UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Aprendizaje profundo para Análisis de Maquetación en documentos manuscritos

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Salvador Lopez, Josep

Tutor/a: Vidal Ruiz, Enrique

Director/a Experimental: PRIETO FONTCUBERTA, JOSE RAMON

CURSO ACADÉMICO: 2021/2022

Agradecimientos

Quisiera, en primer lugar, agradecer a Enrique Vidal por acceder a ser mi tutor durante este trabajo y abrirme las puertas al mundo de la Inteligencia Artificial. Gracias por confiar en todo momento y por la inestimable ayuda y enseñanzas.

Acto seguido, dar las gracias a José Ramón y Lorenzo Quirós, por acogerme desde el primer segundo. Por la paciencia y por la gran cantidad de consejos. Por guiarme en todo momento y la enorme cantidad de horas dedicadas. Me quedo con todo lo aprendido.

Al PRHLT, por darme la oportunidad de utilizar sus máquinas para poder entrenar todos los modelos de este trabajo y realizar todos los experimentos necesarios.

Finalmente, pero no menos importante, agradecer a mi gente más cercana por apoyarme en los momentos más difíciles y por sacarme del caos cuando más lo necesitaba.

Gracias.

Resumen

En la actualidad, contamos con una gran colección de documentos manuscritos a nivel mundial que encierran información valiosa. Gracias a los avances en el campo de la Inteligencia Artificial, disponemos de las herramientas necesarias para poder llevar a cabo la extracción de dicha información. Este proyecto se centra en el estudio sobre las diferentes técnicas que se pueden utilizar, así como en la correcta elección de las arquitecturas empleadas, lo que puede suponer un gran impacto en cuanto a la calidad de la información que se obtiene, pero, sobretodo, en cuanto a eficiencia en relación a recursos y costes.

A lo largo de este trabajo se utilizaron diferentes técnicas de aprendizaje profundo. Por una parte, se realizó un estudio del impacto de los parámetros de las redes neuronales sobre la eficiencia de estas. Seguidamente, con los resultados que se obtuvieron de este estudio, se probaron arquitecturas de redes neuronales convolucionales alternativas a las ya utilizadas, con la finalidad de mejorar el rendimiento de la arquitectura *Mask R-CNN* en las tareas de segmentación y detección de objetos. Para la realización de los experimentos se utilizó un *corpus* de imágenes de documentos manuscritos musicales. Los resultados obtenidos en los experimentos se compararon con los obtenidos por mi compañero Lorenzo Quirós en su Tesis Doctoral.

Palabras clave: aprenentatge automàtic, aprenentatge profund, documents manuscrits, xarxes neuronals convolucionals

Resum

En l'actualitat, comptem amb una gran col·lecció de documents manuscrits a a nivell mundial que emmagatzemen informació valuosa. Gràcies als avanços tecnològics en el camp de la Intel·ligència Artificial, disposem de les ferramentes necessàries per poder portar a cap l'extracció d'aquesta informació. Aquests projecte es centra en l'estudi sobre les diferents tècniques que es poden utilitzar, així com en la correcta elecció de les arquitectures utilitzades, la qual cosa pot suposar una gran impacte pel que fa a la qualitat de la informació que s'obté, però, sobre tot, pel que fa a l'eficiència en relació a recursos i costs.

Al llarg d'aquest treball s'utilitzaren diferents tècniques d'aprenentatge profund. Per una part, es va realitzar un estudi de l'impacte dels paràmetres de les xarxes neuronals sobre l'eficiència d'aquestes. Seguidament, amb els resultats que es van obtindre d'aquest estudi, es van provar arquitectures de xarxes neuronals convolucionals alternatives a les ja utilitzades, amb la finalitat de millorar el rendiment de l'arquitectura *Mask R-CNN* en les tasques de segmentació i detecció d'objectes. Per a la realització dels experiments es va utilitzar un *corpus* d'imatges de documents manuscrits musicals. Els resultats obtinguts en els experiments es van comparar, amb els obtinguts pel meu company Lorenzo Quirós a la seua Tesis Doctoral.

Paraules clau: aprendizaje automático, aprendizaje profundo, documentos manuscritos, redes neuronales convolucionales

Abstract

Nowadays, we have a large collection of handwritten documents worldwide, which contain valuable information. As a result of the techonological advances made in the field of Arfificial Intelligence, we have the crucial tools to carry out the extraction of such information. This project focuses on the study of the different techniques that can be used, as well as on the correct choice of the architectures put into practice, which can have a great impact concerning the quality of the information acquired, but, above all, concerning the efficiency in regard to resources and costs.

Throughout this work, different deep learning techniques were used. On the one hand, it was done an study of the impact of the neural networks parameters on the efficiency of them. Then, with the results obtained from this study, alternative convolutional neural network architectures were tested to those already used with the aim of improving the performance of the Mask R-CNN architecture in segmentation and object detection tasks. In order to perform the experiments, it was used a corpus of images of musical manuscript documents. The results obtained in the experiments were compared with those obtained by my colleague Lorenzo Quirós in his doctoral thesis.

Key words: machine learning, deep learning,handwritten documents, convolutional neural networks

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

Los documentos escritos datan de aproximadamente 5.000 años, por lo que en estos momentos la humanidad tiene recopilados miles de años de información. Gran parte de su contenido sigue siendo un misterio, por lo que todavía existe muchísima información valiosa por descubrir encerrada en dichos documentos. Al digitalizarlos, una parte de esa información está recopilada en forma de miles de millones de píxeles. Estas imágenes digitalizadas no permiten hacer búsquedas sobre el texto y la información que contienen. Asimismo, para transcribir a mano colecciones de cientos de miles imágenes, haría falta cantidades de tiempo absurdas, por lo que realizar este proceso de manera manual sería inviable.

Para la búsqueda de los textos y la información que contienen estos documentos digitalizados se utilizan las técnicas de reconocimiento de texto (*Handwritten Text Recognition*, *HTR* por sus siglas en inglés) [1] y la indexación probabilística (*Probabilistic Indexing*, *PrIx* por sus siglas en Inglés) [2]. Estas técnicas permite obtener la información que comprenden los documentos digitalizados. Sin embargo, estos deben ser procesados en primer lugar para obtener la estructura de los datos en el documento.

Mediante el uso de técnicas como el Análisis de la Estructura de Documentos (*"Document Layout Analysis"* en inglés, *DLA* de aquí en adelante) [3] sobre los documentos manuscritos, podemos obtener la estructura de maquetación de un documento y, de este modo, determinar los componentes que forman dicho documento. Por tanto, el *DLA* nos permite detectar y categorizar las regiones de interés en la imagen de un documento.

El *DLA* está basado en métodos de *Deel Learning*, otro de los campos que ha tenido un avance significativo estos últimos años. Actualmente, el uso de técnicas de Aprendizaje Automático (más conocido por su nombre en inglés *Machine Learning*, *ML*) y Aprendizaje Profundo (*Deep Learning*) posee un gran impacto en el mundo del procesamiento de imágenes de texto manuscrito, así como también en el *DLA*.

1.1 Motivación

Los documentos manuscritos encierran grandes cantidades de información por descubrir. Hoy en día, el *DLA* nos permite obtener la estructura intrínseca de

los documentos, así como extraer gran parte de la información con el uso de técnicas de *HTR* mediante el uso de tecnologías de *Machine Learning* [4] . El balance entre la capacidad de extracción de dicha información y el coste que esta implica supone un gran reto.

Actualmente, disponemos de recursos computacionales limitados, por lo que es de gran interés poder adaptar este conjunto de técnicas a nuestras capacidades y necesidades. Un correcto estudio de cómo las diferentes técnicas de *Machine Learning* y *Deep Learning* pueden afectar al rendimiento (tiempos de ejecución, uso de memoria, etc.), puede conducir a mejorar dicho rendimiento, a allanar el camino a nuevas aplicaciones, así como a reducir los costes y el impacto energético del uso de estas técnicas.

Este proyecto nace debido a que el equipo de trabajo del *Pattern Recognition and Human Language Technology -PRHLT-* trabaja con imágenes de documentos manuscritos para las tareas de *DLA* y el *HTR*. Actualmente, el *PRHLT* se encuentra en búsqueda de mejorar y encontrar nuevas implementaciones en el campo del *DLA*. Con todo esto, una parte de dicho esfuerzo se ha orientado a mejorar el rendimiento de las arquitecturas y a hacer más eficiente en términos de consumo energético y de memoria así como los tiempos de ejecución de las técnicas ya empleadas.

1.2 Objetivos

El *DLA* nos facilita la obtención de la información que comprenden los documentos manuscritos. Una de las tecnologías más utilizadas en la actualidad para realizar dicha tarea son las redes neuronales convolucionales (*Convolutional Neural Networks* o *CNN* por sus siglas en inglés). Las *CNNs* son ampliamente utilizadas para la extracción de características sobre las imágenes de los documentos. No obstante, tienen asociado un alto coste computacional, así como un elevado tiempo de entrenamiento.

Los modelos contaban con millones de parámetros hace unos poco años. Sin embargo, en la actualidad, podemos ver que proyectos como GPT3 [5] cuentan con billones de parámetros. Esta tendencia exponencial supone la creación de modelos cada vez más grandes y complejos. Por tanto, dichos proyectos requieren un gran número de muestras, así como un tiempo de entrenamiento muy elevado. Debido a esto, es de gran interés estudiar el coste computacional y energético para poder adaptar los modelos a las tareas y, de este modo, reducir dichos costes.

El estudio de cómo afectan las diferentes técnicas, algoritmos y parámetros empleados podría reducir los costes computacionales y/o energéticos. Esto, además, permitiría añadir nuevas funcionalidades o mejorar la eficiencia de las ya existentes, así como reducir los costes energéticos asociados a los largos tiempos de entrenamiento de algunos algoritmos, permitiendo, por una parte, ahorrar tiempo y, por otra, minimizar el impacto en el medio ambiente.

En este trabajo estudiamos el impacto de los diferentes parámetros de las redes neuronales convolucionales y el uso de arquitecturas de *CNN* alternativas a

las utilizadas con el fin de entender qué impacto pueden tener, así como seleccionar los mejores resultados y poder llevarlos a la práctica.

1.3 Estructura de la memoria

Esta memoria está formada por cuatro capítulos. En el primer capítulo se ha realizado una introducción, seguida de una motivación al problema y un planteamiento de los objetivos. En el segundo capítulo, se ha llevado a cabo una descripción del problema tratado, así como de los conceptos teóricos y técnicas utilizadas. A continuación, se han definido las métricas que se han utilizado y se ha efectuado un recorrido explicando todos los experimentos que se han llevado a cabo en el tercer capítulo de este trabajo. Finalmente, en el cuarto y último capítulo se han comentado las conclusiones alcanzadas acerca de este proyecto.

1.4 Colaboración

Este proyecto se ha realizado con la colaboración del equipo de la línea de investigación en *HTR* [1] del centro de investigación *Pattern Recognition and Human Language Technology*, el cual utiliza modelos de *Machine Learning*, entre ellos modelos de redes neuronales convolucionales para los problemas relacionados con imágenes de documentos manuscritos.

Esta trabajo se basa en los trabajos previos realizados por Lorenzo Quirós [4] en su tesis doctoral donde hace uso de Redes de propuesta de región (*Region Proposal Networks, RPN*) junto con *CNNs* para el algoritmo *Mask R-CNN* en las tareas de segmentación y detección de objetos en el campo del DLA[2].

[1] Página web del PRHLT en el proyecto de HTR: https://www.prhlt.upv.es/handwritten-text-recognition/

[2] https://github.com/lquirosd/RPN_DLA

CAPÍTULO 2
Fundamentos

2.1 Conceptos Teóricos

A continuación presentamos algunos conceptos y técnicas que van a ser utilizadas para el tratamiento de las imágenes de documentos manuscritos. En esta sección trataremos los conceptos teóricos más relevantes para poder comprender este trabajo.

2.1.1. *Machine Learning* y *Deep Learning*

El aprendizaje automático es una rama de la Inteligencia Artificial que se basa en el estudio de algoritmos que permiten a los computadores aprender de forma automática mediante la experiencia. Según el tipo de algoritmo utilizado, podemos dividir el aprendizaje automático en tres grupos principales [6]: el aprendizaje supervisado; el no supervisado y el aprendizaje por refuerzo. En este trabajo haremos uso de técnicas de aprendizaje supervisado, por lo que es de gran interés explicar este concepto.

Aprendizaje supervisado Este tipo de algoritmos requieren datos etiquetados. Los datos etiquetados están formados por el conjunto de datos de entrada (*input*) junto a su etiqueta correspondiente, que proporciona la información necesaria para describir el mismo. Estos aprenden la salida de nuevas muestras gracias a la optimización de una función, que asocia los valores reales de entrenamiento con su predicción realizada por el modelo, permitiendo generalizar nuevos valores. Las redes neuronales son un tipo de algoritmo que puede entrenarse bajo los tres tipos de técnicas del aprendizaje automático. En esta memoria utilizaremos redes neuronales entrenadas mediante la técnica de aprendizaje supervisado.

Las aplicaciones de este tipo de tecnologías son numerosas y abarcan cualquier área que pueda obtener series de datos válidas, ya sea para análisis o para clasificación. Desde el diagnóstico médico hasta las finanzas. Junto a el Aprendizaje supervisado se encuentra el Aprendizaje profundo o *Deep Learning*.

El *Deep Learning* [6] es un subcampo del *Machine Learning* que estudia cómo algoritmos automáticos emulan el aprendizaje humano, permitiendo a modelos compuestos por múltiples capas aprender representaciones de datos con múl-

tiples niveles de abstracción. En este trabajo vamos a utilizar modelos de *Deep Learning*, concretamente modelos basados en aprendizaje supervisado, con la intención de estudiar y mejorar el desempeño de la arquitectura *Mask R-CNN* [7], de la cual hablaremos más adelante con profundidad.

2.1.2. *Artificial Neural Networks*

Las redes neuronales artificiales son modelos computacionales de aprendizaje supervisado que están inspiradas en el comportamiento de las neuronas biológicas.

La primera aparición de lo que se puede considerar como la unidad básica de una red neuronal data de 1958, donde Frank Rosenblatt desarrolla el algoritmo *Perceptron*[8], un clasificador lineal binario. Más adelante, en 1985, nace el concepto de *BackPropagation* [9], que permitió entrenar redes multicapa.

Las redes neuronales artificiales buscan simular las tres características básicas de una red neuronal biológica: procesamiento en paralelo; memoria distribuida y adaptabilidad. Esto les permite aprender y generalizar salidas (*output*) a partir de un conjunto de datos de relación matemática desconocida.

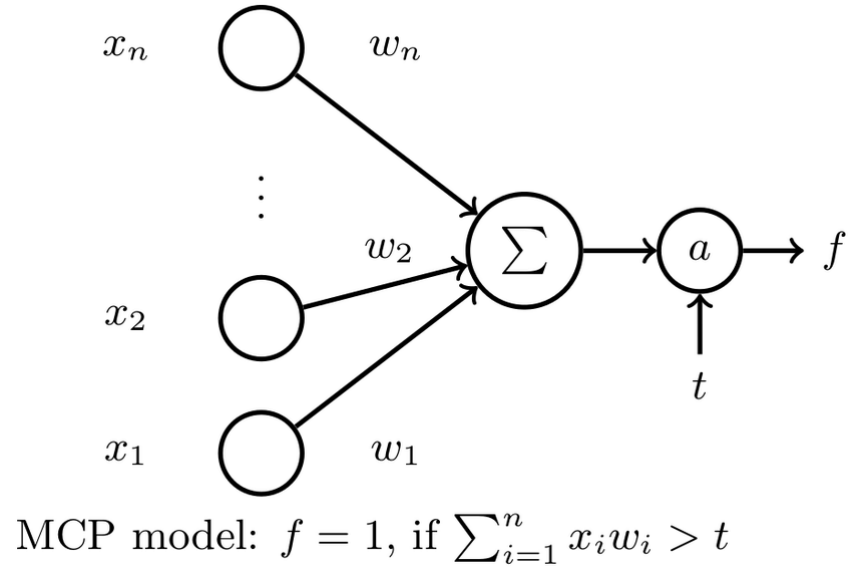

Figura 2.1: Representación de la primera neurona artificial propuesta por Warren McCulloch y Walter Pitts en 1934. Imagen sacada de [10].

Las redes neuronales más básicas están formadas por una capa de entrada; una o varias capas ocultas y una de salida. Cada capa se compone de neuronas que están totalmente conectadas con la siguiente capa, como podemos ver en la figura 2.2 . Este tipo de modelos se llama *full connected*.

Cada capa cuenta con un conjunto de pesos W y valores de sesgo o bias b que se inicializan con valores aleatorios al crear la red. Además, las redes neuronales cuentan con funciones de activación para cada neurona que normalizan su salida.

El proceso de entrenamiento se compone de dos fases. La primera, denominada propagación hacia delante o *forward propagation*, aplica los datos de entrada a la función de la primera capa junto con la función de activación, que alimenta a

la siguiente capa y así hasta llegar a la capa de salida. Aquí se obtiene una predicción de la red para cada muestra. En este paso se calcula la función de pérdida que la red debe minimizar entre el valor de salida de la red y el valor verdadero.

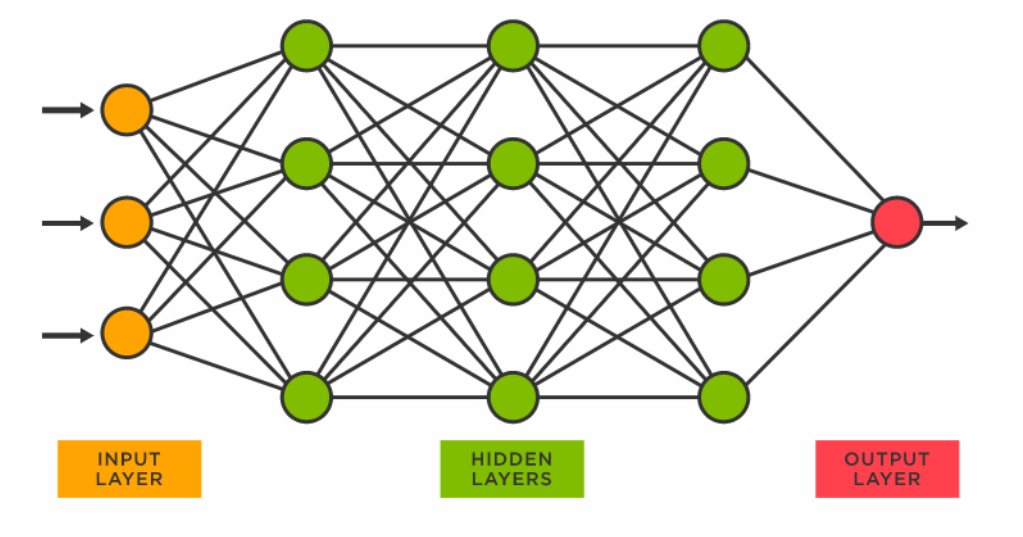

Figura 2.2: Representación de una Red Neuronal. Imagen sacada de [11].

