Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Un estudio sobre Preprocesamiento y Mecanismos de Atención en Redes Neuronales Convolucionales para Clasificación de Lesiones de Piel

Tesis de Licenciatura en Ciencias de la Computación

Miguel Nehmad Alche

Director: Daniel Acevedo

Buenos Aires, 2020

UN ESTUDIO SOBRE PREPROCESAMIENTO Y MECANISMOS DE ATENCIÓN EN REDES NEURONALES CONVOLUCIONALES PARA CLASIFICACIÓN DE LESIONES DE PIEL

El melanoma es una forma de cáncer de piel muy peligrosa para la salud. El diagnóstico temprano del mismo es crucial para aumentar las posibilidades de su cura. En base a ello se puede utilizar algoritmos de visión por computadora para analizar imágenes dermoscópicas de las lesiones en la piel y clasificar si se trata de lesiones benignas o malignas. En este trabajo estudiamos diversas mejoras que se le pueden aplicar a las redes convolucionales: desde métodos de preprocesamiento sobre el dataset hasta mecanismos de attention sobre la red en sí. Descubrimos que el mecanismo de attention residual learning mejora la performance de la red EfficientNet, así también como utilizar algoritmos de preprocesamiento que remueven las frecuencias bajas de las imágenes del dataset.

Palabras claves: Aprendizaje Profundo, Visión por Computadora, Cáncer de piel, Attention, Redes Neuronales Convolucionales.

A STUDY ON PREPROCESSING AND ATTENTION MECHANISMS FOR CONVOLUTIONAL NEURAL NETWORKS TO DETECT SKIN CANCER IN DERMOSCOPIC IMAGES

Melanoma is a very dangerous form of skin cancer. Early diagnosis of it is crucial to increase the chances of its cure. Based on this, computer vision algorithms can be used to analyze dermoscopic images of skin lesions and decide if these correspond to benign or malignant tumors. In this work we study various improvements that can be applied to convolutional networks: ranging from preprocessing methods on the dataset up to attention mechanisms on the network itself. We discovered that the attention residual learning mechanism improves the performance of the EfficientNet network, as well as using preprocessing algorithms that remove low frequencies from the dataset images.

Keywords: Deep Learning, Computer Vision, Skin Cancer, Attention, Convolutional Neural Networks.

AGRADECIMIENTOS

Agradezco a la Universidad de Buenos Aires cuya inmensa generosidad me permitió estudiar y formarme con profesores geniales y amigos de hierro. Cuando tardaba en encontrar la carrera que era para mi, la UBA me permitió explorar las más diversas áreas; cuando por fin encontré mi lugar en la Facultad de Ciencias Exactas y Naturales, la UBA me rodeó con los mejores profesores.

Agradezco también a todos los profesores del Departamento de Computación que me formaron a lo largo de mi carrera. Especialmente a Daniel Acevedo que me acompaño en este proceso. Daniel tiene una combinación de sabiduría y paciencia que debe ser única en el mundo. Gracias Daniel por hacer de este proceso algo inmensamente interesante y también divertido.

Agradezco a mis amigos de la facultad que me acompañaron durante la carrera. A Cata, la creatividad personificada, fabrica de memes y la mejor compañera de banco del mundo; A Agus por las mejores sobremesas en el comedor del pabellón 1; A Facu y a Philip por el tenis y el fútbol necesarios; A Jess, la reina de assembler y de los chistes ácidos; A Facu por su increíble inteligencia y capacidad para enseñar; A Nacho por ponerle humor a cada centímetro de la vida; A Nico San Martín por tantas charlas interesantes en colectivos eternos; A Jony por tantos viajes y el compañerismo en todo momento; A Tom Grosso por iniciarme en el mundo de Apple; A Tomas Alonso por las carcajadas continuas; A Damián y a Gabi por ser uno más brillante que el otro y enseñarme en el proceso; a Luciano por compartir la pasión por las redes neuronales; y al todo el resto Julian, Eric, y compañía por hacer de nuestro grupo, uno hermoso. Gracias chiques, gracias por hacerme reír tanto y tan fuerte tantos años. Sin ustedes abandonaba en álgebra de primer año.

Agradezco a mi amor y compañera Laura Burak. Gracias hermosa por todo. Por tanto apoyo, tanto aguante y tanto amor infinito e incondicional. Por hacer de nuestras carreras un continuo de festejos tras festejos, sonrisas tras sonrisas.

Agradezco a mi amigo y hermano, Fabio Zon. Lo único que no compartimos es el apellido. Todo lo demás, juegos, inventos, canciones, viajes, proyectos, llamadas y sueños siempre lo hicimos juntos. Gracias Fabs, no se hace una carrera de ciencias sin antes soñar con la magia. Agradezco también a Edy Zon, por enseñarme a tirarme de cabeza a la pileta y más que nada a la vida; y a Ruth Nasielski por darme una segunda casa.

Agradezco también a mis amigos. Agradezco a Lean, por compartir mi pasión por la música y los emprendimientos. A Lucas por su compartir nuestro fanatismo con las startups y las ideas disruptivas. A Fei por una amistad que resiste cualquier paso del tiempo. Agradezco a *los clavas*: Fede, Nico, Pedro, Puchi, Rozwa, Javi, Santi, Cami V, Pancha, Clara, Nicky, Cami Scher, Luli por tantos años de amistad y tantas anécdotas hermosas.

Agradezco a mi familia: Leo, Marta, María, Flor, Fede, Raquel, Sasha, Dalit y Sergio, entre tantos. Gracias por hacer del mundo un lugar tan cálido.

Finalmente, agradezco a mi mama Claudia Alché, quien siempre me dio todo en infinita abundancia. Gracias Ma, por ser una guerrera de hierro frente a la vida, más valiente que el destino, por dar todo el amor que un hijo necesita por duplicado.

Índice general

1. INTRODUCCIÓN

1.1. Motivación

De todos las formas de cáncer de piel, el melanoma es una de las que presenta los mayores índices de mortalidad [1]. Es por ello que es de vital importancia que un diagnóstico y tratamiento adecuado del mismo se realicen de la manera más pronta posible.

En vista de tal efecto, diversas técnicas de visión por computadora han surgido con el propósito de detectar de manera temprana este tipo de lesiones de piel. El propósito de estos algoritmos es el de poder acercar al mayor número de personas posible una manera confiable de realizar chequeos periódicos: ya sea mediante una distribución de los mismos en aplicaciones de celular para usuarios finales; como también la creación de herramientas específicas para profesionales de la salud especializados en el tratamiento de la piel.

Sin embargo —no ha de confundirse— estas técnicas no son capaces de dar un diagnóstico 100 % confiable, para lo cual siempre es necesario realizar un estudio histológico.

Esto no quita que los avances tecnológicos que nos depara el futuro no permitan mejorar la técnica hasta que la misma se encuentre a la par de estudios químicos de laboratorio.

Este trabajo es un pequeño paso en esa dirección: una serie de estudios y experimentos para refinar la técnica y acercarla a una meta deseada.

1.2. Objetivos

Las redes neuronales convolucionales han marcado el estado del arte en los últimos años en cuanto a clasificación de imágenes se trata [2].

Es por ello que en el presente trabajo se propone como objetivo experimentar ciertos cambios en las mismas con el fin de comprender mejor su funcionamiento para finalmente mejorar su performance en lo que respecta a clasificación de imágenes de lesiones de piel.

Los experimentos realizados pueden dividirse en dos categorías: los que suceden en una etapa de preprocesamiento y los que suceden dentro de la red en sí.

Dentro de los que suceden en la etapa de preprocesamiento se encuentran: la segmentación del área de la lesión en la imagen; la normalización de colores de las imágenes mediante la aplicación de algoritmos de color constancy; la remoción de variaciones en el tono de las imágenes mediante la aplicación del preprocesamiento de Ben Graham

Dentro de los que suceden en el interior de la red en sí se encuentra el agregado del mecanismo de *attention residual learning* a las redes EfficientNet.

En todos estos experimentos se mide cómo varía el accuracy [1] de la redes al ser aplicados. Los resultados muestran que la segmentación del área de la lesión en las imágenes destruye más información valiosa que el ruido de las mismas que elimina y por tanto empeora la performance.

Respecto de los algoritmos de corrección de color si bien *Shades of Gray* empeora la performance, *Max RGB* logra ciertas leves mejoras.

Finalmente las mejoras significativas suceden mediante la aplicación del método de *Ben Graham* a las redes ResNet como también el agregado del mecanismo de *attention*

[1] Se entiende aquí accuracy como la métrica que se define por el ratio entre casos clasificados correctamente sobre el total de casos analizados

residual learning a las redes EfficientNet. Ambos mecanismos mejoran de manera notoria la performance de cada red respectivamente.

1.3. Trabajo Previo

El estudio de clasificación de lesiones de piel con visión por computadora ha crecido en gran medida en los últimos años. Esto se debe en gran parte al incremento en la capacidad de cómputo de los placas de vídeo, lo cual permitió una explosión en el desarrollo de algoritmos de *Deep Learning* —o aprendizaje profundo— entre los cuales las redes neuronales convolucionales marcan el estado del arte año tras año.

A la vez, la clasificación de lesiones de piel se ha visto beneficiada gracias al trabajo de grandes competencias internacionales como la *International Skin Imaging Collaboration* [3] las cuales ponen a disposición de investigadores de todo el mundo datasets de imágenes dermoscópicas con etiquetas de clasificación asociadas.

Respecto a los algoritmos de clasificación de imágenes, luego del suceso de las redes ResNet [4], se han probado numerosas nuevas mejoras y variaciones. Entre ellas las redes EfficientNet [5] prometen una gran relación entre precisión y costo computacional.

Centrándose en la clasificación de lesiones de piel, trabajos como [6] intentaron combinar el cuerpo de redes de segmentación con redes de clasificación, obteniendo leves mejoras en la precisión.

Por otro lado [7] si bien no utiliza técnicas de redes neuronales, se centra en el preprocesamiento del dataset utilizando algoritmos de *color constancy* para normalizar los tonos de las imágenes que aparecen en los datasets de lesiones de piel.

Respecto a modificaciones a las redes mismas [8] agrega a las redes ResNet una pequeña cantidad de parámetros que permiten realizar un simple pero poderoso mecanismo de attention con bajo costo computacional. A la vez, el trabajo [9] —si bien no se centra en imágenes de lesiones de piel— introduce un mecanismo de attention en el espacio de los canales que parece mejorar significativamente la performance de las redes.

Finalmente técnicas de combinar varios modelos —en lo que se denomina ensemble— fueron estudiadas en [10], sin embargo este tipo de modelos suele incurrir en altos costos computacionales lo cual en ciertos casos se desea evitar (como por ejemplo al querer embeber estos algoritmos en aplicaciones de celular).

2. INTRODUCCIÓN TEÓRICA Y MÉTODOS

2.1. Representación de las imágenes

A lo largo de este trabajo se estudia la manera de clasificar imágenes médicas. Para ello es necesario poder tratar a las imágenes con una representación que sea conveniente desde el punto de vista práctico.

Es por ello que aquí las imágenes se entienden como tensores de 2 dimensiones si la imagen se encuentra en blanco y negro, o de 3 dimensiones si se trata de una imagen a color.

Por simplificación, aquí se entiende a un tensor como un arreglo n-dimensional de números. En el caso de las imágenes en blanco y negro, su representación será un tensor de 2 dimensiones (alto y ancho) donde el valor de cada posición en el tensor estará dada por la intensidad del píxel correspondiente en la imagen, como se puede observar en la figura 2.1.

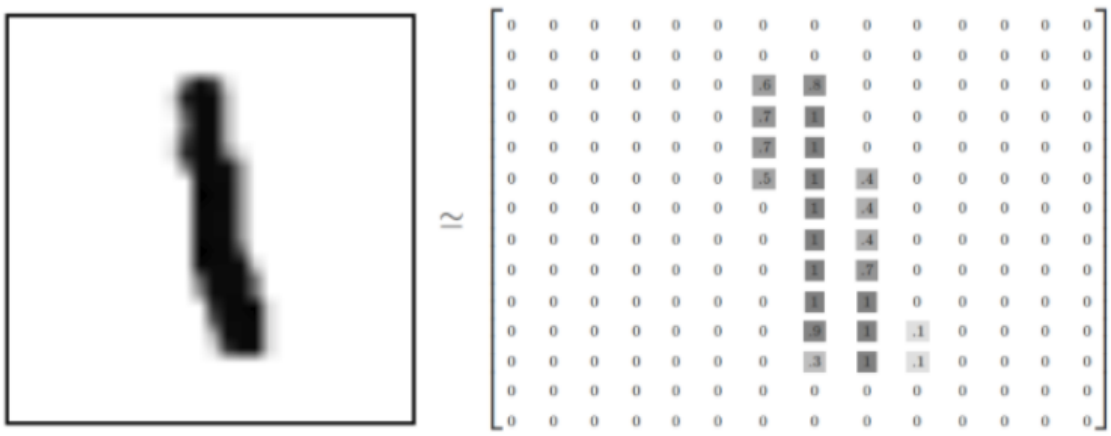

Fig. 2.1: Representación de una imagen como matriz de números.

Similarmente, las imágenes a color se representarán de manera análoga con el único agregado de que en vez de existir un único tensor de dos dimensiones, habrá tres de estos. El motivo es que cada uno de ellos representará la intensidad del píxel correspondiente en su componente roja, verde o azul respectivamente.

Sin embargo, en vez de hablar de 3 tensores de 2 dimensiones, se hablará de un único tensor de 3 dimensiones, que se forma concatenando los 3 tensores de 2 dimensiones a lo largo de una nueva dimensión. Esta nueva dimensión recibe el nombre de canales.

Como a lo largo de este trabajo se utilizan imágenes a color, el formato de entrada a la mayoría de las redes convolucionales que se utilizan será un tensor de 3 dimensiones de tamaño $H \times W \times C$, donde H representa el alto de la imagen, W el ancho y C la cantidad de canales como se puede ver en la figura 2.2.

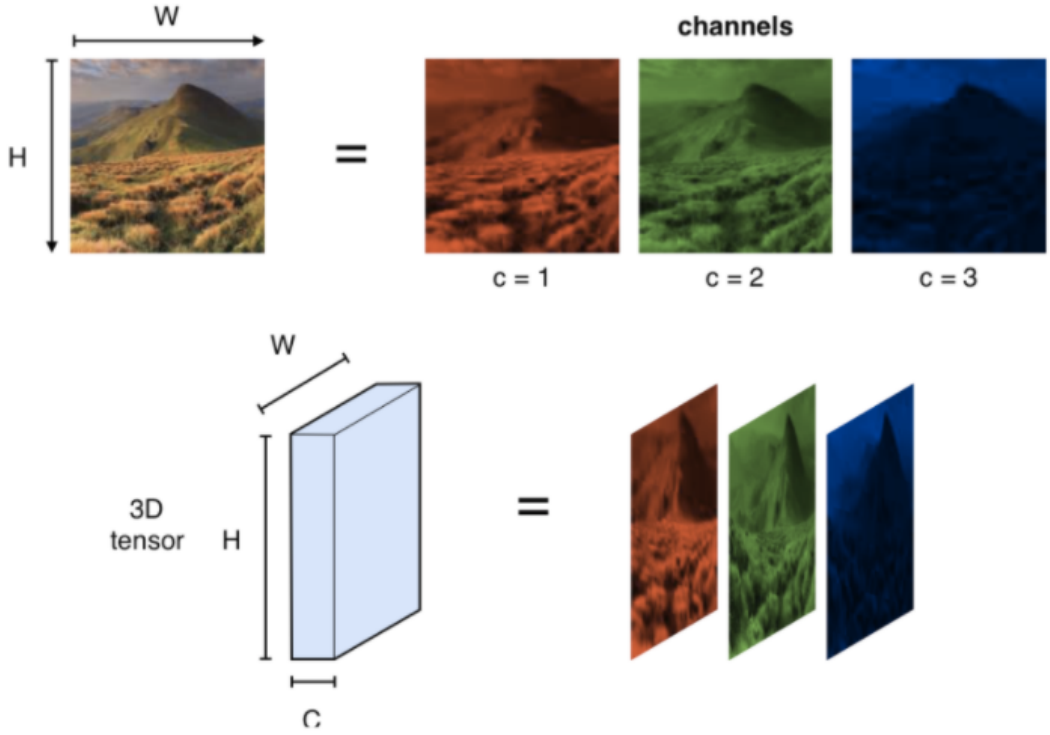

Fig. 2.2: Canales rojo, verde y azul de una imagen a color de alto H y ancho W.

2.2. Redes Neuronales Convolucionales

Las redes neuronales convolucionales —CNN por sus siglas en inglés— son una familia de las redes neuronales que se caracterizan por tener en su arquitectura bloques que procesan la entrada aplicando operaciones de convolución. Están inspiradas en el modelo del Neocognitron planteado por Fukushima en 1980 [11] y su puesta en práctica fue popularizada gracias al trabajo de Yann LeCun et al. [12] al estudiar el reconocimiento de texto en documentos.

Con el advenimiento de la red neuronal convolucional AlexNet [13] que en la competencia de ImageNet Large Scale Visual Recognition en 2012 [14] logró el primer puesto de manera aplastante (superando al segundo lugar por 10.8 puntos porcentuales) las redes neuronales convolucionales fueron ganando merecida popularidad. Hoy en día, las redes neuronales convolucionales dominan el estado del arte en diversas tareas relacionadas con imágenes como segmentación semántica [15], clasificación [2], detección de objetos [16], generación [17] y estimación de pose [18].

Entre las principales ventajas con las que cuentan las redes convolucionales se encuentra la de poder procesar *inputs* de diversos tamaños sin tener que incrementar el número de parámetros entrenables en las capas convolucionales.

En otras palabras, las redes convolucionales pueden procesar imágenes tanto de 28×28 cómo de 500×500 píxeles con la misma cantidad de parámetros. Este mecanismo se conoce con el nombre de *weight sharing* y se debe a que se utilizan *kernels* o filtros de un tamaño fijo sin importar el alto y ancho del input [19].

2.3. Arquitectura de las redes convolucionales

La arquitectura típica de una red neuronal convolucional suele divirse en dos partes:

La primera, llamada *cuerpo* es responsable de realizar una extracción de características de una imagen. Para ello, suele emplear múltiples capas que aplican operaciones de convolución, intercalando entre ellas funciones no lineales y operaciones de *pooling*.

La segunda, llamada *cabeza*, se encarga de adaptar el formato de las características extraídas para que el mismo se adecúe a la tarea concreta a realizar.

