UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Servicios y Sistemas de Telecomunicación

TRABAJO FIN DE GRADO

ANÁLISIS AUTOMÁTICO DE IMÁGENES PARA DETECCIÓN DE MELANOMA

Francisco Javier Martín Ameneiro
Tutor: Juan Carlos San Miguel Avedillo
Ponente: Jesús Bescós Cano

Junio 2020

ANÁLISIS AUTOMÁTICO DE IMÁGENES PARA DETECCIÓN DE MELANOMA

Francisco Javier Martín Ameneiro
Tutor: Juan Carlos San Miguel Avedillo
Ponente: Jesús Bescós Cano

Video Processing and Understanding Lab
Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2020

Trabajo parcialmente financiado por la convocatoria 2019 de ayudas a Proyectos de I+D para jóvenes de la Universidad Autónoma de Madrid el mediante el proyecto "Aiding dignosis by self-supervised deep learning from unlabeled medical imaging", con referencia SI 1/PJI/2019-00414.

Universidad Autónoma
de Madrid

Resumen

El principal objetivo de este trabajo es resolver el problema propuesto por *ISIC 2019:Skin Lesion Analysis Towards Melanoma Detection*, que es una competición de carácter internacional de detección y clasificación de cáncer de piel.

Para empezar a abordar este trabajo, se realizará, en primer lugar, un estudio del arte, dónde definiremos los conceptos básicos tanto del deep learning, como de las redes neuronales convolucionales. Después estudiaremos las redes neuronales convolucionales clásicas y por último nos centraremos en la competición ISIC 2019; analizaremos el dataset que se nos proporciona, las tareas propuestas y los mejores participantes de los tres últimos años.

En el siguiente capítulo, se propondrá la técnica de *Transfer Learning* que se utilizará. A continuación, se presentará Pytorch, la librería de Python sobre la que realizaremos nuestro estudio y se describirá el diseño que tendrá nuestro algoritmo. Además, se escogerán las redes a estudiar (ResNet, DenseNet y SqueezeNet).

En el cuarto capítulo, antes de empezar con las pruebas se explicará como se ha obtenido el dataset que se va a utilizar y las diferentes métricas que se van analizar posteriormente. También se fijarán los hiperparámetros necesarios para proporcionar el mejor rendimiento posible, en primer lugar nos centraremos en el número de épocas y a continuación en la tasa de aprendizaje y el tamaño del batch. Una vez realizado esto, se ejecutarán una serie de experimentos donde veremos como afecta el uso de el data augmentation a las diferentes redes seleccionadas y se realizará un análisis de estos resultados.

Por último, en el quinto capítulo, se realizaran una serie de conclusiones sobre el trabajo realizado en las cuales repasaremos los puntos claves del trabajo y se definirá la red que mejor resultados nos ha proporcionado, además se propondrá una serie de posibles factores para mejorar los resultados de este trabajo en un futuro.

Palabras clave

Aprendizaje profundo, redes neuronales convolucionales, cáncer de piel, Python, *machine learning*.

Abstract

The main objective of this work is to solve the problem proposed by *ISIC 2019:Skin Lesion Analysis Towards Melanoma Detection*, which is an international competition of detection and classification of skin cancer.

To begin this work, we will first study the state of art, where we will define the basic concepts of both deep learning and convolutional neural networks. Then we will study the classical convolutional neural networks and finally we will focus on the ISIC 2019 competition; we will analyze the dataset provided to us, the proposed tasks and the best participants of the last three years.

In the next chapter, the *Transfer Learning* technique to be used will be proposed. Next, we will present Pytorch, the Python library on which we will carry out our study and we will describe the design that our algorithm will have. In addition, the networks to be studied will be chosen (ResNet, DenseNet and SqueezeNet).

In the fourth chapter, before starting with the tests, it will be explained how the dataset to be used has been obtained and the different metrics that will be analyzed later. We will also set the necessary hyperparameters to provide the best possible performance, firstly we will focus on the number of times and then on the learning rate and the size of the batch. Once this has been done, a series of experiments will be carried out where we will see how the use of data augmentation affects the different networks selected and an analysis of these results will be made.

Finally, in the fifth chapter, a series of conclusions will be drawn from the work carried out in which we will review the key points of the work and define the network that has provided us with the best results, in addition to proposing a series of possible factors to improve the results of this work in the future.

Keywords

Deep Leraning, convolutional neural network, skin cancer, Python, machine learning.

Agradecimientos

En primer lugar quiero agradecer a mi tutor Juan Carlos San Miguel, por ayudarme durante todo el año para el desarrollo de este trabajo y por su tiempo dedicado. También agradecer a Jesús Bescós que en este último tramo del trabajo ha invertido tiempo y dedicación para ultimar los últimos detalles del trabajo.
A continuación quiero agradecer a mis padres, por apoyarme en todo momento, ayudarme a tomar decisiones difíciles y animarme a seguir trabajando en los momentos más difíciles de la carrera. Sin ellos no hubiese llegado a este punto de mi vida.
No pueden faltar tampoco todos mis compañeros que he conocido en estos cuatro años, porque sin ellos nada hubiese sido igual, por todas las risas y los buenos momentos en la EPS y también fuera de ella.
Por último agradecer a mi novia Alicia, que junto a mis padres es uno de los pilares de mi vida. Siempre está para animarme cuando más lo necesito y es capaz de hacerme sonreir con tan solo una mirada.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Deep Learning (aprendizaje profundo), a pesar de ser un concepto relativamente novedoso, se está convirtiendo en un ámbito con una gran demanda en el mundo laboral debido a sus múltiples aplicaciones tanto en el rama de imagen como en la de sonido. En este trabajo nos vamos a centrar en la rama de imagen, en concreto en el ámbito médico, mediante el desarrollo de un clasificador de cáncer de piel, el cual será capaz de detectar y clasificar diferentes lesiones cutáneas de manera automática, pudiendo ser de gran utilidad a los expertos en este ámbito.
Desde hace cuatro años, existe una competición llamada *ISIC: Skin Lesion Analysis Towards Melanoma Detection* en la que se compite por conseguir el mejor detector cada año. Para nuestro trabajo hemos utilizado el dataset que proporciona esta competición ya que consta de un amplio número de imágenes y un abanico de hasta 9 tipos de clasificaciones distintas.
A lo largo de este ensayo veremos como mediante el uso de diferentes hiperparámetros (número de épocas, tasa de aprendizaje y tamaño del mini-batch), diferentes tipos de data augmentation y usando diferentes redes neuronales convolucionales, obtendremos mejores o peores resultados. No obstante, todos los conceptos mencionados anteriormente los analizaremos más adelante.

1.2. Objetivos

El objetivo principal de este trabajo es comprender el funcionamiento de las redes neuronales convolucionales y como la variación de diferentes hiperparámetros afectan a su rendimiento siendo capaces, de esta manera, de resolver el reto que nos proponen desde *ISIC 2019:Skin Lesion Analysis Towards Melanoma Detection*. Para poder llegar a este punto, en primer lugar será necesario realizar un estudio previo de los elementos básicos de *deep learning*. A continuación, deberemos asignar los valores adecuados para poder hacer una aproximación inicial a partir de la cual poder realizar las pruebas correspondientes. Por último, realizaremos un análisis de los resultados obtenidos.
Todo este trabajo lo podemos dividir en varias secciones:

1. **Estudio del estado del arte.** Definiremos tanto *deep learning* como las redes neuronales convolucionales, además explicaremos los conceptos básicos que la constituyen. A continuación haremos una pequeña introducción de las arqui-

tecturas clásicas del deep learning y por último se realizará un estudio de la competición ISIC 2019.

2. **Diseño de una propuesta.** Se escogerán las redes en las cuales basaremos nuestro estudio, se presentarán los datos que se van a utilizar y se harán una serie de pruebas para fijar los hiperparámetros.

3. **Realización de experimentos**. Siguiendo la estructura del TFG del año pasado [1] se harán diferentes pruebas en cada una de las redes escogidas variando el data augmentation y usando un dataset que habremos creado previamente. Una vez terminado esto, analizaremos los resultados.

4. **Conclusiones y trabajo futuro.** Una vez realizados nuestros experimentos, deberemos realizar una serie de conclusiones y se propondrá diferentes tareas para poder mejorar los resultados obtenidos.

1.3. Organización de la memoria

Esta memoria consta de los siguientes capítulos:

- Capítulo 1 Introducción.
- Capítulo 2 Estado del arte.
- Capítulo 3 Diseño y Desarrollo.
- Capítulo 4 Evaluación.
- Capítulo 5 Conclusiones y Trabajo Futuro.

Capítulo 2

Estado del arte

2.1. Deep Learning: Definición

El aprendizaje profundo, comúnmente conocido como *Deep Learning*, se puede definir como una inteligencia artificial (IA) que, mediante redes capaces de aprender de manera autónoma a partir de datos desestructurados o sin etiquetar, trata de imitar al cerebro humano en lo que a la toma de decisiones y procesamiento de datos se refiere.

Deep Learning ha ido evolucionando a medida que lo ha ido haciendo la era digital, la cual traía, cada vez más, una gran cantidad de datos. Estos datos, conocidos como "Big Data", tienen su origen en una gran variedad de lugares como las redes sociales, plataformas comerciales, etc. Normalmente estos datos están desestructurados, por lo que a un humano tardaría décadas en poder extraer información de ello, sin embargo, el Deep Learning puede realizar esta tarea de una manera mucho más eficiente.

2.2. Redes neuronales convolucionales

Una red neural convolucional (CNN) es un algoritmo basado en Deep Learning que es muy usado en el campo del reconocimiento y procesado de imágenes. Las redes neuronales están formadas por una serie de capas, de las cuales, las más destacables son: capa convolucional, capa pooling y capa fully-conected.

2.2.1. Capa convolucional

La capa convolucional es el bloque que mayor carga computacional tiene. Como su nombre indica, esta capa se encarga de realizar una convolución al aplicar un filtro, también conocido como kernel, sobre la imagen de entrada. El tamaño de este kernel es variable y tendrá el siguiente aspecto W x H x C (*Anchura x Altura x Número de canales*). Por cada filtro, al realizar la convolucional se creara un mapa de características (feature map).

La operación de la convolución consiste en ir aplicando el kernel de izquierda a derecha y de arriba debajo de la imagen de entrada según el valor de la zancada (*stride*), de manera que si el valor del *stride* es 1, el kernel se moverá de píxel en píxel, sin embargo si el stride es 2, cada vez que se aplique el kernel, este se desplazara dos píxeles. Se puede dar el caso en el que el kernel no encaje perfectamente con la imagen de entrada,

para solucionar este problema se utiliza el padding. Existen dos opciones de padding, la primera consiste en rodear la imagen original de ceros para que el kernel encaje, la segunda directamente descarta las partes donde el kernel no encaja. La primera opción es conocida como *Same Padding* y la segunda como *Valid Padding*.

El objetivo de la capa convolucional es el de extraer características de alto nivel como por ejemplo los bordes, los colores, etc.

