E.T.S. de Ingeniería Industrial, Informática
y de Telecomunicación

Uso de redes convolucionales en la detección de montañas

Grado en Ingeniería Informática

Trabajo Fin de Grado

Alumno: Miguel Azcárate Aragón

Director: Miguel Pagola Barrio

Pamplona, 02/05/2021

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

1 Resumen

La detección de imágenes (Image Retrieval) es un tema que está muy de moda y tiene múltiples utilidades muy interesantes: desde registros de imágenes médicas hasta aplicaciones policiales, militares o arquitectónicas. Esta técnica consiste básicamente en obtener de una imagen de entrada las fotografías más parecidas a la misma, valorando las características propias del elemento introducido.

El principal método y la base de la detección de imágenes son las Redes Neuronales Convolucionales (CNNs). Mediante el uso de CNNs se extraen características de un grupo de fotografías específicas y se entrena una red, que se convertirá en "experta" detectando imágenes parecidas. Además del uso de redes convolucionales clásicas, se van a utilizar dos métodos más específicos, R-MAC y Regional Attention Network, que sirven para ajustar y mejorar la precisión de la red y que se explicarán más adelante.

El objetivo principal es, proporcionada una montaña al sistema, predecir cuál es su altura utilizando los procesos que se han mencionado con anterioridad. La idea es conseguir una red que se acerque lo máximo posible a la altura real del monte sin más información que la figura en la fotografía introducida.

Palabras clave: Red neuronal convolucional, Image retrieval, R-MAC, Regional Attention Network, Visión artificial.

ÍNDICE

1 Resumen 2
2 Introducción 5
3 Justificación y objetivos 6
4 Contexto tecnológico 8
4.1 Introducción: Machine Learning 8
4.2 Red neuronal 9
4.2.3 Biología 9
4.2.4 Perceptrón 10
4.2.5 Red neuronal 11
4.2.6 Aprendizaje 16
4.2.7 Backpropagation 17
4.3 Red neuronal convolucional 20
4.3.1 Introducción 20
4.3.2 Características de la red neuronal convoucional 20
4.3.3 Funcionamiento 24
4.4 R-MAC y Regional Attention Network 26
4.4.1 Introducción 26
4.4.2 R-MAC 26
4.4.3 Context-Aware Regional Attention 29
4.4.4 Context Awareness 30
5 Experimentos realizados 35
5.1 Entrenamiento del módulo Regional Attention Network 35
5.1.1 Introducción y dataset 35
5.1.2 Desarrollo 35
5.1.3 Resultados 36
5.2 Predicción de la altura y resultados finales 39
5.2.1 Introducción y dataset 39
5.2.2 Experimentos con las imágenes de la montaña 41
5.2.3 Experimentos sin las imágenes de la montaña 44
5.2.4 Queries complicadas 47
5.2.5 Conclusiones 50

2 Introducción

Hoy en día, el uso de imágenes en el contexto tecnológico y de internet está cada vez más en auge. Se pueden destacar numerosos ejemplos: métodos de detección de enfermedades por medio de imágenes, técnicas de reconocimiento facial para permitir la entrada solamente con la imagen de la cara o procesos para detectar, por ejemplo, firmas falsificadas. El campo sobre el que se hablará en este trabajo se conoce como *image retrieval*, o recuperación de imágenes. Como ya se ha comentado, consiste en, dada una imagen, buscar la fotografía que más características parecidas tenga con la entrada. Por ejemplo, se ha desarrollado en este proyecto un modelo que tiene dicho funcionamiento con figuras de montañas. Por lo tanto, al introducir una imagen de un monte, deberá devolver las fotos más parecidas a la entrada. Con ello, se podrá calcular más adelante la altura, que es lo que se busca como objetivo final.

Sin embargo, esto no se queda solo aquí. Dentro del campo de la visión artificial existen nuevos algoritmos que consiguen generar fotografías de la nada. Por ejemplo, como se puede ver en la *Figura 2.*1, se logra dado un boceto, imágenes completas de carreteras, edificios o bolsos, o convertir una imagen normal de día a una de noche [1].

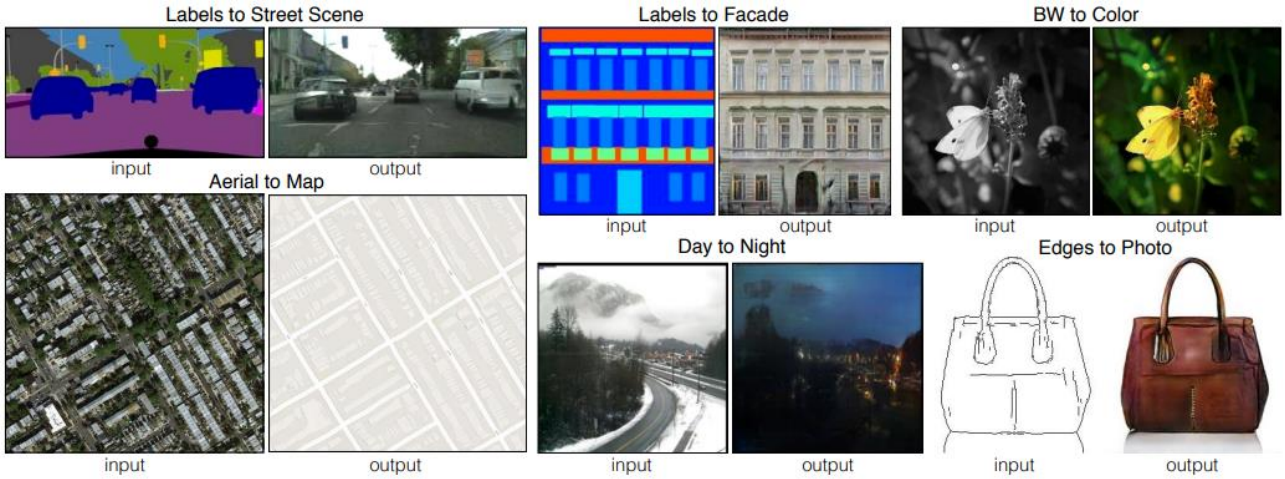

Figura 2.1: Ejemplo de algoritmo generativo [1]

Existen múltiples ejemplos en este ámbito, y se conocen como modelos generativos. Es una especialidad bastante novedosa, y con mucha utilidad y potencial en el futuro.

Teniendo ya una visión global de todo lo que incluye la detección de imágenes y la visión artificial en la tecnología, se va a entrar más en detalle en los objetivos de este trabajo. Se aplicarán técnicas que se han aprendido dentro del grado de Ingeniería Informática, pero también se ampliará lo necesario para poder realizar técnicas de *Deep Learning* más avanzadas.

3 Justificación y objetivos

Este trabajo se enmarca en la realización del Trabajo Fin de Grado en la carrera de Ingeniería Informática. La principal idea es profundizar en un tema tan interesante y útil como son las redes neuronales y más en concreto, en las redes neuronales convolucionales. Estas redes son "expertas" en el campo de la visión artificial y son la base de gran parte de la evolución que se ha desarrollado hoy en día en el ámbito de la detección de imágenes. Más adelante se profundizará más en los fundamentos teóricos de estos elementos. Además, como parte novedosa, se utilizarán dos modelos computacionales, R-MAC y Regional Attention Network para mejorar la precisión y, con ello, los resultados que se quieren obtener.

El objetivo principal de este trabajo es la predicción de la altura de una montaña a partir de una imagen de la mencionada montaña. Para ello, y usando los métodos que se han comentado anteriormente, se puede dividir el trabajo desarrollado en cuatro partes principales. Las dos primeras secciones se encuentran en el apartado de *Contexto tecnológico,* e incluirán imágenes para ilustrar lo que se comenta. Los resultados de la tercera y cuarta parte serán los resultados finales:

1- Manejo del módulo R-MAC

Primero, como base del trabajo, se encuentra el módulo R-MAC. La idea principal es dividir la imagen principal en regiones más pequeñas, y, a partir de esos segmentos más pequeños, hacer que la red busque la información que queremos. Al hacer un problema más pequeño, como dice el refrán "Divide y vencerás", este se vuelve más fácil de tratar. Si tenemos una fotografía de gran tamaño, será más fácil conseguir datos de partes más pequeñas que de partes grandes, evidentemente.

Después, cuando hemos conseguido esa información, nos será mucho más fácil discernir qué zonas nos sirven y cuáles no. Se explicará más convenientemente en la explicación de este apartado. Para comprender todo este concepto se usará el conocido dataset de *Oxford 5k*, que contiene fotos de distintos lugares icónicos de la ciudad de Oxford. Con ello, se para poder ver el funcionamiento de R-MAC.

2- Manejo del módulo Regional Attention Network

Este mecanismo es la continuación lógica de R-MAC. Si R-MAC dividía la imagen en fragmentos para buscar y extraer características, tiene sentido que los trozos que contengan la información que nos interesa tengan más valor que otros que no la tienen. Por lo tanto, Regional Attention Network pondera (en cierto modo) esas subdivisiones con el objetivo de mejorar la precisión del sistema. Se usará en este caso también el dataset de *Oxford 5k*, y usaremos un módulo ya pre entrenado, el cual nos detectará de forma muy precisa las fotos que le serán introducidas (sobre todo de edificios).

3- Entrenamiento del módulo Regional Attention Network.

Más adelante, se va a adaptar R-MAC y Regional Attention Network al tema de este trabajo, que son las montañas. Mientras que el anterior módulo era experto en detectar reproducciones de edificios de Oxford, se va a crear un módulo especialista en extraer

información de fotos de monte. Al realizar este apartado, se espera conseguir un sistema que podrá obtener de forma precisa donde se encuentra un pico montañoso en una imagen de entrada .

4- Predicción de la altura y resultados finales

Por último, una vez que se tiene la montaña ya detectada, se va a tratar de conseguir predecir su altura de la forma más precisa posible. Para esta labor, se usará como base la red Resnet101 pre entrenada con la base de datos ImageNet, junto a los dos módulos anteriormente comentados. Como dataset, se tendrán en un principio 15000 imágenes de montes, junto a un archivo con sus alturas correspondientes. Se explicará más detalladamente en ese apartado también.

4 Contexto tecnológico

4.1 Introducción: Machine Learning

El *Machine Learning* (o aprendizaje automático) es una rama del campo de la Inteligencia artificial [2]. La característica fundamental de este paradigma es que las máquinas aprenden solas y se convierten en "expertas" en un ámbito sin necesitar que el usuario las codifique para ello. Ser expertas en un ámbito significa que, para una serie de entradas de un tema, la máquina sabe cómo reconocer y tratar esos inputs. El abanico de posibles materias de aplicación de estos algoritmos es muy extenso, pudiendo ser aplicados en detección de imágenes, en robótica, en medicina, etc.

Dentro de este conjunto, se puede hacer otra distinción: el *Deep Learning* (o aprendizaje profundo). Este tipo de *Machine Learning* se basa en hacer que la máquina, además de aprender, entrene para mejorar los resultados [2]. Mientras que en el aprendizaje automático hay una serie de modelos muy utilizados, como pueden ser los árboles de decisión, el algoritmo KNN o el método SVM, dentro del *Deep Learning* el principal sistema es la **red neuronal**. Esta técnica es la que se ha implementado en el trabajo, mediante una variante conocida como CNN (Red Neuronal Convolucional). La red está completamente basada en el cerebro biológico, por lo que para comprender su funcionamiento será necesario comenzar por una explicación del sistema nervioso de los seres vivos.

4.2 Red neuronal

4.2.3 Biología

El cerebro humano es todavía en la actualidad un gran misterio. Científicos, neurólogos y biólogos llevan estudiándolo décadas, pero aún se desconoce gran parte de su funcionamiento. Con más de 100 billones de neuronas y 100 trillones de conexiones sinápticas [3], es una estructura tremendamente compleja. Esta estructura biológica es en la que los padres de la inteligencia artificial se basaron para construir los primeros modelos de redes neuronales en los años 50 y 60.

La neurona es la célula principal del sistema nervioso. Es, entre otras cosas, la estructura encargada de transmitir información en el cerebro. Además de compartir los componentes habituales de una célula, consta de cuatro partes que son exclusivas de ella: las dendritas, el soma, el axón y los botones sinápticos [4]. La función primordial que tiene una neurona es el traspaso de información de unas a otras, creando una interconexión de millones de células. Esta conexión entre neuronas se conoce como **sinapsis** [5].

Las dendritas son la parte de la neurona que recibe la información de otra neurona. Esa información es procesada por las propias dendritas y por el soma, el cuerpo de la neurona. En el soma se encuentra también el núcleo de la célula en cuestión y es el punto donde se suman todas las entradas de la neurona [6]. En esta fase se decide si se envía la información recibida o si por el contrario se inhibe. Por medio del axón, los neurotransmisores llegan hasta los botones sinápticos, que entran en contacto con las dendritas de la siguiente célula y transfieren la información. En la *Figura 4.1* se puede ver la estructura de una neurona ejemplo.

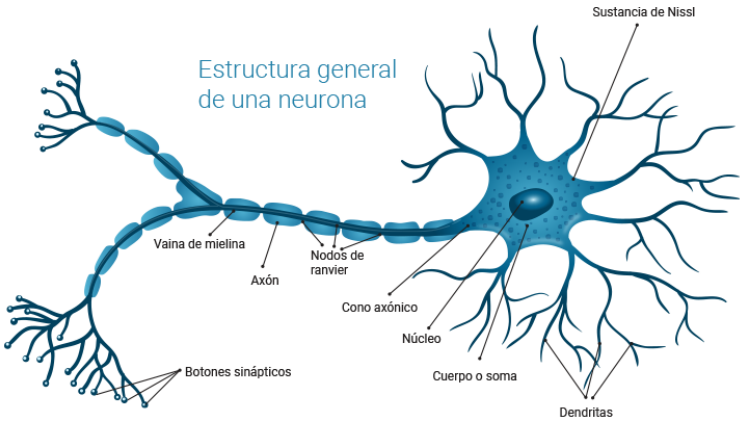

Figura 4.1:Estructura general de una neurona [18]

El proceso sináptico es un proceso complejo, que se basa en pequeños cambios de potencial de la neurona. Esos pulsos se denominan **potenciales de acción** [6]. La sinapsis más común es la sinapsis química (aunque también existe la sinapsis eléctrica). Un impulso eléctrico, generado en el cuerpo de la neurona, hace que se liberen los neurotransmisores. Estos, a su vez, se concentran en los botones sinápticos, haciendo que una dendrita de la siguiente neurona reciba y propague la información.

El conjunto de neuronas que existe en el cerebro se conoce como red neuronal, y es un concepto que se intenta imitar en el ámbito computacional. Aunque más adelante se hablará de

las redes neuronales convolucionales, es interesante comprender la idea general de lo que se quiere reproducir artificialmente. En la *Figura 4.2* se puede ver una comparación entre el cerebro humano, el cual activa una serie de receptores cerebrales (redes neuronales biológicas) para acabar comprendiendo de que se trata. Esa sucesión de pasos es la adaptación que se trata de conseguir en la computación, para conseguir que el ordenador pueda "ver".

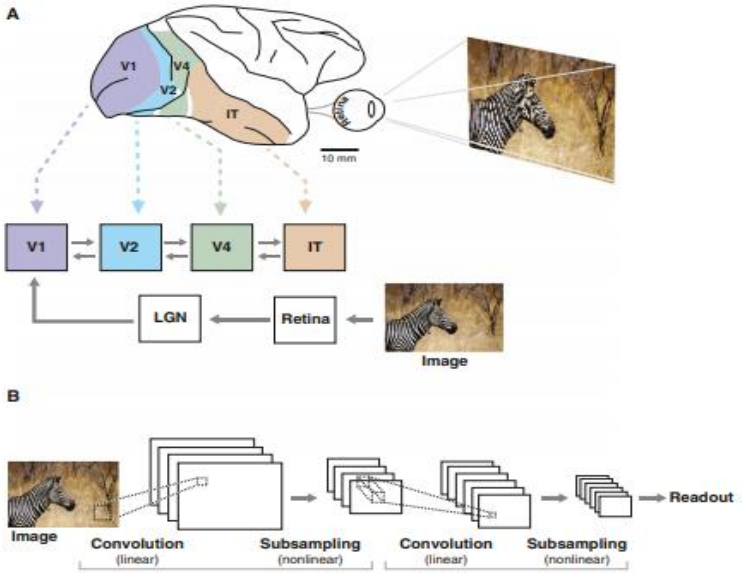

Figura 4.2: Comparativa del cerebro con la red neuronal convolucional [3]

4.2.4 Perceptrón

El perceptrón es la forma más básica de red neuronal. Esta estructura fue desarrollada por primera vez entre los años 1950 y 1960 por el científico Frank Rosenblatt. Este algoritmo y sus predecesores han supuesto un avance muy grande en el campo de la computación. Este elemento, el perceptrón, toma como base estructural la neurona del cerebro humano: recibe una serie de entradas (o "inputs") y devuelve una respuesta, activándose o no [7]. Observando la *Figura 4.3* se puede visualizar esta idea, y se va a realizar la exposición a partir de ella.