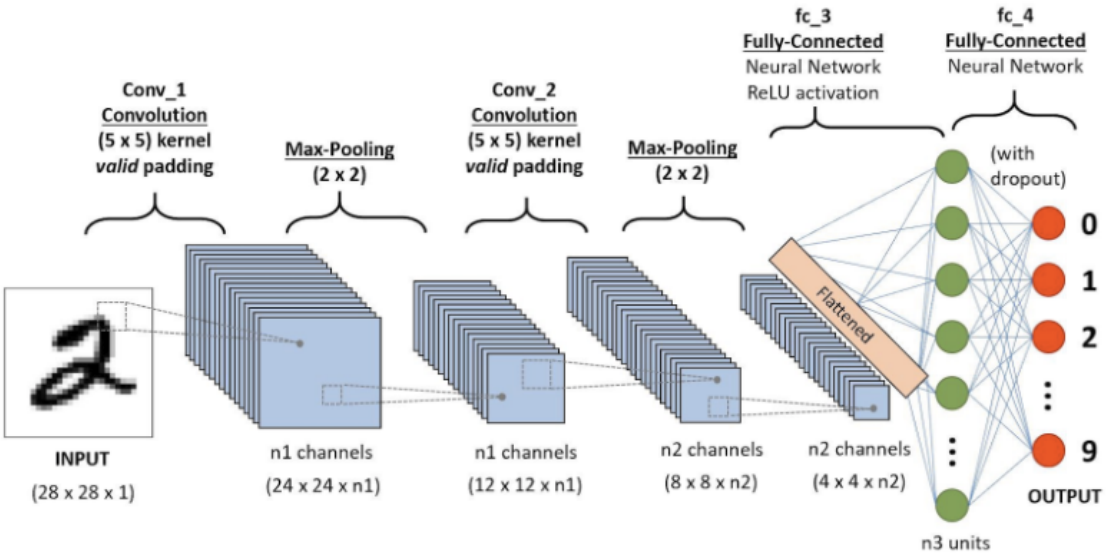

Fig. 2.3: Arquitectura típica de una red convolucional. A la imagen inicial de $28 \times 28 \times 1$ se le aplican sucesivos filtros convolucionales alternados por operaciones no lineales (no presentes en la ilustración) y operaciones de max-pooling. De esta manera mientras que se reduce el tamaño de ancho y alto del tensor con el que se trabaja, aumenta su número de canales. Llegado cierto punto, se aplica una última operación de pooling que termina por convertir cada canal de 4×4 píxeles en un único escalar. La operación de *flattening* permite ver al tensor de 3 dimensiones $1 \times 1 \times n_2$ como un un vector de largo n_2. Por último a este vector se lo procesa a través de una cantidad usualmente pequeña de capas *fully-connected* alternadas por operaciones no lineales que —en el caso de la clasificación— llevarán su largo a que sea igual a la cantidad de clases a predecir.

En el caso de la tarea de clasificación de imágenes, la "cabeza" de la red consiste en un número —usualmente pequeño— de capas fully-connected entre las cuales también se intercalan funciones no lineales.

Una capa *fully connected* se caracteriza por contener un determinado número de neuronas modeladas por perceptrones simples. Cada uno de estos perceptrones tomará un valor de acuerdo a una suma pesada que este realice de la entrada que recibe. Los pesos de aquella *suma pesada* serán los parámetros entrenables de cada perceptrón. El nombre de *fully connected* proviene de que todo perceptrón de una capa es capaz de analizar la totalidad de la entrada, es decir, todo perceptrón está *completamente conectado* a toda porción de la entrada. La cantidad de perceptrones que se elija determinará también el tamaño de salida, es decir, la dimensión del vector resultante.

En la práctica, las capas *fully connected* se modelan como matrices, y su aplicación equivale a la multiplicación de la entrada (el vector) por la matriz. Para una entrada de dimensión e, se puede utilizar una matriz $\mathbf{M} \in \mathbb{R}^{e \times s}$ para obtener una salida de dimensión s.

2.4. Operaciones de Convolución

Antes de definir la operación de convolución, se requiere definir lo que es el *kernel* de una convolución. Un *kernel* o filtro es un tensor de un tamaño menor al input sobre el cual se aplica. Por ejemplo si el input sobre el cual se está realizando la convolución es una imagen de un único canal (la imagen se encuentra en blanco y negro) y tamaño 224×224 píxeles, un kernel para la convolución podría ser una matriz $\mathbf{M} \in \mathbb{R}^{3 \times 3}$.

Convolucionar un kernel con una entrada comienza por multiplicar elemento a elemento el kernel, con la primer porción de la entrada que tenga dimensiones iguales a las del kernel y sumar su resultado obteniendo así lo que constituirá el primer valor de salida.

Formalmente las convoluciones se pueden definir de según la fórmula 2.1, donde $g(x, y)$ es la imágen filtrada, ω es el kernel de convolución, $f(x, y)$ es la imágen original y $-a \leq dx \leq a$, $-b \leq dy \leq b$ indexan a los elementos del kernel (notando al centro en la posición $dx = 0, dy = 0$).

$$g(x, y) = \omega * f(x, y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx, dy) f(x + dx, y + dy) \tag{2.1}$$

Para visualizar más fácilmente esta operación se puede observar la figura 2.4.

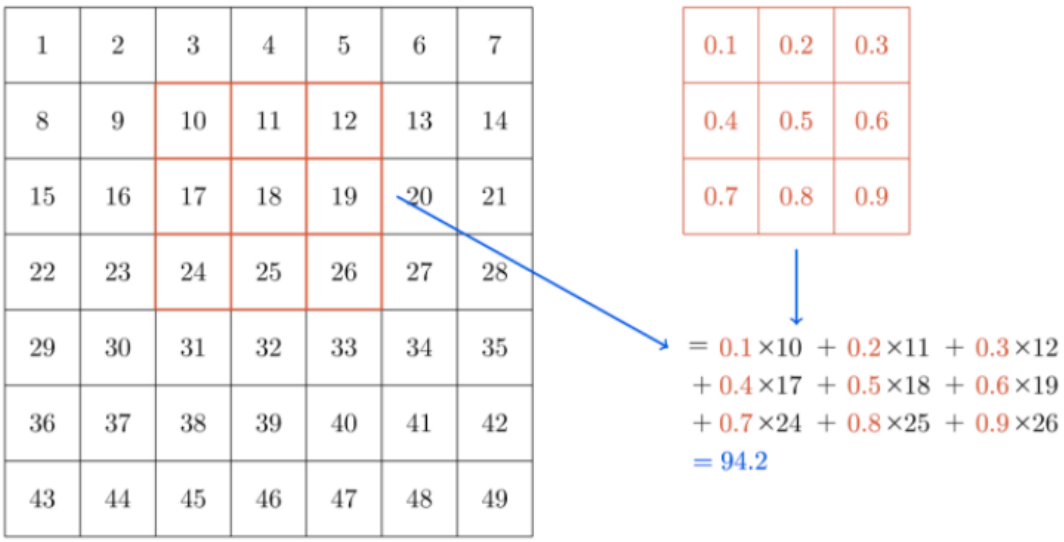

Fig. 2.4: Aplicación del kernel de convolución sobre una porción de la imagen.

Para obtener los restantes valores de salida, habrá de repetirse el proceso, multiplicando cada vez el kernel con una porción distinta de la entrada hasta haber recorrido la imagen por completo.

2.5. ¿Qué sucede en los bordes?

Si se observa atentamente, un kernel de dimensiones 3×3 no puede estar centrado en píxeles correspondientes al borde la imagen pues entonces habría parte del kernel que quede por fuera de la imagen. Para solucionar este problema existe el mecanismo de *padding*. Aplicarle *padding* a una imagen la extiende más allá de sus bordes mediante agregar valores según distintos criterios. El criterio más simple es el de aplicar *zero-padding* extendiendo la imagen mediante rellenar los nuevos valores con 0. Existen otros criterios como *border-padding* que extiende la imagen aplicando el valor que se encuentra en el píxel del borde y *reflection-padding* que hace lo mismo aplicando un reflejo de la imagen a lo largo de la sección de padding. Ver fig 2.5.

Fig. 2.5: Diversas maneras de aplicar padding sobre un input [20].

2.6. Transformación de dimensión

En tareas de clasificación de imágenes a color el input de las redes neuronales convolucionales suelen ser tensores de 3 dimensiones $H \times W \times C$, sin embargo su output debe ser un vector (tensor de dimensión 1) con tantas posiciones como clases de las que elegir [1]. El valor de cada posición representa la probabilidad de que el input procesado corresponda a la cierta clase. Por ende, debido a que constituyen probabilidades la suma de todos los elementos del vector final debe sumar 1. Finalmente, la imagen será clasificada según la clase correspondiente a la posición del vector que arroje la mayor probabilidad.

Ahora bien, ¿cómo se pasa de un espacio de 3 dimensiones —alto, ancho y cantidad de canales— a un espacio de dimensión 1 con tantos elementos como se desee?

La clave está en entender que al aplicar una capa convolucional se puede elegir el tamaño de salida. Respecto a la cantidad de canales como cada filtro de una capa convolucional produce como output un único canal, si se utilizan c filtros entonces la capa de salida tendrá c canales. Para ejemplificar, si a la imagen inicial de 3 canales se le aplica una capa convolucional con 64 filtros, se obtendrá como resultado un output con 64 canales.

[1] En otros casos, como el de la segmentación de imágenes, se querrá un output que determine para cada píxel de la imagen a qué clase corresponde. En estos casos se querrá que el output tenga las dimensiones espaciales de la imagen, pero su tercer componente sea un vector con tantas posiciones como clases de las que escoger. Es decir si la imagen tiene alto y ancho $H \times C$ y existen 5 clases de las que escoger se querrá un output de dimensiones $H \times W \times 5$.

Respecto a las dimensiones espaciales —alto y ancho— de la capa de salida, existen diversos mecanismos para alterarlos. A continuación se desarrolla a las operaciones de pooling y a las configuraciones del stride de la convolución, debido a que estos mecanismos son los que aparecen con mayor frecuencia en la práctica.

Las operaciones de pooling permiten obtener un único valor que represente a otro grupo de valores. La variante de pooling que se encuentra en la práctica con mayor frecuencia es la de *max-pooling*. Esta operación agrupa a los píxeles en bloques de 2×2 y utiliza al máximo de ellos como valor representativo. Al formar una nueva imagen con los valores representativos de cada bloque, la operación de *max-pooling* acaba por reducir alto y ancho de la imagen original a la mitad. De esta manera, la operación de *max-pooling* al ser intercalada entre capas convolucionales, permite ir reduciendo sucesivamente las dimensiones espaciales de las activaciones de la red.

Otro mecanismo para alterar y reducir las dimensiones espaciales es el de configurar el *stride* de las convoluciones. El *stride* de una convolución indica el tamaño del desplazamiento que se realiza al ir deslizando el kernel de convolución sobre la entrada. Un stride de valor igual a 1 hará que el kernel se deslice sobre la imagen de a 1 píxel por vez. Mientras que un stride de valor igual 2 —valores mayores son muy poco frecuentes en la práctica— hará que el kernel se deslice de a 2 píxeles. Para mayor clarificación la figura 2.6 ilustra el accionar del stride sobre el tamaño de salida.

De esta manera las redes neuronales convolucionales van transformando la información, llevándola desde el espacio de imágenes a color, a un espacio con cada vez más canales y cada vez menor alto y ancho.

Si bien la cantidad de canales del resultado de las operaciones de convolución está directamente ligado a la cantidad de filtros que se estén utilizando, entender el tamaño de salida espacial—ancho y alto— de las operaciones de convolución requiere entender a todas las variables que entran en juego. Para una capa típica convolucional —con tamaño de kernel K, stride S y padding P— el tamaño del resultado de ingresar una imagen de dimensión W será:

$$W_{output} = \left\lfloor \frac{W_{input} - K + 2P}{S} + 1 \right\rfloor$$

Luego de una numerosa cantidad de capas convolucionales, es usual en las arquitecturas modernas —llegado un momento donde alto y ancho son considerablemente pequeños— realizar una operación de pooling espacial final, que termine por convertir las activaciones a un espacio unidimensional. Para dar un ejemplo, si al final de todo el pase a través de las capas convolucionales se cuenta con un set de activaciones de dimensión $7 \times 7 \times 256$, la operación de pooling final las transforma a un espacio de dimensión $1 \times 1 \times 256$, lo cual también puede verse como un vector de 256 posiciones.

Una vez llegado a ese punto, las activaciones se encuentran en un formato al cual pueden aplicarse capas *fully-connected* que redimensionen dicho vector unidimensional hasta que su número de posiciones iguale la cantidad de clases a elegir. Siguiendo el ejemplo anterior, el vector de 256 posiciones puede convertirse en un vector de 10 posiciones (en caso de que hubiera 10 clases a predecir).

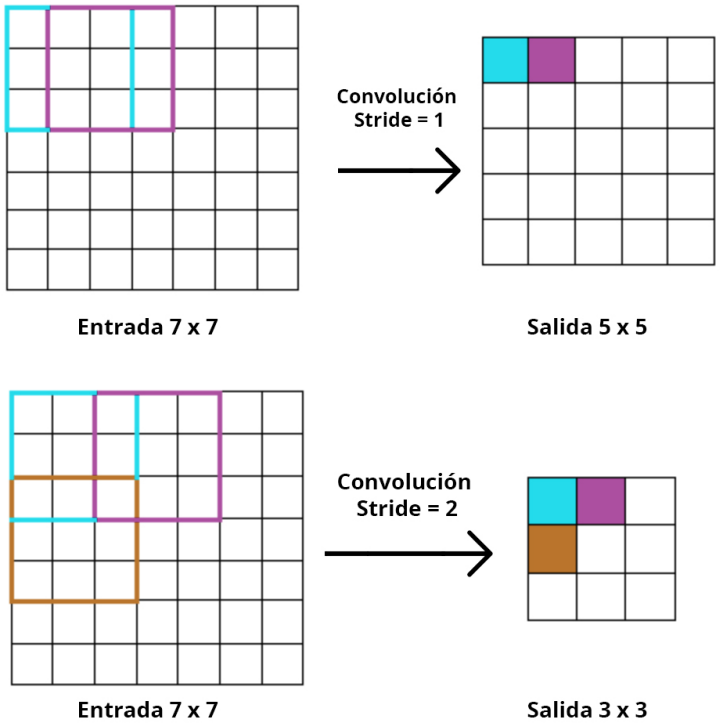

Fig. 2.6: Comparación de tamaños de salida según el tamaño de stride que se utilice. En la figura de arriba se utiliza un stride igual a 1 mientras que en la de abajo se utiliza un valor igual a 2. El recuadro celeste denota la primer posición donde el kernel se multiplica elemento a elemento con la entrada. Mientras que el recuadro violeta denota la segunda posición donde sucede. Se puede observar que cuando se utiliza un stride igual a 2 (abajo) el *recorrido* que puede realizar el kernel sobre la entrada es menor y por tanto las dimensiones de salida también lo son. El stride no solo aplica en sentido horizontal, si no también en el sentido vertical. El recuadro marrón (abajo) indica cuál será el primer desplazamiento del kernel en el sentido vertical cuando se utiliza un stride igual a 2.

2.7. ¿Cómo construyen las redes su entendimiento de las imágenes?

Hasta ahora se ha hablado de las redes neuronales convolucionales desde un punto de vista técnico; se ha visto cómo logran manipular las dimensiones de la información desde el espacio de las imágenes a color hacia un espacio que permite realizar inferencia sobre posibles clasificaciones o características de las imágenes.

Resta ahora analizar qué esta sucediendo en cada sección: qué tarea realiza el cuerpo de la red y qué tarea realiza la cabeza.

2.7.1. Visualizando a las Convoluciones

Las convoluciones permiten realizar un filtrado de las frecuencias de la imágen. Dependiendo de las configuraciones de los pesos del kernel las convoluciones pueden filtrar las señales altas o bajas de las imágenes. En la práctica esto significa que una convolución puede servir por ejemplo para detectar bordes en la imagen (frecuencias altas).

¿Ahora bien, qué sucede cuando se aplican convoluciones de manera sucesiva? [2] En el trabajo *Visualizing and Understanding Convolutional Networks* [21] se estudia este fenómeno: los autores encuentran que cada capa convolucional de la red permite detectar características incrementalmente más complejas que la capa anterior.

En la figura 2.7 se puede observar el efecto: mientras que la primer capa de filtros actúa como detector de bordes, la segunda capa logra detectar features más complejas como círculos, patrones de lineas horizontales, y demás. Continuando con las visualizaciones a lo largo de la red, en la figura 2.8, se visualiza cómo ya en la tercer capa las activaciones reflejan detalles característicos de las clases que la red debe clasificar (ruedas, torsos, etc...).

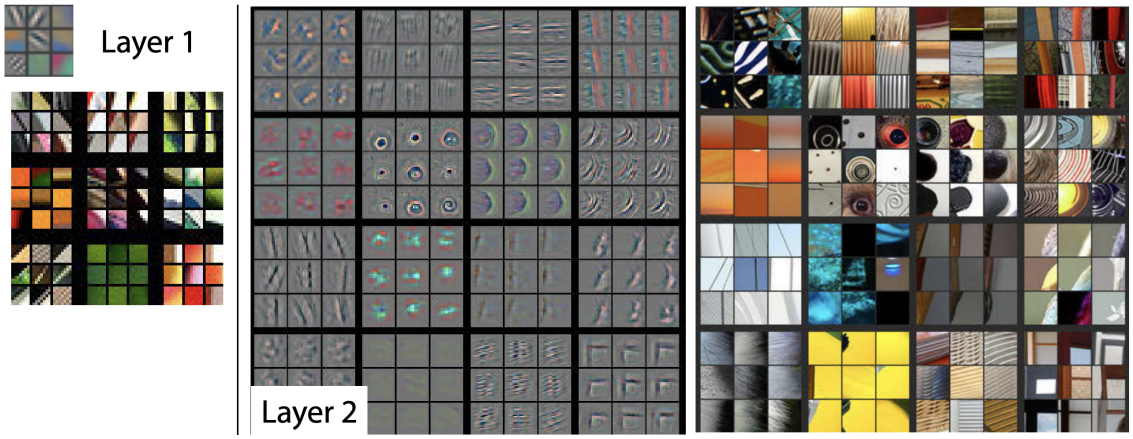

Fig. 2.7: Visualización de las primeras dos capas de una red neuronal convolucional. Para cada capa se visualizan las activaciones (llevadas al espacio de píxeles mediante un proceso que involucra deconvoluciones) e imágenes artificialmente generadas que maximizan las activaciones salientes cuando pasan por los filtros respectivos. [21].

De esta manera la sucesiva aplicación de capas convolucionales permite sofisticar las características detectadas. Luego el cuerpo de la red convolucional permite generar a su salida un vector donde cada posición del mismo representa la presencia de cierta característica compleja en la imagen siendo procesada. Es decir actúa como *feature extractor* —o extractor de características— permitiendo convertir una imagen en un vector que la describe mediante definir si para una cierta cantidad de características, cada una de estas está presente o no en la imagen.

Es necesario notar que parte del poder de las redes convolucionales surge de que no es necesario elegir las características a detectar manualmente, si no que estas surgen automáticamente en el proceso de entrenamiento.

2.7.2. Fine tuning con capas fully connected

La tarea no acaba una vez que se utiliza el cuerpo de la red como *feature extractor* y se cuenta con el vector de características para una imagen; aún es necesario predecir a qué clase pertenece la imagen.

Si bien sería posible dar una arquitectura compuesta únicamente por capas convolucionales de tal manera que el vector de características final tenga tantas posiciones como

[2] Es necesario notar que "de manera sucesiva"siempre las asume intercaladas por operaciones no lineales. De no ser así, dado que las convoluciones son operaciones lineales, sería lo mismo aplicar una cantidad n de convoluciones que una única convolución que capture el efecto total.