2.2.2. Capa pooling

La capa pooling, al igual que la capa convolucional, hace uso de un kernel sobre toda la imagen de entrada, pero en este caso el objetivo de esta capa es el de reducir el número de parámetros cuando las imágenes son muy grandes pero siempre manteniendo la información más relevante. En esta capa también son importantes los valores del *padding* y el *stride* mencionados anteriormente.

Existen dos tipos de pooling: *Max Pooling* y *Average Pooling*. El primero devuelve el valor máximo de la región cubierta por el kernel y el segundo, el Average Pooling, devuelve la media de los valores cubiertos por el kernel.

Esta capa, junto a la capa convolucional, se encargan de extraer las características de las imágenes en una red neuronal convolucional, cuanto más complejas sean estas imágenes, más capas se necesitaran. A continuación vamos a ver la última capa de una red neuronal convolucional que se encarga de la clasificación.

2.2.3. Capa fully-conected

La capa fully-conected compone la última parte de una red neuronal convolucional y su principal objetivo es tomar las imágenes de las capas anteriores, procesarlas y asociar cada imagen con su etiqueta corresponde.
La salida de las capas pooling y las capas convolucionales son aplanadas en un solo vector de valores. Cada una de los valores del vector se corresponde con la probabilidad de la imagen de pertenecer a cada una de las posibles etiquetas. Para realizar esta clasificación se pueden usar diferentes clasificadores como el binario o el softmax, el cual vamos a explicar brevemente a continuación.

2.2.4. Clasificación Softmax

La Clasificación Softmax (*Softmax Classification*) se encarga de realizar la tarea asignar a cada valor del vector aplanado que realiza la capa fully-conected, la probabilidad de que la imagen pertenezca a dicha clase. La suma de estos valores debe ser 1.0 como podemos ver en la Tabla 2.1.
La Clasificación Softmax, se puede ejecutar de dos formas:
La primera de ellas, llamada softmax completo, consiste en realizar la clasificación que hemos visto anteriormente, donde se calculan las probabilidades de todas las clases.
La segunda, conocida como muestreo de candidatos, se encarga de, como su nombre indica, realizar un muestreo entre las clases, de manera que se descartan aquellas que no nos interesen. Usando de ejemplo las etiquetas de la Tabla 2.1, si quisiéramos diferenciar entre dos razas de perro distintas, simplemente necesitaríamos el valor de la

Clases	Probabilidad
manzana	0.001
oso	0.04
golosina	0.008
perro	0.95
huevo	0.001

Tabla 2.1: Ejemplo de Clasificación Softmax

probabilidad de la etiqueta de perro, por lo que no sería necesario calcular las demás probabilidades. Esta técnica reduce significativamente el coste computacional respecto a la técnica softmax completo

2.3. Conceptos básicos

2.3.1. Tipos de datos

Para poder entrenar una red neuronal convolucional de una manera óptima debemos dividir los datos en tres conjuntos: los datos de entrenamiento, los datos de validación y los datos de prueba.

- **Datos de entrenamiento**: Normalmente es el conjunto de datos más grande, su principal función es que la red neuronal los use para aprender y descubrir una serie de conocimientos que, al aplicarlos sobre diferentes parámetros, consigan obtener una tasa de error bajo.

- **Datos de validación**: Este set de datos se encarga de prevenir el overfitting y de elegir los hiperparámetros. El overfitting o sobre-ajuste (Figura 2.1), se puede definir como el efecto producido debido a un entrenamiento excesivo de la red sobre la que estamos trabajando, que provoca que nuestra red solo sea capaz de reconocer los datos que se usan como entrenamiento, y al introducir datos nuevos (datos de prueba), no será capaz de identificarlos. Una manera de prevenir esto, es mediante la adición de ciertas modificaciones a los datos de entrenamiento como por ejemplo ruido, giros, zoom, etc. Estas modificaciones se conocen como data augmentation y ampliaremos nuestros conocimientos sobre ello más adelante. La selección de hiperparámetros sirve para ayudar a nuestra red neuronal a elegir el mejor algoritmo.

- **Datos de prueba**: Este último conjunto de datos es el encargado de evaluar nuestra red neuronal y proporcionar la precisión real que esta tiene. Este set de datos suele ser del mismo tamaño que el set de datos de validación.

En cuanto a la proporción de datos que de dividen en cada uno de estos tres grupos, no existe una norma general, ya que puede variar según el número de datos totales que se tenga, las intenciones del usuario con la red neuronal, etc. No obstante, la mayor parte de las veces el número de datos de entrenamiento es mucho mayor que los datos de validación y prueba que, como hemos dicho anteriormente, suelen ser iguales. Un ejemplo de la división de estos datos podría ser el explicado en la Figura 2.2.

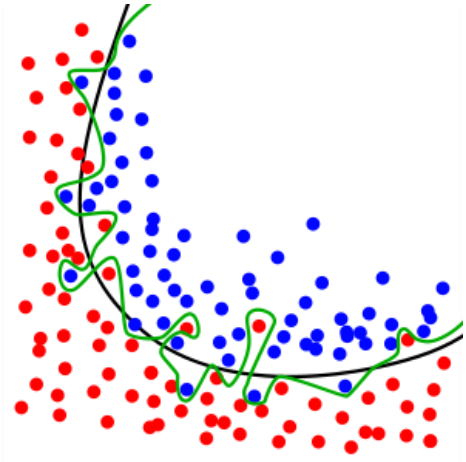

Figura 2.1: Ejemplo de overfitting en el cual podemos ver como ciertos valores son erróneos a pesar del entrenamiento. (Fuente: [2])

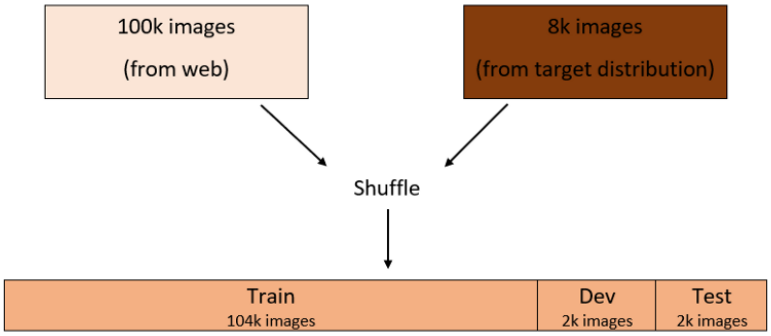

Figura 2.2: En este ejemplo de distribución de datos, partimos de un total de 108k datos. El 96 % de los datos son usados en el entrenamiento, el 2 % se usan como datos de validación y el 2 % restante para datos de prueba. Cuanto menor sea el conjunto de datos totales, menor será la diferencia entre los datos de entrenamiento y los datos de validación y prueba. (Figura 2.2) (Fuente: [3])

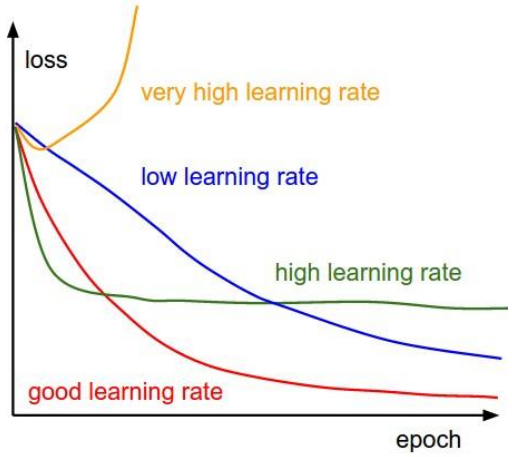

Figura 2.3: En esta gráfica podemos ver diferentes tasas de aprendizajes superpuestas unas a otras. La curva amarilla no convergerá nunca ya que el valor es demasiado alto, mientras que en el caso de la curva azul, al haber tomado un valor demasiado bajo, convergerá muy lento. La curva óptima es la de color rojo. (Fuente: [4])

2.3.2. Tasa de aprendizaje

La tasa de aprendizaje es un hiperparámetro que controla como estamos ajustando los pesos de nuestra red con respecto el descenso de gradiente. Cuanto menor sea el valor de esta tasa, más despacio viaja por la pendiente de descenso, lo cual puede ser una buena forma de no saltarnos ningún mínimo local y de esta manera detectar fácilmente cuando la curva converge , sin embargo, si el valor es demasiado bajo, puede ocurrir que converja muy despacio y eso tampoco nos interesa. Por otra parte, si escogemos un valor alto para la tasa de aprendizaje puede ocurrir que la curva no converja bien o, en el peor de los casos, que diverja. Podemos ver un ejemplo gráfico de este hiperparámetro en la Figura 2.3.

2.3.3. Función de pérdidas

La principal finalidad de la función de pérdidas es medir el error de nuestro modelo al realizar las predicciones. Existen distintas formas de calcular esta función, dos de las más destacadas son:

- **Error absoluto medio (MAE)**: Mide la media del valor abosulto de los errores de predicción de nuestro conjunto de datos. Se caracteriza por ser fácil de interpretar, sin embargo, se ve penalizada cuando aparecen errores grandes. Podemos calcularlo de la siguiente manera (2.1):

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j| \tag{2.1}$$

- **Error cuadrático medio (RMSE)**: Cálcula la raíz cuadrada del error cuadrático medio (MAE). Soluciona el problema que surgía cuando aparecen errores grandes, pero su interpretación es más difícil. Podemos utilizar la siguiente ecuación

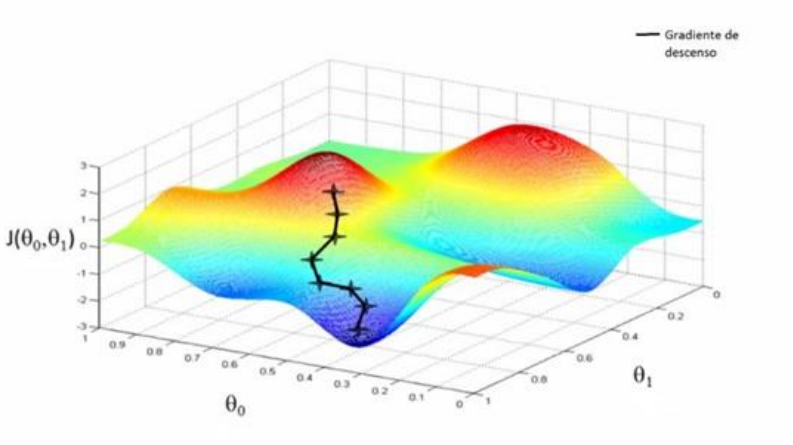

Figura 2.4: Representación gráfica del descenso por gradiente hasta hallar el mínimo de la función. (Fuente: [5])

para calcularlo (2.2):

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|} \tag{2.2}$$

2.3.4. Optimizador: Descenso por gradiente

Uno de los algoritmos de optimización más conocidos en el ámbito del Machine Learning es el descenso por gradiente.
El algoritmo de descenso por gradiente tiene como objetivo final encontrar el mínimo global de una determinada función, para ello hace uso del parámetro de tasa de aprendizaje, explicado anteriormente, cuyo valor se deberá ajustar correctamente para que el algoritmo trabaje de una manera eficiente. Figura 2.4.