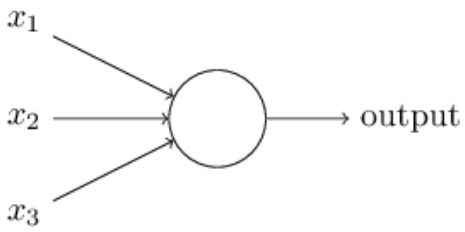

Figura 4.3: Perceptrón básico [14]

El sistema recibe entradas, en este caso x1, x2 y x3. Cada uno de esos inputs va a tener asignado un peso, los cuales se van a denominar w1, w2 y w3 respectivamente. Estos valores van a representar la importancia de cada una de las entradas con respecto al output a devolver. Ya ha sido comentado que la salida del perceptrón va a ser 0 o 1, es decir, se activa o no. Pero ¿cómo

decide el valor de vuelta a partir de las entradas y los pesos?. Este valor es determinado por la suma ponderada de los inputs, y si este valor resultante es mayor o menor que un umbral (threshold), devuelve la correspondiente salida. En la *Figura 4.4* se puede observar cómo algebraicamente se puede determinar este concepto.

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Figura 4.4: Salida del perceptrón [14]

Es decir, el perceptrón "toma decisiones" en base a las entradas que se le asignan en un principio y a los pesos asignados a cada entrada. Evidentemente, si se le colocan diferentes valores, la salida será diferente y se creará un diferente modelo de decisión. Se puede afirmar que el perceptrón es muy útil en sistemas de clasificación binaria, por ejemplo, donde hay un par de clases que se pueden separar por una línea recta.

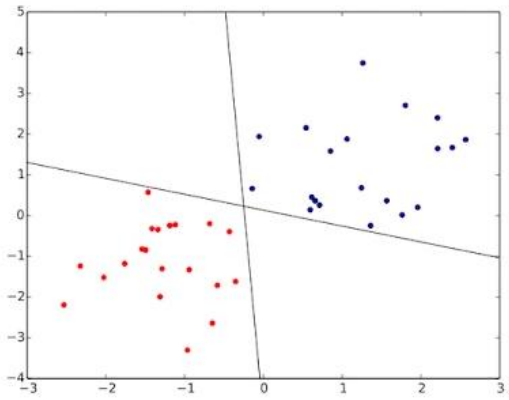

Figura 4.5: Ejemplo de clasificación binaria [7]

En la *Figura 4.5* se puede ver cómo un perceptrón separa linealmente la clase roja y la clase azul. Sin embargo, cuando queremos entrar en problemas más complejos es necesario ampliar la potencia de la red. Incluso para problemas de clasificación binaria más complejos, como en clásico caso del XOR, el perceptrón puede no ser capaz de resolverlo correctamente. Para ello, y siguiendo con el modelo del cerebro humano, que tiene una serie de neuronas interconectadas entre sí, la idea consiste en juntar una serie de perceptrones simples creando un perceptrón multicapa, también llamado red neuronal. Se va a hablar a continuación del funcionamiento de la red neuronal y, lo que es más importante, cómo se consigue que la propia red aprenda en base al error obtenido.

4.2.5 Red neuronal

Se puede decir que una red neuronal es la continuación lógica del perceptrón. En la *Figura 4.6* se puede ver un ejemplo sencillo de red neuronal.

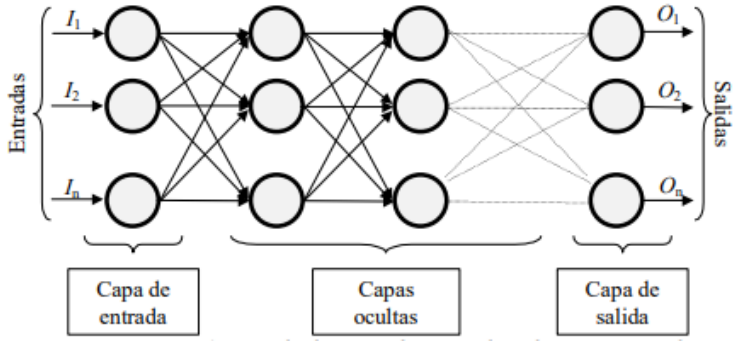

Figura 4.6:Ejemplo de red neuronal [10]

La estructura básica, y que será encontrada en cada red neuronal que exista, tiene tres elementos principales: una capa de entrada, una serie de capas ocultas y una capa de salida. La capa de entrada tendrá un número *n* de inputs. Las capas ocultas harán pasar a los datos introducidos por ellas, extrayendo información y características importantes y la capa de salida devolverá el resultado o resultados de la operación. Al combinar una amplia cantidad de neuronas, se pueden llegar a conseguir resultados muy buenos para problemas complicados, cosa que con neuronas únicas no es posible.

Se van a comentar ahora las diferentes funciones que existen dentro de las redes neuronales: de entrada, de activación y de salida. Estas funciones son las que realizan el paso de los inputs por la red, y las que permiten obtener soluciones. Es evidente que el cambio de una de ellas por otra alterará el sistema interno de la red y modificará el resultado final.

En el perceptrón no existían estos tres tipos ahora mencionados, ya que estaban incluidos en la función final. Ahora, al pasar a un modelo en el que hay un alto número de perceptrones en vez de uno sólo, es necesario cambiar el procedimiento, ya que el anterior es demasiado simple.

4.2.5.1 Funciones de entrada

Estas funciones son las que consiguen recoger las múltiples entradas que le llegan a una neurona y, combinándolas, devolverlas como un único valor. Es algo muy parecido a la suma ponderada que ha sido comentada en la explicación del perceptrón, con la que se combinaban las entradas para obtener una salida. Ahora, teniendo una alta cantidad de neuronas, cada una de ellas, al recibir los datos, empleará una función de entrada para "recoger" todos los inputs. Hay que decir que para cada neurona de la red se empleará una misma función (todas ellas tendrán la misma).

Aunque la más común es el sumatorio ponderado de las entradas, hay otras también muy usadas. Se van a mencionar las tres principales. Primero, la que ya se ha comentado, el sumatorio. Es la suma de los valores de entrada multiplicados por un peso *w*, como se ve en la *Figura 4.7*.

$$\sum_j \left(n_{ij}\, w_{ij}\right), \quad \text{con } j = 1, 2, \ldots, n$$

Figura 4.7: Sumatorio de las entradas [10]

La siguiente más usada es el productorio de las entradas. Es la misma idea que el sumatorio, pero con la multiplicación de los inputs y los pesos entre ellos. En la *Figura 4.8* se puede ver su forma algebraica.

$$\prod_{i} \left(n_{ij}\ w_{ij}\right), \quad \text{con } j = 1, 2, ..., n$$

Figura 4.8:Productorio de las entradas [10]

Por último, se puede usar como función de activación el máximo de todas las entradas pesadas. Se multiplica a cada entrada por su peso y se escoge el máximo de los resultados, como se observa en la *Figura 4.9*.

$$\underset{j}{\text{Max}}\left(n_{ij}\ w_{ij}\right) \quad \text{con } j = 1, 2, ..., n$$

Figura 4.9:Máximo de las entradas [10]

4.2.5.2 Función de activación

Una función de activación es una función que decide si la neurona se activa o no. Es decir, que determina si a partir de unas entradas y de su combinación mediante una función de entrada, devuelve un valor 0 (o -1 en algunos casos) o 1. Haciendo una equivalencia con la biología, esto sería lo mismo que el funcionamiento de una neurona, la cual decide traspasar la información que le llega (valor 1) o que se inhibe y no transfiere nada (valor 0).

Para ello existen varios tipos de funciones a comentar, desde unas funciones que son poco restrictivas a otras más usadas y con las que se obtienen mejores resultados. La más simple que se va a tratar es la función escalón, y luego se comentarán la función sigmoide, la función tangente hiperbólica, la función RELU y la función Softplus.

- Función escalón

La función escalón, también llamada Binary Step, es la más simple de todas. Su funcionamiento consiste en que la neurona se activa si el valor de entrada a dicha neurona es mayor que 0, mientras que si es menor que 0 no se activa. En la *Figura 4.10* se puede ver una representación gráfica de esta idea.

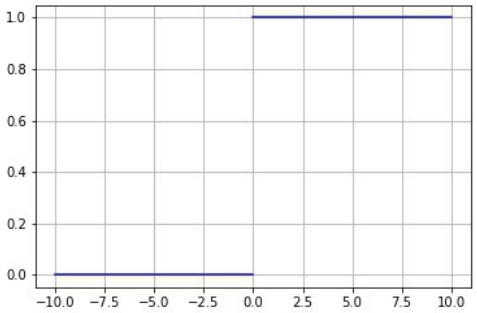

Figura 4.10:Gráfica de la función escalón [16]

Como se puede observar, si el valor de entrada es menor o mayor que 0, no hay problema. Sin embargo, si justamente el valor es 0, ¿qué se podría hacer?. La solución es decidir en ese caso especial si la neurona se activa o no. Si es exactamente igual a 0, por ejemplo, se puede elegir que la neurona devuelva 1. Matemáticamente, esto podría ser definido como se precisa en la *Figura 4.11*:

$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

Figura 4.11: Expresión algebraica de la función escalón [8]

Esta es la forma más simple de definir una función de activación. Es la misma que se determinó en el perceptrón [8], siendo aquí el umbral igual a 0. Sin embargo, es una función que tiene unas limitaciones que impiden que se usen en redes neuronales avanzadas. Las redes neuronales más utilizadas, mediante un mecanismo de aprendizaje, consiguen ajustar los pesos de la red automáticamente partiendo del error obtenido. Esto es posible en gran medida gracias a las derivadas de las funciones de activación. Si realizamos la derivada de la función escalón, nos resulta 0, por lo que el mecanismo de adiestramiento no va a poder progresar en la modificación de los pesos [9]. Por ello, se usan otro tipo de funciones en las que la derivada es distinta de 0 y por ello hace que la red mejore su precisión.

- Función sigmoide

La función sigmoide, también conocida como función logística, es una función muy común en problemas de machine learning. Como características principales, se puede comentar que convierte cualquier rango de valores de entrada en valores equivalentes en el rango [0,1]. Además, esta función, así como las del resto de su familia (como RELU o la tangente hiperbólica) suavizan los outliers que se pueden encontrar dentro del dataset [8], ya que el rango de los extremos está definido a 0 o 1.

En las siguientes figuras podemos observar una gráfica de la función sigmoide (*Figura 4.12*) y de su expresión matemática (*Figura 4.13*).

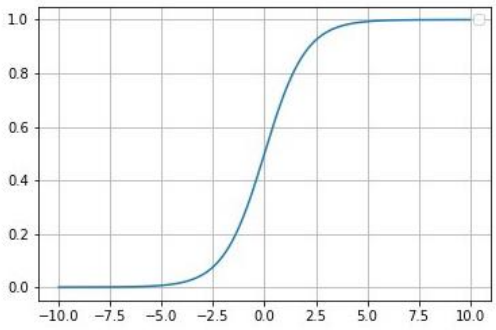

Figura 4.12: Gráfica de la función sigmoide [16]

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Figura 4.13: Expresión algebraica de la función sigmoide [8]

- Función tangente hiperbólica

La función tangente hiperbólica, a diferencia de la sigmoide, devuelve valores en un rango de [-1,1]. Tiene además la ventaja de que puede manejar mejor los números negativos [8]. En la *Figura 4.14* se puede ver la gráfica y en la *Figura 4.15* su expresión algebraica.

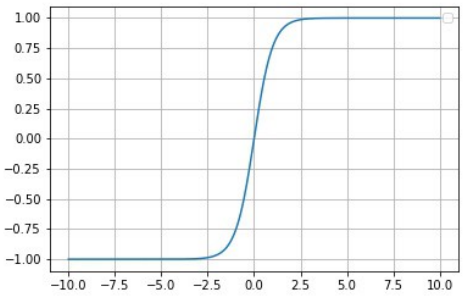

Figura 4.14: Gráfica de la función tangente hiperbólica [16]

$$\tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$$

Figura 4.15: Expresión algebraica de la función tangente hiperbólica [8]

- Función RELU

Esta función da valor 0 a todos los elementos menores o iguales de 0, y el valor de input si este valor es mayor que 0. Es muy fácil de implementar, pero puede causar problemas si muchas de las neuronas, usando esta función, devuelven 0, ya que no se podrían derivar (y por ende aprender). Se han diseñado múltiples soluciones a este problema, pero se va a comentar la función Softplus, que es una especie de suavizado de la función RELU. En la *Figura 4.16* se puede ver la gráfica y en la *Figura 4.17* su expresión algebraica.

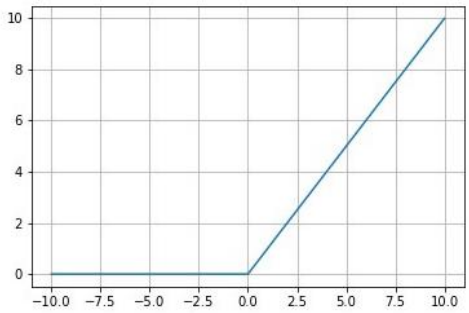

Figura 4.16: Gráfica de la función RELU [7]

$$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$$
$$= \max\{0, x\} = x\mathbf{1}_{x>0}$$

Figura 4.17: Expresión algebraica de la función RELU [2]

- Función Softplus

La función Softplus, como se ha dicho anteriormente, es una función que suaviza la función RELU. Ambas tienen un rango de [0,+inf], pero la diferencia fundamental es que la función Softplus es completamente derivable en todo su rango, lo que la hace más apta para el aprendizaje. Es también muy comparada con las funciones tangente hiperbólica y sigmoide, ya que su uso está muy extendido en la actualidad. Esta será la función de activación que se usará en este trabajo. En la *Figura 4.18* se puede ver la gráfica y en la *Figura 4.19* su expresión algebraica.

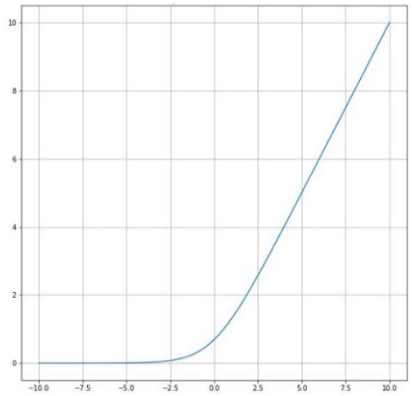

Figura 4.18: Gráfica de la función Softplus [7]

$$\ln(1 + e^x)$$

Figura 4.19: Expresión algebraica de la función Softplus [2]

4.2.5.3 Función de salida

Por último, la función de salida es la función que se ocupa de determinar el valor que se va a transferir a las siguientes neuronas. Se coloca un umbral y si el valor obtenido por la función de activación es menor que ese umbral, no se pasa ninguna salida a las siguientes neuronas. Los valores de salida suelen estar acotados en [0,1] o [-1,1]. La más común y sencilla de las funciones de salida es la propia entrada. Es decir, si el valor que entra en la neurona es superior al umbral colocado, la neurona transfiere ese valor directamente. [10]

4.2.6 Aprendizaje

Ya se han comentado todos los tipos de funciones que se usan en las redes neuronales para conseguir que los inputs pasen por el interior de la red y se consiga un resultado. En este apartado se explicará cómo una red neuronal, a partir de unos pesos de entrada, es capaz de ir modificándolos para conseguir una salida más acertada para unos datos determinados. Se puede decir, por tanto, que lo que hace interesante a las redes neuronales es su capacidad de "autorregularse" para conseguir un resultado satisfactorio. Este proceso termina cuando los pesos permanecen estables, cuando se entiende que no pueden mejorar más el resultado de salida.

