Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

31105151 - Trabajo Fin De Máster En Ingeniería De Software Y Sistemas Informáticos

Modelo Mask R-CNN aplicado a la detección de armas en videovigilancia

Realizado por:

Rafael Ángel Ruiz Lucena

Director:

Dr. Pedro Javier Herrera Caro

Curso: 2021/2022

Convocatoria: Junio

Máster Universitario de Investigación en Ingeniería de Software y Sistemas Informáticos

31105151 - Trabajo Fin De Máster En Ingeniería De Software Y Sistemas Informáticos

Modelo Mask R-CNN aplicado a la detección de armas en videovigilancia

Línea de Investigación:

Sistema de detección de objetos basado en redes neuronales de convolución

Realizado por:

Rafael Ángel Ruiz Lucena

Director:

Dr. Pedro Javier Herrera Caro

Curso: 2021/2022

Convocatoria: Junio

Resumen

Grandes superficies como aeropuertos o espacios públicos están controladas por sistemas de videovigilancia compuestos por centenares de cámaras, que deben proporcionar al operador alertas en tiempo real, e información fiable y certera ante una posible situación de peligro. Este planteamiento induce a reflexionar varios aspectos, por ejemplo, cuáles son los límites visuales de una persona para analizar tantas imágenes en tiempo real, o también sobre las carencias de estos sistemas y cuáles son las posibles aplicaciones que pueden ofrecer nuevas tecnologías, en especial la inteligencia artificial, con el fin de proporcionar nuevas aplicaciones y mejoras a estos sistemas.

El presente trabajo de investigación tiene como objetivo el estudio, análisis y cuantificación de los posibles modelos de inteligencia artificial, para la detección y clasificación de objetos aplicados a la detección de armas, pasando desde un comienzo por la revisión de la problemática planteada, siguiendo un diseño de una solución factible hasta finalmente una evaluación y resultados de esta.

El diseño de la solución propuesta está basado en el modelo Mask R-CNN para la detección y clasificación de objetos. Tras el estudio, análisis y comparación con otros modelos previos, se ha realizado un entrenamiento de dicho modelo, con el mismo *dataset* que los analizados, mostrando finalmente ser un modelo efectivo para la detección de armas, ofreciendo resultados de hasta el 89% de "*average Precision*" y 89% de sensibilidad. Lo cual demuestra un avance en el estado del arte en lo que respecta al campo de investigación, a la vez que una solución viable para mejorar los sistemas de videovigilancia.

Palabras clave: Mask R-CNN, red neuronal convolucional, detección de armas, dataset, videovigilancia.

Contenido

Pág.

Lista de figuras

Pág.

Lista de tablas

Pág.

1. Introducción

1.1 Motivación

En la actualidad existen mecanismos para la prevención y control de desarrollo de actividades que permiten evitar riesgos y promover la salud. Dichos mecanismos que proporcionan seguridad suelen aparecer en dos tipos claramente diferenciados. El primero de ellos consiste en el personal de vigilancia que vela y toma decisiones y acciones para que se eviten posibles riesgos, y por otro lado, sistemas de videovigilancia que precisan de un menor número de personal, pero sí de una inversión tecnológica mínima suficiente como para mantener un control general del entorno en el que se desarrolla la actividad. La prevención de riesgo a través de sistemas de videovigilancia es hoy día un emergente campo de inversión en lo que respecta a seguridad, ya que proporciona ciertas ventajas respecto a otros mecanismos. Una de las ventajas es la rapidez y control que facilita ante una situación de riesgo, lo que hace un campo de estudio aún más atractivo para mejorar con el objetivo de maximizar el potencial que puede ofrecer a un usuario para minimizar el riesgo. Este mecanismo de prevención lleva atado una desventaja, la cantidad de imágenes por segundo que un sistema de videovigilancia suministra al operario tiende a ser mucho mayor que la capacidad del ojo humano en asimilar todas las acciones que transmiten la cámaras y sensores que componen un sistema de videovigilancia. Como también el análisis detallado de esas imágenes para tomar así decisiones que pueden evitar un posible riesgo. Normalmente, un sistema de videovigilancia suele estar compuesto por un sistema de cámaras que graban las acciones que se producen en el entorno, un servidor encargado de transmitir y procesar dichas e imágenes, y una sala de control en la cual se encuentra lo paneles o monitores, así como los operarios que analizan esas imágenes.

Como es de esperar, un panel de videovigilancia no puede mostrar a la vez todas las imágenes que están siendo grabadas al mismo tiempo, por lo que se debe dar prioridad a aquellas que tengan mayor importancia para el vigilante u operario del sistema de seguridad. También se puede alternar dichas imágenes con pequeños intervalos de tiempo que permitan al usuario analizar las imágenes por cierto periodo de tiempo o alarma detectada. Pero si se indaga un poco más en el análisis del caso de uso, puede existir la posibilidad de que se

quiera detectar una acción específica, por ejemplo, la detección de un arma potencialmente peligrosa o el acceso de una persona a un área prohibida. Puede darse la posibilidad de que el sistema de videovigilancia solo contenga un número reducido de cámaras y, por lo tanto, sea fácil detectar dicha acción, pero por otro lado puede ser que el sistema esté instalado en un aeropuerto, estación de tren o en una calle muy transitada, y sea realmente difícil poder analizar si hay una persona que puede poseer un cuchillo o una pistola, acceso a área prohibida, etc.

1.2 Justificación

Tras analizar el entorno en el que se produce está problemática, este trabajo se centra y enfoca en analizar las posibles soluciones existentes hoy en día respecto a dicho contexto. Debido a la necesidad que esta problemática precisa de ofrecer certeza y seguridad ante la detección de un posible peligro es de entender que se pretenda considerar los avances tecnológicos en lo que respecta al análisis de imágenes para ofrecer una solución a dicha problemática. Dentro del campo de estudio del procesado de imágenes existen dos vertientes claramente diferenciadas: los algoritmos de fuerza bruta, lo cuales utilizan algoritmos de cros correlación en los que la imagen objetivo se compara con una imagen referencia extrayendo el valor de correlación producido con el fin de estudiar si la imagen ha cambiado en un área específica y, por otro lado, están las nuevas técnicas de inteligencia artificial. Existen varios mecanismos desde hace años que trabajan con imágenes para detectar diferentes objetos en una imagen. En este trabajo se destaca el papel de los modelos inteligentes, que en los últimos han ofrecido ayuda a gran número de problemas, con el fin de detectar con mayor certeza y rapidez si en una imagen aparece un arma potencialmente peligrosa, gracias principalmente a la capacidad de análisis que un ordenador dedicado puede ofrecer es mucho más rápida de la que un ojo humano puede llegar a alcanzar. De forma más específica este trabajo se centrará en implementar un modelo de inteligencia artificial capaz de detectar armas en un entorno de videovigilancia.

1.3 Objetivos generales y específicos

Con el fin de establecer las bases, en este trabajo se puede definir como objetivo general, el desarrollo de un modelo de inteligencia artificial que permita su entrenamiento para finalmente utilizarlo como una herramienta para la detección de armas en un video.

Como objetivo más específico se pueden destacar:

- Identificar en la literatura cual es el estado actual del arte sobre el tema en cuestión.
- Localizar en la literatura las partes relevantes que permiten un avance en el estudio de la materia y problemática planteada.
- Comparar las técnicas aplicadas en el campo de estudio y la evaluación de otras técnicas utilizadas en el campo de la detección de armas.
- Estudiar los modelos utilizados y *frameworks* de inteligencia artificial que se utilizan para en su desarrollo.
- Desarrollar un modelo de inteligente capaz de detectar un arma dentro de una imagen.
- Entrenar y evaluar el modelo desarrollado.
- Mejorar dicho modelo utilizando nuevas técnicas y otros modelos ya analizados e implementados para la detección de armas.
- Analizar la posible mejora que se ha obtenido y como aportación la nueva rama de investigación que puede surgir.

1.4 Organización de memoria

Esta memoria está organizada en seis capítulos diferentes. El primero de ellos es la introducción en el que se presenta la problemática existente, así como los objetivos que se esperan obtener después de este trabajo. El capítulo dos consiste en el estudio del estado del arte que abarca los primeros algoritmos para el procesamiento de imágenes hasta llegar al empleo de la inteligencia artificial para la detección de armas, así como una descripción de los fundamentos base en inteligencia artificial como perceptrón, red neuronal y red neuronal convolucional. Se presentan dos apartados fundamentales, el marco teórico en el que se explica las bases de las redes neuronales, y a partir de ahí se avanza en el estudio y evolución

de modelos artificiales utilizados para la clasificación y detección de objetos de una forma general hasta llegar a la detección de armas con inteligencia artificial.

Seguidamente, en el tercer capítulo se analiza y detalla la solución propuesta utilizada para resolver la problemática expuesta en la introducción, así como los *frameworks* utilizados para el desarrollo del modelo.

El cuarto capítulo muestra el experimento que se ha realizado, así como el *dataset* utilizado para el entrenamiento y los resultados obtenidos.

El capítulo cinco aporta una conclusión al trabajo sobre los resultados obtenidos por la solución propuesta y un posible trabajo futuro.

El ultimo capitulo contiene la bibliografía de la tesis.

2. Fundamentos teóricos de la investigación

Antes de comenzar con el análisis teórico y estado del arte en el que se enmarca el problema planteado anteriormente, es imprescindible definir desde un inicio las bases conceptuales sobre las cuales se parte para realizar la investigación sobre el estado del arte, así como las referencias de investigación seleccionadas para estudiar la detección y clasificación de objetos, concretamente, en el ámbito de la detección de armas. Si se parte de la abstracción de la problemática existente se puede definir de una forma básica tal y como se muestra en la Figura 1. El primer proceso es la detección y localización del objeto dentro de la imagen, dicho de otra forma, extraer las coordenadas de los datos que componen dicho objeto. El segundo, la clasificación del objeto o grupo de objetos. Por último, obtener una lista de porcentajes que indica con qué probabilidad los objetos detectados se asemejan a las clases que pertenecen.

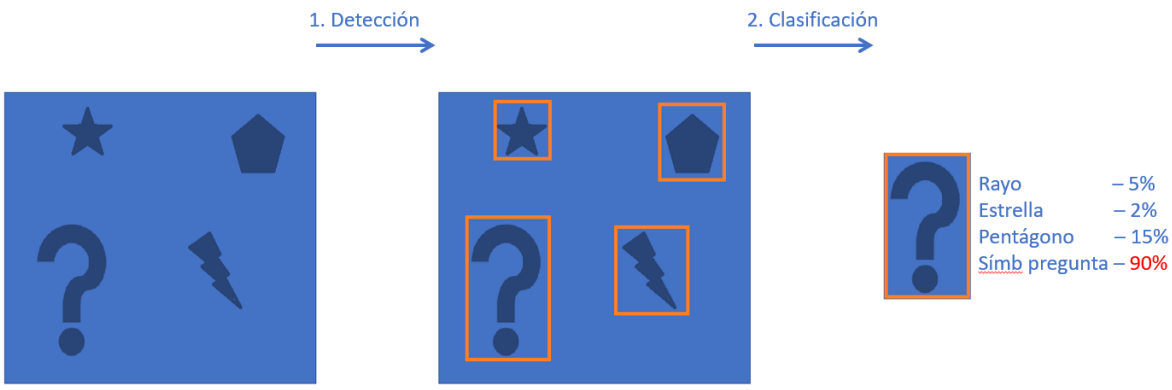

Figura 1 - Detección y Clasificación de Objeto.

Esta investigación se centra en analizar las estructuras y técnicas, así como fundamentos teóricos y posibles ramas que aportan un avance en lo que respecta al aprendizaje automático, para clasificar un objeto dentro un conjunto de clases, así como del estudio de las posibles técnicas existentes para la detección de un objeto en una imagen. El estudio sobre el estado del arte se centra en buscar un método generalizado, que ofrezca una solución versátil que se amolde a los posibles diferentes parámetros de objetos y casos de usos que puedan aparecer en una imagen para la detección y clasificación de un objeto.

El estudio del estado del arte se realiza sobre dos campos de estudio diferentes e independientes (detección y clasificación) que al ser combinados proporcionan una solución a la problemática que se plantea. Puesto que el comienzo del aprendizaje automático se basó en la idea de poder clasificar objetos por sus características, el análisis en el marco teórico se centra en los modelos básicos usados para la clasificación de objetos para así ofrecer desde un comienzo una sólida base de conocimientos que permitan entender la lectura sobre la evolución del estado del arte llegando al estado del arte en las investigaciones que incluyen la detección y segmentación de objetos en imágenes unidas a su clasificación.

2.1 Marco Teórico

Partiendo primeramente desde el problema de la clasificación de un objeto, desde hace varias décadas existe un extenso abanico de métodos que proporcionan solución a la clasificación de objetos, principalmente a través de la extracción de características visuales que implementan una robusta solución, aunque acarreando un mayor coste de computación y complejidad, como los métodos para extraer características visuales. Dos ejemplos serían SIFT [1] y HOG [2] aplicados a la detección de humanos. Estas técnicas parten de la extracción de características para identificar asociaciones complejas entre diferentes imágenes y así poder clasificar objetos. En las pasadas décadas se ha realizado un uso intensivo de este tipo de técnicas en la materia, aunque desde el año 2013 las mejoras realizadas en este tipo de técnicas según muestran los resultados de las competiciones en análisis de imágenes PASCAL VOC[1] ha decaído en gran número [3].

2.1.1 Perceptrón

En 1943, el afán de dos científicos, Warren McCulloch y Walter Pitts de simular el pensamiento humano dio origen a las bases teóricas de la inteligencia artificial [4], lo que le sirvió al psicólogo e informático Frank Rosenblatt como inspiración y fundamento para así dar origen a lo que se conoce hoy día como la primera neurona artificial o “Perceptrón”.

[1] http://host.robots.ox.ac.uk/pascal/VOC/voc2012/

Como se muestra en la Figura 2, un perceptrón está compuesto por un conjunto de entradas binarias a las que se le asigna un peso *w*, que pasa seguidamente a una unión sumadora y finalmente a través de una función de activación que genera un valor de salida [5].

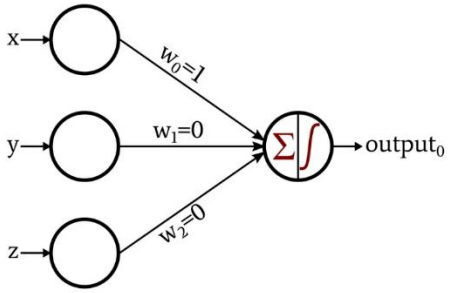

Figura 2 – Perceptrón **[6]**.

En términos matemáticos, la no linealidad de la neurona artificial en la que se basa el perceptrón es:

$$h\,(x) = \begin{cases} 1 \;\; if \quad w_0 \cdot x \;+\; w_1 \cdot y \;+\; w_2 \cdot z \;+ \cdots + w_d \cdot d \;\; x \geq 0 \\ 0 \;\; if \quad w_0 \cdot x \;+\; w_1 \cdot y \;+\; w_2 \cdot z \;+ \cdots + w_d \cdot d \;\; x < 0 \end{cases} \tag{1}$$

2.1.2 Red Neuronal

A mediados de los 80, apareció el concepto de “Perceptrón Multicapa”, que consiste en un conjunto de capas formadas por más de un perceptrón en las que los valores de salida son propagados hacia adelante a las siguientes capas. Esta nueva idea está compuesta por una capa de entrada y otra de salida. Algunas de las aplicaciones de estas técnicas realmente interesante y de reciente relevancia es su uso en medicina para reconstruir imágenes por

resonancia magnética [7] o la Modelización de la propagación de la infección por COVID-19 mediante un perceptrón multicapa [8].