Según los datos utilizados en cada una de las iteraciones, el algoritmo de descenso por gradiente funcionará de manera diferente:

- **Descenso por gradiente en lotes**: Por cada iteración se introducirán todos los datos disponibles, lo que provocará problemas de estancamiento ya que el algoritmo estará calculado el descenso por gradiente con los mismos datos de manera repetitiva y las variaciones acabarán siendo mínimas.

- **Descenso por gradiente estocástico**: Se introducirán datos de manera aleatoria evitando así los problemas de estancamiento que se producían en el caso anterior. La principal limitación de este modelo es su lentitud.

- **Descenso por gradiente estocástico en mini-lotes**: En este caso por cada iteración se introducen varias muestras de datos que mantienen la características de aleatoriedad que tenia el modelo anterior. De esta manera resuelve el problema de lentitud ya que es capaz de realizar varias operaciones en paralelo.

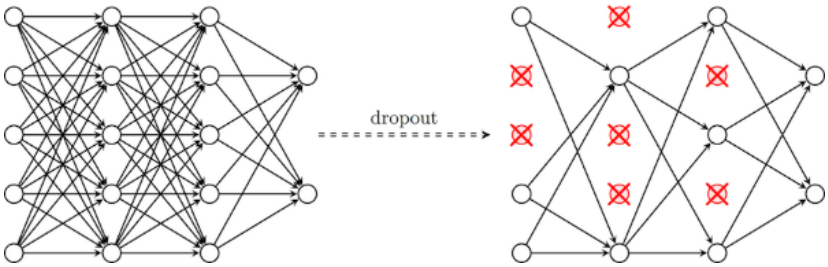

Figura 2.5: Ejemplo gráfico de la aplicación del dropout. En la figura de la izquierda vemos una red neuronal estándar. La figura de la derecha es esa misma red tras haberle aplicado dropout, las casillas que están con una cruz rojas son las que han sido ignoradas. (Fuente: [6])

2.3.5. Dropout

El término dropout se refiere cuando en una red neuronal ignoramos ciertas unidades (neuronas) de manera aleatoria. Es decir para cada escenario de entrenamiento, ciertas neuronas son seleccionadas, con una probabilidad de 1-p, para ser ignoradas, de esta manera la red neuronal se reduce.
La principal función del dropout es prevenir el overfitting. Una red neuronal que ha sido entrenada con una gran cantidad de neuronas, hace que estas creen, durante el entrenamiento, una codependencia entre ellas. Esto hace que el poder que tiene cada neurona de manera individual se reduzca lo que concluye provocando overfitting en los datos de entrenamiento. Podemos visualizarlo mejor mediante el ejemplo de la Figura 2.5

2.3.6. Epoch, Batch e Iteraciones

Una época o *epoch* es cuando un dataset al completo pasa a través de una red neuronal hacia delante y hacia atrás. De esta manera todos los datos son procesados y se podrán actualizar sus pesos.

En las redes neuronales convolucionales no es suficiente con pasar una vez el dataset a través de la red, es decir, no es suficiente con usar una época ya que eso provoca lo que conocemos como *underfitting*. La solución a esto tampoco es usar un número muy elevado de épocas ya que esto puede provocar overfitting. En realidad no hay un número concreto de épocas para que la red neuronal trabaje de manera óptima, para cada dataset este valor irá variando.

Se puede dar el caso que el dataset sea muy grande para introducirlo al completo en el ordenador, en este caso podemos dividir los datos en varios lotes o batches, que tendrán el tamaño correspondiente a los datos de entrenamiento. De la misma manera, cada batch se puede dividir en varios mini-lotes o mini-batches, por lo que el tamaño de estos será menor que el de los batches.

Entrenar las redes con mini-batches hace que el coste computacional disminuya y ocupará menos memoria, pero como inconveniente tiene que la convergencia será más lenta, por lo tanto deberíamos variar los tamaños de los mini-batches hasta encontrar el que trabaja de manera más óptima.

Por último, las iteraciones son el número de batches necesarios para completar una época. Por ejemplo si tenemos un dataset con 20.000 ejemplos y lo dividimos en batches de 500 ejemplos, necesitaremos 40 iteraciones para completar una época.

2.3.7. Precisión, Recall y Accuracy

En las redes neuronales convolucionales utilizamos ciertos parámetros para comprobar si nuestro algoritmo trabaja de una manera correcta. Tres de estos parámetros son la precisión, recall y exactitud.

La precisión se corresponde al número verdaderos positivos que nuestro algoritmo ha detectado entre el número de positivos totales. (2.3):

$$Precisión = \frac{TP}{(TP + FP)} \tag{2.3}$$

El recall se refiere al número de positivos que fueron identificados de manera correcta y se calcula de la siguiente manera (2.4):

$$Recall = \frac{TP}{(TP + FN)} \tag{2.4}$$

Por último la exactitud se puede definir como el número de predicciones correctas que ha predicho nuestro algoritmo respecto al número total de predicciones que este ha calculado, es decir (2.5):

$$Accuracy = \frac{(TP + TN)}{(TP + FP + TN + FN)} \tag{2.5}$$

Donde TP significa positivos verdaderos (*True Positives*), FP es falsos positivos (*False Positives*), TN es negativos verdaderos (*True Negative*) y FN falsos negativos (*False Negative*).

2.4. Arquitecturas clásicas

En esta sección vamos a ver las arquitecturas clásicas que pueden formar una red neuronal convolucional.

2.4.1. LeNet-5

Es una de las arquitecturas más simple, su origen data del año 1998 y era utilizado en muchos bancos para reconocer los números escritos a mano en los cheques.
La red LeNet-5 está formada por dos de capas convolucionales y tres son capas fully-conected, a demás por cada capa convolucional hay una capa pooling media (*average*

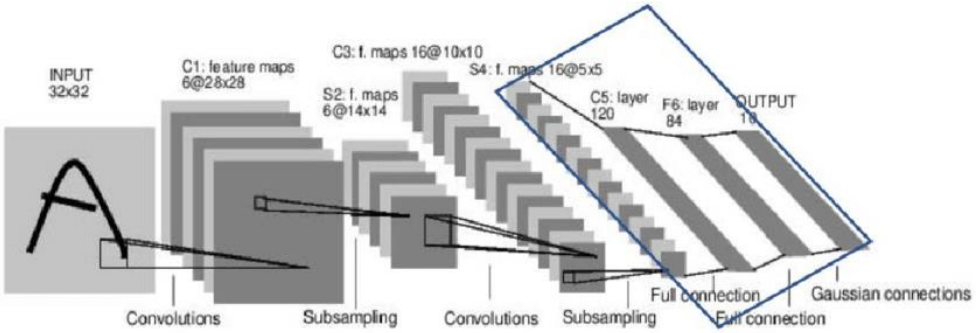

Figura 2.6: Arquitectura de una red LeNet-5 (Fuente: [7])

pooling). Esta arquitectura tiene alrededor de 60.000 parámetros. Figura 2.6.

La imagen de entrada debe ser de 32x32 y debe estar en escala de grises. El kernel de la primera capa es de 5x5x6 y el stride tendrá un valor de uno, por lo que la dimensión de la imagen al final de la primera capa será de 28x28x6.

Tras la primera capa convolucional, la imagen resultante pasa a través de una capa pooling media cuyo kernel es de 2x2x6 y esta vez el stride es dos. Como resultado el tamaño de la imagen se reducirá a 14x14x6.

A continuación aparece la segunda capa convolucional, esta vez el kernel será de 5x5x16 y el stride uno.

La cuarta capa corresponde a la segunda capa pooling media, que al igual que en la segunda capa pero esta vez el número de filtros aumenta a 16, por lo tanto la imagen tras pasar por esta capa reducirá su tamaño de 10x10x16 a 5x5x16.

Finalmente, en la quinta capa, llegamos a las capas fully-conected, esta primera tendrá 120 filtros de tamaño 1x1. La siguiente capa fully-conected está compuesta de 84 unidades. Por último la salida de esta red es la tercera capa fully-conected, compuesta por 10 unidades correspondientes a los dígitos de 0 a 9. (Tabla 2.2).

2.4.2. AlexNet

En el año 2012 el científico ucraniano Alex Krizhevsky, desarrollo una red neuronal convolucional compuesta sesenta millones de parámetros conocida como AlexNet (Figura 2.7). Está formada por cinco capas convolucionales, de las cuales tres de ellas están seguidas por otra capa pooling máximo (*Max Pooling*), y por 3 capas fully-conected

La imagen de entrada debe tener un tamaño de 227x227 y debe ser una imagen RGB, por lo que el número de canales es 3. En la primera capa convolucional el kernel

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Tabla 2.2: Arquitectura LeNet por capas

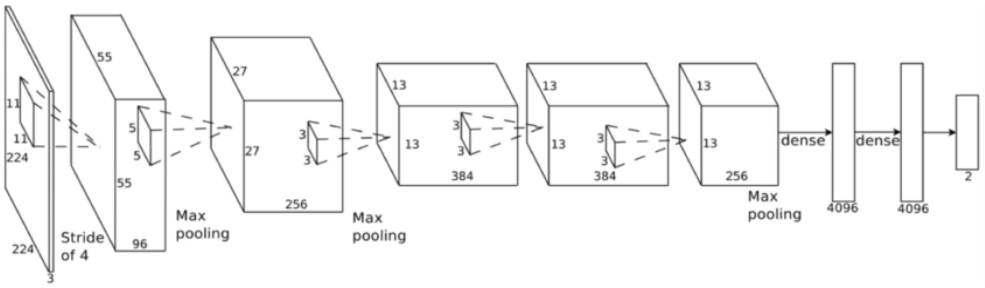

Figura 2.7: Arquitectura de una red AlexNet (Fuente: [8])

Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	227x227x3	-	-	-
1	Convolution	96	55x55x96	11x11	4	relu
	Max Pooling	96	27x27x96	3x3	2	relu
2	Convolution	256	27x27x256	5x5	1	relu
	Max Pooling	256	13x13x256	3x3	2	relu
3	Convolution	384	13x13x384	3x3	1	relu
4	Convolution	384	13x13x384	3x3	1	relu
5	Convolution	256	13x13x256	3x3	1	relu
	Max Pooling	256	6x6x256	3x3	2	relu
6	FC	-	9216	-	-	relu
7	FC	-	4096	-	-	relu
8	FC	-	4096	-	-	relu
Output	FC	-	1000	-	-	softmax

Tabla 2.3: Arquitectura AlexNet por capas

es de 11x11x96 y el stride tiene un valor de cuatro, por lo que la imagen pasa a tener un tamaño de 55x55x96. Tras esta primera capa, aparece la primara capa pooling máximo, cuyo kernel es de tamaño 3x3x96 y el valor del stride es de 2. Una vez que la imagen haya pasado por esta capa, su tamaño se reduce a 27x27x96.

