UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Detección precoz de cáncer de piel en imágenes basado en redes convolucionales.

Cristina Pérez Lorenzo.
Tutor: Juan Carlos San Miguel Avedillo.

Junio 2019

Detección precoz de cáncer de piel en imágenes basado en redes convolucionales.

Cristina Pérez Lorenzo
Tutor: Juan Carlos San Miguel Avedillo

Video Processing and Understanding Lab
Departamento de Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio 2019

Resumen

El objetivo de este Trabajo de Fin de Grado es la implementación de un sistema de *Deep Learning* basado en redes neuronales convolucionales, capaz de resolver un problema de clasificación de imágenes. Para ello, se tomará como punto de partida un caso concreto de aplicación: una competición internacional de diagnóstico de melanomas mediante imágenes.

Se comenzará haciendo un estudio del estado del arte, centrándonos concretamente en los conceptos básicos de las redes neuronales convolucionales. Posteriormente se dará una visión general de la competición, explorando las tres partes diferentes en las que se divide y los recursos que ofrece para llevarlas a cabo. También se realizará un estudio de las aportaciones realizadas por los participantes mejor clasificados en la edición previa a la realización de este trabajo (año 2018).

En el siguiente capítulo, se propondrá un diseño para la tarea 3 sobre el diagnóstico precoz de diferentes tipos de cáncer de piel. Se aplicará la metodología de transferencia de aprendizaje (o *transfer learning*) para entrenar nuestro modelo con las redes preentrenadas AlexNet, VGG y ResNet, de las cuales se describirá su arquitectura para más tarde lograr adaptarla a nuestros datos. Se hará una introducción a Pytorch, que será la librería software empleada para desarrollar el algoritmo de clasificación.

A continuación, se describirá el conjunto de datos utilizado, así como las métricas a utilizar. Se harán una serie de experimentos para determinar los parámetros iniciales para comenzar a entrenar el modelo. Una vez obtenidos, se realizarán las pruebas definitivas con las diferentes redes y divisiones del dataset, extrayendo conclusiones sobre los resultados.

Por último, basándonos en los estudios realizados y los resultados obtenidos, se harán una serie de propuestas como trabajo futuro.

Palabras clave

Deep Learning, redes neuronales convolucionales, melanoma, cáncer de piel, *transfer learning*, Pytorch.

Abstract

The objective of this Final Degree Thesis is the implementation of a Deep Learning system based on convolutional neural networks, capable of solving an image classification problem. To accomplish this, a specific use case will be taken as a starting point: an international melanoma diagnosis competition using images.

First, we will begin studying the state of the art, focusing specifically on the basic concepts of convolutional neural networks. Afterwards, we will give an overview of the competition, exploring the three different parts in which it is divided and the resources it offers to carry them out. There will also be a study of the contributions made by the best ranked participants in the previous edition to the realization of this work (year 2018).

In the next chapter, we will propose a design for the task 3 about early diagnosis of different types of skin cancer. The transfer learning methodology will be applied to train our model with the pre-trained networks AlexNet, VGG and ResNet, of which its architecture will be described and later adapted to our data. There will be an introduction to Pytorch, which will be the software library used to develop the classification algorithm.

Then, we will describe the data set used, as well as the metrics to use. We will do a series of experiments to determine the initial parameters to start training the model. Once we have obtained them, we will carry out the definitive tests with the different networks and divisions of the dataset, drawing conclusions about the results.

Finally, based on the studies carried out and the results obtained, we will make a series of proposals as future work.

Keywords

Deep learning, convolutional neural networks, melanoma, skin cancer, transfer learning, Pytorch.

Agradecimientos

En primer lugar quiero dar las gracias a mi tutor, Juan Carlos San Miguel Avedillo, por su dedicación y el tiempo que ha invertido para resolver las dudas que han ido surgiendo a lo largo de este trabajo.

También dar las gracias a mi familia por apoyarme y confiar siempre en mí, en especial a mi madre, sin la que jamás habría podido llegar tan lejos.

Tampoco puedo olvidarme de todas esas personas que han aparecido en mi vida en estos cuatro años, con las que he tenido el placer de formar una gran piña y, juntos, hemos podido con todo.

Gracias a mis compañeros de trabajo por todos los consejos y apoyo. También a mis amigos, por supuesto, por los ratitos de desconexión cuando más lo necesitaba.

Finalmente, darle las gracias a Javi, por soportarme en mis peores momentos y hacer que, pase lo que pase, siempre acabe con una sonrisa en la cara y la motivación suficiente para comerme el mundo.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

1.1. Motivación

Hoy en día el aprendizaje profundo (*Deep learning*) está en auge, hasta el punto de poder aplicarlo a cualquier ámbito imaginable, llegando incluso al campo de la medicina, por ejemplo, para el diagnóstico asistido por ordenador, para hacer predicciones a partir de conjuntos de datos, para procesar imágenes médicas o para distinguir entre tumor maligno y benigno. La motivación de este trabajo es precisamente este tipo de aplicaciones, ya que permiten mejorar los diagnósticos y, así, facilitar el trabajo de los expertos. Concretamente, este proyecto se va a centrar en la detección precoz de diferentes tipos de cáncer de piel, el cual se encuentra entre los tipos de cáncer más comunes actualmente, siendo producido por el desarrollo de células cancerosas en cualquiera de las capas de la piel.

En muchos casos, la detección de las diferentes clases de cáncer de piel es susceptible a la detección temprana mediante la inspección visual de expertos, por lo tanto, el reto de este trabajo consistirá en desarrollar una herramienta automática para la clasificación de los tipos de cáncer de piel basada en tratamiento de imágenes. Dicho clasificador se pondrá a prueba con diferentes imágenes de cáncer de piel, con las que habrá sido entrenado previamente, y se comprobará su funcionamiento empleando diferentes parámetros y redes neuronales para su entrenamiento.

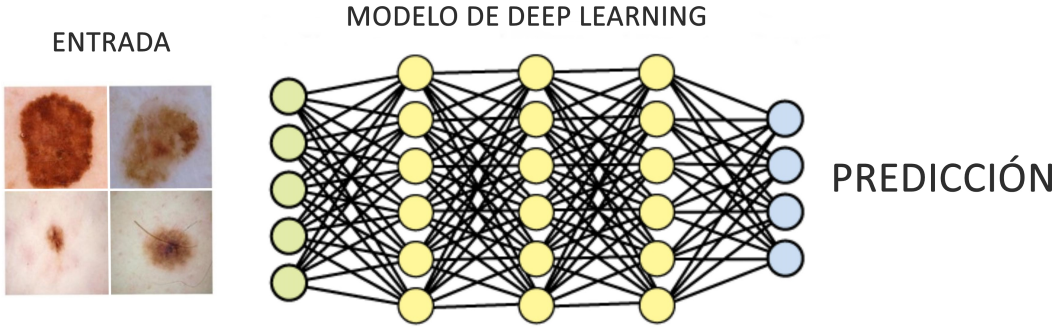


Figura 1.1: Esquema de predicción de tipo de cáncer de piel mediante un modelo de *Deep Learning*.

Existe una competición a nivel internacional (*ISIC challenge* [1]), centrada en la clasificación de este tipo de lesiones cutáneas, hacia la cual se enfocará este trabajo. Proporciona un *dataset* que contiene imágenes etiquetadas con el tipo de cáncer de piel del que se trata, el cual se utilizará para nuestro entrenamiento, así como la documentación de los trabajos realizados por los participantes para tener una visión inicial de los modelos y técnicas empleadas.

1.2. Objetivos

El objetivo de este proyecto consiste en el estudio y desarrollo de diferentes técnicas de *Deep Learning*, con el fin de desarrollar una solución capaz de resolver el problema propuesto por la competición *ISIC 2018: Skin Lesion Analysis Towards Melanoma Detection* [1]. Se realizará un estudio previo de los elementos fundamentales del *Deep Learning*, de los aportes de otros participantes anteriores a la misma competición y de diferentes pruebas sobre los datos proporcionados, con el fin de elegir una configuración inicial adecuada para los entrenamientos posteriores. Por último, se hará una comprobación de los resultados obtenidos para cada una de las distintas configuraciones utilizadas, y se valorará la precisión de los mejores resultados obtenidos, así como las posibles mejoras que se podrían implementar. El objetivo final será implementar una solución capaz de clasificar lo más fielmente posible las imágenes proporcionadas por la competición en los 7 diferentes tipos de cáncer de piel.

Por tanto, el objetivo mencionado puede ser desglosado en varios objetivos generales:

1. Estudio del estado del arte: Se comenzará haciendo un estudio inicial sobre los conceptos básicos del *Deep Learning*, así como de las redes neuronales y, en concreto, las redes neuronales convolucionales. Además, se hará un estudio de la competición ISIC 2018, así como de las propuestas presentadas por los participantes mejor clasificados.

2. Diseño de una propuesta: partiendo del estudio previo realizado, se harán una serie de pruebas para definir los parámetros con los que vamos a entrenar el modelo y se seleccionarán una serie de redes neuronales preentrenadas para utilizarlas en el entrenamiento.

3. Realización de experimentos: Una vez determinados los parámetros y redes a utilizar, se procederá a realizar una serie de experimentos con el fin de observar la repercusión obtenida sobre el modelo.

4. Conclusiones y evaluación de mejoras: Se realizará un análisis de todo el material disponible para este proyecto, así como una comparación de los resultados obtenidos en cuanto a la clasificación de los distintos tipos de cáncer de piel y los obtenidos por el resto de participantes de la competición ISIC 2018. Además, se propondrán una serie de trabajos futuros que podrían mejorar el sistema implementado.

1.3. Organización de la memoria

La memoria consta de los siguientes capítulos:

- Capítulo 1. Capítulo de introducción con los objetivos del TFG.
- Capítulo 2. Capítulo en el que se habla del estado del arte y se presentan los conceptos más importantes que usaremos durante el trabajo.
- Capítulo 3. Capítulo donde se lleva a cabo el diseño y desarrollo del trabajo, haciendo un estudio de las técnicas y herramientas a utilizar.
- Capítulo 4. Capítulo de evaluación en el que se determinan unos parámetros base para llevar a cabo los experimentos finales.
- Capítulo 5. Capítulo en el que se resumirán y valorarán los resultados obtenidos, y se propondrán algunas ideas sobre posibles trabajos futuros.
- Bibliografía.

Capítulo 2

Estado del arte: Deep Learning

2.1. Conceptos básicos

El aprendizaje profundo (*Deep Learning*) define un conjunto de técnicas de la Inteligencia Artificial (AI) que consiste en una técnica de aprendizaje automático que enseña a los ordenadores a hacer lo que resulta natural para las personas: aprender mediante ejemplos. Los modelos de *Deep Learning* se entrenan mediante un amplio conjunto de datos etiquetados y arquitecturas de redes neuronales o *Neural Networks* (NN) formadas por distintas capas, pudiendo obtener resultados con una precisión que incluso supere el rendimiento humano. Estas capas se organizan en una jerarquía de creciente complejidad y abstracción, de forma que el nivel inicial de la red aprende conceptos simples, el siguiente nivel toma esta información sencilla, la combina, compone una información algo un poco más compleja y se lo pasa al tercer nivel, y así sucesivamente (Figura 2.1).

Las redes neuronales están basadas en los sistemas nerviosos biológicos y, por lo tanto, están constituidas por un conjunto de unidades llamadas neuronas o nodos conectados unos con otros. Las neuronas reciben señales (*inputs*) de otras neuronas y en función de las señales recibidas, una neurona envía a su vez una señal a otras neuronas (Figura 2.2). Profundizaremos en las redes neuronales convolucionales o *Convolutional Neural Networks* (CNN), que son un tipo de redes neuronales profundas.

2.1.1. Train/Validation/Test

Para entrenar un modelo de *Deep Learning*, es necesario dividir los datos en tres conjuntos: train, validation y test.