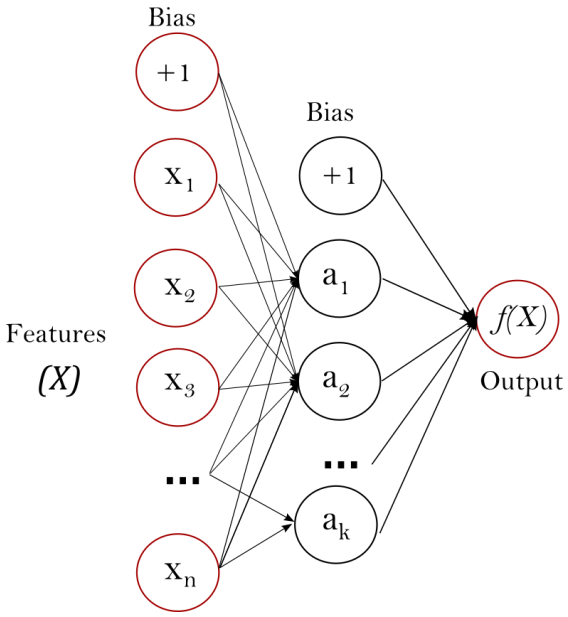

Figura 3 – Perceptrón Multicapa **[9]**.

Uno de los primeros avances en la detección de objetos basada en redes perceptrón multicapa (Figura 3) fue la conocida red de "*neocognitron*" para la detección de patrones, una red de jerárquica en las que los valores son trasladaos a la siguiente capa [10]. Esta técnica fue utilizada en el reconocimiento de dígitos escritos a mano [11] o para identificar anomalías cerebrales a partir del electroencefalograma [12]. Aunque desde hace décadas parece no ser muy relevante dentro de las aplicaciones en detección de objetos puesto que no existe un gran número de referencias recientes dentro de la comunidad de investigadores.

Casi una década más tarde se introdujo uno de los conceptos más importantes que cambiaría el curso del aprendizaje automático, mediante la invención del concepto de *backpropagation* o propagación hacía atrás del error por parte de Rumelhart, Hinton, y Williams, lo que hizo posible entrenar redes neuronales de múltiples capas de manera supervisada. El proceso que realiza este algoritmo de aprendizaje es minimizar el error mediante el método de descenso del gradiente. La red con una entrada se propaga a través de las siguientes capas hasta llegar a la capa de salida, en el siguiente paso se compara el

valor deseado con el que se recibe y se calcula un error que se propaga hacia atrás reajustando los pesos de las capas ocultas. Este método supuso un antes y un después en el campo de los algoritmos de aprendizaje automático [13]. Unas de las notables investigaciones de aquella época fue la realizada por Michael C. Mozer, que permitió detectar patrones temporales en series de datos de una manera eficiente centrándose en la señal de propagación hacia atrás [14].

2.1.3 Red Neuronal Convolucional

No fue hasta el año 1989 que se introdujo el concepto de Red Neuronal Convolucional o *Convolutional Neural Network* (CNN), con el que se demostró que era posible aplicar el algoritmo de propagación hacia atrás en aplicaciones del mundo real como por ejemplo reconocer dígitos escritos a mano [15]. La idea detrás de esta técnica consistía en reducir el número de parámetros libres en la red sin reducir su potencial.

Las CNN tienen un tremendo potencial para aprender características abstractas y distinguir objetos productivamente. Existen tres formas de capas convolucionales en una red neuronal: *convolutional layer*, *pooling layer*, *fully connected layer*.

En su siguiente trabajo, LeCun introdujo formalmente el concepto de CNN y su red conocida oficialmente como LeNet-5.

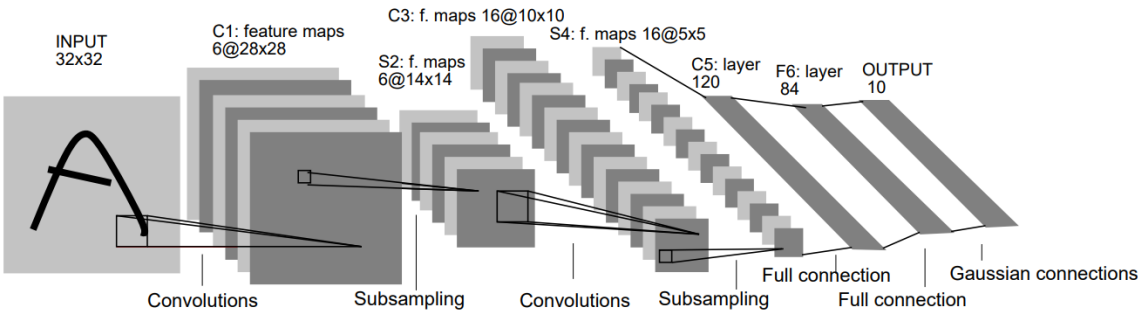

Figura 4 – LeNet-5 **[16]**.

Debido a que fue uno de los grandes avances en el aprendizaje automático, es de vital importancia explicar su estructura. Como se puede ver en la Figura 4, LeNet-5 se compone de siete capas, en las que tres de ellas son capas completamente convolucionales, dos de ellas son capas de submuestreo y dos capas totalmente conectadas [16]:

- Capa *input* 32x32: Esta capa en teoría no realiza ningún aprendizaje, solo se encarga de aceptar imágenes de un tamaño de 32 por 32 pixeles, su cometido simplemente es alimentar con datos a la siguiente capa.
- C1, C3, C5 *feature maps*: Mapa de características generando un conjunto de mapas de características tras aplicar un *kernel*.
- S2, S4 *maps*: reduce la dimensión del mapa de características.
- F6 *Fully Connected* que genera finalmente aplicando una función *Softmax* una salida con 10 valores.

Desde entonces, la red CNN has sido extensamente utilizada para muchas aplicaciones, aunque su uso aminoró tras la aparición de la técnica de clasificación Máquinas de Vector Soporte o *Supported Vector Machine* (SVM) [17]. La idea detrás de esta técnica es la de separar los datos de entrenamiento en dos grupos sin error, lo más alejados posible representados a través de puntos en un hiperplano. Este método fue extensamente utilizado en muchos campos con un gran éxito lo que hizo atraer la atención de la comunidad científica como por ejemplo, siendo uno de los métodos más utilizados para aplicaciones biológicas para detectar patrones en secuencias biológicas y clasificar pacientes en base a su perfil genético [18], o también por otro lado para resolver problemas matemáticos para la clasificación de patrones [19].

No fue hasta el año 2012 cuando Krizhevsky, con la colaboración de otros investigadores relevantes en la inteligencia artificial, que realizaron unas modificaciones en la red LeNet-5 con las que obtuvieron resultados realmente prometedores con el dataset “ImageNet Large Scale Visual Recongnition Chanlenge (ILSVRC) [20]. Las aportaciones de estos autores con respecto a la técnica propuesta por LeCun fue el entrenamiento mucho más rápido de las redes neuronales, puesto que, en lugar de utilizar una función tangente para modelar la salida de la neurona, utilizaron ReLUs (*Rectified Linear Units*). Por otro

lado, implementaron en su propuesta la técnica de "*dropout*", que consiste en poner a cero la salida de cualquier neurona con probabilidad 0.5 para que no participe en la propagación de valores hacia atrás en la red. Fruto de esos avances, CNN comenzó a dar resultados para detectar caras y personas [21].

En lo que se refiere a la clasificación, para distinguir entre otras categorías de objetos, las técnicas utilizadas conocidas más relevantes aparte de SVM son AdaBoost [22] y DPM [23] entre otras. Sin embargo, como se menciona anteriormente, se han obtenido pocas mejoras en los últimos años que hagan relevante seguir esa línea de investigación y análisis, tal y como ha ocurrido en el transcurso y evolución de otras técnicas.

2.2 Estado del arte

Tras presentar en el apartado anterior el marco teórico en el que se presentan las bases teóricas de clasificación a partir de las cuales se desarrolla el estado del arte, este apartado aborda el estudio de las técnicas y modelos utilizados para la detección de objetos y en especial de armas. Hay varios modelos que han sido utilizados para la detección de objetos, sin embargo, tras analizar las investigaciones realizadas en el campo de estudio, el primer modelo utilizado para la detección de armas es el modelo Faster R-CNN. Para poder entender el modelo Faster R-CNN es necesario explicar brevemente los modelos predecesores a este con el fin de entender la evolución que ha llevado al diseño y mejoras de ese modelo para la detección de objetos y finalmente su aplicación en la detección de armas.

2.2.1 R-CNN

R-CNN está basada en la CNN presentada por LeCun explicada en el apartado 2.1.3 aunque con unas ligeras modificaciones y mejoras. R-CNN se basa en elegir un número de regiones posibles como objetos, concretamente una técnica llamada búsqueda selectiva [24]. Esta técnica se basa en extraer un máximo de 2000 posibles regiones de objetos e intentar clasificar esas posibles regiones generadas a través de tres pasos:

- Generar posibles segmentaciones candidatas como objeto.

- Utilizar el algoritmo de Greedy de supresión no máxima para recursivamente combinar en mayores regiones un objeto.
- Utilizar las regiones generadas para producir candidatos finales.

Desde la publicación de este modelo en el año 2014, se ha realizado diferentes aplicaciones como por ejemplo su uso para la detección de pequeños objetos en imágenes siendo una gran desventaja el análisis de una gran cantidad de pequeños objetos [25]. Otra interesante aplicación que poco a poco se puede ir acercando a la detección de armas es uso de dos canales de R-CNN para la detección de acciones en videos, en este trabajo se utiliza dicho modelo para la extracción especifica de la región propuesta para así poder analizar con las posteriores imágenes el estudio de una posible acción en común entre las diferentes imágenes [26]. Otra interesante aplicación del modelo R-CNN para la detección de una figura en una imagen es su aplicación para la detección de peatones, un problema crucial en lo que respecta a los resultados ofrecidos y que demuestra el gran potencial que desde un principio este modelo demostró para la detección de objetos en imágenes [27].

2.2.2 Fast R-CNN

Es de suponer que el generar y evaluar 2000 regiones de posibles objetos conlleva un alto coste computacional, motivo por el que mismo creador de R-CNN fue más allá hasta crear un nuevo modelo mejorado llamado Fast R-CNN superando con creces a los modelos R-CNN y SPPnet [28]. En comparación con trabajos anteriores, Fast R-CNN emplea varias innovaciones notables para mejorar la velocidad de entrenamiento y de *testing*, al tiempo que aumenta la precisión en la detección de objetos lo que supone una enorme ventaja y avance respeto a la versión anterior.

Una de las mejoras más notables de Fast R-CNN es que no es necesario utilizar 2000 regiones a analizar cómo se ha expuesto anteriormente; en su lugar, el primer paso consiste en alimentar directamente las imágenes a una red CNN para generar un mapa de características de la imagen conocido como *"feature map"*. Un mapa de características es básicamente el resultado de aplicar diferentes filtros de diferentes tamaños para extraer las

características de una imagen como bordes, líneas y diferentes características que pueden ser interesantes en una imagen.

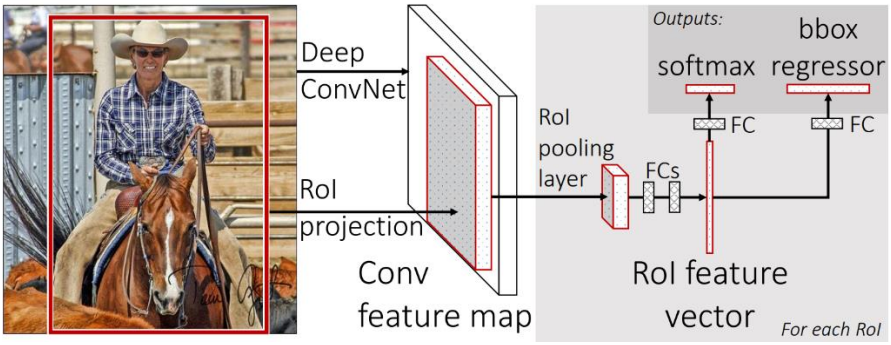

Figura 5 – Arquitectura Fast R-CNN **[28]**.

Es a partir de dicho mapa donde entra el juego la innovación de este modelo, llamada "*RoI pooling layer*" como se muestra en la Figura 5. Esta técnica consiste en identificar las regiones propuestas a través de un *max pooling*, para después dimensionarlas en cuadrados utilizando una capa de agrupación RoI (*Region of interest* - Figura 7) con el fin de crear datos de dimensiones fijas H x W, para finalmente alimentarlo a la red CNN VGG16 [28]. Fast R-CNN tiene la ventaja de entrenar la red VGG16 [29]

Una de las más importantes arquitecturas utilizadas desde 2014 para la detección de objetos es la red VGG16 (Figura 6), la cual sigue siendo hoy en día una de las más utilizadas aún en vanguardia en la clasificación de objetos. La red tiene la ventaja de que no tiene un gran número de *hyper-parametros*, es decir que no contiene un gran número de parámetros fijos permitiendo ser más versátil. Esto es, que se enfoca en utilizar 16 capas convolucionales en total que se reparten en capas con filtros de 3x3, capa *max-pooling* con filtros de 2x2 concatenadas, hasta llegar finalmente a dos capas totalmente conectadas y una función *softmax* al final para proporcionar una salida.

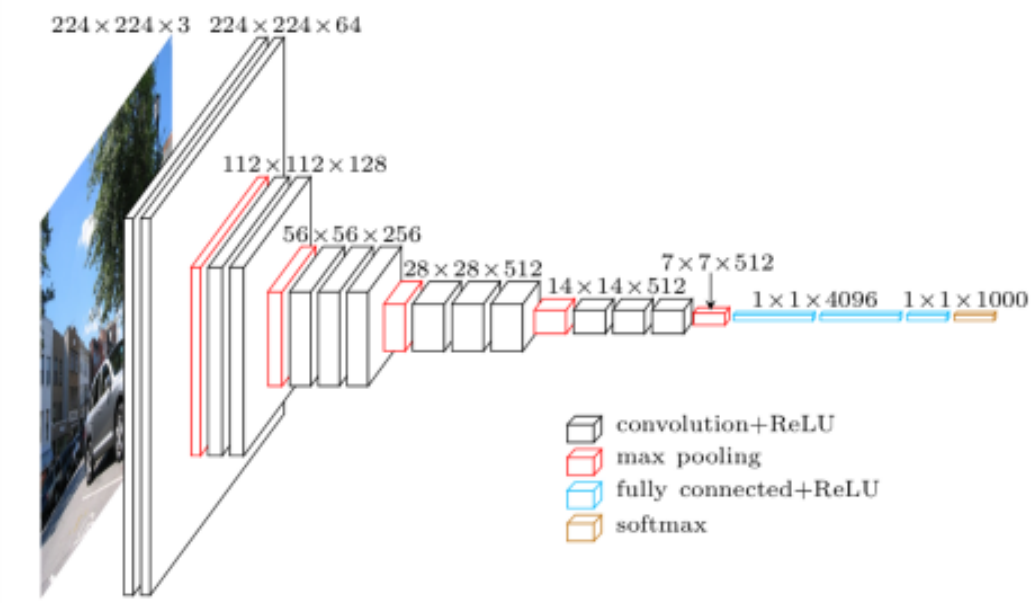

Figura 6 – Arquitectura VGG16 **[30]**.

Fast R-CNN tiene la posibilidad de entrenar 9 veces más rápido que R-CNN, es 213 veces más rápido en el momento de la prueba y logra un mAP[2] más alto en la competición PASCAL VOC 2012 [31] [28]. La mejora más notable de esta técnica respecto a la anterior es la posibilidad de obviar el análisis de 2000 regiones ya que, en definitiva, muchas de ellas suelen solaparse, y a su vez producen un coste extra de computación el cual se puede evitar gracias al analizar el mapa de características para extraer las regiones de interés como muestra la Figura 8.

[2] mAP (mean Average Precision) medición en el campo de la detección y clasificación de objetos que se utiliza para evaluar la precisión de modelos al aplicar un test dataset, sus valores van de 0 a 1.