A continuación, está la segunda capa convolucional con un kernel de 5x5x256 y un stride. Seguidamente, como en la capa anterior, viene la capa pooling máximo, cuyo kernel es igual pero esta vez el número de filtros es 256. Tras estas dos capas, la imagen tendrá un tamaño de 13x13x256.

Las tres siguientes capas son capas convolucionales, las cuales tienen un kernel de tamaño 3x3 y el número de filtros será 384 en las dos primeras y 256 en la tercera, el stride siempre será uno. Tras estas tres capas aparece la tercera capa pooling máximo, con un kernel de 3x3x256 y un stride de 2. Una vez superado este bloque el tamaño de la imagen será 6x6x256.

Una vez pasado este bloque aparecen las capas tres capas fully-conected. La primera está compuesta por 9216 unidades y las dos siguientes por 4096 unidades. Finalmente está la salida softmax con 1000 posibles valores. (Tabla 2.3).

2.4.3. VGG-16

Esta tercera arquitectura fue desarrollada en 2014 por los científicos Simonyan y Zisserman. La estructura es similar a la de AlexNet pero esta red está compuesta por 13 capas convolucionales y 3 capas fully conected y por un total de 138 millones de parámetros. También existe una variante aun mas profunda llamada VGG-19 pero no nos centraremos en ella.

La imagen de entrada debe ser RGB y con un tamaño de 224x224, tras pasar las dos primeras capas convolucionales el tamaño cambiara a 224x224x64 ya que ambas

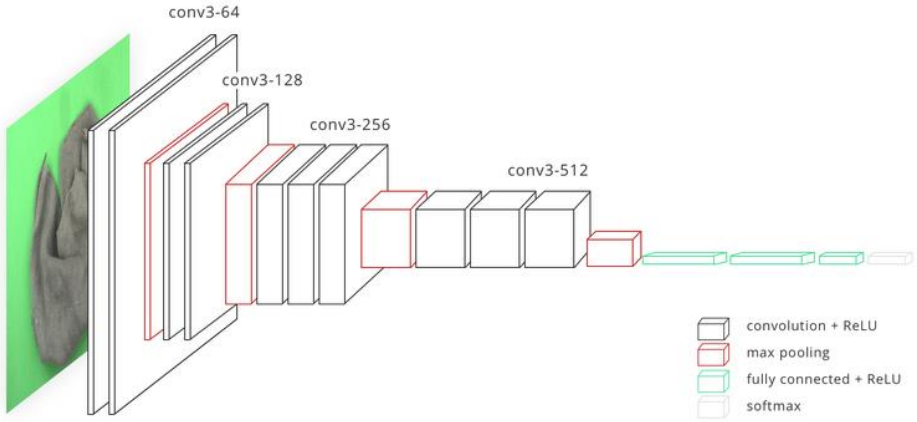

Figura 2.8: Arquitectura de una red VGG-16 (Fuente: [9])

tienen un kernel de 3x3x64. Después aparece una capa pooling máximo con un kernel de 3x3x64 y un stride de 2. La imagen se reducirá a las dimensiones de 112x112x64.

A continuación viene otro bloque de dos capas convolucionales seguida por una capa de pooling máximo. El kernel de las capas convolucionales es de 3x3x128 y el stride es uno. En cuanto al kernel de la capa pooling máximo será igual que en el bloque anterior. Una vez superado esta capa la imagen tendrá un tamaño de 56x56x128.

Después vienen tres bloques formados por tres capas convolucionales seguidas por su correspondiente capa pooling máximo. Al pasar estos tres bloques la imagen será de 7x7x512.

Finalmente están las tres capas fully-conected, las dos primeras formadas por 4096 unidades y la última, que corresponde a la salida, está compuesta por 1000 unidades, que corresponden a las posibles salidas. (Figura 2.8).

2.5. ISIC 2019

Skin Lesion Analysis Toward Malanoma Detection es una competición creada por ISIC (*International Skin Imaging Collaboration*) en la que el principal objetivo es, a partir de una red neuronal convolucional, reconocer distintos tipos de cáncer de piel a partir de imágenes. Las imágenes son proporcionadas por el propio ISIC y conforman un total de 25.331 unidades.
Los diferentes diagnósticos (Figura 2.9) que se deben identificar son:

1. Melanoma (MEL)
2. Nevo melanocítico (NV)
3. Carcinoma basocelular (BCC)

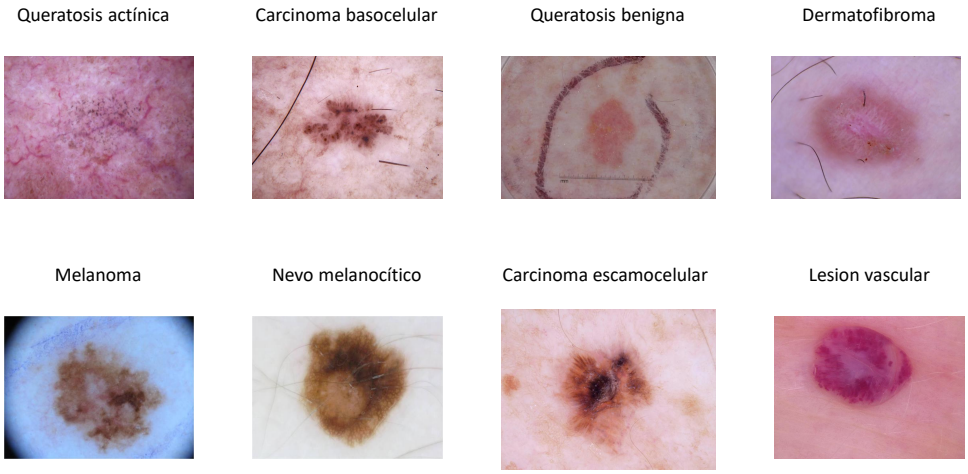

Figura 2.9: Diferentes diagnósticos posibles

4. Queratosis actínica (AK)
5. Queratosis benigna (BKL)
6. Dermatofibroma (DF)
7. Lesión vascular (VASC)
8. Carcinoma escamocelular (SCC)
9. Ninguna de las anteriores (UNK)

Como novedad, este año, se ha añadido una nueva formada de participar que es haciendo uso de metadatos, sin embargo nosotros nos focalizaremos en clasificar las imágenes sin hacer uso de metadatos.
La evaluación de los proyectos está completamente automatizada se van publicando en una lista la cual incluye detalladas estadísticas. El premio es de 4000 dólares, 2000 dólares y 1000 dólares para el primer, segundo y tercer clasificado respectivamente.

2.5.1. Tareas

ISIC 2019 propone tres tareas:

- **Tarea 1. Segmentación de lesiones:** El principal objetico es conseguir los contornos de las diferentes lesiones a partir de las imágenes demoscópicas proporcionadas.
- **Tarea 2. Detección de atributos de la lesión:** En esta tarea nos debemos centrar en la extracción de características de las imágenes proporcionadas, en concreto los atributos que se deben identificar son las redes de pigmentación, redes negativas, estrías, quistes y glóbulos.

AÑO	POSICIÓN	EQUIPO	ARQ. BASE	DATASET EXTERNO	DATA AUGMENT.	ACCURACY
2017	1º	Casio & Shishu	ResNet-50	Sí	Rotation, translation, scaling, flipping.	0.911
2017	2º	Multemedia Procesing Group-UC3M	ResNet-50	Sí	Rotation, re-size.	0.910
2017	3º	RECORD Titans	Inception-v4 y ResNet-101	Sí	Re-size, zoom.	0.908
2018	1º	MetaOptima Technology Inc.	ResNet, DPN, Densenet, Inception	Sí	Horizontal flips, rotations, jitter.	0.885
2018	2º	MetaOptima Technology Inc.	DPN, ResNet, Inception, SeResNext	Sí	Re-size, flips, rotations, brightness.	0.882
2018	3º	MetaOptima Technology Inc.	DPN, ResNet, Densenet, Inception, SeResNext	Sí	Flips, rotations, brightness, saturation.	0.871
2019	1º	DAISYLab	EfficientsNets, SENet154	Sí	Brightness, contrast, flips, rotation.	0.636
2019	2º	DysionAI	ResNet, SeResNext, DeneseNet	No	Scale, rotation,flips, color transformation.	0.607
2019	3º	Almage Lab & PRHLT	DeneseNet, ResNet, SeResNext	No	Poisson noise, gamma contrast.	0.593

Tabla 2.4: Top-3 ISIC entre los años 2017 y 2019.

- **Tarea 3. Diagnóstico de lesiones:** El objetivo de esta tercera tarea es el de identificar el tipo de lesión a partir de las imágenes suministradas. Los tipos de lesión son los que he mencionado en el apartado anterior.

De las tres tareas en este proyecto nos centraremos en la tercera tarea, en la cual las imágenes que nos proporciona ISIC son de formato JPG.
Todas las imágenes siguen la siguiente estructura en su nombre ISICx.jpg donde x corresponde con un numero de 7 dígitos exclusivo para cada imagen.
Las imágenes provienen de HAM10000 Dataset, que es un dataset con imágenes de diferentes pacientes que han sufrido distintas lesiones cutáneas.

2.5.2. Participantes

En la Tabla 2.4 podemos ver los mejores participantes en los últimos tres años de la competición ISIC, en ella se incluyen las principales características de los proyectos más destacados en cada año.
De la clasificación de 2019 cabe destacar que los tres primeros clasificados usan un modelo ensemble, al igual que la mayoría de los clasificados, sin embargo en el puesto 6º podemos ver que el grupo Torus Actions usa un modelo diferente.

- **DAISYLab**: Este equipo ocupa la primera posición del ranking, en su proyecto podemos observar que han añadido otras lesiones que no están presentes en el dataset de entrenamiento. Durante el desarrollo les surgió el problema de desequilibrio de clases, el cual lo resolvieron mediante la pérdida de equilibrio (loss balancing). Como las imágenes del dataset tienen diferentes resoluciones, han decidido cambiarlas mediante diferentes estrategias. Su objetivo final es obtener una eficacia de 70 % aproximadamente.

- **DysionAI:** Este grupo es el segundo clasificado, a diferencia del primero, ellos no usan un dataset externo pero si realizan modificaciones en las imágenes debido a que, tanto la escala como la resolución varían de manera considerable. Afirman que pueden alcanzar el 75.3 % de efectividad.

AÑO	DATASET	WEB	#DATOS TRAIN	#DATOS VAL	#DATOS TEST	#CLASES	BEST ACCURACY
2017	-	https://challenge.kitware.com	2.000	150	600	3	0.911
2018	HAM10000	https://challenge2018.isic-archive.com/	10.015	193	1.512	7	0.885
2019	BCN_20000, HAM10000, MSK	https://challenge2019.isic-archive.com/	25.331	-	8.238	7	0.636

Tabla 2.5: Diferentes datasets usados en la competición ISIC entre 2017 y 2019.

En la Tabla 2.5 podemos observar una comparativa entre los años 2017, 2018 y 2019 en la cual observamos que a medida que pasan los años aumentan el número de datos y con ello el número de clases a identificar. Este aumento en el volumen del dataset implica un aumento de la dificultad y con ello una disminución de la eficacia (accuracy).

