing. Accedido 11-03-2022.

[7] M. I. Skolnik, "Radar." https://www.britannica.com/technology/radar. Accedido 11-03-2022.

[8] A. Moreira, P. Prats-Iraola, M. Younis, G. Krieger, I. Hajnsek, and K. P. Papathanassiou, "A tutorial on synthetic aperture radar," *IEEE Geoscience and Remote Sensing Magazine*, vol. 1, no. 1, pp. 6–43, 2013.

[9] "Seasat (seafaring satellite) mission." https://earth.esa.int/web/eoportal/satellite-missions/s/seasat. Accedido 11-03-2022.

[10] "European space agency." https://www.esa.int/. Accedido 11-03-2022.

[11] "Radarsat." https://en.wikipedia.org/wiki/RADARSAT. Accedido 11-03-2022.

[12] "Nasa-isro sar mission (nisar)." https://nisar.jpl.nasa.gov/. Accedido 11-03-2022.

[13] "Plan espacial nacional." https://www.argentina.gob.ar/ciencia/conae/plan-espacialT. Accedido 11-03-2022.

[14] "Comisión nacional de actividades espaciales." https://www.argentina.gob.ar/ciencia/conae. Accedido 11-03-2022.

[15] J. B. Campbell and R. H. Wynne, *Introduction to remote sensing*. Guilford Press, 2011.

[16] H. Wang and B. Raj, "On the origin of deep learning," 2017.

[17] T. M. Mitchell, *Machine learning*. MTM, 2015.

[18] S. Haykin, *Neural networks and learning machines*. New York: Prentice Hall/Pearson, 2009.

[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, 09 2014.

[21] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Commun. ACM*, vol. 60, no. 6, p. 84–90, 2017.

[24] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[25] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[26] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2015.

[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[29] "Transfer learning - matlab." `https://la.mathworks.com/discovery/transfer-learning.html`. Accedido 11-03-2022.

[30] "A gentle introduction to object recognition with deep learning." `https://machinelearningmastery.com/object-recognition-with-deep-learning/`. Accedido 11-03-2022.

[31] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013.

[32] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[33] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[34] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[35] J. Redmon, "Darknet: Open source neural networks in c." `http://pjreddie.com/darknet/`, 2013–2016.

[36] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[37] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[38] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," 2016.

[39] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft coco: Common objects in context," 2015.

[40] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020.

[41] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *Computer Vision – ECCV 2014*, pp. 346–361, Springer International Publishing, 2014.

[42] N. Mohd Aszemi and D. D. D. Panneer Selvam, "Hyperparameter optimization in convolutional neural network using genetic algorithms," *International Journal of Advanced Computer Science and Applications*, vol. 10, pp. 269 – 278, 06 2019.

[43] R. G. P. John Hertz, Anders Krogh, *Introduction to the Theory of Neural Computation*. CRC Press, 1991.

[44] T. Agrawal, *Hyperparameter Optimization in Machine Learning*. Apress, 2021.

[45] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[46] V. Nguyen, "Bayesian optimization for accelerating hyper-parameter tuning," in *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pp. 302–305, 2019.

[47] "K-fold and montecarlo cross-validation vs bootstrap: a primer." `https://nirpyresearch.com/kfold-montecarlo-cross-validation-bootstrap-primer/`. Accedido 07-03-2022.

[48] Y. LeCun, "1.1 deep learning hardware: Past, present, and future," in *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 12–19, 2019.

[49] "Hardware for machine learning and neural network." https://medium.com/@sayalipangre123/hardware-for-machine-learning-and-neural-network-864544e54e4f. Accedido 07-03-2022.

[50] A. Sunyaev, *Internet Computing*. Springer, 2020.

[51] B. R. Voorsluys W, Broberg J, *Introduction to cloud computing*, pp. 3–42. Wiley, 2011.

[52] "What is aws." https://aws.amazon.com/es/what-is-aws/. Accedido 07-03-2022.

[53] "Amazon sagemaker studio." https://aws.amazon.com/es/sagemaker/studio/. Accedido 07-03-2022.

[54] "Amazon sagemaker pricing." https://aws.amazon.com/es/sagemaker/pricing/. Accedido 07-03-2022.

[55] "Azure ai." https://azure.microsoft.com/en-us/overview/ai-platform/. Accedido 07-03-2022.

[56] "Machine learning studio (classic) pricing." https://azure.microsoft.com/en-us/pricing/details/machine-learning-studio/. Accedido 07-03-2022.

[57] "Vertex ai." https://cloud.google.com/vertex-ai. Accedido 07-03-2022.

[58] "Google colab." https://colab.research.google.com/signup. Accedido 07-03-2022.

[59] D. Burgess, "Automatic ship detection in satellite multispectral imagery," vol. 59, pp. 229–237, 01 1993.

[60] C. C. Wackerman, K. S. Friedman, W. Pichel, P. Clemente-Colon, and X. Li, "Automatic detection of ships in radarsat-1 sar imagery," *Canadian Journal of Remote Sensing*, vol. 27, pp. 371 – 378, 2001.

[61] J. Li, C. Qu, and J. Shao, "Ship detection in sar images based on an improved faster r-cnn," in *2017 SAR in Big Data Era: Models, Methods and Applications (BIGSARDATA)*, pp. 1–6, 2017.

[62] Y. Li, Z. Ding, C. Zhang, Y. Wang, and J. Chen, "Sar ship detection based on resnet and transfer learning," in *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pp. 1188–1191, 2019.

[63] M. Kang, K. Ji, X. Leng, and Z. Lin, "Contextual region-based convolutional neural network with multilayer fusion for sar ship detection," *Remote Sensing*, vol. 9, p. 860, Aug 2017.

[64] Y.-L. Chang, A. Anagaw, L. Chang, Y. C. Wang, C.-Y. Hsiao, and W.-H. Lee, "Ship detection based on yolov2 for sar imagery," *Remote Sensing*, vol. 11, no. 7, 2019.

[65] S. Wei, X. Zeng, Q. Qu, M. Wang, H. Su, and J. Shi, "Hrsid: A high-resolution sar images dataset for ship detection and instance segmentation," *IEEE Access*, vol. 8, pp. 120234–120254, 2020.

[66] Q. Oliver, *Understanding synthetic aperture radar images*. Raleigh, 2004.

[67] Y. Wang, C. Wang, H. Zhang, Y. Dong, and S. Wei, "A sar dataset of ship detection for deep learning under complex backgrounds," *Remote Sensing*, vol. 11, no. 7, 2019.

[68] "Copernicus open access hub." `https://scihub.copernicus.eu/`. Accedido 07-03-2022.

[69] "Alaska satellite facility." `https://asf.alaska.edu/`. Accedido 07-03-2022.

[70] "Snap." `https://step.esa.int/main/toolboxes/snap/`. Accedido 07-03-2022.

[71] "Rus webinar: Processing copernicus data in python using snappy." `https://www.youtube.com/watch?v=PiU68g3WRIY`. Accedido 07-03-2022.

[72] "Rasterio." `https://rasterio.readthedocs.io/`. Accedido 07-03-2022.

[73] "Project jupyter." `https://jupyter.org/`. Accedido 15-06-2022.