



Escuela Técnica Superior de
Ingeniería Informática

TRABAJO FIN DE GRADO

Título del trabajo

Realizado por
Jaime Rodríguez Albuín

Para la obtención del título de
Grado en Ingeniería Informática - Tecnologías Informáticas

Dirigido por
Isabel De Los Ángeles Nepomuceno Chamorro

Realizado en el departamento de
Lenguajes y Sistemas Informáticos

Convocatoria de Junio, curso 2020/21

Agradecimientos

Quiero agradecer sobre todo a mi hermano y a mi madre por estar apoyándome desde el minuto cero incondicionalmente, siempre dispuestos a llevarme y a traerme de Sevilla y siendo sin duda piezas clave en esta etapa de mi vida.

También quiero agradecer a mis tíos Victoria y Manolo por ayudarnos en momentos tan complicados y haber ejercido siempre de familia.

Agradezco a mi tutora Isabel el hecho de aceptarme como alumno incluso llegando un poco tarde. También a todos los profesores que he tenido a lo largo de la carrera, puesto que sé con certeza que la formación que he recibido a lo largo de estos cuatro años me servirá, tanto en los conocimientos como en la forma de afrontar los problemas.

No me voy a olvidar de los maravillosos compañeros que he conocido a lo largo de este viaje, y sobre todo agradezco las amistades con Arturo, Ventura, Emilio, Jesús, Juan, Juan Carlos, Pedro García, Pedro Escobar, Curro, Antonio, Manuel, Javi y Josemi.

Resumen

Trabajo de investigación sobre redes neuronales convolucionales y la detección de paciencias con neumonía, posiblemente provocadas por Covid.

Palabras clave: Redes neuronales, Convolución, Pandas, Keras, Tensorflow, Covid, Kaggle, Google Colab, Inteligencia Artificial, Deep learning, Neumonía, Salud.

Abstract

Research project about convolutional neural networks and pneumonia detection on patients, possibly provoked by Covid

Keywords: Neural networks, Convolution, Pandas, Keras, Tensorflow, Covid, Kaggle, Google Colab, Artificial Intelligence, Deep learning, Pneumonia, Health.

Índice general

1. Introducción	1
2. Planificación temporal y costes	3
3. Estudio Previo	4
3.1. Introducción	4
3.2. Aprendizaje supervisado y no supervisado	4
3.2.1. Aprendizaje supervisado	4
3.2.2. Aprendizaje no supervisado	6
3.3. Redes neuronales	8
3.3.1. Inspiración biológica	8
3.3.2. El perceptrón	10
3.3.3. Red Neuronal Tradicional	15
3.3.4. Algoritmo de propagación hacia atrás:	17
3.3.5. Funciones de pérdida probabilísticas	19
3.3.6. Funciones de pérdida regresivas	20
3.3.7. Optimizadores	21
3.3.8. Regularización	22
3.4. Deep learning	22
3.4.1. Redes convolucionales	24
3.5. Estado del arte	30
3.5.1. Transfer Learning	30
3.5.2. VGG	31
3.5.3. InceptionV3	32
3.5.4. ResNet	34
4. Tecnologías	36
4.1. Introducción	36
4.2. Hardware	36
4.2.1. Equipo personal	36
4.2.2. Google Colab	37
4.3. Software	37
4.3.1. Python3	37
4.3.2. NumPy	38
4.3.3. Tensorflow	38
4.3.4. Keras	38
4.3.5. Pillow	38
4.3.6. OS	39
4.3.7. Flask	39
5. Implementación y pruebas	40
5.1. Primera parte: preprocessado del dataset	40
5.2. Segunda parte: Implementación de las redes	40

5.2.1.	Primera red	40
5.2.2.	Segunda red. Con regularización y data augmentation	44
5.2.3.	Tercera red: VGG16 + Transfer Learning	48
5.2.4.	Pruebas	50
5.2.5.	Cuarta red: InceptionV3 + Transfer Learning	52
5.2.6.	Entrenamiento	54
5.2.7.	Pruebas	55
5.2.8.	Quinta red: ResNet + Transfer Learning	56
5.3.	Tercera parte: Aplicación Web	59
6.	Manual de usuario	63
6.1.	Experimentación	63
6.2.	Página web	63
7.	Pruebas	64
7.1.	Introducción	64
7.2.	Conclusiones	64
8.	Conclusiones	65
9.	Bibliografía	66

Índice de figuras

3.1.	Resultado de aplicar la detección de objetos	4
3.2.	Ejemplo de segmentación de imágenes	5
3.3.	Imagen que demuestra el flujo de la generación de subtítulos	5
3.4.	Resultado de la aplicación de clustering sobre un conjunto de individuos	6
3.5.	Etapas en la reducción de dimensiones.	7
3.6.	Estructura de una neurona biológica.	8
3.7.	Diagrama que muestra el disparo de una neurona	9
3.8.	Comparativa de la estructura de la neurona biológica con el perceptrón	10
3.9.	Gráfica de la función escalón de Heaviside.	11
3.10.	Gráfica de la función ReLU.	12
3.11.	Gráfica de la función sigmoide.	12
3.12.	Gráfica de la función softmax.	13
3.13.	Perceptrón multicapa.	15
3.14.	Detección de una pelota por una red neuronal.	17
3.15.	Actualización de un peso de la última capa.	18
3.16.	Actualización de un peso de las capas ocultas.	19
3.17.	Representación del mecanismo de dropout en las neuronas de la red.	22
3.18.	Corteza visual.	24
3.19.	Estructura de una red convolucional.	25
3.20.	Matriz de valores de intensidad de los píxeles de una imagen con un canal (escala de grises).	26
3.21.	Operación de convolución.	27
3.22.	Distintos kernels con distintos mapas de características.	28
3.23.	Operación de max pooling.	29
3.24.	Adaptación de una red entrenada para reutilizarla.	30
3.25.	Estructura de VGG16.	31
3.26.	Resultados de ILSVRC 2015.	32
3.27.	Estructura de Inceptionv3.	32
3.28.	Estructura de un módulo Inception.	33
3.29.	Estructura de las redes ResNet.	34
4.1.	Equipo utilizado para la elaboración del proyecto.	36
4.2.	Logotipo de Google Colab.	37
5.1.	Visualización de la primera red.	40
5.2.	Sumario de la estructura de la primera red.	41
5.3.	Valores de precisión durante el entrenamiento.	42
5.4.	Valores de pérdida durante el entrenamiento.	42
5.5.	Valores de precisión y pérdida durante las pruebas.	43
5.6.	Ejemplo de data augmentation con radiografías.	44
5.7.	Visualización de la segunda red.	44
5.8.	Sumario de la estructura de la segunda red.	45
5.9.	Valores de precisión durante el entrenamiento.	46