Capítulo 3

Diseño y desarrollo

3.1. Introducción

Una vez se ha analizado el estado del arte y con ello hemos comprendido los conceptos básicos de las redes neuronales convolucionales, nos vamos a centrar, en este tercer capítulo, en qué es y como funciona Pytorch y a continuación, estudiaremos de manera simplificada cómo funcionan las arquitecturas que vamos a usar a lo largo de este trabajo.

3.2. Transfer Learning

Uno de los grandes problemas que surge al utilizar *Deep Learning* se produce cuando queremos entrenar nuestras redes, ya que se trata de un proceso costoso y excesivamente largo que puede a llegar a durar varios días o semanas.
Transfer Learning proporciona una solución a este problema ya que evita tener que entrenar una red desde cero, permitiendo el uso de redes neuronales convolucionales que ya han sido previamente entrenadas. Basandonos en la información que nos proporciona [10] podemos ver que dentro de *Transfer Learning* existen dos escenarios distintos:

- **Ajustar la CNN**: Se inicializará la red con una red pre-entrenada en lugar de hacerlo de manera aleatoria.
- **CNN como estractor de características fijas**: En este caso se congelaran todos los pesos de nuestra red excepto la capa fully-conected. A esta capa se le asignaran nuevos pesos de manera aleatoria y será entrenada.

En nuestro caso, continuaremos el estudio del TFG del año pasado [1] el cual, se demuestra que el segundo escenario devuelve, como norma general, unos mejores resultados.

3.3. Pytorch

Pytorch [11] es una librería de Python que sirve de gran utilidad para trabajar en deep learning. Hoy en día existen otras alternativas para trabajar sobre este ámbito, como por ejemplo, Tensorflow [12], Caffe [13] o Keras [14]. Sin embargo, hemos decidido usar Pytorch para el desarrollo de este trabajo ya que ,a pesar de ser relativamente

nueva, consta de un gran número de tutoriales que nos pueden servir de apoyo..
Pytorch consta con dos cualidades que resaltan sobre el resto. La primera es que realiza cálculos con tensores con aceleración de una GPU. Esto nos permite acelerar los tiempos de ejecución de manera significativa.
La segunda cualidad de Pytorch es que, a diferencia de otras librarías como Tensorflow, es capaz de trabajar con grafos dinámicos en vez de estáticos, lo que tiene como resultado que mientras se ejecuta, se pueden modificar las funciones y por lo tanto el cálculo de nuestro gradiente variará.

3.4. Diseño

En nuestro programa, en primer lugar, importamos todas las librerías necesarias, después definimos nuestras variables, en las que se especifican las rutas donde se encuentran los datasets que vamos a usar, el lugar donde queremos guardar los resultados, etc. Por último, en este primer bloque de definición, damos valor a nuestros parámetros (tasa de aprendizaje, numero de épocas y tamaño de mini-batch), los cuales han sido explicados previamente en la sección 2.3, y elegimos el data augmentation que queramos utilizar.
A continuación, tanto sobre el dataset de train como sobre el de test, tras haber sido cargados, se aplicarán una serie de transformaciones para adoptar el tamaño necesario para que nuestra red pueda trabajar. En este momento, será también cuanto se aplique el data augmentation que hallamos elegido. Data augmentation consiste en la generación artificial de datos, normalmente imágenes, a partir de diferentes técnicas como giros, escalados, recortes etc. Su principal finalidad es hacer que la red que estamos entrenando sea más hábil y como hemos dicho en la sección 2.3.1 evitar que la red memorice los datos de entrenamiento y prevenir, de esta manera, el overfitting.
Una vez tenemos nuestros datos listos, se elige la red preentrenada que queramos utilizar y entramos en un bucle en el cual se calcula, por cada época, el valor de precisión (accuracy) y las pérdidas. Estos valores son guardados en la ruta que hemos definido previamente. Por ultimo, podemos cargar los resultados y representarlos en una gráfica para poder analizarlos de una manera visual.

3.5. Arquitecturas utilizadas

3.5.1. ResNet

Es una arquitectura de redes neuronales convolucionales creada en 2015, su nombre completo es Residual Neural Network[15]. Su creador es Kaiming He y está compuesta por 26 millones de parametros. La arquitectura ResNet se diferencia de las arquitecturas clásicas en que introduce conexiones salteadas (*skip connections*), como resultado de esta innovación se podía entrenar una red neuronal con 152 capas con una complejidad menor que la arquitectura VGG-16. Podemos visualizar su estructura básica mediante la Figura 3.1.
En las redes ResNet predominan dos tipos de bloques, el bloque identidad y el bloque convolucional. El bloque identidad es el bloque estándar usado en las ResNet y se usa cuando se da el caso en el que las dimensiones de entrada y salida del salto son iguales, cuando las dimensiones don diferentes se usa el bloque convolucional.

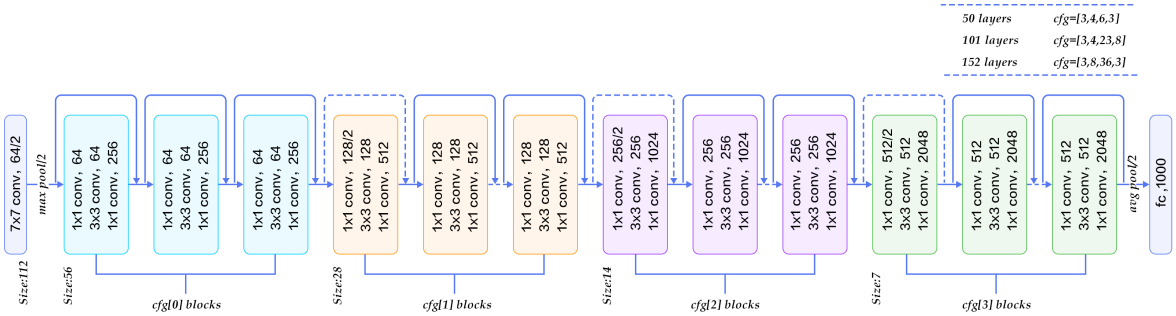

Figura 3.1: Arquitectura de una ResNet. (Fuente: [15])

3.5.2. DenseNet

DenseNet [16] se diferencia de otras arquitecturas en que tiene una profundidad mayor que otras arquitecturas, pero aun así es fácil de optimizar ya que usa menos parámetros.

Una DenseNet está formado por lo que se conoce como *"Dense blocks"* (Bloques densos). Cada uno de estos bloques están formados por una serie de capas conectadas entre sí, de manera que cada capa recibe como input el mapa característico de las capas anteriores.
Cada una de las capas que constituyen un Dense block, por lo general, tienen el siguiente aspecto: Una subcapa de normalización del batch, una subcapa de activación de ReLU y una subcapa convolucional 3x3. Figura 3.2

Para conectar los diferentes Dense blocks que forman la DenseNet y sus correspondientes mapas característicos, se utilizan las capas de transición.
Cada capa de transición está compuesta por una subcapa de normalización de batch, una capa convolucional 1x1 y una capa pooling media.

Anteriormente hemos dicho que las DenseNets tienen menos parámetros de lo normal que otras arquitecturas, esto se debe a dos motivos.
El primero de ellos es la tasa de crecimiento, la cual define el número de mapas característicos va a transmitir cada capa de un Dense block. Debido a que las capas de un Dense block están conectadas entre sí, estas comparten mucha información que puede llegar a ser redundante, por lo tanto se puede decir que la tasa de crecimiento regula cuanta información nueva se transmite a la siguiente capa.
El otro motivo es el cuello de botella, el cual se utiliza para limitar el número de mapas característicos que se transmiten.

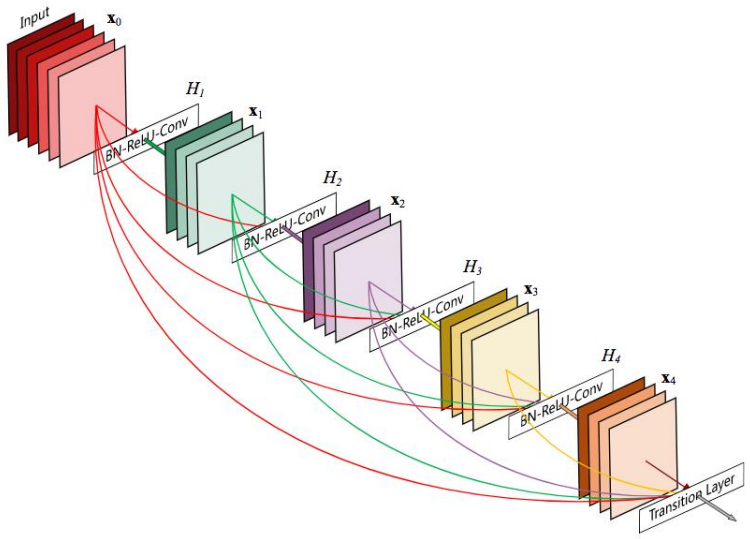

Figura 3.2: Arquitectura de un DenseNet de cinco capas. (Fuente: [16])

Arquitectura	**Compresión**	**Data type**	**Tamaño original /comprimido**	**Reducción respecto AlexNet**	**Top-1 ImageNet Accuracy**	**Top-5 ImageNet Accuracy**
AlexNet	-	32 bit	240 MB	1x	57.2 %	80.3 %
AlexNet	SVD	32 bit	240 MB / 48 MB	5x	56 %	79.4 %
AlexNet	Network Pruning	32 bit	240 MB / 27 MB	9x	57.2 %	80.3 %
AlexNet	Deep Compresion	5-8 bit	240 MB / 6.9 MB	35x	57.2 %	80.3 %
SqueezeNet	-	32 bit	4.8 MB	50x	57.5 %	80.3 %
SqueezeNet	Deep Compresion	8 bit	4.8 MB / 0.66 MB	363x	57.5 %	80.3 %
SqueezeNet	Deep Compresion	6 bit	4.8 MB / 0.47 MB	510x	57.5 %	80.3 %

Tabla 3.1: En esta tabla podemos ver la gran diferencia que existe entre el uso de una SqueezeNet y una AlexNet. La diferencia es tal que una SqueezeNet puede llegara a ser 510 veces menor que una AlexNet mientras mantiene un nivel de eficacia prácticamente idéntico. (Fuente:[17])

3.5.3. SqueezeNet

Estas redes [17] se caracterizan principalmente por su capacidad de aumentar enormemente la capacidad de compresión mientras se mantiene un nivel de eficacia óptimo. En la Tabla 3.1 podemos ver una comparativa entre AlexNet y SqueezeNet, en la cual queda en evidencia los ventajas que proporciona una SquezeNet.

Para conseguir esta gran reducción, SqueezeNet, hace uso de los llamados *fire modules* que se caracterizan por:

- **Reemplazar filtros 3x3 por filtros 1x1.** Esto hace que el número de parámetros necesarios pase a ser 9 veces menor, además no existen pérdidas de información.
- **Reducir las entradas de los filtros 3x3 restantes.** Al reducir el número de filtros se reduce el número de parámetros, esto se consigue gracias a las dos capas por las que están compuestas cada *fire module*: la capa squeeze y la capa expand.