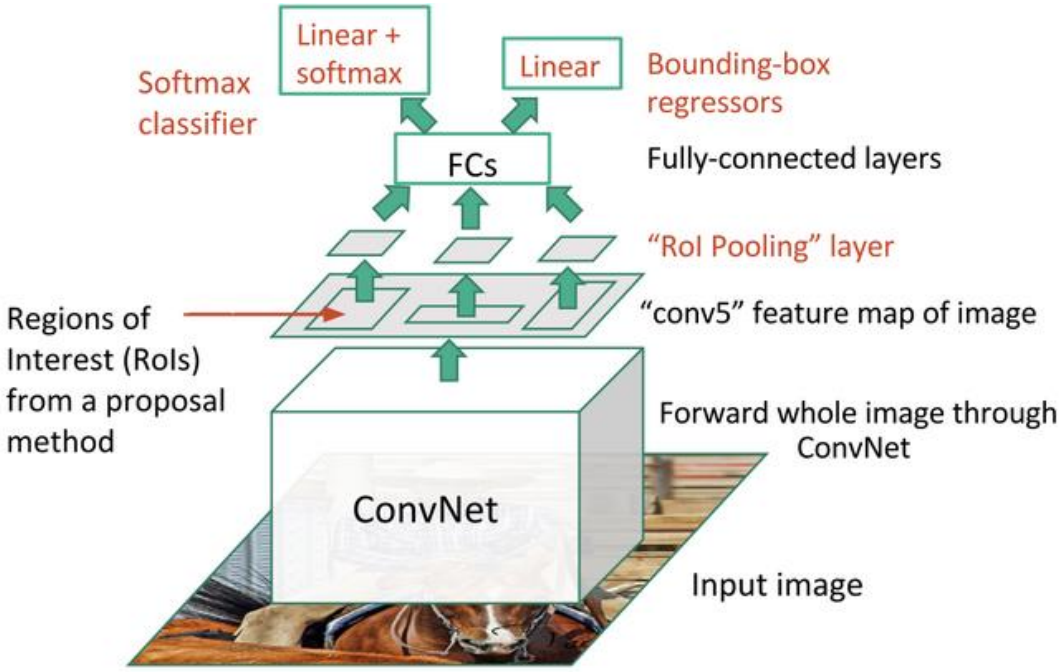

Figura 7 - Region of Interest **[32]**.

Como se mencionó en el anterior apartado, la cantidad de aplicaciones de estos modelos es muy extensa y puede variar la temática en la que se utiliza demostrando el potencial que ofrece. Una interesante aplicación de este modelo es en la detección de rostros proporcionando hasta un 91.87% de resultados en el *"Face Detection Data Set and Benchmark Home"* [33]. Otra interesante aplicación es el uso de este modelo en imágenes bajo el agua lo que hace aún más remarcable la capacidad y adaptabilidad que este modelo ofrece para la detección y clasificación por ejemplo de peces en imágenes bajo el agua llegando a producir un 81.4% de aciertos a lo largo 12 clases diferentes de peces [34], o por otro lado los interesantes resultados que muestra este modelo de hasta el 85.58% de acierto en la detección de señales en la carretera [35].

2.2.3 Faster R-CNN

En 2015, se publicó una nueva versión de Fast R-CNN llamada Faster R-CNN en la que los autores introducen una Red de Propuesta de Regiones (RPN) como muestra la Figura 9 [36]. Esta nueva red se enfoca en aprender a ofrecer propuestas de región de calidad para así alimentar con dichas propuestas o posibles objetos a la red Fast R-CNN. Uno de los

objetivos de los investigadores es reducir el número de regiones para así acelerar el proceso de clasificación, evitando operaciones en regiones innecesarias o que ya han sido comprobadas. Todo ello a través de una red que se encarga de aprender a ofrecer regiones.

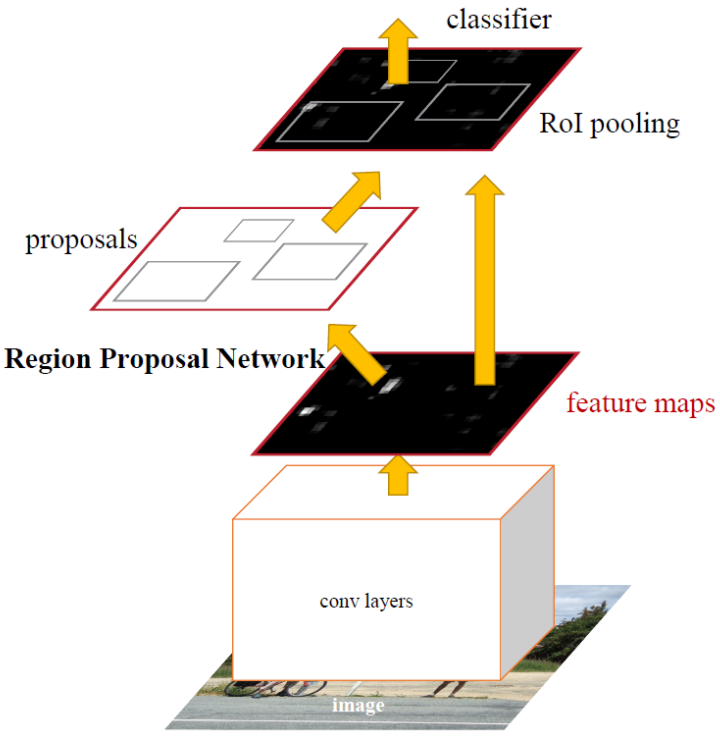

Figura 8 - Faster R-CNN **[36]**.

La RPN es una red neuronal convolucional que se aplica sobre la última capa del mapa de características. Esta red neuronal acepta una entrada de una caja de dimensiones n x n. Cada caja es analizada por dos capas, una capa de regresión y una de clasificación. El número máximo de propuestas para cada caja se denota por k, por lo que la capa de regresión proporciona 4k posibles coordenadas de un número de propuestas k, y la capa de clasificación un número de 2k, que consiste en la probabilidad de si una propuesta es un objeto o no. A cada propuesta se le llama según los autores "*anchors*", que en definitiva son cajas o cuadros como propuestas de objetos.

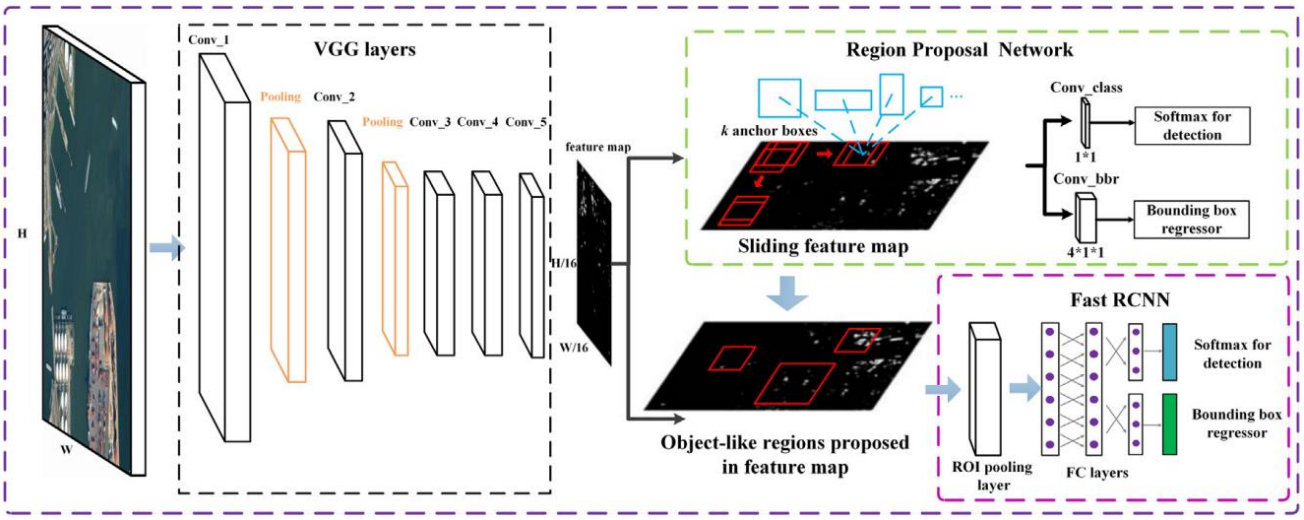

Figura 9 – Arquitectura Faster R-CNN **[37]**.

Faster R-CNN (Figura 9) ha sido durante varios años y sigue siendo hoy en día uno de los modelos de inteligencia artificial más extensamente utilizados para la detección de objetos en tiempo real a la vez que consigue una notable precisión para pequeños objetos. Este modelo muestra un gran avance en la detección de objetos de ahí su extenso y activo uso en un gran número de aplicaciones, dejando prácticamente como obsoletos los otros modelos anteriormente presentados. Uno de los campos más notables en los que se aplica este nuevo modelo es en aplicaciones de detección de objetos en tiempo real como por ejemplo la detección de vehículos a través del entrenamiento de este modelo con el *dataset* de vehículos KITTI alcanzando unos resultados de 95.14% de aciertos [38]. Otra aplicación realmente relacionada con la detección de objetos en tiempo real, así como de crucial importancia ya por su sensibilidad en los resultados es la detección de peatones obteniendo unos resultados del 30% para el *Caltech Pedestrian Dataset* [39].

Es tras la introducción de este modelo cuando se comienzan a surgir proyectos de inteligencia artificial para la detección de armas, con anterioridad a estos modelos, se utilizaban algoritmos de fuerza bruta para la detección de formas, características y objetos los cuales están muy obsoletos como para incluirlos en el estudio de la temática de este trabajo.

Es llegado a este punto cuando es de interés enlazar todo el desarrollo anterior con el tema en cuestión de este trabajo. Unos de los primeros grupos de investigadores interesados en aplicar técnicas de visión artificial para la detección de armas fueron Roberto Olmos, Siham Tabika y Francisco Herrera, de la Universidad de Granada. Como muestran en su trabajo [40], son pioneros en utilizar el modelo Faster R-CNN para la detección de armas en vídeos de vigilancia. Otros autores también vieron el potencial de aplicar este modelo a la detección de armas utilizando el *dataset Internet Movie Firearms Database* obteniendo un porcentaje de acierto de un 84.26% lo que demuestra el potencial de este modelo [41]. Aplicación interesante es la que se muestra el siguiente trabajo en el que el modelo se entrena con el fin de detectar también cuchillos aparte de armas obteniendo resultados de 85.46% para armas y 46.8% para cuchillos con un *dataset* que seguidamente se mencionará y utilizará para entrenamiento del modelo propuesto en este trabajo [42].

Más tarde, los mismos autores de la Universidad de Granada y basado en los primeros resultados obtenidos en su trabajo de 2018, se centró en analizar las razones por las cuales objetos pequeños eran detectados como falsos positivos, especialmente objetos cuya manera de sostenerlos era similar a la de un arma de fuego. Es por ello por lo que introdujeron una técnica llamada "ODeBiC", acrónimo de *Object Detection with Binary Classifiers based on deep learning*. Esta técnica consiste en utilizar el potencial que ofrecen las técnicas de binarización como OVA y OVO [43], para mejorar la detección de objetos pequeños [44]. Para analizar dicho modelo, crearon una base de datos con la que entrenan y evalúan el modelo, la cual será utilizada para entrenar el modelo propuesto en este trabajo.

Otros autores también mostraron interés en analizar el mismo problema que algunos investigadores ya habían estudiado, pero aplicando distintos modelos para la extracción del mapa de características [45], hasta ahora el uso común era de la arquitectura VGG16 sustituyéndola por otras como Inception-ResNetV2, MobiletNetV2, ResNet-50 aunque utilizando un el *dataset* estándar de armas mencionado anteriormente (*Internet Movie Firearms Database*) obteniendo los siguientes resultados que muestran la Tabla 1.

CNN Architecture	Resultados
	Precisión media de la prueba (mAp)
Faster R-CNN	73%
Faster R-CNN(Inception-ResNetV2)	81%
Faster R-CNN(VGG16)	72%
Faster R-CNN(MobileNetV2)	70%
YOLOv3(ResNet50)	76%

Tabla 1 - Media de precisión entre diferentes estructuras base empleadas en Faster R-CNN **[45]**.

Otra aplicación algo distinta pero que también utiliza este modelo para la detección de armas es un trabajo en el que el *dataset* o imágenes que se utilizan son de rayos equis [46].

Ren y col., los investigadores que desarrollaron el modelo Faster-RCNN, han mostrado a lo largo de sus investigaciones los grandes resultados de sus innovaciones en lo que respecta a la detección y clasificación de objetos, lo que lleva a incitar a seguir los trabajos de dichos autores en lo que respecta a la idea de propuesta por regiones, que aporta una gran rapidez y robustez a los modelos de redes multicapa en su aprendizaje [36].

También es argumento de peso el uso de este modelo para la detección de armas, puesto que se han demostrado los notables resultados obtenidos a lo largo de diferentes investigaciones en los años anteriores, tanto en la detección más estándar de objetos, como pueden ser vehículos o peatones, así como los sorprendentes resultados en la detección de armas. Todo ello, incita a seguir analizando el modelo planteado o posibles nuevos modelos que puedan ofrecer unas mejores prestaciones.

Como consecuencia del estudio del estado del arte llevado a cabo y expuesto a lo largo de este capítulo, en este trabajo se ha decidido analizar la viabilidad del modelo Mask R-CNN, aplicado al ámbito de la detección de armas en videovigilancia. Dicho modelo

supone una evolución de los anteriores, por lo que los resultados que refleje la aplicación de dicho modelo supondrán una aportación a este campo de investigación.

Algunas de las claras mejoras o variaciones que el modelo Mask R-CNN presenta respecto al modelo Faster R-CNN son:

- La mayor evolución del modelo se concreta en añadir una hebra nueva para predecir la máscara del objeto en paralelo con la hebra existente para el reconocimiento del cuadro delimitador o caja del objeto.
- En lugar de utilizar la arquitectura base neuronal VGG16 para extraer el mapa de características se utiliza la estructura ResNet-101.
- En lugar de utilizar *RoiPool* como se ha utilizado en Fast y Faster R-CNN para extraer las propuestas de regiones como objetos, se ha implementado una nueva técnica llamada *RoiAlign* que muestra mejores resultados, según los creadores de Mask R-CNN.

Desde la publicación de este modelo, el número de publicaciones ha aumentado enormemente sobre la detección y clasificación de objetos, pues posee un enorme poder de precisión, utilizándose para aplicaciones en medicina, defensa, así como un gran número de nuevas aplicaciones. Una interesante aplicación del modelo Mask R-CNN es para la detección de barcos a través de una pequeña modificación llamada "*Soft-Non-Maximum Suppression*" al final de la capa de clasificación y producción de las cajas de objetos, produciendo unos resultados del 87.74% en comparación con el 79.90% de Mask R-CNN [47]. Una aplicación más que resalta la importancia que este método ha despertado en los investigadores es su uso en la delicada y sensible detección de tumores de mamas en ecografías obteniendo el impresionante resultado del 85% de precisión [48].

En el siguiente capítulo se estudia a fondo la estructura de dicho modelo y la viabilidad de su aplicación a la detección de armas, siendo esta la principal aportación de este trabajo de fin de máster.

3. Solución propuesta

3.1 Modelo Mask R-CNN

Como se ha explicado anteriormente y bajo las conclusiones extraídas en los apartados anteriores, el modelo Mask R-CNN es el que se aplica en este trabajo para la detección de armas. Existen varias razones por las que utilizar dicho modelo es ventajoso que seguidamente serán expuestas con detalle, sin embargo, el argumento con más peso existente para utilizar este modelo es la innovación que aporta utilizar un nuevo modelo mejorado y diferente a los anteriores para la detección de armas. El uso de un nuevo modelo en una línea de investigación promueve a realizar mejoras y modificaciones para futuras investigaciones.

De forma un tanto abstracta, Mask R-CNN consiste en tres flujos o hebras de detección, las dos primeras etapas se basan en el modelo Faster R-CNN que contiene dos salidas, una para la etiqueta de clase u objeto y la segunda para el cuadro delimitador. La tercera es encargada de proporcionar la máscara correspondiente a cada objeto clasificado.