5.10. Valores de pérdida durante el entrenamiento.	46
5.11. Valores de precisión y pérdida durante las pruebas.	47
5.12. Sumario de la estructura de la tercera red.	48
5.13. Red neuronal acoplada a VGG16.	49
5.14. Valores de precisión durante el entrenamiento de VGG16.	49
5.15. Valores de pérdida durante el entrenamiento de VGG16.	50
5.16. Valores de precisión y pérdida durante las pruebas de VGG16.	50
5.17. Sumario de la estructura de la cuarta red.	52
5.18. Red neuronal acoplada a InceptionV3.	52
5.19. Valores de precisión durante el entrenamiento de InceptionV3.	54
5.20. Valores de pérdida durante el entrenamiento de InceptionV3.	54
5.21. Valores de precisión y pérdida durante las pruebas de InceptionV3.	55
5.22. Sumario de la estructura de la quinta red.	56
5.23. Red neuronal acoplada a ResNet.	57
5.24. Valores de precisión durante el entrenamiento de ResNet.	57
5.25. Valores de pérdida durante el entrenamiento de ResNet.	58
5.26. Valores de precisión y pérdida durante las pruebas de ResNet.	58
5.27. Diagrama de flujo de la aplicación web.	59
5.28. Primera vista de la aplicación web.	59
5.29. Selección con las cinco redes disponibles.	60
5.30. Carga de una imagen en la aplicación web.	60
5.31. Formulario con la red seleccionada y la imagen cargada.	61
5.32. Resultado de una predicción con un paciente sano.	61
5.33. Resultado de una predicción con un paciente enfermo.	62

Índice de extractos de código

1. Introducción

Motivación

La principal fuente de motivación ha sido la pandemia. Mi principal objetivo fué encontrar una manera de predecir un diagnóstico sobre las personas que pudieran padecer de COVID. Entre síntomas hay uno que siempre ha sido uno de los principales indicios de dicha enfermedad, y es la neumonía provocada por el virus. Esta neumonía es la que provoca la tos en los pacientes, y el hecho de encontrar esta propuesta en la página web de los TFG de la ETSII fué un flechazo.

Además, el hecho de ver las redes neuronales tan por encima en IA y AIA me dejó con las ganas de investigar. No conocía las redes convolucionales, y me han parecido un descubrimiento espectacular.

Contexto

Hoy en día nos encontramos en uno de los picos más altos en cuanto a la inteligencia artificial. También se ha juntado con una pandemia sin precedentes, como está siendo la provocada por el Covid-19. Viendo la necesidad que había de encontrar maneras de detectar el covid busqué un trabajo relacionado con ello. Había otras maneras de detectar el Covid a parte de las radiografías, como puede ser la tos, pero yo desarrollaré modelos basados en radiografías.

Objetivos

A lo largo de este proyecto intentaré alcanzar varios objetivos:

- Encontrar un modelo lo suficientemente fiable como para acertar la mayoría de predicciones.
- Desarrollar una aplicación web sencilla, de manera que cualquiera podría subir su imagen y realizar dicha predicción.
- Mostrar una manera de utilizar la plataforma Google Colab, conectándola al servicio de almacenamiento de datos en la nube de Google: "Google Drive".

Glosario

Batch División del conjunto de datos que se proporcionará en cada iteración.

Covid-19 El "Sars Cov 2" es una enfermedad vírica que durante 2020 y 2021 ha provocado una pandemia.

Epoch Iteración sobre el conjunto de datos entero.

Google colaborate Plataforma en la nube de Google proporcionada para programar notebooks de python online, sin tener que utilizar hardware en local.

GPU "Unidad de procesamiento gráfico" en español. También llamadas tarjetas gráficas.

Imagenet Es una base de datos de gran escala utilizada para la investigación en software de reconocimiento de objetos. Más de 14 millones de imágenes han sido insertadas a mano para indicar claramente qué objetos aparecen en cada imagen, perteneciendo a más de 22000 categorías.

Tensor En matemáticas y en física, un tensor es cierta clase de entidad algebraica de varios componentes que generaliza los conceptos de escalar, vector y matriz de una manera que sea independiente de cualquier sistema de coordenadas elegido.

w Vector de pesos sinápticos asociados a una capa.

X Conjunto de características de un ejemplo x.

y Clase esperada de un ejemplo.

2. Planificación temporal y costes

3. Estudio Previo

3.1. Introducción

En este capítulo introduciremos las redes convolucionales desde el perceptrón hasta el estado del arte y el dataset que trabajaremos.

3.2. Aprendizaje supervisado y no supervisado

Primero deberemos preguntarnos, ¿qué significa el "aprendizaje" en máquinas? La respuesta depende de qué tipo de aprendizaje buscamos:

3.2.1. Aprendizaje supervisado

Este acercamiento al aprendizaje automático busca obtener una función a partir de un conjunto de datos, es decir, relaciones que asocian entradas con salidas. Este acercamiento al aprendizaje automático incluso llega a proporcionar las formas de clasificar más comunes. Dependiendo de la salida diferenciaremos entre modelos de clasificación, si la salida es un valor categórico (como una enumeración, o un conjunto finito de clases), y modelos de regresión, si la salida es un valor en un espacio continuo.

Las aplicaciones son muy numerosas y siguen estando a la orden del día, tales como reconocimiento de voz, clasificación de imágenes (como en este trabajo) o traducción de idiomas. También existen otras aplicaciones más exóticas que merecen la pena mencionar:

- Detección de objetos: dada una imagen dibujar una caja que delimita los objetos a detectar. Este problema puede ser presentado como un problema de clasificación (obteniendo muchos candidatos a los que dibujar la caja, elegir si se dibujan o no) o como una unión entre un problema de clasificación y uno de regresión, donde las esquinas de dicha caja se predigan mediante vectores de regresión.

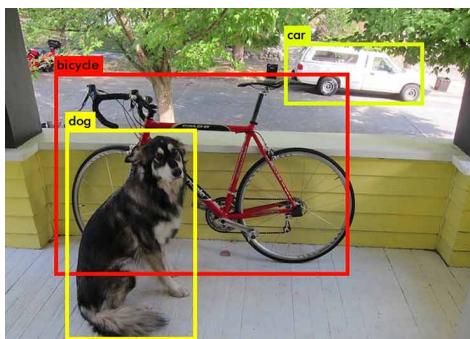


Figura 3.1: Resultado de aplicar la detección de objetos

- Segmentación de imágenes: dada una imagen, dibujar una máscara píxel a píxel sobre un objeto en específico.