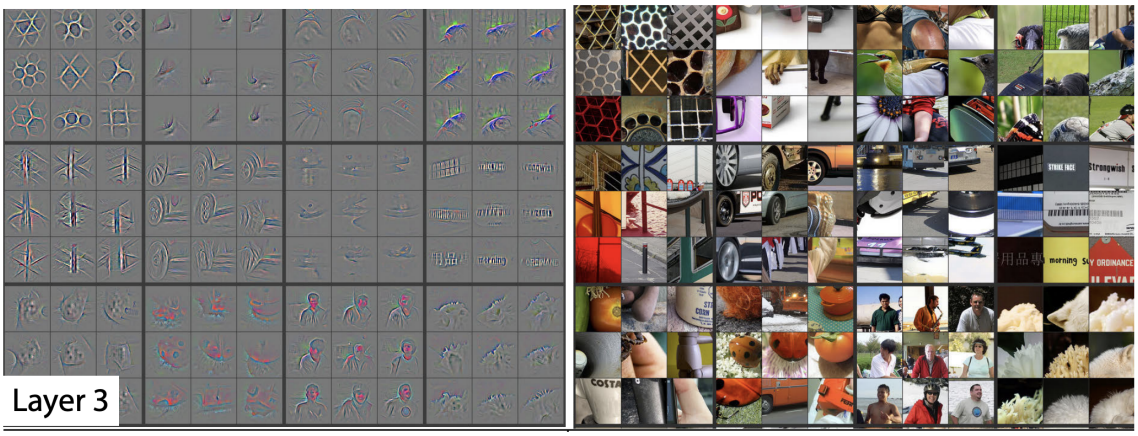

Fig. 2.8: Visualización de la tercera capa de una red neuronal convolucional. Se utilizan las mismas técnicas de visualización que en la figura 2.7. Se puede observar como ya en esta capa los elementos que los filtros activan se asemejan a elementos de la vida real (como ruedas, torsos de personas, etc...) y parecieran ser elementos característicos de las clases que se intenta clasificar (autos, personas, etc...) [21].

clases a elegir y se realice un mapeo 1 a 1 entre features detectados [3], este modelo estaría desperdiciando una gran optimización posible.

Los elementos de la realidad son entes complejos y constan de varias partes, algunas se comparten y otras los diferencian. Las capas *fully connected* permiten realizar una última abstracción que termina delineando el concepto de la clase a clasificar mediante la posibilidad de recombinar diversas características de las extraídas por la red. Para ejemplificar, el concepto de auto se puede formar mediante combinar las features 'rueda', 'puerta' y 'luces delanteras'.

Una pregunta que podría surgir en esta etapa es ¿ya que las convoluciones funcionan tan bien en el cuerpo, por qué no se utilizan también para esta última recombinación de características?

Si bien es cierto que podría utilizarse una convolución en este espacio de 1 dimensión (el vector de features extraídos), esto no haría tanto sentido. El motivo de ello que es que las convoluciones tienen sentido cuando la información sobre la que se las aplica tiene cierta estructura de continuidad en el espacio (en el caso de las imágenes, píxeles cercanos suelen representar a un mismo objeto). Ahora bien, cada posición del vector de características final surge de tomar un valor representativo de un cierto canal de las activaciones finales y los canales no están ordenados entre sí [4]. De modo que no existe ninguna relación entre las posiciones del vector de características y aplicar convoluciones sobre este deja de tener sentido.

Es en estos momentos donde el concepto de capa *fully connected* posiciona como la mejor opción. Las capas *fully connected* permiten hacer una recombinación de todos los elementos de la entrada sin importar su posición. Permiten decidir para cada clase a clasificar cuánto influye cada una de las características detectadas. Es una operación más costosa que las convoluciones pero a cambio de no tener prerrequisitos sobre la estructura

[3] Se podría entender que la clase a la que corresponde la imagen es la que corresponde a la característica activada con mayor valor

[4] Recordemos que en una capa de la red cada canal de las activaciones surge de un filtro completamente independiente de los otros

(i.e: relaciones espaciales) de los datos de entrada.

2.8. Deep Residual Networks

Las redes Deep Residual Networks o *ResNets* [4] obtuvieron gran popularidad cuando ganaron la competencia ImageNet Large Scale Visual Recognition Challenge (ILSVRC) en 2015, al lograr un error de tan solo 3.57 % en el set de testing.

La característica principal de las *ResNets* es que agregan a las capas convolucionales una conexión directa —también llamada *skip connection*— desde la entrada a la salida. En concreto, en estas capas el output es el resultado de sumar la entrada sin procesar, con la entrada procesada. Ver fig 2.9.

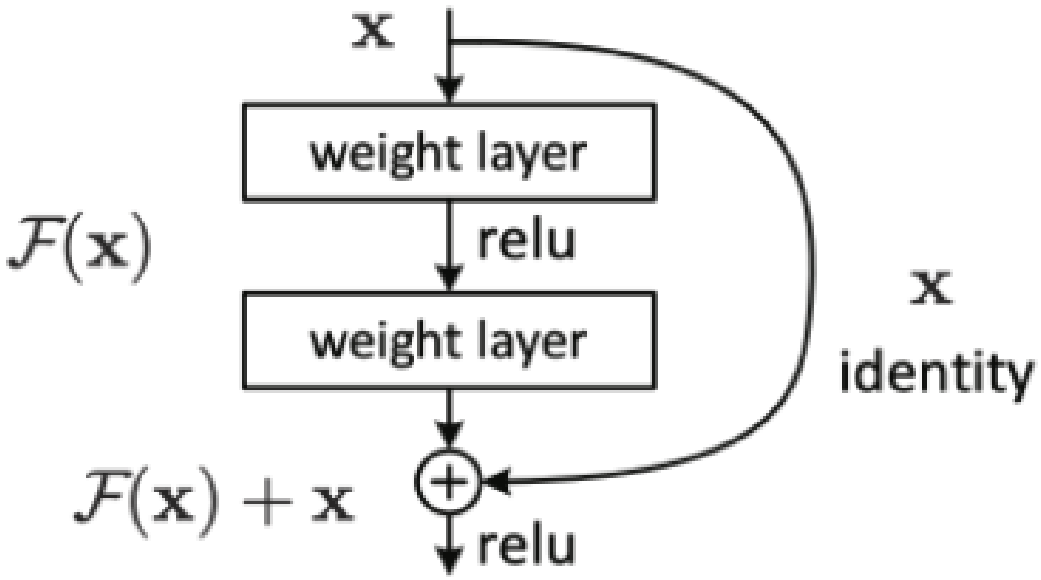

Fig. 2.9: Bloque básico con *Skip connection* presente en las redes resnet [4]. En la figura las *weight layers* son capas convolucionales y ReLU —cuya sigla significa Rectified Linear Unit— es la operación no lineal que se utiliza para romper la linearidad de la aplicación de dos convoluciones sucesivas. En la práctica, ReLU retorna el máximo entre el input y 0 para cada posición. Llámese entonces $\mathcal{F}(\mathbf{x})$ a la aplicación de las dos convoluciones —*weight layers*— intercaladas por una ReLU. Esta $\mathcal{F}(\mathbf{x})$ sería un bloque típico que podría encontrarse en una red convolucional cualquiera. La sutileza del agregado de la skip connection se visualiza claramente cuando ese bloque se convierte de $\mathcal{F}(\mathbf{x})$ a un bloque que produce $\mathcal{F}(\mathbf{x})+\mathbf{x}$ es decir, al output $\mathcal{F}(\mathbf{x})$ se le agrega nuevamente el input $\mathbf{x}$. Este pequeño cambio es el que caracteriza a las redes *ResNets* y el responsable por su notable performance.

¿Cuál es el motivo detrás de dicho agregado?

Los autores plantean que existe una degradación de la performance que surge al implementar modelos demasiado profundos: la *accuracy* comienza a saturarse para luego empeorar. Se descarta que el motivo de esto sea overfitting pues [22] demuestra que también se incrementa el error en el training set.

Luego parecería ser que no toda profundidad de arquitectura es igual de fácil de optimizar. Pues el planteo sencillo es que si existe un modelo poco profundo que funciona bien, debería existir un modelo profundo que funcione al menos tan bien como el. La existencia

de dicho modelo se prueba por construcción, pues se puede generar el modelo profundo a partir de agregar capas que realicen la función identidad al modelo poco profundo.

Sin embargo, en la práctica esto queda refutado ya que los modelos demasiado profundos suelen incrementar el error en el dataset de entrenamiento. Es decir, parecería que no les resulta sencillo a los modelos profundos llevar a que ciertas capas realicen la función identidad.

Es por ello que el agregado que hacen los autores intenta facilitar la tarea. Por medio de agregar una *skip connection* lo que realmente se está haciendo es permitiendo a la capa utilizar la función identidad como base. Es decir, si los pesos entrenables tienden a 0, el output de la capa ahora dejaría de ser 0 para pasar a ser la función identidad sobre la entrada.

En [23] los autores demuestran que agregar la *skip connection* hace que el número de condición del hessiano de la función de pérdida en el punto inicial 0 sea invariante con la profundidad de la red, lo que haría entrenar modelos profundos con la misma facilidad que modelos poco profundos.

2.9. ResNet-34 y ResNet-50

A partir de la introducción de la *skip connection* se generaron ciertos modelos estándar con diferentes cantidades de capas según el costo computacional y la performance que se desee.

El primero de ellos es **ResNet-34** [4] que contiene 34 capas convolucionales. Esta arquitectura puede observarse en la figura 2.10. La notación dentro de cada bloque representa el tamaño del kernel, seguido por la cantidad de canales y finalmente la utilización o no de *stride* de tamaño 2. Las lineas punteadas representan *skip connections* que suceden al mismo tiempo que se incrementa la cantidad de canales. Para ejemplificar 3×3 conv, 64 /2 representa un bloque convolucional que utiliza kernels de tamaño 3×3 (y profundidad igual a la cantidad de canales de entrada), producirá 64 canales de salida (son 64 filtros los que se emplean) y utilizará un stride de 2 reduciendo a la mitad las dimensiones alto y ancho del output con respecto al input. La operación denotada por *pool,/2* que se encuentra luego del primer bloque convolucional representa la aplicación de *max-pooling* en sentido espacial, reduciendo alto y ancho a la mitad. La operación denotada por *avg pool* luego del último bloque convolucional representa la aplicación de un pooling que reduce la información espacial de cada canal a un único escalar formado por el promedio de los valores de los mismos. Finalmente el bloque denotado por *fc 1000* denota una capa *fully-connected* con 1000 valores de salida, es decir una matriz $M \in \mathbb{R}^{c_{in} \times 1000}$, donde c_{in} representa la cantidad de canales de entrada del bloque.

Si bien *ResNet-34* se ha vuelto un modelo muy popular —quizá debido a que se posiciona en un punto medio práctico entre costo computacional y performance obtenida— también existen otros modelos estándar. Por un lado *ResNet-18* es una variante similar pero con 18 capas convolucionales en vez de 34, útil si es que se desea obtener un menor costo computacional a la vez que se cede precisión. Por otro lado, *ResNet-50*, *ResNet-101* y *ResNet-152* hacen lo inverso: aumentan el número de capas (y con ello el costo computacional) a cambio de poder proveer resultados más precisos.

Ahora bien, aumentar el número de capas no es el único cambio que introducen los modelos *ResNet-50* en adelante. Los autores He et al.[4] en busca de reducir el costo computacional de estos nuevos modelos a un nivel manejable y práctico, cambian el diseño

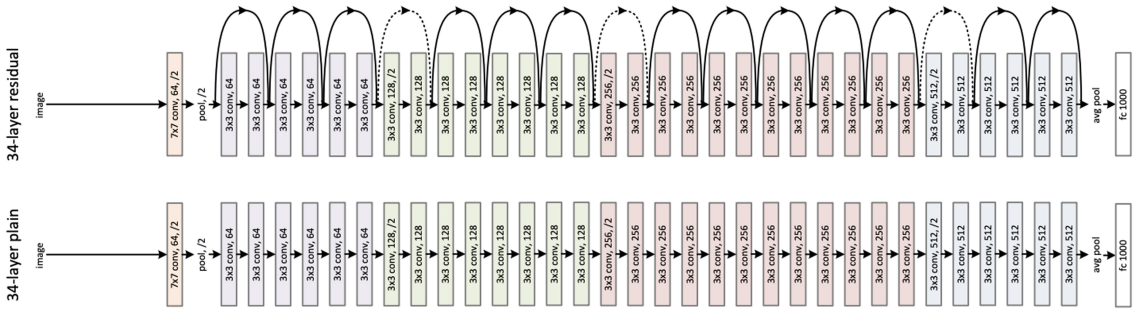

Fig. 2.10: Arquitectura de una ResNet-34 (arriba) [4] comparada con una red idéntica (abajo) sin la *skip connection*.

de cómo se implementa la *skip connection*: ahora en lugar de realizar la suma del input al output cada 2 bloques convolucionales, lo harán cada 3, pero utilizando el diseño de *bottleneck*.

El *bottleneck* —cuello de botella— utiliza 3 bloques convolucionales. El propósito del primero de ellos es el de reducir la cantidad de canales con la que se está trabajando. El propósito del segundo es el de realizar una convolución típica sobre este input reducido. Finalmente, el propósito del tercero es el de restaurar la cantidad de canales original que contenía el input del primero bloque. Es decir, como su nombre implica, *bottleneck* reduce la cantidad de canales para realizar la operación de convolución pero luego restaura las dimensiones originales del input.

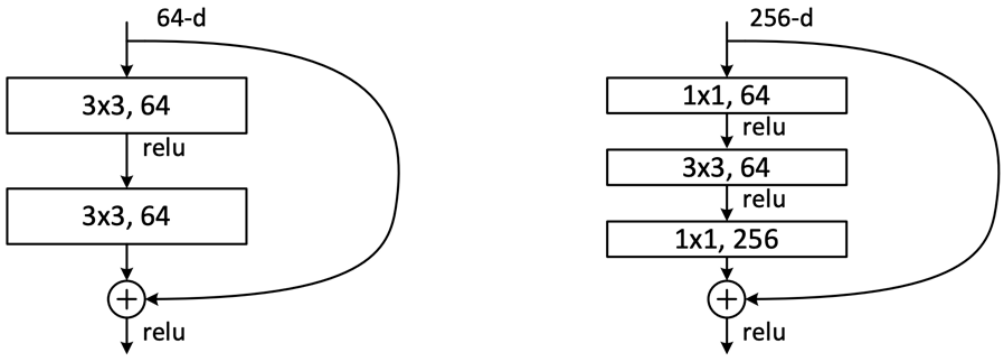

Fig. 2.11: Diseño de bloque con *skip connection* para las redes ResNet-18 y ResNet-34 (izquierda) en comparación con diseño de bloque *bottleneck* para los modelos ResNet-50 en adelante (derecha) [4]. Los parámetros de los bloques deben interpretarse de la misma forma que se explica en la figura 2.10. Los parámetros de 64-d y 256-d representan la cantidad de filtros que se utilizan en cada capa y por ende también canales de output que genera cada bloque. ReLU hace referencia a la aplicación de la función no-lineal Rectified Linear Unit que toma el máximo elemento a elemento entre dicho elemento y 0.

Como se puede observar en la figura 2.11 el primer y último bloque convolucional de un diseño *bottleneck* utilizan un kernel de alto y ancho iguales a 1. El motivo de esto es que el propósito de ellos es el de realizar la reducción y restauración de la cantidad de canales respectivamente, por ende para ello no se necesita más que un kernel de tamaño

1×1. El único momento donde el kernel sí tiene dimensiones de 3×3 es en el bloque convolucional del medio, al cual se le provee un input de 64 canales y no 256, reduciendo así el costo computacional incurrido.

Con estos cambios en mente, la red *ResNet-50* se forma tomando a la red *ResNet-34* y reemplazando cada bloque de 2 capas, por un bloque *bottleneck* resultando así en una red de 50 capas.

2.10. EfficientNet

Las arquitecturas EfficientNet superan a las arquitecturas ResNet —equivalentes en poder de cómputo— en cada vez más tareas [5].

El aporte de las arquitecturas *EfficientNet* consiste en dar una manera definida de cómo escalar los modelos de las arquitecturas cuando se dispone de mayor poder de cómputo.

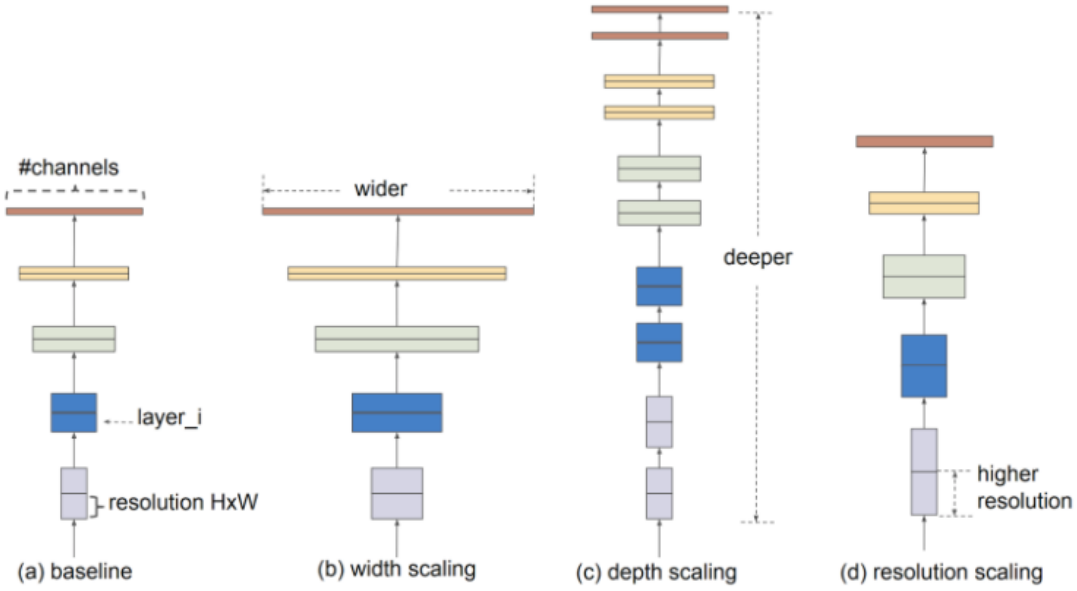

Fig. 2.12: Diversas maneras de agrandar una red neuronal [5].

La decisión entre optar por agregar canales a las capas (*width scaling*), optar por agregar capas al modelo (*depth scaling*) u optar por agregar resolución espacial a las capas (*resolution scaling*) se especifica mediante una restricción que supone aprovechar al máximo el poder de cómputo disponible. Ver fig 2.12.

Dicha restricción está definida por las siguiente inecuaciones:

$$depth : d = \alpha^{\phi}$$

$$width : w = \beta^{\phi}$$

$$resolution : r = \gamma^{\phi}$$

$$s.t. : \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

Donde ϕ es un parámetro ajustable por el usuario que regula los recursos disponibles y α, β, γ son parámetros que se determinan por una pequeña grid search. Según la convención utilizada por los autores un valor de $d = 1$ implica 18 capas convolucionales y un valor de $r = 1$ implica un tamaño de 224x224 de las imágenes.

Esta restricción da lugar a la familia de redes llamadas *EfficientNet-b1* a *EfficientNet-b7*, según su capacidad y complejidad.

El modelo base EfficientNet-b0 sobre el que se producirá el escalamiento es también de crucial importancia. La elección del mismo se realiza utilizando *Neural Architecture Search* similar a [24] optimizando una función objetivo que se define por

$$Acc(m) \times \left(\frac{FLOPS(m)}{T}\right)^{w}$$

Donde $ACC(m)$ y $FLOPS(m)$ constituyen la *accuracy* y los *flops* del modelo m, T constituye el target de *flops* a alcanzar y $w = -0{,}07$ es un parámetro que regula la importancia de los $flops$ en la optimización.