Como para todos los modelos, Mask R-CNN posee una función de perdida a optimizar, para Mask R-CNN es:

$$L = L_{class} + L_{box} + L_{mask} \tag{2}$$

Donde $L_{class} + L_{box}$ provienen del modelo Faster R-CNN. Como se muestra en [48] las funciones de pérdida de Faster R-CNN se definen como:

$$L_{class} + L_{box} = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \frac{1}{N_{box}} \sum_i p_i^* L_1^{smooth}(t_i - t_i^*) \tag{3}$$

Donde L_{cls} se desarrolla como:

$$L_{cls}\{(p_i, p_i^*)\} = -p_i^* \log p_i^* - (1 - p_i^*) \log(1 - p_i^*) \tag{4}$$

Y donde finalmente L_{mask} es la pérdida media de entropía cruzada binaria:

$$L_{mask} = -\frac{1}{m^2}\sum_{1\le i,j\le m}\left[y_{ij}\log o\, y_{ij}^{k} + (1-y_{ij})\log\left(1-o\, y_{ij}^{k}\right)\right] \quad (5)$$

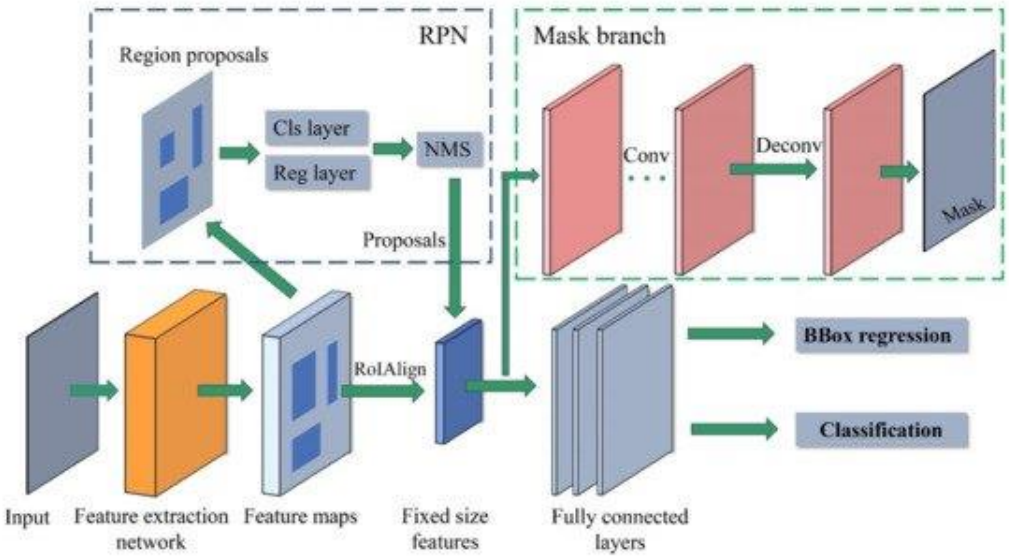

Figura 10 – Arquitectura Mask R-CNN **[49]**.

Si se observan las Figuras 9, 10, se puede apreciar cómo aparece una parte inicial y de real importancia como es la red de extracción de características. Sin esta parte la red no sabe realmente dónde hay partes significativas en la imagen que pueden dar a la existencia de un posible objeto. En los modelos Fast R-CNN y Faster R-CNN se utiliza una red VGG-16 la cual se encarga de extraer las características de la imagen como se ha explicado anteriormente. Sin embargo, como los autores explican [50], este tipo de redes a veces generan "residuos de aprendizaje". A veces el valor de aprendizaje que se propaga a lo largo de la red es siempre el mismo, lo que no genera ningún beneficio, solo coste computacional. Tras analizar dicho problema en las redes utilizadas para la extracción de características, los autores crearon un nuevo modelo para la extracción de características de una imagen llamado ResNets. Existen diferentes versiones, pero la utilizada en Mask R-CNN e implementada en este trabajo es la llamada capa ResNet-101 puesto que obtiene mejores resultados comparado a otras arquitecturas Tabla 2. y como se muestra en Tabla 3 [50].

Nombre de capa	**Tamaño salida**	**18-capas**	**34-capas**	**50-capas**	**101-capas**	**152-capas**
conv1	112x112	7x7, 64, progeso 2				
conv2_x	56x56	3x3 max pool, progreso 2				
		$\begin{bmatrix} 3x3, & 64 \\ 3x3 & 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3x3, & 64 \\ 3x3 & 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 1x1, & 64 \\ 3x3 & 64 \\ 1x1 & 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1x1, & 64 \\ 3x3 & 64 \\ 1x1 & 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1x1, & 64 \\ 3x3 & 64 \\ 1x1 & 256 \end{bmatrix} \times 3$
conv3_x	28x28	$\begin{bmatrix} 3x3, & 128 \\ 3x3 & 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3x3, & 128 \\ 3x3 & 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1x1, & 128 \\ 3x3 & 128 \\ 1x1 & 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1x1, & 128 \\ 3x3 & 128 \\ 1x1 & 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1x1, & 128 \\ 3x3 & 128 \\ 1x1 & 512 \end{bmatrix} \times 8$
conv4_x	14x14	$\begin{bmatrix} 3x3, & 256 \\ 3x3 & 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3x3, & 256 \\ 3x3 & 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1x1, & 256 \\ 3x3 & 256 \\ 1x1 & 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1x1, & 256 \\ 3x3 & 256 \\ 1x1 & 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1x1, & 256 \\ 3x3 & 256 \\ 1x1 & 1024 \end{bmatrix} \times 36$
conv5_x	7x7	$\begin{bmatrix} 3x3, & 512 \\ 3x3 & 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3x3, & 512 \\ 3x3 & 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1x1, & 512 \\ 3x3 & 512 \\ 1x1 & 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1x1, & 512 \\ 3x3 & 512 \\ 1x1 & 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1x1, & 512 \\ 3x3 & 512 \\ 1x1 & 2048 \end{bmatrix} \times 3$
	1x1	Average pool. 1000d fc, softmax				
FLOPs		$1.8x10^9$	$3.6x10^9$	$3.8x10^9$	$7.6x10^9$	$11.3x10^9$

Tabla 2 - Diferentes combinaciones posibles de la red ResNet-X **[51]**.

Net-depth-features	AP	AP 50	AP 75
ResNet-50-C4	30.3	51.2	31.5
ResNet-101-C4	32.7	54.2	34.3
ResNet-50-FPN	33.6	55.2	35.3
ResNet-101-FPN	**35.4**	**57.3**	**37.5**

Tabla 3 - La arquitectura troncal ResMet101-FPN demuestra los mejores resultados **[50]**.

3.1.1 ResNet-101

Este modelo se compone de 5 etapas:

1. **Etapa 1 conv1**: Esta primera parte se compone de una capa de convolución con un filtro de dimensión 64, un *kernel* de tamaño 7x7 y un *stride* de compresión de valor 2. Seguidamente contiene una función de activación *relu* acompañada finalmente por un *max pooling* de tamaño 3x3 y *stride* 2. En ANEXO A se puede ver la implementación detallada.

2. **Etapa 2 conv2**: Esta parte se compone de un conjunto de 3 bloques en los que cada bloque contiene tres capas de convolución como aparece implementado en ANEXO B:

    1) Bloque 1 convolucional (Implementación ANEXO C):
        - [1x1, 64]: Filtro de dimensión 64 y un *kernel* de 1x1 con normalización y función de activación *relu*.
        - [3x3, 64]: Filtro de dimensión 64 y un *kernel* de 3x3 con normalización y función de activación *relu*.
        - [1x1, 256]: Filtro de dimensión 256 y un *kernel* de 1x1 con normalización que va acompañado de una "red de atajo" la cual es una capa de convolución que es añadida como mejora de optimización como se explica en [51].

    2) Bloque 2. "Bloque de identidad": Dicho bloque no contiene una "red de atajo" como en el bloque anterior pero antes de añadir la función *relu* a la última capa se añade la red *input* para finalmente terminar con una función de activación (Implementación ANEXO D):
        - [1x1, 64]: Filtro de dimensión 64 y un *kernel* de 1x1 con normalización y función de activación *relu*.
        - [3x3, 64]: Filtro de dimensión 64 y un kernel de 3x3 con normalización y función de activación *relu*.
        - [1x1, 256]: Filtro de dimensión 256 y un kernel de 1x1 con normalización que va acompañado de una de la red input para terminar con función de activación *relu*.

    3) Bloque 3. Exactamente igual que el bloque anterior.

3. **Etapa 3 conv3**: Como se ve en la Figura 11, esta etapa se compone de 4 bloques con un total de 3 capas convolucionales cada bloque con filtros de dimensión

128, 128, 512 y los tamaños de los kernels permanecen a los de la etapa anterior (Ver implementación en ANEXO E).

4. **Etapa 4 conv4**: Esta etapa contiene un total de 23 bloques, uno de ellos es un bloque convolucional y el resto de ellos son 22 bloques de identidad. Los filtros utilizados para las capas son de dimensión 256, 256 y 1024, el tamaño de los kernels permanece igual (Implementación ANEXO F).

5. **Etapa 5 conv5**: La etapa final se compone de tres bloques, de los cuales dos de ellos son bloques identidad. Todos poseen en común filtros de dimensiones 512, 512 y 2048 con los mismo kernels que anteriormente se han utilizado (Ver implementación ANEXO G).

Una vez creada la red ResNet-101 se utilizan los bloques creados para extraer las zonas de interés a través de una Red Piramidal de Características hacia abajo [52] como muestra la implementación en ANEXO H.

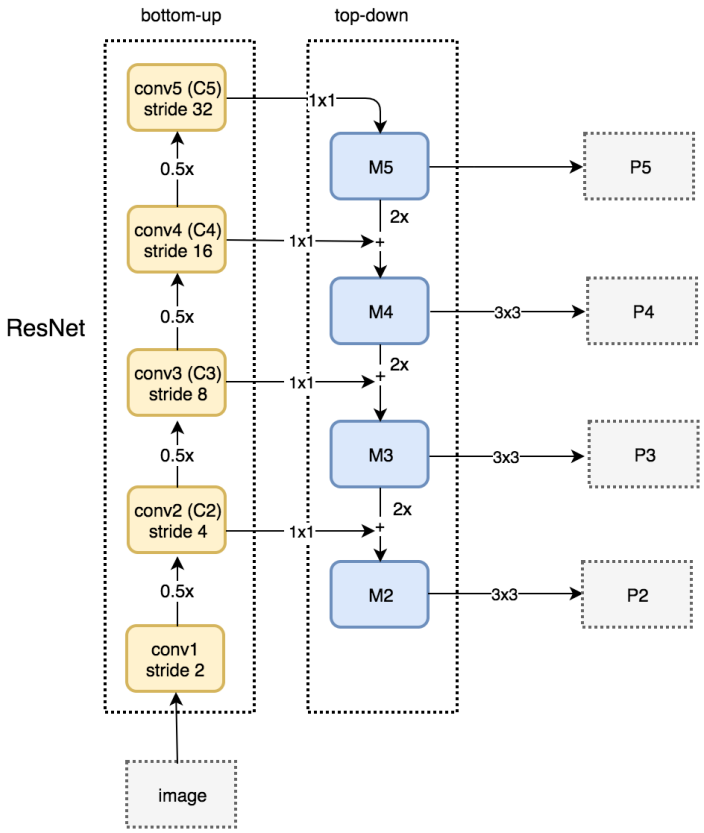

Figura 11- Red piramidal de características hacia abajo (Ver ANEXO H) **[53]**.

3.1.2 Region Proposal Network RPN

A continuación, la red ResNet-101 se conecta con el modelo "*Region Proposal Network*" ya expuesto en el modelo Faster R-CNN, aunque se analiza con más detalle y en profundidad en este apartado (Figura 12).

RPN es un complejo modelo que se encarga de generar las propuestas de región para los objetos de la imagen a través de dos capas, una de clasificación y otra de regresión. La clasificación genera la probabilidad de que la salida proporcionada sea un posible objeto. La red de regresión genera las coordenadas de dicho objeto.

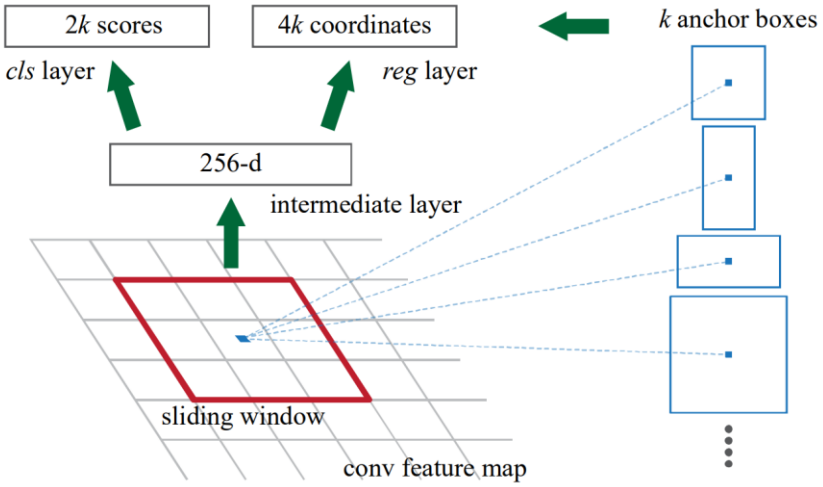

Figura 12 – Region Proposal Network **[36]**.

Una de las partes más cruciales de este modelo es la red convolucional que comparten tanto como la estructura final *rpn_box* y *rpn_probs*, que proporcionan la clasificación y regresión, las muestras positivas tienen IoU (intersección sobre unión) > 0,7 y las negativas IoU < 0,3. RPN desliza una pequeña ventana espacial n x n sobre el mapa de características de toda la imagen. En el centro de cada ventana deslizante, se predicen varias regiones de varias escalas y proporciones simultáneamente. Un *anchor* es una combinación de (centro de la ventana deslizante, escala, proporción). Por ejemplo, 3 escalas + 3 relaciones producen k=9 *anchors* en cada posición de deslizamiento. La base de ese modelo está en que se comparte la estructura '*shared'* desde un principio. Se utiliza kernel_size 3 de acuerdo con lo que sus creadores recomiendan (Ver ANEXO I).

RPN trabaja obteniendo la salida del modelo ResNet-101 y produce 3 salidas que son el clasificador, la regresión y una caja que contiene las coordenadas del objeto (propuesta de región), que más tarde alimentaran la siguiente red *RoIAlign*. Un paso importante de mencionar es que para cada capa de características (P2, P3, P4, P5, P6) se crea una red RPN.

3.1.3 RoIAlign

Una parte crucial y novedosa de este modelo es el aplicar en paralelo distintos métodos que simplifiquen el proceso de detección como es el caso de Fast R-CNN, que

aplica la clasificación del cuadro delimitador y de regresión en paralelo (Ver Figura 13). En este aspecto, Mask R-CNN se encarga de exactamente lo mismo en paralelo a través de una nueva capa llamada *RoIAlign*, que produce como resultado una máscara binaria para cada RoI. Esta es otra de las variaciones e innovaciones respecto a modelos anteriores, por ejemplo, al uso de RoI Pooling en Faster R-CNN. Una de las ventajas de RoIAlign es que no pierde datos en el proceso puesto que no utiliza cuantificación, sino que aplica interpolación bilineal [54] en cuatro lugares de muestras en cajas iguales que dividen un cuadro propuesto para finalmente agregar el resultado a un tamaño fijo (Ver Figura 13).

Desde que se propuso en 2014 el modelo Fast R-CNN se ha refinado y mejorado en muchos aspectos los modelos para la región de interés como *RoIPool* o *RoIWarp*, aunque en este modelo se implementa *RoIAlign* el cual ofrece una precisión mucho mejor comparado a las otras técnicas.