Antes de avanzar, se debe comentar que no hay sólo un modelo de aprendizaje. Los modelos se pueden separar en dos tipos: el aprendizaje supervisado y el aprendizaje no supervisado. Aun con esta diferenciación, la fórmula de cambio de pesos de la red se puede generalizar de la siguiente manera [10]:

Peso Nuevo = Peso Viejo + Cambio de peso

Siendo matemáticamente lo que se muestra en la *Figura 4.20*:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$$

Figura 4.20:Generalización de la modificación de pesos [10]

Siendo $w_{ij}(t+1)$ el nuevo peso, $w_{ij}(t)$ el antiguo peso y t referencia a un nuevo paso en el aprendizaje de la red neuronal, en la cual se actualiza.

- Aprendizaje supervisado

En este modelo, el aprendizaje es controlado por un agente externo, que determina cómo de correcta es la salida de una red en base a una entrada determinada. Si ese agente, que espera un cierto valor, recibe otro resultado diferente, procederá a modificar los pesos con el objetivo de hacer que el valor resultante se acerque más a sus expectativas [10]. La salida en este modelo es, por tanto, un valor, fruto de la salida obtenida con la esperada.

- Aprendizaje no supervisado

Al contrario que en el aprendizaje supervisado, en el cual existía un elemento externo que regulaba los pesos internos de la red para obtener un resultado positivo, en este paradigma no existe tal agente. El sistema debe regular los valores encontrando por sí mismo características y peculiaridades sin ninguna información del entorno [10]. La salida en este aprendizaje, al no poder obtener un valor de como en el aprendizaje supervisado, es una medida de semejanza entre las instancias a procesar (un claro ejemplo es el caso del *clustering*, donde se asignan categorías a las entradas dependiendo de su similitud).

Aun teniendo varios modelos de aprendizaje, la red neuronal clásica funciona mediante los algoritmos de supervisión. Es cierto que existen variantes de redes para, por ejemplo, resolver problemas de clustering, pero en este trabajo resultan más interesantes los otros, ya que la red va a funcionar de manera supervisada.

4.2.7 Backpropagation

En este apartado se va a explicar cómo una red neuronal autorregula sus pesos. Podemos definir cuatro fases [11]. Este algoritmo se incluye dentro de la categoría de aprendizaje supervisado, y es de los más comunes en este campo:

1- Inicialización de los pesos

El objetivo de este proceso es optimizar los pesos para hacer que la red minimice su error. Para encontrar esos pesos óptimos primero hay que inicializarlos, lógicamente. Esta inicialización se deberá realizar de forma aleatoria, ya que de otra manera se podría estar interfiriendo en el rendimiento de la red sin quererlo.

2- Propagación hacia delante (Forward Propagation)

Una vez que se han inicializado los pesos, se calculan las salidas de la red. Las salidas, como se ha visto, se producen mediante un proceso de traspaso de información entre las capas del sistema. Usando las funciones de entrada, activación y salida, los inputs van recorriendo la red hasta llegar a la capa de salida, donde se obtiene un output. Ese output puede ser el que se espera, por lo cual no se realizaría ningún cambio en los pesos, o, por el contrario, puede que haya fallado en la predicción. En ese caso, se procedería a modificar la red.

3- Cálculo del error

Para hacer que la red cambie con respecto a lo lejos o cerca que ha estado de su predicción (ya que no es lo mismo que el sistema haya errado por poco o por mucho), hay que calcular el error. Para ello, existen múltiples opciones, como la Entropía Cruzada (Cross Entropy Loss), variantes de funciones de verosimilitud (como por ejemplo Negative Log Likelihood o Gaussian Negative Log Likelihood), o la que se usará en este trabajo, el Error Cuadrático Medio (Medium Squared Error). Este último método sigue la siguiente fórmula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\bar{y}_i - y_i)^2$$

Siendo $\bar{y}_i$ el vector de predicciones realizadas e y_i el vector de los valores reales de los inputs.

El porqué de trabajar con este tipo de error es debido al objetivo del proyecto, que es predecir la altura de una montaña teniendo solo su imagen. Se funcionará de la siguiente manera: en la red se introducirá una serie de montañas, de las cuales se sabrá la altura. La red devolverá un valor para cada una de ellas y se calculará el error. El MSE, por lo tanto, será muy útil en este caso, ya que, si la red acierta la altura de la montaña, el error será 0, mientras que si falla por muchos metros el error será muy alto.

4- Propagación hacia atrás (Backpropagation)

Conociendo ya el error resultante de la operación de Forward Propagation, se va a hacer que la red modifique los pesos mediante la propagación hacia atrás. Debido a que el objetivo es hacer que la salida se parezca lo máximo posible a la entrada, tiene sentido que se trate de minimizar el fallo para cada una de las neuronas que forman el sistema.

Usando la regla de la cadena, podemos calcular cuánto ha tenido que ver un elemento de la red en el error total. Tiene sentido que se ajuste más una neurona que provoca un mayor fallo en el sistema que una que provoca uno menor. Queremos obtener la derivada parcial del error total con respecto a cada neurona (por eso eran interesantes las funciones de activación derivables).

Por ejemplo, supongamos que queremos calcular esto para una neurona x_i que se encuentra en la última capa del modelo (es decir, que está directamente conectada con la salida). En la *Figura 4.21* podemos observar la estructura.

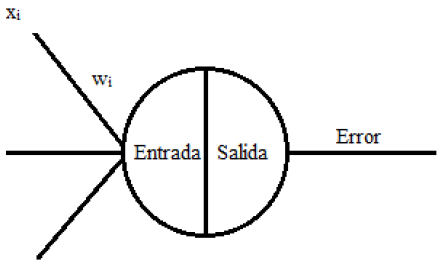

Figura 4.21:Ejemplo de Backpropagation

Para su peso w_i, se calcularía la modificación de pesos de la siguiente manera. Primero, la derivada parcial se realizaría según la *Figura 4.22*. Con ello, se puede conocer que "responsabilidad" tiene la neurona y su peso en el error total generado.

$$\frac{\delta Error}{\delta w_i} = \frac{\delta Error}{\delta Salida} * \frac{\delta Salida}{\delta Entrada} * \frac{\delta Entrada}{\delta w_i}$$

Figura 4.22: Derivada parcial de w_i

Después, para modificar w_i, se haría según la *Figura 4.23*. Dependiendo del valor de la derivada parcial, el peso se cambiará más o menos. Además, se añade un parámetro λ, llamado ratio de aprendizaje, que marca como de rápido va a aprender la red. Con un λ pequeño, el modelo modificará los pesos de manera lenta, mientras que con un λ elevado los cambiará rápido (pudiendo caer en mínimos locales). Cualquier opción tiene sus ventajas e inconvenientes, por lo que elegir el factor de aprendizaje cuidadosamente es vital.

$$w_{i'} = w_i - \lambda * \frac{\delta Error}{\delta w_i}$$

Figura 4.23: Modificación del peso w_i

Esta tarea se realizará en cada una de las neuronas que conforman la red. Evidentemente, para las capas internas habrá que añadir componentes a la derivada parcial, pero se ha elegido este ejemplo por su simplicidad. Este proceso de modificación de pesos se irá realizando hasta que la salida no pueda mejorar más, y con ello el error se haya reducido lo máximo posible.

4.3 Red neuronal convolucional

4.3.1 Introducción

En este apartado se comentará la evolución del concepto de red neuronal: la red neuronal convolucional (o CNN, Convolutional Neural Network). Esta idea es la adaptación de la red neuronal clásica a la detección de imágenes. Las redes normales funcionan bien tomando decisiones sobre inputs simples, pero a la hora de tratar con datos más complejos, como las imágenes, no sirven. Ahí entran las redes neuronales convolucionales.

Este tipo de sistema es el causante de que, hoy en día, podamos, por ejemplo, tener reconocimiento de voz en nuestros dispositivos móviles, que existan algoritmos que generen cuadros desde cero copiando el estilo de pintores clásicos (ver redes neuronales generativas), o en nuestro caso, que podamos detectar la altura de una montaña a través de su imagen.

Primero se verán conceptos específicos de las redes neuronales convolucionales, como la idea de convolución, kernel o padding. Después, aplicando esos conceptos, se van a explicar las diferencias que hay entre las redes clásicas y las redes convolucionales (destinado a las imágenes), y cuál es su funcionamiento interno.

4.3.2 Características de la red neuronal convoucional

- Kernel

El kernel es, en las redes neuronales, el filtro que se le aplica a una imagen para conseguir cierta información o característica. Esta información puede ir desde obtener los bordes de una imagen hasta enfocar o desenfocar la misma. Es, en definitiva, un detector de características. En la *Figura 4.24* se puede observar una imagen a la que se le ha pasado un filtro de detección de bordes.

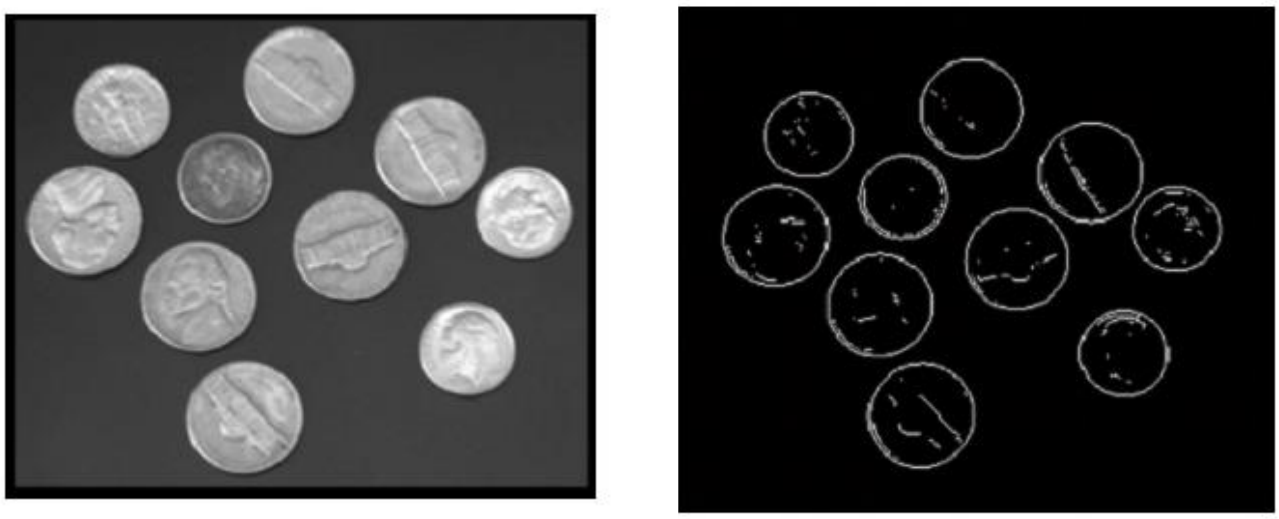

Figura 4.24: Ejemplo de kernel detector de bordes [18]

En este caso, el filtro ha sido una matriz de dimensión 3x3x1 (Altura x Anchura x Dimensiones, que en el caso de imágenes a color sería 3 en vez de 1) de forma:

$$M = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

A la imagen, en este caso de las monedas, se le aplicará el kernel de detección de bordes mediante el proceso de convolución, con el consiguiente resultado de una imagen con los bordes de las monedas.

○ Convolución

La convolución es una operación que, mediante el uso de filtros, consigue obtener de una imagen características que queremos (por ejemplo, bordes). Consiste en realizar un producto escalar de un grupo de píxeles de la imagen con un filtro (como se ha visto antes, el kernel es una matriz). El filtro irá recorriendo todas las posiciones de la imagen, realizando ese producto escalar, consiguiendo el efecto que deseamos en toda la imagen. Siguiendo con el ejemplo de las monedas de la *Figura 4.24*, se comenzaría a recoger la imagen desde la esquina superior izquierda. Se realizaría el producto escalar del filtro con la imagen y se guardaría ese valor. Se movería el kernel una posición a la derecha (como se puede ver en la *Figura 4.25*) y se repetiría el proceso con el resto de la imagen. El resultado sería una imagen con la característica deseada marcada.

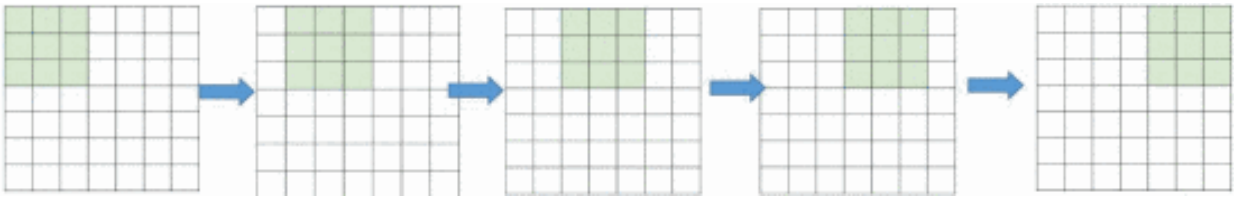

Figura 4.25: Paso de la convolución [19]

Un problema que hay que comentar de la operación de convolución clásica sobre una imagen es que la imagen resultante "reduce" su tamaño. Supongamos que, como en la *Figura 4.26*, se tiene una imagen de 6x6x1. Al aplicarle un kernel 3x3x1 a todas las posiciones de la imagen, la imagen resultante tendría unas dimensiones de 4x4x1. Esto es así debido a que el kernel no sabe qué hacer con los píxeles de los bordes, por lo que sólo los usa para operar los píxeles del centro. Si en el mismo ejemplo, el filtro tuviera una dimensión 5x5x1 en vez de 3x3x1, la imagen resultante sería una imagen 2x2x1. Este efecto se conoce como efecto de borde. Ya que la pérdida de información en los bordes es un gran inconveniente, se solucionará con la técnica de **Padding**.

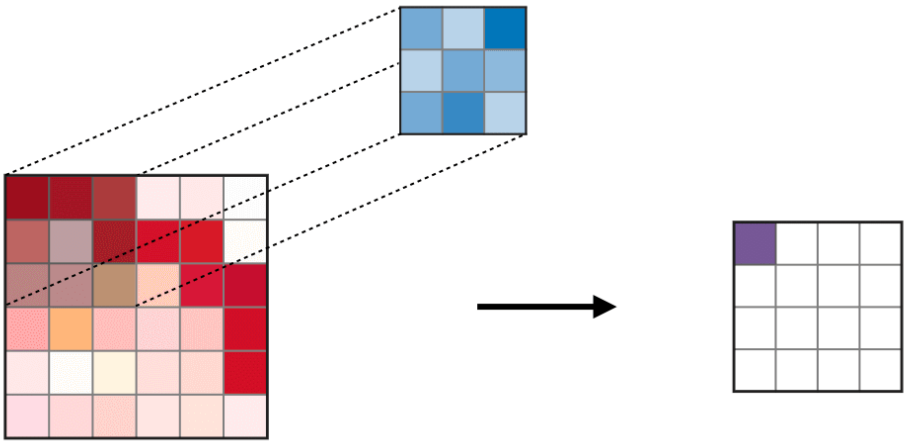

Figura 4.26: Operación de la convolución [20]

- Stride

Antes de definir qué es el Padding, se va a describir otra variable que se puede modificar dentro de este apartado. El Stride en la operación de convolución se refiere al número de píxeles que se mueve la ventana antes de cada producto escalar. En la *Figura 4.25* se ha mencionado que el filtro se mueve una posición a la derecha cada vez. En este mismo ejemplo, el stride sería igual a 1 (se mueve una posición antes de cada operación). Si por ejemplo el stride fuera igual a 2, el filtro se movería dos posiciones.