La segunda fase es la de propagación hacia atrás o *backpropagation*. Aquí se calcula el gradiente de la función de coste respecto de los pesos que componen la red, desde la capa de salida hasta la capa de entrada. Seguidamente, estos pesos se actualizan con los gradientes calculados de la siguiente manera:

$$w = w - \eta * (y - \hat{y}) * x \tag{2.1}$$

w es el conjunto de pesos del modelo, $(y - \hat{y})$ es el error de predicción para el modelo en los datos de entrenamiento atribuidos al peso, y x es el valor de entrada. El producto entre el error de predicción y el valor de entrada se multiplica por el factor de aprendizaje o *learning rate* η.

Este proceso de entrenamiento se aplica en un método iterativo denominado descenso por gradiente estocástico o *stochastic gradient descent*[12], el cual realiza el proceso descrito en mini-lotes o *mini-batches* de muestras aleatorias. El número de muestras que se pasa en cada *mini-batch* se llama *Batch Size*. Un *mini-batch* es un subconjunto de imágenes del dataset completo. La red obtiene una predicción y calcula el error que ha tenido en el proceso cada vez que se pasa un *mini-batch*. Al número de iteraciones necesario para ver el conjunto de datos entero se le llama *Epoch*.

Uno de los problemas de las redes neuronales con respecto al número de *epochs* es el sobre entrenamiento o sobreajuste (*overfiting*). Cuando una red se entrena durante demasiado tiempo, el modelo puede llegar a recordaruna gran cantidad de ejemplos en lugar de aprender nuevas características. Esto puede llevar a que, durante el entrenamiento, el modelo mejore notablemente su precisión con el conjunto de *train*. Sin embargo, una vez el modelo ya está entrenado y se prueba con el conjunto de *test*, esta precisión difiere de manera negativa con respecto a la alcanzada con el conjunto de entrenamiento.

Modificar los diferentes parámetros de las redes sobre la misma arquitectura puede suponer resultados diferentes en cuanto a precisión, tiempo de ejecución. etc. Por esta razón, un correcto estudio de los parámetros utilizados puede suponer mejoras con respecto a la precisión, reducir tiempos de entrenamiento y/o disminuir la cantidad de memoria necesaria empleada para entrenar la red, por lo que ajustar estos parámetros será uno de los estudios que realizaremos más adelante en esta memoria.

2.1.3. *Convolutional Neural Networks*

Las redes neuronales convolucionales o *CNN* surgen en 1980 por Kunihiko Fukusima[13], científico informático que estudió acerca de cómo los seres vivos procesan la visión. Con el paso del tiempo, en 1998, estas redes fueron mejoradas gracias al uso del *backpropagation* en el entrenamiento. Más adelante, en el año 2011, se empezaron a implementar este tipo de redes utilizando unidades de procesamiento gráfico (*graphics processing unit* o GPUs por sus siglas en inglés) [14] , lo que permitió unos tiempos de computo de los modelos más rápido.

Las *CNN* son un tipo de red neuronal artificial utilizada para la clasificación de imágenes, aunque pueden utilizarse en otros campos. La principal característica de las *CNN* es su capacidad de detectar dependencias espaciales y temporales de los datos de entrada. En el caso de imágenes, es posible extraer características y relacionarlas entre ellas. Una red neuronal convolucional típica consta de dos tipos de capas: capas convolucionales y de *pooling*.

Las capas de convolución son las encargadas de extraer la información de los datos de entrada mediante la aplicación de las operaciones de convolución. La convolución es la operación matemática que combina dos funciones para producir una tercera.

En el caso de las *CNN*, la convolución se realiza cogiendo la imagen como una matriz de dos dimensiones y un filtro o kernel, que es otra matriz de menor tamaño a la imagen. Se superpone el filtro sobre la matriz que representa la imagen y se realiza la convolución, la cual multiplica los valores de los píxeles de la imagen con los valores del filtro, obteniéndose así una nueva matriz del tamaño del filtro. Los valores de esta submatriz se suman para añadirlos finalmente a la matriz que devuelve la operación de convolución. Podemos ver una representación de la operación de convolución en 2.3. Este proceso se realiza tantas veces como posibles posiciones tiene el filtro sobre la imagen de entrada. El filtro se desliza por la imagen utilizando un paso o *stride*, que indica la cantidad de píxeles a desplazar para realizar la siguiente convolución.

La convolución sigue un orden, de izquierda a derecha y de arriba abajo. La matriz resultante de una capa de convolución se llama mapa de características o *feature map*, el cual representa las características aprendidas de los datos de entrada. Este mapa de características puede tener unas dimensiones distintas a la entrada, ya que esta se ve afectada por el tamaño del filtro y el valor del *stride*. Por tanto, el mapa de características tendrá una altura igual a las posibles posiciones del filtro en el eje vertical y un ancho igual a las posibles posiciones del filtro sobre el eje horizontal. La profundidad será igual al número de filtros aplicados en la capa de convolución.

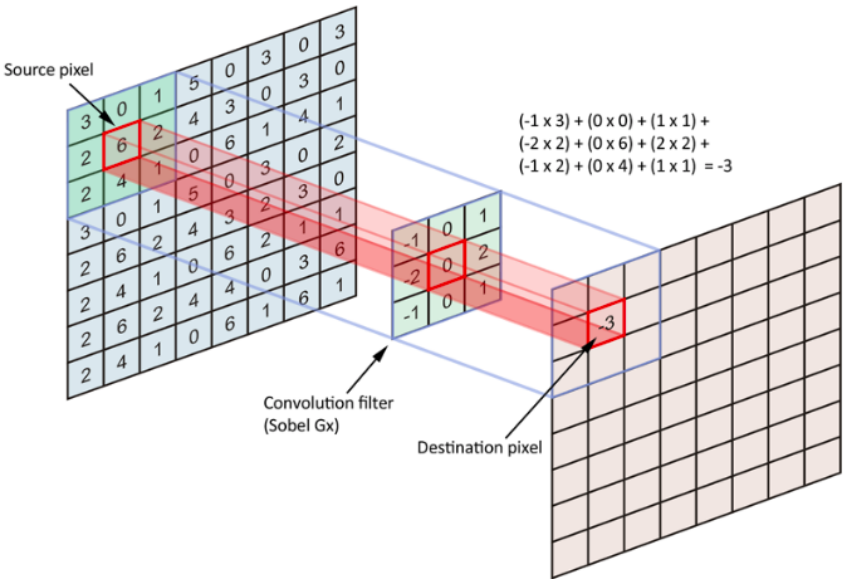

Figura 2.3: Funcionamiento de una capa de convolución. El *kernel* se coloca encima de la imagen de entrada y se multiplican los valores de ambos, se suman estos valores y se guardan en la posición central de la matriz resultante. Imagen sacada de [15].

Para mantener el tamaño de la imagen de entrada aparece el concepto de *zero-padding*, que consiste en agregar una o varias capas de píxeles de valor cero que rodean la entrada, de modo que el mapa de características no reduce la dimensionalidad espacial.

Después de la convolución es posible aplicar la capa de *pooling*. Esta capa recibe como entrada los mapas de características creados por la capa de convolución. Su función es reducir la dimensionalidad para minimizar el número de parámetros que debe procesar la red. En estas capas se utiliza una ventana sobre el mapa de características que aplica una operación de *AveragePooling* o *MaxPooling*, como podemos ver en 2.4. Dependiendo de la operación realizada, calcula la media o el valor máximo. Esta ventana tiene un desplazamiento igual al de los filtros en las capas de convolución, lo que permite reducir el tamaño del mapa de características sin perder información relevante.

Por tanto, la estructura de una *CNN* que podemos ver en la 2.5 toma una imagen como entrada y la pasa a través de capas de convolución y de *pooling*, las cuales extraen las características. Una vez terminado el proceso de extracción de características, se conecta la salida con una red neuronal artificial llamada *full connected* que permite identificar la clase a la que pertenece la entrada.

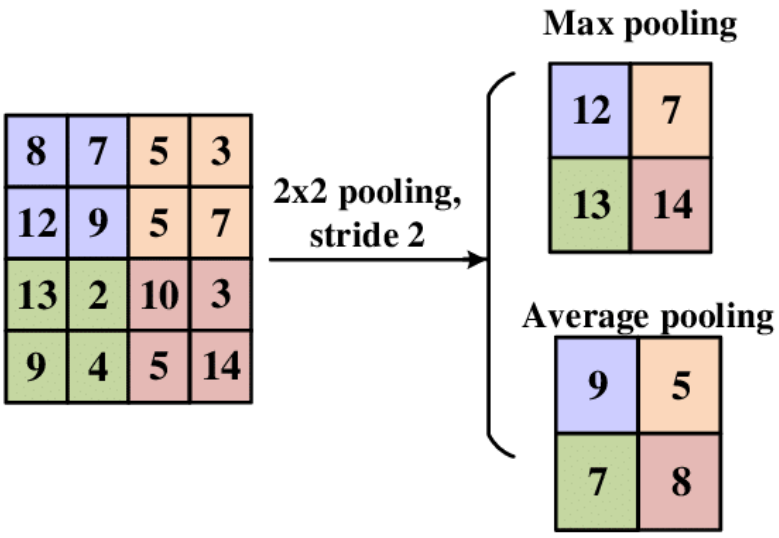

Figura 2.4: Operación de *Pooling* de una red convolucional. En la parte superior podemos ver una operación de *MaxPooling* y abajo la operación de *AveragePooling*. Imagen obtenida de [16].

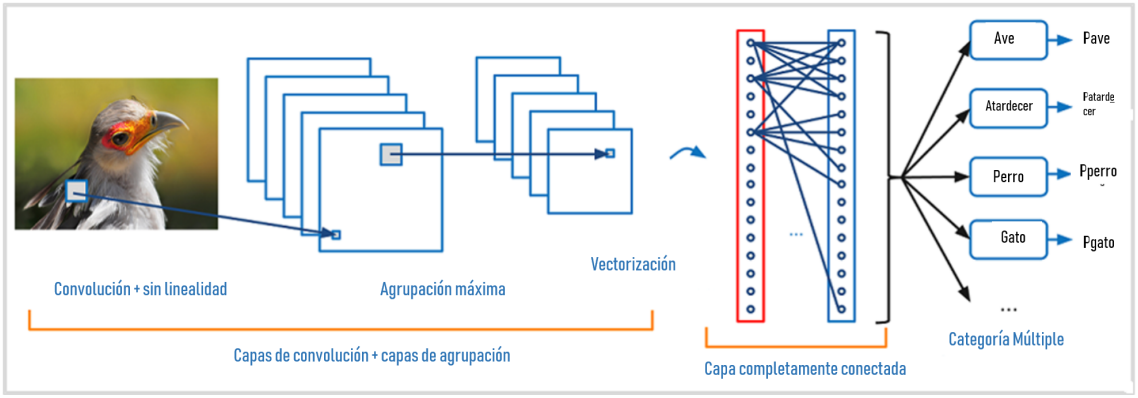

Figura 2.5: Funcionamiento de una Red Neuronal Artificial. En la parte izquierda podemos ver la primera parte formada por capas de convolución y de *pooling* para extraer características. A la derecha estas características se procesan por una red *full connected*. Imagen sacada de [17].

2.2 Arquitecturas Empleadas

Para este trabajo, hemos utilizado algunas de las arquitecturas ya existentes debido a su gran potencia de reconocimiento y eficiencia para resolver los complejos problemas de segmentación y reconocimiento. Es interesante profundizar en el concepto de las siguientes arquitecturas para entender por qué tienen un buen desempeño en este tipo de problemas.

Estos últimos años, el crecimiento de las *CNN* en el ámbito del campo de visión por computador ha sido significativo y ha comportado numerosos avances. Junto con el uso de GPUs para el entrenamiento de las redes neuronales, cabe destacar el nacimiento en 2010 de la competición de *ImageNet*[1]. El proyecto *Ima-*

[1]Página web de *ImageNet*: https://image-net.org/

geNet [18] consiste en una gran base de datos, que actualmente cuenta con más de 14 millones de imágenes etiquetadas a mano con más de 20.000 categorías. La celebración de este tipo de competiciones junto con el crecimiento de la tecnología condujo a un avance constante de las redes neuronales convolucionales, creando de este modo arquitecturas cada vez más complejas y profundas con una mayor capacidad de reconocimiento.

El crecimiento en la profundidad de las redes trajo consigo uno de los problemas que encontramos en la actualidad a la hora de crear redes neuronales convolucionales más complejas: el desvanecimiento del gradiente. El desvanecimiento del gradiente o *vanishing gradient* [19] se produce cuando las redes aumentan su profundidad. Como consecuencia, al llevar a cabo la retropropagación del gradiente, este llega a valores muy pequeños y provoca que no se consigan optimizar eficientemente los pesos de las primeras capas. Esto supone que la red reduzca la calidad de sus resultados a medida que aumenta la profundidad. Para resolver este problema, se introdujo el concepto que veremos a continuación en la siguiente arquitectura.

2.2.1. *Resnet*

Una *Resnet* o *Residual Neural Network* es, en esencia, un tipo de arquitectura de red convolucional basada en bloques de tipo residual. Aunque el concepto de conexión residual es más antiguo [20], este tipo de arquitecturas se presentó en 2015 en [21] y supuso una gran revolución en el mundo de la visión por computador. Se proclamó ganadora del concurso *ImageNet*, reduciendo la tasa de error de forma significativa respecto a sus rivales y consiguiendo crear arquitecturas mucho más profundas, mitigando el problema del desvanecimiento del gradiente.

El concepto que introduce *Resnet* trata de incorporar saltos o «atajos» en las conexiones entre capas, lo que proporciona una conexión directa entre la entrada de una capa y la salida de otra. Estos atajos permiten que el gradiente de la red se propague hasta las primeras capas, pudiendo así crear arquitecturas más profundas y, por ende, alcanzar una mayor precisión al poder extraer mejores características.

Como podemos apreciar en la figura 2.6, se suma la entrada de una capa con la salida de subsecuente. Esta suma permite pasar el gradiente tanto por las capas de la red como por las conexiones residuales o *shortcut connections*. El gradiente es 1 con respecto a cada una de las entradas a la suma, por lo que este no se multiplica por ningún factor que pueda provocar que disminuya, sino que fluye de manera natural a lo largo de todas las capas, mitigando o disminuyendo de manera significativa el problema de desvanecimiento del gradiente.

2.2.2. *Vovnet*

Cuando se trata de crear redes más eficientes se suelen tener en cuenta los *Floating Point Operation per Second* (*FLOPS*) y los parámetros de la red. Sin embargo, los resultados de estas métricas son valores indirectos, por lo que para medir la complejidad de cálculo se deben utilizar métricas directas como la velocidad o la latencia.

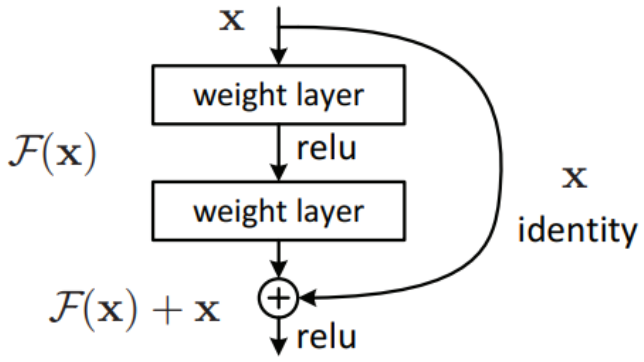

Figura 2.6: Concepto de conexión residual. Sacado de [21].

A la hora de hablar de redes neuronales convolucionales, uno de los factores con mayor coste es el *Memory Access Cost* (*MAC*), principal fuente de consumo de energía y tiempo. Concretamente, el *Dynamic Random Access Memory* (*DRAM*), es el acceso con el coste más elevado.

Las tareas del mundo real exigen alcanzar la mejor precisión posible dentro de un entorno de recursos limitado. Como se propone en [22], hay que tener en cuenta métricas directas para la evaluación de los modelos. Es por esto que se crea el tipo de arquitectura *Vovnet*, que consigue disminuir el *MAC* y por ende la complejidad y los tiempos de entrenamiento de los modelos.

La ventaja del uso de GPUs es la computación paralela. Cuando recibe un tensor grande, la GPU se puede utilizar por completo, pero el problema llega cuando, en redes como *Resnet* o *DenseNet* [23], una capa convolucional se divide en varias más pequeñas. Aunque a efectos prácticos es el mismo, la computación de la GPU es ineficiente. Esto se ve en el uso de convoluciones con un kernel 1×1, que, aunque reduce el número de *FLOPS* no favorece a la eficiencia. En *Resnet* se utilizan convoluciones con kernel 1×1 después de una convolución con kernel 3×3, reduciendo así la cantidad de *FLOPS* y el número de parámetros. No obstante, esto provoca que la capa con kernel 3x3 se divida en dos capas más pequeñas, llevando a tener más cálculos secuenciales y reduciendo la velocidad de inferencia. Esto se conoce como cuello de botella.

Otro punto a destacar en estas redes es el aumento lineal del tamaño del canal de entrada (filtros) con respecto a la profundidad. Esto supone un problema, puesto que hace que el cálculo general pueda crecer cuadráticamente en cuanto a la profundidad.

La arquitectura de *Vovnet* presenta el concepto de módulo *One-Shot Aggregation* (*OSA*). Este tipo de bloque o módulo se basa en la potencia de representación de características que tienen las redes *DenseNet*. Sin embargo, dicho módulo solo realiza una única agregación a la vez, como podemos ver en 2.7. Al concatenar las características en la última capa a la vez, se reduce significativamente el *MAC* y se mejora la eficiencia de cálculo de la GPU. *Vovnet* también aborda el problema de aumentar linealmente el canal de entrada, ya que mantiene constante el número de canales con respecto a la profundidad de la red. Gracias a esto, se reduce el coste de acceso a la memoria disminuyendo las operaciones por elemento.

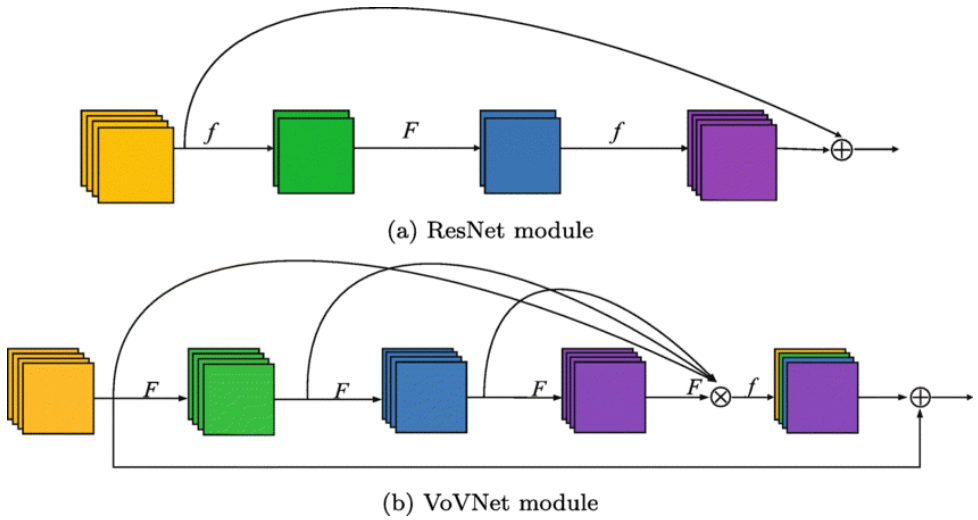

Figura 2.7: Comparación módulo *resnet* (a) con el módulo *OSA* (b), donde podemos apreciar la agregación única al final del bloque. Imagen sacada de [22].