Figura 3.2: Ejemplo de segmentación de imágenes

- Generación de secuencias: dada una imagen, predecir un subtítulo que la describa. Este problema a veces puede ser reformulado como una secuencia de

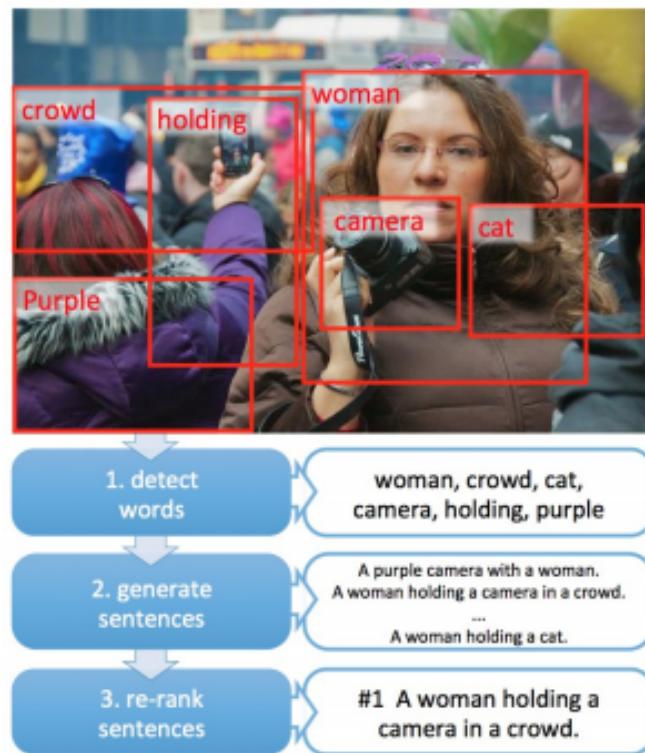


Figura 3.3: Imagen que demuestra el flujo de la generación de subtítulos

3.2.2. Aprendizaje no supervisado

Esta rama del aprendizaje automático se utiliza cuando no nos interesa ajustar pares (entrada/salida), sino en aumentar el conocimiento estructural de los datos de entrada. Consiste en encontrar transformaciones interesantes de los datos de entrada sin la ayuda de etiquetas, con el propósito de visualizar los datos, comprimirlos, eliminar ruido o simplemente aumentar el entendimiento de los datos en cuestión.

Su principal aplicación se encuentra en el análisis de datos, a suele ser un paso necesario en la mejora del entendimiento de los datos de un dataset antes de la resolución por un algoritmo de aprendizaje supervisado.

Tipos de algoritmos no supervisados más comunes:

- Clustering: busca agrupar una serie de elementos según su semejanza. Por ello, un *cluster* es una agrupación de elementos similares.

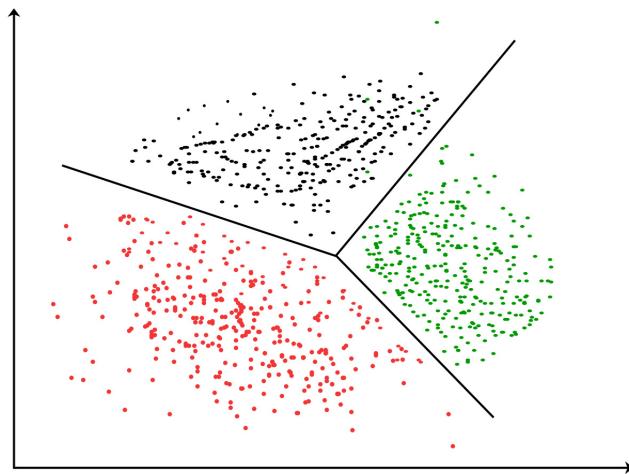


Figura 3.4: Resultado de la aplicación de clustering sobre un conjunto de individuos

- Reducción de dimensiones: es la transformación de datos de un espacio dimensional elevado a otro de menores dimensiones, manteniendo en este nivel menor de dimensionalidad características importantes.

Este tipo de algoritmos es muy común en campos que manejan grandes cantidades de observaciones y/o variables, como el procesamiento de señales, el reconocimiento de voz o la bioinformática.

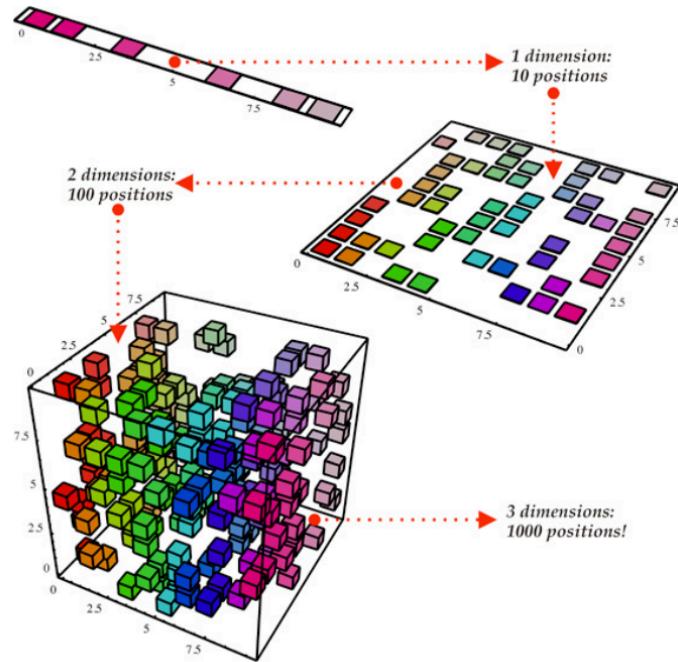


Figura 3.5: Etapas en la reducción de dimensiones.

3.3. Redes neuronales

En esta sección profundizaremos en las redes neuronales, desde la inspiración biológica hasta las redes convolucionales más famosas y desarrolladas.

3.3.1. Inspiración biológica

Una red neuronal biológica está compuesta por aproximadamente 86 billones de neuronas, conectadas entre ellas. Los científicos estiman que existen más de 500 millones de billones de conexiones en el cerebro humano. Incluso las redes neuronales artificiales más largas ni se acercan a este número.

Desde un punto de vista informatizado, podemos decir que una neurona es una unidad excitable que puede procesar y transmitir información mediante señales eléctricas y químicas. Esta unidad (la neurona) es el principal componente en nuestro sistema nervioso.

Estructura

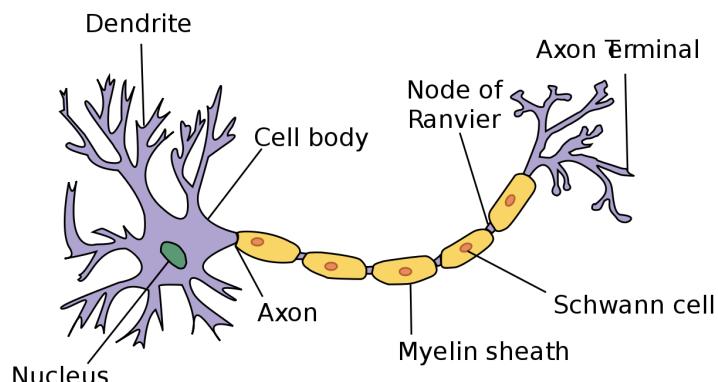


Figura 3.6: Estructura de una neurona biológica.