- Conjunto de entrenamiento (*training*): conjunto de datos con el que se construyen distintos modelos para resolver el problema. Se seleccionan uno o varios modelos finales. Una vez entendido el objetivo para el cual se quiere entrenar la red neuronal, se clasifican los datos mediante "etiquetas", siendo éstas las distintas clases posibles de resultados en los problemas de clasificación, o mediante números o vectores en los problemas de regresión. Para definir la arquitectura, se redimensionan todas las imágenes al mismo tamaño, se normalizan los datos para homogeneizar y se experimenta con

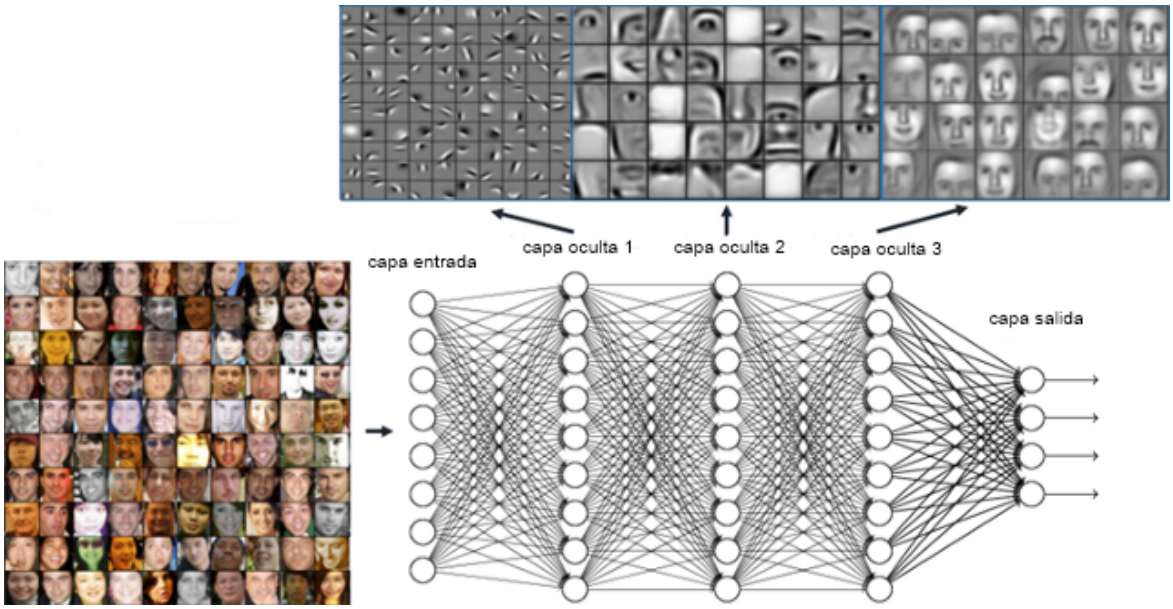
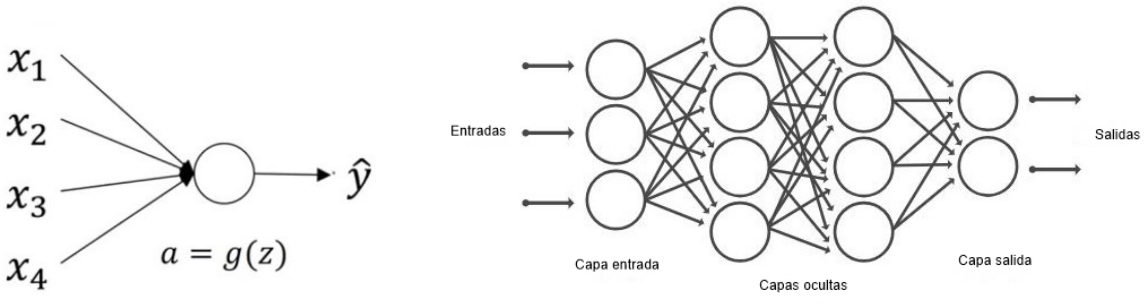


Figura 2.1: Clasificación en base a expresiones faciales empleando un modelo de *Deep Learning*. Muestra las imágenes de entrada, así como las diferentes características que se extraen en cada capa. Extraído de [2].



(a) (b)

Figura 2.2: Ejemplo de (a) una sola neurona y (b) de una red neuronal.

diferentes espacios de colores.

- Conjunto de validación (*validation*): conjunto de datos con el que se validan los modelos finales del conjunto de entrenamiento y se determina el modelo definitivo según: modelo con mejor ajuste a los datos, modelo más "cercano" al problema de predicción. Se suele utilizar para el ajuste de los hiperparámetros según los resultados obtenidos.
- Conjunto de prueba (*test*): se realizan pruebas de funcionamiento del modelo y se obtiene el error "real" cometido respecto al modelo seleccionado. Solo se aplica una vez que el modelo ya esté completamente entrenado. Los datos del conjunto de prueba deberían ser de la misma distribución de datos que los del conjunto de validación.

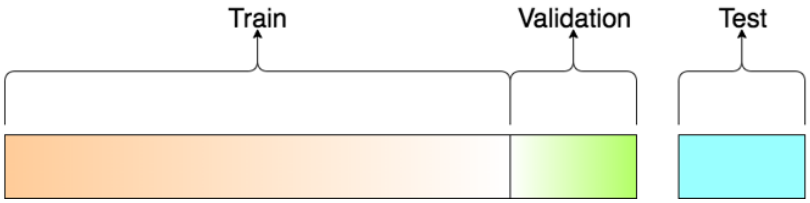


Figura 2.3: Ejemplo de división de un conjunto de datos en *training*, *validation* y *test*.

No existe una regla general sobre el tamaño del que debe ser cada conjunto, aunque es habitual dividir el conjunto de datos de forma que el mayor sea el de train, y los de validation y test sean más pequeños (Figura 2.3). En *Deep Learning* de forma genérica se suele utilizar 80 % entrenamiento, 10 % validación y 10 % prueba. Para un modelo con muchos hiperparámetros será conveniente tener un gran conjunto de validación, mientras que si no tiene hiperparámetros se podría incluso prescindir de este conjunto.

2.1.2. Bias/Variance

- Error debido al sesgo (*bias*): Diferencia entre la predicción esperada (o promedio) de nuestro modelo y el valor correcto que tratamos de predecir. Por lo tanto, el sesgo mide lo lejos que están las predicciones de los modelos buscados. Problema de alto sesgo: subajuste (*underfitting*). Se dice que un modelo sufre de subajuste cuando no es capaz de capturar la tendencia subyacente de los datos, es decir, cuando el modelo no se ajusta lo suficientemente bien a los datos.
- Error debido a la varianza (*variance*): Variabilidad de una predicción del modelo para un punto de datos determinado, es decir, indica cuánto varían las predicciones para un punto dado entre diferentes realizaciones del modelo. Problema de alta varianza: sobreajuste (*overfitting*). Se dice que un modelo sufre de sobreajuste cuando está demasiado ajustado al conjunto de datos de entrenamiento y en vez de encontrar generalidades sobre los datos, comienza a aprender del ruido y las particularidades de este conjunto de datos. Como consecuencia, al producirse un sobreentrenamiento, el algoritmo de aprendizaje puede quedar ajustado a unas características muy específicas de los datos de entrenamiento que no tienen relación causal con la función objetivo.

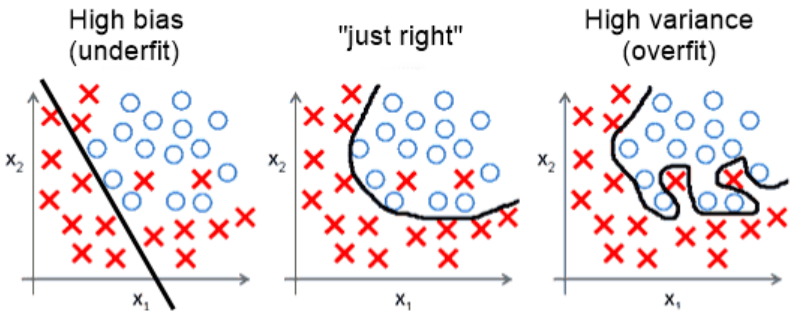


Figura 2.4: Ejemplo gráfico del problema de alto sesgo (*underfit*) y alta varianza (*overfit*), frente a lo que sería un buen resultado ("*just right*").

2.1.3. Tamaño del batch

Pasar todo el conjunto de datos a la red neuronal puede ser muy costoso computacionalmente y muy lento, por lo tanto, lo que se hace es dividirlo en lotes (*batches*) o conjuntos de datos. El tamaño del lote o *batch size* es el número total de datos de entrenamiento que contiene cada batch, hay tres opciones:

- Modo *batch*: donde el tamaño del batch es igual al conjunto de datos total, por lo que los valores de iteración y época son equivalentes.
- Modo de *mini batches*: donde el tamaño del batch es mayor que uno, pero menor que el tamaño total del conjunto de datos.
- Modo estocástico: donde el tamaño del batch es igual a uno, por lo tanto, los parámetros del gradiente y pesos de la red neuronal se actualizarán después de cada muestra.

Normalmente las redes neuronales se entrenan más rápido con mini batches debido a que se actualizan los pesos después de cada propagación y, además, requiere menos memoria. Por lo tanto, el modo empleado en los experimentos será el de mini batches.

Lo ideal para conseguir un estudio completo sobre los distintos tamaños de mini-batch sería probar los valores del mini-batch más grandes llegando incluso a igualar el tamaño del batch.

Por lo general, a menor tamaño de mini batch los pesos oscilarán más sin poder llegar a obtener un resultado generalizado debido a que la cantidad de datos será insuficiente. Según se va aumentando el tamaño, las oscilaciones serán mucho más suaves, pero también se dará una convergencia más lenta.

2.1.4. Iteración y época

Una iteración es una sola actualización de los pesos de un modelo durante el entrenamiento. Consiste en el cómputo de los gradientes de los parámetros con respecto a la pérdida en un solo lote (*batch*) de datos.

Para el entrenamiento de datos con una red neuronal lo más normal es que no sea suficiente con pasar a través de todos los datos una sola vez, y por tanto es necesario pasar el

conjunto de datos completo varias veces a la misma red neuronal, es por ello por lo que se definen las épocas. Una época (*epoch*) es el recorrido de entrenamiento completo por todo el conjunto de datos a través de la red neuronal, realizando una pasada hacia delante y una pasada hacia atrás para actualizar los pesos, por todos los ejemplos de entrenamiento.

Por lo tanto, siendo N el número total de ejemplos, las iteraciones necesarias para completar una época resultan N/(tamaño batch). Por ejemplo, teniendo 1000 datos de entrenamiento, y su tamaño de lote es 500, entonces se necesitarán 2 iteraciones (1000/500) para completar 1 época.

Según se aumenta el número de épocas, más veces se cambian o actualizan los pesos de las capas de la red neuronal, produciendo una curva desde ajuste insuficiente (*underfitting*), a óptima, e incluso a un ajuste excesivo (*overfitting*).

2.1.5. Tasa de aprendizaje

La tasa de aprendizaje o *learning rate* es un hiperparámetro que controla cuánto estamos ajustando los pesos de la red con respecto al gradiente de pérdida. Cuanto menor sea el valor, más lento viaja a lo largo de la pendiente descendente, lo cual sería una forma de asegurarse de no perder ningún mínimo local, pero podría llevarnos mucho tiempo hasta que llegara a converger, o quedarse estancado y no llegar nunca. Por otro lado, un valor muy grande de *learning rate* podría tener oscilaciones tan grandes que nunca llegue a alcanzar el punto óptimo o incluso divergir. Se pueden ver varios ejemplos de diferentes comportamientos de la curva de pérdida según el *learning rate* escogido en la Figura 2.5.

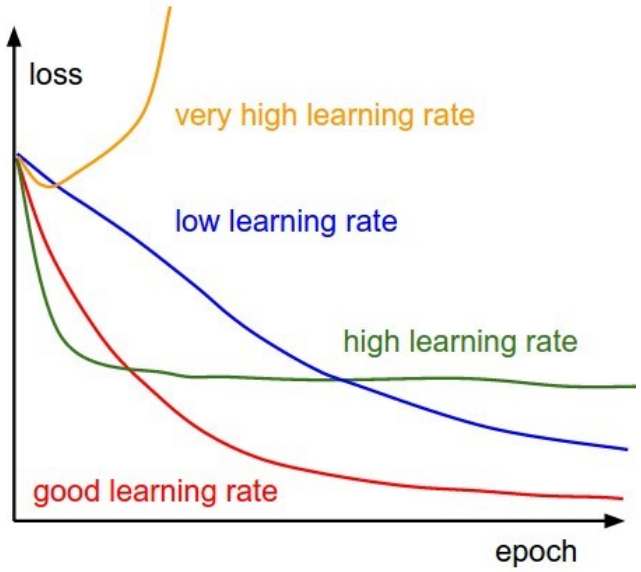


Figura 2.5: Curvas *loss/epochs* según el *learning rate*. La curva amarilla se da para un *learning rate* muy alto que no llegará a converger, mientras que la azul, que es para un *learning rate* muy bajo, converge pero muy lentamente. Lo ideal sería encontrar un valor que lleve a una convergenia similar a la curva roja.

2.1.6. Data Augmentation

El aumento de datos de imágenes o *data augmentation* es una técnica que se utiliza para ampliar de forma artificial el tamaño de un conjunto de datos de entrenamiento mediante la creación de versiones modificadas de las imágenes de las que se dispone en el conjunto de datos.

Capacitar a los modelos de redes neuronales con más datos puede dar lugar a modelos mucho más hábiles y precisos, ya que las transformaciones del *data augmentation* pueden crear variaciones de las imágenes que mejoren la capacidad de los modelos entrenados para generalizar lo que han aprendido a nuevas imágenes.

Se pueden emplear una infinidad de transformaciones diferentes a las imágenes (Figura 2.6), algunas de las más comunes en *data augmentation* son los volteos verticales y horizontales, rotaciones, recortes, y modificaciones en el brillo, contraste y saturación.