El elemento base de la red es el bloque MBConv [25] al cual se le agrega el mecanismo de Squeeze & Excitation [9].

2.11. Bloques MBConv

EfficientNet utiliza como sus bloques constitutivos a los denominados *MBConv*. La primera aparición de estos bloques en las redes neuronales sucede con la arquitectura *MobileNet* [25].

La motivación original de la arquitectura *MobileNet* era la de obtener un modelo que pueda realizar inferencia de manera rápida *on the edge* es decir, en los mismos dispositivos móviles. Es por ello que los bloques *MBConv* tienen ciertas características que les permiten reducir la cantidad de FLOPS requerida, sin perjudicar demasiado la accuracy del modelo.

Entre las características de los bloques *MBConv*, una de las más distintivas es la de implementar *Depthwise Separable Convolutions*.

2.11.1. Depthwise Separable Convolutions

Una capa de convolución estándar toma un tensor de tamaño $h_i \times w_i \times c_i$, (donde h es la altura, w el ancho y c la cantidad de canales) y aplica una cantidad c_j de kernels convolucionales de tamaño $k \times k \times c_i$ cada uno, produciendo así un output de tamaño $\mathrm{h}_i \times w_i \times c_j$. El costo de este tipo de convoluciones estándar es de $\mathrm{h}_i \cdot w_i \cdot c_i \cdot c_j \cdot k \cdot k$.

La idea detrás de las *depthwise separable convolutions* es la de reducir este costo mediante separar la capa convolucional en 2 capas distintas:

La primera capa, llamada *depthwise convolution*, crea un kernel convolucional de profundidad 1 por cada uno de los canales de entrada. Cada uno de estos kernels se aplica únicamente y de forma independiente sobre su canal respectivo. El output se forma concatenando la salidas de la aplicación de cada kernel. Por lo tanto, la *depthwise convolution* mantiene la cantidad de canales de input como cantidad de canales de output. La aplicación de esta capa también puede ser visualizada en la figura 2.13

La segunda capa —llamada *pointwise convolution*— es una convolución estándar con kernel de tamaño 1×1. Dicho tamaño del kernel hace que el output sea una combinación lineal de los canales de entrada. A diferencia de la capa anterior, en esta puede utilizarse

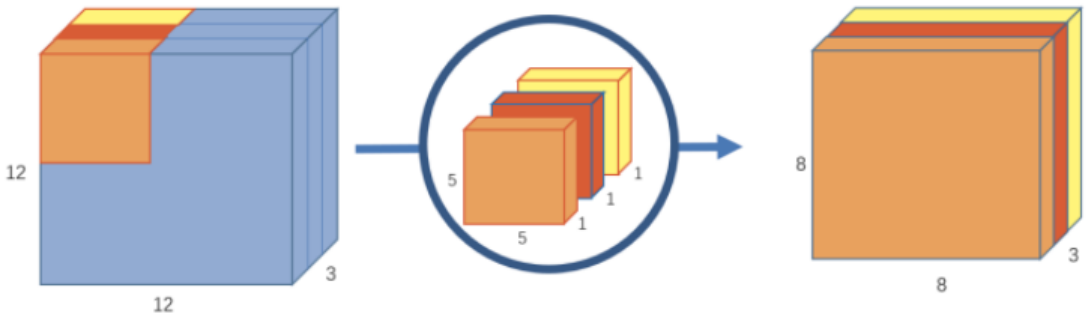

Fig. 2.13: Depthwise convolution [25]. A la izquierda se observa un input de tamaño $12 \times 12 \times 3$. Como se cuenta con 3 canales, se crean 3 kernels de profundidad 1 con alto y ancho arbitrario. En el ejemplo se toman alto y ancho de tamaño 5, de manera que cada uno de los 3 kernels tiene dimensión $5 \times 5 \times 1$. Cada uno de los kernels se aplicará sobre su canal respectivo (en la ilustración esto se denota con la utilización de colores). A la derecha de la ilustración se observa el output de dimensión $8 \times 8 \times 3$. Como se puede ver la cantidad de canales permanece inalterada. El motivo de que alto y ancho se reduzcan a 8 es que los kernels tienen alto y ancho de tamaño 5 y no se esté utilizando padding (esto se explica en la sección 2.5).

la cantidad de filtros que se considere necesaria. Esta libertad de elección de cantidad de filtros permite a su vez elegir la cantidad de canales de salida. Cada uno de los filtros puede visualizarse con la figura 2.14

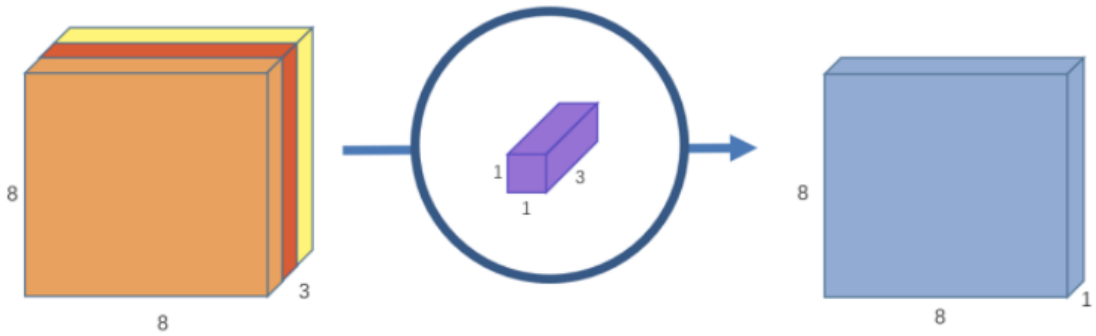

Fig. 2.14: Aplicación de un único kernel Pointwise convolution [25]. A la izquierda se observa un input de tamaño $8 \times 8 \times 3$. Se aplica un único kernel de dimensión $1 \times 1 \times 3$, por ende el resultado tendrá un único canal. Como alto y ancho del kernel son iguales a 1, no existen problemas de padding y por ende se mantienen alto y ancho de 8, dando un output de tamaño $8 \times 8 \times 1$.

De esta manera, por dar un ejemplo, si se utilizan 256 de estos filtros, se puede lograr un set de activaciones con 256 canales. Tal como se ilustra en la figura 2.15

Esta separación de las típicas capas convolucionales en 2 capas distintas con las características antes mencionadas mantiene empíricamente los mismos resultados, pero reduciendo el costo computacional a $h_i \cdot w_i \cdot c_i(k^2 + c_j)$. Ya que el costo de la primera es de: $h_i \cdot w_i \cdot k \cdot k \cdot c_i$ Y el costo de la segunda es de $h_i \cdot w_i \cdot c_i \cdot c_j$.

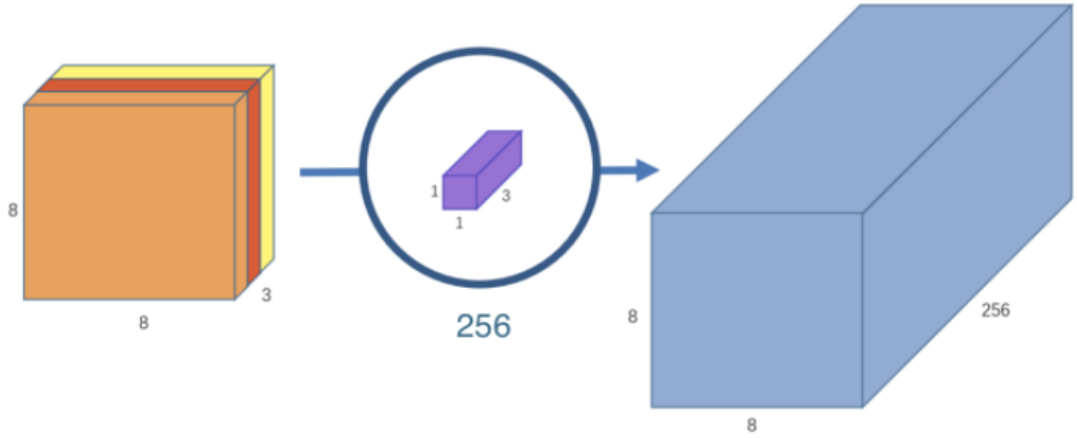

Fig. 2.15: Pointwise convolution [25]. Al igual que en la figura 2.14 a la izquierda se observa un input de tamaño $8 \times 8 \times 3$. Sin embargo, en esta figura se visualiza la aplicación de 256 kernels de dimensión $1 \times 1 \times 3$, dando un resultado con 256 canales. Por los mismos motivos que en la figura 2.14, se mantienen alto y ancho de tamaño 8.

2.11.2. Inverted Residuals

Otro cambio significativo que aporta *MobileNet* es el de la utilización de *Inverted Residuals*. Este cambio tiene que ver en la manera en que varían los canales en los cuellos de botella o *bottlenecks*. En los bloques bottleneck de la familia de las redes ResNet se hace una reducción de canales previo a aplicar la operación de convolución.

Como se explica en la sección 2.9 el motivo detrás de ello es el de reducir el costo computacional de la operación de convolución mediante proveer un input con menos canales (en el caso que ejemplifica la imagen 2.11, la operación de convolución se hace sobre 64 canales en vez de 256).

Sin embargo en MobileNet esta dinámica se invierte: en lugar de reducir los canales y luego volver a expandirlos, los canales se amplían previo a realizar la operación de convolución y luego se vuelven a reducir. Esto se puede visualizar en la figura 2.16

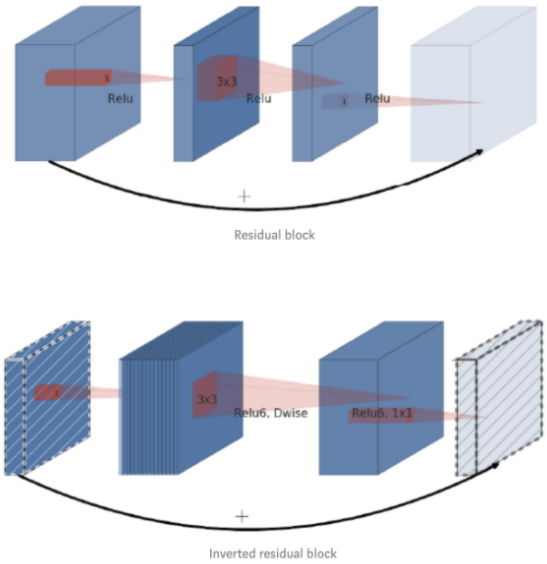

Fig. 2.16: Comparación entre bottlenecks de *ResNets* (arriba) e inverted bottlenecks de MobileNet (abajo) [25].

Según Sandler et al. [25], en su modelo se ve una natural separación entre el input/output de los bloques constitutivos (capas bottleneck) y las transformaciones entre capas —las funciones no-lineales que convierten inputs en outputs—. Ellos interpretan a lo primero como la *capacidad* de la red y a lo segundo como la *expresividad* de la misma.

Con estas distinciones en mente, los autores argumentan que su arquitectura goza de buenos resultados ya que —gracias a su alta expresividad— dota a los modelos capacidad para capturar información compleja, pero —gracias a su reducida capacidad— mantiene el costo computacional dentro de un rango aceptable.

2.11.3. EfficientNet-b0

De todos los modelos que componen la familia de las redes EfficientNet (b0-b7), en el presente trabajo se utilizará el modelo b0. El modelo b0 es el modelo de menor tamaño lo cual permite reducir el tiempo que conllevan los experimentos y así permitir un desarrollo de forma iterativa e incremental sobre los mismos.

Los modelos de redes neuronales convolucionales se suelen poder descomponer en distintas *etapas* cada una compuesta por repetidas *capas* de un mismo tipo. En el caso de EfficientNet-b0 se cuenta con 9 etapas y un total de 18 capas como se puede ver en la figura 2.1.

Etapa i	Operador $\hat{\mathcal{F}}_i$	Resolución $\hat{H}_i \times \hat{W}_i$	#Canales $\hat{C}_i$	#Capas $\hat{L}_i$
1	Conv3x3	224×224	32	1
2	MBConv1, k3x3	112×112	16	1
3	MBConv6, k3x3	112×112	24	2
4	MBConv6, k5x5	56×56	40	2
5	MBConv6, k3x3	28×28	80	3
6	MBConv6, k5x5	14×14	112	3
7	MBConv6, k5x5	14×14	192	4
8	MBConv6, k3x3	7×7	320	1
9	Conv1x1 & Pooling & FC	7×7	1280	1

Tab. 2.1: Modelo EfficientNet-B0. Cada fila describe una etapa i compuesta por $\hat{L}_i$ capas de tipo $\hat{\mathcal{F}}_i$, input de resolución $\langle \hat{H}_i, \hat{W}_i \rangle$ y $\hat{C}_i$ canales de salida [5].

2.12. Mecanismos de Atención

En el presente trabajo se aplica la técnica de *Attention Residual Learning* —al que ahora nos referiremos como ARL— a las redes *ResNet50* con el propósito de reproducir los resultados de [8] para luego poder estudiar su efecto en la familia de redes EfficientNet.

Dentro de estas últimas, se estudia el impacto que *ARL* produce tanto en las arquitecturas originales como en modelos donde se elimina el mecanismo de attention presente en ellas conocido como *Squeeze & Excitation*.

2.12.1. Attention Residual Learning

En el paper "*Attention Residual Learning for Skin Lession Classification*"[8] los autores logran simular el efecto de una capa de attention sin incurrir en el agregado costo computacional de una cantidad significativa de nuevos parámetros.

Para ello, su técnica se basa en agregar una segunda *skip-connection* a los bloques ResNet, donde el input original es multiplicado elemento a elemento por el output del bloque al cual se le aplica la función *softmax* de manera espacial.

El output final del bloque queda entonces compuesto por: el output típico de un bloque resnet (con su *skip-connection* regular) sumado a este nuevo agregado de ARL el cual es es controlado por un escalar *alpha* que regula la intensidad del efecto. Esto se puede ver formalmente en la siguiente ecuación donde y representa el output, x el input, F es el bloque convolucional, α es el escalar que regula la intensidad del efecto, $\cdot$ es la multiplicación elemento a elemento y finalmente $\mathfrak{N}$ es la función *softmax* aplicada espacialmente.

$$y = x + F(x) + \alpha \cdot \mathfrak{N}[F(x)] \cdot x \tag{2.2}$$

A su vez la función *softmax* aplicada espacialmente se define por la siguiente ecuación donde O representa el input y $m^c_{i,j}$ representa el valor en la posición (i, j) del canal c del output m.

$$\mathfrak{N}^S(O) = \left\{ m \mid m^c_{i,j} = \frac{e^{o^c_{i,j}}}{\sum_{i',j'} e^{o^c_{i',j'}}} \right\}.$$

La diferencia en los bloques que produce este agregado puede visualizarse mejor al observar la ilustración 2.17.

La idea detrás del mecanismo es la siguiente: los bloques convolucionales de las *ResNet* producen una transformación de su input que detecta ciertos features. La aplicación de *softmax* sobre este output permite resaltar las posiciones más activadas y llevar a valores muy pequeños las restantes. Es entonces que la aplicación de *softmax* resulta en un conjunto de valores que indican para cada canal qué activaciones son las más importantes en este flujo de la información. Este conjunto de valores se aplica mediante multiplicarlo elemento a elemento con el input aumentando ciertos valores y disminuyendo otros para quedarse con las posiciones que la red "consideró" de mayor relevancia en dicho bloque.

Finalmente es importante notar que los únicos parámetros entrenables que ARL agrega a la red son los denominados *alpha* que actúan como reguladores de la influencia de ARL sobre la ResNet original. En otras palabras, de ser que el efecto de ARL no ayuda a la clasificación, los parámetros alpha pueden tender a 0, anulando todo efecto agregado a la red original.

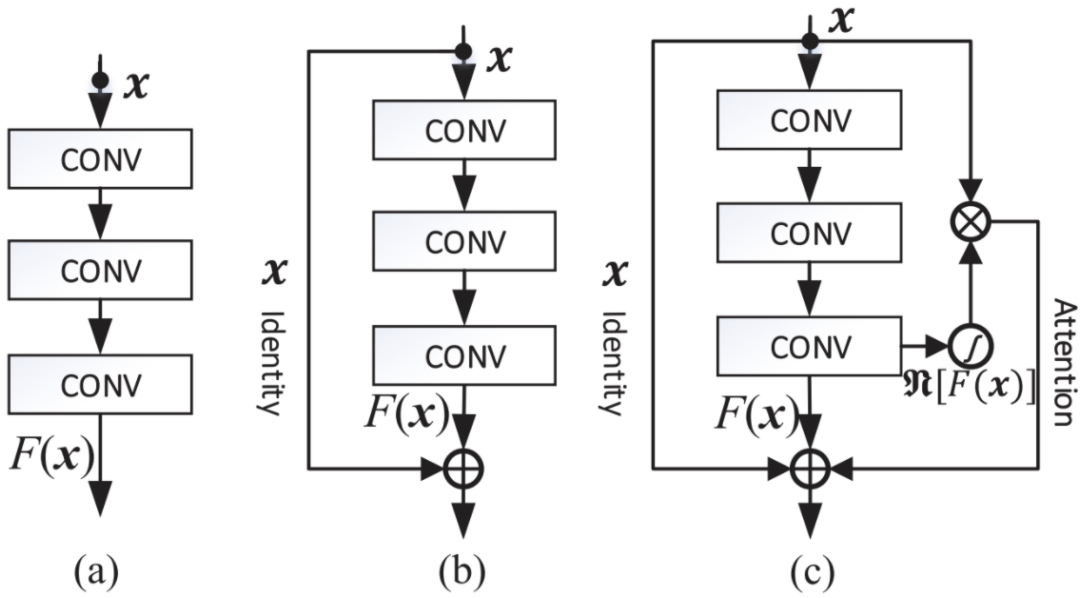

Fig. 2.17: Comparación entre bloques distintos bloques utilizados en las redes neuronales profundas. (a) representa un bloque normal, (b) un bloque con la *skip-connection* típico de *resnets* y (c) representa el bloque con la *skip-connection* y el mecanismo de *attention residual learning* agregado [8].

2.12.2. Squeeze & Excitation

Squeeze & Excitation es una optimización que aplica la arquitectura EfficientNet a su modelo base. Dicha optimización apareció primero con las redes que llevaron su nombre, las Squeeze & Excitation Networks o SENets [26].

Su propósito consiste en dar una manera sencilla de aprovechar posibles relaciones entre los canales de las activaciones de la red para mejorar el desempeño de la misma. Como su nombre indica, este método consta de dos partes: Squeeze y Excitation.