Las partes más importantes que componen una neurona son:

- El soma (o cuerpo celular): procesa las activaciones entrantes y las convierte en activaciones de salida.
- El núcleo: contiene material genético en forma de ADN.
- Las dentritas: son fibras que nacen desde el soma y provee de zonas receptoras para recibir las activaciones de otras neuronas.
- Axones: son fibras que actúan como líneas transmisoras que envían la activación a otras neuronas.

Funcionamiento

Los pasos clave en el funcionamiento de una neurona son:

1. Las señales de otras neuronas se almacenan en las dendritas.
2. El soma agrupa las señales entrantes (espacial y temporalmente).
3. Cuando se recibe la suficiente cantidad de señal la neurona se "dispara" (produce una diferencia de potencial).
4. Esta diferencia de potencial es transmitido a lo largo del axón hacia otras neuronas o hacia estructuras fuera del sistema nervioso (Por ejemplo músculos).
5. En caso de no recibir suficiente señal la señal almacenada en las dendritas caerá rápidamente y la neurona no se disparará.

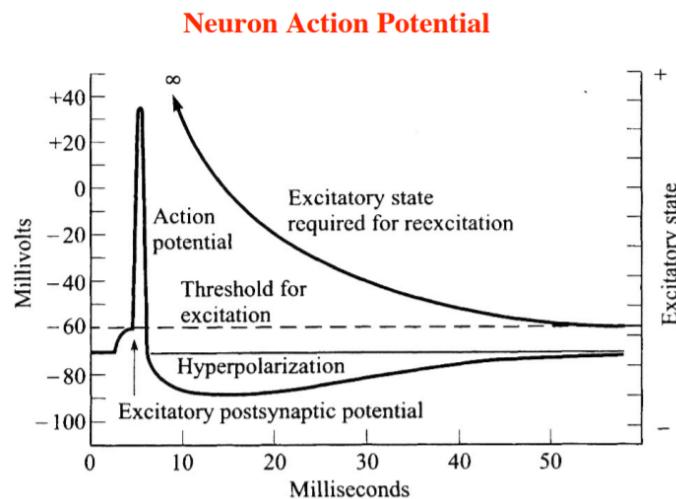


Figura 3.7: Diagrama que muestra el disparo de una neurona

3.3.2. El perceptrón

El perceptrón es un modelo lineal usado para la clasificación binaria. En el campo de las redes neuronales el perceptrón es considerado la neurona artificial.

El primer modelo matemático de dicha neurona artificial fué presentado por el psiquiatra y neuroanatomista Warren McCulloch y el matemático Walter Pitts en 1943. Este modelo se considera el precursor del perceptrón, y es llamado "Threshold Logic Unit." "TLU". Dicho modelo era capaz de simular puertas lógicas, como la AND, OR o la XOR.

Estructura

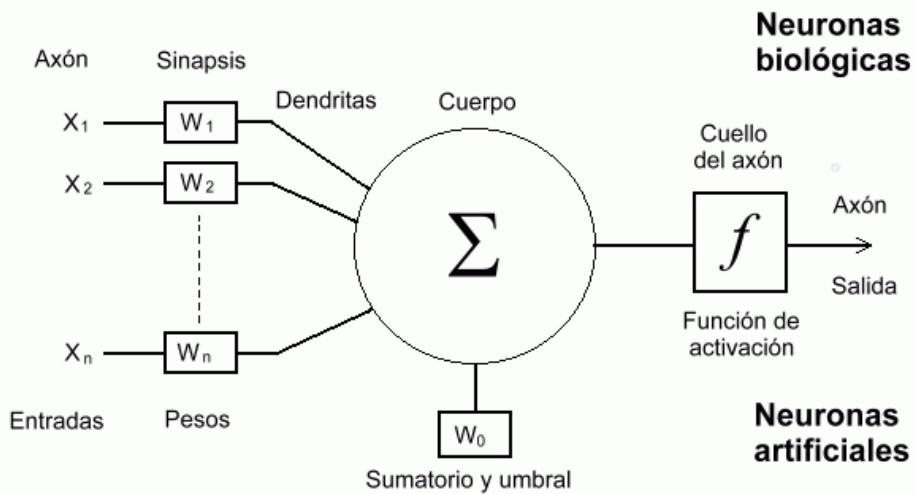


Figura 3.8: Comparativa de la estructura de la neurona biológica con el perceptrón

En la figura anterior podemos apreciar las distintas partes de la estructura de una neurona artificial:

- El conjunto de elementos de entrada (x_1, x_2, \dots, x_n).
- Los pesos asociados a cada entrada (w_1, w_2, \dots, w_n), los cuales se multiplicarán por cada elemento de entrada.
- Una función de agregación (\sum) que sumará las multiplicaciones de los pesos con sus respectivos elementos de entrada. A esta agregación se le suma un factor de parcialidad (bias). Este factor de parcialidad es utilizado para acelerar o frenar la activación de un nodo.
- Una función de activación (f).

Como hemos dicho, el perceptrón es un algoritmo para **aprender** un clasificador binario. Los elementos externos llegarán al perceptrón para devolver un valor. Por ello, teniendo nuestro perceptrón $f(\mathbf{x})$, siendo \mathbf{x} dichos elementos externos, siendo \mathbf{b} el factor de parcialidad (bias), siendo \mathbf{w} los pesos asociados a cada elemento externo y siendo $\mathbf{w}^* \mathbf{x}$ el producto escalar de \mathbf{x} y \mathbf{w} :

$$f(x) = \begin{cases} 1 & \text{si } w * x + b > 0 \\ 0 & \text{E.O.C} \end{cases}$$

Este clasificador funciona solo en caso de que el conjunto de datos de entrada sea linealmente separable. Este hecho produce que durante el entrenamiento de dicho perceptrón la clasificación erre, produciendo un clasificador de muy baja confianza.

Para poder entender cómo funciona dicho algoritmo primero tendremos que observar las distintas funciones de activación que podemos utilizar.