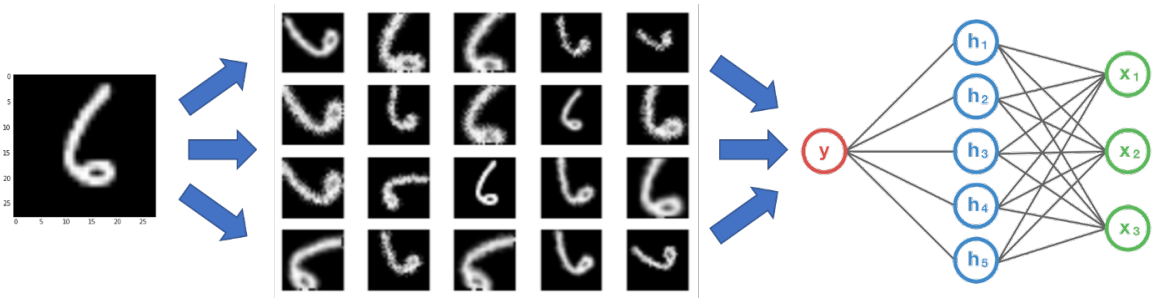


Figura 2.6: La imagen de entrada es el número 6 escrito a mano perteneciente al conjunto MNIST, al cual se le han aplicado diferentes transformaciones de data augmentation, ampliando el conjunto de datos de una imagen de partida, a 20 imágenes que pasarán al modelo para comenzar el entrenamiento.

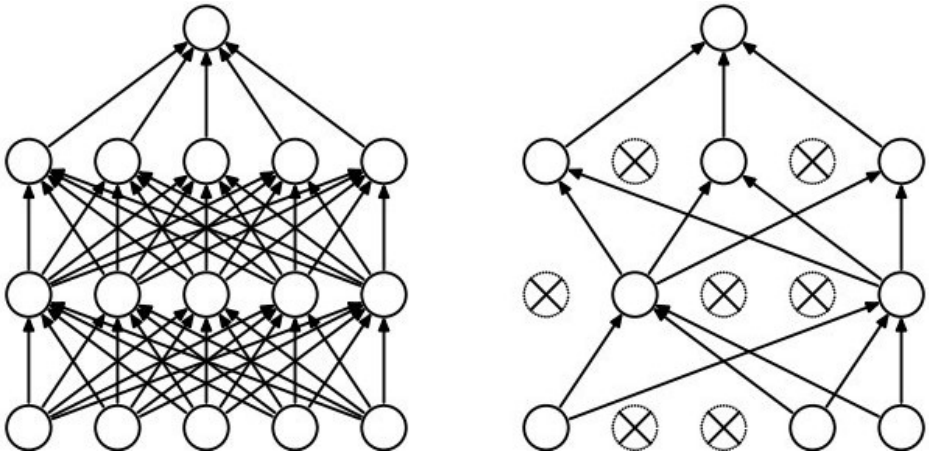
2.1.7. Regularización: Dropout

La regularización se emplea para solucionar o prevenir el problema de alta varianza (*overfit*) y reducir los errores en la red neuronal. El parámetro de regularización se define como λ. Con la regularización lo que se pretende es, además de minimizar la función de coste j, reducir la complejidad, lo que se denomina minimización del riesgo estructural:

Minimize (Loss(Data|Model) + (lambda) x Complexity(Model))

El *dropout* es un tipo de regularización que consiste.en recorrer cada una de las capas de la red y establecer la probabilidad de eliminar algunos de los nodos de cada una (Figura 2.7). En cada iteración (mini-batch), se adjudica a cada neurona de capa una probabilidad p de ser eliminados, empleando normalmente valores pequeños en las primeras capas y valores mayores en las capas intermedias.

Como efectos, el *dropout* hace que las neuronas sean menos dependientes de la salida de las neuronas conectadas, además, la red logra aprender características más robustas que son útiles para más subconjuntos de neurona, y puede promediar sobre diferentes redes entrenadas con diferentes inicializaciones aleatorias.



(a) (b)

Figura 2.7: Ejemplo de aplicación de *dropout*. (a) Red Neuronal estándar con 2 capas ocultas. (b) Resultado de aplicar dropout a la red de la izquierda. Las neuronas con una cruz son las que se han descartado. Extraída de [3].

2.1.8. Inicialización de una red neuronal

Antes de entrenar una red neuronal, el primer paso a llevar a cabo es la inicialización de los parámetros, puesto que si se realiza correctamente, la optimización se logrará en menos tiempo, ya que en el entrenamiento se tratará de adaptar estos parámetros (pesos y sesgos) de forma que la red se comporte de la manera deseada.

Se define como ω_i la matriz de pesos y como b el vector de sesgos. Se emplean funciones de activación que utilizan una suma ponderada (Ecuación 2.1), es decir, se encarga de devolver una salida (generalmente en un rango determinado) a partir de un valor de entrada. Para un número de dimensiones i, el resultado predicho se podría expresar como z. Podemos interpretar que cada peso ω_i representa la influencia relativa de la entrada por la cual se multiplica, x_i.

$$z = b + \sum_i \omega_i x_i \tag{2.1}$$

Cada entrada (*input* x_i) tiene un peso asociado ω_i, el cual se inicializa de forma aleatoria, por lo general, y con valores pequeños, e irá variando en el proceso de aprendizaje con el fin de minimizar el error cometido entre la salida obtenida por la red neuronal y la salida deseada. Generalmente los sesgos se inicializan a cero.

2.1.9. Descenso de Gradiente

El descenso de gradiente es uno de los algoritmos de optimización más populares en el los modelos de *Deep Learning*. Este algoritmo también es conocido como *Batch* o *Mini Batch* descenso de gradiente.

El algoritmo de *Batch* calcula el error para cada ejemplo del conjunto de datos de entrenamiento, pero solo actualiza el modelo después de haber evaluado todo el conjunto (cada

trainig epoch). Las actualizaciones del modelo y, por lo tanto, la velocidad de entrenamiento, pueden ser muy lentas para grandes conjuntos de datos.

Mini Batch es una variación del algoritmo de descenso de gradiente que divide el conjunto de datos de entrenamiento en lotes pequeños, de forma que se calcula el error y se va actualizando el modelo por lotes, sin necesidad de evaluar todo el conjunto para poder actualizarlo. Al ser más alta la frecuencia de actualización, permite una convergencia más robusta, evitando los mínimos locales.

Se requiere la función de pérdidas de *Batch* para guiar el entrenamiento $J(\theta)$ (Ecuación 2.2) que compara la salida obtenida con el resultado esperado. Se realizan una serie de comparaciones sobre un número de N entradas/salidas, donde $\widehat{y}_i$ es la salida de la red para una entrada i dada, y el resultado esperado (*ground-truth*) se corresponde con y_i. Otra forma muy común de determinar el error obtenido, es el error cuadrático medio o *Mean Square Error* (MSE) (Ecuación 2.3) que mide el promedio de los errores al cuadrado, es decir, la diferencia entre el estimador y lo que se estima.

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 + \lambda^p \sqrt{\sum_{\omega j} |\omega j|^p} \tag{2.2}$$

$$J_{MSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{2.3}$$

2.1.10. Ajuste de hiperparámetros

Los hiperparámetros se definen como aquellos parámetros que contienen los datos que rigen el entrenamiento, por lo tanto sus valores se establecen antes de comenzar el proceso de aprendizaje. Mientras que los parámetros del modelo especifican cómo transformar los datos de entrada en la salida deseada, los hiperparámetros definen cómo se estructura realmente nuestro modelo. Algunos de los hiperparámetros a ajustar son el número de capas, el número de neuronas por cada capa, *learning rate* o el tamaño del *mini batch*.

Lo ideal sería seguir un proceso de ajuste consistente en definir un modelo, posteriormente determinar el rango de posibles valores para todos los hiperparámetros y, por último, definir un criterio evaluativo que permita juzgar el modelo, como la función de coste, y determinar los mejores hiperarámetros en base a los resultados obtenidos. Este ajuste de hiperparámetros se suele llevar a cabo sobre el conjunto de validación (Capítulo 2.1.1).

2.2. Redes Neuronales Convolucionales

Las Redes Neuronales Convolucionales son un tipo de redes neuronales multicapa que se usan para procesar imágenes. Están basadas en operaciones de convolución, lo cual consiste en filtrar una imagen utilizando una máscara, y cada píxel de salida es combinación lineal de los píxeles de entrada (Figura 2.8). Dicha entrada es, generalmente, una imagen W x H x D (ancho x alto x número de canales). Al emplear convoluciones se reduce la carga computacional del sistema.

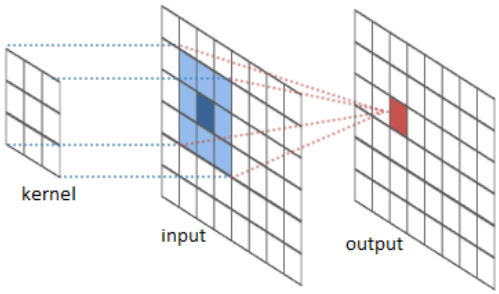


Figura 2.8: Resultado de convolución con un *kernel* de 3x3. Tomando el píxel mostrado en azul oscuro, se calcula la media ponderada de sus vecinos determinados por el tamaño y forma del *kernel* y se produce el píxel de salida (color rojo). Extraído de [4].

Estas redes se forman empleando capas de 3 tipos principalmente:

- Capas Convolucionales
- Capas de *Pooling*
- Capas Totalmente Conectadas (*Fully Connected*)

La arquitectura básica de las redes neuronales convolucionales o CNN (*Convolutional Neural Networks*), entrega una salida para toda la imagen, la cual está totalmente conectada (todas conectadas con todas) (Figura 2.9).

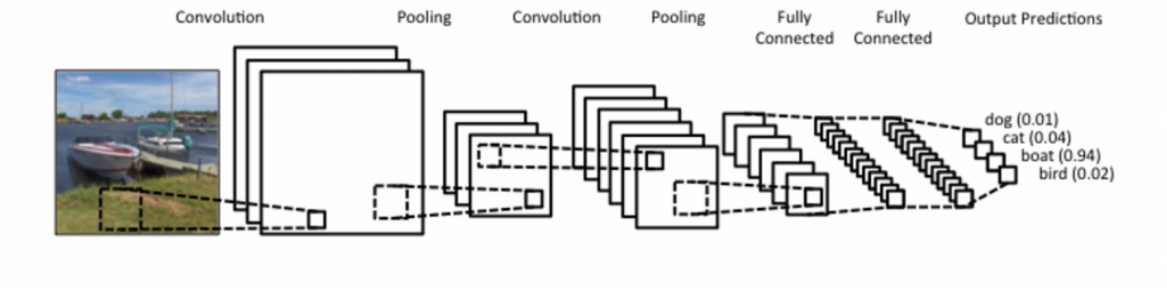


Figura 2.9: Esquema básico de una red neuronal convolucional. Extraído de [5].

2.2.1. Capa Convolucional

La capa convolucional es el bloque de construcción central de una red convolucional que realiza la mayor parte del trabajo pesado computacionalmente. Estas capas tienen K filtros (o *kernels*) de dimensión (W x H x D), elegida por el diseñador. Cada filtro, mediante convolución, genera un mapa de rasgos o características de tamaño (WH+ 1) x (W-H+1)x P, donde P es el número de filtros que se utilizarán.

El modo de operación en estas capas consiste en superponer el filtro y la imagen de entrada, se calcula la convolución entre los elementos de la imagen y del filtro, y el resultado se va almacenando en la llamada matriz de activación. Una vez se ha recorrido toda la imagen

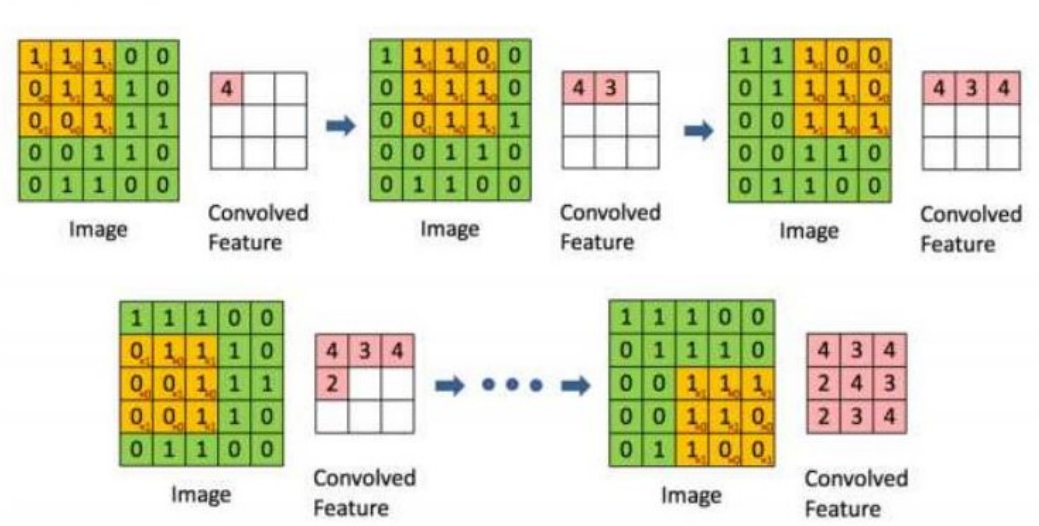


Figura 2.10: Ejemplo de obtención de matriz de activación mediante operaciones básicas de convolución. Se emplea un filtro de 3x3 con valores $[1, 0, 1; 0, 1, 0; 1, 0, 1]$, el cual se va convolucionando píxel a píxel de izquieda a derecha y de arriba a abajo.

mediante este proceso de forma iterativa (de izquierda a derecha y de arriba a abajo), se obtiene la matriz de activación (Figura 2.10).