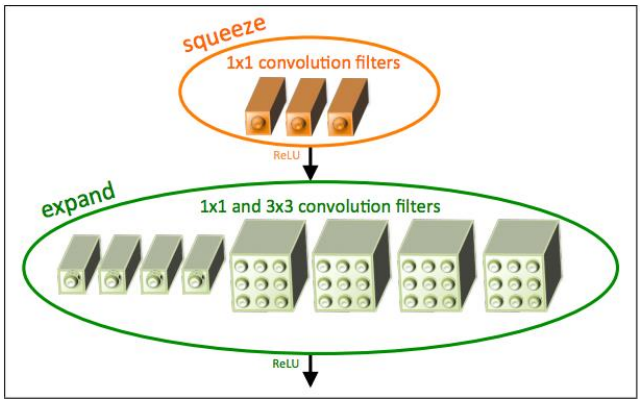

Figura 3.3: Podemos observar las dos capas que conforman un *fire module*: la capa squeeze, formada solamente de filtros 1x1, y la capa expand, compuesta tanto de filtros 1x1 como de filtros 3x3. (Fuente: [17])

Como podemos observar en la Figura 3.3 la capa squeeze esta formado por filtros 1x1 mientras que la capa expand consta tanto de filtros 1x1 como filtros 3x3. Reduciendo la cantidad de filtros de la capa squeeze que alimentan a la capa expand, se reducen las conexiones entre los filtros 3x3 lo que conlleva una reducción de los parámetros.

- **Reducir el muestreo para obtener mapas de activación grandes.** Esta ultima característica de los *fire modules*, a diferencia de las dos anteriores, tiene como objetivo aumentar el nivel de precisión. Principalmente se basa en que cuanto mayor sean los mapas de activación mayor sera la precisión.

Capítulo 4

Evaluación

4.1. Introducción

De manera análoga al Trabajo Fin de Grado del año pasado [1], hemos realizado una serie de experimentos usando nuestro propio datasets y las arquitecturas que hemos comentado en el sección 3.5. Para ello en primer lugar hemos tenido que realizar una serie de pruebas para fijar los hiperparámetros, tanto para los datasets basados en rand puro como para los basados en rand por clases. A continuación, para cada bloque de datasets hemos ido realizando distintas pruebas con diferentes data augmentations. Para finalizar, realizaremos una comparativa de los resultados obtenidos al final del capítulo.

4.2. Marco de evaluación

4.2.1. Dataset

El dataset utilizado para el desarrollo de este trabajo es el que la propia competición ISIC nos proporciona, como comentamos anteriormente en la sección 2.5, el dataset consta de un total de 25.331 imágenes que se corresponden a ocho distintos diagnósticos distintos de lesión cutánea y una novena clase que se usa si la imagen no se asocia a ninguna de las ocho anteriores. En la Figura 4.1 podemos observar varios ejemplos de cada clase. Además de las imágenes se nos proporciona un archivo *GroundTruth.csv* el cual relaciona cada imagen con la clase a la que se corresponde.
Todas las imágenes tienen un tamaño de 600x450 píxeles menos la clase de queratosis benigna (BKL), la clase de melanoma (MEL) y la clase nevo melanocítico (NV) cuyas imágenes tienen unas dimensiones de 1024x680, 1024x768 y 1022x767 respectivamente. No obstante, estas dimensiones serán modificadas al ser introducidas en nuestro algoritmo.
Con el objetivo de poder realizar más experimentos e intentar encontrar el conjunto de datos más óptimo para cada una de las distintas redes que vamos a utilizar, hemos creado, a partir del dataset original, hasta seis datasets distintos, cada uno con un conjunto de datos de entrenamiento y otro conjunto de datos de test, siguiendo dos técnicas:

- **Rand puro:** Consiste en, a partir de las 25.331 imágenes de las que partimos, elegir una de manera completamente aleatoria y a partir de un porcentaje previamente definido enviarla al conjunto de train o al conjunto de test. En nuestro

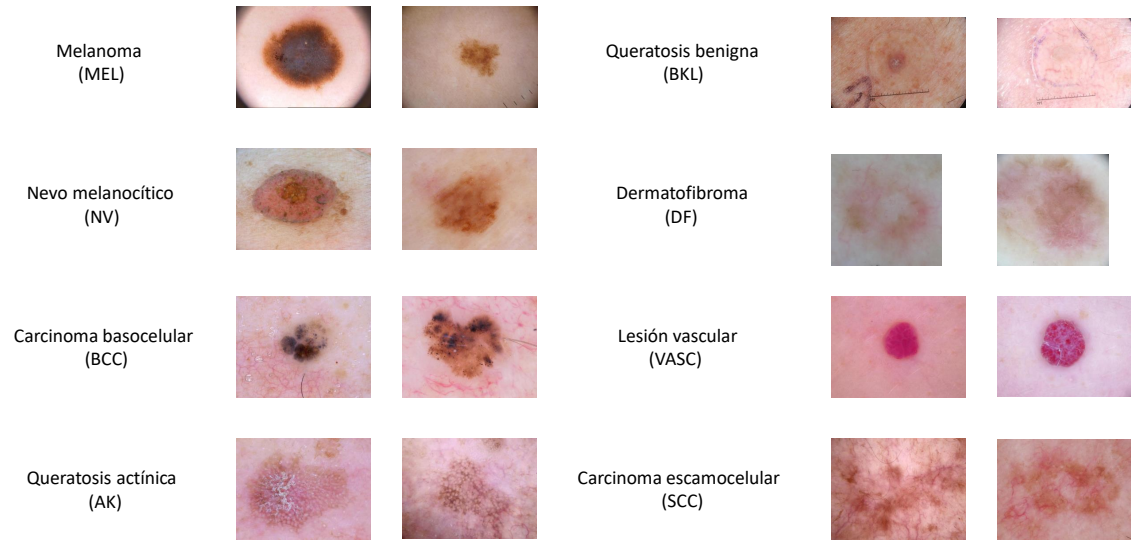

Figura 4.1: Diferentes diagnósticos según el dataset que nos proporciona ISIC 2019, proveniente de HAM10000 Dataset.

		MEL		NV		BCC		AK		BKL		DF		VASC		SCC		UNK	
		Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
Rand puro	set_80_20_1	3603	919	10288	2587	2650	673	678	189	2104	520	186	53	195	58	481	147	0	0
	set_80_20_2	3656	866	10307	2568	2666	657	676	191	2111	513	194	45	196	57	492	136	0	0
	set_80_20_3	3614	908	10297	2578	2674	649	693	174	2079	545	184	55	214	39	498	130	0	0
Rand por clases	set_80_20_1	1578	423	1606	395	1586	415	723	144	1596	405	196	43	196	57	494	134	0	0
	set_80_20_2	1609	392	1596	405	1614	387	682	185	1617	384	205	34	206	47	494	134	0	0
	set_80_20_3	1606	395	1580	421	1586	415	679	188	1635	366	193	46	210	43	497	131	0	0

Tabla 4.1: Podemos ver las diferentes distribuciones según el tipo de de técnica que hallamos elegido para crearlo. Como era de esperar, los datasets de rand por clases al tener un límite fijado en 2000, tienen un menor número de imágenes totales.

casos hemos creado tres datasets a partir de esta técnica siguiendo una proporción de 80 % train y 20 % test.

- **Rand por clases:** Esta segunda técnica consiste en fijar un valor límite que indica la cantidad máxima de imágenes que puede haber por clase, en nuestro caso el valor que adopta esta variable límite es de 2000, por lo que como máximo habrá 2000 imágenes por clase. Una vez tenemos las imágenes dentro nuestro limite, las enviamos al conjunto de entrenamiento o al conjunto de test. De manera análoga a la técnica de rand puro, hemos creado tres datasets siguiendo la misma proporción 80/20.

En la Tabla 4.1 podemos ver como han quedado distribuidas las imágenes en los seis datasets diferentes creados.

```
Epoch 11/24
----------
train Loss: 0.2748 Acc: 0.8852
val Loss: 0.1868 Acc: 0.9346

Epoch 12/24
----------
train Loss: 0.3648 Acc: 0.8361
val Loss: 0.1962 Acc: 0.9281

Epoch 13/24
----------
train Loss: 0.2771 Acc: 0.8852
val Loss: 0.2869 Acc: 0.8889

Epoch 14/24
----------
train Loss: 0.2786 Acc: 0.8893
val Loss: 0.1935 Acc: 0.9346
```

Figura 4.2: Es una captura de los resultados que nos devuelve el tutorial de Pytorch en el cual el conjunto de datos de validación, que es nuestro equivalente al conjunto de test, obtiene mejores resultados que el conjunto de datos de entrenamiento. (Fuente: [10])

4.2.2. Métricas

Las dos variables que medimos en este trabajo son *accuracy*, explicada previamente en la sección 2.3.7, y las pérdidas 2.3.3.
Para calcular el valor de la *accuracy* podemos usar la fórmula comentada anteriormente (Ecuación 2.5) en la cual se dividen el numero de predicciones correctas entre el número total de predicciones. El valor resultante estará comprendido entre el rango [0 - 1], cuanto más cerca se encuentre del 1 significa que nuestra predicción será más precisa.
En cuanto a las pérdidas, va a tomar un valor, que cuanto mayor sea significará que mas pérdidas tiene nuestro modelo y que por lo tanto peor son nuestras predicciones.
Teóricamente, el conjunto de datos de train debería obtener unos resultados mejores que el conjunto de datos de test, ya que el modelo que utilizamos es entrenado con los datos del conjunto train, los cuales son diferentes a los que se encuentran en el conjunto test. Sin embargo, vamos a ver como durante todas las pruebas que vamos a ir realizando ocurre justo lo contrario, es decir, nuestro conjunto de datos test va a obtener unos resultados mejores que el conjunto train. Esto puede ser en que para desarrollar nuestro código nos hemos basado en [**?**], el cual es un tutorial de Pytorch dónde ocurre lo mismo. En la Figura 4.2 podemos visualizar esto.

4.2.3. Ajuste de hiperparámetros

Los hiperparámetros que debemos ajustar para nuestro modelo son :

- Número de épocas
- *Learning rate* (Tasa de aprendizaje)

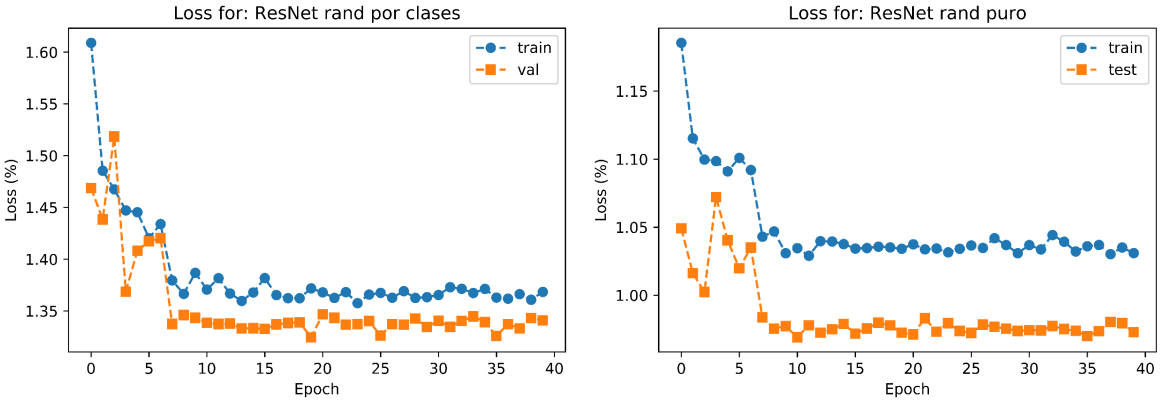

Figura 4.3: En esta figura podemos ver la convergencia de las redes ResNet para 40 épocas usando los dos tipos de datasets. La figura de la izquierda representa el dataset creado a partir de rand por clases y el de la derecha representa el dataset creado a partir de rand puro.