Funciones de activación

Las funciones de activación son funciones que dependiendo de lo que reciban se comportarán de una manera u otra. Un ejemplo es la función escalón de Heaviside, que devuelve 0 si el número que recibe es negativo o 1, si el número es mayor o igual que 0:

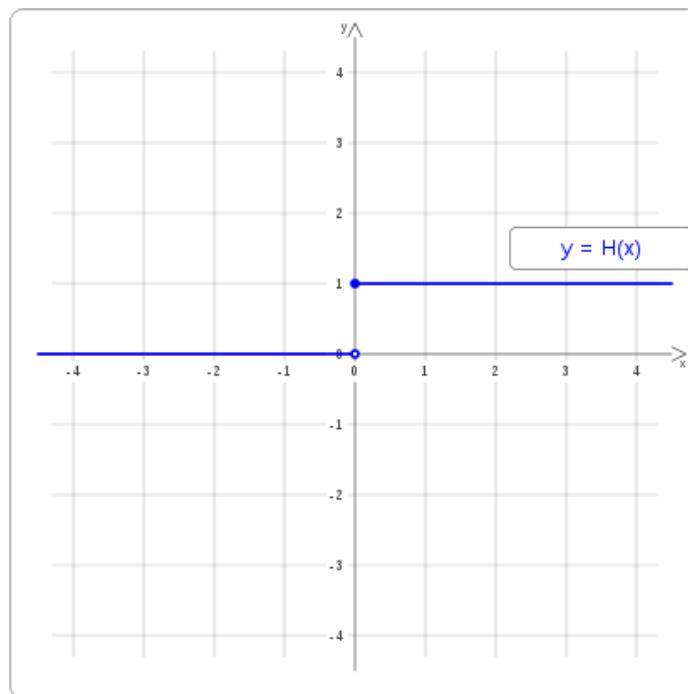


Figura 3.9: Gráfica de la función escalón de Heaviside.

Empezaremos por comentar las funciones de activación más comunes en el campo de las redes neuronales:

- ReLU (Rectifier Linear Unit): es la función que define la parte positiva de sus argumentos.

$$f(x) = x^+ = \max(0, x)$$

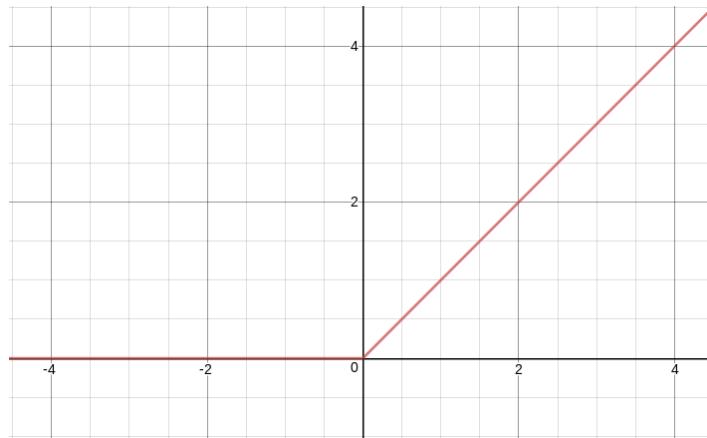


Figura 3.10: Gráfica de la función ReLU.

- Sigmoide (σ): es una de las funciones de activación más utilizadas. En este campo sirven para introducir la "no linealidad." en un modelo. Ya que el producto escalar entre los pesos y x es una combinación lineal, al añadirle esta "no linealidad." el resultado de dicha combinación se suaviza.

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

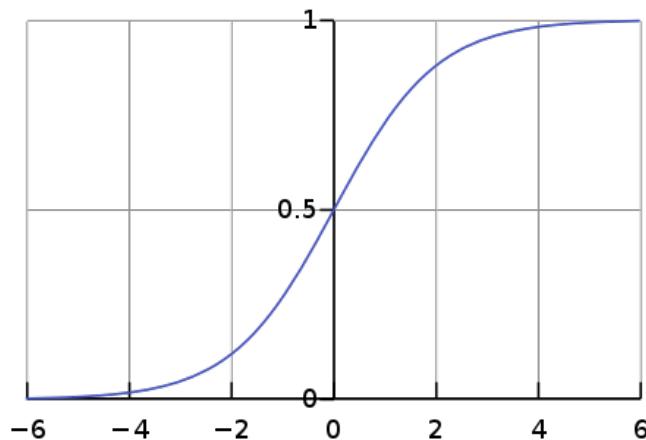


Figura 3.11: Gráfica de la función sigmoide.

- Softmax (función exponencial normalizada): es utilizada en la última capa de los clasificadores basados en redes neuronales, debido a que devuelve un vector de K valores reales a otro vector con K valores reales que suman 1. Si uno de los valores es negativo o pequeño, la función le dará poca probabilidad, mientras que si algún valor es más grande que el resto le dará más probabilidades.

Esta función solo se puede utilizar cuando las clases del clasificador son mutuamente exclusivas.

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

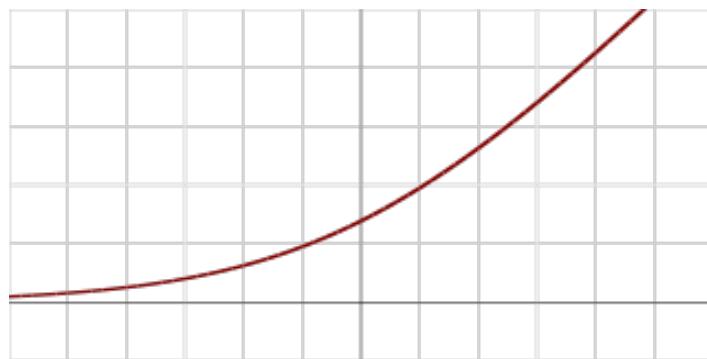


Figura 3.12: Gráfica de la función softmax.

Entrenamiento:

Poseyendo un conjunto de datos con N muestras, un vector de pesos (w_i), los valores asociados a cada característica de la muestra (x_i),

El entrenamiento del perceptrón sigue el siguiente procedimiento:

Algoritmo de entrenamiento del perceptrón 1:

Input: Datos de entrenamiento D

Output: Pesos w ya entrenados

Iniciarizar $w \leftarrow 0, b \leftarrow 0$

while no estén correctamente clasificados todos los x **do**

for $(x, y) \in D$ **do**
 if $y(w^*x + b) \leq 0$ **then**
 $w \leftarrow w + y * x$
 $b \leftarrow b + y$
 end

end

end

Componentes:

- b : es el factor de parcialidad (bias).
- y : la etiqueta esperada

3.3.3. Red Neuronal Tradicional

Una serie de perceptrones pueden agruparse entre sí para formar redes neuronales. Para ello cada neurona se conectará con todas las neuronas de las capas posteriores y anteriores.

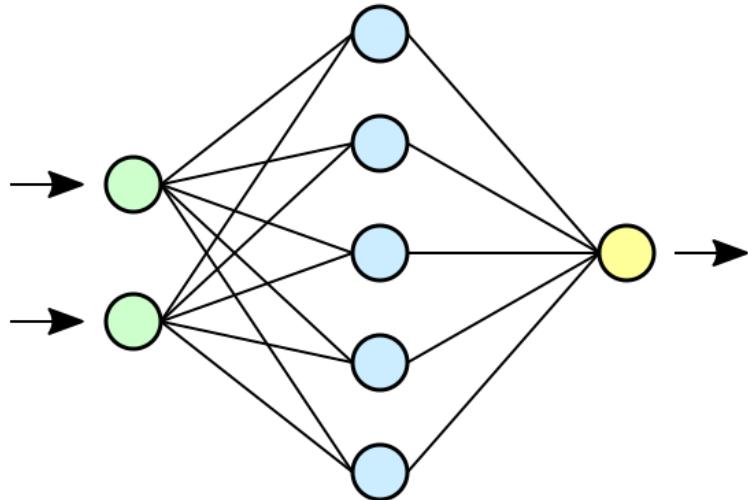


Figura 3.13: Perceptrón multicapa.