Un mismo filtro (neurona) sirve para extraer el mismo rasgo en cualquier parte de la imagen, teniendo en cuenta el carácter estacionario de la imagen (los rasgos existentes en una parte de la imagen, pueden ser los mismos que existen en otra zona distinta), permitiendo reducir el número de conexiones y parámetros a entrenar.

Para cada capa, además de elegir el tamaño del filtro a utilizar, hay dos parámetro más que se pueden cambiar para modificar el comportamiento de cada capa: *stride* y *padding*. El *stride* (o zancada) determina el número de píxeles que se irá desplazando la ventana (filtro) al realizar la convolución. El *padding* (o relleno) consiste en añadir más píxeles auxiliares (ceros) en los bordes de la entrada, de forma que al hacer la convolución con el filtro, obtengamos una salida del tamaño deseado.

2.2.2. Capa Pooling

Se sub-muestrea cada resultado generado en las capas convolucionales mediante la operación *max pooling* o *average pooling* sobre regiones contiguas de tamaño (P x P). *Max pooling* consiste en seleccionar únicamente el valor mayor de una ventana o región de la matriz de activación obtenida en la capa de convolución, descartando el resto de valores. *Average pooling* opera de la misma forma, pero el valor obtenido es la media de los valores de la ventana o región. Por lo tanto la salida será el máximo o la media de la entrada en una ventana (Figura 2.11).

Si la matriz de entrada a la capa *pooling* es de tamaño (W x H x D), la matriz de salida será de (W/P x H/P x D). Al emplear *pooling* se reducen las dimensiones de la matriz de activación y, por lo tanto, se disminuye el número de parámetros y tiempo de computación. De esta forma, será más sencillo reducir el problema de *overfitting* explicado anteriormente

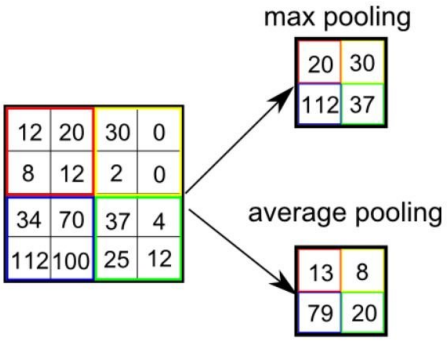


Figura 2.11: Ejemplo de operaciones de *max* y *average pooling*, con *stride*=2.

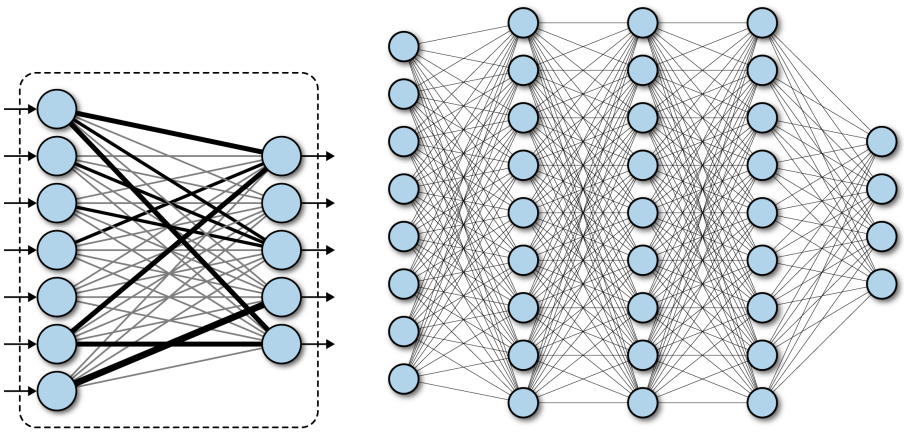
(Capítulo 2.1.2), además de hacer que el modelo sea invariante a pequeñas transformaciones.

2.2.3. Capa Fully Connected

Las neuronas de una capa totalmente conectada (*fully connected*) tienen conexiones completas con todas las activaciones de la capa anterior, como podemos ver en la Figura 2.12. La dimensión de cada capa de salida depende de la dimensión de cada capa de entrada. Las capas *fully connected* se aplican al final del esquema de las CNN. Finalmente, la última capa consistirá en un clasificador (binario, *Softmax*...) que determina a qué clase pertenece la imagen de entrada.

Introduciéndonos en las matemáticas, podemos representar la entrada a la capa totalmente conectada como $x \epsilon \mathbb{R}_m$, una función no lineal como σ y los parámetros entrenables o pesos como ω, la i-ésima salida $y_i \epsilon \mathbb{R}$ se pude definir como se muestra en la Ecuación 2.4.

$$y_i = \sigma(\omega_1 x_1 + ... + \omega_m x_m) \tag{2.4}$$



(a) (b)

Figura 2.12: (a) Capa totalmente conectada en una red neuronal. (b) Red multicapa totalmente conectada.. Extraído de [6].

2.2.4. Clasificación Softmax

Esta clasificación permite clasificar el problema en K clases diferentes. Su resultado es la probabilidad que tiene la entrada de pertenecer a una determinada clase. La salida será una vector de K elementos donde cada componente corresponde a la probabilidad de pertenecer a cada clase.

Se emplea para "comprimir" un vector K-dimensional, $\hat{y}$, de valores reales arbitrarios en un vector K-dimensional, $\sigma(\hat{y})$, de valores reales en el rango [0,1]. La función está dada por las Ecuaciones 2.5 y 2.6.

$$\sigma : R^K \rightarrow [0,1]^K \tag{2.5}$$

$$\sigma(\hat{y}_j) = \frac{e^{\widehat{y_j}}}{\sum_{j=1}^{K} e^{\widehat{y_j}}} \tag{2.6}$$

Selecciona la $\sigma(\hat{y}_j)$ con el valor máximo (el índice correspondiente es la clase seleccionada). El efecto conseguido consiste en resaltar los valores más grandes, y suprimir aquellos que quedan significativamente por debajo del valor máximo, con el objetivo de realizar una buena clasificación según las diferentes clases del problema.

Es utilizada como capa final de los clasificadores basados en redes neuronales.

2.3. Competición ISIC 2018

2.3.1. Descripción de la competición y dataset proporcionado

El objetivo de este desafío es desarrollar herramientas de análisis de imágenes que permitan el diagnóstico automático de melanoma a partir de imágenes dermatoscópicas. El desafío se divide en tres tareas:

- Tarea 1: Segmentación de la lesión. El objetivo es presentar predicciones automatizadas de los límites de la segmentación de la lesión dentro de las imágenes dermatoscópicas, con detección de contorno de imagen.
- Tarea 2: Detección de atributos de lesión. La finalidad es obtener predicciones automatizadas de las ubicaciones de los atributos dermatoscópicos (patrones visuales de lesiones de la piel establecidos cínicamente) dentro de las imágenes dermatoscópicas, mediante segmentación de imágenes.
- Tarea 3: Clasificación de lesión cutánea. El objetivo es presentar predicciones automatizadas de clasificación de enfermedades dentro de imágenes dermatoscópicas, con clasificación de imágenes de múltiples clases.

Nuestros datos han sido extraídos del conjunto de datos proporcionado por la competición "*ISIC 2018: Skin Lesion Analysis Towards Melanoma Detection*" [1, 7]. Por lo tanto, los datos de entrada al clasificador serán imágenes de lesiones dermatoscópicas en formato JPEG

(provenientes del conjunto de datos HAM10000) de pacientes que se presentaron para la detección de cáncer de piel. HAM10000 (Human Against Machine, con 10000 imágenes de entrenamiento) contiene una gran colección de imágenes dermatoscópicas de múltiples fuentes de lesiones cutáneas pigmentadas comunes.

2.3.2. Tarea 3: Clasificación de lesión cutánea

Nos centraremos en la tarea 3, en la que el objetivo es presentar predicciones automatizadas de clasificación de enfermedades dentro de imágenes dermatoscópicas.

La aplicación clínica en la clasificación de lesiones de la piel tiene dos objetivos eventualmente: brindar información específica y opciones de tratamiento para una lesión, y detectar el cáncer de piel con una sensibilidad y especificidad razonables.

El conjunto de datos empleado para esta tarea consta de las 10000 imágenes dermatoscópicas proporcionadas por la competición, que pueden servir como un conjunto de entrenamiento para el aprendizaje académico automático. Los casos incluyen una colección representativa de todas las categorías diagnósticas importantes en el ámbito de lesiones pigmentadas.

Los datos de respuesta del clasificador son conjuntos de clasificaciones binarias para cada uno de los 7 estados de enfermedad (Figura 2.13), lo que indica el diagnóstico de cada imagen de lesión de entrada. Todos estos datos están codificados dentro un único archivo CSV, con cada respuesta de clasificación en una fila. Las columnas del archivo serán:

1. image: un identificador de imagen de entrada del formulario ISIC_
2. MEL: "Melanoma" diagnóstico confianza
3. NV: "Nevo melanocítico" diagnóstico confianza
4. BCC: "Carcinoma basocelular" diagnóstico de confianza
5. AKIEC: "Queratosis actínica / enfermedad de Bowen (carcinoma intraepitelial)" diagnóstico de confianza
6. BKL: “Queratosis benigna (lentigo solar, queratosis seborreica, queratosis similar al liquen plano)” diagnóstico de confianza
7. DF: "Dermatofibroma" diagnóstico de confianza
8. VASC: “Lesión vascular” diagnóstico confianza

2.3.3. Ranking de participantes en la competición

Antes de abordar el diseño de la solución a este problema de clasificación, se ha hecho un estudio de las mejores aproximaciones obtenidas en la tarea 3 de la competición ISIC 2018. En la Tabla 2.1se puede observar un resumen de las características más notables de cada uno de los proyectos.

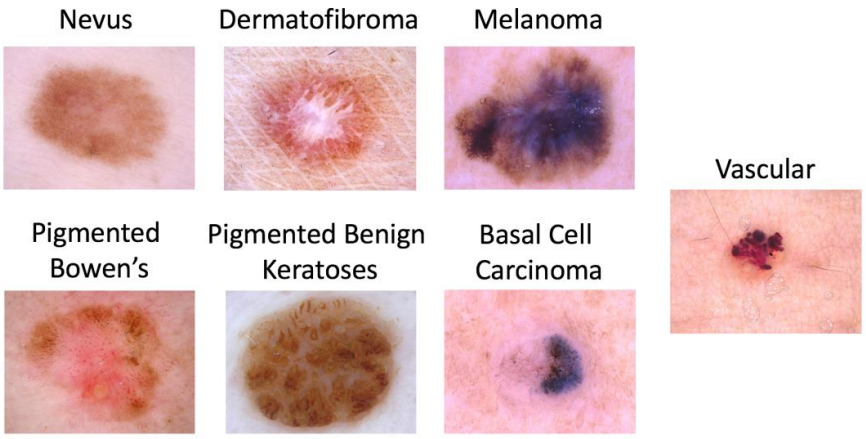


Figura 2.13: Posibles clasificaciones de las enfermedades. Extraído de [1].

Todos los participantes estudiados en la tabla utilizan redes neuronales convolucionales y residuales, además estas no las entrenan de cero, sino que emplean redes base preentrenadas y ajustan las últimas capas para adaptar el modelo a su problema, esta técnica empleada se conoce como *transfer learning*, la cual se explicará más detalladamente en el Capítulo 3.2.

Incluso los participantes que utilizan datasets adicionales para entrenar su modelo aplican técnicas de data augmentation, por lo cual, al tener tan pocos datos en el conjunto que se utilizará, podría ser interesante aplicarlo. Las transformaciones más utilizadas parece ser que son: giros horizontales y verticales, recortes, y variación en brillo, contraste y saturación.

Los participantes mostrados en la Tabla 2.1 corresponden a los equipos 5 diferentes que han quedado en las primeras posiciones, sin embargo, los tres primeros puestos en las clasificaciones finales de la tarea 3 han sido para *MetaOptima Technology Inc.* A continuación, se hará un breve estudio del trabajo que han realizado.

El equipo *MetaOptima Technology Inc.* ha empleado un dataset adicional al proporcionado por la competición (Figura 2.14). Han entrenado diferentes arquitecturas por separado, inicializándolas con los pesos obtenidos de las redes preentrenadas a utilizar. La pérdida y la precisión se han calculado promediando los resultados mediante validación cruzada. Se han aplicado diferentes transformaciones de data augmentation sobre las imágenes de entrenamiento, como volteos horizontales, rotaciones y color de jitter, todo ello de forma aleatoria dentro de un rango. Una vez realizado el entrenamiento, produce predicciones para los 7 diagnósticos de una manera relativamente imparcial, ya que, posteriormente, estas probabilidades las tratan como características para entrenar un clasificador de XGBoost (una librería software de código abierto). Su conjunto final se compondrá por los modelos de los que se haya obtenido mejor rendimiento, junto con un metamodelo. El metamodelo es una media de las predicciones realizadas por todos los modelos no seleccionados para el conjunto ponderado por el inverso de su pérdida logarítmica con validación cruzada. Se recalibra el conjunto y se cambia el anterior. Para las imágenes de test, para cada configuración de modelo, promedian las predicciones hechas por cada uno de los diferentes modelos entrenados.