- Padding

Como antes se ha comentado, uno de los problemas de la operación de convolución es el llamado efecto de borde, donde se pierde información. Para solventar esta situación, se realiza la técnica de padding. Aunque hay muchas variantes de ella, la más común y sencilla es la de Zero-padding. Esta consiste en añadir a cada lado de la imagen un conjunto de ceros, por lo cual el filtro puede operar en los bordes sin problema. En la *Figura 4.27* podemos ver este tipo de padding. A la imagen original en amarillo se le añaden filas y columnas extra (marcadas con la línea discontinua). La imagen resultante, ahora sí, tendrá la misma dimensión que la original.

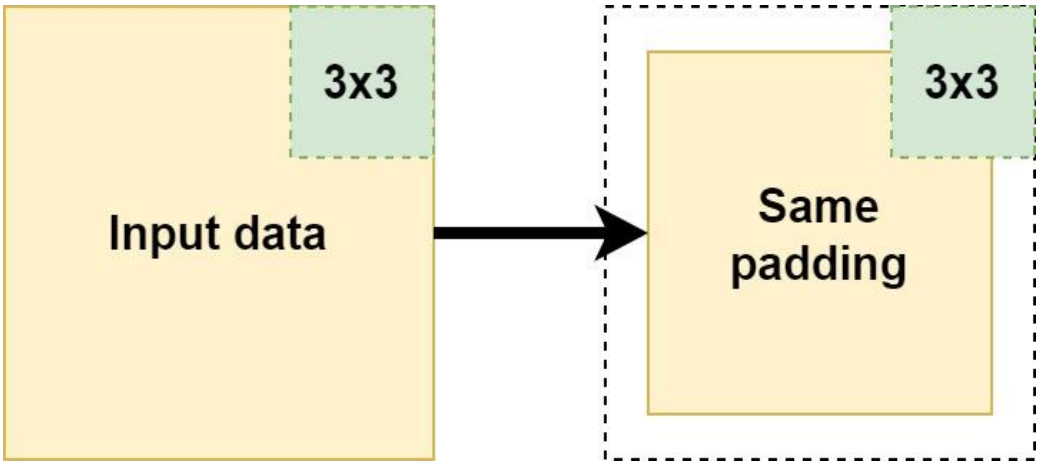

Figura 4.27: Ejemplo de Zero Padding [18]

- Pooling

El otro tipo de capas fundamentales que existen en una red neuronal convolucional, aparte de las capas de convolución, son las capas de pooling. La idea es reducir la complejidad de una capa a la siguiente, tratando de mantener el máximo de información posible. Se podría considerar como un proceso de reducción de resolución en el campo de procesamiento de imagen. Al reducir la complejidad de las capas se consigue acelerar el proceso, manteniendo gran parte de la información que nos interesa. Las técnicas de pooling más comunes son el Max Pooling (*Figura 4.28*) y el Average Pooling (*Figura 4.29*).

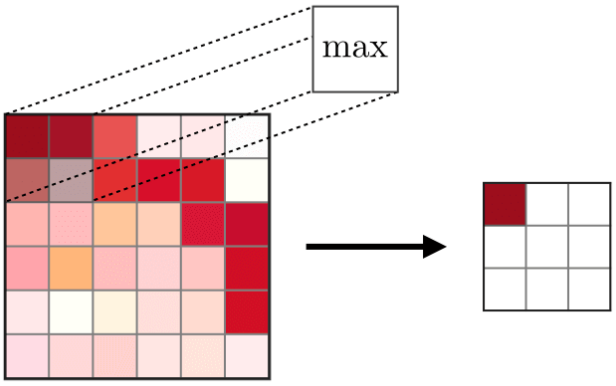

Figura 4.28: Ejemplo de Max Pooling [20]

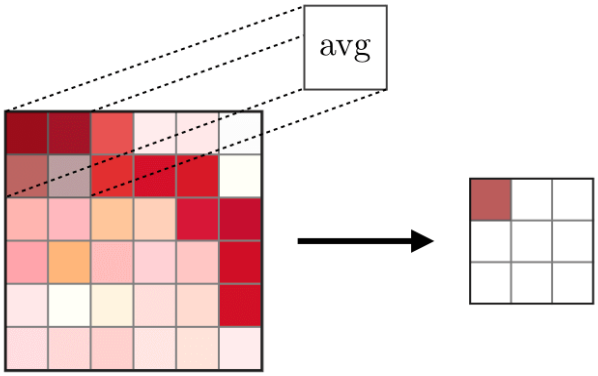

Figura 4.29: Ejemplo de Average Pooling [20]

El Max Pooling "parte" la imagen en sub cuadrados, y devuelve el valor máximo de esa región, como se puede ver en el ejemplo. Esto lo hace hasta completar toda la imagen. Con ello, se logra reducir la imagen a la mitad de su tamaño original. El Average Pooling, por otro lado, funciona de la misma manera que el Max Pooling, solo que, en vez de escoger el máximo valor de esa región, realiza la media de todos los valores y devuelve eso como "representante" de ese sub cuadrado.

4.3.3 Funcionamiento

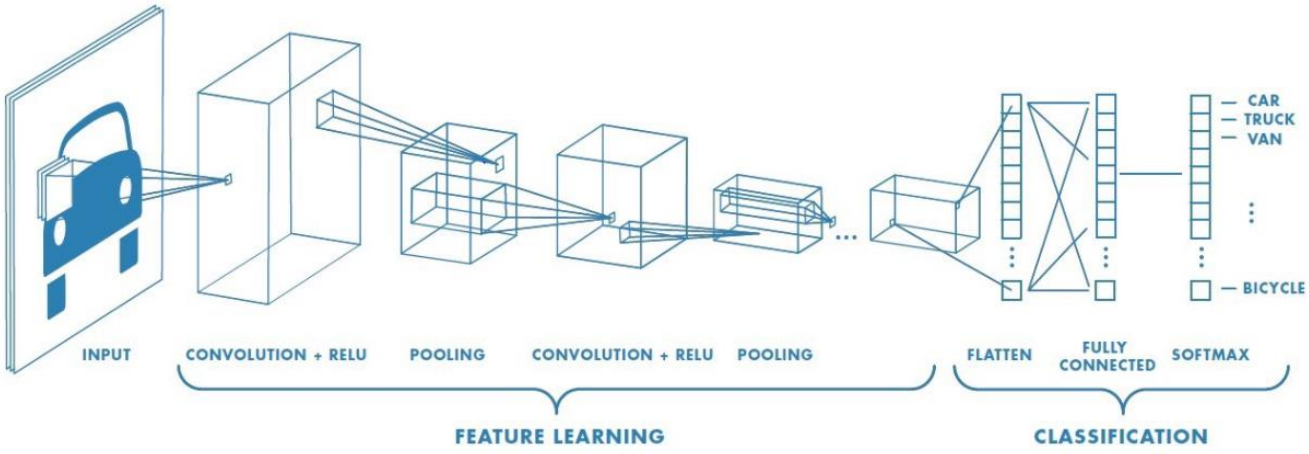

Figura 4.30: Ejemplo de Red Neuronal Convolucional [17]

Habiendo visto los elementos de procesamiento de la imagen de input en una red neuronal convolucional, como son la convolución y el pooling, se va a hablar ahora de cómo se obtiene una predicción en base a una fotografía de entrada.

Como se puede ver en la *Figura 4.30* hay dos fases. Primero se encuentra la fase de aprendizaje de características, donde se usan los conceptos del anterior apartado. Después viene la clase de clasificación (o regresión), donde esas características se pasan a una red neuronal para que esta la clasifique.

La imagen de entrada (ya sea en escala de grises o a color) se somete a un proceso en el que se alterna una convolución (aprendizaje de información) con un Pooling (reducción de información). El resultado de este proceso es una estructura que contiene toda la información interesante de la imagen en un tamaño mucho menor del de la foto en cuestión. Se puede denominar a esta información como mapa de características, ya que es el conjunto de todos los rasgos interesantes de la imagen.

Teniendo toda la información que se desea en una estructura de anchura y altura mucho menor que la imagen inicial, pero de una tercera dimensión mucho mayor que la original, se va a pasar a la fase de clasificación. Esa tercera dimensión antes se correspondía con el color de la imagen (3 si es una fotografía a color y 1 si era en escala de grises) y ahora contiene la información que se extraído con los filtros. Para ello, se va a convertir esa matriz de información en un vector columna (*flatten*). Una vez conseguido ese vector unidimensional, se va a introducir en una fully connected layer (una capa completamente conectada).

Una capa completamente conectada es una red neuronal normal, en la que todas las neuronas de una capa están conectadas con todas las neuronas de la siguiente. Se usa este tipo de estructura ya que es un modelo simple y efectivo. Es posible encontrar el caso en el que esta capa tenga a su vez varias capas ocultas, siguiendo la estructura de red neuronal expuesta en la *Figura 4.30*. Por ejemplo, en la experimentación desarrollada, se utiliza como base la red convolucional Resnet101. Esta tiene al final de su proceso una fully connected layer que conecta el aprendizaje de características con la clasificación. La información obtenida de la primera fase se guarda en una estructura con tamaño 4098x1, y se une con una capa de 2048 neuronas (por lo tanto, las 4098

neuronas de la entrada estarán conectadas con las 2048 neuronas de la siguiente capa). De esta capa se conseguirán las predicciones correspondientes.

4.4 R-MAC y Regional Attention Network

4.4.1 Introducción

Se va a dar una explicación teórica de los métodos usados en este trabajo. Primero se va a hablar del método R-MAC, que es la idea base de todo lo posteriormente realizado. Después, se explicará el método Context-Aware Regional Attention, que mejora a R-MAC y permite obtener mejores resultados en Image Retrieval.

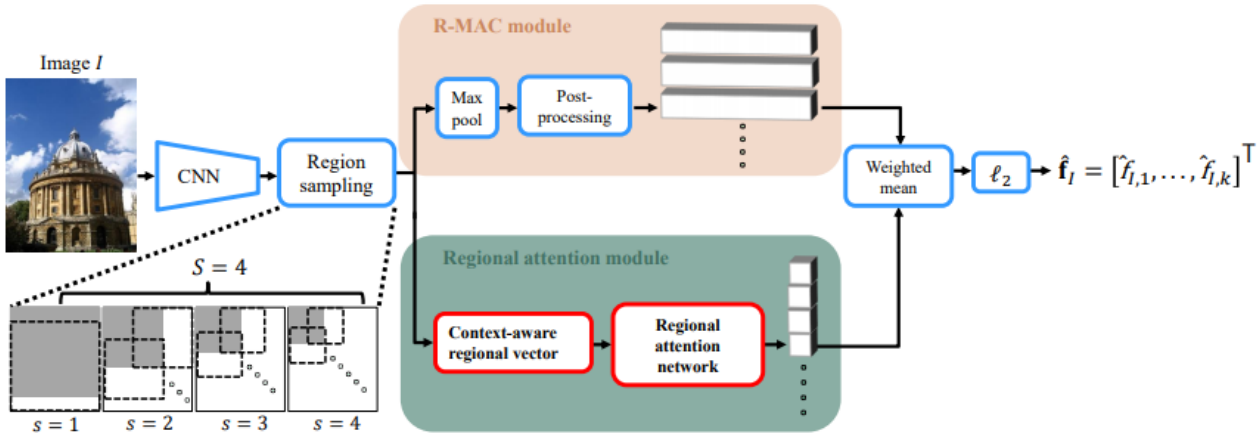

Figura 4.31: En esta imagen podemos observar el funcionamiento general de nuestro sistema. El vector fI contiene el vector de características de la imagen I que usaremos para realizar el image retrieval [12]

Para aclarar, estos recursos se introducen después de que la red neuronal convolucional ha obtenido toda la información de la imagen, y antes de pasar dicha información a una fully connected layer. Con ello se intenta de alguna manera "optimizar" y "mejorar" las características aprendidas para obtener un mejor resultado. En la *Figura 4.31* se puede percibir visualmente lo expuesto.

4.4.2 R-MAC

R-MAC (Regional-Maximum Activation of Convolutions) es un agregador de características locales en una imagen. Con esta agregación se puede discriminar en la imagen qué zonas son las que pueden identificar correctamente la imagen y cuáles no. Con estas regiones, por supuesto, se realizará la comparación con el resto de las imágenes para obtener un resultado.

El método sigue los siguientes pasos. Primero, se obtiene el mapa de características de una imagen a través de una CNN. Una vez conseguido ese mapa se hace una segmentación en

cuadrados con un tamaño R_s, siendo s una escala específica de R-MAC. Esta escala sigue una estructura de ventana deslizante, es decir, que recorre la imagen hasta haber tratado toda su superficie. Como rasgo principal de dicha ventana se puede comentar que dos ventanas vecinas tendrán entre ellas un 40% de overlap. La variable s tiene un rango s = 1..S. Se puede comprender mejor esta subdivisión de la imagen si se observa la *Figura 4.31* (Abajo a la izda.) como ejemplo de esta segmentación, siendo S = 4. El tamaño de cada Rs se puede calcular como Rs = 2min(W, H) / (s+1) con W siendo la anchura y H la altura del mapa creado por la CNN.

Después de esta división se realiza un Max-pooling para cada una de las regiones y un post-procesamiento, que consiste en una normalización l2 y un PCA-Whitening. Por último, para calcular el vector de características global, con el que se podrá realizar el image retrieval, se pueden usar dos tipos de pooling: sum pooling o mean pooling. Se ha demostrado que el mean pooling es mejor en este caso, ya que hace el entrenamiento más estable, por lo que se usará este.

En la *Figura 4.32* se va a visualizar cómo funciona este módulo con un ejemplo de prueba. La primera imagen es la imagen original, y la segunda es la imagen procesada con la que se genera el vector de características en la CNN. Las siguientes imágenes son las primeras 8 regiones que se generan con R-MAC (en este ejemplo en concreto se crean 85 regiones). Como se puede ver, las regiones con el aumento de la escala S se hacen cada vez más pequeñas.

R-MAC ha demostrado ser un método muy efectivo en image retrieval. Sin embargo, se ha podido observar que es mejorable, debido a que trata uniformemente todas las regiones valoradas, cuando lo óptimo sería que trabajara sólo con las regiones más "importantes". Por ejemplo, si se quisiera detectar en una foto un edificio, tendrían más importancia para el ojo humano las regiones en las que estuviera dicho edificio que las regiones que mostraran las nubes del cielo o el suelo, por poner un caso.

Por otro lado, se ha observado que cuando aumentamos el número de escalas (S), el rendimiento baja considerablemente [12]. Estos problemas han hecho que se use otro módulo nuevo para mejorar a R-MAC, que prestará más atención a ciertas regiones en lugar de tratarlas a todas uniformemente teniendo en cuenta el **contexto** espacial. Con ello, se mejorará R-MAC sin la necesidad de aumentar la escala.