2.2.3. *Mask R-CNN*

En marzo de 2017, el equipo de *Facebook AI Research* (*FAIR*) publicó el *paper* [7], donde se propuso la arquitectura *Mask R-CNN*. Esta se basa en *Faster R-CNN* [24], aunque mejora la precisión en la detección de instancias. Además, *Mask R-CNN* añade la tarea de segmentación de instancias.

Para entender mejor el funcionamiento de esta arquitectura es necesario comprender una serie de conceptos:

Object detection: Técnica que permite que, dada una imagen de entrada, se conozca la clase a la que pertenece cada objeto que aparece dentro de esta imagen, así como las coordenadas de una caja delimitadora o *bounding box* que contiene dentro cada objeto.

Image Segmentation: Consiste en generar una máscara a nivel de píxel que contiene las formas exactas de cada objeto que aparece en la imagen, lo que implica conocer su *bounding box*. Aquí diferenciamos dos tipos de segmentación:

- **Semantic Segmentation**: La segmentación semántica clasifica cada píxel en un conjunto fijo de categorías sin diferenciar instancias de objetos. Se ocupa de la identificación/clasificación de objetos similares como una sola clase desde el nivel de píxel.
- **Instance Segmentation**: La segmentación de instancias se ocupa de la detección correcta de todos los objetos de una imagen, independientemente de si hay dos o más objetos de una misma clase. Combina la detección, la localización y la clasificación de objetos.

El software *Detectron2*, utilizado en este trabajo, se ha desarrollado para la detección y segmentación de objetos. Este usa la arquitectura *Mask R-CNN*, empleada en [4] y en la cual se basa esta memoria. *Mask R-CNN* permite detectar los

(a) Object Detection (b) Semantic Segmentation (c) Instance Segmentation

Figura 2.8: Comparación de *Object Detection*, *Semantic Segmention* y *Insatance Segmentation*. Imagen sacada de [25].

diferentes objetos de una imagen de entrada, donde cada objeto tiene asociado una clase, un *bounding box* y una máscara.

La arquitectura de *Mask R-CNN* como podemos apreciar en 2.11 recibe como entrada una imagen, la cual se procesa mediante un *backbone*, objeto de estudio clave de este trabajo, formado por una *CNN* en conjunto con un *Feature Pyramid Network* (*FPN*). La *CNN*, la cual procesa las imágenes, está dividida en diferentes *stages* con respecto a la profundidad de la red. Este proceso se conoce como *bottom-up*, que reduce la dimensionalidad de las imágenes en cuanto a tamaño y la aumenta respecto a la cantidad de canales.

En cada *stage* se pasan los mapas de características aprendidos hacia la *FPN* mediante conexiones laterales paralelas. El proceso que realiza la *FPN* se conoce como *top-down*, el cual vuelve a aumentar el tamaño de los mapas de características recibidos por la *CNN* en cada una de los *stages*. Podemos ver el proceso que realiza el *backbone* en la figura 2.9. Después de procesar la imagen con el *backbone* y extraer sus características, se pasan los resultados por una *Region Proposal Network* (*RPN*), la cual obtiene mapas de características con muchas propuestas de región. Podemos ver cómo funciona una *RPN* en la figura 2.10.

Utilizando estas propuestas de región sobre el mapa de características se obtienen regiones del mapa de características que pasan por una capa *RoIAlign* que, al igual que en *Faster R-CNN* con la capa *RoI Pooling*, transforma los *RoIs* de distinto tamaño en un tamaño fijo. Para generar la segmentación de instancias, las regiones del mapa de características pasan por una red llamada *Fully Convolutional Network* (*FCN*), formada por dos capas de convolución que permiten a *Mask R-CNN* obtener una máscara binaria que clasifica cada píxel de la imagen cubierta por la región propuesta. Esta máscara se generará por cada región de interés y por cada clase de objeto.

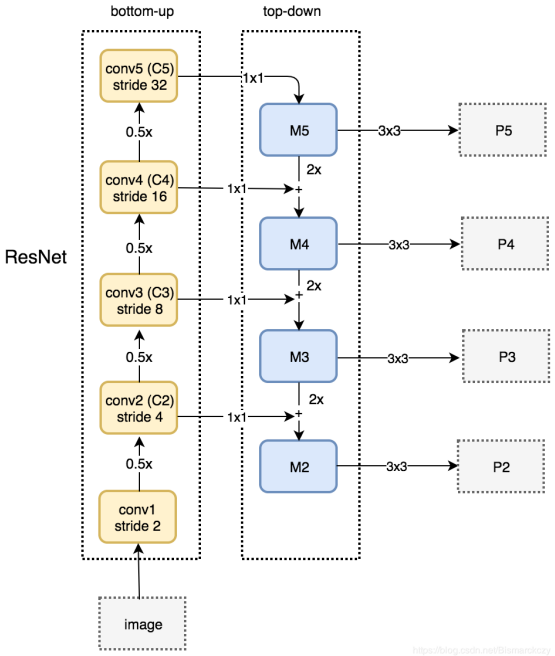

Figura 2.9: Estructura de un *backbone* para *Mask R-CNN*. Podemos ver el proceso de *bottom-up* realizado por la *CNN* y el *top-down* por la *FPN*. Imagen sacada de [26].

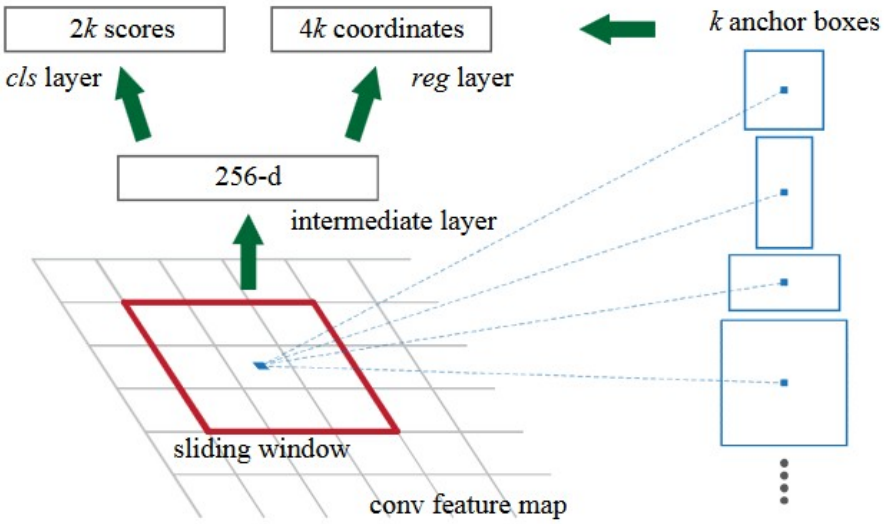

Figura 2.10: Funcionamiento de una *RPN*. Imagen sacada de [24].

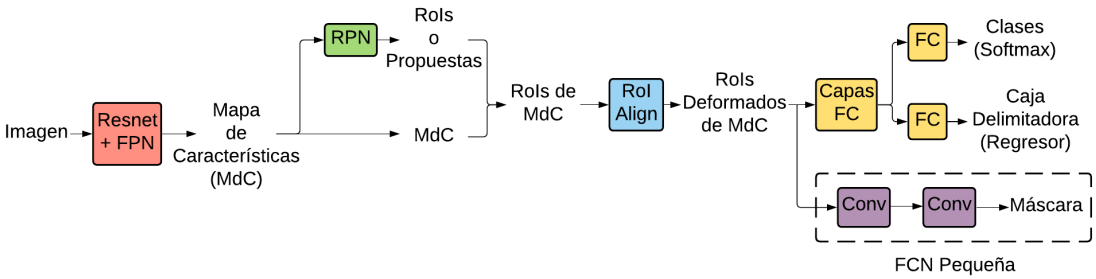

Figura 2.11: Arquitectura de *Mask R-CNN*. Una imagen de entrada pasa por el *backbone* (*Resnet*) con *FPN*, dando como resultado un Mapa de Características (*MdC*).Este se procesa por la *RPN* para obtener *RoIs*, que se utilizan para obtener los RoIs del mapa de características, que luego pasan por el *RoIAlign*. El resultado son*RoIs* deformados del *MdC*, estos pasan a las *FC* para obtener las clasificaciones y las cajas delimitadoras de los *RoIs*.

2.3 Recursos Hardware

Para poder entrenar arquitecturas basadas en redes neuronales convolucionales como *Mask R-CNN*, es necesaria una gran potencia de cálculo, dado que este proceso puede tardar días. Esta potencia la podemos conseguir mediante el uso de GPUs y su capacidad de procesar cálculos matriciales en paralelo que reducen los tiempos de entrenamiento.

Para llevar a cabo todos los experimentos de esta memoria se han utilizado las máquinas ofrecidas por el centro de investigación PRHLT de la Universidad Politécnica de Valencia, ya que vamos a trabajar con imágenes en 2D de gran tamaño y entrenar modelos de *Deep learning* que cuentan con muchos parámetros. Esta potencia ha sido obtenida mediante el uso de dos máquinas del centro con los siguiente componentes:

- 2 GPUs NVIDIA GeForce RTX 2080 con 8 GB cada GPU.
- 128 GB de memoria principal.

2.4 Recursos Software

Para llevar a cabo este trabajo hemos hecho uso del software *detectron2* [27], implementado mediante el entorno de trabajo *pytorch*. *Detectron2* proporciona una plataforma simple y a la vez potente para las tareas de visión por computador llevadas a cabo, permitiendo la fácil modificación del código y parámetros, tarea esencial en proyectos de investigación. Estos últimos años, *pytorch* se ha convertido en uno de los *framework* más populares en el mundo de *Machine Learning*, puesto que utiliza *python*, lenguaje de alto nivel que optimiza los datos utilizados en modelos de *Deep Learning*.

2.4.1. *PyTorch*

Pytorch es una biblioteca o *framework* de *Machine Learning*, de código abierto, basada en la biblioteca de *Torch* y utilizada ampliamente en proyectos de visión por computador y procesamiento del lenguaje. Se desarrolló por FAIR y se creó para ser flexible y modular en cuanto a la investigación en el campo del *Deep Learning*. Proporciona un paquete de *Python* diseñado para realizar cálculos numéricos, haciendo uso de la programación de tensores. Asimismo, permite su ejecución en GPU, utilizando el paralelismo de datos y distribuyendo el trabajo computacional para una mayor eficiencia y rapidez a la hora de entrenar modelos.

2.4.2. *Detectron2*

Detectron2 es una biblioteca modular de código abierto creada por *FAIR* para el uso de modelos de visión por computador basada en *PyTorch*. Es una mejora

de *Detectron* más flexible que utiliza *PyTorch* como modelo de programación más intuitivo.

Detectron2 incorpora el uso de arquitecturas como *Mask R-CNN* y permite la fácil modificación del código y los parámetros e hiperparámetros. Cuenta con una gran cantidad de modelos preentrenados y soporta cuatro tipos de tareas de visión por computado: *Object detection, semantic segmentation, instance segmentation* y *panoptic segmentation*.

2.4.3. *CUDA*

CUDA es una arquitectura de cálculo paralelo creada por *NVIDIA,* la cual aprovecha la potencia de cálculo proporcionada por las GPUs y ofrece un mayor rendimiento del sistema.

2.4.4. *Nvidia cuDNN*

Se utiliza como complemento a *CUDA*. Se trata de bibliotecas orientadas hacia técnicas de *Deep Learning* que aceleran la GPUs.

CAPÍTULO 3
Experimentos

En este capítulo vamos a observar con detalle los experimentos realizados en esta memoria. En primer lugar, veremos los diferentes modelos que utilizaremos para el problema planteado. A continuación, comentaremos las métricas de evaluación para más adelante poder comparar el rendimientos de los modelos y así poder elegir aquel que se adapte mejor a nuestras necesidades. Seguidamente, se evaluarán modelos con diferentes parámetros para encontrar el que mejor desempeño tenga en nuestra tarea. Para finalizar este capítulo, compararemos el caso base con los modelos que hayan logrado los mejores resultados. De este modo, podremos detectar las mejoras obtenidas y sacar conclusiones sobre este estudio.

3.1 Modelos Propuestos

Las redes neuronales propuestas como *backbone* para *Mask R-CNN* son redes neuronales convolucionales, las cuales son las encargadas de extraer las características necesarias para las siguientes etapas de la arquitectura. En estos modelos reside una parte significativa del tiempo de ejecución total, así como el uso y consumo de la memoria de la GPU.

En esta sección observaremos los modelos utilizados. Para empezar, contamos con algunas variantes de la arquitectura *Resnet* [21]. Hemos utilizado arquitecturas ya creadas como *Resnet 101* y *Resnet 50*. También hemos creado nuestras propias arquitecturas basadas en *Resnet*, como *Resnet 50b*, *Resnet 50c* y *Resnet 31*. Asimismo, hemos trabajado con la arquitectura *Vovnet* [22] y hemos hecho uso de sus diferentes modelos predefinidos, desde la arquitectura más simple, como *Vovnet 19Lite*, hasta arquitecturas de gran profundidad como *Vovnet 99*.

3.1.1. *Resnet*

Como ya hemos explicado en el marco teórico, *Resnet* es un tipo de red neuronal convolucional que utiliza conexiones residuales para poder propagar el gradiente hasta las primeras capas sin que este se degrade por el camino. *Resnet* está formada básicamente por un *stem*, parte inicial de la red que no tiene conexión residual y seguido de las *stages*. Para *Mask R-CNN*, *Resnet* está formada por 1 *stem*, seguido de 4 *stages* originalmente enumeradas de la 2 a la 5, las cuales pa-

san la información extraída a la *FPN* mediante las conexiones laterales de forma paralela.

Cada *stage* de *Resnet* está formada por los llamados *bottlenecks*. En *Resnet* tenemos dos tipos de *bottlenecks*, como podemos ver en la figura 3.1. A la izquierda tenemos un *basic bottleneck*, el cual está formado por una capa de convolución con un *kernel* 1×1 que reduce la cantidad de canales para poder procesarlos por la siguiente convolución con un *kernel* 3×3 que extrae nuevas características. Finalmente, se procesa por otra convolución con *kernel* 1×1 que aumenta de nuevo el número de canales y se fusiona con la entrada inicial (*input*) a través la conexión residual, que se realiza mediante una suma de ambas salidas que deben tener el mismo tamaño (*shape*).

A la derecha de la figura 3.1 podemos ver el *identity bottleneck*. Este se encuentra al principio de cada *stage* y funciona de la misma forma que el *basic bottleneck*, con la diferencia de que añade una convolución con un *kernel* 1x1 en la conexión residual para poder igualar el número de canales de entrada (*input*) con la salida y sumarlos al final del *bottleneck*.

Dependiendo de la profundidad de la red, cada *stage* está formada por un *identity bottleneck*, seguido de una cantidad de *basic bottlenecks*. A mayor profundidad, más *basic bottlenecks* habrá.

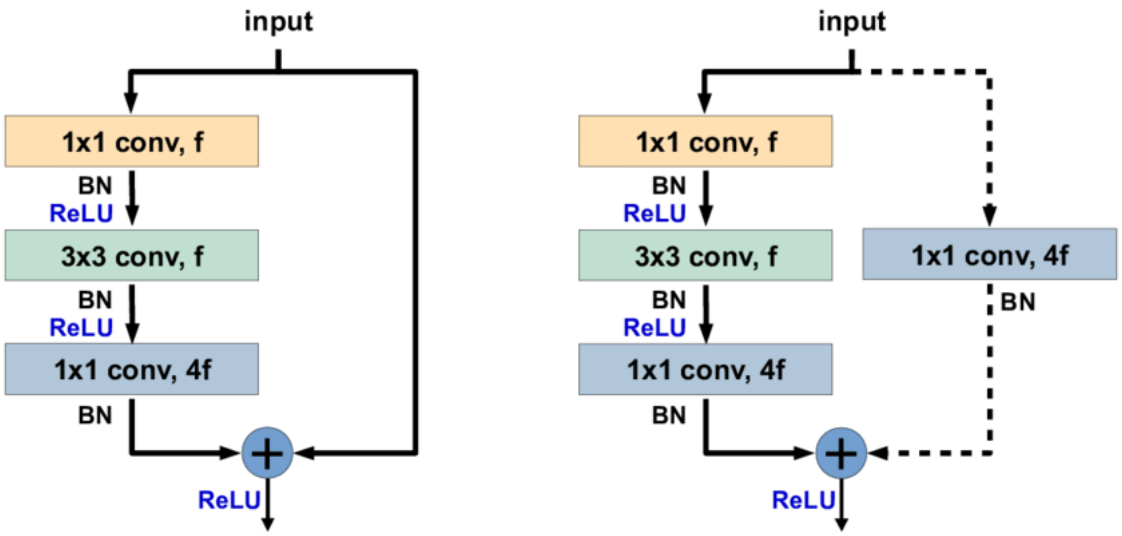

Figura 3.1: A la izquierda tenemos el *Basic Bottleneck* de *Resnet*. A la derecha podemos ver el *Identity Bottleneck* el cual se sitúa al inicio de cada *stage*.

Por lo que respecta al *stem*, hemos modificado el *stem* original de *Resnet* para comprobar el comportamiento del modelo en cuanto a precisión. Por lo tanto, contamos con las arquitecturas *Resnet50a , b y c*. Cada una de ellas hace referencia a un tipo diferente de *stem*, siendo *Resnet 50a* el *stem* original. Podemos ver las configuraciones de los diferentes *stems* en la figura 3.2.

En cuanto al *stem* de *Resnet 50b*, hemos optado por modificar la profundidad de este para ver el impacto que podría tener utilizar un mayor número de convoluciones, ya que, a mayor profundidad, más posibles características encontraría el *stem*. Finalmente, el *stem* de *Resnet 50c* es una fusión de los dos *stems* anteriores. Utiliza conexiones paralelas para aprovechar, por un lado, las ventajas de apilar

tres capas convolucionales con un *kernel* 3×3, y por otro lado, para beneficiarse de la capacidad de extracción de características de una capa convolucional con un *kernel* más grande, como es del de 7×7. Ambos caminos se fusionan antes de la operación de *average pooling*, por lo que es necesario que las salidas de ambos caminos tengan el mismo tamaño (*shape*).