Estructura:

Las redes neuronales tradicionales poseen esta estructura:

- **Capa de entrada:** la primera capa de nuestra red neuronal. Por ella alimentaremos nuestra red con datos.
- **Capas ocultas (o totalmente conectadas):** estas serán las capas intermedias de nuestra red neuronal. Las neuronas de estas capas estarán totalmente conectadas tanto a la capa anterior como a la posterior. Son la clave para que una red neuronal pueda modelar funciones no lineales.
- **Capa de salida:** esta capa será la encargada de predecir la clase del elemento de entrada.
- **Conexiones:** Los pesos asociados a estas conexiones serán los que se ajustarán a la hora de realizar las predicciones a lo largo del entrenamiento. El algoritmo que usarán para entrenarse será el algoritmo de propagación hacia atrás.

La salida de cada neurona (n_i) vendrá dada por la suma ponderada de cada peso de cada conexión de entrada (w_{ij}) por su elemento de entrada correspondiente(x_i) mas el valor bias propio de cada neurona (b_i), siendo f la función de activación de cada nodo en la capa oculta y h la función de activación de la capa de salida, quedarán las ecuaciones de la figura 3.13 en las capa oculta y de salida de esta manera:

$$\begin{aligned}
 a_1^{(1)} &= f(w_{11}x_1^{(0)} + w_{21}x_2^{(0)} + b_1), \\
 a_2^{(1)} &= f(w_{12}x_1^{(0)} + w_{22}x_2^{(0)} + b_2), \\
 a_3^{(1)} &= f(w_{13}x_1^{(0)} + w_{23}x_2^{(0)} + b_3), \\
 a_4^{(1)} &= f(w_{14}x_1^{(0)} + w_{24}x_2^{(0)} + b_4), \\
 a_5^{(1)} &= f(w_{15}x_1^{(0)} + w_{25}x_2^{(0)} + b_5), \\
 o &= h(w_{25}x_1^{(1)} + w_{25}x_2^{(1)} + w_{25}x_3^{(1)} + w_{25}x_4^{(1)} + w_{25}x_5^{(1)} + b_6)
 \end{aligned} \tag{3.1}$$

Entrenamiento de la red neuronal.

Esta es la etapa más importante a la hora de conseguir un modelo preciso y rápido. Cuanto mejor es una red neuronal, mejor captará las señales y mejor detectará lo que es ruido.

Este proceso de aprendizaje consistirá en re-ajustar los pesos y los bias para cambiar la relevancia de ciertas características, aumentándola o disminuyéndola en base a los bits de cada peso.

En la mayoría de los dataset hay características fuertemente relacionadas con ciertas etiquetas (como puede ser en una imagen la relación entre la forma esférica de un balón). Las redes neuronales aprenden a base de prueba y error, aprendiendo ciegamente estas características ajustando los pesos para poder dar veredictos más acertados según avanza el proceso de aprendizaje.

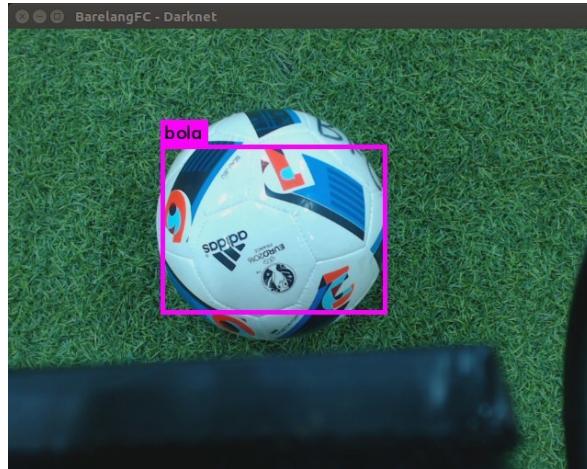


Figura 3.14: Detección de una pelota por una red neuronal.

3.3.4. Algoritmo de propagación hacia atrás:

Este algoritmo es el responsable de reducir el error durante la fase de entrenamiento de una red [8].

Algoritmo de Backpropagation (descenso del gradiente) 2:

Input: Datos de entrenamiento \mathbf{D} , tasa de aprendizaje η

Output: Pesos \mathbf{w} ya entrenados

Iniciar $\mathbf{w} \leftarrow$ valores aleatorios entre -1 y 1

while no se alcance una condición de terminación o el número de epoch **do**

for $(x, y) \in \mathbf{D}$ **do**

$y' = o(x);$

$error_i = y - y'$

$W_{ji} \leftarrow W_{ji} + \eta * a_j * error_i * g'(\Sigma_i)$

for Cada capa subyacente de la red **do**

$\Delta_j \leftarrow g'(\Sigma_j) * \Sigma_i * W_{ji} * \Delta_i$

$W_{kj} \leftarrow W_{kj} + \eta * a_k * \Delta_j$

end

end

end

Explicación del pseudo-código:

El algoritmo tiene 2 entradas:

- Los datos de entrenamiento, los cuales poseen sus respectivas salidas esperadas
- La tasa de aprendizaje, la cual marcará la velocidad a la que se actualizarán los pesos según avance el algoritmo.

El algoritmo inicializará los pesos de manera aleatoria, dentro del rango [-1,1] y comenzará el intervalo que iterará sobre cada elemento del conjunto de entrenamiento

hasta encontrar o bien una condición de terminación, como alcanzar un porcentaje de precisión, o bien se alcance el número de epoch preestablecido.

Lo primero que calculará el algoritmo será la salida de la red para el elemento actual (o). Después compararemos la salida esperada de ese elemento y la compararemos con la salida obtenida en el paso anterior y lo guardamos.