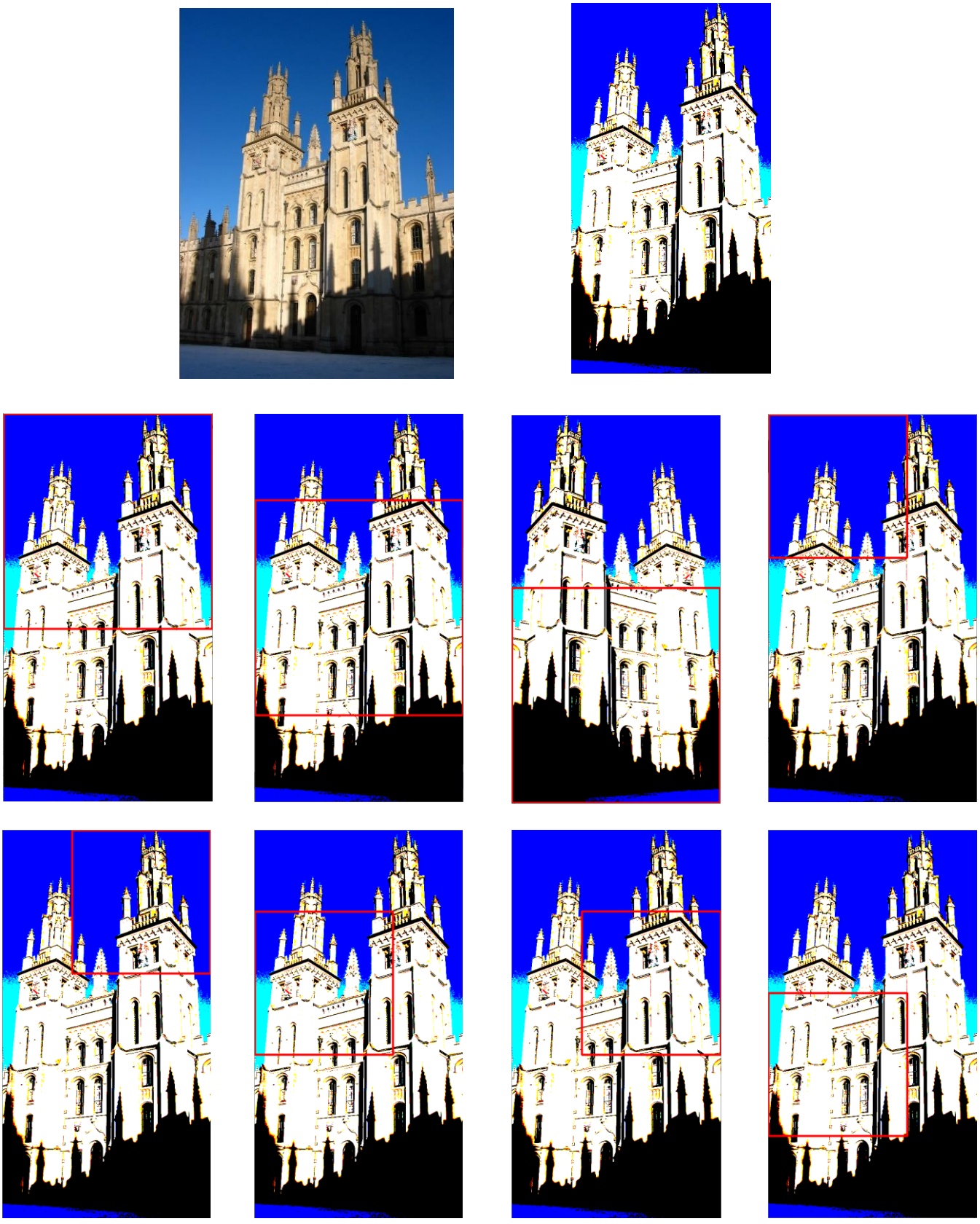

Figura 4.32: Ejemplo del funcionamiento del módulo R-MAC en la imagen all_souls_000013

4.4.3 Context-Aware Regional Attention

Notación	Desripción
$\mathbf{V_I}$	Mapa de características de una imagen I
Q	Conjunto de regiones del mapa de características de V_I
R	Una región de características del conjunto Q
P(M(R))	Una región R al pasar por un max-pooling y del post-procesado
k	Una región R tras pasar por un mean-pooling J(·)

Antes de comenzar con la explicación del método es necesario tener clara la notación que usaremos. Al mapa de características que se extrae de la CNN lo llamaremos $\mathbf{V_I}$. Es decir, CNN(I) = $\mathbf{V_I}$. Teniendo un método para construir un conjunto de regiones del mapa de características, como podría ser el anteriormente explicado R-MAC, se obtiene que **Q** es el conjunto de regiones creadas de la forma R-MAC($\mathbf{V_I}$) = **Q**. Se necesitará posteriormente también la cardinalidad de Q, |Q|.

Se llamará **R** a una de las regiones de características del conjunto total **Q**. P(M(**R**)) es un vector creado al pasar una región **R** por un max-pooling M(·) y por un post-procesamiento P(·) (una normalización l2 y un PCA-Whitening). Por último, **k** es un vector de características construido de la forma k = J(**R**), siendo J(·) un mean-pooling.

Para el cálculo del vector de características final se usará la siguiente función [12]:

$$\mathbf{f_I} = [f_{I,1}, \dots, f_{I,k}]^T = \frac{\sum_{\mathbf{R}\epsilon Q} \boldsymbol{\phi}(k)\mathbf{P}(\mathbf{M}(\mathbf{R}))}{|Q|}$$

Para obtener los pesos de cada región, con el objetivo de mejorar el método R-MAC, se utilizará una función Φ(·) sobre la anteriormente mencionada k. Dicha función se calcula de la siguiente manera:

$$\Phi(k) = \text{softplus}(W_c\pi(k) + b_c)$$

$$\pi(k) = \tanh(W_r k + b_r)$$

donde $W_r \in \mathbb{R}^{dxk}$ y $W_c\ \mathbb{R}^{1xk}$ son matrices asociadas a una transformación lineal, y $b_r \in \mathbb{R}^d$ y $b_c\ \mathbb{R}^1$ son el vector de bias y el escalar de bias respectivamente [12]. La función π(·) realiza una transformación lineal con una reducción de la dimensión del vector espacial, seguido de una función no lineal de activación como es tanh. Después, ya en la función Φ(·) se realizará una transformación lineal de vector a escalar y después una activación softplus, con lo que se consigue el peso que estábamos buscando.

Aunque este método considera diferentes pesos e importancias entre regiones, puede que sea insuficiente en algunos casos para comprender la atención que se le da a una región sin su contexto global. Por ejemplo, con el supuesto que antes se ha comentado de la detección de un edificio, puede que una subregión local tenga solamente una pequeña parte del techo de la casa, por lo que si no se valorara el resto de fotografía carecería de sentido. Sin embargo, si se evalúan las regiones de alrededor o la imagen al completo, esa región del techo pertenece al edificio, y evidentemente sí hay que darle importancia. Esta idea es la que se va a aplicar en el siguiente punto. En resumen, para poder tener en cuenta tanto la región local como el global de la imagen, se implementa un módulo para calcular el peso de una región específica teniendo el contexto espacial en el que se encuentra.

4.4.4 Context Awareness

Existen métodos que, para valorar el valor de prominencia que posee un píxel (básicamente si pertenece a un borde o no), tienen en cuenta o bien a sus vecinos más cercanos o bien a toda la imagen. El modelo aquí usado puede ser considerado como uno de estos métodos de detección que tienen en cuenta la imagen global. En la siguiente fórmula se puede ver cómo se tiene en cuenta el contexto para generar el vector de características $\mathbf{f_I}$:

$$\mathbf{f_I} = [f_{I,1}, \ldots, f_{I,k}]^T = \frac{\sum_{\mathbf{R}\epsilon\mathbf{Q}} \boldsymbol{\phi}(\mathbf{k} \oplus \mathbf{J}(\mathbf{V_I}))\mathbf{P}(\mathbf{M}(\mathbf{R}))}{|Q|}$$

donde $\oplus$ representa una concatenación de vectores en el espacio. Por lo tanto, al usar esa concatenación del vector de la región **k** (segmento local) con el vector de características $J(\mathbf{V_I})$ (segmento de la imagen global) como entrada a la función $\Phi(\cdot)$ tenemos en cuenta la información espacial al completo.

Vamos a mostrar algunos ejemplos del funcionamiento de este módulo. El mapa de calor representa la zona que recibe más importancia de la foto. El vector de características, por lo tanto, habrá tenido en cuenta esa zona más que el resto. Primero veremos la imagen original, después la imagen de los pesos y por último la que tiene el mapa de calor.

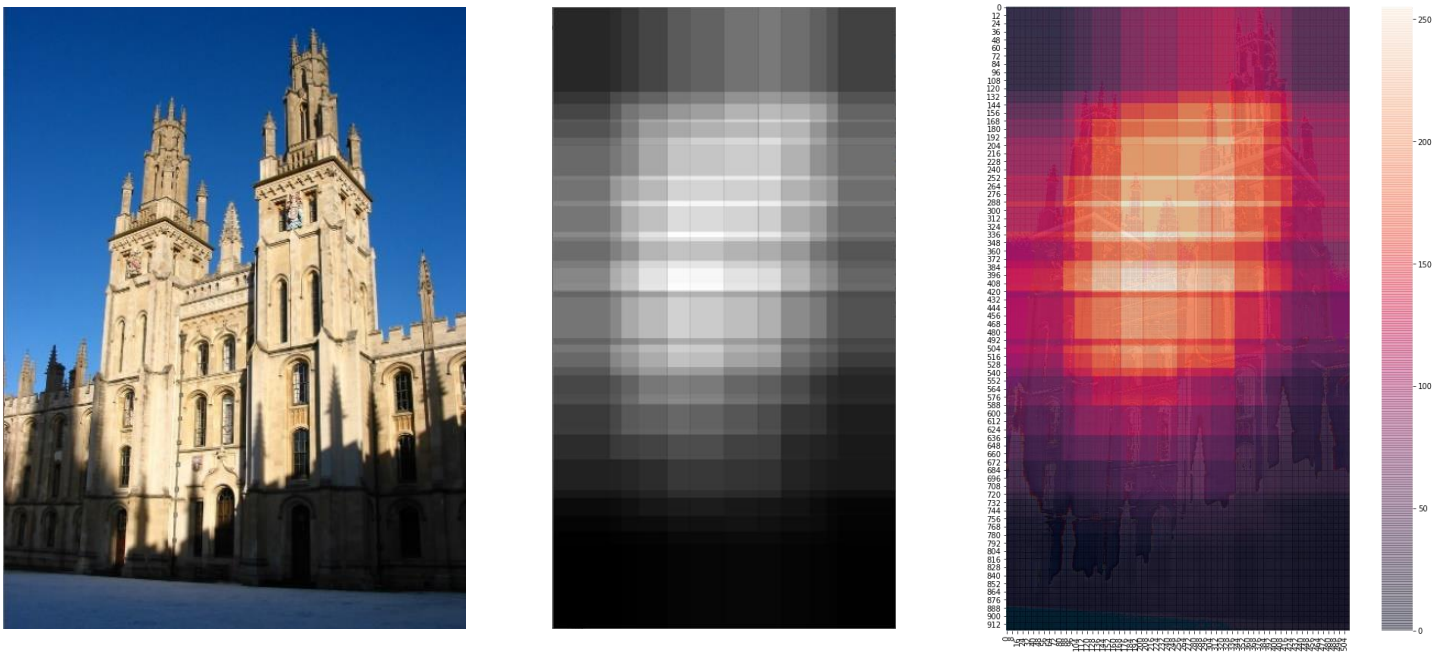

Figura 4.33: all_souls_000013

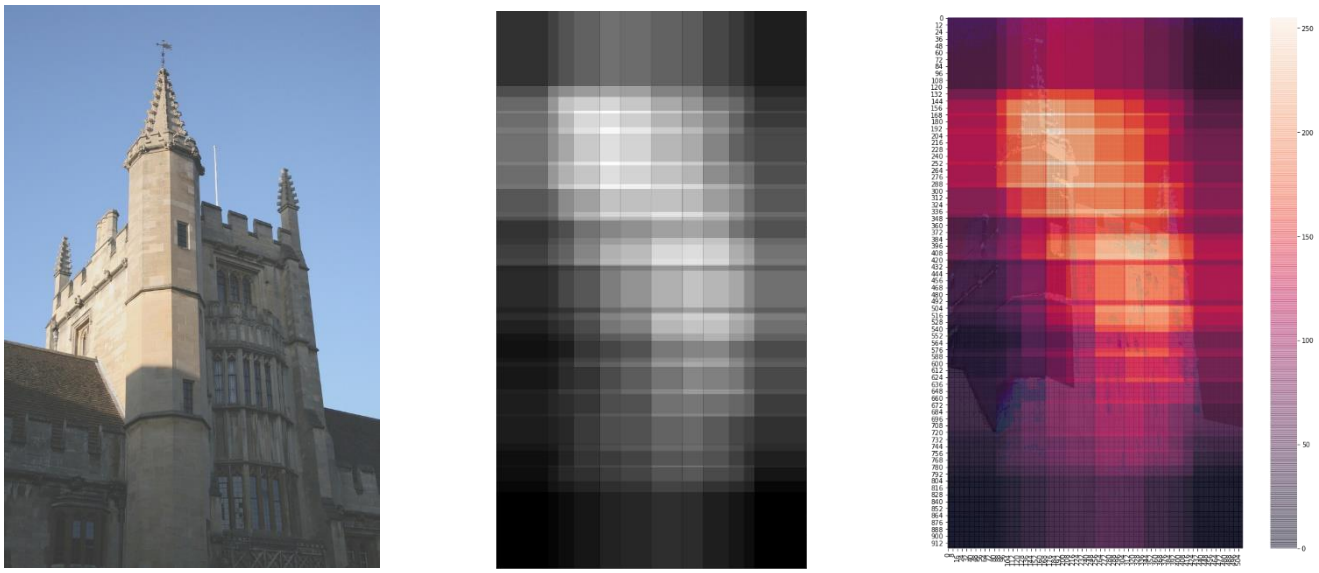

Figura 4.34: magdalen_001145

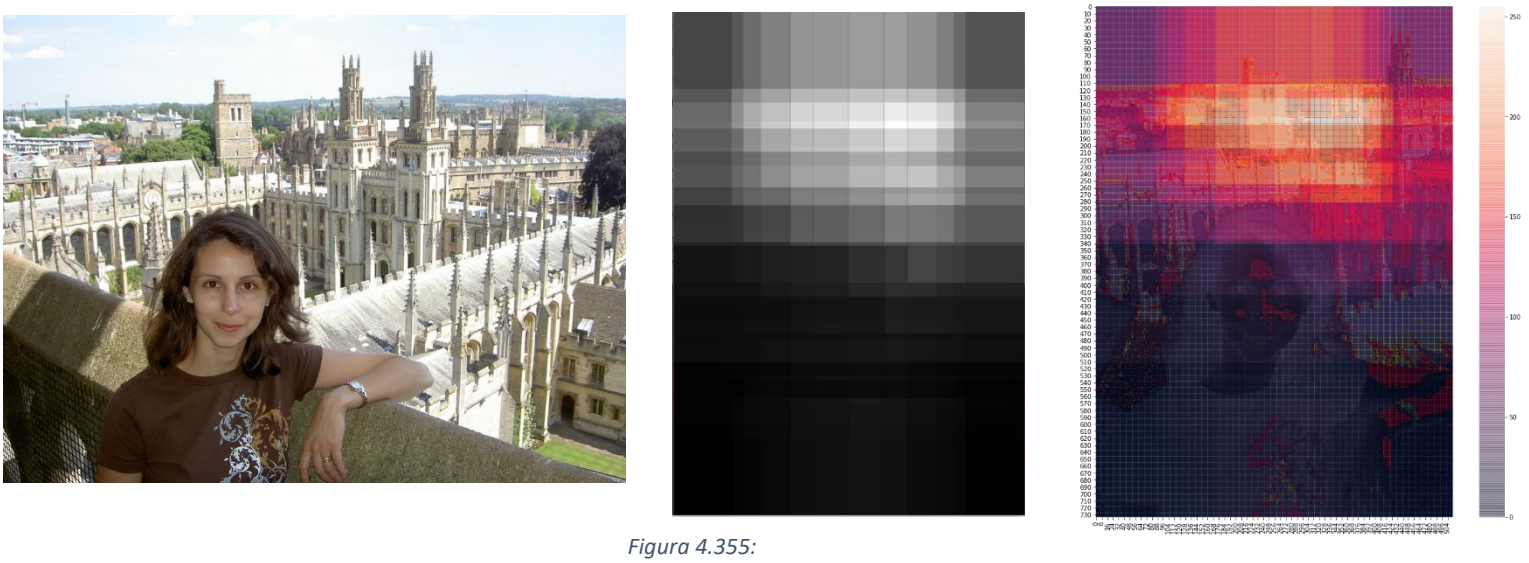

Figura 4.355: oxford_001231

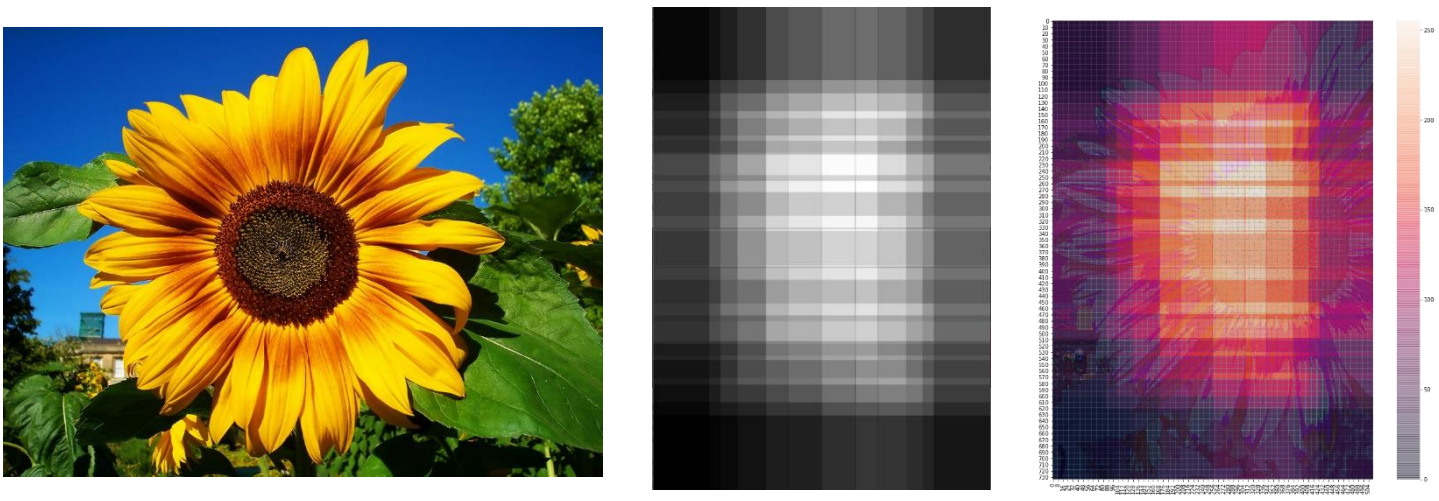

Figura 4.35: oxford_003568

Al estar la red preparada y entrenada para la detección de edificios, con el objetivo de realizar image retrieval de dichas construcciones, los mapas de calor nos muestran dónde la red neuronal se está enfocando más para su labor. Se puede ver como en la *Figura 4.33* y *Figura 4.34* la red nos marca como más importantes las zonas altas de los edificios, ya que en teoría esas partes son más distinguibles de otras obras (La parte de arriba de la *Figura 4.33 e*s diferente a la parte de arriba de la *Figura 4.34*, lo que va a hacer que nuestro modelo sepa qué edificio es cada uno).a