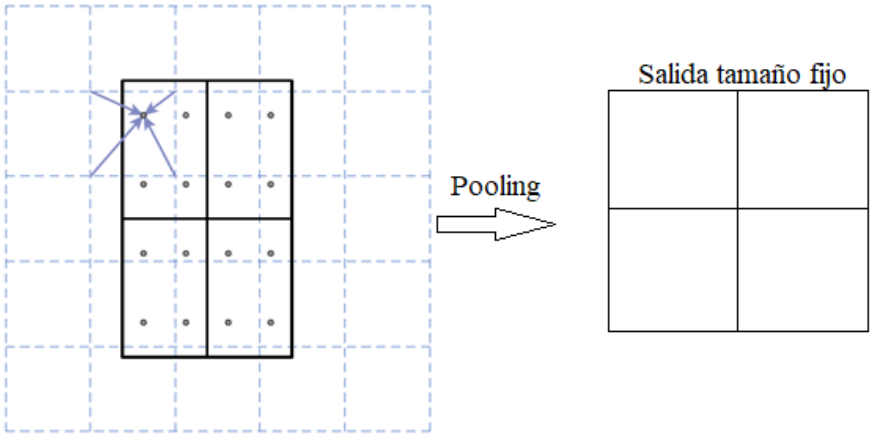

Figura 13 – RoIAlign.

Algo realmente interesante de este modelo y que lo hace realmente especial es durante su entrenamiento, cada muestra RoI contiene una función de perdida que lo hacen ser mucho más preciso en la detección de objetos (Ver ANEXO J y ANEXO K)

3.1.4 Feature Pyramid Network

- **Clasificador y Regresor**

Utilizando la estructura piramidal diseñada anteriormente como base, uno de los últimos pasos es generar el clasificador y regresor final. Como se puede ver en la

implementación en el ANEXO L, se construye una pirámide con una arquitectura "*top-down"* (Ver Figura 14 – **"*Feature Pyramid Network*"** .) con conexiones laterales que comparten la capa "*shared*" a cada nivel, generando así una clasificación y regresor [52] [55]).

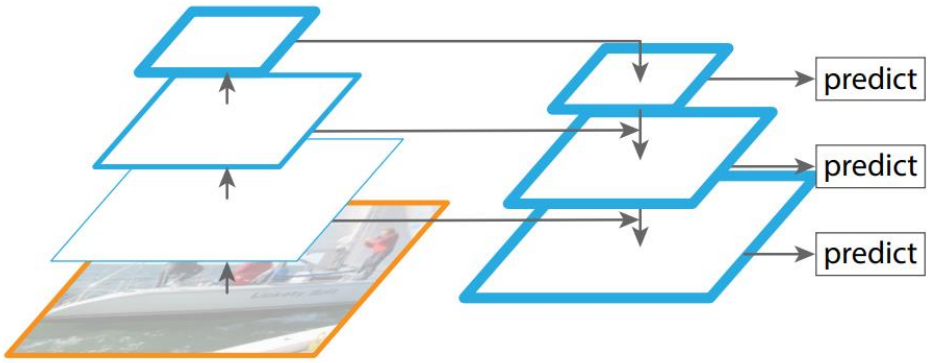

Figura 14 – "*Feature Pyramid Network*" **[55].**

- **Máscara**

Esta última capa es la encargada de generar la lista de puntos que componen la máscara del objeto detectado. Esta capa también está basada en la capa implementada anteriormente con RoIAlign como se puede ver en ANEXO M. Básicamente explicado de una forma más simple, esta parte realiza la computación binaria de cada punto de la capa a través de una función *sigmoid* con el fin de comprobar si el punto pertenece a la capa o no, es decir, proporciona una lista de valores booleanos.

3.2 Frameworks utilizados

3.2.1 Tensorflow

TensorFlow es un entorno de trabajo desarrollado en Python para Aprendizaje Automático. De código abierto y desarrollado por Google, proporciona un ecosistema completo y flexible de herramientas, bibliotecas y recursos que permiten a los desarrolladores avanzar en el estado del arte de *Machine Learning*. También, permite crear y desplegar aplicaciones fácilmente, proporciona robustez a la hora de desarrollar y entrenar modelos de inteligencia artificial [56].

3.2.2 Keras

Keras es una API de aprendizaje profundo (*Deep Learning*, en terminología inglesa) escrita en Python, que se ejecuta sobre la plataforma de aprendizaje automático TensorFlow. Se desarrolló con el objetivo de permitir una experimentación rápida y poder pasar de la idea al resultado lo más rápido posible, parte clave para hacer una buena investigación [57].

Con el fin de que el lector pueda entender mejor el código de los siguientes apartados, se muestra seguidamente una de las funciones que proporciona Keras más utilizadas a lo largo del código:

- Conv2D: Keras es responsable automáticamente de generar las salidas que aparecen en la Figura 15 a través de la configuración de los parámetros que se proporcionan a la función (Ver ANEXO N).

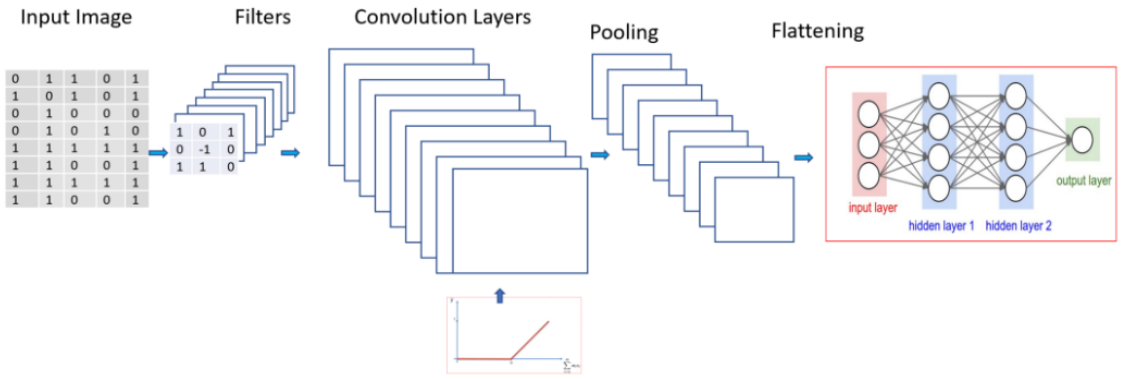

Figura 15 – Conv2D **[58]**.

4. Resultados

4.1 Dataset

La mayoría de los trabajos de investigación realizados hasta la fecha se basan en *datasets* generalizados y públicos, como son COCO o ImageNet, con un gran número de muestras lo que permite a la comunidad comparar directamente el rendimiento de sus modelos con los de otras investigaciones de una forma más general, ya que no se aplica a una problemática específica. Sin embargo, como esta investigación se centra en buscar una solución factible a un problema específico, y basado en los resultados ya obtenidos de otro trabajo realizado por otros investigadores, se decidió desde un principio utilizar el mismo *dataset* compuesto y creado por ellos [40] [44].

El *dataset* utilizado está compuesto por un conjunto de seis clases de objetos las cuales son "*pistola*", "*cuchillo*", "*smartphone*", "*billete*", "*monedero*" y "*tarjeta*" como se puede observar en algunas muestras que ofrecen al lector una visión del contenido con el que se va a entrenar el modelo Figura 16, Figura 17, Figura 18, Figura 19, Figura 20 y Figura 21.

Figura 16 – Muestras de pistolas de Sohas dataset **[40]**.

Figura 17 – Muestras de cuchillos de Sohas dataset **[40]**.

Figura 18 – Muestras de tarjetas de Sohas dataset **[40]**.

Figura 19 – Muestras de monederos de Sohas dataset **[40]**.

Figura 20 – Muestras de billetes de Sohas dataset **[40]**.

Figura 21 – Muestras de smartphones de Sohas dataset **[40].**

La variación de objetos que incluye se debe a que los autores de investigaciones anteriores se han percatado que a veces los modelos tienden a confundir armas con otros objetos que las personas tienen en sus manos. Las estadísticas del *dataset* en los que respecta a la distribución de instancias para el *dataset* de training y test son las que muestran las tablas Tabla 4 y Tabla 5.

Dataset para entrenamiento							
Clase	**Pistola**	**Cuchillo**	**Smartphone**	**Billete**	**Monedero**	**Tarjeta**	**Total**
Nº	1375	1825	551	405	494	216	4848

Tabla 4 – Sohas Training dataset.

Dataset para test							
Clase	**Pistola**	**Cuchillo**	**Smartphone**	**Billete**	**Monedero**	**Tarjeta**	**Total**
Nº	80	452	130	46	65	57	830

Tabla 5 - Sohas Test dataset.

4.2 Conjunto de entrenamiento

Para el entrenamiento de los modelos se ha utilizado la plataforma Colab de Google que permite entrenar modelos de una forma fácil, sin embargo, para entrenar el modelo y obtener realmente el potencial de la tarjeta gráfica durante largo tiempo es necesario hacer una suscripción.

La tarjeta gráfica asignada por Colab para el entrenamiento es Tesla P100-PCIE, el modelo se entrenó por más de 10 horas y 50 épocas (*epochs*):

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   41C    P0    28W / 250W |      0MiB / 16280MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

4.3 Resultados del entrenamiento

Como se puede apreciar en las siguientes figuras, hay un gran número de aciertos, llegando al 80.96% de acierto para el modelo Mask R-CNN, como muestra la Tabla 6. En la Figura 22 y Figura 23, se puede apreciar el porcentaje de acierto por cada clase. En este gráfico se observa como para algunas clases de objetos el acierto es mucho mayor que para otras clases, como por ejemplo "*cuchillo*" y "*billete*" son detectadas a un 95.58% y 93.48% respectivamente. Por el contrario, las clases "*tarjeta*" y "*smartphone*" son detectadas en un 53.06% y 21.05% respectivamente, lo que parece interesante en los resultados es que se podría suponer que el modelo no está compensado con el número de imágenes que se ha entrenado de cada clase, por ejemplo hay 452 imágenes de cuchillo respecto a 46 imágenes de billetes, por lo que se podría inducir que el modelo probablemente confunde un billete con una tarjeta debido a su forma similar y también, debido a que la forma en la que se sostiene el billete y tarjeta son similares.

El número de falsos positivos es 75 y de falsos negativos 83:

Clase	**Pistola**	**Cuchillo**	**Smartphone**	**Billete**	**Monedero**	**Tarjeta**	**Total**
Instancias	80	452	130	46	65	57	830
Aciertos	72	432	69	43	44	12	672
Porcentaje Aciertos	90,00	95,58	53,08	93,48	67,69	21,05	80,96

Tabla 6 - Resultados de prueba.

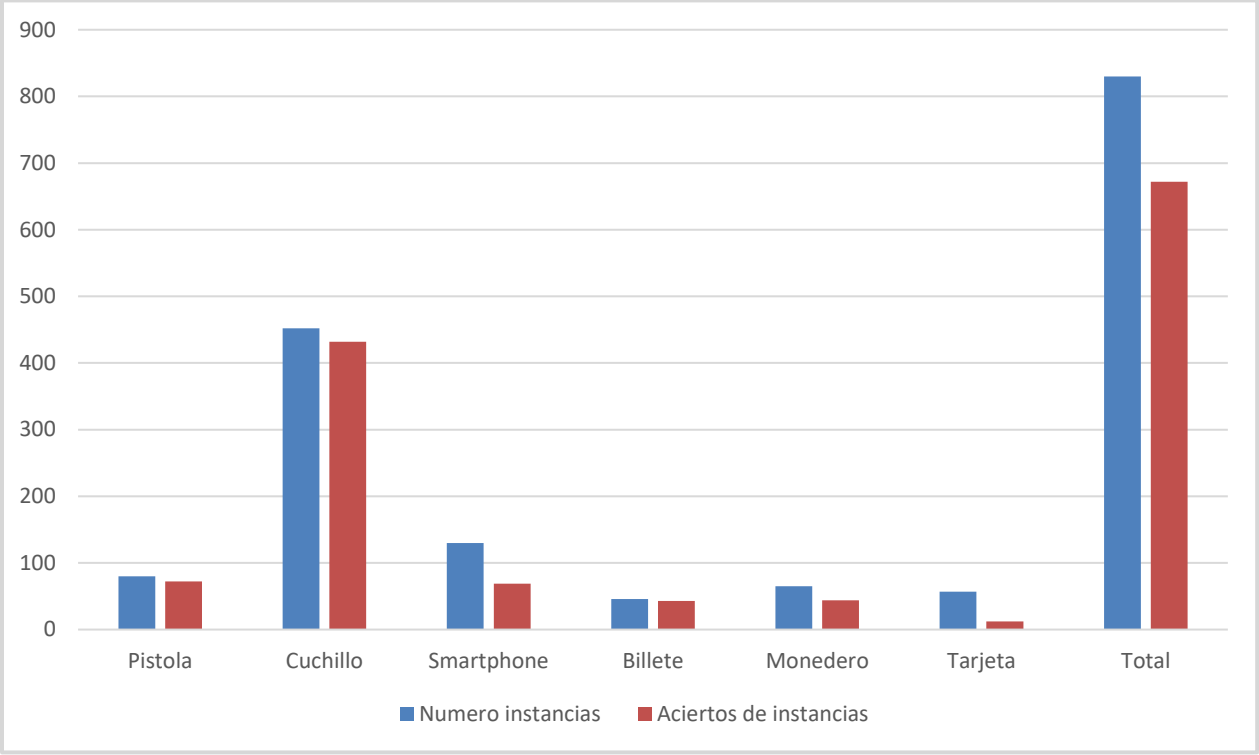

Figura 22 – Resultados Mask R-CNN.

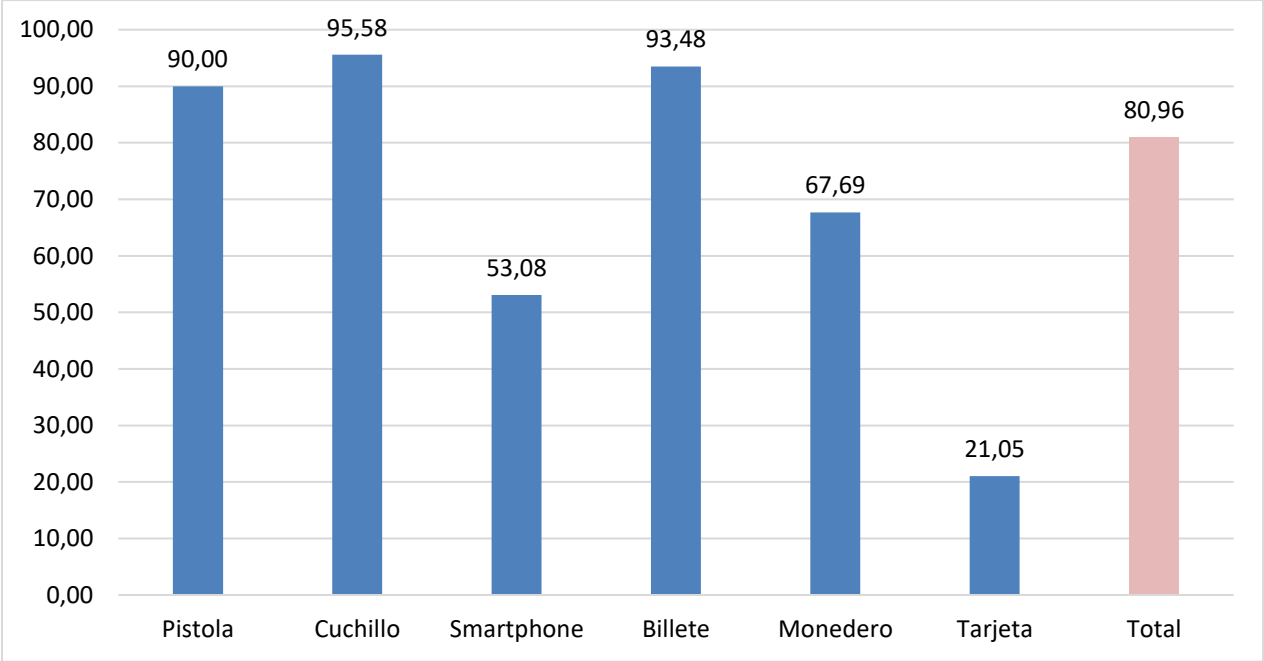

Figura 23 – Resultados Mask R-CNN en porcentaje.