Squeeze

Para poder trabajar con los canales de las activaciones sin incurrir en un gran costo computacional, se genera un *descriptor* por cada canal. Dicho *descriptor* consiste en un número z_c que se obtiene al calcular el promedio del canal a lo largo del alto y el ancho, también conocido como *Global Average Pooling*. Esto se puede definir formalmente en la siguiente ecuación donde $\mathbf{u}_c(i,j)$ representa el valor en la posición (i,j) del canal c del input $\mathbf{u}$, $\mathbf{F}$ es la función que realiza el *squeeze* y z_c es el escalar obtenido para el canal c al cual llamaremos *descriptor*.

$$z_c = \mathbf{F}_{sq}\left(\mathbf{u}_c\right) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \mathbf{u}_c(i,j)$$

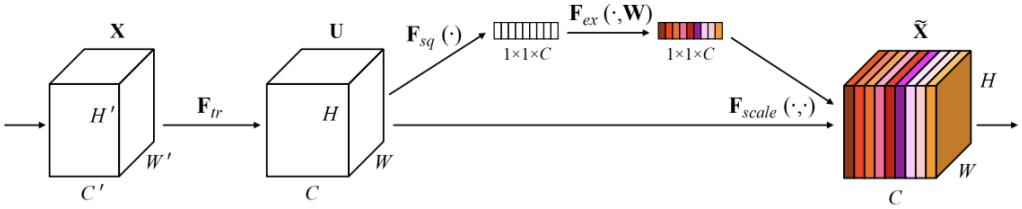

Fig. 2.18: Ilustración del accionar de Squeeze & Excitation. Luego del alguna transformación F_{tr} (como podría ser un bloque convolucional) el mecanismo comienza por generar descriptores con el proceso de *squeeze* representado por F_{sq}, luego determina cuáles de ellos son los relevantes con el proceso de *excitation* representado por F_{ex}, finalmente pesa los canales del input con los valores obtenidos en el proceso de *scale* representado por F_{scale} [9].

Excitation

Una vez que se cuenta con un *descriptor* z_c por cada canal c, se desea poder explotar las relaciones (posiblemente no lineales) que existan entre ellos. Por tal motivo, los descriptores son alimentados a una capa *feedforward* que reduzca su dimensión en un ratio r seguida por una *non-linearity* (usualmente una capa ReLU) para terminar en otra capa *feedforward* que los retorne a su dimensión original.

Este proceso se describe formalmente de la siguiente manera, donde $\mathbf{z}$ es un vector de descriptores (cada uno obtenido por medio de aplicar la función *squeeze* a un canal del input), δ es la capa ReLU, W_1, W_2 son las capas feedforward y $\mathbf{s}$ es el vector que contiene los pesos finales por los cuales "exitar" (multiplicar) a cada canal.

$$\mathbf{s} = \mathbf{F}_{ex}(\mathbf{z}, \mathbf{W}) = \sigma(g(\mathbf{z}, \mathbf{W})) = \sigma\left(\mathbf{W}_2 \delta\left(\mathbf{W}_1 \mathbf{z}\right)\right)$$

Una vez que se cuenta con los pesos $\mathbf{s}$ para "excitar" a los canales, la tarea consiste en sencillamente multiplicar cada escalar $\mathbf{s}_c$ por el canal $\mathbf{u}_c$. Recordando que $\mathbf{s}$ pertenece a $\mathbb{R}^{1\times1\times C}$ mientras que u pertenece a $\mathbb{R}^{H\times W\times C}$, esta multiplicación da por resultado el output $\hat{\mathbf{X}} \in \mathbb{R}^{H\times W\times C}$

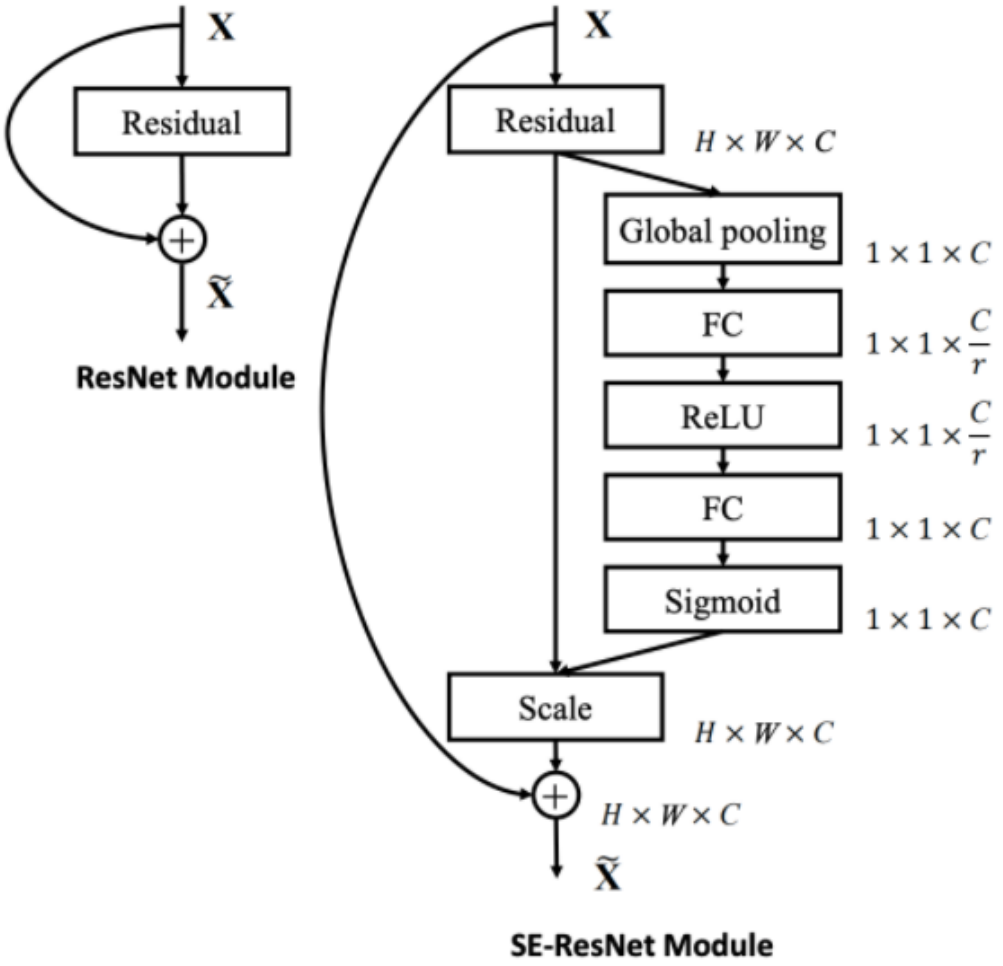

Fig. 2.19: Comparación entre un bloque residual normal y uno con *Squeeze & Excitation*. En la figura de la derecha podemos ver como la atención en el espacio de los canales producida por el mecanismo *Squeeze & Excitation* se combina con la *skip-connection* propia de los bloques residuales. En la ilustración *Global pooling* representa la etapa de *squeeze* a la que nos referimos anteriormente, *FC* hace referencia a una capa *fully-connected*, *ReLU* y *Sigmoid* representan a la función no lineal que lleva su mismo nombre respectivamente y finalmente *Scale* hace referencia a la manera de pesar los canales con los descriptores obtenidos del proceso de *excitation*, dando por resultado el output $\tilde{\mathbf{X}}$ [9].

2.12.3. Inserción del mecanismo de ARL a un modelo EfficientNet

El mecanismo de attention ARL que implementaremos sobre EfficientNet-b0 se aplica donde sucede la skip-connection. La skip-connection solo aparece en los bloques MBConv los cuales conforman 16 de las 18 capas. Sin embargo, la skip-connection no puede aplicarse en las 16 capas existentes ya que 7 de ellas duplican la cantidad de canales del output con respecto al input (haciendo imposible coincidir dimensiones de entrada y salida). Es por ello que aplicaremos el mecanismo ARL en tan solo $16-7=9$ capas y tendremos entonces 9 nuevos parámetros α a entrenar. Esto se puede ver con claridad en la tabla 2.2 donde la columna ARL marca con el símbolo '$\star$' las capas específicas que tienen el mecanismo de

ARL incorporado.

Etapa i	Operador $\hat{\mathcal{F}}_i$	Resolución $\hat{H}_i \times \hat{W}_i$	#Canales $\hat{C}_i$	ARL
1	Conv3x3	224×224	32	
2	MBConv1, k3x3	112×112	16	
3	MBConv6, k3x3	112×112	24	
	MBConv6, k3x3	112×112	24	$\star$
4	MBConv6, k5x5	56×56	40	
	MBConv6, k5x5	56×56	40	$\star$
5	MBConv6, k3x3	28×28	80	
	MBConv6, k3x3	28×28	80	$\star$
	MBConv6, k3x3	28×28	80	$\star$
6	MBConv6, k5x5	14×14	112	
	MBConv6, k5x5	14×14	112	$\star$
	MBConv6, k5x5	14×14	112	$\star$
7	MBConv6, k5x5	14×14	192	
	MBConv6, k5x5	14×14	192	$\star$
	MBConv6, k5x5	14×14	192	$\star$
	MBConv6, k5x5	14×14	192	$\star$
8	MBConv6, k3x3	7×7	320	
9	Conv1x1 & Pooling & FC	7×7	1280	

Tab. 2.2: Modelo EfficientNet-B0. Cada fila describe una capa de tipo $\hat{\mathcal{F}}_i$, input de resolución $\langle \hat{H}_i, \hat{W}_i \rangle$ y $\hat{C}_i$ canales de salida. La última columna indica con '$\star$' la capa donde es posible la inserción del mecanismo de ARL.

Es deseable poder agregar el mecanismo de ARL a modelos ya entrenados de modo obtener los beneficios de la técnica de transfer learning.

Para ello debemos ser capaces de realizar un cambio en la arquitectura de la red, sin perder los valores de los parámetros del modelo ya entrenado.

Afortunadamente, Python permite hacer esto al ser un lenguaje interpretado y dinámicamente tipado. El proceso consiste en seleccionar a todos los bloques en donde se desee agregar ARL y realizar los siguientes cambios:

- Adicionar un parámetro entrenable que cumplirá la función de alpha
- Sobreescribir el método de instancia forward — que es el que se ejecuta al hacer el forward-pass en el entrenamiento— por uno que agregue el mecanismo de ARL al bloque.

3. DATASETS Y PREPARACIÓN DE DATOS

3.1. Datasets utilizados

Los datasets empleados en este trabajo surgen gracias al trabajo de la International Skin Imaging Collaboration (ISIC) [1]. Año tras año, dicha institución publica un nuevo compendio de imágenes de lesiones de piel debidamente etiquetadas para su uso en diversas tareas. La ISIC facilita datasets que pueden ser utilizados para tareas de segmentación, clasificación y detección y localización de características relevantes de lesiones.

En el presente trabajo se emplean:

- El datasets del año 2017 para clasificación, con motivo de comparar resultados con papers anteriores
- El dataset de segmentación del año 2018 con motivo de construir un segmentador de imágenes que permita ayudar en la tarea a redes de clasificación

El dataset de 2017 para clasificación consta de un set de entrenamiento con 2000 imágenes, de las cuales 374 corresponden a melanoma, 254 a seborrheic keratosis y las restantes 1372 como nevi benignos. A la vez consta de un dataset para validación con 150 imágenes de las cuales 30 corresponden a melanoma 42 a seborrheic keratosis y las restantes 78 a nevi benignos.

El dataset de segmentación del año 2018 contiene 2594 imágenes y sus correspondientes máscaras de *ground truth* que indican para cada píxel si este forma parte o no de la lesión.

3.2. Métodos de preprocesamiento

Segmentación del área de la lesión

Las imágenes del dataset están tomadas haciendo foco en la lesión a clasificar. Sin embargo, buena parte de la imagen está abarcada por piel circundante que no forma parte de la lesión en sí. Es por ello que surge la idea de segmentar la imagen, dividiendo qué porción de la imagen corresponde a la lesión en sí y qué porción es piel circundante. Esto se realiza con la idea de que dicha segmentación le facilitará a la red saber que en qué sección de la imagen concentrarse.

Para poder segmentar las imágenes hemos utilizado otra red neuronal que sirve para dicha tarea: la U-Net [27].

La U-Net es una red que tiene la particularidad de poder producir un output de las mismas dimensiones que el input. Esto es ideal para tareas de segmentación donde se desea saber a qué clase corresponde cada píxel de la imagen. En el presente caso se busca saber para cada píxel si este corresponde a la clase *lesión* o si el mismo pertenece a la clase *no-lesión* (también entendible como tejido circundante).

El nombre de la red proviene de la forma de U que toma la ilustración de su arquitectura 3.1. La primera mitad de la red podría verse como una red convolucional típica: lleva un input desde el espacio de imágenes a una representación con gran cantidad de

[1] https://challenge.isic-archive.com/

canales y reducida dimensión espacial. En efecto, en la práctica es usual utilizar alguna familia de las redes *resnet* para realizar esta primera parte. La segunda mitad de la red U-Net es responsable de realizar el *upsampling* de las dimensiones para poder producir una salida con las mismas dimensiones que la imagen original. Para ello la red utiliza *up-convolutions* [2] a cuyos outputs se les concatenan las activaciones correspondientes de la etapa de *downsampling*. Esto se ilustra mejor en la figura 3.1.

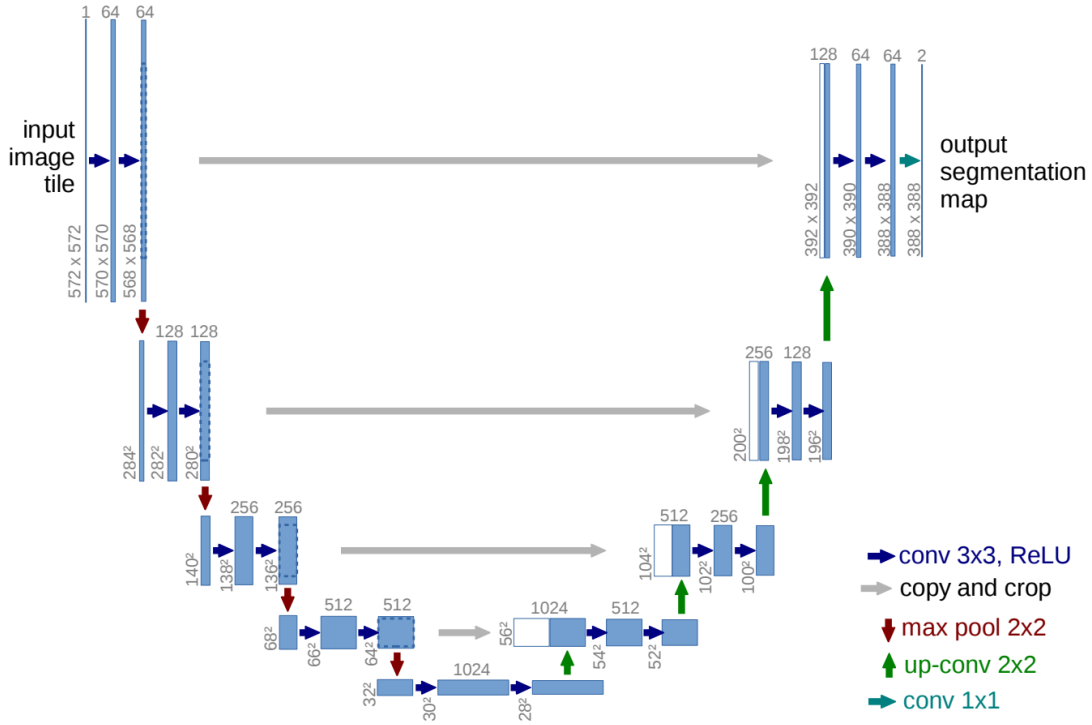

Fig. 3.1: Arquitectura de una red U-Net. Cada recuadro azul corresponde a un set de activaciones. Por encima de ellos se ubica el número de canales de los mismos, mientras que al costado se ubican alto y ancho. Las flechas entre los recuadros denotan las distintas operaciones [27].

Para poder utilizar la U-Net es necesario entrenarla con un dataset acorde. Afortunadamente, la competencia de ISIC también provee un dataset específicamente diseñado para dicha tarea [3].

El dataset consiste en las imágenes originales e imágenes segmentadas por expertos en el área. Por ejemplo, se puede ver en la figura 3.2 el par que corresponde a una imagen y a su segmentación.

[2] Los autores llaman *up-convolutions* a operaciones de *upsampling* seguidas por convoluciones con kernels de tamaño 2×2

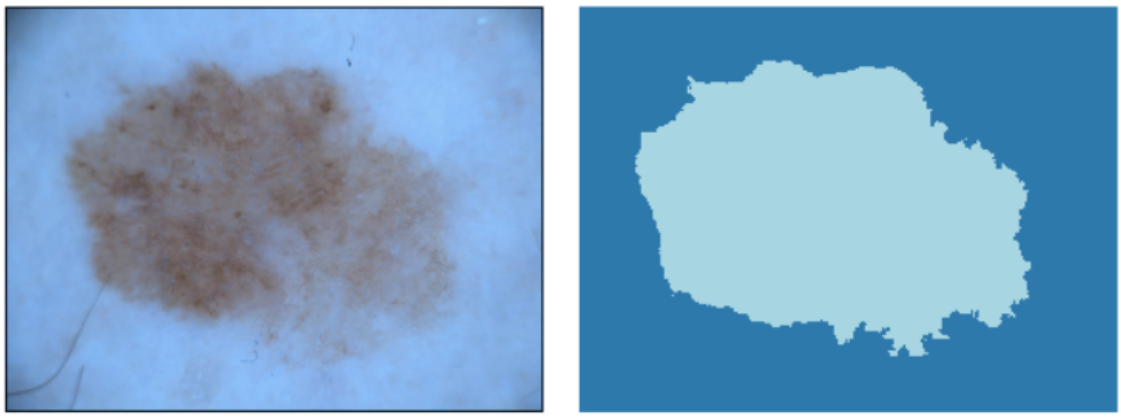

Fig. 3.2: Imagen de una lesión junto a la máscara que delimita qué parte constituye la lesión en sí.

Hemos redimensionado el dataset a que tenga un tamaño de 224×224, por medio de escalar las imágenes preservando su *aspect ratio* hasta que su lado menor sea de 224 píxeles, y luego realizando un *center-crop* de manera que ambos lados sean de 224 píxeles.

En la figura 3.3 se muestran imágenes luego de ser redimensionadas a 224×224 donde hemos aplicado la segmentación como una máscara de tinte azul sobre la parte de la imagen que no constituye lesión.

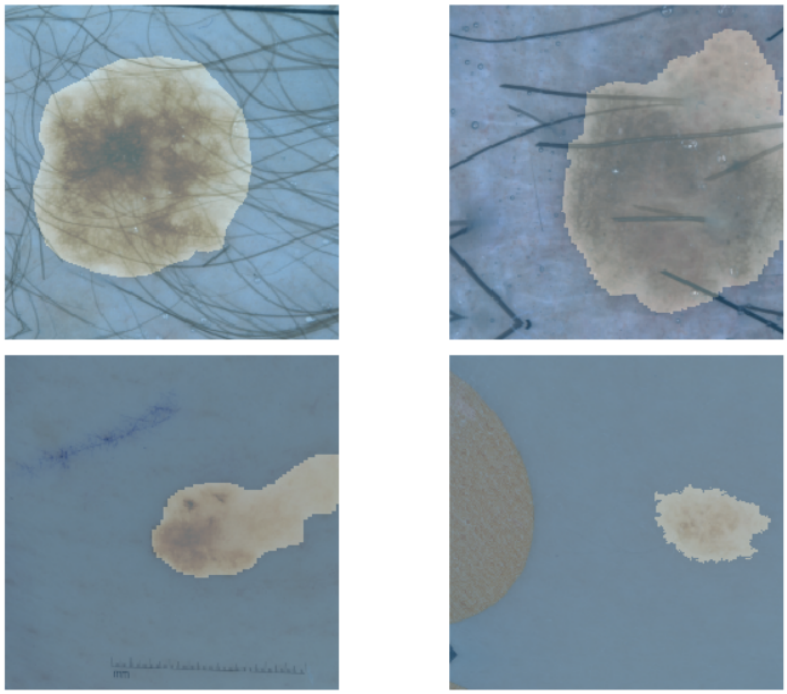

Fig. 3.3: Set de entrenamiento: lesiones con la máscara aplicada. La región de interés se muestra con sus colores originales, mientras que en tonalidad azul se muestra la parte que no pertenece a la lesión.