Tal y como se ha descrito anteriormente, hemos utilizado cinco arquitecturas de *Resnet* diferentes. Por una parte, analizaremos el impacto de la profundidad de la red. Por otra parte, examinaremos cómo afecta el *stem* inicial.

Para más adelante poder comparar las diferentes ejecuciones con distintos modelos, se guardará el tiempo de ejecución, así como el uso de la memoria GPU. También se generarán los resultados en cuanto a precisión en las tareas de segmentación y detección de objetos, para las que utilizaremos el conjunto de *test* del *dataset*.

En la tabla 3.1, podemos ver las diferentes configuraciones de las cinco arquitecturas de *Resnet*. Todas las arquitecturas que encontramos tienen un aspecto en común: el aumento de manera lineal en número de filtros empleados en cada *stage*. En el primer *stage* tenemos 64 canales de entrada y 256 de salida. Para el segundo *stage*, el número tres, contamos con 128 canales en la entrada y 512 en la salida. El *stage* cuatro recibe como entrada 256 canales y tiene una salida de 1024. Finalmente, para el *stage* número cinco, tenemos 512 canales en la entrada y 2048 en la salida.

En cuanto a las arquitecturas, encontramos en primer lugar la *Resnet 31*, la arquitectura menos profunda, que consta de 31 convoluciones en total. Los *stages* 2 y 4 están formados por un *identity bottleneck*, seguido de un *basic bottleneck*. Los *stages* 3 y 5 están formados por tres *bottlenecks*, un *identity bottleneck* al inicio de cada *stage* y dos *basic bottlenecks*.

Por lo que concierna a las arquitecturas de *Resnet 50*, explicaremos la configuración de *Resnet 50a*, ya que la única diferencia entre los tres tipos de modelos de *Resnet 50* es el *stem* inicial. *Resnet 50* está formada por un *identity bottleneck* al inicio de todas sus *stages* y por tres *basic bottlenecks* para el *stage* número 2, cuatro para el *stage* 3, seis para el *stage* 4 y, finalmente, tres para el *stage* 5. Para *Resnet 50a* contamos con un total de 50 capas convolucionales, mientras que para *Resnet 50b* contamos con 52 capas y para *Resnet 50c* 53.

Por último, podemos apreciar la arquitectura de *Resnet 101*, formada por el *stem* original. Como las arquitecturas anteriores, cuenta con un *identity bottleneck* al inicio de cada *stage* y con dos *basic bottlenecks* para el primer *stage* y tres para el segundo. Seguidamente, vemos 22 *basic bottlenecks* para el *stage* 4 y, para finalizar, dos *basic bottlenecks* en el último *stage*.

3.1.2. *Vovnet*

Vovnet introduce el módulo *One-Shot Aggregation* (*OSA*), el cual realiza una agregación de todas las salidas de las convoluciones en la última capa a la vez. Dicho módulo se explicará a continuación.

Tabla 3.1: Topologías *Resnet*.

Stage	Resnet 31	Resnet 50A	Resnet 50B	Resnet 50C	Resnet 101
Stem	Conv 7x7, 64 , s=2 , p=3 AvgPool 3x3 , s=2	Conv 7x7, 64 , s=2 , p=3 AvgPool 3x3 , s=2	Conv 3x3, 64, s=2, p=2 Conv 3x3, 64, s=1, p=1 Conv 3x3, 64, s=1, p=1 AvgPool 3x3 , s=2	[Conv 3x3, 64, s=2, p=2 Conv 3x3, 64, s=1, p=1 Conv 3x3, 64, s=1, p=1] & [Conv 7x7, 64 , s=2 , p=3] AvgPool 3x3 , s=2	Conv 7x7, 64 , s=2 , p=3 AvgPool 3x3 , s=2
Stage 2	[Conv 1x1 , 64 Conv 3x3 , 64 Conv 1x1, 256] x 2	[Conv 1x1 , 64 Conv 3x3 , 64 Conv 1x1, 256] x 3	[Conv 1x1 , 64 Conv 3x3 , 64 Conv 1x1, 256] x 3	[Conv 1x1 , 64 Conv 3x3 , 64 Conv 1x1, 256] x 3	[Conv 1x1 , 64 Conv 3x3 , 64 Conv 1x1, 256] x 3
Stage 3	[Conv 1x1, 128 Conv 3x3, 128 Conv 1x1 512] x 3	[Conv 1x1, 128 Conv 3x3, 128 Conv 1x1 512] x 4	[Conv 1x1, 128 Conv 3x3, 128 Conv 1x1 512] x 4	[Conv 1x1, 128 Conv 3x3, 128 Conv 1x1 512] x 4	[Conv 1x1, 128 Conv 3x3, 128 Conv 1x1 512] x 4
Stage 4	[Conv 1x1. 256 Conv 3x3, 256 Conv, 1x1 1024] x 2	[Conv 1x1. 256 Conv 3x3, 256 Conv, 1x1 1024] x 6	[Conv 1x1. 256 Conv 3x3, 256 Conv, 1x1 1024] x 6	[Conv 1x1. 256 Conv 3x3, 256 Conv, 1x1 1024] x 6	[Conv 1x1. 256 Conv 3x3, 256 Conv, 1x1 1024] x 23
Stage 5	[Conv 1x1, 512 Conv 3x3, 512 Conv 1x1, 2048] x 3	[Conv 1x1, 512 Conv 3x3, 512 Conv 1x1, 2048] x 3	[Conv 1x1, 512 Conv 3x3, 512 Conv 1x1, 2048] x 3	[Conv 1x1, 512 Conv 3x3, 512 Conv 1x1, 2048] x 3	[Conv 1x1, 512 Conv 3x3, 512 Conv 1x1, 2048] x 3

Tabla 3.2: Topologías *Vovnet*.

Stage	Vovnet 19 Lite	Vovnet 19	Vovnet 39	Vovnet 57	Vovnet 99
Stem	Conv 3x3, 64, s=2 Conv 3x3, 64, s=1 Conv 3x3, 128, s=1	Conv 3x3, 64, s=2 Conv 3x3, 64, s=1 Conv 3x3, 128, s=1	Conv 3x3, 64, s=2 Conv 3x3, 64, s=1 Conv 3x3, 128, s=1	Conv 3x3, 64, s=2 Conv 3x3, 64, s=1 Conv 3x3, 128, s=1	Conv 3x3, 64, s=2 Conv 3x3, 64, s=1 Conv 3x3, 128, s=1
OSA Module Stage 2	[[Con 3x3, 128]x 3 concat & Conv 1x1 256] x1	[[Con 3x3, 128]x 3 concat & Conv 1x1 256] x1	[[Con 3x3, 128]x 5 concat & Conv 1x1 256] x1	[[Con 3x3, 128]x 5 concat & Conv 1x1 256] x1	[[Con 3x3, 128]x 5 concat & Conv 1x1 256] x1
OSA Module Stage 3	MaxPool 3x3 s=2 [[Con 3x3, 160]x 3 concat & Conv 1x1 512] x1	MaxPool 3x3 s=2 [[Con 3x3, 160]x 3 concat & Conv 1x1 512] x1	MaxPool 3x3 s=2 [[Con 3x3, 160]x 5 concat & Conv 1x1 512] x1	MaxPool 3x3 s=2 [[Con 3x3, 160]x 5 concat & Conv 1x1 512] x1	MaxPool 3x3 s=2 [[Con 3x3, 160]x 5 concat & Conv 1x1 512] x3
OSA Module Stage 4	MaxPool 3x3 s=2 [[Con 3x3, 192]x 3 concat & Conv 1x1 762] x1	MaxPool 3x3 s=2 [[Con 3x3, 192]x 3 concat & Conv 1x1 768] x1	MaxPool 3x3 s=2 [[Con 3x3, 192]x 5 concat & Conv 1x1 768] x2	MaxPool 3x3 s=2 [[Con 3x3, 192]x 5 concat & Conv 1x1 768] x4	MaxPool 3x3 s=2 [[Con 3x3, 192]x 5 concat & Conv 1x1 768] x8
OSA Module Stage 5	MaxPool 3x3 s=2 [[Con 3x3, 224]x 3 concat & Conv 1x1 1024] x1	MaxPool 3x3 s=2 [[Con 3x3, 224]x 3 concat & Conv 1x1 1024] x1	MaxPool 3x3 s=2 [[Con 3x3, 224]x 5 concat & Conv 1x1 1024] x2	MaxPool 3x3 s=2 [[Con 3x3, 224]x 5 concat & Conv 1x1 1024] x3	MaxPool 3x3 s=2 [[Con 3x3, 224]x 5 concat & Conv 1x1 1024] x3

Las arquitecturas de *Vovnet*, al igual que las de *Resnet*, están formadas por un *stem* seguido de 4 *stages*, de la 2 a la 5. Estas *stages* están formadas por uno o más módulos OSA dependiendo de la profundidad de la red.

Cada módulo OSA está formado por cinco capas convolucionales con *kernel* 3x3 secuenciales, tal y como podemos apreciar en la figura 3.6. Todas las salidas de las capas se concatenan con profundidad. Posteriormente, encontramos una convolución con *kernel* 1×1 que aumenta el número de canales de la salida del módulo, al igual que en los *bottlenecks* de*Resnet*.

Para este trabajo hemos utilizado cinco modelos distintos de *Vovnet* para ver cómo afecta la profundidad y la complejidad de la red en cuanto a los resultados y los tiempos de ejecución. En la tabla 3.2 podemos apreciar las diferentes configuraciones de las arquitecturas de *Vovnet* empleadas.

En cuanto a las arquitecturas menos profundas, encontramos la *Vovnet 19Lite* y la *Vovnet 19*, que constan de 19 convoluciones en total. Las cuatro *stages* están formadas por un módulo *OSA*, el cual contiene tres convoluciones con *kernel* 3×3 de forma secuencial. Una vez realizada la agregación de las tres capas convolucionales, se aplica una convolución con *kernel* 1×1 para aumentar el número del canales de la salida del módulo.

Finalmente, contamos con una configuración distinta en el módulo OSA para las topologías de *Vovnet 39, Vovnet 57* y *Vovnet 99*. Este módulo está formado por cinco capas convolucionales con *kernel* 3×3 de forma secuencial, la salida de las cuales se concatena, siendo procesada al final del módulo por una convolución con *kernel* 1×1 para aumentar el número de canales de salida. La diferencia entre las topologías reside en la cantidad de módulos *OSA*.

Vovnet 39 contiene un módulo para las *stages* 2 y 3, seguido de dos módulos para las *stages* 4 y 5. En cuanto a *Vovnet 57*, las dos primeras *stages* también están constituidas por un único módulo OSA. La *stage* 4 contiene cuatro módulos y la *stage* 5 tres.

Por último, *Vovnet 99*, al igual que todas las topologías anteriores, cuenta con un solo módulo para el primer *stage* . El segundo *stage* está compuesto por tres módulos, mientras que el *stage* 4 contiene un total de ocho módulos OSA. Para terminar, el último *stage* está formado por tres módulos *OSA*.

Figura 3.2: Comparación de los 3 *stems* utilizados en los experimentos.

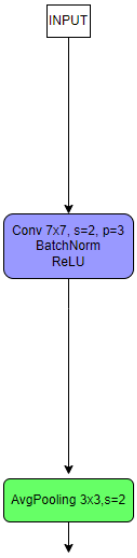

Figura 3.3: *Stem* Original de *Resnet*.

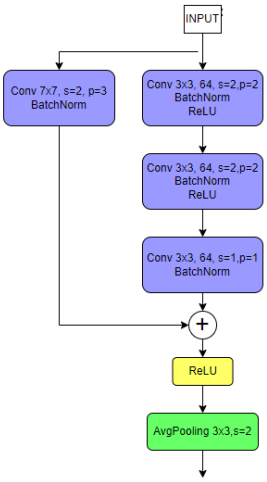

Figura 3.5: *Stem* C. Este *stem* es una combinación en paralelo de los dos *stems* anteriores.

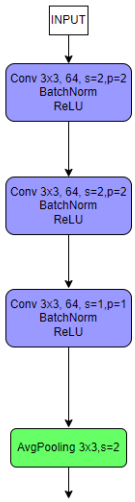

Figura 3.4: *Stem* B. Este esta formado por 3 capas convolucionales con *kernel* 3×3 para sustituir el funcionamiento de la capa 7×7 del *stem* original.

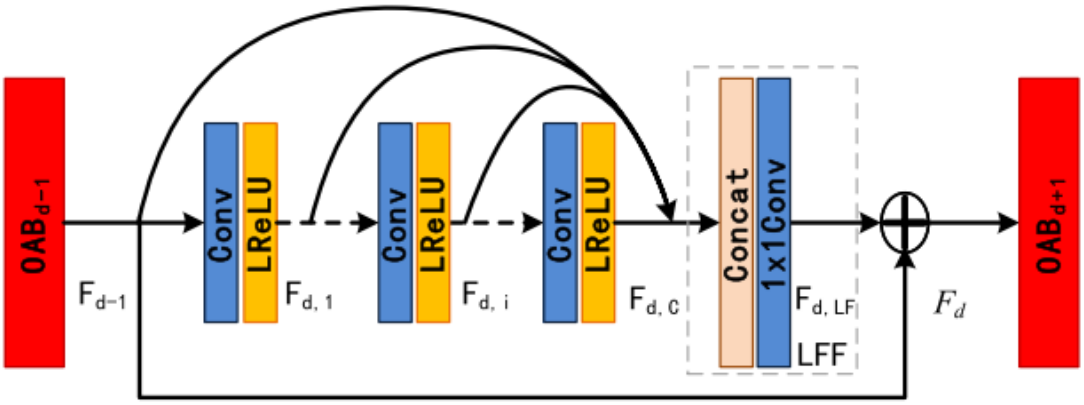

Figura 3.6: Representación del funcionamiento de un módulo *OSA* de la arquitectura *Vovnet*. Este ejemplo cuenta con 3 capas convolucionales con *kernel* 3×3 en su interior. Imagen sacada de [28].

3.2 Métricas de evaluación de resultados

Para evaluar los modelos utilizados como *backbone*, tendremos en cuenta la calidad y la eficiencia de los resultados obtenidos en el conjunto de test. En esta sección, presentaremos las métricas utilizadas para ambas mediciones. No obstante, antes de proceder a dicha presentación, es de gran interés explicar las tareas que realiza la arquitectura *Mask R-CNN*. La segmentación semántica de objetos es una de las tareas que lleva a cabo esta arquitectura. Este proceso consiste en clasificar cada píxel según su categoría. De ahora en adelante, acortaremos el nombre de dicho proceso y lo denominaremos segmentación. La otra tarea que realiza *Mask R-CNN* es la de detección de objetos. Esta devuelve las coordenadas de una caja que delimita el objeto encontrado. Estas cajas se conocen como *bounding boxes*. A partir de este momento, designaremos esta tarea *bounding box*

Para finalizar, la segmentación y el *bounding box* serán evualuados con las métricas de calidad.

3.2.1. Métricas de calidad

Tanto para la detección de objetos como para la segmentación de imágenes es necesario poder medir la calidad de las predicciones del algoritmo utilizado. La precisión meda promedio o *mean average precision* (mAP) es una métrica estándar calculada a partir de la precisión promedio AP.

Como consecuencia del uso de *Detectron2*, el cálculo de las métricas de calidad se encuentra en formato *COCO*. Este formato utiliza distintos umbrales de decisión, lo que permite obtener diferentes aspectos de precisión. El origen de este formato proviene de la creación de un *dataset* llamado *COCO* [29]. Debido a la enorme popularidad que alcanzó este, la comunidad empezó a utilizar el formato que empleaba el *dataset* para evaluar los resultados, por lo que terminó por denominarse formato *COCO*.

Este formato no distingue entre AP y mAP. Para evaluar la precisión de la segmentación y el *bounding box* se utilizan las siguientes medidas:

- AP: Es la precisión promedio de todas las categorías.
- AP50: Es la precisión media que se obtiene al emplear el *set* de validación con un umbral de decisión de 0.5.
- AP75: Es la precisión media que se obtiene al emplear el *set* de validación con un umbral de decisión de 0.75.
- APs o APSmall: Es la precisión media que se obtiene al emplear el *set* de validación únicamente en los objetos pequeños con un área menor a 32^2 píxeles. Para el *dataset* utilizado en esta memoria no contamos con objetos de dicho tamaño, por lo que estos resultados no aparecerán en los resultados que hemos obtenido a lo largo de los experimentos.
- APm o APmedium: Se trata de la precisión media que se obtiene al emplear el *set* de validación únicamente en los objetos medianos con un área entre 32^2 y 96^2 píxeles.

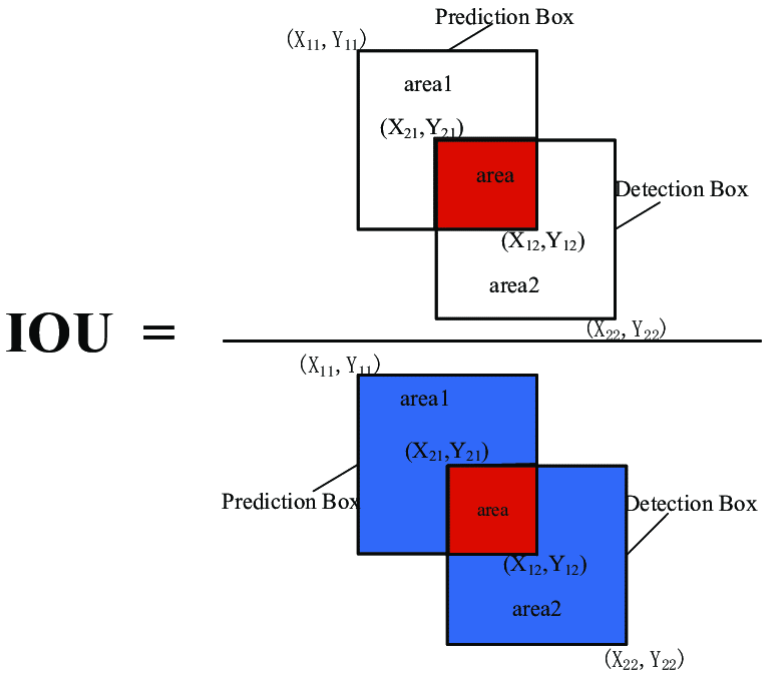

Figura 3.7: Intersection over union. Fórmula de ejemplo del cálculo de I*IoU*. Imagen obtenida de [30].

- APl o APlarge: Es la precisión media que se obtiene al emplear el *set* de validación únicamente en los objetos grandes con un área mayor a 96^2 píxeles

Para entender mejor estos parámetros, es esencial tener claros una serie de conceptos.

En primer lugar, debemos entender el significado de la medida conocida como *Intersection over Union* (*IoU*) o Intersección sobre la unión. Esta métrica se obtiene al dividir el área de la intersección de cuadro delimitador real(*ground truth box*) y el cuadro predicho(*detected box*) entre el área total, formada por la unión entre ambos (*ground truth* + *detected box*). Podemos ver una representación gráfica en la figura 3.7.