A partir de aquí actualizaremos los pesos sinápticos. Los primeros pesos que actualizaremos serán los que estén conectados a la última capa utilizando esta regla:

$$W_{ji} \leftarrow W_{ji} + \eta * a_j * error_i * g'(\Sigma_i)$$

Para actualizar cada peso le añadiremos la multiplicación del valor de la activación de la neurona a_j el valor del error anteriormente capturado y la derivada de la función de activación g' con el valor de la multiplicación elemento a elemento entre el conjunto de valores de activación de la capa anterior y los pesos asociados:

$$\Sigma_i = W_i * A_i$$

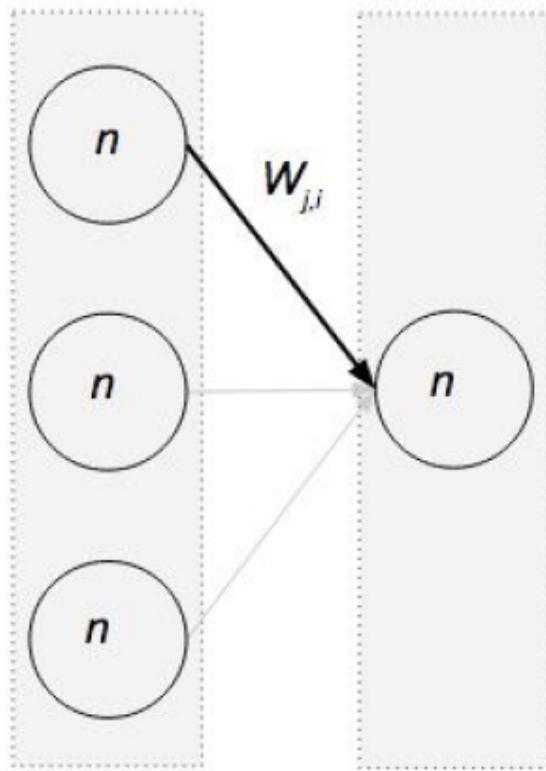


Figura 3.15: Actualización de un peso de la última capa.

Una vez actualizados los pesos asociados a la capa de salida realizaremos la propagación hacia atrás con cada capa anterior hasta actualizar todos los pesos. La forma de actualizar estos pesos es distinta:

$$\Delta_j \leftarrow g'(\Sigma_j) * \Sigma_i * W_{ji} * \Delta_i$$

En Δ_j almacenaremos la nueva regla de propagación para el error. Esta ecuación observa cada nodo de la capa actual y obtiene el valor del error actual Δ_i multiplicado por el peso de la conexión entre la capa anterior y la actual, multiplicándolo por el

resultado de la multiplicación punto a punto entre los pesos y los valores de activación de la anterior capa (Σ_i), y multiplicándolo por la derivada de la función de activación de la capa anterior por su Σ_j ($g'(\Sigma_j)$). Esto nos ofrece el error fracional para el nodo de la capa anterior conectado con el nodo de la capa actual, el cual se usará para actualizar el peso de la conexión entre ambos.

Ahora actualizaremos los pesos:

$$W_{kj} \leftarrow W_{kj} + \eta * a_k * \Delta_j$$

Al actualizar un peso de la capa anterior le añadiremos a su valor la multiplicación entre la tasa de aprendizaje η , el valor de activación de la neurona a_k y el error asociado Δ_j

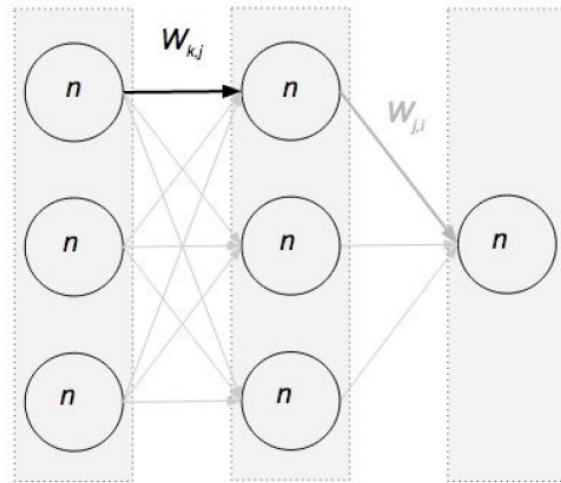


Figura 3.16: Actualización de un peso de las capas ocultas.

3.3.5. Funciones de pérdida probabilísticas

Son las funciones que proporcionarán a la red una manera de cuantificar el valor que hay que reducir para mejorar las predicciones.

Entropía cruzada binaria

A la verdadera posibilidad de que el valor de la clase de un elemento la llamaremos p_i , mientras que al valor de probabilidad que ha predicho lo llamaremos q_i . Dicha probabilidad es modelada usando a función logística: $g(z) = 1/(1 + e^{-z})$, siendo z el valor proporcionado por la salida de la función de activación de la anterior capa.

La probabilidad de que la salida $y = 1$ es proporcionada por la siguiente fórmula:

$$q_{y=1} = \hat{y} \equiv g(w * x) = 1/(1 + e^{-w*x})$$

De manera análoga, para calcular $y = 0$ tendríamos que restarle a 1 la probabilidad anteriormente calculada:

$$q_{y=0} = 1 - \hat{y}$$

Una vez obtenidos las distribuciones de probabilidad $p \in \{y, 1-y\}$ y $q \in \{\hat{y}, 1-\hat{y}\}$ podremos usar la fórmula de la entropía cruzada para medir la disparidad entre ambas distribuciones:

$$H(p, q) = -\sum_i p_i * \log(q_i) = -y * \log(\hat{y}) - (1 - y) * \log(1 - \hat{y})$$

Pérdida de Hinge

Su uso más común es para la clasificación binaria. También es muy utilizada en otro tipo de modelo llamado “clasificador de modelos de margen máximo”, comúnmente utilizados en máquinas de vectores de soporte. Para este modelo, los valores de las salidas esperadas deberán ser: $\{-1, 1\}$.

En la siguiente ecuación mostraré la función de Hinge, siendo y la salida esperada e \hat{y} siendo la salida obtenida:

$$\ell(y) = \max(0, 1 - y * \hat{y})$$

3.3.6. Funciones de pérdida regresivas

Error cuadrático medio

Esta función de pérdida la usaremos cuando se requiera un valor real de salida (en clasificación binaria no tiene porqué ser un 1 o un 0, puede ser un valor real entre ellos).

Considerando que debamos predecir solo una salida, el error de dicha predicción tendrá la siguiente fórmula:

$$L(w, b) = \frac{1}{N} \sum_{i=1}^N (\hat{y}_i - y_i)^2$$

Esta fórmula calcula la media del error cuadrático de cada ejemplo, siendo N el número de elementos con los que se ha alimentado al modelo (puede ser de un epoch, batch...).

Error cuadrático medio probabilístico

Con esta variedad del error cuadrático medio se puede obtener una probabilidad basada en la relación entre el error de la predicción con el valor real:

$$L(w, b) = \frac{1}{N} \sum_{i=1}^N \frac{100 * |\hat{y}_i - y_i|}{y_i}$$

3.3.7. Optimizadores

Una vez obtenidas las funciones de pérdidas tendremos que encontrar una manera de minimizar los resultados obtenidos en base a los parámetros de nuestro modelo.

Estos son algunos de ellos:

Adagrad

Es un algoritmo de optimización que utiliza gradientes. Adapta la tasa de aprendizaje a los parámetros, realizando actualizaciones grandes en los pesos cuando se descubre un ejemplo con características poco frecuentes, y actualizaciones chicas para los ejemplos con rasgos comunes.