Utilizando una U-Net preentrenada en el dataset COCO [28], un batch-size de 8, un learning rate de 3e-4 para la cabeza de la red y 3e-5 para el cuerpo de la red, con un

scheduling que sigue la política de One Cycle [29] logramos dar con un *dice score* de 0.89 luego de entrenar la red por 12 épocas. Las primeras 2 entrenando solo la cabeza de la red, y las últimas 10 entrenando la red en su totalidad.

En la figura 3.4 se muestra la segmentación que produce nuestra red en imágenes del set de validación comparándola con la segmentación original.

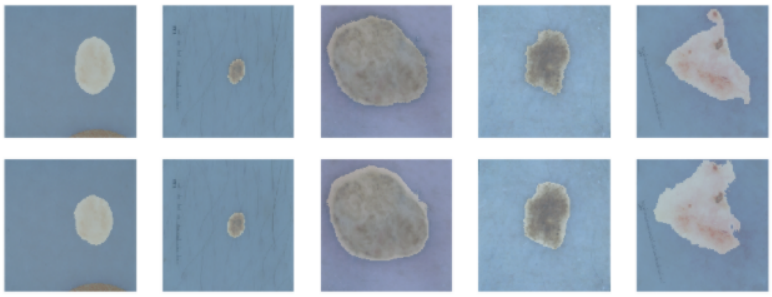

Fig. 3.4: Ejemplos de máscara original (fila superior) con la segmentación que produce nuestra red (fila inferior) en el set de validación.

Como se puede apreciar, si bien el resultado no es exactamente igual, la red logra captar y delimitar la sección que corresponde a la lesión respecto de la que corresponde a piel circundante.

Transformación de las imágenes a clasificar

Una vez que contamos con una red capaz de segmentar el área de la lesión en las imágenes, surge luego la cuestión de cómo transmitirle a la red de clasificación la información sobre la segmentación que hemos obtenido.

Hemos optado por aplicar una máscara en las imágenes de modo que deje pasar información de color solamente en las secciones que corresponden a la lesión. Respecto a las secciones que no forman parte de la lesión hemos pasado la imagen a escala de grises.

En la figura 3.5 se visualiza el resultado:

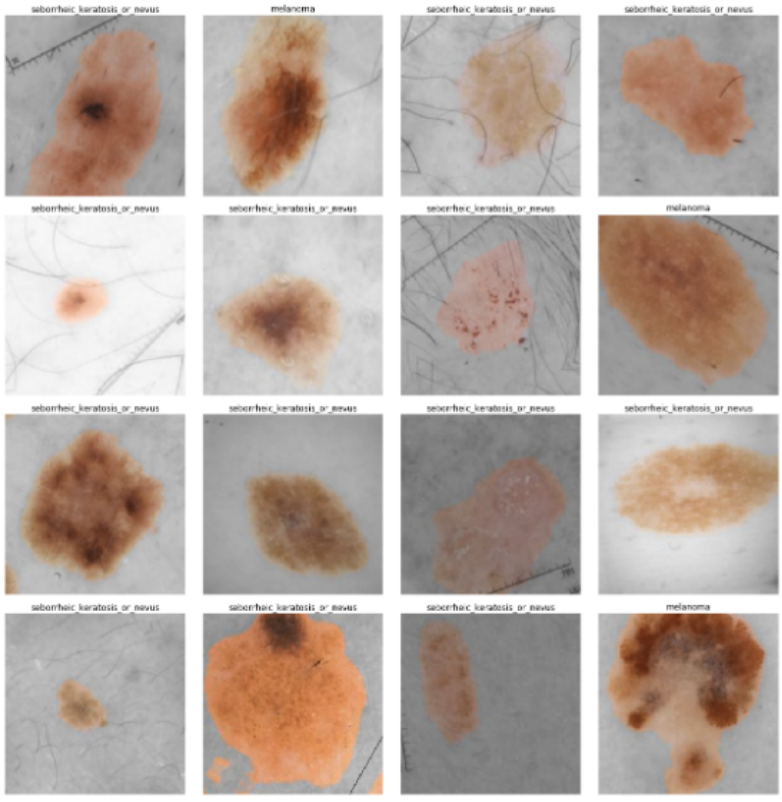

Fig. 3.5: Ejemplos de cómo la máscara permite el paso de la información de color.

3.2.1. Algoritmos de Corrección de Color / Color Constancy

El dataset de imágenes de lesiones de piel que utilizamos proviene de diversos centros clínicos Esta diversidad introduce ruido proveniente de la variedad en los elementos para capturar las imágenes tanto así como de las diferencias en las condiciones de iluminación.

Por tanto, la intención de aplicar los métodos de preprocesamiento de color es lograr una mayor uniformidad en las imágenes —pero sin descartar información valiosa y particular de cada una— lo cual le facilite la tarea de clasificación a las redes neuronales.

Los métodos de *color constancy* consisten transformar los colores de una imagen que ha sido capturada bajo una fuente de luz desconocida, de modo de que la imagen parezca obtenida bajo una fuente de luz canónica. Se suele asumir que esta fuente de luz es la luz blanca perfecta [7].

La implementación de esta transformación consiste en dos pasos:

- En el primero es necesario estimar la fuente de luz bajo la que fue tomada la imagen. A esta fuente se la denomina *estimated iluminant* y se la representa mediante un vector $[e_R\ e_G\ e_B]^T$ donde cada posición refleja la intensidad de cada canal de la supuesta fuente de luz.

- En el segundo, se debe utilizar la fuente de luz obtenida del primer paso para recalibrar los colores de la imagen.

Primer paso: Estimación de la fuente de luz

Respecto del primer paso, para estimar la fuente de luz con la que fue tomada una imagen a color existen diversos algoritmos. En nuestro caso abordaremos los algoritmos de Max-RGB y Shades of Gray. Sea I una imagen a color, luego cada componente del *iluminant* e_c, $c \in \{R, G, B\}$ se estima según:

Max-RGB: El algoritmo de *Max-RGB* forma el *estimated iluminant* mediante seleccionar para cada canal de una imagen el valor máximo que figura en el.

$$\max_{\mathbf{x}} I_c(\mathbf{x}) = ke_c$$

Shades of Gray: El algoritmo de *Shades of Gray* forma el estimated iluminant mediante la siguiente ecuación cuyo parámetro p es libre de ser elegido.

$$\left(\frac{\int (I_c(\mathbf{x}))^p dx}{\int dx}\right)^{1/p} = ke_c$$

En las ecuaciones antes descritas, I_c representa el canal c de la imagen; $\mathbf{x} = (x, y)$ es la posición espacial del píxel en cuestión y k es una constante de normalización para que el vector tenga norma 1.

Segundo paso: Recalibrar los colores de la imagen
Una vez que se cuenta con el vector $\mathbf{e} = [e_R e_G e_B]^T$, se puede recalibrar la imagen mediante dividir cada canal I_c por su respectiva componente e_c y posteriormente multiplicarlo por $1/\sqrt{3}$. La división por la componente del *estimated iluminant* supondría la quita de la fuente de luz desconocida, mientras que la multiplicación por $1/\sqrt{3}$ supondría aplicar la fuente de luz canónica. Si nombramos I_c^t al canal c transformado de la imagen, entonces este se obtiene mediante:

$$I_c^t = I_c \times \frac{1}{\sqrt{3}e_c}$$

Una vez que se cuenta con la forma de aplicar *color constancy*, es necesario elegir el lugar correcto dónde implementarla. En este trabajo se opta por realizar la transformación de color constancy de manera previa a toda otra transformación de *data augmentation*.

3.2.2. Preprocesamiento de Ben Graham

El Preprocesamiento de Ben Graham surge del ganador de la competencia de retinopatía diabética en la plataforma Kaggle [30].

El método consiste en aplicar una serie de transformaciones a la imagen que otorga como resultado filtrar las frecuencias bajas de la misma. Para ello el método utiliza los siguientes pasos:

Comienza por crear una copia de la imagen original y aplicar un *Blur Gaussiano*, con un tamaño de kernel que se define automáticamente a partir de la varianza de la función de densidad de la campana de Gauss. Se toma una varianza igual al resultado de dividir el tamaño de la imagen (en nuestro caso 224) por 10. Luego se continúa restándole a la

imagen original, la imagen procesada (eliminando así las frecuencias bajas de la imagen original).

Al resultado se lo escala, mediante multiplicarlo por 4 y sumar 128.

Esto se puede ver formalmente en la ecuación 3.1 en la cual I_{in} representa a la imagen de entrada, I_{out} a la de salida y G es el kernel de convolución con el cual se hace blur de la imagen.

$$I_{out} = 4(I_{in} - G * I_{in}) + 128 \tag{3.1}$$

Se entiende que el procesamiento de Ben-Graham hace un proceso cercano a eliminar las frecuencias bajas de la imagen: al hacer blur a una imagen, quedan solo las frecuencias bajas de la misma; al restarle a una imagen una copia de si misma con blur, se le restan las frecuencias bajas, por tanto le quedan solo las altas.

Al eliminar las frecuencias bajas de la imagen, el resultado de aplicar el preprocesamiento de Ben Graham produce los siguientes efectos: atenúa los efectos de las diferencias entre las fuentes de luz con las que fueron tomadas las imágenes y disminuye las diferencias entre los tonos de piel de los pacientes.

3.2.3. Visualizando las transformaciones

En las figuras 3.6, 3.7 se muestra la diferencia entre un conjunto de imágenes elegidas aleatoriamente, antes y después de aplicar la transformación utilizando los algoritmos de color constancy con Max-RGB o con Shades of Gray, o bien aplicando el preprocesamiento de Ben-Graham.

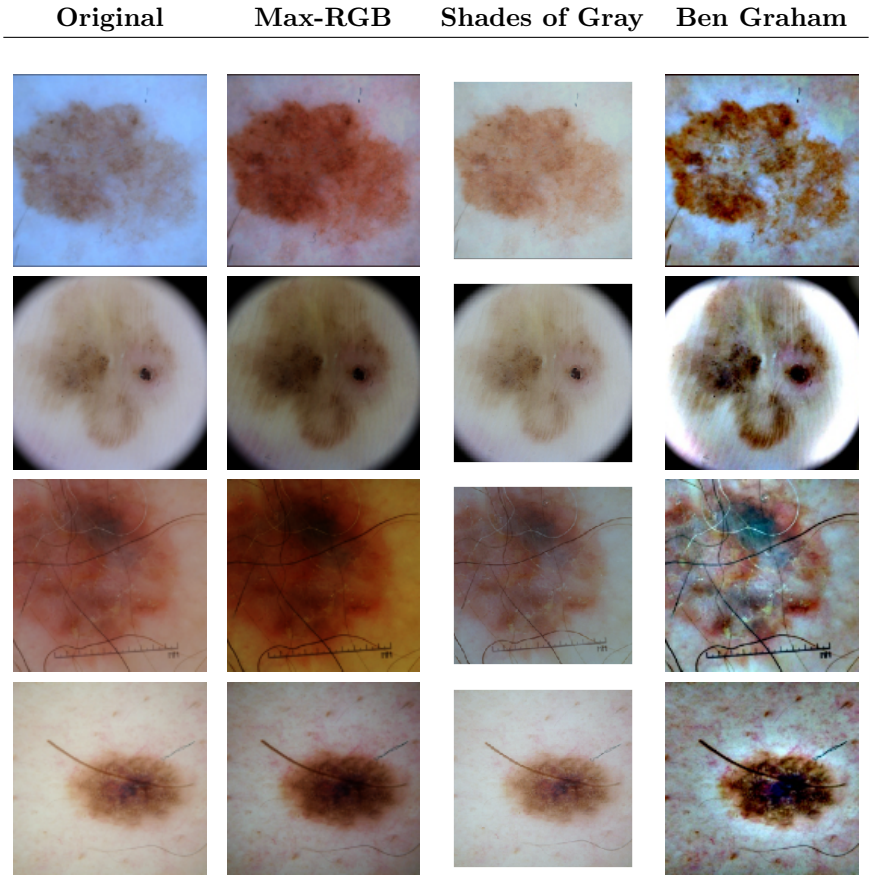

Fig. 3.6: Comparación entre distintos métodos de preprocesamiento para imágenes correspondientes a Melanoma.

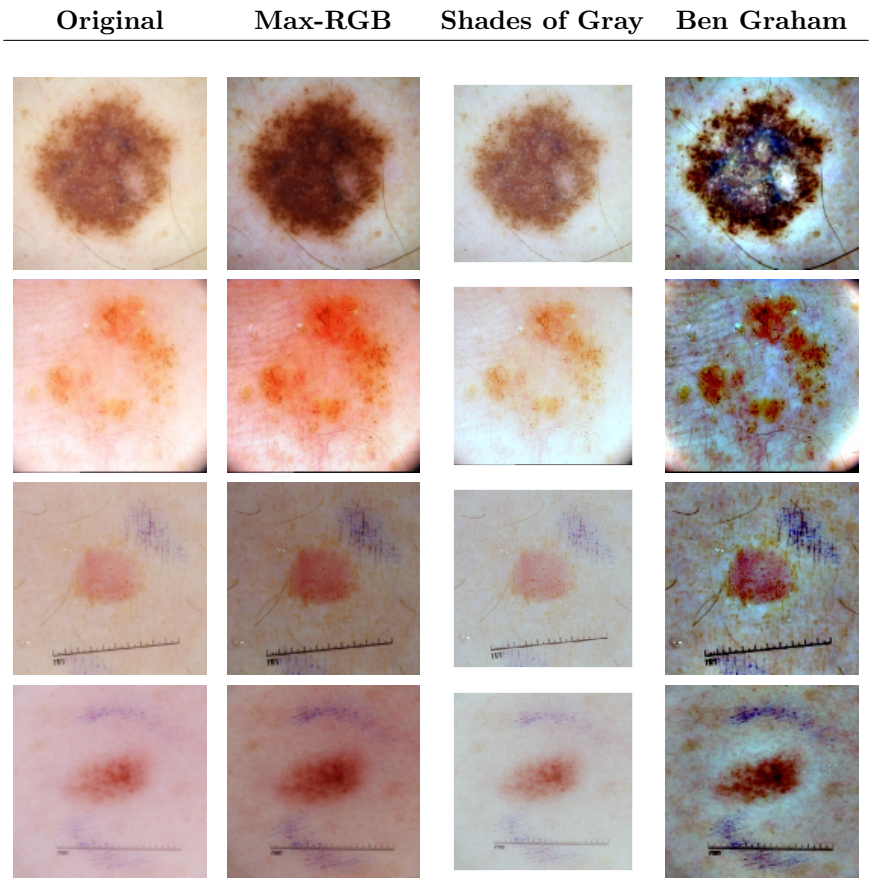

Fig. 3.7: Comparación entre distintos métodos de preprocesamiento de imágenes que no son Melanoma.

4. EXPERIMENTACIÓN

4.1. Experimento 1: Segmentación del área de la lesión

4.1.1. Hipótesis

Las fotografías de lesiones de piel son muy variadas. Estas se diferencias por diversos motivos, ya sea por el ángulo con el que está tomada la foto, el tono de la piel, la presencia o no de vello en la misma, etc...

El supuesto es que estas diferencias entre las imágenes —ajenas a la lesión en sí— introducen ruido que dificulta la tarea a la red de clasificación.

Por ende en este experimento, se crea un nuevo dataset en el cual intenta remover información —mediante eliminar la información de color— de toda parte de la imagen que no forme parte de la lesión en sí.

Para la creación de dicho nuevo dataset se entrena una red de segmentación con el objetivo de delinear qué porción de la imagen pertenece a la lesión en sí y que porción no. Esta segmentación permite luego eliminar fácilmente la información de color la sección que no pertenece a la lesión en sí.

Luego el experimento consistirá en comparar si la red de clasificación clasifica mejor en el dataset original, o en el dataset en el cual solo hay información de color en el área de la lesión.

4.1.2. Preparación

Se trabaja sobre el dataset de 2017 el cual se duplica y se segmenta. De esta manera se cuenta con 2 datasets cuya única diferencia es que el segundo fue segmentado por medio de una U-NET como se describe en la sección 3.2. Es decir, para cada imagen se obtiene la máscara de segmentación que delimita el área de la lesión y únicamente se permite información de color allí dentro.

Las imágenes tienen un tamaño de 224×224 píxeles y se utiliza batch size de 16. Se aplican las siguientes técnicas de data augmentation: rotación de 180 grados para ambos lados y zoom de hasta 30 % en distintas regiones de la imagen.

Para cada dataset se entrena una ResNet-50 preentrenada en ImageNet. Para ello, el entrenamiento se realiza en 2 etapas. En la primera se entrena solo una nueva cabeza del modelo que tiene como tamaño de salida un vector de largo 2 correspondiente a las 2 clases que se intenta predecir —melanoma u otros—. Este entrenamiento se realiza por 4 épocas, aplicando un learning rate con política One Cycle [29] de máximo 3e-3.

En la segunda etapa, se entrena al modelo entero por 20 épocas, aplicando la política One Cycle con máximo de 3e-4.

Este proceso se repite entero 10 veces —cada vez con un seed distinto— de manera de poder contar con datos robustos.

4.1.3. Resultados e interpretación

La tabla 4.1 se forma tomando el máximo *accuracy* alcanzado por cada corrida y luego calculando la media, desvío estándar y cuartiles de ellos.

Dataset	mean	std	25 %	50 %	75 %	max
Baseline	**0.876667**	**0.008462**	**0.873333**	**0.880000**	**0.880000**	**0.886667**
Segmentado	0.846000	0.011089	0.840000	0.843333	0.851667	0.866667

Tab. 4.1: Resultados para el experimento 1. Se muestran media, desvío estándar y cuartiles para 10 corridas con el dataset original y el segmentado.

Como se puede observar en la tabla de resultados, el dataset de control ("Baseline") obtiene mejores resultados que el dataset segmentado. Esto da a entender que el método de dejar pasar información de color solamente en el área cercana a la imagen perjudica al entrenamiento de la red neuronal.

Quizás el motivo de esto se deba a que la red neuronal —en el dataset original— se vale de más características de la imagen que la lesión en sí (como pueden ser el tono de los colores, la luminosidad de la imagen, la presencia o no de vello) que son destruidas al segmentar la imagen en el área de la lesión.

Otro motivo a considerar sería suponer que las variaciones de la forma y el tamaño de la porción segmentada en la imagen introducen ruido de por sí y dificultan la tarea a la red neuronal.

Un tercer supuesto que se puede plantear es que la tarea de segmentación no fue hecha con la precisión necesaria y que los errores en esta etapa perjudicaron a la etapa de clasificación. Sin embargo, un contra-argumento que se puede plantear a este supuesto es que la precisión que se busca en la tarea de segmentación no requiere de los mismos niveles de precisión que otras tareas de segmentación como podría ser el dataset COCO [28], ya que en el presente caso solo se busca delimitar una región que contenga a la lesión pero no se requiere precisión perfecta a nivel pixel —como si se requiere a la hora de segmentar a un ciclista en una fotografía de una avenida—.