Dicho cuadro delimitador o *ground truth box*, referencia real al objeto, se crea a mano e indica las coordenadas de las esquinas que lo forman, lo que genera un valor entre 0 y 1. Cuando más cercano a 1 esté, más precisa será la predicción. Puesto que conseguir un $IoU = 1$ es una tarea muy complicada, diferentes umbrales son aplicados con la finalidad de determinar si una predicción es buena. Estos umbrales, denominados umbrales de decisión, varían dependiendo del tamaño de los objetos a detectar, por lo que puede haber umbrales para objetos pequeños, medianos o grandes. Asimismo, estos puede modificarse en función de la complejidad y el tipo de problema, por lo que cualquier algoritmo que proporcione como resultado cajas delimitadoras o *bounding boxes* puede evaluarse con *IoU*.

Esta idea del cálculo del AP en la tarea de *bounding box* se puede extender a la tarea de segmentación y las máscaras predichas. En este caso, tomando en consideración la máscara original y la predicha, se comparan ambas punto a punto, es decir, píxel a píxel. Podemos calcular la *IoU* de las máscaras de la misma manera

que con los *bounding boxes*. Para el caso de la segmentación sería la división entre la intersección de la máscara original y la predicha con la unión de ambas.

Seguidamente, explicaremos el cálculo de AP. La definición general de la Precisión Promedio (AP) es encontrar el área bajo la curva de precisión-recuperación. El resultado de integrar la curva mediante la ecuación 3.1.

$$AP = (\int_0^1 p(r)dr) \tag{3.1}$$

El AP obtiene un resultado diferente para cada una de las clases que se encuentran en la imagen, por lo que se usa el mAP (Precisión Promedio) para hallar el AP promedio de todas las clases. Podemos calcularlo con el uso de la ecuación 3.2.

$$mAP = \frac{(\sum_{i=1}^{k} AP_i)}{K} \tag{3.2}$$

En este trabajo, para las tareas de segmentación y *bounding box*, se han utilizado las métricas de COCO empleadas por *Detectron2*, concretamente el *AP, AP50, AP75, APm y APl*.

3.2.2. Métricas de eficiencia

Con respecto a las métricas de eficiencia, nos serviremos de los tiempos de ejecución de la arquitectura *Mask R-CNN* además del uso de memoria de las GPUs.

Como observaremos con más profundidad en la sección del impacto del número de *epochs*, *Detectron2* funciona por iteraciones, por lo que es crucial pasar esta métrica a *epochs* con la finalidad de comparar adecuadamente los modelos. Mediremos los tiempos de ejecución bajo el mismo número de *epochs* para equiparar las diferencias en los tiempos de ejecución, buscando un balance entre una reducción de tiempos y mantener la calidad de los resultados.

Con el objetivo de seguir la línea de los experimentos, explicaremos previamente la diferencia entre *epohs* e iteraciones. *Epoch* es el tiempo necesario para que un modelo observe todas las imágenes de un *dataset*. El número de iteraciones I viene dado por el tamaño del *dataset*, $|D_{tr}|$ y el *batch* utilizado b_{size} , ya que, una *epoch* tendrá tantas iteraciones como cantidad de posibles *batches* tenga el *dataset* completo.

$$I = \frac{|D_{tr}|}{b_{size}} \tag{3.3}$$

Para medir el uso de la memoria, se mantendrá monitorizada la GPU mediante *scripts* en *bash* en constante ejecución. Como las GPUs tienen un máximo de memoria disponible, concretamente 7982MiB, mediremos el consumo medio con la intención de buscar la configuración de parámetros y el *backbone* con bajo consumo de memoria sin perder calidad.

3.3 Descripción del *dataset VORAU-253*

Para realizar este trabajo hemos utilizado el *dataset VORAU-253*. En este capítulo examinaremos con más detenimiento este *dataset* y las técnicas utilizadas junto a los recursos empleados para resolver el problema.

Vorau Abbey library Cod. 253 (VORAU-253), fechado alrededor del año 1450, es un *dataset* de manuscritos musicales con notación gótica alemana. Está etiquetado manualmente y se utiliza para la tarea de reconocimiento de música escrita a mano (HMR, *Handwritten Music Recognition*). El *dataset* está formado por un total de 228 imágenes, dividido al azar con 128 imágenes para el conjunto de entrenamiento y 100 para el test.

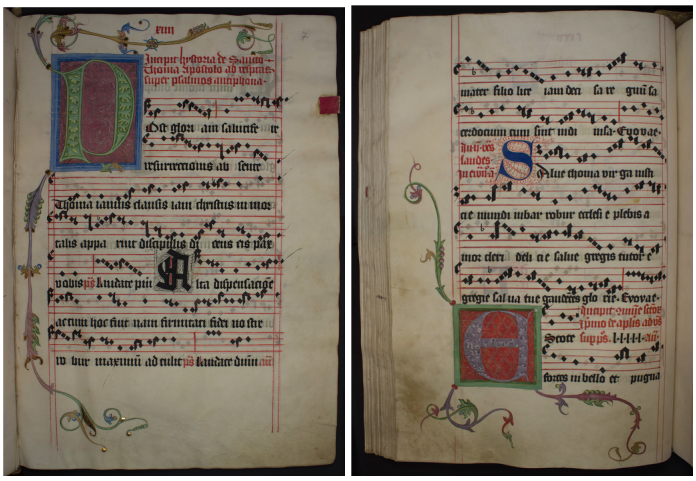

Figura 3.8: Ejemplo imágenes *dataset VORAU*.

Asimismo, el *dataset* está formado por 3 tipos de regiones diferentes, *staff*, *lyrics* y *dorp-capital*:

- *Staff*: Ilustra los conjuntos de líneas y espacios horizontales con los tonos musicales correspondientes.
- *Lyrics*: Reproduce el texto que aparece debajo del *staff*. Se trata de las palabras que se cantan.
- *Drop-capital*: Representa la letra que puede aparecer al principio de una palabra o de una línea de *Staff* .

Podemos ver un ejemplo más detallado de estos tres tipos de regiones en la imagen 3.9.

3.4 Experimentos y Resultados

En este apartado, explicaremos el caso base del que partimos al realizar esta memoria y nombraremos los resultados y los parámetros que se utilizaron en la ejecución con el propósito de poder compararlos posteriormente con los resultados de nuestros experimentos.

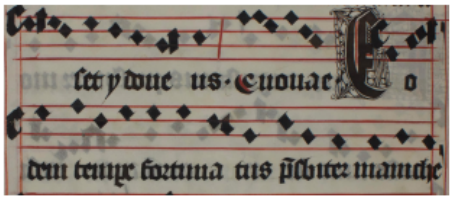

Figura 3.9: Ejemplo de los 3 tipos de regiones de *VORAU*.

Seguidamente, estudiaremos el impacto de las diferentes modificaciones en los parámetros y analizaremos la manera en que estas afectan al rendimiento, a los tiempos de ejecución, al uso de la memoria GPU y a la precisión.

Finalmente, una vez obtenidos los parámetros óptimos, se realizarán las diferentes pruebas con las distintas topologías de redes nombradas en los apartados anteriores con el fin de analizar su desempeño con los parámetros obtenidos. De este modo, podremos encontrar la mejor configuración parámetros-topología y solucionar el problema planteado en esta memoria.

3.4.1. Caso Base

En este trabajo partimos de los experimentos realizados por Lorenzo Quirós en su Tesis Doctoral [4], donde se utiliza la arquitectura *Mask R-CNN* para las tareas de segmentación y detección de objetos del *dataset VORAU-253*.

Los resultados obtenidos al replicar los experimentos se muestran en la tabla 3.3. Estos experimentos han sido realizados con un *batch* de cuatro imágenes, un *learning rate* de 0.02 y la topología *Resnet 50* como *backbone*. Con esta configuración, se han obtenido unos tiempos de ejecución de 10 horas y 48 minutos. Cabe destacar que el máximo tamaño de *batch* posible con los recursos disponibles es de seis imágenes para *Resnet 50*.

Con respecto al uso de memoria, la GPU1 tiene un consumo medio de 4941MiB, a diferencia de la GPU2, que posee un consumo de 4901MiB.

Tabla 3.3: Resultados Experimento Caso Base. Se han utilizado las métricas de *COCO* para las tareas de segmentación y *bounding box*.

Task	AP	AP50	AP75	APm	APl
Bounding Box	68.66	96.45	80.48	44.88	68.85
Segmentation	67.38	95.86	78.25	38.71	67.68

3.4.2. Efecto de la cantidad de *epochs* en el entrenamiento

En esta sección, llevaremos a cabo un estudio del impacto del número de *epochs* con respecto a la precisión del modelo. Primeramente, cabe señalar que *detectron2* funciona por iteraciones, por lo que durante la primera etapa de experimentación se pudo apreciar que, al modificar el tamaño de *batch size* y lanzar el experimento con las mismas iteraciones, los tiempos de entrenamiento aumentaban o disminuían significativamente. Todos los resultados se obtuvieron utilizando el conjunto de *test* del *dataset*.

Como hemos mencionado, no podemos comparar los diferentes experimentos bajo el número de iteraciones. Es esencial convertir el número de iteraciones en *Epochs*. El caso base está ejecutado con los valores por defecto de 90.000 iteraciones, por lo que modificaremos este parámetro y, en consecuencia, el número de *epochs*. El objetivo de este apartado es el estudio del impacto de la cantidad de *epochs* y la observación de los resultados que se obtienen en cuanto a precisión y tiempo de ejecución.

Por último, cabe destacar que todos los experimentos de este apartado se han realizado con *Resnet 50* como *backbone*, con el *batch size* de dos imágenes y con el *learning rate* de 0.02.

Inicialmente, un modelo entrenado con *Resnet 50* como *backbone*, con un *batch* de cuatro imágenes y con 90.000 iteraciones, reflejaba un tiempo de ejecución de 10 horas y 48 minutos. Sin embargo, únicamente modificando el tamaño de *batch* a dos imágenes se obtenía como resultado un tiempo de ejecución de 5 horas y 58 minutos. Estos dos experimentos, por tanto, no son comparables. Es imprescindible equiparar bajo el mismo número de *epochs* y no de iteraciones, por lo que cabe señalar cuál es la diferencia entre ambos casos.

Tal y como se mencionó en el apartado de redes neuronales, una *epochs* es el número de iteraciones necesarias para ver el conjunto de *train* del *dataset*. Una vez comprendemos esto, podemos calcular cuántas iteraciones son necesarias para un número concreto de *epochs* y un *batch* determinado. Para el cálculo de las *epochs* tendremos, por tanto, que el número de *epochs* E es igual al *batch size* multiplicado por el número de iteraciones y dividido por el tamaño del *dataset*.

En la ecuación 3.4 observamos cómo calcular el número de *epochs* en función de un número determinado de iteraciones y de *batch size* dados, siendo, por un lado, b_{size} el tamaño del *batch size* y el número de iteraciones y, por otro, $|D_{\text{tr}}|$ el tamaño del conjunto de *train* del *dataset*.

$$E = \frac{i * b_{\text{size}}}{|D_{\text{tr}}|} \tag{3.4}$$

Para el caso comentado, con un *batch* de dos imágenes tenemos:

$$E_{b=2} = \frac{90000*2}{128} = 1407$$

Por otro lado, con un *batch* de cuatro imágenes tenemos:

$$E_{b=4} = \frac{90000*4}{128} = 2812$$

Podemos apreciar la imposibilidad de comparar estos dos experimentos, puesto que poseemos el doble de *epochs* para un *batch* de cuatro imágenes. Pese a que el número de *epochs* es significativamente mayor, los resultados son, tal y como podemos observar en las tablas 3.4 y 3.5, prácticamente idénticos en cuanto a precisión.

Tabla 3.4: Comparación resultados caso Base con *batch* 2 y 4. Resultados para la tarea de *Bounding Box*.

Batch Size	AP	AP50	AP75	APm	APl
2	69.43	97.68	80.84	43.91	69.97
4	68.97	96.94	81.63	43.63	69.23

Tabla 3.5: Comparación resultados experimento base con *batch* 2 y 4. Resultados de la tarea de segmentación.

Batch Size	AP	AP50	AP75	APm	APl
2	67.82	96.10	78.41	36.70	68.14
4	67.41	96.19	78.66	40.84	67.64

Tabla 3.6: Comparación tiempos de entrenamiento con *batch* 2 y 4 del caso base.

Batch Size	*Trainig Time*
2	5h 58min
4	10h 48min

Como podemos observar en los resultados anteriores, pese a que la configuración con un *batch* de cuatro imágenes se realicen el doble de *epochs*, la precisión de los resultados es menor a la obtenida con la configuración con un *batch* de dos imágenes, lo que es un indicador de que la red tiene un sobre entrenamiento.

Sin embargo, como podemos ver en la tabla 3.6, los tiempos son significativamente menores. Esto ha dado pie a experimentar con diferentes cantidades de *epochs*, con la intención de encontrar un valor óptimo.

Una vez sabemos cómo calcular el número de iteraciones para una cantidad de *epochs* concreta, hemos realizado experimentos con 900, 450, 225 y 115 *epochs*. La configuración para estos experimentos es de *Resnet 50* como *backbone*, un *batch* de dos imágenes y un *learning rate* de 0.02.

Asimismo, como bien podemos comprobar en la gráfica 3.10, aunque la cantidad de *epochs* disminuya, el AP promedio se mantiene estable e incluso mejora tanto para la segmentación como para *Bounding Box*, lo que confirma el sobreentrenamiento de la red.

Una vez obtenidos los resultados, hemos optado por utilizar 225 y 450 *epochs* en los siguientes experimentos. Pese a que con las ejecuciones con 115 *epochs* se consigue el menor tiempo de ejecución, los resultados en cuanto a precisión disminuyen, por lo que se ha optado por no tener en cuenta esta cantidad.

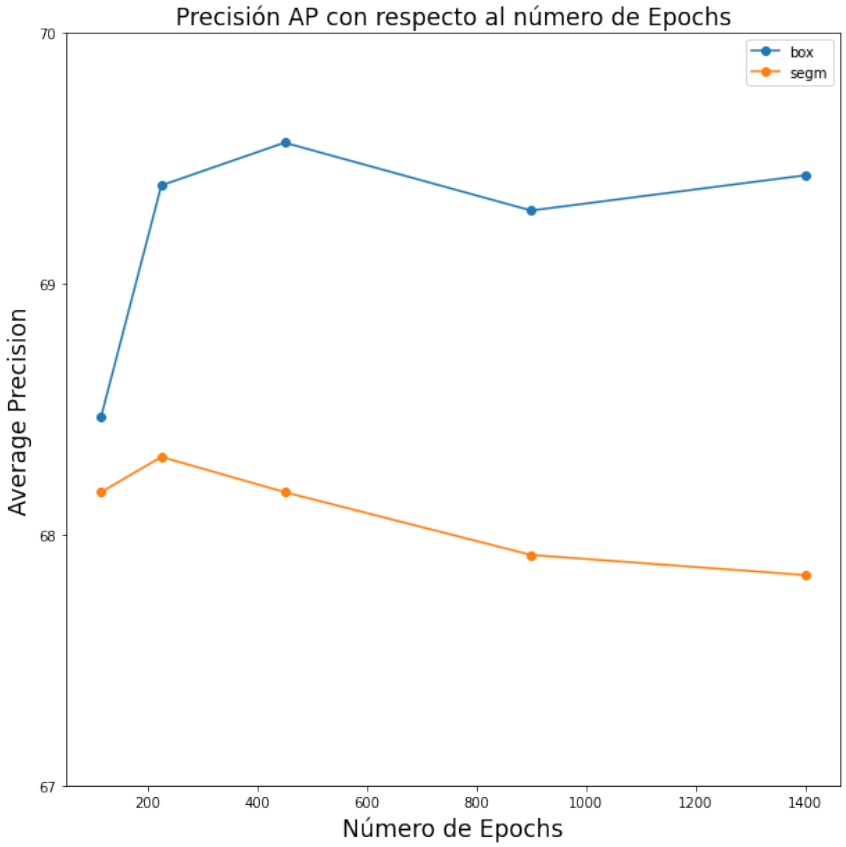

Figura 3.10: Comparación de los APs con respecto a la cantidad de *epochs*.

Podemos observar los resultados de la ejecución con 450 y 225 *epochs* en la tabla 3.7.

Tabla 3.7: Resultados de la ejecución con 450 y 225 *epochs*.

Epochs	*Task*	*Task* AP	AP50	AP75	APm	APl
450	*Bounding Box*	69.56	97.24	81.89	40.97	69.85
	Segmentation	68.17	96.25	79.48	37.36	68.43
225	*Bounding Box*	69.39	97.42	81.86	46.89	69.63
	Segmentation	68.31	96.26	79.51	29.35	68.68

Por último, cabe mencionar que, al reducir el número de *epochs* a la mitad, los tiempos de entrenamiento también se reducen a la mitad. Con la ejecución con 450 *epochs*, obtenemos un tiempo de entrenamiento de 1 hora y 54 minutos, mientras que, con 225 *epochs*, obtenemos un tiempo de 57 minutos.

3.4.3. Ajuste del *learning rate*

El *step*, parámetro ofrecido por *Detectron2*, ha sido otro objeto de experimentación en esta memoria. A lo largo de la ejecución, el *learning rate* puede modificarse para reducir su valor. Cuando se llega al *step* indicado, el valor que tiene el *learning rate* en la *epoch* en ese momento se multiplica por el factor γ.

Si observamos la figura 3.11, podemos apreciar un salto en la curva de pérdida en la *epoch* 150. Esto se debe a que en esta *epoch* se aplica el factor *GAMMA* en el *learning rate*. En este apartado, nos centraremos en modificar dicho *step* con la finalidad de descubrir de qué manera influye en cuanto a la precisión de los resultados.

Si observamos la configuración de *detectron2*, podemos apreciar que el parámetro *step* está configurado por iteraciones. Podemos indicar en qué momento se va a decrementar el *learning rate* con un factor *GAMMA*, así como señalar la cantidad de decrementos que se van a realizar a lo largo de la ejecución. En el caso base con 2813 *epochs*, contamos con una configuración del *step* de 1857 y 2500 *epochs*, lo que corresponde con un 66,66 % y un 88,88 % de total de *epochs*. Estos son los valores por defecto que tiene *detectron2* para el parámetro *step*.