Es también interesante comentar el ejemplo visto en la *Figura 4.35*. Se puede ver una foto de una chica con un edificio al fondo. Como bien indica el mapa de calor, la red se enfoca directamente en la construcción de atrás, dándole ninguna importancia a la persona que se encuentra en el primer plano. Esto es muy interesante, ya que la red entrenada demuestra con este ejemplo ser robusta en imágenes con distracciones, lo que hace ser mucho más fiable (en fotografías con elementos que no corresponden a edificios, la red sigue enfocándose en las edificaciones).

Por último, en la *Figura 4.36* se puede observar una foto de un girasol, sin ninguna edificación alrededor. En este caso, el mapa de calor no tiene ningún sentido (evidentemente). Con esto se puede ver cómo la red, aun estando entrenada para detectar edificios, si le llegara una imagen de otro ámbito, intentaría sacar las máximas coincidencias de la foto para intentar "cuadrarla" en el modelo.

Con el objetivo de demostrar que, efectivamente, la red funciona mejor con regiones que sin ellas, se han obtenido resultados con el dataset de Oxford. Este dataset contiene 5000 imágenes de la ciudad, entre las que se encuentran edificios emblemáticos, personas, estatuas, etc. Anteriormente en este proyecto ya se han mostrado algunos ejemplos, como en la *Figura 4.32*, *Figura 4.33* o *Figura 4.34*, y ahora vamos a comprobar que la red es capaz de analizar, detectar y clasificar las imágenes que se le introducen.

Este proceso es el mismo que se realizará en la parte final de la experimentación. Se escogen una serie de fotografías "query", que van a servir como elementos de prueba. La red se encargará, para cada uno de esos ejemplos, ordenar el resto de las imágenes del conjunto según su parecido a las queries. Por ejemplo, para la primera prueba, que es el edificio *all_souls*, si la red funciona correctamente tendrá que colocar en primeras posiciones del ranking generado al resto de imágenes correspondientes a ese edificio. Si, en cambio, funciona de manera incorrecta, el sistema puede que confunda ese edificio con otro, con lo cual la precisión evidentemente se vería reducida.

El número de queries es 55. En la *Figura 4.*37 se va a poder observar los resultados, que son muy interesantes. En naranja se va a poder observar la red con el módulo Regional Attention y en azul sin él. En el eje de abscisas se puede ver el conjunto de queries empleadas, y en el eje de ordenadas se encuentra el error obtenido en cada prueba.

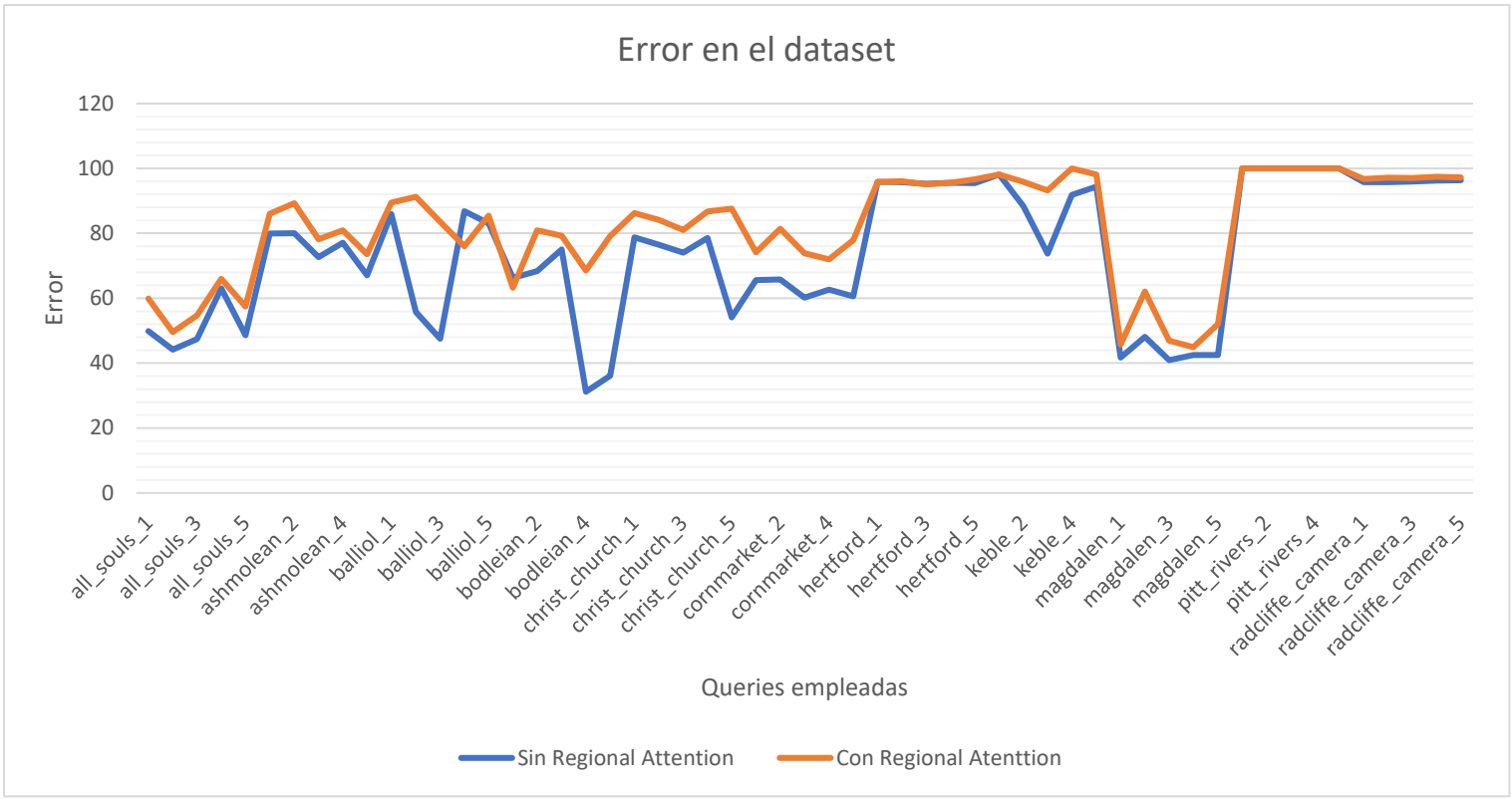

Figura 4.36: Error en el dataset de Oxford

Es evidente que, en líneas generales, el sistema con regiones funciona mejor en la detección de fotografías de ese dataset en concreto que el modelo sin regiones. Es curioso como las gráficas son bastante simétricas entre sí, haciendo ver que ambas tienen la misma base y diferentes ampliaciones. La mejora, por tanto, es indudable.

Por otro lado, la diferencia de resultados entre una query y otra se puede deber a la diferencia de dificultad entre ellas. Por ejemplo, se van a observar dos ejemplos de los resultados obtenidos, *magdalen_1*, con una precisión de menos del 50% y *pitt_rivers_2* con un acierto del 100%. Como se puede ver en la *Figura 4.38*, la estructura *magdalen* a analizar no está directamente enfocada, y además hay elementos que no pertenecen al edificio (coches, personas, un árbol,etc.) que pueden hacer dudar a la red. Sin embargo, en la *Figura 4.39* la construcción conocida como *pitt_rivers* está perfectamente enfocada y centrada en la foto, y no hay factores externos como en el anterior ejemplo. Se puede afirmar que la primera figura es más "complicada" para la red, al tener que barajar un número mayor de características.

Se puede entender, en definitiva, que estos aspectos influyen en la detección de las imágenes, y que, si se quiere construir una red robusta y que soporte las posibles variaciones entre las fotografías, se va a tener que entrenar con imágenes desde diferentes posiciones y con elementos que puedan confundir al sistema. Cuando llegue una foto de testeo difícil, si se ha entrenado el sistema de esta manera, el rendimiento que dará la red será mejor que si no se ha realizado así.

Figura 4.37: Query magdalen_1

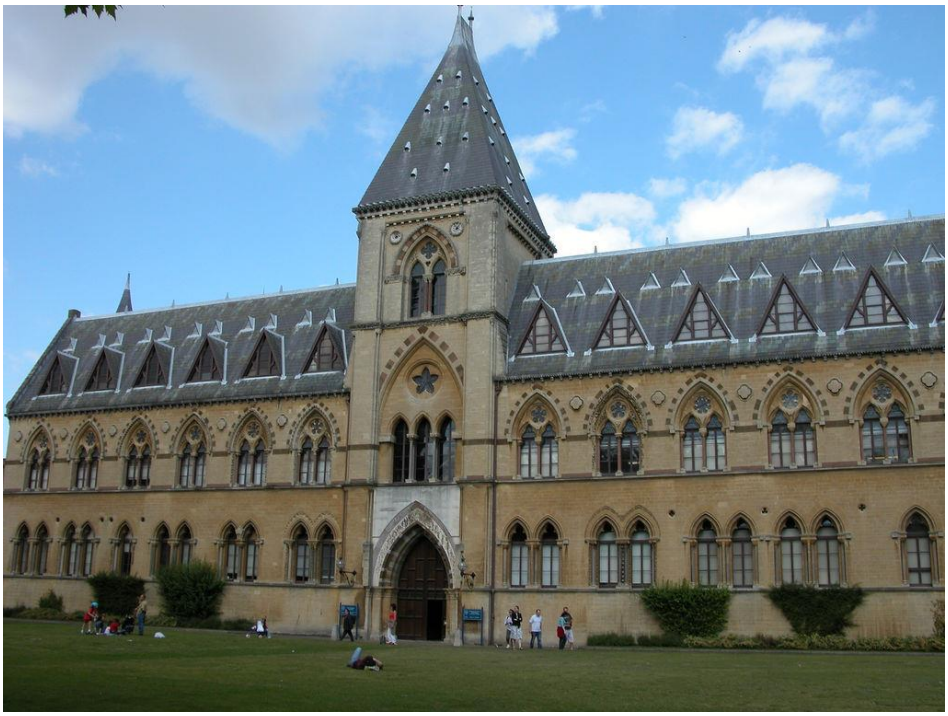

Figura 4.38: Query pitt_rivers_2

5 Experimentos realizados

5.1 Entrenamiento del módulo Regional Attention Network

5.1.1 Introducción y dataset

Con el objetivo de poder identificar una montaña en una imagen de entrada, se va a entrenar un módulo encargado en esa tarea (similar al que se mostró anteriormente con las imágenes de edificios de Oxford). Se va a intentar comprobar que, usando el módulo Regional Attention Network, la precisión de la red mejora con respecto a la red sin dicho módulo.

El dataset elegido consta de 10000 imágenes, de las cuales 5000 pertenecen a la categoría de montañas (la clase 1) y las otras pertenecen a cualquier cosa que no sea una montaña (la clase 0). Esta selección se ha realizado por medio de la web Flickr, y se compone de imágenes sin Copyright.

Se puede afirmar que se está ante un problema binario con clases balanceadas. Se podría pensar que hay un problema claro, ya que las imágenes son de diferentes tamaños. Sin embargo, esto no es un inconveniente, porque se realiza un preprocesamiento de imágenes en el cual se modifican todas ellas a una dimensión máxima de 1024x1024. De esta manera, la escala S de R-MAC anteriormente explicada equivale de S=5, haciendo el proceso exacto.

Se divide aleatoriamente el total del dataset en un set de entrenamiento (90% del total) y en un set de validación (10% del total). La estructura seguida para crear este dataset está basada en la conocida database ImageNet. Después de entrenar con el set de train y de validar, se almacenará el modelo creado para poder usarlo en la predicción de las alturas.

5.1.2 Desarrollo

La red usada está basada en el módulo Resnet101 (para tratar de obtener una buena calidad del mapa de características), pero se ha modificado. Primero, se ha cambiado la capa final, una fully connected layer, que antes estaba preparada para predecir mil clases, para que ahora prediga solamente dos clases (montaña y no montaña). Además, se ha introducido entre la salida de la red convolucional CNN y la entrada de la capa final los módulos R-MAC y Context-Aware Regional Attention (ver *Figura 4.31*) para mejorar la precisión.

En cada iteración de la fase train se han introducido 4 imágenes aleatorias del set de entrenamiento, de ambas clases indistintamente. Además, se ha generado una serie de checkpoints para almacenar el estado actual de la red cada 150 iteraciones. Por otro lado, en cada iteración de validación se han introducido 3 imágenes, realizándose 100 iteraciones en total. En esta parte los parámetros de la red no continúan con el entrenamiento y se quedan congelados. Cuando salen de él vuelven a estar activados. El proceso se repite una y otra vez hasta que la precisión se estabiliza. La red, evidentemente, intentará devolver 1 si se encuentra ante una foto de una montaña y 0 si no se da ese caso.

5.1.3 Resultados

Tras el entrenamiento con el módulo y sin él, los resultados fueron tal y como se esperaba. Como se puede ver en la *Figura 5.1*, aunque no son unos resultados demasiado superiores (sobre todo en validación), sí que hay una diferencia de precisión aceptable. Eso demuestra que, efectivamente, mediante el uso del módulo específico de detección de montañas, la red consigue detectar mejor las montañas que sin él.

	Train	**Validación**
Con Regional Attention Network	0.89	0.863
Sin Regional Attention Network	0.823	0.85

Figura 5.1: Resultados del entrenamiento del módulo Regional Attention Network

Para mostrar lo que "ve" nuestra red, se van a exponer varios ejemplos de la red con el módulo entrenado. Primero se verá la imagen original y después la imagen de pesos. Se han escogido imágenes "complicadas" para la red, donde el monte está esquinado, o detrás de personas, por ejemplo. Esto es así para demostrar que la red busca la figura del monte, aunque tenga distracciones de por medio. Si colocáramos imágenes sencillas de montañas en el centro de la imagen, el mapa de calor estaría siempre colocado en el centro y no podríamos ver correctamente el funcionamiento.