Algunas figuras de los resultados obtenidos:

Figura 24 – Detección cuchillo.

Figura 25 – Detección cuchillo

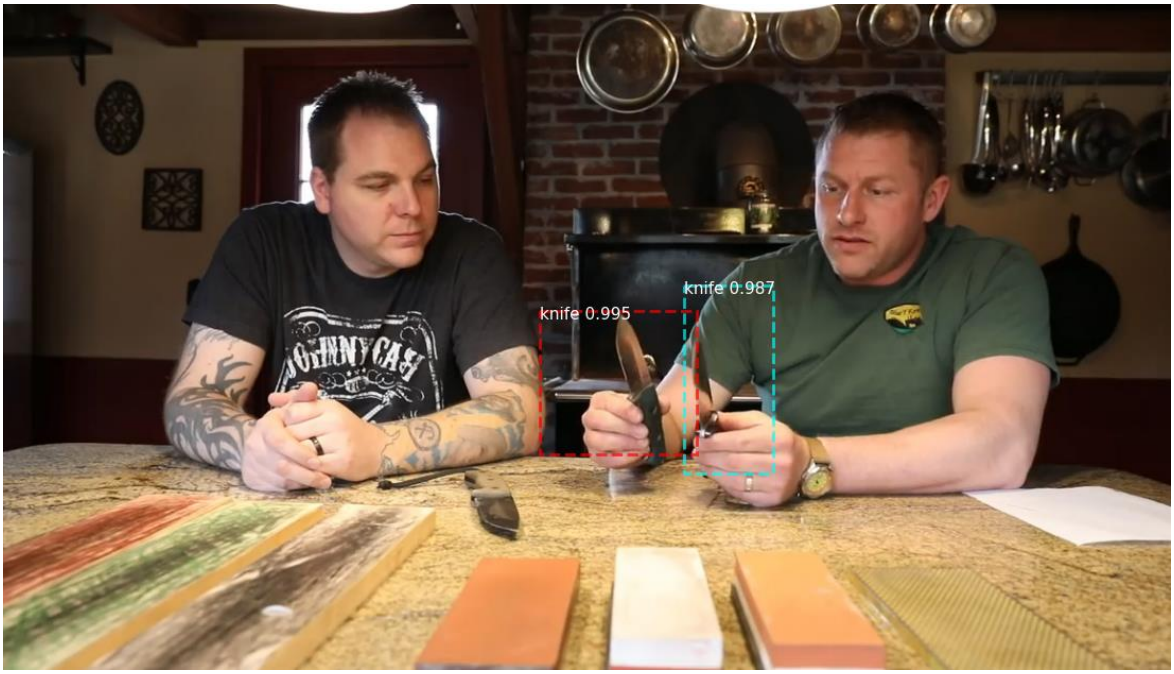

Figura 26 – Detección de dos cuchillos en una imagen.

Figura 27 – Detección de cuchillo en Video Cámara de seguridad.

Figura 28 – Detección de monedero.

Figura 29 – Detección de billete.

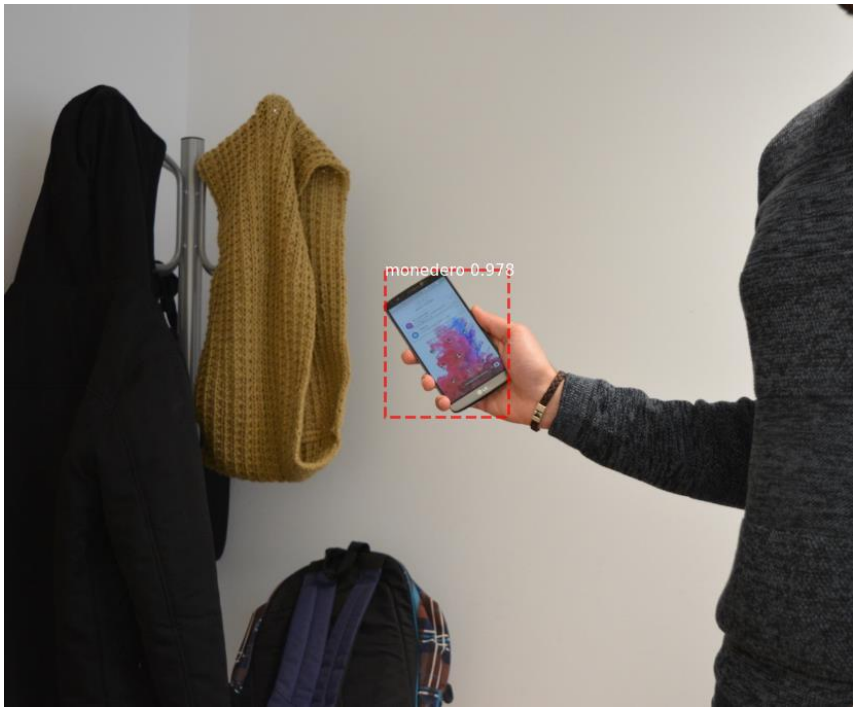

Figura 30 – Detección de falso positivo.

Figura 31 – Detección de false positivo.

Figura 32 – Detección de pistola

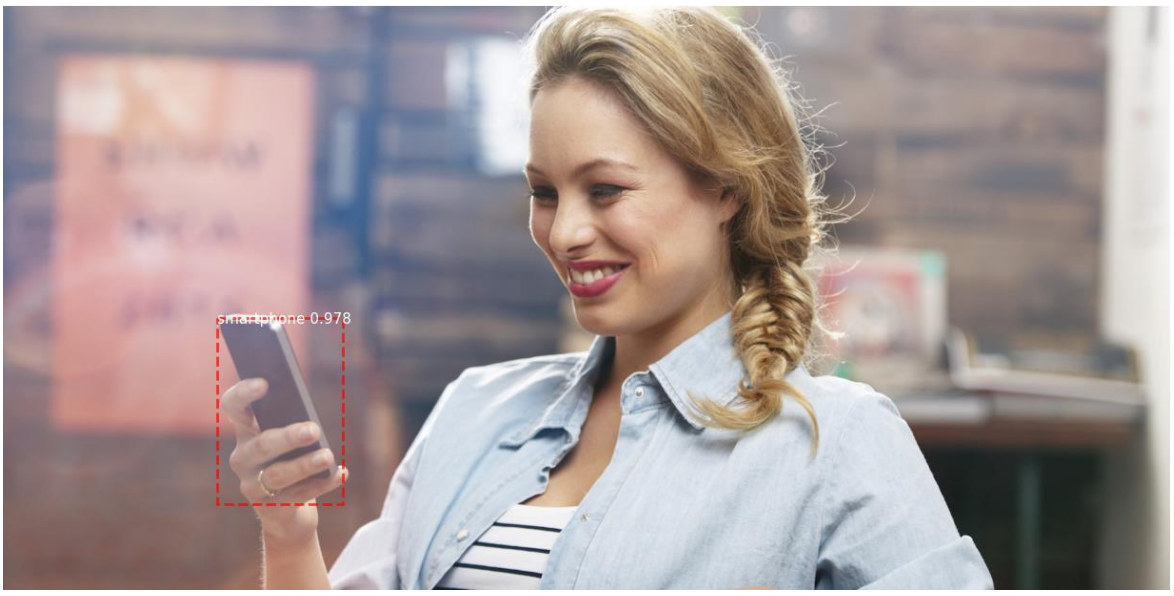

Figura 33 – Detección de smartphone

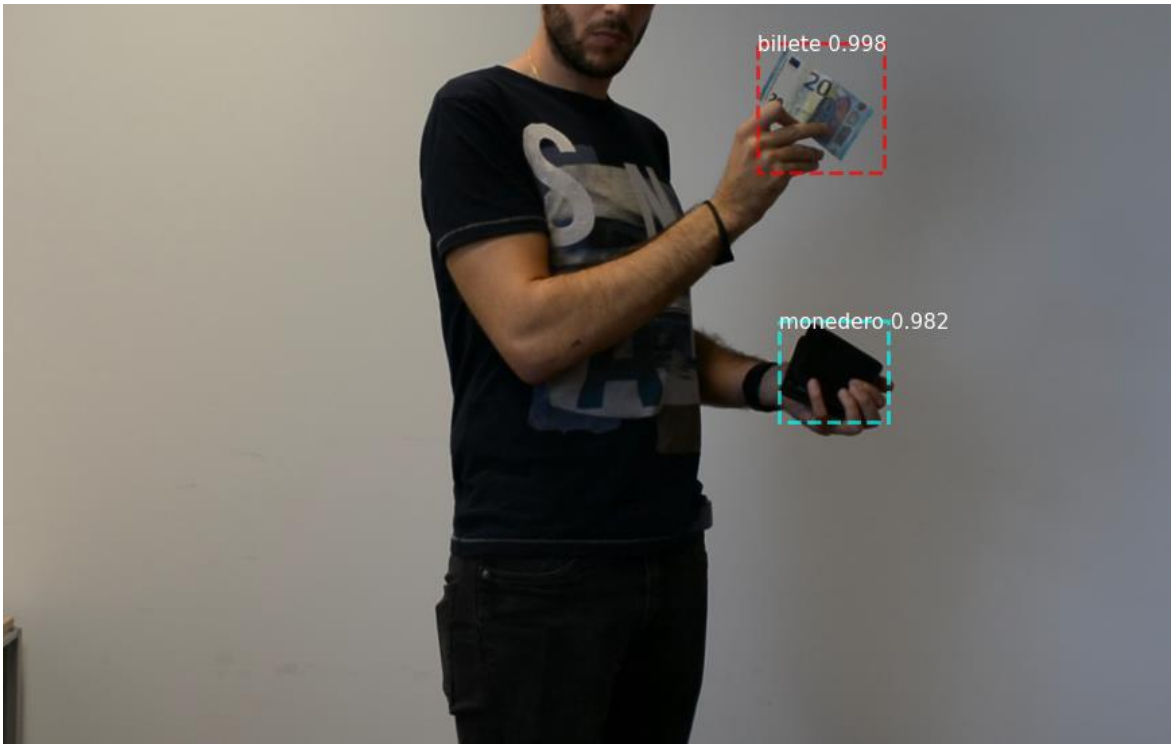

Figura 34 – Detección de diferentes objetos en una imagen.

4.4 Evaluación de resultados

En análisis estadístico, la clasificación binaria "*F-score*" mide la precisión con la que se realiza una prueba. Para ello se realizan tres operaciones, la precisión (*precision*), la sensibilidad (*recall*) y el coeficiente *Dice.* El mayor valor posible para estas operaciones es 1.0 indicando perfección, siendo por otro lado 0 el peor valor que se puede obtener [59].

Precisión: La precisión muestra la exactitud de la segmentación. En concreto, la precisión mide el porcentaje de positivos correctos en la segmentación y se calcula mediante la siguiente fórmula:

$$Precision = \frac{Verdadero\ positivos}{Verdadero\ positivos + Falso\ positivos} \qquad (6)$$

$$Precision = \frac{672}{672 + 75} = 0.899$$

Sensibilidad: La sensibilidad muestra la precisión de la segmentación. Precisamente, mide el porcentaje de positivos correctos detectados entre todas las verdades básicas relevantes y se calcula mediante la siguiente fórmula:

$$Recall = \frac{Verdadero\ positivos}{Verdadero\ positivos + Falso\ negativos} \quad (7)$$

$$Recall = \frac{672}{672 + 83} = 0.890$$

Coeficiente Dice (Dice): muestra la conexión entre la máscara predicha y la realidad. En concreto, los dados se calculan mediante la media armónica de la precisión y la sensibilidad o *Recall*, como se ilustra en la siguiente fórmula:

$$Dice = \frac{2\ x\ Verdadero\ positivos}{2\ x\ Verdadero\ positivos + Falso\ postivos + \ Falso\ negativos} \quad (8)$$

$$Dice = \frac{2\ x\ 672}{2\ x\ 672 + 75 + \ 83} = 0.894$$

Esta investigación se ha realizado con el fin de evaluar la importancia de la inteligencia artificial para la predicción de objetos en aplicaciones en las que los resultados son de alta sensibilidad y la efectividad toma una gran importancia en la solución. El resultado medio de precisión finalmente ha sido del 89% basado en el modelo original, simplemente sin modificaciones tal y como es presentado por sus creadores. Los resultados están detrás de algunas implementaciones [44] como se puede ver en Tabla 7, aunque es importante mencionar que los resultados son relativos a un gran número de parámetros. En esta investigación se alcanzan mejores resultados (84.21%) que en otras investigaciones realizadas con el mismo *dataset* [40] tras un proceso de modificaciones e intentos para obtener mejores resultados, por lo que si se sometiese este modelo al mismo proceso de entrenamiento y modelaje, los resultados podrían llegar a ser mucho mejores que en al resto de investigaciones. Por otro lado, los datos obtenidos anteriormente de Sensibilidad y

Coeficiente Dice son de 0.890 y 0.894, aproximándose casi a 1.0 lo que demuestra y valida de forma estadística la efectividad de este modelo como solución a la problemática.

Investigación	**Enfoque**	**Dataset**	**Media precisión**
[40]	Faster R-CNN con VGG16	Sohas Dataset	84.21%
[44]	Faster R-CNN con ODeBiC	Sohas Dataset	93.87%
Este estudio	**Mask R-CNN con ResNet101 original sin ninguna modificación**	**Sohas Dataset de** [44]	**89%**

Tabla 7 - Media de precisión entre diferentes papers basado en **[45]**.

Una de las posibles razones que pueden ser la razón por la que este modelo funciona tan bien puede ser el uso de ResNet101 las cuales no propagan valores falsos o repetitivos a la largo de la red "*backbone*". Esta red es encargada de crear el mapa de características lo que aporta datos mucho más exactos con los que alimentar la red completa, lo que produce que desde un principio los datos de posibles objetos con los que se trabaja sean más precisos. Por otro lado, otro aspecto importante es el uso de RoIAlign con el que como se comentó anteriormente, ofrece menos perdida de datos usando interpolación lineal.

5. Conclusiones y trabajo futuro

5.1 Conclusiones

En el presente trabajo de fin de máster, se ha llevado a cabo una profunda investigación y trabajo experimental con el fin de intentar mejorar el estado actual para el problema de la detección de armas. Con la implementación del modelo Mask R-CNN y su correspondiente entrenamiento, se han demostrado resultados prometedores después de un extenso entrenamiento del modelo tal y como muestra el apartado anterior de resultados.

Desde un principio, la elección de la problemática resultó un aspecto difícil. La comunidad hasta día de hoy no ha realizado grandes avances en el campo de estudio de la detección de armas, sino que han concentrado sus esfuerzos en aplicaciones más generales que permitan la clasificación y segmentación de objetos en general, siendo lo únicos campos de estudio más relevantes de investigación la visión artificial para automovilismo, conducción inteligente y campos como la medicina. El estudio de la literatura ha sido por lo tanto algo más complejo, y con un cierto grado de incertidumbre. Al principio fue necesario leer y comparar el éxito de modelos usados en otras investigaciones similares, con el fin de extraer una idea más abstracta de cuales podían ser las líneas de investigación a seguir para este trabajo. Por suerte, algunos autores tienen investigaciones sobre la problemática, lo que sirvió de un cierto modo de guía para entender qué modelos se podrían utilizar para avanzar en la línea correcta de investigación.