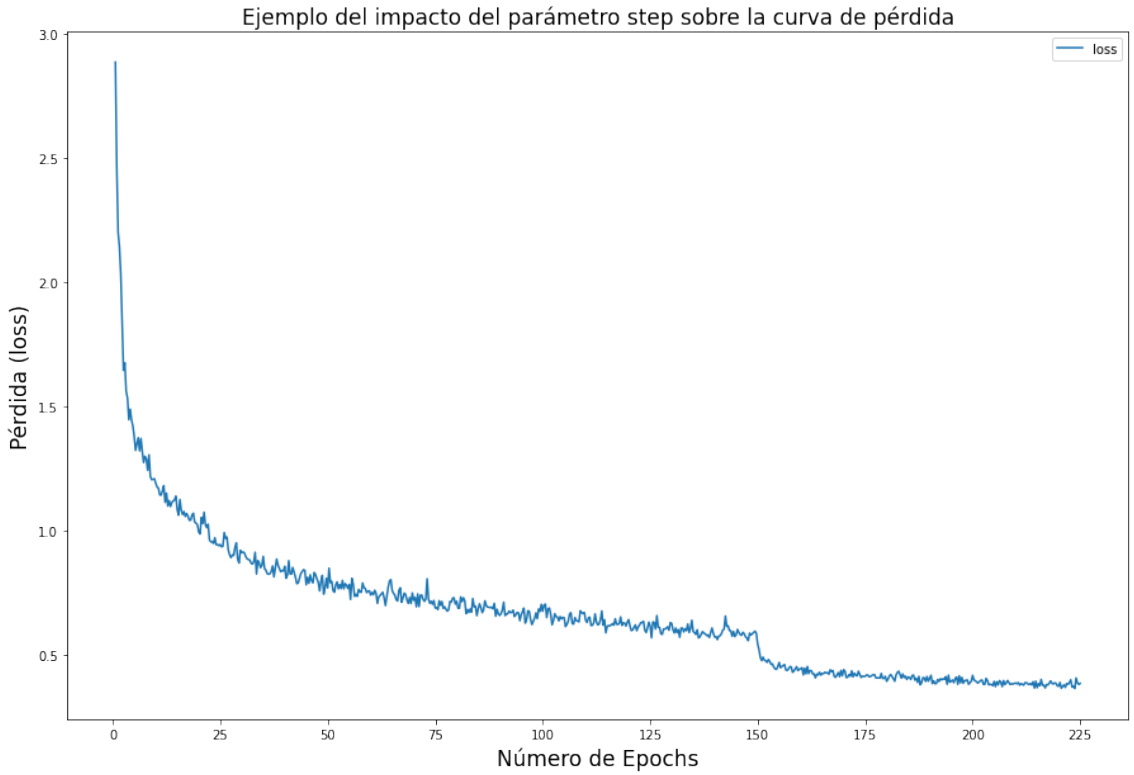

Figura 3.11: Como afecta el parámetro *step* a la curva de pérdida de la red. Concretamente sobre *mask R-CNN* con un *backbone Resnet 50* y un *batch size* de 2 imágenes.

En este apartado, juntamente con los resultados obtenidos en el apartado anterior, utilizaremos diferentes valores para el *step* con la intención de encontrar qué configuración es la más adecuada para nuestro caso de estudio. Concretamente, se ha experimentado con los siguientes valores de *step*:

- **Modificación 1**: Primer cambio al 40 % de las *epochs* o iteraciones y segundo cambio al 70 %.
- *Modificación 2*: Primer cambio al 75 % de las *epochs* o iteraciones y segundo cambio al 90 %.

Si aplicamos estas modificaciones sobre los resultados del apartado anterior obtenemos:

- Para 450 *epochs* con una configuración inicial de 300 y 400 *epochs*.
  - **Modificación 1**: 180 y 315 *epochs*.
  - Modificación 2: 335.5 y 405 *epochs* .
- Para 225 *epochs* con una configuración inicial de 150 y 200 *epochs*.
  - **Modificación 1**: 90 y 157.5 *epochs*.
  - Modificación 2: 168.75 y 202.5 *epochs* .

Una vez se han aplicado las modificaciones, podemos apreciar que, en la gráfica 3.12, los resultados son prácticamente idénticos tanto para la configuración inicial como para la segunda modificación. Estos resultados nos permiten considerar de gran interés modificar el *learning rate* en la parte final del entrenamiento.

Por otro lado, también podemos observar que el segundo *step* realizado no tiene ningún impacto sobre la curva de pérdida, por lo que bastaría con aplicar únicamente el primer *step*. En las tablas 3.8 y 3.9 podemos apreciar los resultados obtenidos y, de este modo, comparar las modificaciones y los cambios producidos en la precisión.

Tabla 3.8: Comparación resultados experimento base y las dos modificaciones en los *steps* con 225 *epochs*. Resultados para la tarea de *Bounding Box*.

Configuración	AP
Inicial	**69.39**
Modificación 1	68.66
Modificación 2	69.06

Tabla 3.9: Comparación resultados experimento base y las modificaciones en los *steps* con 225 *epochs*. Resultados de l atarea de segmentación

Configuración	AP
Inicial	**68.31**
Modificación 1	67.88
Modificación 2	68.09

Como podemos observar en la gráfica 3.13, los experimentos con 450 *epochs* se comportan del mismo modo que los experimentos con 225 *epochs*. Esto nos confirma que, en este caso en concreto, la mejor elección es actualizar el *learning*

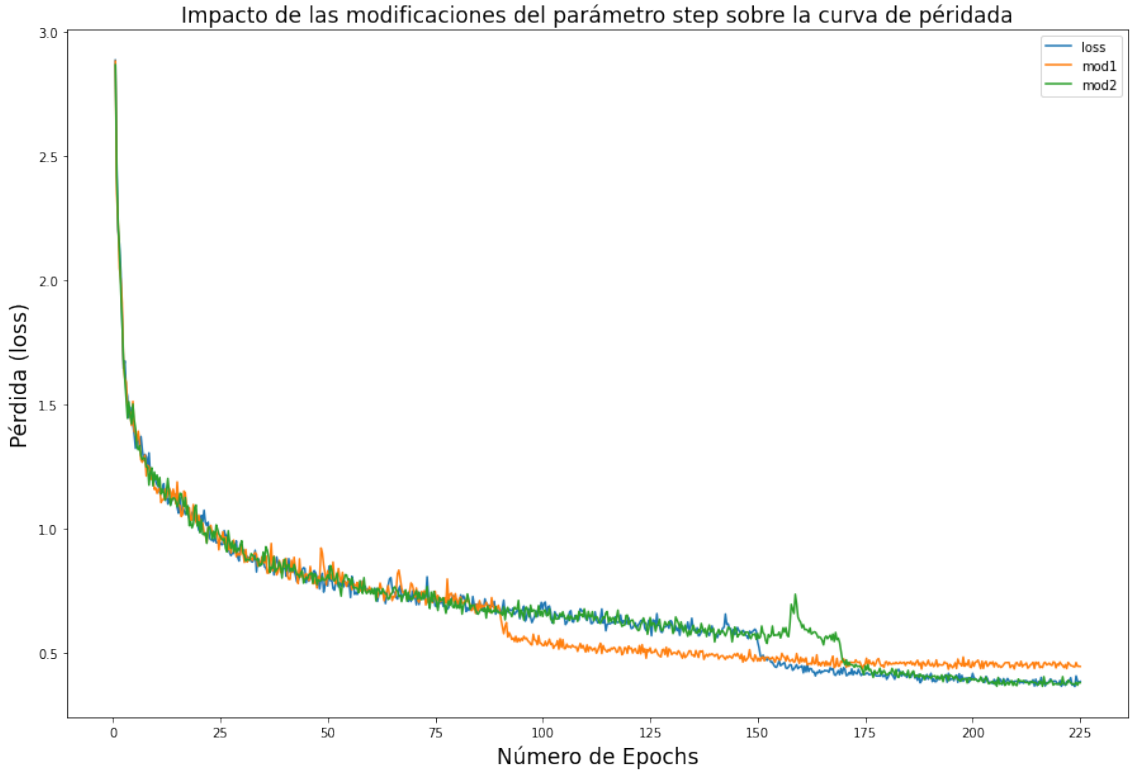

Figura 3.12: Comparación de las modificaciones realizadas sobre el parámetro *steps* de *detectron2* para 225 *epochs*.

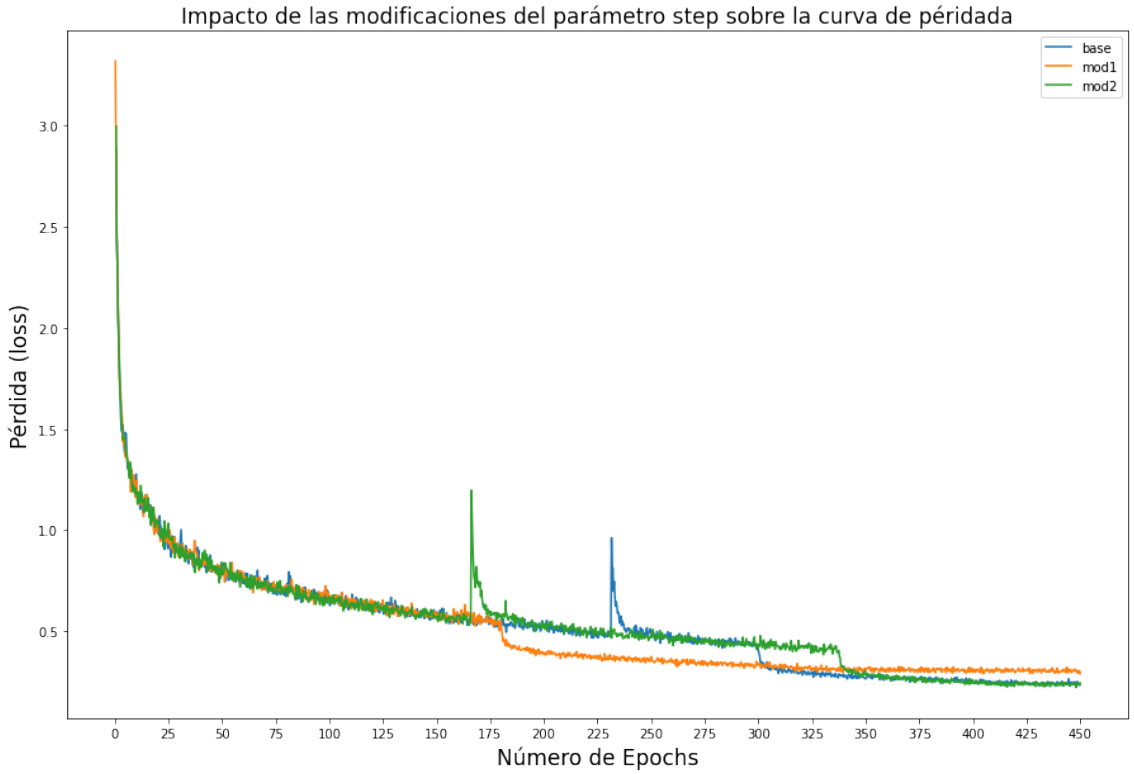

Figura 3.13: Comparación de las modificaciones realizadas sobre el parámetro *steps* de *detectron2* para 450 *epochs*.

rate en el tramo final del entrenamiento. Los resultados obtenidos con 450 *epochs* pueden observarse en las tablas 3.10 y 3.11.

Los resultados obtenidos con la configuración base, así como con la segunda modificación, son prácticamente idénticos, por lo que para los próximos experimentos se ha optado por utilizar la configuración inicial.

Tabla 3.10: Comparación resultados experimento base y las dos modificaciones en los *steps* con 450 *epochs*. Resultados para la tare de *Bounding Box*.

Configuración	AP
Inicial	**69.47**
Modificación 1	68.78
Modificación 2	69.35

Tabla 3.11: Comparación resultados experimento base y las dos modificaciones en los *steps* con 450 *epochs*. Resultados para la tarea de Segmentación.

Configuración	AP
Inicial	67.96
Modificación 1	67.65
Modificación 2	**68.03**

3.4.4. Impacto del tamaño del *batch*

El tamaño de *batch* es uno de los parámetros por excelencia en el entrenamiento de redes neuronales. A nivel de conceptos, se sugiere que, a mayor tamaño de *batch*, mejor será el cálculo del error de predicción. Esto supone que los pesos se modificarán de manera más precisa. Sin embargo, esto no siempre ocurre debido a la complejidad de las características que presenta cada *dataset*.

Esta complejidad supone un gran reto. Por un lado, un tamaño de *batch* alto puede producir que la variación de las modificaciones en los pesos sea demasiado ligera. Por otro, puede conllevar dificultades a la hora de procesar las imágenes del *batch* debido al elevado número de cálculos necesarios, lo que provoca a su vez un problema de insuficiencia de memoria. En este apartado, llevaremos a cabo un estudio de cómo el tamaño de *batch* afecta en el *corpus* de *VORAU-253*.

En este apartado estudiaremos, por una parte, cómo afecta el tamaño del *batch* al consumo de memoria GPU. Por otra parte, observaremos cómo este tamaño incide en los tiempos de ejecución y en la precisión de los resultados obtenidos.

Para los siguientes experimentos se ha utilizado 450 y 225 *epochs*, *Resnet 50* como *backbone* y los *steps* de la configuración inicial. Además, contrastaremos los resultados con la configuración anterior modificando el *backbone* utilizando la topología *Vovnet 39*.

Como podemos ver en la figura 3.14,para las ejecuciones con *Resnet 50* como *backbone*, al ejecutar con diferente tamaño de *batch*, la curva de pérdida del entrenamiento es prácticamente idéntica para los diferentes tamaños utilizados. Como podemos ver en las tablas 3.12 y 3.13, se obtiene mayor precisión con el tamaño de *batch* de dos imágenes, tanto en la tarea de *bounding box* como en la de segmentación.

Por último, podemos ver en la tabla 3.13 que, con un tamaño de *batch* de dos imágenes, se obtienen los mejores resultados y el uso de memoria es significativamente menor. Aunque si bien el tiempo más largo es el de esta ejecución, observamos una variación de 11 y 13 minutos con respecto a los otros tamaños de *batch*, diferencias de tiempos intrascendentes teniendo en cuenta las ventajas obtenidas.

Tabla 3.12: Comparación resultados de ejecución con diferentes tamaños de *batch* para 450 *epochs* con *Resnet 50* como *backbone*.Resultados para la tarea de *Bounding Box*.

Batch Size	AP
2	**69.30**
4	69.15
6	69.16

Con el fin de ver cómo puede afectar el tamaño de *batch* en estos experimentos, se ha monitoreado el uso de las GPUs mediante *scripts* en *bash*. Se guardó el porcentaje de uso de cada GPU a cada segundo de ejecución para poder comparar posteriormente el uso medio de la GPU en cada ejecución.

Con *Resnet 50* como *backbone*, para un *batch* de dos imágenes, el uso de las GPUs se mantiene constante la mayor parte de la ejecución del entrenamiento,

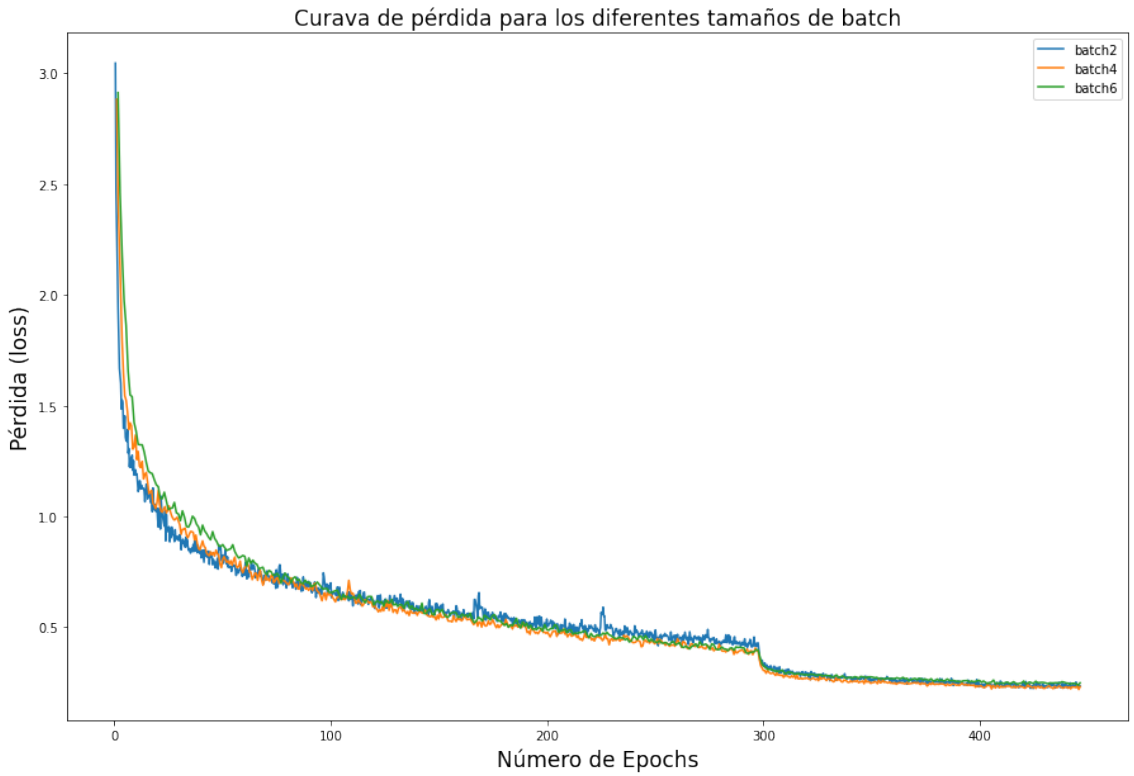

Figura 3.14: Comparación de las curvas de pérdida de *Resnet50* con 450 *epochs* de entrenamiento con tamaño de *batch* de 2,4 y 6 imágenes.

Tabla 3.13: Comparación resultados de ejecución con diferentes tamaños de *batch* para 450 epochs con *Resnet 50* como *backbone*. Resultados para la tarea de Segmentación.

Batch Size	AP
2	**68.02**
4	67.83
6	67.51

Tabla 3.14: Comparación resultados de ejecución con diferentes tamaños de *batch* en cuanto a uso de memoria y tiempos de ejecución.

Batch Size	*Training Time*	Memoria GPU1	Memoria GPU2
2	1h 55min	**3319MiB**	**3299MiB**
4	1h 44min	4835MiB	4541MiB
6	1h 42min	7845MiB	7835MiB

oscilando entre valores del 85 % y del 95 %. Hay momentos que estos sufren desfases cortos, puesto que la GPU espera recibir las imágenes procesadas de la CPU, lo que conlleva bajadas en el porcentaje de uso de la GPU. Para el caso de *batch* de cuatro imágenes, tenemos reducciones de hasta un 30 % de uso de memoria.

Al aumentar el tamaño del *batch* a cuatro imágenes, observamos una clara diferencia. Con este tamaño de *batch*, el porcentaje de uso oscila entre un 70 % y un 90 %, por lo que podemos apreciar que las GPUs tienen ahora un menor uso. Con respecto a los desfases, pueden llegar hasta valores de un 10 % de uso, siendo más frecuentes que con el *batch* de dos imágenes. Esta tendencia se confirma al

utilizar un *batch* de seis imágenes que repite los patrones observado con el *batch* de cuatro imágenes.

Este comportamiento puede deberse a que en nuestro caso la CPU carga las imágenes y, una vez estas están cargadas, las pasa a la GPU. Como la GPU es más rápida que la CPU (posee un mayor número de *workes* asignados) si aumentamos el tamaño del *batch*, la CPU tiene una mayor carga de trabajo. Esto produce que la GPU espera recibir información para continuar procesando, produciendo desajustes de tiempo entre CPU y GPU. La CPU tarda más en procesar las imágenes que la GPU en realizar el *forward* y el *backward propagation*, haciendo la CPU un cuello de botella al aumentar el tamaño de *batch*.