Figura 5.2:Ejemplo 1 del módulo

Figura 5.3: Ejemplo 3 del módulo

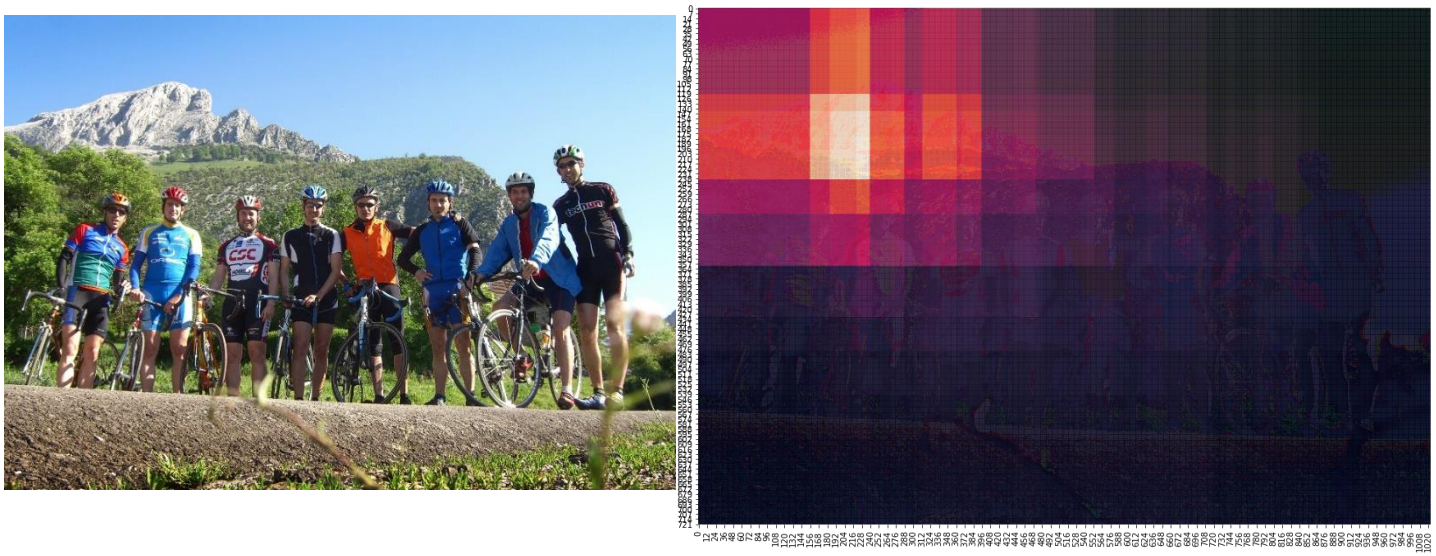

Figura 5.4: Ejemplo 2 del módulo

En el primer ejemplo (*Figura 5.2*) se ve claramente cómo la red "aparta" de su predicción a los dos hombres que se hallan en medio, detectando que no son parte de la montaña. Es muy interesante, ya que se demuestra que la red funciona como se le requería. En el segundo ejemplo (*Figura 5.3*), en la que la montaña esta esquinada arriba a la izquierda, también se puede observar cómo la red dirige toda su atención a esa zona, obviando prácticamente el resto de la foto (que es justamente lo que se desea). Por último, en la *Figura 5.4*, en el que las montañas se encuentran centradas en el fondo la imagen, también podemos considerar los resultados positivos, ya que prácticamente solo se enfoca en la cordillera que nos interesa.

En definitiva, se puede afirmar que este apartado de experimentos ha sido exitoso a grandes rasgos. Es cierto que la diferencia de precisión entre las pruebas con Regional Attention Network y sin él no es demasiado amplia, pero se puede considerar que efectivamente es mejor con el módulo experto.

5.2 Predicción de la altura y resultados finales

5.2.1 Introducción y dataset

Por último, se van a mostrar los resultados obtenidos en la fase de predicción de la altura de la montaña. Para ello hay que explicar previamente una serie de cosas. Primero hay que comentar como está formado el dataset elegido. Después, cómo se ha conseguido a partir de ese dataset las imágenes a predecir y el método de cálculo de las alturas correspondientes. Sabiendo esto, ya se podrán analizar los resultados obtenidos.

Como dataset se han escogido 5000 imágenes de un total de unos 1500 montes. Cada monte tiene su altura correspondiente almacenadas en un archivo .csv. Por otro lado, se han creado veinticuatro "queries", que son las veinticuatro imágenes de montañas del dataset a las que se les va a predecir la altura. Con esas veinticuatro predicciones se calculará el error del sistema con el método MSE (explicado anteriormente). Ahora bien. ¿cómo se predicen las alturas?. De cada una de las queries obtenidas, la red va a ordenar el total de las imágenes del conjunto (las 5000) según se "parezca" más o menos a la query en sí. Por ejemplo, se tiene la imagen de la *Figura 5.5*, una de las queries.

Figura 5.5: Query número 1 - Agassiz Peak05

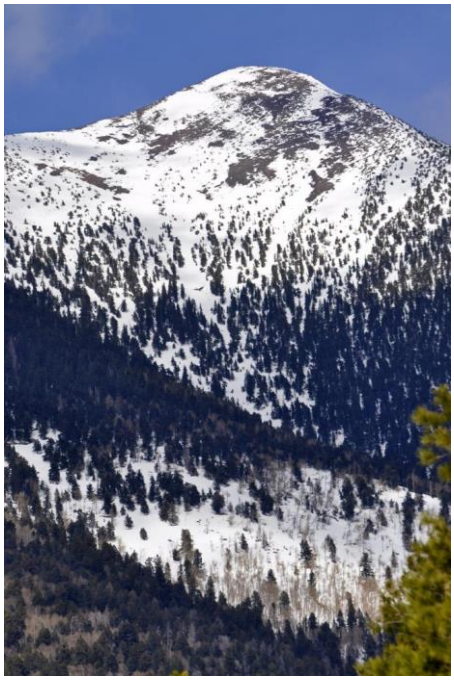

Figura 5.6: Primer resultado de la query número 1 - Agassiz Peak03

Se trata de la montaña Agassiz Peak, con una altura de 3,767 metros. Según la red, la imagen que más se parece a la *Figura 5.5* es la mostrada en la *Figura 5.6*. En este caso, la red funciona correctamente, ya que detecta que la imagen más parecida a la montaña es otra foto de esta misma montaña, por lo que es un éxito. Si en vez de detectar otra montaña, el error de esa predicción se calcularía de la siguiente manera:

$$Error = (altura\_query\ - altura\_predicción)^2$$

Siendo $altura\_query$ la altura de la imagen query y $altura\_predicción$ la altura de la imagen que se ha predicho. Por supuesto, este experimento no se debe realizar solamente con la primera predicción, ya que no se presentaría una visión completa del experimento. Por ello, se van a realizar los cálculos de las 24 queries con los 3, 5 y 10 primeros resultados. Además, para cada experimento (con 3, 5 y 10 imágenes) se va a calcular la altura de tres maneras: haciendo la media, la media ponderada y el máximo. Por ejemplo, tomando la query número uno, se calculará su altura cogiendo los 3 resultados más parecidos y se realizará la media, la media ponderada y el máximo para obtener una altura aproximada. Después se realizará con los 5 y con los 10 primeros resultados, siguiendo el mismo proceso y así sucesivamente con el resto de queries.

También, antes de pasar a los resultados, hay que comentar el uso de "distractors" y qué redes se han utilizado. Un distractor es un elemento que se introduce en un dataset para tratar de confundir al sistema. En este caso, un distractor es una imagen dentro del conjunto de montañas que no es una montaña. Si la red, en el proceso de detectar las imágenes más parecidas a las queries se equivoca e introduce dentro del top un distractor, significa que ha confundido y que, por lo tanto, el error será *a priori* mayor. Si en la experimentación se encuentra un distractor a la hora de predecir las alturas de las montañas, se ha decidido que el valor que se asignará será un valor negativo elevado, para ampliar el error de la red.

Se van a realizar los experimentos sobre cuatro redes: dos de ellas van a tener los módulos de las regiones y dos de ellas no. Las redes poseedoras de regiones van a ser, por un lado, la que hemos entrenado en el apartado *5.1: Entrenamiento del módulo Regional Attention Network* con regiones, y, por el otro, la red Resnet101 original con regiones. En las tablas de resultados se denominarán *Con regiones* y *Resnet101* respectivamente. Después, las redes sin regiones van a ser la red entrenada en el apartado mencionado anteriormente, pero sin regiones, y la red Resnet101 sin regiones. Se llamarán *Sin regiones* y *Resnet101 sin regiones*. Se espera, por lo tanto, que las mejores redes sean las que tienen regiones, y entre esas dos, que la que mejor funcionamiento de sea nuestra red.

Por último, después de realizar los experimentos con las fotos de montañas normales, se ha decidido probar con unas queries que se han llamado **Queries complicadas**. Estas son, para que se entienda bien, fotos de montañas con distractors en la propia imagen. Por ejemplo, en la *Figura 5.7* se puede ver una imagen del K2 en la que hay un helicóptero en frente. Todas estas queries tienen algo para intentar "confundir" al sistema: una persona o varias, casas, vehículos, etc. Con todo esto aclarado, se va a pasar a mostrar los resultados.

Figura 5.7: Ejemplo de query complicada

5.2.2 Experimentos con las imágenes de la montaña

En este subapartado se van a exponer los resultados de los experimentos realizados con las imágenes de las montañas. El objetivo es, por tanto, que la red detecte para toda montaña query el resto de las fotos de esa misma montaña en los primeros lugares del ranking. Si eso ocurre, el error será bajo, mientras que si las montañas escogidas por la red no se parecen a la prueba el error será mayor.

Por ejemplo, para la query *Agassiz Peak05*, el ranking obtenido por la red construida en este proyecto es el siguiente:

1- Agassiz Peak03
2- Agassiz Peak01
3- Agassiz Peak04
4- Humphreys Peak02

Esto implica que el funcionamiento para esta query es correcto. Sin embargo, para por ejemplo la query *Petrechema03*, se ha obtenido:

1- Pointe de Merdassier03
2- Malika Parbat00
3- Pigne d'Arolla05
4- Corno Grande04

En este caso, la red no ha sido capaz de conseguir colocar al resto de imágenes del monte Petrechema en los primeros lugares, por lo que el error aumentará.

Primero, en los resultados sin distractors, la red no evaluará elementos que no son montañas, por lo que no podrá confundirse, mientras que en el siguiente apartado puede fallar y predecir un elemento que no es un monte con una query.

5.2.2.1 Sin distractors

- Media

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	695.34	**667.20**	739.42	732.75
5	806.89	773.50	823.04	836.22
10	871.25	811.14	954.98	932.29

- Media ponderada

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	538.83	**486.20**	557.76	529.36
5	551.56	504.78	555.60	550.72
10	703.24	660.11	742.542	735.84

- Máximo

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	1231.55	**1175.33**	1281.33	1306.15
5	1622.987	1625.54	1640.77	1685.49
10	2435.84	2407.23	2386.66	2332.23

5.2.2.2 Con distractors

- Media

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	695.34	**667.20**	739.42	732.75
5	806.89	773.50	1172.48	1176.68
10	871.25	964.57	1652.03	1608.33

- Media ponderada

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	538.83	**486.20**	557.76	529.36
5	551.56	504.78	660.30	652.72
10	703.24	686.03	1107.58	932.27

- Máximo

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	1231.55	**1175.33**	1281.33	1306.15
5	1622.98	1625.54	1640.77	1685.49
10	2435.84	2407.23	2386.66	2332.23

Se ha podido ver que los mejores resultados vienen siempre de la mano de la red resnet101, lo cual es algo que no se suponía. Al haber entrenado un modelo específico para la detección de montañas, se esperaba que esta fuera la que devolviera un mayor acierto.

Por otro lado, haciendo una comparativa entre la red del proyecto con regiones-sin regiones y de Resnet101 con regiones-sin regiones, los resultados obtenidos por las redes convolucionales con ese módulo son mejores que las redes sin ese mismo módulo, lo que confirma lo que ya se había supuesto, que se da una mejora significativa en un sistema base simplemente introduciendo Regional Attention.

5.2.3 Experimentos sin las imágenes de la montaña

Mientras que en el anterior apartado de resultados se tenían dentro del dataset otras imágenes de las montañas query (de una montaña query se tenían otras fotos desde diferentes perspectivas, tamaños,etc.), en este caso se van a realizar los mismos ejemplos sin esas fotos. Por ejemplo, para el caso de la query *Agassiz Peak05*, el ranking nos mostraba como la red detectaba como casos más parecidos a imágenes del propio monte. Sin embargo, ahora, al eliminar esas fotografías, el ranking obtenido quedaría de la siguiente manera:

1- Humphreys Peak02
2- Tuc de Salana01
3- Colima01
4- Kintla Peak02

Esto evidentemente hace que la precisión de la red se vea reducida, ya que en ningún caso el sistema va a conseguir detectar la propia montaña query (lo que supondría un error igual a 0), ya que no hay fotografías seleccionables de ese monte. Como antes, se van a realizar los experimentos con distractors y sin distractors.

5.2.3.1 Sin distractors

- Media

Top / Red	**Con regiones**	**Resnet101**	**Sin regiones**	**Resnet101 sin regiones**
3	765.82	**739.36**	809.51	826.58
5	871.97	799.65	883.88	886.30
10	920.96	887.92	993.57	963.03

- Media ponderada

Top / Red	**Con regiones**	**Resnet101**	**Sin regiones**	**Resnet101 sin regiones**
3	574.84	**550.09**	625.49	608.28
5	605.03	546.57	606.44	607.85
10	748.63	701.44	803.76	799.47

- Máximo

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	1191.27	**1108.72**	1329.27	1337.67
5	1709.45	1583.55	1640.77	1786.69
10	2480.33	2451.88	2424.20	2405.13

5.2.3.2 Con distractors

- Media

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	765.82	**739.36**	809.51	826.58
5	871.97	799.65	1243.52	1245.94
10	920.96	1041.34	1691.17	1815.41

- Media ponderada

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	574.84	**550.09**	625.49	608.28
5	605.03	546.57	766.75	766.67
10	748.63	727.36	1273.15	1065.84

- Máximo

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	1191.27	**1108.72**	1329.27	1337.67
5	1709.45	1583.55	1793.08	1786.69
10	2480.33	2451.88	2424.20	2405.13

Se repiten los patrones del anterior apartado. La mejor red sigue siendo la red Resnet101 con regiones, seguida de la generada en este proyecto, y los sistemas con regiones aciertan más que esos mismos sistemas sin regiones. Lo cierto es que hay un pequeño aumento en el error (de unos 100-200 metros con respecto a las pruebas con las imágenes) debido a la eliminación de las imágenes de las montañas query, lo cual no se puede considerar como un incremento significativo y puede indicar que los modelos son robustos como se pretendía.

5.2.4 Queries complicadas

Por último, se van a realizar los mismos casos, pero con las llamadas "queries complicadas". Una query complicada es una imagen de una montaña que contiene elementos que pueden hacer equivocarse a la red. Un ejemplo se puede ver en la *Figura 5.7*, siendo el elemento distractor el helicóptero. El error conseguido es bastante más alto que en las anteriores queries, debido a la dificultad de estas pruebas. Por ejemplo, para la query *Grand Cornier 10*, como se puede ver en la *Figura 5.8*, la red se equivoca con la *Figura 5.9* asignándola como una de las imágenes más parecidas dentro del dataset a la prueba, lo cual se puede quizás explicar debido a que en la foto aparecen dos personas, con nieve de fondo, etc. Es decir, tienen características similares entre sí. Con esto, se demuestra el porqué de la dificultad de estas queries.