Un cuarto supuesto podría plantearse en la base de que el método de transferencia de la información de segmentación a la etapa de clasificación no fue óptimo. En este experimento se ha optado por eliminar la información de color de toda área que no esté alcanzada por la máscara de segmentación, pero bien se podría haber optado por otros métodos. Basándose en esta idea también se ha intentado reemplazar todo pixel que se encuentre por fuera del área de la lesión con un tono sólido de gris constante(128,128,128). Sin embargo, este cambio ha arrojado resultados similares al experimento en cuestión y no pareciera merecer más indagaciones. A pesar de ello, la búsqueda de un buen método de transferencia de la información de segmentación a la etapa de clasificación queda abierta y se considera para trabajos futuros.

4.2. Experimento 2: Impacto de algoritmos de corrección de color en la imagen

4.2.1. Hipótesis

En concordancia al supuesto del experimento 1, se cree que las diferencias en la luminosidad con las que fueron tomadas las imágenes introduce ruido que dificulta la tarea de la red de clasificación.

Por ende en este experimento se intenta verificar si la aplicación de algoritmos de constancia de color —*Max RGB* y *Shades of Gray*— en el proceso de homogeneizar las imágenes del dataset, también le facilite la tarea a la red de clasificación.

A su vez, además de los algoritmos de constancia de color *Max RGB* y *Shades of Gray*, se intenta con un tercero: el preprocesamiento de Ben Graham. Este preprocesamiento no constituye *per–se* un algoritmo de constancia de color, si no más bien algo cercano a un filtro pasa altos. Sin embargo en el proceso de filtrar las frecuencias bajas (por ser un filtro pasa altos) se cree que también se estarán atenuando las diferencias que introducen las distintas iluminaciones al haber sido tomadas las imágenes. Por lo cual un filtro pasa altos es quizá todo lo que se necesita —o esto es lo que se intentará verificar en el experimento— para eliminar el ruido proveniente de distintas iluminaciones a la hora de tomar las imágenes.

De modo que en resumidas cuentas tendremos 3 algoritmos —Max RGB, Shades of Gray y el preprocesamiento de Ben Graham— e intentaremos ver cuál de ellos le facilita más la tarea clasificación a la red neuronal. En este experimento se medirá tanto *accuracy* como *auroc*. Agregamos la medida de esta segunda métrica para respaldar los datos obtenidos con las mediciones de *accuracy*. A la vez, la medición de *auroc* para clasificadores binarios (melanoma versus no melanoma) permite dar una noción no solo de cuán buenos son detectando el melanoma cuando existe (true positive rate) si no también de cuán malos son al diagnosticar falsamente la presencia de melanoma cuando la misma no estaba presente (false positive rate).

4.2.2. Preparación

Se trabaja sobre el dataset de 2017. Las imágenes tienen un tamaño de 224 x 224 píxeles. Se aplican las siguientes técnicas de data augmentation como base: rotación de 180 grados para ambos lados, zoom de hasta 30 % en distintas regiones de la imagen y alteraciones en la luminosidad. Se utiliza un batch size de 16 y se hace uso de Oversampling como mecanismo para solventar el desbalance de clases.

Para cada algoritmo de correción de color —Max RGB y Shades of Gray— se crea una etapa de preprocesamiento en donde a cada imagen se le corrije la luminosidad de acuerdo al mismo. Para el preprocesamiento de Ben Graham, se crea un nuevo dataset copia del original con la técnica aplicada.. Se entrenan 4 redes ResNet-50 Pre Entrenadas en ImageNet. La primera servirá como datos de control para comparación, la segunda se entrenará sobre el dataset corregido con Max-RGB, la tercera sobre el dataset corregido con Shades of Gray y la cuarta sobre el dataset preprocesado con el método Ben Graham.

Para ello, el entrenamiento se realiza en 2 etapas. En la primera se entrena solo una nueva cabeza del modelo que tiene como tamaño de salida un vector de largo 2 correspondiente a las 2 clases que se intenta predecir —melanoma u otros—. Este entrenamiento se realiza por 4 épocas, aplicando un learning rate con política One Cycle [29] de máximo

3e-3. En la segunda etapa, se entrena al modelo entero por 20 epocas, aplicando la política One Cycle con máximo de 3e-4.

Este proceso se repite entero 10 veces —cada vez con un seed distinto— de manera de poder contar con datos robustos.

4.2.3. Resultados e interpretación

Dataset	mean	std	25 %	50 %	75 %	max
Baseline	0.874000	**0.006630**	0.868333	0.873333	0.878333	0.886667
Max RGB	0.876667	0.009558	0.868333	0.880000	0.885000	0.886667
Shades of Gray	0.866667	0.008889	0.866667	0.866667	0.871667	0.880000
Ben Graham	**0.887333**	0.012746	**0.881667**	**0.886667**	**0.893333**	**0.906667**

Tab. 4.2: Resultados de accuracy para el experimento 2. Se muestran media, desvío estándar y cuartiles para 10 corridas con el dataset sin ninguna corrección de color, el dataset procesado con el algoritmo de *Max RGB*, el dataset procesado con el algoritmo de *Shades of Gray* y el dataset procesado con el algoritmo de *Ben Graham*.

Dataset	mean	std	25 %	50 %	75 %	max
Baseline	0.885389	0.014831	0.880555	0.889861	0.894861	0.905000
Max RGB	0.891306	0.011410	0.882153	0.891528	0.900695	0.908333
Shades of Gray	0.883333	0.013051	0.872361	0.883055	0.895556	0.899167
Ben Graham	**0.898667**	0.016924	**0.883681**	**0.900833**	**0.907153**	**0.928333**

Tab. 4.3: Resultados de auroc para el experimento 2. Se muestran media, desvío estándar y cuartiles para 10 corridas con el dataset sin ninguna corrección de color, el dataset procesado con el algoritmo de *Max RGB*, el dataset procesado con el algoritmo de *Shades of Gray* y el dataset procesado con el algoritmo de *Ben Graham*.

La tablas 4.2 y 4.3 se forman tomando el máximo *accuracy* y *auroc* respectivamente alcanzado por cada corrida y luego calculando la media, varianza y cuartiles de ellos. Como se puede observar el entrenamiento que utiliza el método de Ben Graham supera a todos los otros métodos en ambas métricas, otorgando un *accuracy* máximo promedio entre todas las corridas de 0.8873, y un *auroc* máximo promedio entre todas las corridas de 0.8986, mientras que el entrenamiento sin ningún procesamiento obtiene 0.8740 y 0.8854 en *accuracy* y *auroc* respectivamente.

Así mismo, el método de Max RGB también parece impactar positivamente —aunque en menor medida que el método de Ben Graham— en los resultados con *accuracy* y *auroc* máximo promedio entre las corridas de 0.8767 y 0.8913 respectivamente. Por su parte, el método de Shades of Gray parece en este caso perjudicar la performance.

Respecto a la razón de por qué el método de Ben Graham otorga tan buenos resultados, se puede hipotetizar que se debe a que su acción de filtrar las frecuencias bajas de la imagen da lugar a que la red pueda concentrarse en las frecuencias más altas de la misma. Es decir, permite a la red hacer foco en los detalles más finos de las lesiones como por ejemplo pueden ser los bordes de la misma.

Como un segundo motivo, se puede ver en la tabla comparativa que el resultado de aplicar el método de Ben Graham también logra cierto efecto de corrección de color:

todas las imágenes ahora se encuentran inclinadas hacia tintes azules y marrones. Es decir la información global del tono de la imagen se homogeniza junto a la eliminación de frecuencias bajas de la misma.

4.3. Experimento 3: Implementación de ARL sobre ResNet-50

4.3.1. Hipótesis

En el presente experimento se reproduce la implementación de ARL sobre ResNet-50 estudiando como su agregado mejora la performance de las redes tal como declara el trabajo previo [8].

El fin de este experimento es entonces sentar una fundación sólida para el próximo experimento, en el cual se hará este mismo cambio —la adición de ARL— a la red EfficientNet.

4.3.2. Preparación

Se trabaja sobre el dataset de 2017. Las imágenes tienen un tamaño de 224 x 224 píxeles. Se utiliza batch size de 16. Se aplican las siguientes técnicas de data augmentation como base: rotación de 180 grados para ambos lados y zoom de hasta 30 % en distintas regiones de la imagen. A la vez se hace OverSampling para suplir las irregularidades del desbalance en las clases. Notar que en este experimento —a diferencia de los anteriores— no se utilizan técnicas de data augmentation relacionadas con la alteración de la luminosidad (el propósito de esto es ser más fiel a las técnicas utilizadas en el paper).

Se entrenan 2 redes ResNet-50 preentrenadas en ImageNet. La primera servirá como baseline por comparación, mientras que a la segunda se le aplicará la modificación de ARL.

En total se modifican 16 capas, por lo cual se cuenta con 16 nuevos parámetros α a entrenar (ver ecuación 2.2). El entrenamiento se realiza en 2 etapas. En la primera se entrena solo una nueva cabeza del modelo que tiene como tamaño de salida un vector de largo 2 correspondiente a las 2 clases que se intenta predecir —melanoma u otros—. Este entrenamiento se realiza por 4 épocas, aplicando un learning rate con política One Cycle [29] de máximo 3e-3. En la segunda etapa, se entrena al modelo entero por 20 épocas, aplicando la política One Cycle con máximo de 3e-4.

4.3.3. Resultados e interpretación

Dataset	mean	std	25 %	50 %	75 %	max
Baseline	0.876667	**0.008462**	0.873333	0.880000	0.880000	0.886667
ARL	**0.880000**	0.009428	**0.875000**	**0.880000**	**0.885000**	**0.893333**

Tab. 4.4: La tabla se forma tomando el máximo *accuracy* alcanzado por cada corrida y luego calculando la media, varianza y cuartiles de ellos.

Como se puede observar en 4.4, luego de 10 corridas con distintos seeds iniciales, el *accuracy* máximo promedio alcanzado por el modelo que utiliza ARL supera al modelo de control.

A modo de constatar que el mecanismo de atención está efectivamente siendo utilizado por el modelo, en la figura 4.1 analizamos cómo se modifica el valor de los *alphas* a lo largo de las épocas de una de las corridas.

La figura 4.1 modela el valor de cada alpha a lo largo de las epochs. Los *alphas* se encuentran ordenados de izquierda a derecha y de arriba hacia abajo según su ocurrencia en un forward pass. Es decir el sub-gráfico de más arriba a la izquierda representa al

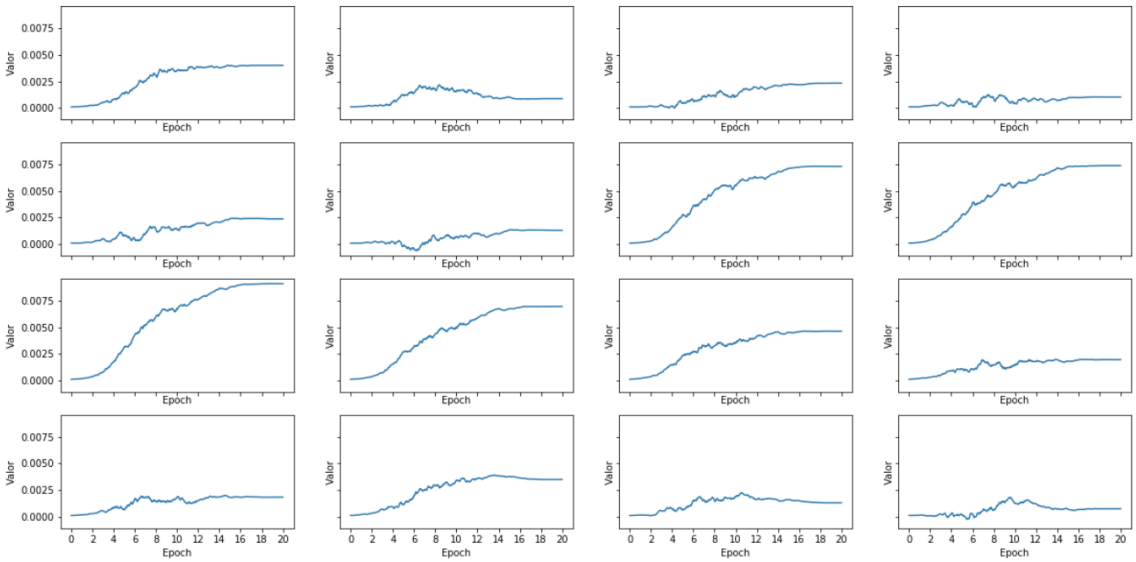

Fig. 4.1: Evolución de los parámetros alpha para cada capa bottleneck.

alpha de la primera capa *bottleneck* del modelo (aquella que recibe *features* de bajo nivel). Mientras que el sub-gráfico de abajo a la derecha representa al alpha de la última capa (la más cercana a la cabeza del modelo). Se puede observar que los *alphas* —y con ellos el mecanismo de atención— parece cobrar más valor en las capas intermedias. Especialmente se observa un crecimiento mayor de los alphas en el rango de capas bottleneck que se encuentran entre la sexta hasta la novena posición inclusive.

Esto nos asegura estar realizando una correcta implementación del algoritmo, lo cual nos permite avanzar al siguiente experimento de este trabajo: la adición de ARL a EfficientNet.

4.3.4. Análisis cualitativo con GradCAM

Adicionalmente a las métricas que permiten un análisis cuantitativo en la sección 4.3.3, es posible realizar un análisis cualitativo de los modelos mediante observar el resultado de *GradCAM* [31]. La técnica *GradCAM* permite visualizar mapas de calor sobre las imágenes para comprender qué secciones de las mismas influyen mayormente en la clasificación. En la figura 4.2 se expone una comparación entre los mapas de calor generados por el modelo original versus los generados por el modelo con ARL.

Si bien las diferencias entre los mapas de calor de los ejemplos de la figura 4.2 son sutiles, es posible visualizar que el modelo con ARL disminuye el área resaltada. Esto se condice con el concepto de *attention* que tiende a concentrar el foco de la red en las áreas que considera de mayor relevancia.

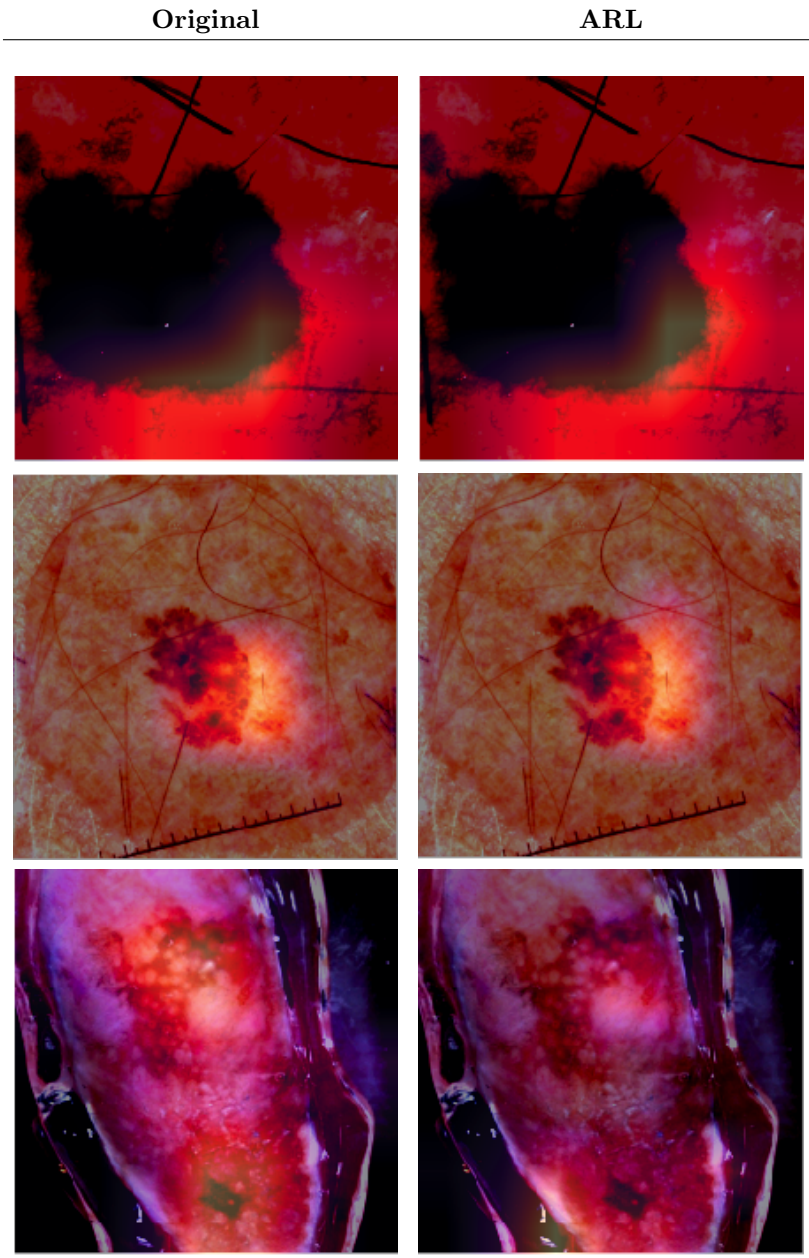

Fig. 4.2: Comparación de mapas de calor GradCAM para modelos con o sin ARL. Todos son ejemplos donde el modelo con ARL predijo correctamente la presciencia de melanoma mientras que el modelo original no lo hizo. Se puede observar en el segundo y tercer ejemplo como el modelo de ARL disminuye el área resaltada por el mapa de calor. Esto se condice con la *atención* que aplica el modelo al procesamiento de las imágenes. Respecto del primer ejemplo, aunque las diferencias sean notoriamente mas sutiles, el mapa de calor de la imagen con ARL parece centrarse más en el borde derecho de la lesión.

4.4. Experimento 4: Implementación de ARL sobre EfficientNet-b0

4.4.1. Hipótesis

En el presente experimento se intenta estudiar el efecto que produce agregar el mecanismo de ARL sobre los modelos *Efficient-Net*.

Específicamente, no solo interesa el agregado de el mecanismo al modelo de base, si no que también se estudia la manera en que se relaciona con el mecanismo de attention ya presente: *Squeeze & Excitation*.

Para ello estudiamos cómo se comportan las 4 variantes correspondientes a tener o no activados estos dos mecanismos de attention.

4.4.2. Preparación

Se trabaja sobre el dataset de 2017. Las imágenes tienen un tamaño de 224 x 224 píxeles. Se utiliza batch size de 16. Se aplican las siguientes técnicas de data augmentation como base: rotación de 180 grados para ambos lados y zoom de hasta 30 % en distintas regiones de la imagen. A la vez se hace OverSampling para suplir las irregularidades del desbalance en las clases

Se entrenan 4 redes EfficientNet-b0 preentrenadas en ImageNet. La primera servirá como baseline por comparación, a la segunda se le suprimirá el mecanismo Squeeze & Excitation, a la tercera se le suprimirá a la vez Squeeze & Excitation pero se le agregará ARL y finalmente la última contará con ambos mecanismos de attention.

En total se modifican 9 capas, por lo cual se cuenta con 9 nuevos parámetros alpha a entrenar. El motivo de esta diferencia con respecto a *ResNet-50* (donde hay 16 capas) es que en EfficientNet, no todas las capas de *inverted bottlenecks* hacen uso de la skip connection.