Debido a que las gráficas generadas son muy extensas en tamaño, no se pueden apreciar si las incluimos en este apartado de la memoria. Para una mejor visualización, se adjuntan estas gráficas en los anexos de la memoria.

Podemos ver en las tablas 3.15 y 3.16 que los resultados son a nivel general similares, aunque de menor precisión, que confirman la tendencia obtenida con 450 *epochs*. Con respecto a los tiempos de entrenamiento y el uso de memoria podemos ver en la tabla 3.17 que los tiempos de ejecución se reducen en la mitad, mientras que el uso de memoria GPU es prácticamente el mismo.

Tabla 3.15: Comparación resultados de ejecución con diferentes tamaños de batch para 225 epochs con *Resnet 50* como *backbone*. *Bounding Box*.

Batch size	AP
2	69.39
4	68.57
6	68.77

Tabla 3.16: Comparación resultados de ejecución con diferentes tamaños de *batch* para 225 *epochs* con *Resnet 50* como *backbone*. Segmentación

Batch size	AP
2	68.31
4	67.45
6	67.86

Pasando a la ejecución de estos experimentos con *Vovnet 39* como *backbone* para ver como puede afectar el tipo de arquitectura, podemos observar claramente en la gráfica 3.15 que a menor tamaño de *batch* mejor converge para nuestro caso de estudio. Viendo los resultados en cuanto a precisión y tiempos de ejecución en las tablas 3.18 , 3.19 y 3.20 podemos apreciar que, la precisión alcanzada con un *batch* de 2 imágenes es superior, mejorando los resultados de *Resnet 50*.

Tabla 3.17: Comparación resultados de ejecución con diferentes tamaños de *batch* en cuanto a uso de memoria y tiempos de ejecución.

Batch size	*Training Time*	Mem. GPU1	Mem. GPU2
2	57min	**3025MiB**	**3285MiB**
4	50min	4885MiB	5071MiB
6	50min	7443MiB	6463MiB

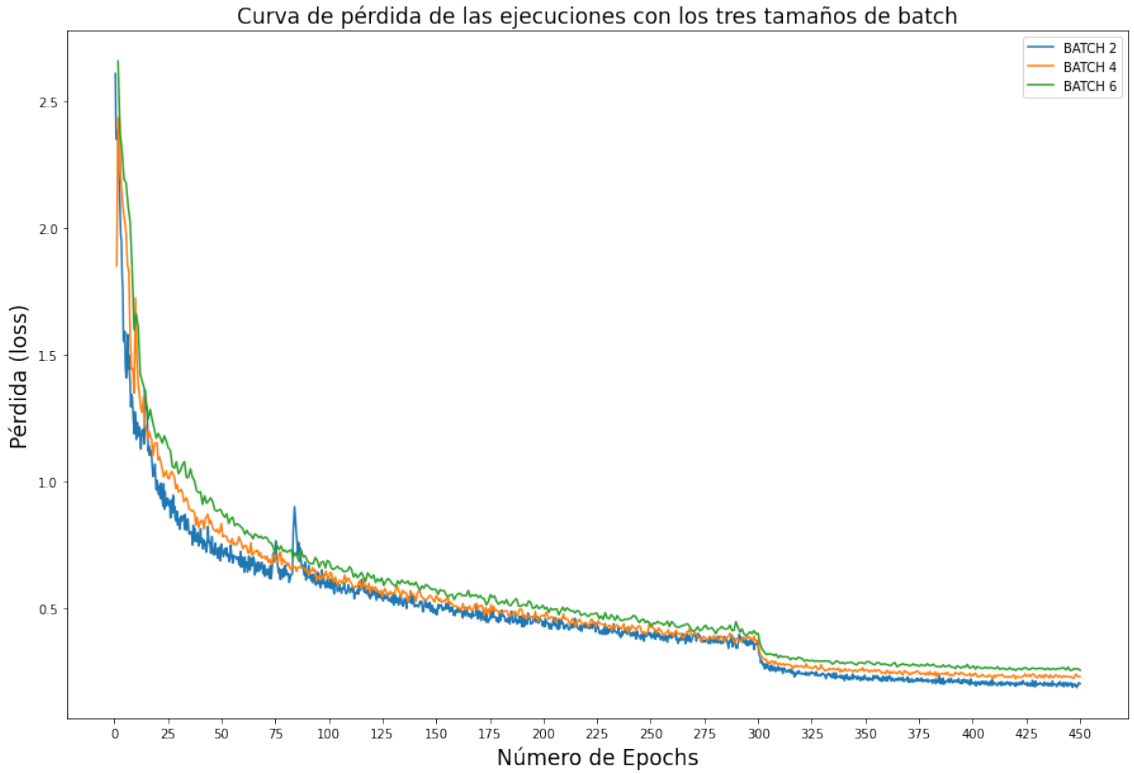

Figura 3.15: Comparación de las curvas de pérdida de *Vovnet 39* con 450 *epochs* de entrenamiento con tamaño de *batch* de 2, 4 y 6 imágenes.

Tabla 3.18: Resultados de la ejecución con los diferentes tamaños de *batch* para 450 *epochs* con *Vovnet* como *backbone*. Resultados para la tarea de *Bounding box*.

Batch size	AP
2	70.23
4	69.29
6	68.85

Tabla 3.19: Resultados de la ejecución con los diferentes tamaños de *batch* para 450 *epochs* con *Vovnet* como *backbone*.Resultados para la tarea de Segmentación.

Batch size	AP
2	68.72
4	67.93
6	67.76

Tabla 3.20: Resultados de la ejecución con los diferentes tamaños de *batch* en cuanto a uso de memoria y tiempos de ejecución.

Batch size	*Training Time*	Mem. GPU1	Mem. GPU2
2	2h 10min	4087MiB	4027MiB
4	1h 58min	6683MiB	6585MiB
6	1h 57min	7791MiB	7897MiB

3.4.5. Uso de las diferentes Arquitecturas de CNNs con la configuración de parámetros obtenida

Una vez realizado el estudio de cómo afectan los parámetros sobre el *corpus VORAU-253*, aplicaremos los resultados obtenidos sobre las topologías presentadas en el apartado Modelos Propuestos.

En esta sección, pretendemos observar cómo afectan los parámetros que hemos estudiado en los apartados anteriores con respecto al uso de memoria GPU, a la precisión y a los tiempos de ejecución

Los parámetros obtenidos serán utilizados como configuración para las topologías propuestas en la sección Modelos Propuestos con el propósito de examinar su rendimiento y de seleccionar aquellas con las prestaciones más adecuadas para el objetivo de esta memoria.

Tabla 3.21: Comparación resultados de ejecución con diferentes *Resnets* como *backbones* para *Mask R-CNN*. Métricas de calidad

Backbone	*Task*	AP	AP50	AP75	APm	APl
Resnet 31	*Bounding Box*	68.84	97.51	80.62	40.56	69.06
	Segmentation	67.42	96.35	78.33	33.13	67.71
Resnet 50	*Bounding Box*	69.34	97.55	81.89	40.51	69.56
	Segmentation	68.02	96.55	78.93	35.14	68.28
Resnet 50b	*Bounding Box*	68.12	94.47	80.01	40.13	68.78
	Segmentation	67.20	96.09	77.98	35.21	67.23
Resnet 50c	*Bounding Box*	67.94	96.94	79.74	42.00	68.28
	Segmentation	66.49	95.96	77.19	35.28	66.86
Resnet 101	*Bounding Box*	69.81	97.29	81.96	42.31	70.06
	Segmentation	68.39	96.66	79.63	37.59	68.71

Tabla 3.22: Comparación resultados de ejecución con diferentes *Resnets* como *backbones* para *Mask R-CNN*. Tiempos de entrenamiento y uso de memoria GPU.

Backbone	*Training Time*	Mem GPU1	Mem GPU2
Resnet 31	1h 45min	2795MiB	3089MiB
Resnet 50	1h 54min	3247MiB	3363MiB
Resnet 50b	1h 54min	2998MiB	3354MiB
Resnet 50c	1h 56min	2791MiB	3583MiB
Resnet 101	2h 26min	3965MiB	3837MiB

Si observamos los resultados en cuanto a nivel de uso de memoria GPU,*Resnet 50* es la topología que mejor prestaciones nos ofrece, teniendo en cuenta la precisión y el tiempo de ejecución. *Resnet 50* no es la topología con el menor tiempo de ejecución ni con la precisión más elevada, pero sí la que mejor balance tiene entre todas las métricas utilizadas.

Finalmente, por lo que respecta a las topologías de *Vovnet*, *Vovnet 39* es la que mejor desempeño nos ofrece. Pese a ser una topología no muy profunda en comparación con *Vovnet 57 o 99*, es la que mejores valores de AP ha dado. Si bien el uso de memoria es algo más elevado que *Vovnet 19 Lite*, los tiempos de ejecución

Tabla 3.23: Comparación resultados de ejecución con diferentes *Vovnets* como *backbones* para *Mask R-CNN*. Métricas de calidad.

Backbone	Task	AP	AP50	AP75	APm	APl
Vovnet 19L	*Bounding Box*	68.12	97.27	80.27	37.95	68.33
	Segmentation	66.89	96.41	77.96	36.50	67.33
Vovnet 19	*Bounding Box*	68.29	96.92	79.76	35.37	68.53
	Segmentation	66.99	96.11	77.61	30.22	67.29
Vovnet 39	*Bounding Box*	70.00	97.55	82.08	38.18	70.30
	Segmentation	68.75	96.29	79.84	32.05	69.14
Vovnet 57	*Bounding Box*	69.95	97.58	82.43	40.13	70.15
	Segmentation	68.62	95.98	80.53	32.82	68.95
Vovnet 99	*Bounding Box*	70.29	97.63	82.88	42.76	70.59
	Segmentation	68.91	96.06	80.34	36.85	69.23

Tabla 3.24: Comparación resultados de ejecución con diferentes *Vovnets* como *backbones* para *Mask R-CNN*.

Backbone	*Training Time*	Mem GPU1	Mem GPU2
Vovnet 19L	1h 37min	3461MiB	3285MiB
Vovnet 19	1h 50min	4061MiB	3669MiB
Vovnet 39	2h 10min	4087MiB	4087MiB
Vovnet 57	2h 20min	4065MiB	4474MiB
Vovnet 99	3h 18min	5551MiB	5205MiB

difieren por muy pocos minutos, haciendo de *Vovnet 39* la mejor topología para nuestro estudio.

3.4.6. Impacto en el consumo y los costes

En el apartado anterior, hemos analizado el desempeño que han realizado todas las arquitecturas empleadas. Bajo el estudio de optimización de los parámetros y la correcta elección de la arquitectura empleada, podemos disminuir el consumo de memoria y reducir los tiempos de ejecución sin perder prestaciones en cuanto a precisión.

A continuación, compararemos los resultados del caso base con los resultados obtenidos con las nuevas configuraciones para incidir en cómo enriquecen a la arquitectura. Gracias a estos resultados, nace la posibilidad de añadir nuevas funcionalidades que puedan derivar en una mejora del rendimiento de la arquitectura *Mask R-CNN*.

Los resultados obtenidos inciden en una reducción del consumo energético y del consumo económico, disminuyendo de este modo el impacto sobre el medio ambiente.

Con los resultados obtenidos en el apartado anterior, podemos observar que la mejor topología es *Resnet 50* con respecto a las arquitecturas de *Resnet*.

Vovnet 39 es la topología que mejor se adapta en cuanto a las arquitecturas de *Vovnet* propuestas.

Seguidamente, compararemos los resultados del caso base con los resultados de estas dos arquitecturas con la nueva configuración de parámetros con la finalidad de observar los beneficios de este estudio.

Tabla 3.25: Comparación de los tiempos de entrenamiento junto a el uso de memoria GPU del Caso Base con las dos mejores topologías.

Backbone	*Training Time*	Mem GPU1	Mem GPU2
Caso Base	10h 48min	4941MiB	4901MiB
Resnet 50	**1h 54min**	**3247MiB**	**3363MiB**
Vovnet 39	2h 10min	4087MiB	4087MiB

En cuanto a tiempos de ejecución,como podemos ver en la tabla 3.25, hemos conseguido bajar de 10 horas y 48 minutos a 1 hora 54 minutos para la topología *Resnet50* y 2 horas y 10 minutos para *Vovnet 39*. Esto implica un ahorro de prácticamente 9 horas para la nueva configuración de parámetros con *Resnet 50* mientras que para *Vovnet* es de 8 horas y 38 minutos. Son mejoras significativas en cuanto a tiempos de entrenamiento.

Respecto a el uso de memoria GPU, *Resnet 50* con la nueva configuración de parámetros consigue bajar 1,7GiB la memoria empleada, mientras que *Vovnet* baja 0,9GiB la memoria GPU requerida.

Esta reducción de tiempos de entrenamiento y memoria nos llevan a una reducción en cuanto el consumo energético requerido. Mediante el uso de *scripts* en *bash* se ha monitorizado la potencia a cada segundo que utiliza cada GPU, calculado la potencia media requería para el entrenamiento y así poder calcular el coste energético y por ende económico.

Para el caso base se ha calculado que cada GPU aplica una potencia media de 177,3W (0.177kW). Como tenemos 2 GPUs durante 10 horas y 48 minutos esta ejecución desarrolla un trabajo de:

2 * 0.177kW * 10,8(10h 48mins) = 3.823kWh

Para *Resnet 50* tenemos que para cada GPU se aplica una potencia media de 180W (0.18kW). Se tiene 2 GPUs durante 1 horas y 54 minutos desarrollan un trabajo de:

2 * 0.18kW * 1,9(1h 54mins) = 0.684KWh

En cuanto a *Vovnet 39*, tenemos que cada GPU aplica una potencia media de 186.5W (0.1865kW). Al tener 2 GPUs durante 2 horas y 10 minutos se desarrolla un trabajo de:

2 * 0.1865kW * 2.16 (2h 10mins) = 0.808kWh

Como podemos observar en la tabla 3.26 hemos conseguido un ahorro en cuanto a consumo energético, y por ende económico superior al 80 % para la topología *Resnet 50* con la nueva configuración de parámetros, mientras que para *Vovnet 39* tenemos un ahorro de un 78.8 %.

Tabla 3.26: Porcentaje de ahorro en cuanto a consumo energético.

Backbone	Porcentaje Ahorro
Resnet 50	82.1 %
Vovnet 39	78.8 %

Para nuestro caso de estudio, si asumimos que el coste de la electricidad es de 0.246 Euros/kWh, podemos calcular el ahorro económico por ejecución.

Con el trabajo calculado, al multiplicarlo por el coste por kWh obtenemos el coste total del entrenamiento del caso base:

0.246Euros/KWh * 3.823KWh = 0.94 Euros, es decir, 94 céntimos.

Por tanto, para *Resnet 50* tendemos un consumo económico por ejecución de:

94 cént. - (94 cnént. * 0.821) = 16.82 céntimos/ejecución.

Mientras que, para *Vovnet 39* contamos con un consumo de:

94 cént. - (94 cént. * 0.788) = 19.8 céntimos/ejecución.

Para la arquitectura *Mask R-CNN* y el *dataset* utilizados en esta, el ahorro de unos céntimos puede parecer poco a primera vista, pero si escalamos este tipo de estudios a modelos gigantescos de grades compañías, el ahorro puede llegar a ser de cientos o miles de dolares viendo los porcentajes de ahorro alcanzados.

3.5 Observaciones y valoración de los resultados

A lo largo de los experimentos, se han utilizado otros valores de *learning rate* de 0.002 y 0.2, aunque los resultados obtenidos no son relevantes.

Si centramos la mirada en la precisión, podemos observar en la tabla 3.27 *Vovnet 39* que se consigue mejorar la precisión en prácticamente un 1,5 % para la tarea del *bounding box* y un 1,4 % para la tarea de segmentación.

Esto indica que, con una arquitectura menos profunda como *Vovnet 39*, se pueden obtener resultados iguales o superiores en cuanto a precisión que con arquitecturas más profundas como *Resnet 50* o *Resnet 101*. Asimismo, podemos deducir que la arquitectura del caso base tiene un sobreentrenamiento, de ahí la importancia de ajustar el número de *epochs*.

Por otro lado, en la tabla 3.28 se observa como con *Resnet50* como *backbone* conseguimos el tiempo de ejecución más bajo, así como el coste económico por ejecución más reducido. Con este *backbone* hemos obtenido un ahorro de 77.2 céntimos por ejecución.

Para finalizar con este apartado, hemos conseguido bajar el uso de la GPU de unos 4,9 GiB a 3,3 GiB, ajustando el tamaño de *batch* para la misma topología de *Resnet 50*. Esto nos ofrece 1,6 GiB más de memoria GPU para poder utilizar en otras tareas dentro de la arquitectura *Mask R-CNN*. Podemos ver un ejemplo de los resultados obtenidos por *Mask R-CNN* para la tarea de segmentación en la imagen 3.16.

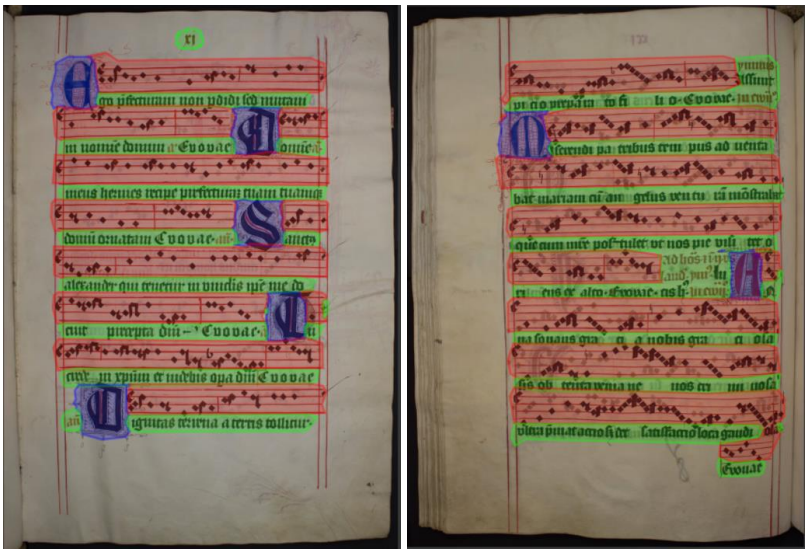

Figura 3.16: Ejemplo resultados segmentación mediante Mask R-CNN. En azul tenemos el *drop-capital*. En rojo se muestra los *staff* y en verde los *lyrics*.

Tabla 3.27: Resultados de las tareas de Segmentación y *bounding box*. Comparación del caso base con *Vovnet 39* y *Resnet 50* con la mejor selección de parámetros.