Figura 5.8: Query complicada Grand Cornier 10

Figura 5.9: Primer resultado de la búsqueda para la query Grand Cornier 10

5.2.4.1 Sin distractors

- Media

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	998.0	1157.25	958.41	**954.77**
5	1045.56	1277.06	1062.41	1134.80
10	1211.09	1269.41	1295.52	1360.53

- Media ponderada

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	759.10	844.14	**686.48**	755.45
5	743.90	881.01	699.54	755.14
10	973.76	1104.61	1022.28	1051.03

- Máximo

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	1599.33	1766.08	**1531.0**	1642.95
5	2245.29	1827.20	2092.83	2139.37
10	2546.5	2329.12	2644.37	2690.54

5.2.4.2 Con distractors

- Media

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	1297.18	**1221.5**	1660.16	1684.9
5	1272.88	1406.14	1693.0	1771.09
10	1308.51	1448.75	1946.79	1776.3875

- Media ponderada

Top / Red	Con regiones	Resnet101	Sin regiones	Resnet101 sin regiones
3	983.56	**820.00**	1236.94	1306.82
5	931.21	873.72	1234.26	1287.33
10	1113.55	1189.75	1630.63	1526.77

- Máximo

Top / Red	**Con regiones**	**Resnet101**	**Sin regiones**	**Resnet101 sin regiones**
3	1466.0	1766.08	**1367.91**	1479.875
5	2244.58	2302.33	2045.83	2092.375
10	2510.45	2558.79	2511.83	2567.33

Curiosamente, en las imágenes sin distractors parece que se "invierte" el funcionamiento. Las redes con regiones predicen peor las alturas que las que si emplean regiones. Con la introducción de distractors, el funcionamiento vuelve a su estado normal, consiguiéndose mucho mejores resultados con el sistema entrenado en este proyecto y con Resnet101. Esto recupera el sentido que tenían todos los resultados anteriores, donde las regiones hacían obtener errores menores que a sus análogos más simples.

5.2.5 Conclusiones

Obtenidos ya los resultados, se puede afirmar que no son del todo satisfactorios, y que en parte se pueden considerar como extraños. Se van a enumerar algunas conclusiones interesantes conseguidas:

1. El mejor método para obtener la altura de una montaña de entrada es la media ponderada. Esto tiene sentido, ya que, al asignar mayor peso a las primeras montañas detectadas, y suponiendo que estas son las más parecidas al monte query, es normal que el error sea menor que con la media normal.

2. Por otro lado, el peor método de detección es, sin duda, cogiendo el máximo. También tiene sentido, ya que prioriza la máxima altura sin tener en cuenta el resto de los montes en del ranking. El error es siempre mucho más amplio que con los otros métodos.

3. Es curioso como cuando se escogen más montes para realizar la predicción de la altura, el error aumenta. Es decir, como regla general cuantos más elementos del ranking se escogen, mayor es el error. Se puede encontrar sentido también a esto, ya que, al coger un número mayor de montes, los cuales cada vez van a ser menos parecidos a la query, su altura va a ser cada vez más diferente a dicha query (en teoría).

4. Con respecto a los distractors, es también evidente que la red funciona mejor sin ellos que con ellos, ya que confunden a la red ampliando su error. Es también remarcable que la red más robusta (menos propensa a elegir un distractor como montaña) es la red que se ha entrenado en este trabajo. Esto se puede considerar un éxito, ya que en los experimentos “no complicados” (es decir, obviando las queries complicadas), nuestro modelo es el único que no detecta ningún distractor en las pruebas. El resto de ellas sí que en algunas imágenes las confunden con imágenes de no montes.

5. Sin embargo, una de las suposiciones que se habían realizado no ha resultado ser cierta. Se pensaba que al haber entrenado una red específicamente para la detección de montañas, esta funcionaría mejor que la red Resnet101, que es de carácter más generalista. Por el contrario, aunque los resultados son bastante parecidos entre estos dos modelos con el sistema de regiones activo, como norma general la red externa funciona de forma más precisa.

Con respecto a este último punto, es cierto que en las queries complicadas el comportamiento de los módulos se iguala en cuanto a error, pero no se puede afirmar que haya una diferencia suficiente para sacar conclusiones. Este funcionamiento extraño quizás se puede explicar debido a la gran precisión que tiene Resnet101, fruto de un entrenamiento muy exhaustivo (la red está entrenada con más de un millón de imágenes [13]). Por lo tanto, aunque se podía pensar que funcionaría mejor un modelo especialista en montañas que uno preparado para detectar hasta 1000 clases, como es Resnet, lo cierto es que esta última ha conseguido adaptarse a la predicción de 2 montes.

Como ejemplo de lo expuesto con respecto al error se va a visualizar las gráficas de los resultados con las imágenes de las montañas (el apartado 5.2.2 de este proyecto). Se va a proceder de esta manera ya que al comportarse todos los casos de manera similar (todos siguen una misma distribución, aproximadamente) este ejemplo es el más sencillo. Se podrá comparar como con el mismo método de predicción (la media), existen diferencias entre las pruebas sin distractors y con ellos.

En la *Figura 5.10* se puede ver como más o menos el error esta acotado entre 600 y 1000 metros para cada una de las redes usadas. Pese a que como se ha comentado anteriormente, el mejor sistema es el Resnet101 con regiones, se puede visualizar que los modelos son bastante similares.

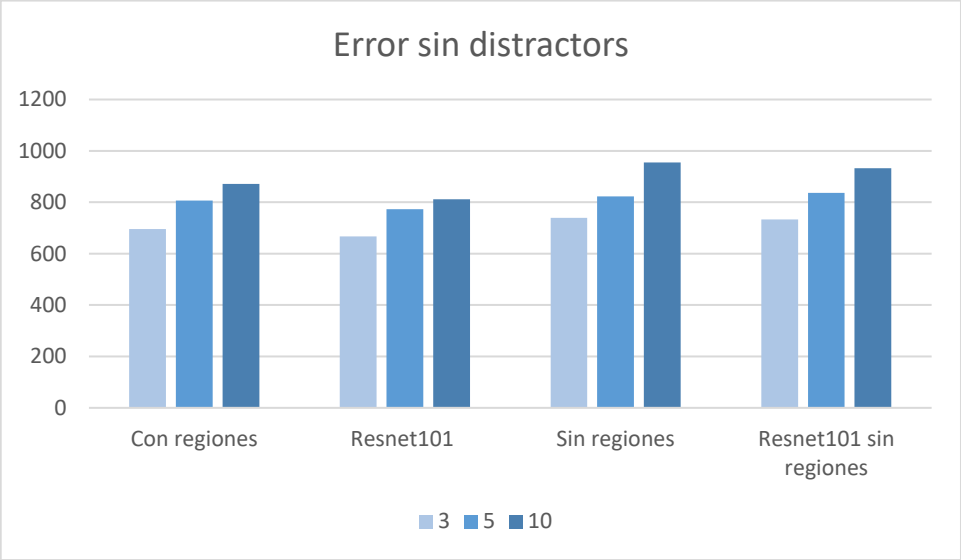

Figura 5.10: Resultados sin distractors

Sin embargo, en la *Figura 5.11*, que son los mismos experimentos con distractors, hay una clara diferencia. Se puede ver cómo mientras las redes con regiones se mantienen más o menos estables, las dos sin la mejora de *Regional Attention Network* empeoran en gran medida su rendimiento. Por lo tanto, se puede afirmar que claramente, al usar este paradigma se mejora la precisión de la red neuronal usada.

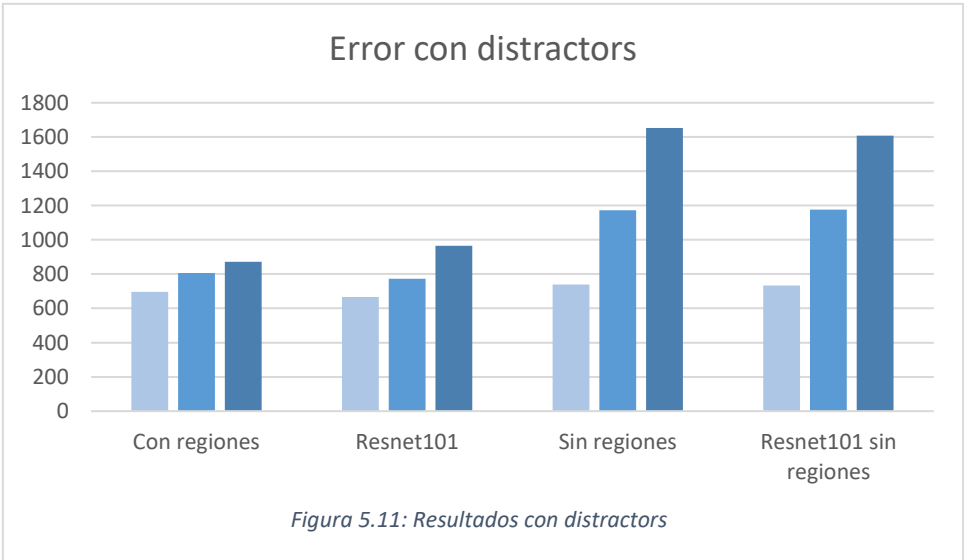

Figura 5.11: Resultados con distractors

Pensando ahora en cómo mejorar el rendimiento de la red de este trabajo para poder competir con Resnet101, quizás se podrían ampliar los datasets de entrenamiento de ambas fases (la parte de adiestramiento del módulo Regional Attention Network y la parte de predicción de alturas de montañas). Esto no se ha podido realizar debido a un límite de memoria, ya que el conjunto de datasets tiene un tamaño de 50GB, pero si se deseara mejorar este trabajo ese sería un punto importante.

Por otro lado, se podía haber elegido un número de fotografías query para realizar experimentos, pero esto también es complicado por culpa de lo costoso que ha resultado ser el entrenamiento y las pruebas (con tiempos de hasta 5-6 horas).

Aun con todo, los resultados son bastante aceptables. La altura de las montañas en este dataset va desde pequeños montes de 100-200 metros hasta los ochomiles. Por lo tanto, se puede pensar que un error de 500 o 600 metros es poco, en comparación con el gran intervalo y posibilidades de fallo que hay. En una situación en la que las imágenes a detectar son del monte en sí, sin ningún distractor integrado, y usando la media ponderada, se ha demostrado que el trabajo realizado es muy bueno y bastante preciso.

Bibliografía

[1] P. Isola, . J.-Y. Zhu, . T. Zhou y . A. A. Efros, «Image-to-Image Translation with Conditional Adversarial Networks,» 21 Noviembre 2016. [En línea]. Available: https://arxiv.org/pdf/1611.07004.pdf. [Último acceso: 7 Mayo 2021].

[2] «'Machine learning': ¿qué es y cómo funciona?,» 8 Noviembre 2019. [En línea]. Available: https://www.bbva.com/es/machine-learning-que-es-y-como-funciona/. [Último acceso: 6 Mayo 2021].

[3] D. Daniel Cox y T. Dean, «Neural Networks and Neuroscience-Inspired Computer Vision,» 22 Septiembre 2014. [En línea]. Available: https://www.cell.com/action/showPdf?pii=S0960-9822%2814%2901039-2. [Último acceso: 25 Abril 2021].

[4] G. Albeleira, «¿Cuántos tipos de neuronas que existen?,» Hablemos de Neurociencia , 6 Noviembre 2017. [En línea]. Available: https://hablemosdeneurociencia.com/tipos-neuronas/. [Último acceso: 26 Abril 2021].

[5] M. J. Vega Pedrero, «Conexión neuronal: el potencial de acción,» Hablemos de Neurociencia , 7 Noviembre 2016. [En línea]. Available: https://hablemosdeneurociencia.com/conexion-neuronal-potencial-accion/. [Último acceso: 26 Abril 2021].

[6] F. Izaurieta y C. Saavedra, «Redes neuronales artificiales,» 2000. [En línea]. Available: https://d1wqtxts1xzle7.cloudfront.net/36957207/Redes_neuronales.pdf?1426217567=&response-content-disposition=inline%3B+filename%3DRedes_Neuronales_Artificiales.pdf&Expires=1619438787&Signature=MsSoH0OxCFIjZSRRGSb4Wms-pM3iBQjpo6GYLE1NobDC-z2iLuq9ZJ~PTZADuY. [Último acceso: 26 Abril 2021].

[7] F. Ramírez, «Historia de la IA: Frank Rosenblatt y el Mark I Perceptrón, el primer ordenador fabricado específicamente para crear redes neuronales en 1957,» Telefonica, 20 Julio 2018. [En línea]. Available: https://empresas.blogthinkbig.com/historia-de-la-ia-frank-rosenblatt-y-e/. [Último acceso: 25 Marzo 2021].

[8] F. Ramírez, «Las matemáticas del Machine Learning: Funciones de activación,» Telefónica, 4 Junio 2020. [En línea]. Available: https://empresas.blogthinkbig.com/las-matematicas-del-machine-learning-funciones-de-activacion/. [Último acceso: 29 Marzo 2021].

[9] S. I. Serengil, «Step Function as a Neural Network Activation Function,» aa, 15 Mayo 2017. [En línea]. Available: https://sefiks.com/2017/05/15/step-function-as-a-neural-network-activation-function/. [Último acceso: 29 Marzo 2021].

[10] D. J. Matich, «Redes Neuronales: Conceptos Básicos y Aplicaciones.,» Universidad Tecnológica Nacional, Marzo 2001. [En línea]. Available: https://www.frro.utn.edu.ar/repositorio/catedras/quimica/5_anio/orientadora1/monografias/matich-redesneuronales.pdf. [Último acceso: 27 Marzo 2021].

[11] S. I. Serengil, «The Math Behind Neural Networks Learning with Backpropagation,» 21 Enero 2017. [En línea]. Available: https://sefiks.com/2017/01/21/the-math-behind-backpropagation/. [Último acceso: 10 Abril 2021].

[12] J. Kim y S.-E. Yoon, «Regional Attention Based Deep Feature for,» 2018. [En línea]. Available: http://sglab.kaist.ac.kr/RegionalAttention/preprint.pdf. [Último acceso: 2 Abril 2021].

[13] K. He, X. Zhang, S. Ren y J. Sun, «Deep Residual Learning for Image Recognition,» 10 Diciembre 2015. [En línea]. Available: https://arxiv.org/pdf/1512.03385.pdf. [Último acceso: 1 Mayo 2021].

[14] M. A. Nielsen, «Neural Networks and Deep Learning,» Determination Press, Diciembre 2019. [En línea]. Available: http://neuralnetworksanddeeplearning.com/chap1.html. [Último acceso: 26 Marzo 2021].

[15] A. Rubiales, «Explicación de las Funciones de activación en Redes Neuronales y práctica con Python.,» Medium, 16 Octubre 2020. [En línea]. Available: https://rubialesalberto.medium.com/explicaci%C3%B3n-funciones-de-activaci%C3%B3n-y-pr%C3%A1ctica-con-python-5807085c6ed3. [Último acceso: 29 Marzo 2021].

[16] C. Pere, «What is activation function ?,» Medium, 9 Junio 2020. [En línea]. Available: https://towardsdatascience.com/what-is-activation-function-1464a629cdca. [Último acceso: 29 Marzo 2021].

[17] S. Saha, «A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way,» Towards Data Science, 15 Diciembre 2018. [En línea]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. [Último acceso: 4 Abril 2021].

[18] S. Silva y E. Freire, «Intro a las redes neuronales convolucionales,» Bootcamp AI, 23 Noviembre 2019. [En línea]. Available: https://bootcampai.medium.com/redes-neuronales-convolucionales-5e0ce960caf8. [Último acceso: 5 Abril 2021].

[19] S. Albawi, T. Abed Mohammed y S. Al-Zawi, «Understanding of a Convolutional Neural Network,» 2017. [En línea]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8308186&casa_token=BuS2McpIpeYAAAAA:RpSUwZNhOGwyD3_xv4EZjKezZIolZwA1BG3QyhOm7ttFWVfuAWjPQaUFiVFLuGGqmNkSH5wIug. [Último acceso: 25 Abril 2021].

[20] A. Afshine y A. Shervine , «Convolutional Neural Networks cheatsheet,» Stanford, 26 Noviembre 2018. [En línea]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks. [Último acceso: 25 Abril 2021].

[21] C. Versloot , «What is padding in a neural network?,» Machine Curv, 7 Febrero 2020. [En línea]. Available: https://www.machinecurve.com/index.php/2020/02/07/what-is-padding-in-a-neural-network/. [Último acceso: 25 Abril 2021].

[22] M. Campos Soberanis, «Inspiración biológica de las redes neuronales artificiales,» 13 Febrero 2018. [En línea]. Available: https://medium.com/soldai/inspiraci%C3%B3n-

biol%C3%B3gica-de-las-redes-neuronales-artificiales-9af7d7b906a. [Último acceso: 26 Abril 2021].