Participante	Ranking	Dataset adicional	Red base	Data augmentation	Resultado
MetaOptima Technology Inc. [8]	#1	Si	ResNet, DPN, Densenet, Inception	Horizontal flips, rotations , jitter (brightness, contrast, saturation)	0.885
DAISYLab [9]	#4	Si	ResNet, Inception	Random crop, horizontal flip, vertical flip, jitter (brightness, contrast, saturation)	0.856
Medical Image Analysis Group, Sun Yat-sen University [10]	#5	No	SENet, PNASNet, ResNet	Random crop, horizontal flip, vertical flip, jitter (brightness, contrast, saturation)	0.845
Li [11]	#7	No	ResNet, DenseNet, Inception	—	0.815
Ask Sina [12]	#8	No	Xception, DenseNet	Horizontal flip, vertical flip	0.812

Tabla 2.1: Ranking de la tarea 3 del ISIC 2018 y los recursos utilizados por cada participante. Los resultados obtenidos corresponden a la precisión multiclase balanceada, que se corresponde con la media de la matriz de confusión obtenida de la precisión para cada clase.

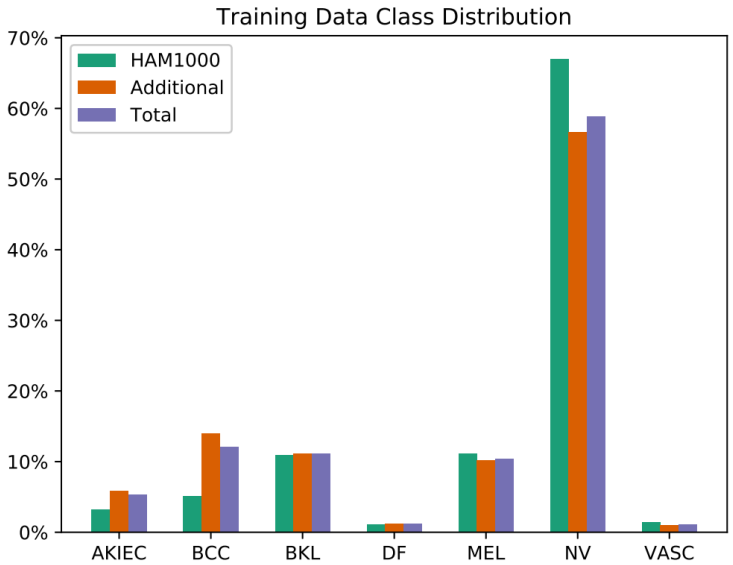


Figura 2.14: Proporción de tipos de diagnósticos en el dataset utilizado por MetaOptima Technology Inc.. Extraído de [8].

Capítulo 3

Diseño y desarrollo

3.1. Introducción

Una vez se ha analizado el estado del arte se hará una breve descripción de las redes pre entrenadas que se han utilizado en este trabajo para el problema de clasificación de imágenes, utilizando la técnica de *transfer learning*, la cual se detallará también a continuación. Por otro lado, se verá también una pequeña introducción a Pytorch, que será el paquete de Python a utilizar para la resolución del problema, así como el procedimiento para desarrollar el algoritmo de clasificación y el entorno en el que se ha trabajado.

3.2. Transfer learning

Uno de los grandes problemas en el ámbito del *Deep Learning* es la gran cantidad de tiempo que es necesario emplear en el entrenamiento de las redes, pudiendo llegar a durar incluso meses los casos más complejos y dependiendo del hardware disponible. Por estos motivos es por lo que se ha optado por utilizar la técnica de *transfer learning,* en vez de entrenar una red desde cero.

El *transfer learning* es una técnica de *Deep Learning* que permite aprovechar una red neuronal convolucional ya existente y que ya haya sido entrenada para alguna tarea similar a la que vayamos a abordar, y se modifica para entrenar con nuestros datos. Por lo tanto, en lugar de tener que entrenar una red desde cero, el trabajo se simplificará a inicializar la red con una preentrenada o al ajuste fino de la última capa de la red preentrenada para adaptarla a nuestro problema.

Esta técnica es muy útil cuando no tenemos suficientes datos como para entrenar un nuevo dominio con una red neuronal desde cero (como es este caso) y existe un gran conjunto de datos preexistentes que se pueden transferir al problema. Se pueden encontrar dos escenarios principales de *transfer learning* [13]:

- Ajuste de precisión de la CNN: en lugar de inicialización aleatoria, se inicializa la red con una red preentrenada. El resto del entrenamiento se realiza de manera estándar (Figura 3.1).

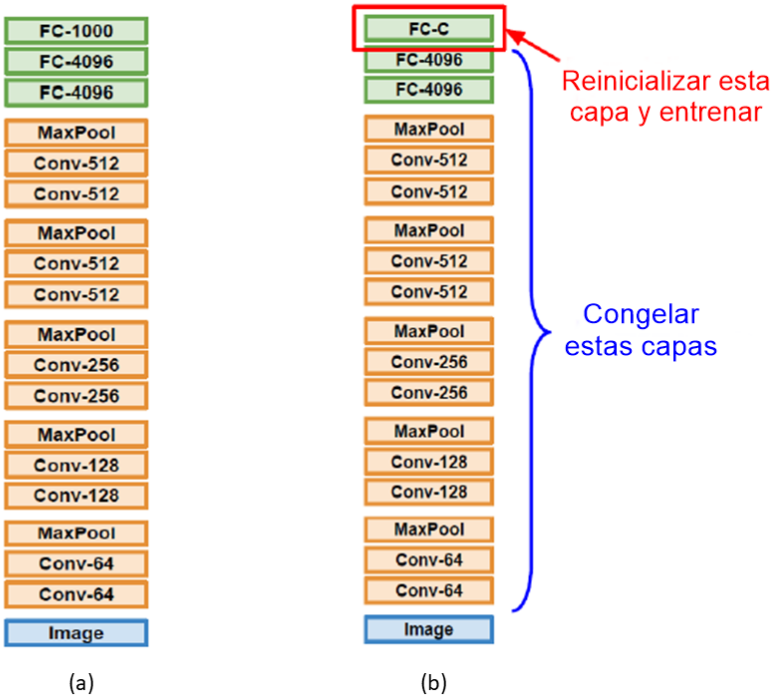


Figura 3.1: Ejemplos de *transfer learning* en CNNs. (a) Ajuste de precisión de la CNN mediante un modelo preentrenado. (b) Se congelan todos los pesos de la red preentrenada y la úlima se reinicia y se entrena.

- CNN como extractor de características fijas: en este caso se congelan los pesos de toda la red, excepto la de la capa final *fully connected*. Esta se reemplaza por una nueva con pesos aleatorios y solo se entrena esta capa (Figura 3.1).

3.3. Pytorch

Pytorch [13] es una plataforma de *Deep Learning* de código abierto que fue lanzada en 2017 por el grupo de inteligencia artificial de Facebook. Proporciona una interfaz muy sencilla para la creación de redes neuronales pese a trabajar de forma directa con tensores. Otra gran ventaja es que trabaja con grafos dinámicos en lugar de estáticos, por lo que las operaciones o computaciones se van realizando conforme se ejecuta el programa. Está diseñado para integrarse en Python, por lo que puede utilizar todas sus librerías y paquetes. Por otro lado, tiene soporte para ejecutar en GPU, utilizando internamente CUDA, una API que conecta la CPU con la GPU, lo cual permite acelerar procesos tradicionalmente muy lentos, como el entrenamiento de modelos. Se ha elegido este lenguaje de programación para el desarrollo del trabajo de clasificación de imágenes debido a todas sus funcionalidades y ventajas comentadas anteriormente.

3.3.1. Algoritmo base y desarrollo

El algoritmo implementado con Pytorch se basa en técnicas de *transfer learning* para el entrenamiento de los datos. En primer lugar se importan las librerías necesarias (*import torch, torchvision, numpy...*). Se declaran los parámetros e hiperparámetros, como el número de épocas, el tamaño del mini batch, la carpeta en la que se encuentran los datos, etc. Posteriormente se carga el conjunto de datos del que disponemos para entrenar el modelo. En este caso disponemos de 10000 imágenes, las cuales se dividirán en un conjunto de entrenamiento y otro de validación. Una vez cargados los datos, se aplican transformaciones redimensionándolos al tamaño requerido por la red neuronal que se vaya a utilizar. También se aplicarán transformaciones de data augmentation al conjunto de entrenamiento para estudiar el efecto que tiene en los resultados de clasificación. Posteriormente se configura para utilizar la GPU en caso de que esté disponible, si no se utilizará CPU.

Se carga una red neuronal preentrenada, la cual se entrenará aplicando alguna de las dos técnicas de *transfer learning* vistas en el Capítulo 3.2. El entrenamiento del modelo se definirá en un bucle que vaya entrenando época a época, calculando la pérdida y el accuracy en cada una de ellas, tanto para el conjunto de train como para el de validation, así como el tiempo de duración del entrenamiento, los cuales se guardarán para su uso posterior.

Una vez obtenidos los resultados del entrenamiento, se representarán gráficamente aquellos que nos interese estudiar.

3.4. Redes utilizadas

3.4.1. AlexNet

AlexNet [14] es una red neuronal convolucional diseñada por Alex Krizhevsky. La arquitectura de la red (Figura 3.2) se compone de 5 capas convolucionales (CNN) y 3 completamente conectadas (*fully connected*). En las primeras se detectarán patrones muy simples (líneas, esquinas, cambios de contraste, etc). Las intermedias, mediante combinaciones de las anteriores, serán capaces de detectar formas geométricas. Las capas más profundas podrán detectar patrones más complejos (objetos, caras, etc).

La capa inicial es la que recibe las imágenes de entrada, que se normalizan para que su dimensión sea 227x227x3, correspondiendo al alto H (*height*), ancho W (*width*) y profundidad D (*depth*) o canales que representan los valores RGB (*red, green, blue*). El objetivo de las cinco capas convolucionales es la extracción de características. Las dos primeras incluyen una etapa de *pooling* 3x3 con solapamiento, con el objetivo de reducir las dimensiones de la salida de la red para aumentar la profundidad de la siguiente capa de convolución. Con esto se podrán conseguir características más complejas, aunque se elimine parte de información espacial.

La función de las capas *fully connected* es la clasificación. En ellas las neuronas se encuentran totalmente conectadas unas con otras. Las dos primeras utilizan capas de *dropout* como método para reducir el *overfitting*, ya que evita que los pesos se adapten excesivamente a los datos de entrenamiento.

La capa final consiste en una *fully connected* cuyo tamaño de salida es 1000, que es el número de clases para el que fue preentrenada. Después de esta, hay una función de activación *softmax*, que se encarga de calcular la probabilidad que tiene cada dato de entrada de pertenecer a cada una de las clases. A partir de estas probabilidades, la capa de clasificación se encargará de definir cuál es la clase final a la que pertenece el dato de entrada.

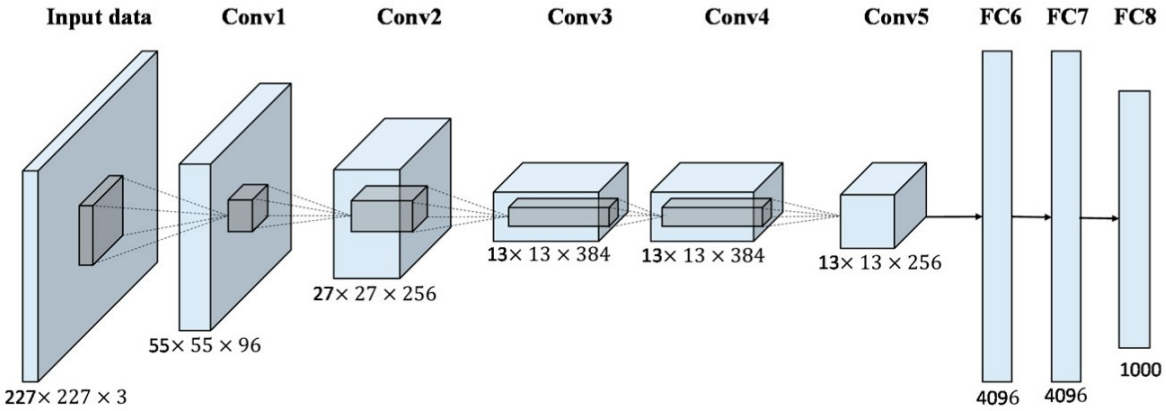


Figura 3.2: Arquitectura de AlexNet. Extraído de [15].

3.4.2. VGG

La arquitectura de la red VGG [16] fue presentada por Simonyan y Zisserman en su artículo de 2014. Realiza una mejora con respecto a AlexNet al reemplazar filtros grandes del tamaño del kernel por múltiples filtros de tamaño 3 x 3 del kernel, uno tras otro. Esta red se caracteriza por su simplicidad, ya que utiliza solo 3 × 3 capas convolucionales apiladas una encima de la otra en mayor profundidad. La reducción del tamaño del volumen se consigue mediante max pooling. Luego, dos capas *fully connected*, cada una con 4096 neuronas, seguidas de un clasificador *softmax*. VGG-16 significa que se utilizan 16 capas de pesos en la red.