Similar a los entrenamientos anteriores, el entrenamiento se realiza en 2 etapas. En la primera se entrena solo una nueva cabeza del modelo que tiene como tamaño de salida un vector de largo 2 correspondiente a las 2 clases que se intenta predecir —melanoma u otros—. Este entrenamiento se realiza por 4 épocas, aplicando un learning rate con política One Cycle [29] de máximo 3e-3. En la segunda etapa, se entrena al modelo entero por 20 épocas, aplicando la política One Cycle con máximo de 3e-4.

4.4.3. Resultados e interpretación

Dataset	mean	std	25 %	50 %	75 %	max
Baseline(SE)	0.857333	0.007166	0.853333	0.860000	0.860000	0.866667
No attention	0.834667	0.007569	0.833333	0.833333	0.840000	0.846667
ARL	0.829333	0.012649	0.821667	0.833333	0.838333	0.846667
SE y ARL	**0.862000**	0.007730	**0.860000**	**0.863333**	**0.866667**	**0.873333**

Tab. 4.5: La tabla se forma tomando el máximo *accuracy* alcanzado por cada corrida y luego calculando la media, varianza y cuartiles de ellos.

Como se puede observar en 4.5 y 4.6, luego de 10 corridas con distintos seeds iniciales, tanto el *accuracy* como el *auroc* máximo promedio alcanzados por el modelo que utiliza ARL superan al modelo de control.

Dataset	mean	std	25 %	50 %	75 %	max
Baseline(SE)	0.852667	0.009516	0.848472	0.850972	0.861042	0.865278
No attention	0.788500	0.016842	0.780347	0.794306	0.797639	0.813056
ARL	0.786250	0.012818	0.769722	0.785555	0.796875	0.804445
SE y ARL	**0.853611**	0.009903	**0.848958**	**0.855694**	**0.861250**	**0.866667**

Tab. 4.6: La tabla se forma tomando el máximo *auroc* alcanzado por cada corrida y luego calculando la media, varianza y cuartiles de ellos.

A modo de constatar que el mecanismo de atención está efectivamente siendo utilizado por el modelo, analizamos cómo se modifica el valor de los alphas a lo largo de las épocas de una de las corridas.

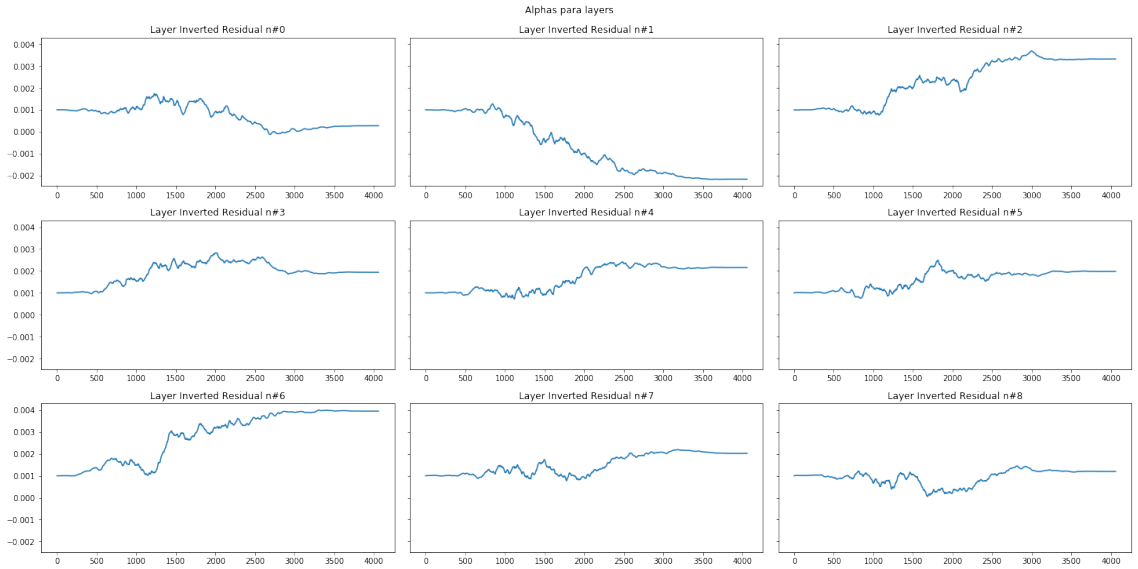

Fig. 4.3: Evolución de los parámetros alpha para cada capa bottleneck.

A diferencia de Resnet-50, en el modelo EfficientNet de los 15 inverted blocks solamente 9 utilizan efectivamente conexiones residuales. Es por ello, que solo contamos con 9 parámetros α en este caso —a diferencia del experimento anterior donde contábamos con 16—.

A primera vista se puede notar una gran diferencia con el experimento anterior: los niveles que alcanzan los parámetros α son considerablemente menores. Mientras que en este experimento existen parámetros α que por momentos llegan a picos de 0.004, en el experimento anterior existían parámetros α con picos de 0.075. Es decir, hay una diferencia de al menos 1 orden de magnitud en los valores que se alcanzan.

A la vez, como una segunda diferencia, en este experimento se visualizan parámetros α con valores negativos cuando en el experimento anterior no habían existido. El motivo de esto es incierto. Por un lado siempre que el parámetro α aumente en valor absoluto se entiende que se está agregando expresividad a la red. Sin embargo, si ese valor es negativo, se entiende que esa expresividad está siendo encauzada en sentido de negar el efecto que produce dicho valor. En otras palabras, en la segunda capa el modelo se beneficia al negar

el efecto que produce aplicar softmax en sentido espacial.

Más allá de esta sorprendente observación, la tabla de resultados da como claro ganador al modelo que contiene tanto ARL como Squeeze & Excitation: su máximo *accuracy* promedio es al menos un punto porcentual mayor al del modelo base —modelo con solo Squeeze & Excitation—.

Otra interesante observación surge de analizar el comportamiento de los modelos restantes: el que no contiene ningún mecanismo de attention y el que contiene solamente al nuevo mecanismo agregado ARL. Al interpretar la tabla de resultados, se ve que el modelo sin attention supera al modelo con ARL. Esto es extraño pues introducir ARL al comparar modelos con Squeeze & Excitation, parecía mejorar la performance. Sin embargo en este caso, parece empeorar. Esto nos lleva a pensar que en el caso de *Efficient-Net*, ARL produce un efecto compuesto al estar acompañado de Squeeze & Excitation; y es este mismo efecto compuesto el que parece mejorar los resultados.

4.4.4. Análisis cualitativo con GradCAM

Adicionalmente a las métricas que permiten un análisis cuantitativo en la sección 4.3.3, es posible realizar un análisis cualitativo de los modelos mediante observar el resultado de *GradCAM* [31]. La técnica *GradCAM* permite visualizar mapas de calor sobre las imágenes para comprender qué secciones de las mismas influyen mayormente en la clasificación. En la figura 4.4 se expone una comparación entre los mapas de calor generados por el modelo original versus los generados por el modelo con ARL.

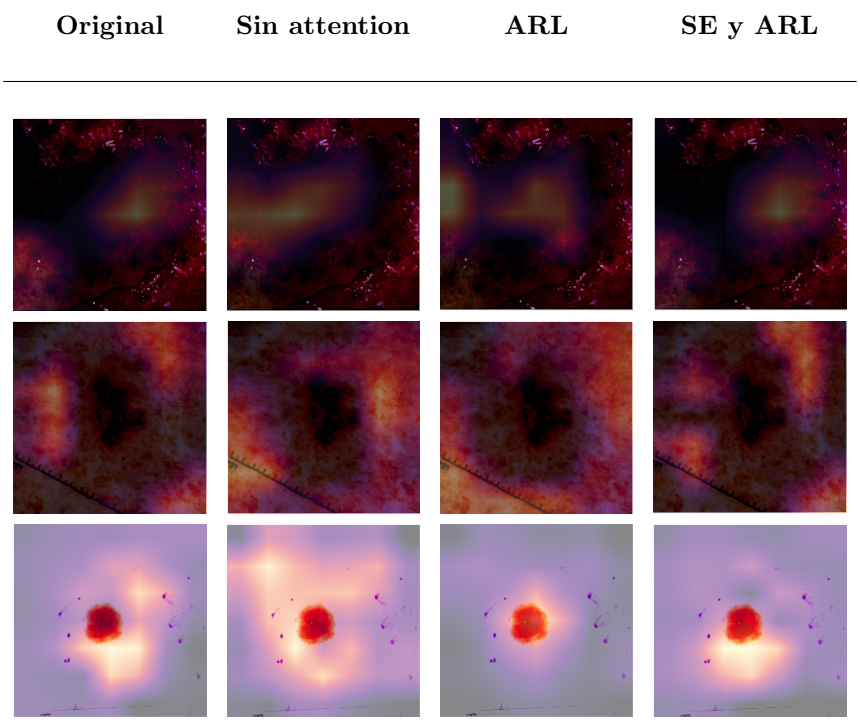

Fig. 4.4: Comparación de mapas de calor GradCAM para modelos con o sin ARL. En los ejemplos de la primer y segunda fila solo el modelo con SE y ARL predijo correctamente la presencia de melanoma. En el ejemplo de la tercer fila todos los modelos predijeron correctamente la ausencia de melanoma.

Al observar la figura 4.4 la tercer fila es la que parece exponer de manera más clara las diferencias. La introducción de ARL parece reducir notablemente el área de la imagen que la red considera relevante para la clasificación. Es decir, se puede ver visualmente que el mecanismo de atención funciona correctamente. Esto es especialmente notorio en el modelo que cuenta con ARL como único método de atención (modelo de EfficientNet al cual se le remueve el mecanismo de *Squeeze & excitation*), aunque también se observa de manera más sutil en el modelo que cuenta con ambos mecanismos ARL y SE.

Un segundo aspecto llamativo para notar aparece en el ejemplo de la segunda fila donde se puede observar como todos los modelos parecen centrarse en el área circundante a la lesión en lugar de la lesión en si misma. Esto sería un indicio que el tejido circundante a la lesión también aporta información valiosa. Particularmente en la imagen original correspondiente al ejemplo de la segunda fila el tejido circundante figura cubierto de marcas rojas, lo cual no pareciera ser información descartable. Esto puede verse como contraejemplo a la hipótesis del experimento de la sección 4.1 lo cual justificaría en parte sus resultados[1].

[1] Esto se debe a que dado que en el experimento de la sección 4.1 se intenta segmentar el área de la lesión mediante eliminar información de color del tejido circundante, se estaría precisamente eliminando la parte que la red considera relevante para obtener su predicción.

5. CONCLUSIÓN

En el presente trabajo se han estudiado diversos mecanismos para mejorar la performance de la clasificación de lesiones de piel utilizando redes neuronales convolucionales.

Respecto al experimento en la sección 4.1 que estudia el impacto de la segmentación del área de la lesión, se ha encontrado que el mecanismo utilizado para dicho fin no mejora los resultados. Las causas de este resultado pueden ser varias, como se expone en detalle en la sección 4.1.3. Sin embargo esto no concluye que segmentar el área de la lesión no pueda en el futuro mejorar los resultados utilizando un método distinto al expuesto en el presente trabajo. Queda entonces como trabajo futuro experimentar con nuevas maneras de transmitir a la red la información sobre qué región de la imagen constituye una lesión en sí.

Respecto a los experimentos sobre el preprocesamiento del dataset en la sección 4.2 se ha encontrado que el preprocesamiento de Ben Graham mejora notablemente el *accuracy* de la clasificación. No sucede lo mismo con los algoritmos de corrección de color de *Max RGB* y *Shades of Gray* los cuales arrojan resultados inferiores al dataset de control original.

Respecto a los experimentos de las secciones en la sección 4.3 y 4.4 los cuales estudian el impacto de introducir mecanismos de atención en las redes —particularmente el mecanismo ARL— se ha encontrado que en ambos casos la introducción de ARL mejora los resultados. El resultado del experimento de la sección 4.3 sirve como una verificación adicional a los resultados de [8], mientras que el resultado de 4.4 extiende el mecanismo y comprueba que también mejora la performance en los modelos de la familia *EfficientNet*.

Las redes neuronales han dominado el podio de algoritmos de clasificación de imágenes en los últimos años. Sin embargo estas no constituyen una *silver bullet* ya que dentro de sus características está la de ser modelos cuya interpretación no es sencilla de realizar. Esto hace que justificar los motivos de algún resultado que arrojó la red se torne una tarea sumamente difícil y que mejorar la performance de los modelos se vuelva un proceso de prueba y error basado en la intuición del investigador. Esto es lo contrario a lo que sucede en algoritmos cuyo funcionamiento es fácilmente interpretable, es claro el motivo de los resultados que arrojan y el proceso de mejora puede realizarse de manera más precisa y guiada.

Surge entonces el dilema de decidir si en cuanto a salud se trata se prefiere modelos más precisos pero de baja interpretabilidad a modelos menos precisos pero con alta interpretabilidad. ¿Con cuál de ellos se sentiría más cómodo un paciente de un hospital? La respuesta no es del todo clara. Sin embargo a pesar de ser un dilema, no es necesario conformarse con una sola de las dos opciones. Como investigadores de ciencias de la computación recae en nosotros la tarea de profundizar nuestro entendimiento de las redes neuronales —u otros nuevos futuros algoritmos— de modo que dicho dilema deje de existir y se cuente con modelos sumamente precisos e interpretables.

BIBLIOGRAFíA

[1] Dora Loria, Abel González y Clara Latorre. "Epidemiología del melanoma cutáneo en Argentina: análisis del Registro Argentino de Melanoma Cutáneo". En: 16 (mar. de 2010).

[2] Hugo Touvron, Andrea Vedaldi, Matthijs Douze y Hervé Jégou. *Fixing the train-test resolution discrepancy: FixEfficientNet*. 2020. eprint: arXiv:2003.08237.

[3] Noel C. F. Codella, M. Emre Celebi, Kristin Dana, David Gutman, Brian Helba, Harald Kittler, Philipp Tschandl, Allan Halpern, Veronica Rotemberg, Josep Malvehy y Marc Combalia. *International Skin Imaging Collaboration (ISIC) Challenge: using dermoscopic image context to diagnose melanoma*. Mar. de 2020. DOI: 10.5281/zenodo.3715750. URL: https://doi.org/10.5281/zenodo.3715750.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren y Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. eprint: arXiv:1512.03385.

[5] Mingxing Tan y Quoc V. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". En: (2019). eprint: arXiv:1905.11946.

[6] J. Yang, F. Xie, H. Fan, Z. Jiang y J. Liu. "Classification for Dermoscopy Images Using Convolutional Neural Networks Based on Region Average Pooling". En: *IEEE Access* 6 (2018), págs. 65130-65138.

[7] C. Barata, J. S. Marques y M. E. Celebi. "Improving dermoscopy image analysis using color constancy". En: *2014 IEEE International Conference on Image Processing (ICIP)*. 2014, págs. 3527-3531.

[8] J. Zhang, Y. Xie, Y. Xia y C. Shen. "Attention Residual Learning for Skin Lesion Classification". En: *IEEE Transactions on Medical Imaging* 38.9 (2019), págs. 2092-2103. DOI: 10.1109/TMI.2019.2893944.

[9] J. Hu, L. Shen y G. Sun. "Squeeze-and-Excitation Networks". En: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, págs. 7132-7141.

[10] Feng-ying Xie, Haidi Fan, Li Yang, Zhi-guo Jiang, Ru-song Meng y Alan Bovik. "Melanoma Classification on Dermoscopy Images Using a Neural Network Ensemble Model". En: *IEEE Transactions on Medical Imaging* PP (dic. de 2016), págs. 1-1. DOI: 10.1109/TMI.2016.2633551.

[11] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". En: *Biological Cybernetics* 36 (1980), págs. 193-202.

[12] Y. Lecun, L. Bottou, Y. Bengio y P. Haffner. "Gradient-based learning applied to document recognition". En: *Proceedings of the IEEE* 86.11 (1998), págs. 2278-2324.

[13] Alex Krizhevsky, Ilya Sutskever y Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". En: *Advances in Neural Information Processing Systems 25*. Ed. por F. Pereira, C. J. C. Burges, L. Bottou y K. Q. Weinberger. Curran Associates, Inc., 2012, págs. 1097-1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[14] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg y Li Fei-Fei. "ImageNet Large Scale Visual Recognition Challenge". En: *International Journal of Computer Vision (IJCV)* 115.3 (2015), págs. 211-252. DOI: 10.1007/s11263-015-0816-y.

[15] Andrew Tao, Karan Sapra y Bryan Catanzaro. *Hierarchical Multi-Scale Attention for Semantic Segmentation*. 2020. eprint: arXiv:2005.10821.

[16] Siyuan Qiao, Liang-Chieh Chen y Alan Yuille. *DetectoRS: Detecting Objects with Recursive Feature Pyramid and Switchable Atrous Convolution*. 2020. eprint: arXiv:2006.02334.

[17] Jonathan Ho, Ajay Jain y Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. eprint: arXiv:2006.11239.

[18] Adrian Bulat, Jean Kossaifi, Georgios Tzimiropoulos y Maja Pantic. *Toward fast and accurate human pose estimation via soft-gated skip connections*. 2020. eprint: arXiv:2002.11098.

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

[20] Jeremy P. Howard. *Data augmentation in computer vision*. URL: https://docs.fast.ai/vision.augment.

[21] Matthew D Zeiler y Rob Fergus. *Visualizing and Understanding Convolutional Networks*. 2013. arXiv: 1311.2901 [cs.CV].

[22] Kaiming He y Jian Sun. *Convolutional Neural Networks at Constrained Time Cost*. 2014. eprint: arXiv:1412.1710.

[23] Sihan Li, Jiantao Jiao, Yanjun Han y Tsachy Weissman. *Demystifying ResNet*. 2016. eprint: arXiv:1611.01186.

[24] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard y Quoc V. Le. "MnasNet: Platform-Aware Neural Architecture Search for Mobile". En: (2018). eprint: arXiv:1807.11626.

[25] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov y Liang-Chieh Chen. "MobileNetV2: Inverted Residuals and Linear Bottlenecks". En: (2018). eprint: arXiv:1801.04381.

[26] Jie Hu, Li Shen y Gang Sun. "Squeeze-and-Excitation Networks". En: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Jun. de 2018.

[27] O. Ronneberger, P.Fischer y T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". En: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. Vol. 9351. LNCS. (available on arXiv:1505.04597 [cs.CV]). Springer, 2015, págs. 234-241. URL: http://lmb.informatik.uni-freiburg.de/Publications/2015/RFB15a.

[28] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick y Piotr Dollár. *Microsoft COCO: Common Objects in Context*. 2014. eprint: arXiv:1405.0312.

[29] Leslie N. Smith. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay*. 2018. eprint: `arXiv:1803.09820`.

[30] Benjamin Graham. *Diabetic Retinopathy Detection Competition Entry*. `https://github.com/btgraham/SparseConvNet/blob/kaggle_Diabetic_Retinopathy_competition/competitionreport.pdf`. 2015.

[31] Ramprasaath R. Selvaraju, Abhishek Das, Ramakrishna Vedantam, Michael Cogswell, Devi Parikh y Dhruv Batra. "Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization". En: *CoRR* abs/1610.02391 (2016). arXiv: `1610.02391`. URL: `http://arxiv.org/abs/1610.02391`.