- Tamaño del mini-batch

Para calcular estos tres parámetros usaremos la red ResNet. En primer lugar nos centraremos en encontrar el número de épocas a partir del cual las gráficas convergen y adoptan un valor mas o menos constante. Como vamos a tratar los datasets de rand puro y los de rand por clases de forma independiente deberemos realizar esta prueba por cada tipo.
Para ello ejecutaremos utilizaremos una red ResNet, que será la que utilizaremos de base para ajustar los parámetros, con un número de épocas suficientemente grande como para que de tiempo a converger, en nuestro caso hemos utilizado 40 épocas. Como podemos observar en la Figura 4.3 para los datasets rand puro podemos afirmar que aproximadamente a partir de 17 épocas converge, en cuanto a los datasets rand por clase, estos convergen aproximadamente en las 15 épocas.

Una vez hemos fijado el valor del numero de épocas para ambos tipos de datasets, podemos focalizarnos en encontrar los valores de la tasa de aprendizaje y del tamaño del mini-batch para los cuales se obtienen unos mejores resultados. Nuevamente analizaremos los dos tipos de datasets de manera separada y usaremos, una vez más, la red ResNet como aquella sobre la que realizar nuestras pruebas. A pesar de haber obtenido el número de épocas a partir del cual ambas datasets convergen, vamos usar un valor fijo para ambos de 25 épocas para asegurarnos de que en cada experimento sea capaz de converger adecuadamente.
Los valores que puede adoptar la tasa de aprendizaje son 0.01, 0,001 ó 0.0001. Como vimos en la sección 2.3.2 cuanto menor sea este valor, mas lento convergerá.
En cuanto a los posibles valores que adoptará el tamaño del mini-batch, estos variarán entre 16, 32, 64 y 128.

Rand puro			
Tamaño del mini-batch	**Tasa de aprendizaje**	**Accuracy Train**	**Accuracy Test**
16	0.01	0.6065	0.6321
16	0.001	0.6270	0.6564
16	0.0001	0.6173	0.6467
32	0.01	0.6265	0.6506
32	0.001	0.6360	0.6576
32	0.0001	0.6163	0.6327
64	0.01	0.6359	0.6520
64	0.001	0.6344	0.6510
64	0.0001	0.6086	0.6253
128	0.01	0.6402	0.6590
128	0.001	0.6324	0.6477
128	0.0001	0.5951	0.6059

Tabla 4.2: En esta tabla se representan todos los valores obtenidos en las pruebas realizadas sobre nuestra red ResNet, que hacen referencia a los datasets obtenidos mediante la técnica de rand puro. Se muestra en color verde el mejor valor entre todos los obtenidos.

Los resultados obtenidos tras haber realizado todas las pruebas los podemos visualizar en las Tablas 4.2 y 4.3, que representan los valores de los datasets rand puro y rand por clases respectivamente.
Una vez hemos analizado ambas gráficas podemos concluir que ambas técnicas, tanto la de rand puro como la de rand por clases, adoptan un los mismos valores tanto para la tasa de aprendizaje como para el tamaño del mini-batch. Por lo tanto, a partir de ahora para las pruebas restantes usaremos, para los dataset de rand puro, un número de épocas igual a 17, una tasa de aprendizaje de 0.01 y un tamaño del mini-batch de 128; para los datasets de rand por clases, el número de épocas sera 15 y el los valores de los otros dos hiperparámetros será el mismo. (Tabla 4.4).

4.3. Pruebas y resultados

A continuación, una vez hemos fijado todos los hiperparámetros (número de épocas, tasa de aprendizaje y tamaño del mini-batch) para los dos tipos de datasets, procederemos a realizar un estudio sobre cada una de las tres redes elegidas, Resnet, DenseNet y SqueezeNet, las cuales han sido explicadas en la sección 3.5.
En este estudio, haremos diferentes pruebas sobre cada una de estas redes en las cuales el papel del data augmentation será clave, ya que lo iremos variando para analizar los efectos que tiene sobre la efectividad (*accuracy*) y sobre las pérdidas. Usaremos tres tipos de data augmentation:

- **Giro horizontal (*horizontal flip*)**: Consiste en girar la imagen de entrada en sentido horizontal.
- **Giro vertical (*vertical flip*)**: Consiste en girar la imagen de entrada en sentido

Rand por clases			
Tamaño del mini-batch	**Tasa de aprendizaje**	**Accuracy Train**	**Accuracy Test**
16	0.01	0.4944	0.5020
16	0.001	0.4928	0.4970
16	0.0001	0.4643	0.4697
32	0.01	0.4967	0.4980
32	0.001	0.4988	0.5025
32	0.0001	0.4493	0.4539
64	0.01	0.5100	0.5144
64	0.001	0.4997	0.5050
64	0.0001	0.4214	0.4405
128	0.01	0.5149	0.5149
128	0.001	0.4818	0.4821
128	0.0001	0.3901	0.4092

Tabla 4.3: En esta tabla se representan todos los valores obtenidos en las pruebas sobre nuestra red ResNet, que hacen referencia a los datasets obtenidos mediante la técnica de rand por clases. Se muestra en color verde el mejor valor entre todos los obtenidos.

	Número de epocas	**Tasa de aprendizaje**	**Tamaño del mini-batch**
Rand puro	17	0.01	128
Rand por clases	15	0.01	128

Tabla 4.4: En esta tabla podemos ver los resultados que hemos obtenido tras haber realizado todas las pruebas para el ajuste de hiperparámetros. A partir de ahora se utilizaran estos valores para realizar las siguientes pruebas sobre las diferentes redes.

	Data augmentation			Rand puro acc.		Rand por clases acc.	
Set 80/20	**Flip Hor.**	**Flip Vert.**	**Jitter**	**Train**	**Test**	**Train**	**Test**
1	-	-	-	0.6412	0.6625	0.5136	0.5084
2	-	-	-	0.6396	0.6600	0.5210	0.5335
3	-	-	-	0.6426	0.6575	0.5114	0.5197
1	x	-	-	0.6387	0.6584	0.5171	0.4940
2	x	-	-	0.6436	0.6503	0.5185	0.5351
3	x	-	-	0.6441	0.6589	0.5094	0.5342
1	x	x	-	0.6440	0.6613	0.5144	0.5129
2	x	x	-	0.6420	0.6507	0.5176	0.5371
3	x	x	-	0.6419	0.6591	0.5113	0.5302
1	x	x	x	0.6448	0.6547	0.5143	0.5109
2	x	x	x	0.6421	0.6499	0.5166	0.5300
3	x	x	x	0.6444	0.6613	0.5235	0.5317

Tabla 4.5: Resultados tras realizar las diferentes pruebas con data augmentation sobre una red ResNet. Se muestran en color verde, azul y naranja los mejores resultados obtenidos en el primer, segundo y tercer set de datos respectivamente de ambas clases.

vertical.

- **Variación del color (*jitter color*)**: Consiste en variar las crominancias y la luminancia de la imagen original.

Al igual que hemos estado haciendo a lo largo de el estudio, analizaremos los dos tipos de datasets individualmente.

4.3.1. Resultados ResNet

En primer lugar analizaremos la red ResNet ya que es la que hemos utilizado como apoyo para realizar todas las pruebas anteriores. Los resultados de esta prueba están reflejados en la Tabla 4.5. En ella podemos observar que el conjunto de datos generados a partir de rand puro obtiene mejores resultados que los generados a partir de rand por clases.
En los datos procedentes de rand puro, los mejores resultados tienden a encontrarse cuando no se le incluye ningún tipo de data augmentation, aunque existe la excepción del tercer dataset, el cual obtiene su mejor resultado al usar los tres tipos de data augmentation propuestos.
Sobre los datos de rand por clases, los tres mejoran al incorporar giros verticales y horizontales.
Podemos concluir diciendo que según el conjunto de datos que usemos aplicaremos de manera distinta el data augmentation. En los datasets procedentes de rand puro no añadiremos data augmentation y en rand por clases podremos añadir los dos tipos de giros.

4.3.2. Resultados DenseNet

Una vez analizada la red ResNet, pasaremos a estudiar la red DenseNet, cuyos resultados se encuentran en la Tabla 4.6. En la comparativa entre ambos tipos de

	Data augmentation			Rand puro acc.		Rand por clases acc.	
Set 80/20	**Flip Hor.**	**Flip Vert.**	**Jitter**	**Train**	**Test**	**Train**	**Test**
1	-	-	-	0.6559	0.6749	0.5335	0.5262
2	-	-	-	0.6572	0.6676	0.5456	0.5539
3	-	-	-	0.6585	0.6737	0.5328	0.5566
1	x	-	-	0.6537	0.6726	0.5340	0.5377
2	x	-	-	0.6581	0.6696	0.5448	0.5534
3	x	-	-	0.6590	0.6773	0.5366	0.5496
1	x	x	-	0.6539	0.6724	0.5353	0.5332
2	x	x	-	0.6543	0.6692	0.5393	0.5534
3	x	x	-	0.6580	0.6727	0.5314	0.5541
1	x	x	x	0.6549	0.6741	0.5350	0.5327
2	x	x	x	0.6571	0.6726	0.5357	0.5559
3	x	x	x	0.6554	0.6719	0.5362	0.5586

Tabla 4.6: Resultados tras realizar las diferentes pruebas con data augmentation sobre una red DenseNet. Se muestran en color verde, azul y naranja los mejores resultados obtenidos en el primer, segundo y tercer set de datos respectivamente de ambas clases.

datasets, volvemos a apreciar como la técnica rand puro devuelve mejores resultados que la técnica rand por clases.
Al analizar ambas técnicas de manera individual vemos que tienen comportamientos distintos frente al data augmentation. Los sets de rand puro tienden a tener mejores resultados cuando usamos los giros horizontales, y los sets de rand por clases devuelven mejores resultados cuando a los giros horizontales y verticales le añadimos el *jitter color*.