En cuanto a la arquitectura (Figura 3.3), la entrada a la primera capa debe ser una imagen fija de 224x224x3 RGB. La imagen se pasa a través de una serie de capas convolucionales donde los filtros utilizados son de tamaño 3x3. En una de las configuraciones se utilizan un filtro de 1x1, que se puede ver como una transformación lineal de los canales de entrada. Todas estas capas convolucionales incluyen pooling al final, de forma que se reduzcan las dimensiones de la salida de cada una. Todas las capas ocultas cuentan con la función de activación ReLU.

También está formada por tres capas *fully connected,* las dos primeras con 4096 neuronas, y la tercera 1000, equivalente a las clases para las que fue preentrenada. Finalmente se encuentra la capa softmax, encargada de clasificar en las diferentes clases según las probabilidades establecidas.

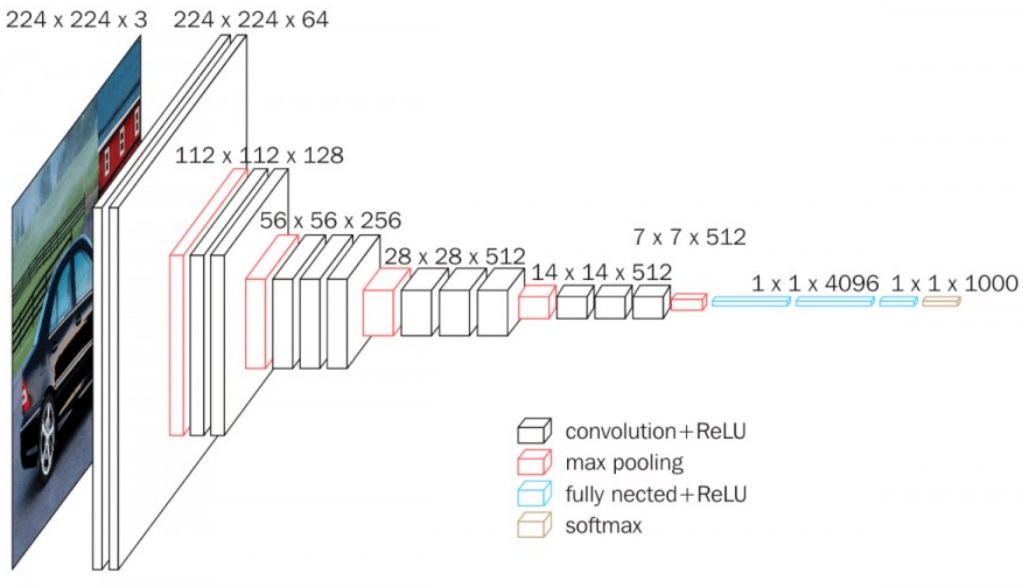


Figura 3.3: Arquitectura VGG-16. Extraído de [16]

3.4.3. ResNet

ResNet [2] proviene de *Residual Neural Network* (red neuronal residual), es una red neuronal artificial que utiliza conexiones de omisión o atajos, para saltar sobre algunas capas produciendo bloques residuales (Figura 3.4), típicamente saltos dobles o triples. Esto nos permite llevar información importante en la capa anterior a las siguientes capas. Una de las motivaciones para saltarse capas es evitar el problema del desvanecimiento de los gradientes, reutilizando las activaciones de una capa anterior hasta que la capa adyacente aprenda su peso. Además, aunque esta conexión parezca una adición al enfoque estándar de CNN, sorprendentemente, acelera el entrenamiento de la red.

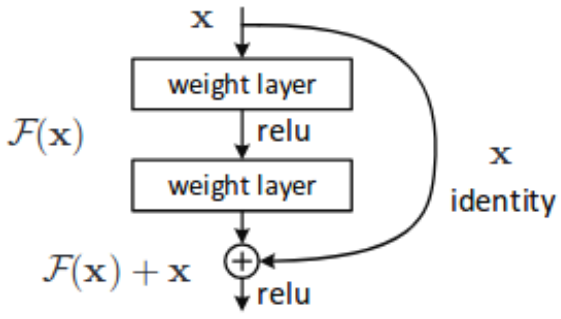


Figura 3.4: Bloque residual. Se añade una conexión de salto o acceso directo para agregar la entrada x a la salida después de algunas capas con pesos entrenables. Extraído de [2].

Respecto a su arquitectura, la imagen de entrada a la primera capa debe ser de 224x224x3 RGB, la cual se introducirá en la primera capa convolucional de la red (Conv-1). A continuación nos encontramos con cuatro capas convolucionales (Conv-2, Conv-3, Conv-4, Conv-5) obtenidas de los bloques residuales, lo cual podemos ver con más detalle en la Figura 3.5.

Posteriormente, se encuentra una capa *average pooling* que tiene el objetivo de reducir la salida de la capa anterior.

Por último, se introduce una capa *fully connected* (FC-6) para el número total de clases a aprender (en este caso está adaptado a 1000) y luego se aplicará una función de activación de *softmax* para generar las probabilidades de salida finales, y tomar una decisión respecto a su clasificación.

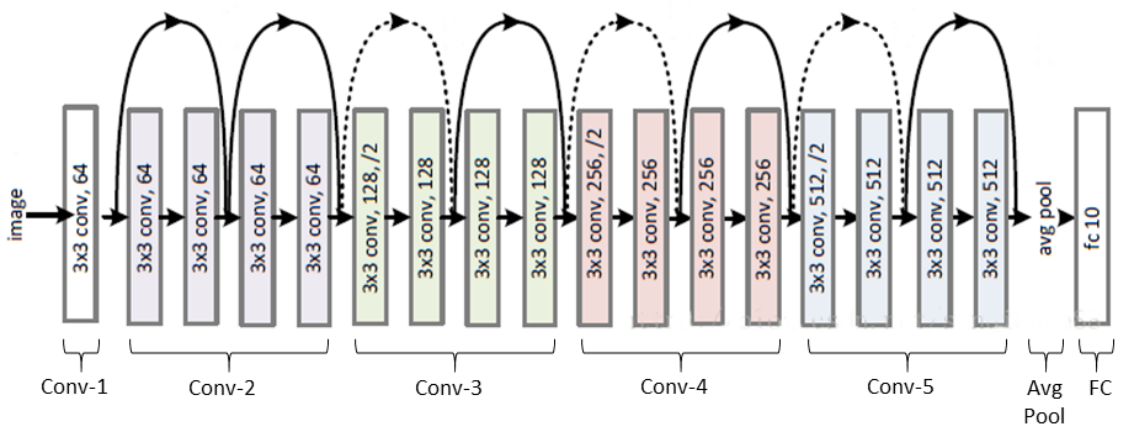


Figura 3.5: Arquitectura ResNet. Extraído de [2]

Capítulo 4

Evaluación

4.1. Introducción

En este capítulo se mostrarán los distintos experimentos realizados para establecer unos valores concretos óptimos de los hiperparámetros. Una vez determinados estos parámetros, se tomarán como configuración base para las diferentes pruebas con redes preentrenadas y distribuciones del conjunto de datos que se harán posteriormente. Para poder hacer uso de la GPU y acelerar los entrenamientos se ha hecho uso de una máquina virtual con GPU disponible para realizar este trabajo.

4.2. Marco de evaluación

4.2.1. Dataset

El conjunto de datos (*dataset*) utilizado ha sido el proporcionado por la competición ISIC 2018 [1], más concretamente el HAM10000 [7] (*"Human Against Machine with 10000 training images"*) dataset que cuenta con 10000 imágenes de manchas correspondientes a cáncer de piel, además de un archivo csv que incluye el *ground truth* correspondiente a cada imagen según el tipo de cáncer de piel que es (Figura 4.1). Todas las imágenes están en formato jpg, con un tamaño de 600x450 (ancho x alto) pixeles, el cual será ajustado para servir como entrada a cada red utilizada.

Por otra parte, al tener un único conjunto de 10000 imágenes etiquetadas con las clases correspondientes a clasificar, se dividirá en dos partes: conjunto de *train* y conjunto de *validation* (*val*). Esta división se ha llevado a cabo seleccionando el porcentaje correspondiente a cada conjunto requerido, en este caso se han hecho divisiones para los siguientes conjuntos:

- 80 % train (8000 imágenes) y 20 % validation (2000 imágenes)
- 70 % train (7000 imágenes) y 30 % validation (3000 imágenes)
- 55 % train (5500 imágenes) y 45 % validation (4500 imágenes)

De forma que se pueda estudiar los resultados obtenidos a partir de diferentes combinaciones

de los conjuntos de train y validation, que serán los que nos permitan establecer correctamente nuestro modelo de clasificación y ajustar los hiperparámetros.

MEL	NV	BCC	AKIEC	BKL	DF	VASC
1113	6690	514	327	1099	115	142

Tabla 4.2: Distribución de la cantidad de imágenes del dataset que hay en cada una de las 7 clases distintas.

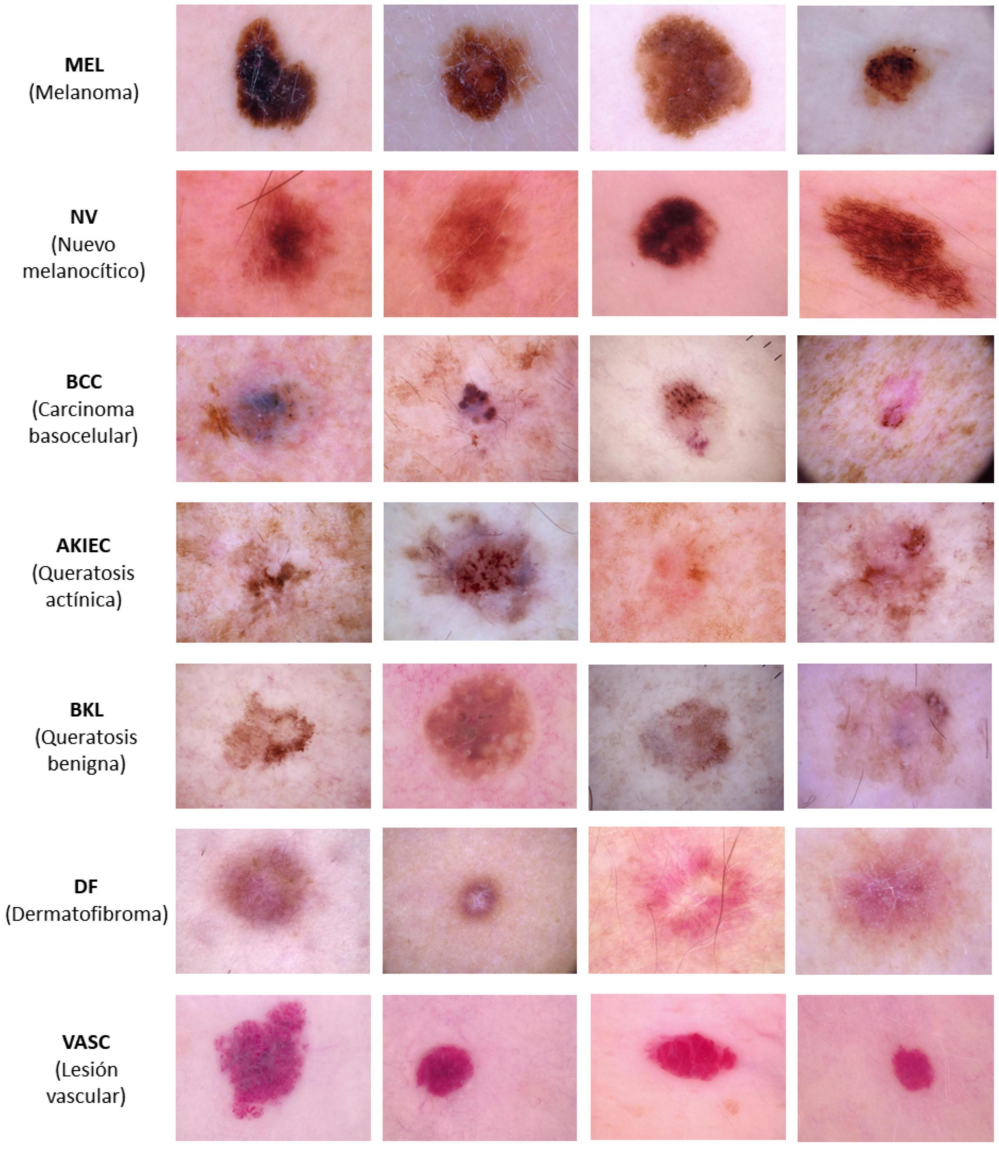


Figura 4.1: Ejemplos de imágenes del *dataset* de cada uno de los 7 tipos de cáncer de piel a clasificar en el problema.

4.2.2. Métricas

Para la evaluación de los algoritmos se tendrán en cuenta tanto la precisión (*accuracy*) como la función de pérdida (*loss*).

La precisión o accuracy es una métrica para evaluar modelos de clasificación. Mide la proporción entre las muestras bien clasificadas y el total de datos. Arroja una idea sobre el rendimiento global del sistema a evaluar. Tomará valores entre 0 y 1, siendo el 1 el 100 % de acierto, por lo que el objetivo será maximizarlo todo lo posible. Queda definido como:

$$Accuracy = \frac{\#NúmeroPrediccionesCorrectas}{\#NúmeroTotalPredicciones} \tag{4.1}$$

La función de pérdida o loss toma el par de entradas (salida, destino) y calcula un valor que estima lo lejos que está la salida del objetivo. Existen diferentes funciones de pérdida, por ejemplo el error cuadrático medio (MSE), visto en el Capítulo 2.1.9. Sirve como medida del error cometido, por lo tanto el objetivo será minimizar su valor. Para ello se hace uso de algoritmos de optimización, como por ejemplo el de descenso por gradiente.