Tras la revisión de la literatura, un aspecto importante es que muchos autores no arriesgaban a utilizar otros modelos nuevos, sino que intentaban usar los modelos ya desarrollados y probar diferentes estructuras bases, con el fin de intentar mejorar los resultados. Por otro lado, frecuentemente intentan realizar pequeñas modificaciones al final de la red que permitan dar mejores resultados, peno nunca intentar usar un nuevo modelo y ver cuales podían ser los resultados. Por suerte tras el análisis de un modelo que prometía algunas mejoras respecto a otros modelos ya usados, el modelo Mask R-CNN resultó ser un gran acierto.

Los autores originales de Mask R-CNN proporcionan una implementación del modelo, aunque tras años de avances en los *frameworks* de desarrollo usados, es realmente imposible utilizar y entender ese código. Con el fin de entrenar el modelo fue necesario una completa adaptación a los nuevos *frameworks* de Tensorflow y Keras. Más adelante y otro aspecto que ha jugado un factor importante fue la elección del dataset. Muchos autores muestran resultados impresionantes con unos *dataset* pero evitan realizar pruebas con otros datos, por lo que este se trabajó se basó en un *dataset* personalizado creado por otros investigadores para esta problemática. El *dataset* tuvo que ser adaptado totalmente puesto que la implementación no aceptaba la estructura de datos que se proporcionaba.

Por último, uno de los aspectos dentro de los objetivos más tedioso y por así decirlo "*desmotivante*" fue el de entrenar el modelo ya que entrenar uno de estos modelos puede durar más de 20 horas dependiendo del tamaño del *dataset*. Existen miles de posibles parámetros a modificar dentro del entrenamiento de un modelo, muchos de ellos pueden provocar que el modelo inicializado necesite tanta memoria que incluso no pueda ser soportado por el equipo, por lo que a las 10 horas de entrenamiento muestre un error imposible de descifrar. Como desgraciadamente el autor de esta tesis no es un experto en la temática, la única forma posible de entrenar fue la de prueba y error. La capacidad del ordenador personal tras días de pruebas mostró realmente difícil que era llevar a cabo un entrenamiento del modelo. La siguiente estrategia fue buscar un ordenador con más capacidad por lo que se realizó una suscripción gratis de Azure, lo que significó trasladar y adaptar todo el código a otra plataforma siendo necesario adaptar la estructura del proyecto, así como instalar todos los paquetes de Python necesarios en el ordenador remoto. Finalmente, Azure no mostró muchos avances puesto que la cuenta de estudiantes no proporcionar muchas prestaciones. Como última opción se usó una suscripción pro a Colab de Google, lo que permitió entrenar al fin el modelo.

Finalmente se puede decir que los objetivos de esta tesis desde un punto de vista personal han sido alcanzados, no han ofrecido mejores resultados que las últimas investigaciones, pero al menos sí mejores que las implementaciones estándar de otros modelos, por lo que se demuestra la hipótesis planteada en un principio. Por otro lado,

nombrar un dato curioso y es que hasta día de hoy no existe investigación en la problemática estudiada ni en alguna otra relacionada, en la que todo el trabajo realizado sea por un solo autor, normalmente hay varios grupos de investigación puesto que es un trabajo enorme y complejo el de realizar modificaciones y entrenar dichos modelos.

5.2 Trabajo futuro

En el futuro, hay algunas posibles ideas para mejorar los resultados obtenidos realizando como algunas modificaciones en el modelo utilizado, por ejemplo, un primer paso es utilizar otros modelos más avanzados que ResNet-101 para extraer las características utilizadas.

Otra opción es implementar el operador ODeBiC en la capa de RoiAlign con el fin de comprobar si añadiendo el mismo operador binario existe la posibilidad de mejorar la detección entre tarjetas y billetes, lo cual mejoraría enormemente los resultados del modelo si se consigue solucionar esa problemática.

Una nueva hebra de investigación que surge fruto de este trabajo es el estudio de la problemática surgida por la clasificación de billetes y tarjetas. Se presupone que el modelo ha aprendido a diferenciar correctamente lo que es una tarjeta de un billete, aunque sus formas sean similares, pero y si el origen de la confusión en realidad se encuentra en que ambos se sostienen de la misma manera (Ver Figuras 17 y 19), o planteado de una forma diferente, si por ejemplo sostenemos varios objetos con formas diferentes de la misma forma que un cuchillo o una pistola, serán también detectados de forma errónea.

Bibliografía

[1] D. G. Lowe, «Distinctive Image Features from Scale-Invariant Keypoints,» *International Journal of Computer Vision 60,* p. 91–110, November 2004.

[2] B. Triggs y N. Dalal, «Histograms of oriented gradients for human detection,» *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05),* pp. 886-893 vol. 1, 2005.

[3] X. Wu, Z. Zhong-Qiu, P. Zheng y S.-t. Xu, «Object Detection with Deep Learning: A Review,» *IEEE TRANSACTIONS ON NEURAL NETWORKS AND LEARNING SYSTEMS FOR PUBLICATION,* 2019.

[4] M. Warren S. y P. Walter, «A logical calculus of the ideas immanent in nervous activity,» *The bulletin of mathematical biophysics,* p. 115–133, 1943.

[5] F. Rossenblatt, «The perceptron: A probabilistic model for information storage and organization in the brain,» *Psychological Review,* pp. 65(6), 386–408, 1958.

[6] R. Keim, «How to Use a Simple Perceptron Neural Network Example to Classify Data,» *Allaboutcircuits,* 17 Noviembre 2019.

[7] K. Kwon, D. Kim y P. HyunWook, «A parallel MR imaging method using multilayer perceptron,» *The International Journal of Medical Physics Research and Practice,* 2017.

[8] Z. Car, S. Sandi Baressi, N. Andelic, I. Lorencin y V. Mrzljak, «Modeling the Spread of COVID-19 Infection Using a Multilayer Perceptron,» *Computational and Mathematical Methods in Medicine,* 2020.

[9] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel y B. Thirion, «Scikit-learn: Machine Learning in Python,» de *Journal of Machine Learning Research*, 2011, pp. 2825-2830.

[10] K. Fukushima, «Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position,» *Biol. Cybernetics,* p. 193–202, 1980.

[11] K. Fukushima, «Neocognitron for handwritten digit recognition,» *Neurocomputing,* pp. 161-180, 2013.

[12] G. P, S. Baskar, P. Mohamed Shakeel y V. R. Sarma Dhulipala , «Identifying brain abnormalities from electroencephalogram using evolutionary gravitational neocognitron neural network,» *Multimed Tools,* p. 10609–10628, 2020.

[13] D. E. Rumelhart, G. E. Hinton y R. J. Williams, «Learning representations by back-propagating errors,» *Nature,* p. 533–536, 1986.

[14] M. C. Mozer, «A Focused Backpropagation Algorithm for Temporal Pattern Recognition,» *Complex Systems,* pp. 349-381, 1989.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard y L. D. Jackel, «Backpropagation Applied to Handwritten Zip Code Recognition,» *Neural Copmutation,* pp. 541-551, 1989.

[16] Y. LeCun, B. Léon, B. Yoshua y H. Patrick, *IEEE,* pp. 2278 - 2324, 1998.

[17] C. Cortes y V. Vapnik, «Support-vector networks,» *Machine Learning,* p. 273–297, 1995.

[18] W. S. Noble, «Support vector machine applications in computational biology,» de *Kernel methods in computational biology*, 2004, pp. 71 - 87.

[19] L. Yuh-Jye y M. O.L, «SSVM: A Smooth Support Vector Machine for Classification,» *Computational Optiomization and Applications,* pp. 5-22, 2001.

[20] A. Krizhevsky, I. Sutskever y G. E. Hinton, «ImageNet classification with deep convolutional neural networks,» *Proceedings of the 25th International Conference on Neural Information Processing Systems,* pp. 1097-1105, 2012.

[21] P. Sermanet, K. Kavukcuoglu, S. Chintala y Y. LeCun, «Pedestrian Detection with Unsupervised Multi-Stage Feature Learning,» *Computer Vision and Pattern Recognition,* p. 12, 2013.

[22] Y. Freund y R. E. Schapire, «Experiments with a new boosting algorithm,» *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning,* pp. 148 - 156, 1996.

[23] R. B. G. D. M. a. D. R. P. F. Felzenszwalb, «Object Detection with Discriminatively Trained Part-Based Models,» *IEEE Transactions on Pattern Analysis and Machine Intelligence,* pp. 1627 - 1645, 2010.

[24] R. B. Girshick, J. Donahue, T. Darrell y J. Malik, «Rich feature hierarchies for accurate object detection and semantic,» *CoRR,* 2013.

[25] C. Chen, M.-Y. Liu, O. Tuzel y J. Xiao, «R-CNN for Small Object Detection,» de *Asian Conference on Computer Vision*, Springer, 2017, pp. 214-230.

[26] X. Peng y C. Schmid, «Multi-region Two-Stream R-CNN for Action Detection,» de *European Conference on Computer Vision*, Springer, 2016, pp. 744-759.

[27] P. Dong y W. Wang, «Better region proposals for pedestrian detection with R-CNN,» de *Visual Communications and Image Processing*, IEEE, 2016, pp. 27-30.

[28] R. Girshick, «Fast R-CNN,» *Microsoft Research,* 2015.

[29] K. Simonyan y A. Zisserman, «Very Deep Convolutional Networks for Large-Scale Image Recognition,» *ICLR,* 2015.

[30] M. u. Hassan, «neurohive.io,» Noviembre 2018. [En línea]. Available: https://neurohive.io/en/popular-networks/vgg16/.

[31] M. Everingham, «Visual Object Classes Challenge 2012 (VOC2012),» 2012. [En línea]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/.

[32] S. K, «towardsdatascience.com,» Abril 2019. [En línea]. Available: https://towardsdatascience.com/region-of-interest-pooling-f7c637f409af.

[33] Y. D. H. B. Y. Z. K. H. K. Wang, «Use fast R-CNN and cascade structure for face detection,» *Visual Communications and Image Processing,* pp. 1-4, 2016.

[34] M. S. H. Q. L. C. Xiu Li, «Fast accurate fish detection and recognition of underwater images with Fast R-CNN,» *OCEANS 2015 - MTS/IEEE Washington,* pp. 1-5, 2015.

[35] Q. L. Y. Y. F. C. B. Z. R. Qian, «Road surface traffic sign detection with hybrid region proposal and fast R-CNN,» *12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD),* pp. 555-559, 2016.

[36] S. Ren, K. He, R. Girshick y J. Sun, «Faster R-CNN: towards real-time object detection with region proposal networks,» *Proceedings of the 28th International Conference on Neural Information Processing Systems,* pp. 91 - 99, 2015.

[37] Z. a. S. H. a. Z. S. a. Z. J. a. L. L. a. Z. H. Deng, «Multi-scale object detection in remote sensing imagery with convolutional neural networks,» *ISPRS Journal of Photogrammetry and Remote Sensing,* p. 145, 2018.

[38] Q. Fan, L. Brown y J. Smith, «A Closer Look at Faster R-CNN for Vehicle Detection,» *IEEE Intelligent Vehicles Symposium,* pp. 19-22, 2016.

[39] X. Zhao, W. Li, Y. Zhang, T. A. Gulliver, S. Chang y Z. Feng, «A Faster RCNN-based Pedestrian Detection System,» *IEEE 84th Vehicular Technology Conference,* pp. 1-5, 2016.

[40] R. Olmos, S. Tabik y F. Herrera, «Automatic handgun detection alarm in videos using deep learning,» *Neurocomputing,* vol. 275, pp. 66-72, 2018.

[41] K. V. Gyanendra y A. Dhillon, *Proceedings of the 7th International Conference on Computer and Communication Technology,* pp. 84-88, 2017.

[42] M. M. Fernandez-Carrobles, O. Deniz y F. Maroto, «Gun and Knife Detection Based on Faster R-CNN for Video Surveillance,» *Iberian Conference on Pattern Recognition and Image Analysis,* pp. 441-452, 2019.

[43] A. R. Abdul Rafiez, M. Norwati y P. Thinagaran, «Single Classifier, OvO, OvA and RCC Multiclass Classification Method in Handheld Based Smartphone Gait Identification,» *AIP Conference Proceedings,* vol. 1891.

[44] F. Pérez-Hernández, S. Tabik, A. Lamas, R. Olmos, H. Fujita y F. Herrera, «Object Detection Binary Classifiers methodology based on deep learning to identify small objects handled similarly: Application in video surveillance,,» *Knowledge-Based Systems,* p. Volume 194, 2020.

[45] R. M. Alaqil, J. A. Alsuhaibani, B. A. Alhumaidi, R. A. Alnasser, R. D. Alotaibi y H. Benhidour, «Automatic Gun Detection From Images Using Faster,» *First International Conference of Smart Systems and Emerging Technologies,* 2020.

[46] h. Karakaya, I. $afak, O. Gztilrk, M. Bal y Y. E. Esin, «Gun Detection with Faster R-CNN in X-Ray Images,» *IEEE Signal Processing and Communications Applications,* 2020.

[47] S. Nie, Z. Jiang, H. Zhang, B. Cai y Y. Yao, «Inshore Ship Detection Based on Mask R-CNN,» *IEEE International Geoscience and Remote Sensing Symposium,* pp. 693-696, 2018.

[48] J.-Y. Chiao, K.-Y. Chen, K. Y.-K. Liao, P.-H. Hsieh, G. Zhang y T.-C. Huang, «Detection and classification the breast tumors using Mask R-CNN on sonograms,» *Medicine (Baltimore),* 2019.

[49] Z. Yang, R. Dong, H. Xu y J. Gu, «Instance Segmentation Method Based on Improved Mask R-CNN for the Stacked Electronic Components,» *Electronics,* p. 886, 2020.

[50] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» Microsoft Research, 2015.

[51] K. He, G. Gkioxari, P. Dollar y R. Girshick, «Mask R-CNN,» *Facebook AI Research,* 2017.

[52] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan y S. Belongie, «Feature Pyramid Networks for Object Detection,» Facebook AI Research, Cornell University and Cornell Tech, 2017.

[53] J. Hui, «https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c,» Mayo 2018. [En línea]. Available: https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.

[54] P. Monasse, «Extraction of the Level Lines of a Bilinear Image,» *Image Processing On Line,* pp. 205-219, 2019.

[55] W. Zou, Z. Zhang, Y. Peng, C. Xiang, S. Tian y Z. Lu, «A Strong Correlation Learning Framework for Region Proposal,» *Hal open science,* 2021.

[56] Google, «tensorflow,» [En línea]. Available: https://www.tensorflow.org/.

[57] T. Keras, «keras,» [En línea]. Available: https://keras.io/.

[58] R. Khandelwal, «Towardsdatascience,» Mayo 2020. [En línea]. Available: https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c.

[59] K. Zou, S. Warfield, A. Bharatha, C. Tempany, M. Kaus, S. Haker, F. Jolesz y R. Kikinis, «Statistical validation of image segmentation quality based on a spatial overlap index.,» *Acad Radiol,* 2004.