4.3.3. Resultados SqueezeNet

Por último, vamos a realizar las pruebas sobre una SqueezeNet, los resultados que nos devuelven los podemos encontrar en la Tabla 4.7.
Una vez más, al igual que en las dos redes anteriores, los datasets rand puro tienen un valor de *accuracy* mayor que los datasets rand por clases. En el análisis individual de ellos, vemos que los sets pertenecientes al tipo de rand por clases no siguen una tendencia clara ya que cada set obtiene un mejor resultado con distitas formas de uso del data augmentation. Sin embargo en el caso de los datasets de rand por clases si que podemos intuir una tendencia más clara ya que dos de los tres sets obtienen sus mejores resultados al incluir en data augmentation los dos tipos de giros y el *color jitter*.

4.3.4. Comparación de los resultados obtenidos

Una vez hemos analizado los resultados que obtenemos al realizar las pruebas sobre las tres redes distintas, vamos a compararlas entre sí.
Una característica común de todas las arquitecturas es que en todas ellas el conjunto de datos perteneciente a rand puro obtiene mejores resultados que los datos pertenecientes a rand por clases. Esto se puede deber a que los datasets de rand puro tienen mas imágenes y por lo tanto su entrenamiento es mejor.
La red que devuelve mejores resultados, como era de esperar, es DenseNet. Como hemos

	Data augmentation			Rand puro acc.		Rand por clases acc.	
Set 80/20	**Flip Hor.**	**Flip Vert.**	**Jitter**	**Train**	**Test**	**Train**	**Test**
1	-	-	-	0.6305	0.6362	0.4879	0.4936
2	-	-	-	0.6261	0.6374	0.4993	0.5102
3	-	-	-	0.6312	0.6325	0.4937	0.5042
1	x	-	-	0.6292	0.6197	0.4913	0.4926
2	x	-	-	0.6305	0.6322	0.4883	0.5127
3	x	-	-	0.6275	0.6149	0.4911	0.5147
1	x	x	-	0.6279	0.6257	0.4910	0.5010
2	x	x	-	0.6305	0.6402	0.4925	0.5173
3	x	x	-	0.6244	0.6167	0.4906	0.5217
1	x	x	x	0.6271	0.6403	0.4929	0.5045
2	x	x	x	0.6294	0.6382	0.4951	0.5259
3	x	x	x	0.6269	0.6273	0.4887	0.5102

Tabla 4.7: Resultados tras realizar las diferentes pruebas con data augmentation sobre una red SqueezeNet. Se muestran en color verde, azul y naranja los mejores resultados obtenidos en el primer, segundo y tercer set de datos respectivamente de ambas clases.

	ResNet	**DenseNet**	**SqueezeNet**
Rand puro acc.	0.6625	0.6773	0.6403
Rand por clases acc.	0.5371	0.5586	0.5259

Tabla 4.8: En esta tabla podemos ver los mejores resultados obtenidos para los datos test de cada una de las tres redes estudiadas. En ella podemos ver como los mejores resultados los consigue la DenseNet y los peores la SqueezeNet.

visto en la sección 3.5.2, las redes DenseNet, al estar fuertemente conectadas obtienen mejores resultados que otras redes cuando el dataset usado no es suficientemente grande, es lo que ocurre cuando usamos los datos procedentes de rand por clases. La red que peor resultados devuelve es la SqueezeNet, esto también entra dentro de lo normal ya que la diferencia de *accuracy* entre SqueezeNet y ResNet es pequeña, y por lo que se caracteriza SqueezeNet es por aumentar su capacidad de compresión manteniendo un nivel de *accuracy* óptimo, como vimos en la sección 3.5.3. Podemos ver los mejores valores obtenidos en cada una de las redes en la Tabla 4.8

En cuanto al uso del data augmentation, los dos tipos de datasets difieren un poco en este aspecto. Los sets pertenecientes a rand por clases tienden a funcionar mejor cuando añadimos los tres tipos de data augmentation (giro horizontal, giro vertical y *color jitter*). Sin embargo , los sets pertenecientes al tipo rand puro no tienen una mayor variabilidad en este aspecto ya que no siguen una tendencia tan clara como en el caso anterior, no obstante, se podría decir que, se obtienen mejores resultados cuando no se usa data augmentation, ya que en todas las redes, al menos existe un set, dentro de los mejores resultados, que cumple esta condición.
Por ultimo, en la comparativa entre nuestros resultados y los resultados obtenidos por los participantes de la competición *ISIC 2019*, podemos ver que los valores que nos devuelven las diferentes arquitecturas usando los datos creados mediante la técnica de rand por clases no son del todo competitivos. Sin embargo, aquellos que hemos obtenido

mediante la técnica de rand puro son bastante interesantes, ya que se superan a los primeros clasificados de la competición cuyos resultados podemos observar en la Tabla 2.4 que hemos visto en la sección 2.5. No obstante, la validez de los datos obtenidos está en duda debido al factor que comentamos anteriormente en la sección 4.2.2, en casi todas las pruebas observamos como el valor de la *accuracy* en test es mayor que en train, cuando teóricamente debería ser al revés.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

El objetivo principal del trabajo es estudiar como las usando diferentes redes y diferentes tipos de data augmentation podemos obtener mejores o peores resultados. Para el desarrollo de este estudio podemos remarcar una serie de aspectos que han sido claves:

- En primer lugar, destacar todos los conocimientos adquiridos gracias al desarrollo del estado del arte, en cual hemos hecho una introducción en el deep learning y en las redes neuronales convoluciones, asentando los conceptos básicos y estudiando las arquitecturas clásicas.

- Para el desarrollo del trabajo nos hemos basado en dos conjuntos de datos, creados a partir de dos técnicas distintas, por un lado la técnica de rand puro y por otro la técnica de rand por clase. Ambos conjuntos se han analizado de manera individual, es decir, con un cálculo de hiperparámetros diferenciado y con un análisis de las pruebas también separado.

- Se ha utilizado una red ResNet como base para realizar el ajuste de hiperparámetros, primero fijando el numero de épocas y a continuación, realizando una serie de pruebas para conseguir los valores de la tasa de aprendizaje y el tamaño del mini-batch más óptimos.

- Siguiendo el modelo del Trabajo Fin de Grado del año pasado [1] hemos realizado diferentes pruebas en las cuales variábamos el data augmentation para analizar como afectaban a las diferentes redes usadas (ResNet, DenseNet y SqueezeNet) y también a los dos conjuntos de datos.

- Una vez hemos realizado todas las pruebas y hemos analizado los resultados, concluimos que la técnica rand puro obtiene mejores resultados que la técnica rand por clases. No obstante la técnica de rand por clases tiene un menor tiempo de ejecución, por lo que podría tener algún otro uso en aplicaciones que no requieran mucha precisión. Además, observamos que la red que mejores resultados nos devuelve es la DenseNet.

5.2. Trabajo futuro

Tras haber analizado los objetivos que hemos conseguido a lo largo de este trabajo, haremos una serie de recomendaciones que se podrían realizar en el futuro para conseguir resultados más ambiciosos:

- Centrarnos en la técnica de rand puro para crear los datasets, ya que es la que mejores resultados ofrece. A partir de ahí se podría probar a crear datasets con otras distribuciones train/test diferentes a 80/20 como por ejemplo 70/30 o 60/40 y analizar si los resultados son mejores.
- Para intentar mejorar la técnica de rand por clases, se propone aumentar el límite de imágenes por clase, en nuestro caso fijado en 2000, para aumentar la *accuracy* y conseguir un mayor equilibrio entre tiempo de ejecución y fiabilidad.
- Se propone usar diferentes tipos de data augmentation como el uso del zoom (tanto zoom in como zoom out), rotaciones o translaciones para observar su comportamiento. También se sugiere sustituir tanto la ResNet como la SqueezeNet por otras redes, ya que son las que peores resultados nos ofrecen. Las redes que se proponen como sustitutas son la red Inception-v4, que utiliza unos módulos Inception y su estudio podría resultar interesante. La otra red propuesta es la red Dual Path Net, que de manera simplificada se podría decir que mezcla los conceptos de ResNet y DenseNet en una misma red.
- Una de los defectos del estudio es el hecho de que el conjunto de datos test devuelve valores de *accuracy* mas elevados que el conjunto de datos train, cuando, como hemos dicho a lo largo del ensayo, debería ser al revés. En un futuro trabajo se debería encontrar el motivo de esto y corregirlo si fuese posible.

Bibliografía

[1] C. Perez Lorenzo, *Detección precoz de cáncer de piel en imágenes basado en redes convolucionales.*, pp. 27–36. 06 2019. 2, 19, 25, 35

[2] T. Around, "Train, validation, test set in machine learning— how to understand." `https://medium.com/@tekaround/`, Noviembre 2018. Accedido en noviembre de 2019. 6

[3] N. Assawiel, "What to do when your training and testing data come from different distributions." `https://www.freecodecamp.org/news/`, Noviembre 2018. Accedido en noviembre de 2019. 6

[4] H. Zulkifli, "Understanding learning rates and how it improves performance in deep learning." `https://towardsdatascience.com/`, Enero 2018. Accedido en noviembre de 2019. 7

[5] S. Contreras and F. De la Rosa, "Aplicación de deep learning en robótica móvil para exploración y reconocimiento de objetos basados en imágenes," 09 2016. 8

[6] C. Dabakoglu, "What is convolutional neural network (cnn) ? — with keras." `https://medium.com/@cdabakoglu/what-is-convolutional-neural-network-cnn-with-keras-cab447ad204c`, Diciembre 2018. Accedido en noviembre de 2019. 9

[7] C.-C. J. Kuo, M. Zhang, S. Li, J. Duan, and Y. Chen, "Interpretable convolutional neural networks via feedforward design," *Journal of Visual Communication and Image Representation*, vol. 60, 03 2019. 11

[8] A. Pedraza, J. Gallego, S. Lopez, L. Gonzalez, A. Laurinavicius, and G. Bueno, "Glomerulus classification with convolutional neural networks," pp. 839–849, 06 2017. 12

[9] D. Siegmund, A. Prajapati, F. Kirchbuchner, and A. Kuijper, *An Integrated Deep Neural Network for Defect Detection in Dynamic Textile Textures: 6th International Workshop, IWAIPR 2018, Havana, Cuba, September 24–26, 2018, Proceedings*, pp. 77–84. 09 2018. 14

[10] S. Chilamkurthy, "Transfer learning for computer vision tutorial." `https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html#transfer-learning-for-computer-vision-tutorial`, 2017. Accedido en mayo de 2020. 19, 27

[11] G. de Inteligencia Artificial de Facebook, "Documentación pytorch." `https://pytorch.org/`, 2017. Accedido en mayo de 2020. 19

[12] G. Brain, "Documentación tensorflow." `https://www.tensorflow.org/`, 2015. Accedido en mayo de 2020. 19

[13] Y. Jia, "Documentación caffe." `https://caffe.berkeleyvision.org/`, 2017. Accedido en mayo de 2020. 19

[14] F. Chollet, "Documentación keras." `https://keras.io/`, 2015. Accedido en mayo de 2020. 19

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. 20, 21

[16] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. 21, 22

[17] F. Iandola, S. Han, M. Moskewicz, K. Ashraf, W. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," 02 2016. 22, 23