El algoritmo que se empleará en este caso es el descenso de gradiente estocástico (*SGD*), que consiste en un método iterativo para optimizar una función objetivo. Se denomina estocástico porque utiliza muestras seleccionadas aleatoriamente (o mezcladas) para evaluar los gradientes. Esta es la forma en la que se van actualizando los pesos (*weight*) de la red:

$$weight = weight - LearningRate * gradient \tag{4.2}$$

4.3. Ajuste de hiperparámetros

Se procederá a hacer un estudio y evaluación de las épocas, el tamaño del mini-batch y el learning rate, de forma que se lleguen a establecer unos valores fijos considerados óptimos para el posterior entrenamiento con diferentes redes y conjuntos de datos. Todas las pruebas para el ajuste de hiperparámetros se realizarán sobre la red preentrenada AlexNet.

Para establecer el número de épocas, ya que no hay respuesta correcta a cuál es el adecuado y éste será diferente para cada conjunto de datos, habrá que hacer un estudio del conjunto de datos respecto a nuestro modelo de forma que se consigan las épocas adecuadas para asegurar la convergencia de la red.

Se han realizado diferentes pruebas para determinar un número fijo de épocas a utilizar en los diferentes entrenamientos y se ha comprobado que a partir de 15 o 20 épocas, tanto el accuracy como el loss convergen. En la figura 4.2 podemos ver un ejemplo de ejecución en el que se ve que a partir de 15-20 épocas el resultado es una recta constante. En la Tabla 4.3 también se puede apreciar claramente que el accuracy a partir de las 15 épocas no sufre ninguna variación, al igual que el loss, que a partir de la 20 se mantiene constante.

Por lo tanto el número de épocas establecido para el resto de experimentos se ha fijado en 30 de forma que nos aseguremos la convergencia, aunque como se ha visto anteriormente (Figura 4.2) a partir de 20 épocas los resultados empiezan a converger..

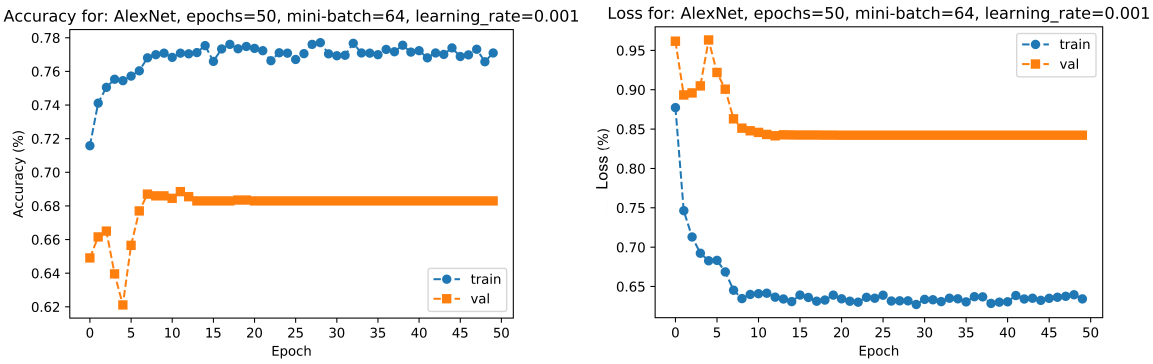


Figura 4.2: Resultado gráfico en 50 épocas para un entrenamiento utilizando AlexNet. (a) Accuracy train/val. (b) Loss train/val

Epoch	0	5	10	15	20	25	30	35	40	45	49
Accuracy	0.6490	0.6565	0.6854	0.6830	0.6830	0.6830	0.6830	0.6830	0.6830	0.6830	0.6830
Loss	0.9615	0.9218	0.8456	0.8424	0.8421	0.8421	0.8421	0.8421	0.8421	0.8421	0.8421

Tabla 4.3: Resultado obtenido de accuracy y loss para el conjunto de validación en diferentes épocas.

En cuanto a los tamaños del *mini batch*, Algunos tipos de hardware consiguen mejoras de tiempo con tamaños específicos de los lotes, especialmente utilizando GPU, como es nuestro caso, donde se suelen utilizar potencias de 2. Por lo tanto para los experimentos se harán diferentes pruebas empleando los tamaños de *mini batch* en los rangos de 16 a 128, es decir, 16, 32, 64 y 128.

Para los experimentos referidos al *learning rate*, se probará el valor determinado como estándar, que es el 0.001. Trataremos de ver también el efecto obtenido al utilizar un valor más pequeño de *learning rate* , 0.0001, el cual convergirá de forma más lenta; así como uno mayor que el estándar, que será 0.01, el cual deberá tener una convergencia más rápida.

4.3.1. Resultados

Para los experimentos se han aplicado las dos técnicas de *transfer learning* mencionadas anteriormente en cuanto a las capas entrenadas: en el primer caso el entrenamiento se hará sobre la red preentrenada entera, y en el otro caso se congelarán los pesos de todas las capas excepto de la última, que será la capa a entrenar y se corresponderá con la capa fully connected 8 (FC-8) de AlexNet (Figura 3.2).

AJUSTE DE HIPERPARÁMETROS	
Tamaño mini batch	16, 32, **64**, 128
Learning rate	0.01, **0.001**, 0.0001
Épocas	30
Técnica *transfer learning*	Modelo preentrenado completo, **última capa reentrenada**

Tabla 4.4: Valores escogidos para realizar las pruebas con el fin de ajustar los hiperparámetros. Resaltados en negrita los que se han seleccionado finalmente.

EXP.	CAPAS ENTRENADAS (AlexNet)	MINI BATCH	LEARNING RATE	MEJOR ACCURACY	
				TRAIN	VAL
1	Todas preentrenadas	16	0.01	0.7479	0.6690
2	Todas preentrenadas	16	0.001	0.7730	0.6775
3	Todas preentrenadas	16	0.0001	0.7693	0.6860
4	Todas preentrenadas	32	0.01	0.7601	0.6640
5	Todas preentrenadas	32	0.001	0.7791	0.6845
6	Todas preentrenadas	32	0.0001	0.7626	0.6845
7	Todas preentrenadas	64	0.01	0.7654	0.6795
8	Todas preentrenadas	64	0.001	0.7762	**0.6955**
9	Todas preentrenadas	64	0.0001	0.7532	0.6675
10	Todas preentrenadas	128	0.01	0.7761	0.6825
11	Todas preentrenadas	128	0.001	0.7715	0.6815
12	Todas preentrenadas	128	0.0001	0.7403	0.6525
13	Última (FC-8)	16	0.01	0.7460	0.6485
14	Última (FC-8)	16	0.001	0.7762	**0.6960**
15	Última (FC-8)	16	0.0001	0.7685	0.6840
16	Última (FC-8)	32	0.01	0.7507	0.6640
17	Última (FC-8)	32	0.001	0.7812	0.6845
18	Última (FC-8)	32	0.0001	0.7632	0.6770
19	Última (FC-8)	64	0.01	0.7636	0.6660
20	Última (FC-8)	64	0.001	0.7786	**0.6855**
21	Última (FC-8)	64	0.0001	0.7527	0.6605
22	Última (FC-8)	128	0.01	0.7732	0.6850
23	Última (FC-8)	128	0.001	0.7717	0.6845
24	Última (FC-8)	128	0.0001	0.7416	0.6540

Tabla 4.5: Tabla de experimentos para diferentes configuraciones de *transfer learning*, tamaños de *mini batch* y división del conjunto de datos. Los tres mejores resultados han sido resaltados en color negro, azul y rojo sucesivamente.

Observando la Tabla 4.5, vemos que el mejor resultado se ha dado en el experimento 14, con un valor de 0.6960 (negrita), el segundo mejor ha sido en el experimento 8, con un valor de 0.6955 (azul), y en tercer lugar el experimento 20 con un valor de 0.6855 (rojo).

Por lo tanto, podemos concluir que el learning rate que mejor resultado da es el de 0.001, ya que ha sido el utilizado para estas tres configuraciones.

Por otro lado, podemos ver en la tabla que, por lo general, se obtienen mejores resultados para el entrenamiento aplicado a la última capa. Esto puede ser debido a que al entrenar sobre una red con todas las capas preentrenadas puede que los pesos que tenían anteriormente no se estén ajustando de forma correcta a nuestros datos, de forma que el entrenamiento no esté siendo tan efectivo como al entrenar de cero la última capa.

En cuanto al tamaño del mini batch, nos quedaremos con el de 64 ya que es de los que mejores resultados obtiene.

4.4. Resultados sobre redes preentrenadas

Una vez fijados los parámetros mini-batch (64) y learning rate (0.001), así como la técnica de *transfer learning* que consiste en congelar los pesos de todas las capas y entrenar únicamente la última, se procede a hacer los experimentos sobre las distintas redes preentrenadas (AlexNet, VGG y ResNet) y a aplicar data augmentation. Como la última capa de la arquitectura de estas tres redes es la *fully connected*, esta será la que se entrene congelando el resto. Además, se ha adaptado las salidas de forma que clasifique en 7 clases distintas, en lugar de 1000, que es para lo que habían sido originalmente entrenadas. Se ha tomado la decisión de emplear las transformaciones de data augmentation que consisten en giro horizontal (*horizontal flip*), giro vertical (*vertical flip*) y la aplicación de brillo, contraste y saturación (*jitter*), ya que estas fueron las que más destacaron en el estudio hecho previamente sobre los resultados de la competición (Capítulo 2.3.3).

Los siguientes experimentos están realizados sobre 20 épocas, ya que se comprobó anteriormente que a partir de este punto ya habían convergido.

4.4.1. Resultados sobre AlexNet

Una vez realizados distintos experimentos utilizando AlexNet como red base, podemos ver en la Tabla 4.6 que, empleando las mismas transformaciones de data augmentation, en todos los casos se dan mejoras en la precisión (accuracy) de los datos de validación al aumentar la proporción de datos de validación con respecto a los de entrenamiento. Además, al aplicar las transformaciones de flip vertical y horizontal también vemos algo de mejora respecto a los experimentos en los que no se ha añadido data augmentation. Sin embargo, el hecho de emplear color jitter, que añade cambios en el brillo, contraste y saturación de las imágenes, da resultados bastante similares, e incluso peores, respecto a las primeras pruebas en las que no se había añadido ninguna transformación.

Por lo tanto, podemos concluir que los mejores resultados utilizando la red AlexNet se dan para una división del conjunto de datos en 55 % de entrenamiento y 45 % de validación, y sin aplicar transformaciones de data augmentation, o aplicando únicamente los volteos horizontales y verticales.

EXP.	RED BASE	DATA AUGMENTATION			PROPORCIÓN TRAIN/VAL	MEJOR ACCURACY	
		FLIP H	FLIP V	JITTER		TRAIN	VAL
1	AlexNet	NO	NO	NO	80 % / 20 %	0.7730	0.6760
2	AlexNet	NO	NO	NO	70 % / 30 %	0.7780	0.7022
3	AlexNet	NO	NO	NO	55 % / 45 %	0.7736	**0.7411**
4	AlexNet	SI	NO	NO	80 % / 20 %	0.7760	0.6845
5	AlexNet	SI	NO	NO	70 % / 30 %	0.7742	0.7015
6	AlexNet	SI	NO	NO	55 % / 45 %	0.7736	**0.7400**
7	AlexNet	SI	SI	NO	80 % / 20 %	0.7718	0.6815
8	AlexNet	SI	SI	NO	70 % / 30 %	0.7740	0.7128
9	AlexNet	SI	SI	NO	55 % / 45 %	0.7716	**0.7362**
10	AlexNet	SI	SI	SI	80 % / 20 %	0.7451	0.6730
11	AlexNet	SI	SI	SI	70 % / 30 %	0.7437	0.7111
12	AlexNet	SI	SI	SI	55 % / 45 %	0.7435	0.7311

Tabla 4.6: Tabla experimentos realizados con AlexNet, aplicando data augmentation y distintas subdivisiones del conjunto de datos. Mejores resultados destacados en negro, azul y rojo respectivamente.

4.4.2. Resultados sobre VGG

Observando la Tabla 4.7 de las pruebas llevadas a cabo con la red neuronal VGG-16, llegamos a conclusiones similares que con la red AlexNet respecto a las proporciones del conjunto de datos, ya que se nota la mejoría de los resultados del *accuracy* de validación al utilizar 55 % de entrenamiento y 45 % de validación, respecto a las otras dos proporciones con las que se ha experimentado. En cuanto a las transformaciones de *data augmentation*, se puede ver un ligero aumento de la precisión al emplear volteos horizontales y verticales respecto a las pruebas en las que no se realiza ninguna transformación, sin embargo, en este caso también vuelve a empeorar el resultado al aplicar *color jitter*.