Backbone	*Task*	AP	AP50	AP75	APm	APl
Caso Base	*Bounding Box*	68.66	96.47	80.48	**44.88**	68.65
	Segmentation	67.38	95.86	78.25	**38.71**	67.68
Resnet 50	*Bounding Box*	69.34	97.55	81.89	40.51	69.56
	Segmentation	68.02	**96.55**	78.93	35.14	68.28
Vovnet 39	*Bounding Box*	**70.00**	**97.55**	**82.08**	38.18	**70.30**
	Segmentation	**68.75**	96.29	**79.84**	32.05	**69.14**

Tabla 3.28: Resumen de los tiempos de entrenamiento así como los costes económicos asociados al total de la ejecución.

Backbone	*Training Time*	Coste económico asociado
Caso Base	10h 48min	94 céntimos
Resnet 50	**1h 54min**	**16.8** céntimos
Vovnet 39	2h 10min	19.8 céntimos

CAPÍTULO 4
Conclusiones

En este capítulo presentaremos las distintas conclusiones a las que hemos llegado.

En primer lugar, cabe recordar que este trabajo tenía, inicialmente, dos objetivos. El primero de ellos era realizar un estudio de cómo afectaban los parámetros de las redes neuronales en el rendimiento de estas con la intención de obtener la mejor configuración de parámetros. El segundo consistía en utilizar diversos modelos de redes neuronales convolucionales como *backbone* para *Másk R-CNN* con la finalidad de mejorar el rendimiento de dicha arquitectura en las tareas de segmentación y detección de objetos.

Para alcanzar el primer objetivo, hemos modificado la biblioteca *detectron2* con el propósito de realizar el estudio del impacto de los parámetros. Asimismo, a lo largo del trabajo, hemos ido cambiando estos parámetros y hemos ido probando las diferentes configuraciones hasta encontrar la óptima para nuestro *dataset VORAU*.

Para conseguir el segundo objetivo, hemos realizado un estudio a nivel teórico acerca de qué topologías de CNNs podrían funcionar mejor como *backbone* para las tareas de *DLA*. Finalmente, se llegó a la conclusión de que las topologías de *Vovnet* junto con las de *Resnet* ya empleadas eran las más indicadas para este trabajo.

Una vez seleccionadas las topologías, hemos llevado a cabo los experimentos con la nueva configuración de parámetros. Además, hemos utilizado todas las topologías presentadas a lo largo de la memoria para ver cuál de ellas se adaptaba mejor a nuestras necesidades.

Respecto a los resultados obtenidos, se ha mejorado de manera significativa el rendimiento, no solo alcanzando el objetivo de reducir el consumo de memoria GPU y los tiempos de entrenamiento, sino también llegando a mejorar la precisión en cuanto a AP en algunos casos.

4.1 Trabajos Futuros

Como línea de investigación futura, se proponen las siguientes opciones:

- Explorar nuevas topologías, así como la creación de nuevas CNNs con el fin de aunar las características que más se adaptan a nuestro propósito.
- Realizar pruebas más robustas con el uso de *corpus* más complejos, como documentos con tablas o mayor cantidad de objetos y clases a detectar. Esto requiere más tiempo de cómputo.
- Buscar nuevas funcionalidades para la arquitectura *Mask R-CNN* para seguir profundizando en el campo del DLA.
- Usar arquitecturas alternativas de detección y segmentación de objetos como YOLO.
- Experimentar con otro tipo de redes neuronales como las recurrentes o los mecanismos de atención (*transformers*).

Bibliografía

[1] R. Reeve Ingle, Yasuhisa Fujii, Thomas Deselaers, Jonathan Baccash y Ashok C. Popat. "A Scalable Handwritten Text Recognition System". En: *CoRR* abs/1904.09150 (2019). arXiv: 1904.09150. URL: http://arxiv.org/abs/1904.09150.

[2] Paul Thompson. "Looking back: On relevance, probabilistic indexing and information retrieval". En: *Information Processing Management* 44.2 (2008), págs. 963-970. ISSN: 0306-4573. DOI: https://doi.org/10.1016/j.ipm.2007.10.002. URL: https://www.sciencedirect.com/science/article/pii/S0306457307002087.

[3] Tianlong Ma, Xingjiao Wu, Xin Li, Xiangcheng Du, Zhao Zhou, Liang Xue y Cheng Jin. "Document Layout Analysis with Aesthetic-Guided Image Augmentation". En: *CoRR* abs/2111.13809 (2021). arXiv: 2111.13809. URL: https://arxiv.org/abs/2111.13809.

[4] Lorenzo Quirós. "Layout Analysis for Handwritten Documents. A Probabilistic Machine Learning Approach". En: (2022). URL: http://hdl.handle.net/10251/181483.

[5] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever y Dario Amodei. "Language Models are Few-Shot Learners". En: *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[6] Ian Goodfellow, Yoshua Bengio y Aaron Courville. "Deep Learning". En: (2016). http://www.deeplearningbook.org.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollár y Ross B. Girshick. "Mask R-CNN". En: *CoRR* abs/1703.06870 (2017). arXiv: 1703.06870. URL: http://arxiv.org/abs/1703.06870.

[8] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". En: 386–408 (1986). URL: https://doi.org/10.1038/323533a0,.

[9] Hinton G. Williams R. Rumelhart D. "Learning representations by back-propagating errors". En: (1958). URL: https://doi.org/10.1037/h0042519.

[10] Juan Carlos y Juan Carlos Cuevas-Tello. "Apuntes de Redes Neuronales Artificiales - Handouts for Artificial Neural Networks". En: (2018). DOI: 10.13140/RG.2.2.13177.57447.

[11] TIBCO. *¿Qué es una red neuronal?* URL: https://www.tibco.com/es/reference-center/what-is-a-neural-network.

[12] Sebastian Ruder. "An overview of gradient descent optimization algorithms". En: *arXiv preprint arXiv:1609.04747* (2016).

[13] K Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". En: 193–202 (1980). URL: https://doi.org/10.1007/BF00344251.

[14] Dan iresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella y Jürgen Schmidhuber. "Flexible, High Performance Convolutional Neural Networks for Image Classification." En: (2011).

[15] Daphne Cornelisse. *"An intuitive guide to Convolutional Neural Networks*. URL: https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/.

[16] Imran Ali. *Pooling layer operation*. URL: https://www.researchgate.net/figure/Pooling-layer-operation-oproaches-1-Pooling-layers-For-the-function-of-decreasing-the_fig4_340812216.

[17] Huawei. *¿Qué es la red neuronal convolucional?* URL: https://forum.huawei.com/enterprise/es/\%C2\%BFqu\%C3\%A9-es-la-red-neuronal-convolucional/thread/746075-100757.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li y Li Fei-Fei. "ImageNet: A large-scale hierarchical image database". En: (2009), págs. 248-255. DOI: 10.1109/CVPR.2009.5206848.

[19] Takudzwa Fadziso. "Overcoming the Vanishing Gradient Problem during Learning Recurrent Neural Nets (RNN)". En: *Asian Journal of Applied Science and Engineering* 9 (2020), 207–218. URL: https://www.upright.pub/index.php/ajase/article/view/36.

[20] Rupesh Kumar Srivastava, Klaus Greff y Jürgen Schmidhuber. "Highway Networks". En: *CoRR* abs/1505.00387 (2015). arXiv: 1505.00387. URL: http://arxiv.org/abs/1505.00387.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun. "Deep Residual Learning for Image Recognition". En: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[22] Youngwan Lee, Joong-won Hwang, Sangrok Lee, Yuseok Bae y Jongyoul Park. "An Energy and GPU-Computation Efficient Backbone Network for Real-Time Object Detection". En: *CoRR* abs/1904.09730 (2019). arXiv: 1904.09730. URL: http://arxiv.org/abs/1904.09730.

[23] Gao Huang, Zhuang Liu y Kilian Q. Weinberger. "Densely Connected Convolutional Networks". En: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[24] Shaoqing Ren, Kaiming He, Ross B. Girshick y Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". En: *CoRR* abs/1506.01497 (2015). arXiv: 1506.01497. URL: http://arxiv.org/abs/1506.01497.

[25] Derrick Mwitin. "The Definitive Guide of Semantinc Segmentation for Deep Learning in Python". En: (). URL: https://cnvrg.io/semantic-segmentation/.

[26] Developer PAPER. *Mask R-CNN*. URL: hhttps://developpaper.com/mask-r-cnn/.

[27] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo y Ross Girshick. *Detectron2*. https://github.com/facebookresearch/detectron2. 2019.

[28] Rui Tang, Lihui Chen, Lihui Chen, Zhibing Lai, Marcelo Keese Albertini y Xiaomin Yang. "Lightweight network with one-shot aggregation for image super-resolution". En: (2021). URL: https://doi.org/10.1007/s11554-021-01127-6.

[29] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár y C. Lawrence Zitnick. "Microsoft COCO: Common Objects in Context". En: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[30] Shuhong Cheng, Kaopeng Zhao y Dianfan Zhang. "Abnormal Water Quality Monitoring Based on Visual Sensing of Three-Dimensional Motion Behavior of Fish". En: *Symmetry* 11 (sep. de 2019), pág. 1179. DOI: 10.3390/sym11091179.

APÉNDICE A
Uso de Memoria GPU

En este apéndice mostraremos las gráficas obtenidas de los *scripts* para monitorizar el uso de memoria GPU, que no hemos podido mostrar en la memoria por temas de espacio y no sobrecargarla de imágenes. Son los correspondientes a una *Resnet 50* como *backbone*, utilizando los tres tamaños de *batch*.

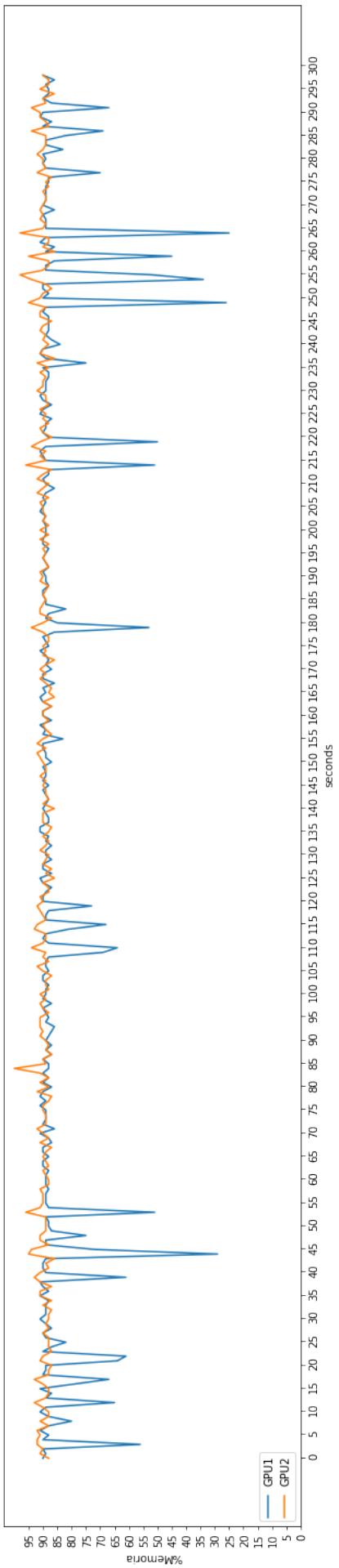

Figura A.1: Uso de memoria de las dos GPUs durante 5 minutos. Corresponde a una ejecución con un *batch* de dos imágenes, para una topología *Resnet 50* como *backbone*.

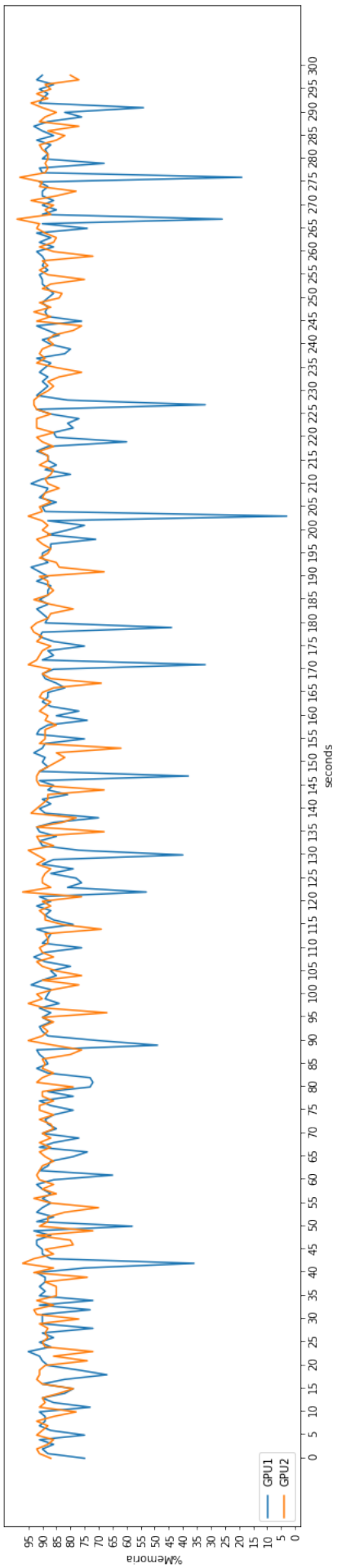

Figura A.2: Uso de memoria de las dos GPUs durante 5 minutos. Corresponde a una ejecución con un *batch* de cuatro imágenes, para una topología *Resnet 50* como *backbone*.

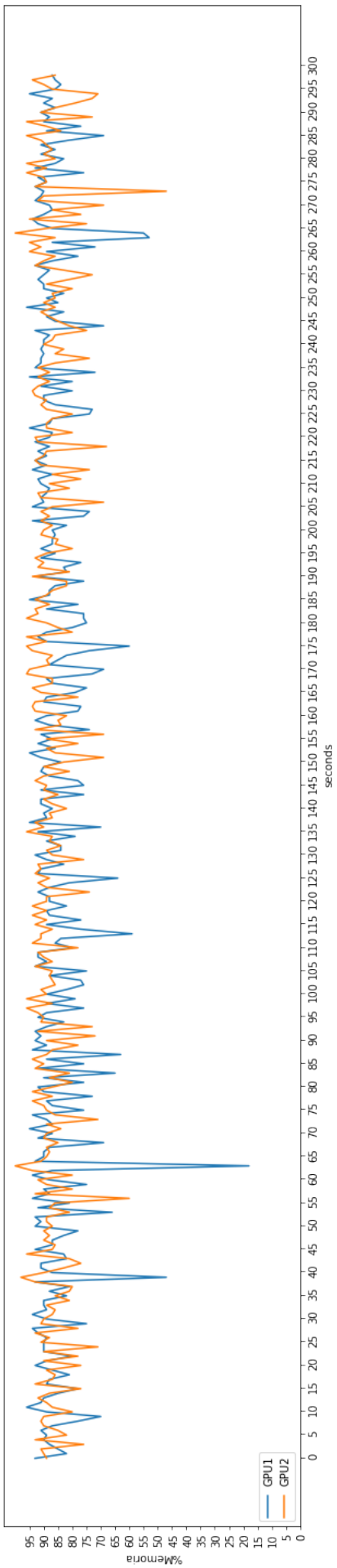

Figura A.3: Uso de memoria de las dos GPUs durante 5 minutos. Corresponde a una ejecución con un *batch* de seis imágenes, para una topología *Resnet 50* como *backbone*.

ANEXO

OBJETIVOS DE DESARROLLO SOSTENIBLE

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				x
ODS 2. **Hambre cero.**				x
ODS 3. **Salud y bienestar.**				x
ODS 4. **Educación de calidad.**				x
ODS 5. **Igualdad de género.**				x
ODS 6. **Agua limpia y saneamiento.**				x
ODS 7. **Energía asequible y no contaminante.**		x		
ODS 8. **Trabajo decente y crecimiento económico.**				x
ODS 9. **Industria, innovación e infraestructuras.**				
ODS 10. **Reducción de las desigualdades.**				x
ODS 11. **Ciudades y comunidades sostenibles.**				x
ODS 12. **Producción y consumo responsables.**		x		
ODS 13. **Acción por el clima.**		x		
ODS 14. **Vida submarina.**				x
ODS 15. **Vida de ecosistemas terrestres.**				x
ODS 16. **Paz, justicia e instituciones sólidas.**				x
ODS 17. **Alianzas para lograr objetivos.**				x

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

En este anexo vamos a hacer una reflexión sobre el impacto de este trabajo con las ODS (Objetivos de Desarrollo Sostenible).

Con respecto a la ODS 7, Energía asequible y no contaminante, este trabajo se centra en como optimizar una arquitectura de *deep learning* para minimizar el consumo. Mediante el estudio realizado, podemos ver en el apartado de , Impacto en el consumo y los costes, haciendo uso de los parámetros y las topologías adecuadas, podemos pasar de un trabajo eléctrico de 3,83kWh a 0.684kWh. Esto puede servir de referencia para futuros trabajo y sobretodo para ser conscientes de que con los estudios y la gestión adecuada podemos reducir el consumo de energía que necesitamos y que consumimos.

Esto nos abre la puerta a comentar la siguiente ODS que es la número 12, Garantizar modalidades de consumo y producción sostenible. Ser conscientes que estas prácticas tienen un impacto en el medio ambiente nos ha llevado a que, si bien es cierto que la Inteligencia Artificial nos abre un nuevo mundo de posibilidades para procesar información, no debe ser a costa del uso desmesurado de recursos. Una gestión adecuada de estos mediante el estudio, nos ha llevado a reducir tiempos de entrenamiento de las arquitecturas empleadas, y por ende, el consumo energético y económico.

Como podemos ver en el apartado del Impacto en el consumo y los costes, mediante el estudio de como afectan los diferentes parámetros así como comprender como las diferentes redes utilizadas pueden tener un impacto en los recursos utilizados, hemos conseguido bajar tiempos de entrenamiento de 10h y 48 minutos a prácticamente 2 horas, lo que supone unos tiempos 5 veces más rápidos y por consecuencia, 5 veces menos consumo, obteniendo resultados mejores. Es de gran importancia que apliquemos estas prácticas sin perder de vista el impacto que estas tienen en el medio ambiente, y en nuestras manos esta aprovechar los conocimientos aprendidos para hacer de estas tecnologías más sostenibles.

Para último, hablaremos de la ODS número 13, Adoptar medidas urgentes para combatir el cambio climático y sus efectos. Es algo obvio, que en pleno sigo XXI el cambio climático uno de los mayores problemas a los que la humanidad se enfrenta y que, si nada cambia, dentro de unos años se convertirá en un problema irreversible. Siendo conscientes de esto, esta en nuestras manos utilizar los conocimientos y técnicas disponibles para reducir todo lo que esté en nuestras manos el consumo energético. Las técnicas utilizadas en esta memoria implican un consumo eléctrico. Con el estudio realizado, hemos conseguido reducir la potencia eléctrica requerida para los entrenamientos de la arquitectura *Mask R-CNN* para obtener los mismos resultados, o mejores, como hemos visto anteriormente de 3,83kWh a 0.684kWh, es decir, 5,6 veces menos consumo, obteniendo los mismos resultados.