Lista de Símbolos y abreviaturas

Abreviaturas

Abreviatura	Término
SIFT	Scale-Invariant Feature Transform
HOG	Histogram of Oriented Gradiants
PASCAL VOC	Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Classes
CNN	Convolutional Neural Network
DPM	Deformable Part Model
RPN	Region Proposal Network
FPN	Feature Pyramid Network
AP	Average Precision
Resnet	Residual Neural Network

Anexo

```
#Stage 1
x = KL.ZeroPadding2D((3, 3))(input_image)
x = KL.Conv2D(64, (7, 7), strides=(2, 2), name='conv1', use_bias=True)(x)
x = BatchNorm(name='bn_conv1')(x, training=train_bn)
x = KL.Activation('relu')(x)
C1 = x = KL.MaxPooling2D(pool_size=(3, 3), strides=(2, 2), padding='same')
```

ANEXO A – ResNet101, Etapa 1 conv1 **[60]**

```
# Stage 2
x = self.conv_block(x, 3, [64, 64, 256], stage=2, block='a', strides=(1,
1), train_bn=train_bn)
x = self.identity_block(x, 3, [64, 64, 256], stage=2, block='b',
train_bn=train_bn)
C2 = x = self.identity_block(x, 3, [64, 64, 256], stage=2, block='c',
train_bn=train_bn)
```

ANEXO B – ResNet101, Etapa 2 conv2 **[60]**

```
der conv_block(input_tensor, kernel_size, filters, stage, block,
               strides=(2, 2), use_bias=True, train_bn=True):
    nb_filter1, nb_filter2, nb_filter3 = filters
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    x = KL.Conv2D(nb_filter1, (1, 1), strides=strides,
                  name=conv_name_base + '2a', use_bias=use_bias)(input_ten-
sor)
    x = BatchNorm(name=bn_name_base + '2a')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.Conv2D(nb_filter2, (kernel_size, kernel_size), padding='same',
                  name=conv_name_base + '2b', use_bias=use_bias)(x)
    x = BatchNorm(name=bn_name_base + '2b')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.Conv2D(nb_filter3, (1, 1), name=conv_name_base + '2c',
use_bias=use_bias)(x)
    x = BatchNorm(name=bn_name_base + '2c')(x, training=train_bn)

    shortcut = KL.Conv2D(nb_filter3, (1, 1), strides=strides,
                         name=conv_name_base + '1', use_bias=use_bias)(in-
put_tensor)
    shortcut = BatchNorm(name=bn_name_base + '1')(shortcut, trai-
ning=train_bn)

    x = KL.Add()([x, shortcut])
    x = KL.Activation('relu', name='res' + str(stage) + block + '_out')(x)
    return x
```

ANEXO C – ResNet101, Bloque estandar convolucional **[60]**

```
def identity_block(input_tensor, kernel_size, filters, stage, block,
                   use_bias=True, train_bn=True):

    nb_filter1, nb_filter2, nb_filter3 = filters
    conv_name_base = 'res' + str(stage) + block + '_branch'
    bn_name_base = 'bn' + str(stage) + block + '_branch'

    x = KL.Conv2D(nb_filter1, (1, 1), name=conv_name_base + '2a',
                  use_bias=use_bias)(input_tensor)
    x = BatchNorm(name=bn_name_base + '2a')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.Conv2D(nb_filter2, (kernel_size, kernel_size), padding='same',
                  name=conv_name_base + '2b', use_bias=use_bias)(x)
    x = BatchNorm(name=bn_name_base + '2b')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.Conv2D(nb_filter3, (1, 1), name=conv_name_base + '2c',
                  use_bias=use_bias)(x)
    x = BatchNorm(name=bn_name_base + '2c')(x, training=train_bn)

    x = KL.Add()([x, input_tensor])
    x = KL.Activation('relu', name='res' + str(stage) + block + '_out')(x)
    return x
```

ANEXO D – ResNet101, Bloque identidad convolucional **[60]**

```
# Stage 3
x = self.conv_block(x, 3, [128, 128, 512], stage=3, block='a',
train_bn=train_bn)
x = self.identity_block(x, 3, [128, 128, 512], stage=3, block='b',
train_bn=train_bn)
x = self.identity_block(x, 3, [128, 128, 512], stage=3, block='c',
train_bn=train_bn)
C3 = x = self.identity_block(x, 3, [128, 128, 512], stage=3, block='d',
train_bn=train_bn)
```

ANEXO E – ResNet101, Etapa3 conv3 **[60]**

```
# Stage 4
x = conv_block(x, 3, [256, 256, 1024], stage=4, block='a',
train_bn=train_bn)
block_count = 22
for i in range(block_count):
    x = identity_block(x, 3, [256, 256, 1024], stage=4, block=chr(98 + i),
train_bn=train_bn)
C4 = x
```

ANEXO F – ResNet101, Etapa4 conv4 **[60]**

```
# Stage 5
x = conv_block(x, 3, [512, 512, 2048], stage=5, block='a',
train_bn=train_bn)
x = identity_block(x, 3, [512, 512, 2048], stage=5, block='b',
train_bn=train_bn)
C5 = x = identity_block(x, 3, [512, 512, 2048], stage=5, block='c',
train_bn=train_bn)
```

ANEXO G – ResNet101, Etapa5 conv5 **[60]**

```
TOP_DOWN_PYRAMID_SIZE = 256

P5 = KL.Conv2D(TOP_DOWN_PYRAMID_SIZE, (1, 1), name='fpn_c5p5')(C5)
P4 = KL.Add(name="fpn_p4add")([
    KL.UpSampling2D(size=(2, 2), name="fpn_p5upsampled")(P5),
    KL.Conv2D(TOP_DOWN_PYRAMID_SIZE, (1, 1), name='fpn_c4p4')(C4)])
P3 = KL.Add(name="fpn_p3add")([
    KL.UpSampling2D(size=(2, 2), name="fpn_p4upsampled")(P4),
    KL.Conv2D(TOP_DOWN_PYRAMID_SIZE, (1, 1), name='fpn_c3p3')(C3)])
P2 = KL.Add(name="fpn_p2add")([
    KL.UpSampling2D(size=(2, 2), name="fpn_p3upsampled")(P3),
    KL.Conv2D(TOP_DOWN_PYRAMID_SIZE, (1, 1), name='fpn_c2p2')(C2)])
# Attach 3x3 conv to all P layers to get the final feature maps.
P2 = KL.Conv2D(TOP_DOWN_PYRAMID_SIZE, (3, 3), padding="SAME",
name="fpn_p2")(P2)
P3 = KL.Conv2D(TOP_DOWN_PYRAMID_SIZE, (3, 3), padding="SAME",
name="fpn_p3")(P3)
P4 = KL.Conv2D(TOP_DOWN_PYRAMID_SIZE, (3, 3), padding="SAME",
name="fpn_p4")(P4)
P5 = KL.Conv2D(TOP_DOWN_PYRAMID_SIZE, (3, 3), padding="SAME",
name="fpn_p5")(P5)

P6 = KL.MaxPooling2D(pool_size=(1, 1), strides=2, name="fpn_p6")(P5)
```

ANEXO H – Red Piramidal de características hacia abajo **[60]**

```
# Region Proposal Network
# RPN Model
RPN_ANCHOR_STRIDE = 1
k, ANCHORS_PER_LOCATION = 3
TOP_DOWN_PYRAMID_SIZE = 256

input_feature_map = KL.Input(shape=[None, None, TOP_DOWN_PYRAMID_SIZE],
                             name="input_rpn_feature_map")

# Shared convolutional base of the RPN
d_shared = KL.Conv2D(512, (3, 3), padding='same', activation='relu',
                   strides=RPN_ANCHOR_STRIDE,
                   name='rpn_conv_shared')(input_feature_map)

# Anchor Score. [batch, height, width, anchors per location * 2].
d_scores = KL.Conv2D(2 * ANCHORS_PER_LOCATION, (1, 1), padding='valid',
               activation='linear', name='rpn_class_raw')(d_shared)

# Reshape to [batch, anchors, 2]
rpn_class_logits = KL.Lambda(
    lambda t: tf.reshape(t, [tf.shape(input=t)[0], -1, 2]))(d_scores)

# Softmax on last dimension of BG/FG.
rpn_probs = KL.Activation(
    "softmax", name="rpn_class_xxx")(rpn_class_logits)

# Bounding box refinement. [batch, H, W, anchors per location * depth]
# where depth is [x, y, log(w), log(h)]
d_coordinates = KL.Conv2D(ANCHORS_PER_LOCATION * 4, (1, 1),
padding="valid",
               activation='linear', name='rpn_bbox_pred')(d_shared)

# Reshape to [batch, anchors, 4]
rpn_bbox = KL.Lambda(lambda t: tf.reshape(t, [tf.shape(input=t)[0], -1,
4]))(d_coordinates)


return [rpn_class_logits, rpn_probs, rpn_bbox]
```

ANEXO I – *Region Proposal Network* **[60]**

```
Class PyramidROIAlign(self, inputs):
    # Crop boxes [batch, num_boxes, (y1, x1, y2, x2)] in normalized coords
    boxes = inputs[0]

    # Image meta
    # Holds details about the image. See compose_image_meta()
    image_meta = inputs[1]

    # Feature Maps. List of feature maps from different level of the
    # feature pyramid. Each is [batch, height, width, channels]
    feature_maps = inputs[2:]

    # Assign each ROI to a level in the pyramid based on the ROI area.
    y1, x1, y2, x2 = tf.split(boxes, 4, axis=2)
    h = y2 - y1
    w = x2 - x1
    # Use shape of first image. Images in a batch must have the same size.
    image_shape = parse_image_meta_graph(image_meta)['image_shape'][0]
    # Equation 1 in the Feature Pyramid Networks paper. Account for
    # the fact that our coordinates are normalized here.
    # e.g. a 224x224 ROI (in pixels) maps to P4
    image_area = tf.cast(image_shape[0] * image_shape[1], tf.float32)
    roi_level = log2_graph(tf.sqrt(h * w) / (224.0 / tf.sqrt(image_area)))
    roi_level = tf.minimum(5, tf.maximum(
        2, 4 + tf.cast(tf.round(roi_level), tf.int32)))
    roi_level = tf.squeeze(roi_level, 2)

    # Loop through levels and apply ROI pooling to each. P2 to P5.
    pooled = []
    box_to_level = []
    for i, level in enumerate(range(2, 6)):
        ix = tf.compat.v1.where(tf.equal(roi_level, level))
        level_boxes = tf.gather_nd(boxes, ix)

        # Box indices for crop_and_resize.
        box_indices = tf.cast(ix[:, 0], tf.int32)

        # Keep track of which box is mapped to which level
        box_to_level.append(ix)

        # Stop gradient propogation to ROI proposals
        level_boxes = tf.stop_gradient(level_boxes)
        box_indices = tf.stop_gradient(box_indices)

        # Crop and Resize
        # From Mask R-CNN paper: "We sample four regular locations, so
        # that we can evaluate either max or average pooling. In fact,
        # interpolating only a single value at each bin center (without
        # pooling) is nearly as effective."
        #
        # Here we use the simplified approach of a single value per bin,
        # which is how it's done in tf.crop_and_resize()
        # Result: [batch * num_boxes, pool_height, pool_width, channels]
        pooled.append(tf.image.crop_and_resize(
            feature_maps[i], level_boxes, box_indices, self.pool_shape,
            method="bilinear"))

    # Pack pooled features into one tensor
    pooled = tf.concat(pooled, axis=0)
```

ANEXO J – RoIAlign I **[60]**

```
    box_to_level = tf.concat(box_to_level, axis=0)
    box_range = tf.expand_dims(tf.range(tf.shape(input=box_to_level)[0]),
1)
    box_to_level = tf.concat([tf.cast(box_to_level, tf.int32), box_range],
                             axis=1)

    # Rearrange pooled features to match the order of the original boxes
    # Sort box_to_level by batch then box index
    # TF doesn't have a way to sort by two columns, so merge them and sort.
    sorting_tensor = box_to_level[:, 0] * 100000 + box_to_level[:, 1]
    ix = tf.nn.top_k(sorting_tensor, k=tf.shape(
        input=box_to_level)[0]).indices[::-1]
    ix = tf.gather(box_to_level[:, 2], ix)
    pooled = tf.gather(pooled, ix)

    # Re-add the batch dimension
    shape = tf.concat([tf.shape(input=boxes)[:2],
tf.shape(input=pooled)[1:]], axis=0)
    pooled = tf.reshape(pooled, shape)
    return pooled
```

ANEXO K – RoIAlign II **[60]**

```
def fpn_classifier_graph(rois, feature_maps, image_meta,
                         pool_size, num_classes, train_bn=True,
                         fc_layers_size=1024):

    """
    # ROI Pooling
    # Shape: [batch, num_rois, POOL_SIZE, POOL_SIZE, channels]
    x = PyramidROIAlign([pool_size, pool_size],
                        name="roi_align_classifier")([rois, image_meta] +
feature_maps)
    # Two 1024 FC layers (implemented with Conv2D for consistency)
    x = KL.TimeDistributed(KL.Conv2D(fc_layers_size, (pool_size,
pool_size), padding="valid"),
                           name="mrcnn_class_conv1")(x)
    x = KL.TimeDistributed(BatchNorm(), name='mrcnn_class_bn1')(x,
training=train_bn)
    x = KL.Activation('relu')(x)
    x = KL.TimeDistributed(KL.Conv2D(fc_layers_size, (1, 1)),
                           name="mrcnn_class_conv2")(x)
    x = KL.TimeDistributed(BatchNorm(), name='mrcnn_class_bn2')(x,
training=train_bn)
    x = KL.Activation('relu')(x)

    shared = KL.Lambda(lambda x: K.squeeze(K.squeeze(x, 3), 2),
                       name="pool_squeeze")(x)

    # Classifier head
    mrcnn_class_logits = KL.TimeDistributed(KL.Dense(num_classes),

name='mrcnn_class_logits')(shared)
    mrcnn_probs = KL.TimeDistributed(KL.Activation("softmax"),

name="mrcnn_class")(mrcnn_class_logits)

    # BBox head
    # [batch, num_rois, NUM_CLASSES * (dy, dx, log(dh), log(dw))]
    x = KL.TimeDistributed(KL.Dense(num_classes * 4, activation='linear'),
                           name='mrcnn_bbox_fc')(shared)
    # Reshape to [batch, num_rois, NUM_CLASSES, (dy, dx, log(dh), log(dw))]
    s = K.int_shape(x)
    if s[1] is None:
        mrcnn_bbox = KL.Reshape((-1, num_classes, 4), name="mrcnn_bbox")(x)
    else:
        mrcnn_bbox = KL.Reshape((s[1], num_classes, 4),
name="mrcnn_bbox")(x)

    return mrcnn_class_logits, mrcnn_probs, mrcnn_bbox
```

ANEXO L – Capa de *Feature Pyramid Network* **[60]**

```
def build_fpn_mask_graph(rois, feature_maps, image_meta,
                         pool_size, num_classes, train_bn=True):

    x = PyramidROIAlign([pool_size, pool_size],
                        name="roi_align_mask")([rois, image_meta] +
feature_maps)

    # Conv layers
    x = KL.TimeDistributed(KL.Conv2D(256, (3, 3), padding="same"),
                           name="mrcnn_mask_conv1")(x)
    x = KL.TimeDistributed(BatchNorm(),
                           name='mrcnn_mask_bn1')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.TimeDistributed(KL.Conv2D(256, (3, 3), padding="same"),
                           name="mrcnn_mask_conv2")(x)
    x = KL.TimeDistributed(BatchNorm(),
                           name='mrcnn_mask_bn2')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.TimeDistributed(KL.Conv2D(256, (3, 3), padding="same"),
                           name="mrcnn_mask_conv3")(x)
    x = KL.TimeDistributed(BatchNorm(),
                           name='mrcnn_mask_bn3')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.TimeDistributed(KL.Conv2D(256, (3, 3), padding="same"),
                           name="mrcnn_mask_conv4")(x)
    x = KL.TimeDistributed(BatchNorm(),
                           name='mrcnn_mask_bn4')(x, training=train_bn)
    x = KL.Activation('relu')(x)

    x = KL.TimeDistributed(KL.Conv2DTranspose(256, (2, 2), strides=2,
activation="relu"),
                           name="mrcnn_mask_deconv")(x)
    x = KL.TimeDistributed(KL.Conv2D(num_classes, (1, 1), strides=1,
activation="sigmoid"),
                           name="mrcnn_mask")(x)
    return x
```

ANEXO M – Capa de las máscara final de Mask R-CNN **[60]**

```
tf.keras.layers.Conv2D Conv2D(filters,
             kernel_size,
             strides=(1, 1),
             padding='valid',
             data_format='channels_last',
             dilation_rate=(1, 1),
             activation=None,
             use_bias=True,
             kernel_initializer=None,
             bias_initializer=init_ops.zeros_initializer(),
             kernel_regularizer=None,
             bias_regularizer=None,
             activity_regularizer=None,
             kernel_constraint=None,
             bias_constraint=None,
             trainable=True,
             name=None,
             **kwargs):
```

ANEXO N – Parámetros de una red convoluicional 2D en Keras.