EXP.	RED BASE	DATA AUGMENTATION			PROPORCIÓN TRAIN/VAL	MEJOR ACCURACY	
		FLIP H	FLIP V	JITTER		TRAIN	VAL
1	VGG-16	NO	NO	NO	80 % / 20 %	0.7356	0.6490
2	VGG-16	NO	NO	NO	70 % / 30 %	0.7347	0.6735
3	VGG-16	NO	NO	NO	55 % / 45 %	0.7273	**0.6934**
4	VGG-16	SI	NO	NO	80 % / 20 %	0.7336	0.6550
5	VGG-16	SI	NO	NO	70 % / 30 %	0.7325	0.6755
6	VGG-16	SI	NO	NO	55 % / 45 %	0.7260	**0.6936**
7	VGG-16	SI	SI	NO	80 % / 20 %	0.7339	0.6525
8	VGG-16	SI	SI	NO	70 % / 30 %	0.7341	0.6725
9	VGG-16	SI	SI	NO	55 % / 45 %	0.7268	**0.6938**
10	VGG-16	SI	SI	SI	80 % / 20 %	0.7167	0.6365
11	VGG-16	SI	SI	SI	70 % / 30 %	0.7190	0.6609
12	VGG-16	SI	SI	SI	55 % / 45 %	0.7144	0.6898

Tabla 4.7: Tabla experimentos realizados con VGG-16, aplicando data augmentation y distintas subdivisiones del conjunto de datos. Mejores resultados destacados en negro, azul y rojo respectivamente.

4.4.3. Resultados sobre ResNet

Por último, se han hecho otros 12 experimentos sobre la red preentrenada ResNet-18, cuyos resultados se muestran en la Tabla 4.8. Una vez más, vemos que los mejores *accuracies* se dan para los conjuntos formados por un 55 % de datos de entrenamiento y 45 % de validación. El mejor resultado tiene lugar para el experimento en el que se aplican transformaciones de *flip* horizontal y vertical, el cal no dista demasiado del segundo mejor resultado, en el que únicamente se aplica *flip* horizontal como *data augmentation*, por lo tanto podemos concluir que, para el caso de ResNet-18, aplicar transformaciones de volteos sí produce una mejoría en los resultados. Por otro lado, al emplear el *color jitter* volvemos a obtener peores resultados en la precisión de los datos de validación.

EXP.	RED BASE	DATA AUGMENTATION			PROPORCIÓN TRAIN/VAL	MEJOR ACCURACY	
		FLIP H	FLIP V	JITTER		TRAIN	VAL
1	ResNet-18	NO	NO	NO	80 % / 20 %	0.7701	0.6675
2	ResNet-18	NO	NO	NO	70 % / 30 %	0.7750	0.6945
3	ResNet-18	NO	NO	NO	55 % / 45 %	0.7669	**0.7142**
4	ResNet-18	SI	NO	NO	80 % / 20 %	0.7741	0.6665
5	ResNet-18	SI	NO	NO	70 % / 30 %	0.7709	0.6945
6	ResNet-18	SI	NO	NO	55 % / 45 %	0.7663	**0.7189**
7	ResNet-18	SI	SI	NO	80 % / 20 %	0.7732	0.6735
8	ResNet-18	SI	SI	NO	70 % / 30 %	0.7716	0.6998
9	ResNet-18	SI	SI	NO	55 % / 45 %	0.7729	**0.7200**
10	ResNet-18	SI	SI	SI	80 % / 20 %	0.7425	0.6595
11	ResNet-18	SI	SI	SI	70 % / 30 %	0.7469	0.6792
12	ResNet-18	SI	SI	SI	55 % / 45 %	0.7398	0.7112

Tabla 4.8: Tabla experimentos realizados con ResNet-18, aplicando data augmentation y distintas subdivisiones del conjunto de datos. Mejores resultados destacados en negro, azul y rojo respectivamente.

4.5. Comparativa

Comparando los resultados obtenidos en las Tablas 4.6, 4.7 y 4.8 de los experimentos realizados sobre las distintas redes preentrenadas escogidas (AlexNet, VGG-16 y ResNet-18), podemos concluir varias cosas:

- El modelo funciona mejor para la proporción 55 % train y 45 % validation. Esto puede ser debido a que se está produciendo *overfitting*, es decir, los datos de entrenamiento se están aprendiendo de "memoria" los las imágenes con las que entrena y, por lo tanto, al aplicar su aprendizaje en el conjunto de validación obtiene peores resultados ya que las imágenes son diferentes a las que se ha aprendido.

- Por lo general, las transformaciones aplicadas de data augmentation no están dando buenos resultados. Se ve claramente que al aplicar *color jitter*, es decir, brillo, contraste y saturación, los resultados empeoran.

- Los mejores resultados se han obtenido para la red AlexNet. Como era de esperar, también se han logrado mejores resultados utilizando ResNet que con VGG.

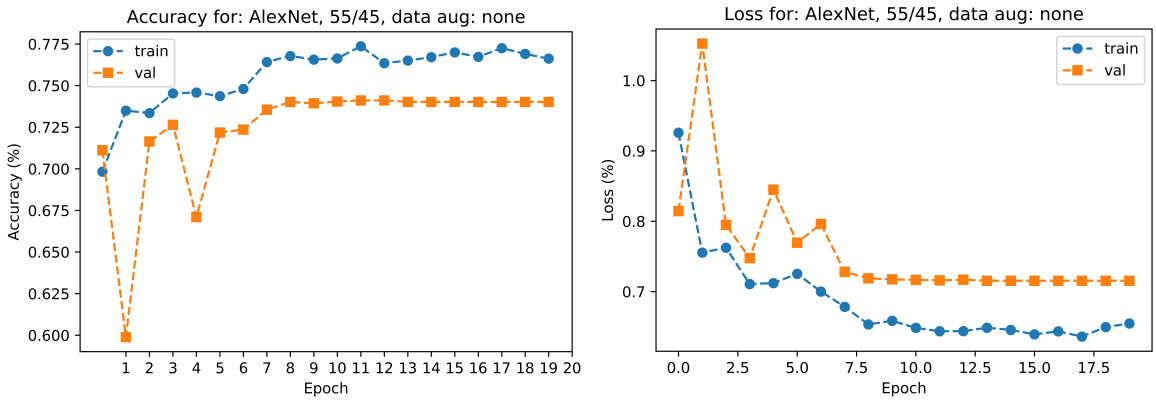


Figura 4.3: Representación gráfica del *accuracy* y *loss* obtenido para *train* y *val* del mejor resultado (0.7411), es decir, para la red AlexNet, sin aplicar data augmentation y con división del conjunto en 55 % train y 45 % val.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

A lo largo de este proyecto se han llevado a cabo una serie de tareas que han contribuido a cumplir el objetivo principal de este Trabajo de Fin de Grado (Capítulo 1.2).

El estudio del estado del arte ha sido un punto clave, ya que un estudio básico sobre redes convolucionales y los hiperparámetros a entrenar han resultado muy útiles para la comprensión del trabajo. Además, el análisis de otras propuestas de los participantes de la competición en la que nos hemos basado, ha sido de gran ayuda para aportar ideas a la fase de diseño y desarrollo.

Gracias a la técnica de *transfer learning* se ha conseguido optimizar el entrenamiento del modelo, y con ello poder estudiar el comportamiento de las tres diferentes redes preentrenadas seleccionadas, AlexNet, VGG y ResNet.

A pesar de no contar con las mismas condiciones que los participantes clasificados en la competición, se han obtenido resultados con un acierto cerca del 75 %. Solo disponíamos del *dataset* de entrenamiento etiquetado con su ground truth, el cual hemos dividido de forma que dispusiésemos de dos conjuntos, uno de *train* y otro de *validation*, por lo que el conjunto de entrenamiento ha quedado todavía más reducido. Además, muchos de los participantes han utilizado datasets adicionales al proporcionado por el ISIC, el cual era bastante pequeño para un problema de clasificación de este tipo.

Los porcentajes obtenidos, tanto en este trabajo como en la competición en general, me parecen insuficientes tratándose de un problema tan delicado como es la determinación de un tipo de cáncer concreto, en el cual sería conveniente tener la máxima precisión posible para poder determinar una cura específica. Sin embargo, gracias al *Deep Learning*, con esta clase de herramientas se está ofreciendo un sistema muy potente y útil a la hora de determinar el diagnóstico de este tipo de enfermedades, lo cual servirá de gran ayuda a los médicos especialistas a la hora de tomar las decisiones finales, proporcionándoles más precisión y rapidez en los diagnósticos.

5.2. Trabajo futuro

A la vista de los resultados que se han obtenido en este trabajo se propone trabajar en las siguientes mejoras para lograr una mayor precisión en los resultados:

- En este proyecto se han utilizado redes convolucionales preentrenadas bastante básicas con el fin de entender su arquitectura y poder entrenar de forma sencilla el conjunto de datos dado. Sin embargo, existen otras redes con estructuras mucho más complejas o incluso las mismas redes utilizadas, pero con muchas más capas. Para un posible trabajo futuro se propone utilizar por ejemplo ResNet-152, Vgg-19, u otras redes diferentes como Densenet o Inception.

- Otra posible mejora podría ser probar a introducir más capas al final, antes de la clasificación softmax, de forma que vayan reduciendo el número de neuronas en cada capa de forma gradual en lugar de pasar, por ejemplo, de 4096 a 7, como ocurre empleando AlexNet, en tan solo la última capa. Además de ir añadiendo *dropout* en cada capa para poder evitar el *overfitting*.

- Sería interesante añadir más parámetros de *data augmentation*, ya que la cantidad de datos de la que se dispone es bastante pequeña, y con esto lograr un entrenamiento más efectivo. Estos nuevos parámetros de data augmentation podrían ser rotaciones y recortes aleatorios.

- Al segmentar el conjunto de datos en *train* y *validation* en el porcentaje definido para cada uno, sería conveniente tratar de mantener la misma proporción en ambos para las 7 clases a clasificar. Además, sería interesante que, para cada entrenamiento, esta segmentación fuera aleatoria de forma que no entrenara siempre sobre los mismos datos.

- Otro posible trabajo futuro de cara a ver cómo se están comportando los datos, podría ser calcular el *accuracy* por separado para cada una de las clases, de forma que se pueda estudiar cuál falla y cuál funciona correctamente.

- Por último, una forma interesante de tratar de evitar el desbalanceo de clases y conseguir un entrenamiento más eficiente para los 7 tipos de cáncer de piel a clasificar, sería aplicar sobremuestreo a aquellas clases de las que tengamos menos datos con respecto a las otras.

Bibliografía

[1] N. C. F. Codella, D. Gutman, M. E. Celebi, B. Helba, M. A. Marchetti, S. W. Dusza, A. K. K. Liopyris, N. Mishra, H. Kittler, and A. Halpern:, "Skin lesion analysis toward melanoma detection: A challenge at the 2017 international symposium on biomedical imaging (isbi), hosted by the international skin imaging collaboration (isic)," 2017. 2, 16, 18, 27

[2] Towards Data Science, "An overview of resnet and its variants." https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035, 2017. 6, 25, 26

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," in *Journal of Machine Learning Research*, pp. 1929–1958, 2014. 11

[4] Ikuper, "Bringing parallelism to the web with river trail." http://intellabs.github.io/RiverTrail/tutorial/, 2016. 13

[5] Clarifai company, "Convolutional neural networks." https://www.clarifai.com/technology, 2016. 13

[6] R. B. Zadeh and B. Ramsundar, "Fully connected deep networks," in *TensorFlow for Deep Learning*, vol. Chapter 4, 2019. 15

[7] T. P. and R. C. amp; Kittler H. in *The HAM10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. Sci. Data 5, 180161 doi.10.1038/sdata.2018.161*, 2018. 16, 27

[8] A. Nozdryn-Plotnicki, J. Yap, and W. Yolland, "Ensembling convolutional neural networks for skin cancer classification," 2018. 19

[9] N. Gessert, T. Sentker, F. Madesta, R. Schmitz, H. Kniep, I. Baltruschat, R. Werner, and A. Schlaefer, "Skin lesion diagnosis using ensembles, unscaled multi-crop evaluation and loss weighting," in *arXiv:1808.01694*, 2018. 19

[10] J. Zhuang, W. Li, S. Manivannan, R. Wang, J. Zhang, J. Liu, J. Pan, G. Jiang, Z. M. Group, S. of Data, C. Scienc, S. Y. sen University, C. C. S. of Science, Engineering, U. of Dundee. Department of computer science, and U. of Jaffna, "Skin lesion analysis towards melanoma detection using deep neural network ensemble," 2018. 19

[11] K. M. Li and E. C. Li, "Skin lesion analysis towards melanoma detection via end-to-end deep learning of convolutional neural networks," in *arXiv:1807.08332*, 2018. 19

[12] M. K. Amro, B. Singh, and A. Rizvi, "Skin lesion classification and segmentation for imbalanced classes using deep learning," 2018. 19

[13] Grupo de Inteligencia Artificial de Facebook, "Documentacion de pytorch." https://pytorch.org/, 2017. 21, 22

[14] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012. 23

[15] X. Han, Y. Zhong, L. Cao, and L. Zhang, "Pre-trained alexnet architecture with pyramid pooling and supervision for high spatial resolution remote sensing image scene classification," in *Remote Sensing*, 2017. 24

[16] Neurohive, "Vgg16: Convolutional network for classification and detection." https://neurohive.io/en/popular-networks/vgg16/, 2018. 24, 25