Es la raíz cuadrada de la suma de los cuadrados del historial de gradientes. Adagrad acelera el entrenamiento al principio y lo frena según alcanza la convergencia, consiguiendo un proceso de entrenamiento más suave.

De base siempre usará la tasa de aprendizaje de entrada a la que llamaremos η (como anteriormente), pero para "actualizarla" lo que haremos será multiplicarla por los elementos de un vector $\{G_{jj}\}$, el cual es la diagonal del producto tensorial de la matriz:

$$G = \sum_{\tau=1}^t g_\tau * g_\tau^T$$

siendo $g_\tau = \nabla Q_i(w)$ el gradiente en la iteración τ . La diagonal que buscamos y que utilizaremos para "modificar" el valor de la tasa de aprendizaje será:

$$G_{j,j} = \sum_{\tau=1}^t g_{\tau,j}^2$$

En cada iteración actualizaremos $G_{j,j}$, al tener que calcularse los gradientes de los pesos, los cuales actualizaremos de esta manera:

$$w_j = w_j - \frac{\eta}{\sqrt{G_{j,j} g_j}}$$

RMSProp

Es otro método en el cual la tasa de aprendizaje se adapta a los parámetros del modelo. La idea principal es dividir la tasa de aprendizaje por el promedio de los gradientes cuadráticos adyacentes para cada peso, y por eso se llama así ("root mean square"). Las ecuaciones de dicho promedio y de la regla de actualización de los pesos serán:

$$\begin{aligned} E[g^2]_t &= \beta E[g^2]_{t-1} + (1 - \beta)(\nabla Q_i(w))^2 \\ w_t &= w_t - \frac{\eta}{\sqrt{E[g^2]_{t-1}}} * \nabla Q_i(w) \end{aligned}$$

Siendo β un número llamado "factor de olvido" o "parámetro de movimiento medio".

3.3.8. Regularización

Ayuda a la red a minimizar el sobreajuste y la hiperparametrización. Pesos con valores pequeños llevan a hipótesis más simples, las cuales son más generalizables que pesos sobrecargados con los elementos de entrenamiento.

En cuanto más crezca el conjunto de entrenamiento, más se palian los efectos de la regularización, lo cual es apropiado, puesto que un exceso de aprendizaje de características relativos a una serie de ejemplos de entrenamiento tiende a sobreentrenamiento. Por ello, el mayor regularizador es una gran cantidad de datos.

Para poder añadir regularización a nuestra red insertaremos un mecanismo:

Dropout

Es usado para mejorar el entrenamiento de una red neuronal. Para ello omitirá en cada iteración neuronas de manera aleatoria, por lo que no podrán contribuir tanto en la propagación hacia delante como en la propagación hacia atrás.

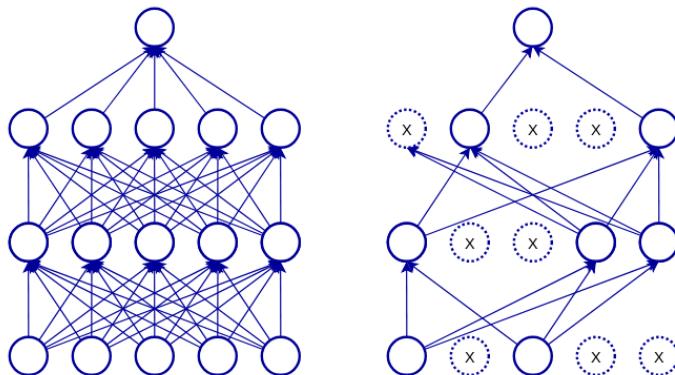


Figura 3.17: Representación del mecanismo de dropout en las neuronas de la red.

3.4. Deep learning

Entre las redes neuronales tradicionales y el deep learning (o aprendizaje profundo) existen varias diferencias:

- Poseen más capas y neuronas por capa
- Las conexiones entre ellas son más complejas
- Los avances tecnológicos, que permiten más potencia de cómputo
- La extracción automática de características

Este aumento de neuronas y capas es propiciado por la necesidad de resolver problemas más complejos. También existen más tipos de capas, como las capas convolucionales en las redes convolucionales o las conexiones que van de una neurona a ella

misma, en las redes recurrentes. Al tener más conexiones habrá una mayor cantidad de parámetros a optimizar, por lo que el avance tecnológico tan grande que hemos tenido las últimas décadas con los procesadores gráficos ha sido una parte vital para que se propiciase dicha evolución.

Profundizaremos en la que será la protagonista de este proyecto: las redes convolucionales.

3.4.1. Redes convolucionales

El principal objetivo de una red neuronal convolucional es aprender las características de una imagen mediante convoluciones. Este tipo de redes rinden muy bien de cara al reconocimiento de objetos en imágenes o videos, siendo de manera consistente las que mejor funcionan en las competiciones. Otra función que pueden tener las redes convolucionales es la de análisis y reconocimiento de sonidos.

Gran parte de la fama que posee el deep learning proviene del reconocimiento de imágenes ofrecido por este tipo de redes. También son pioneras en avances de la visión en máquinas, con aplicaciones para robots, drones o coches autónomos.

Estas redes aprovechan su potencial cuando las imágenes poseen algún tipo de estructura, como va a ser en nuestro caso.

Inspiración biológica

La estructura de las redes convolucionales se inspira en la corteza visual (o córtex visual) de los animales. Esta corteza es sensible a los detalles de lo que vemos, o lo que es lo mismo, al **campo visual**. Las células de la corteza visual son capaces de observar y asimilar las correlaciones espaciales de las imágenes que nuestro cerebro procesa, actuando como filtro.

Hay dos clases de células en esta región del cerebro:

- Las simples: se activan cuando detectan bordes.
- Las complejas: se activan cuando lo que intenta procesar la corteza es más complicado y son invariantes respecto a la posición, es decir, que son capaces de entender qué está observando el individuo sin importar la posición del patrón.

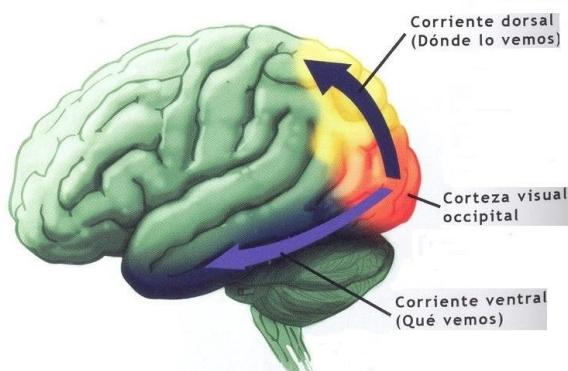


Figura 3.18: Corteza visual.

Estructura

En esta sección nos enfocaremos en la estructura de una red convolucional, observando sus capas y entendiéndolas.

En la siguiente imagen vemos ilustrada su estructura:

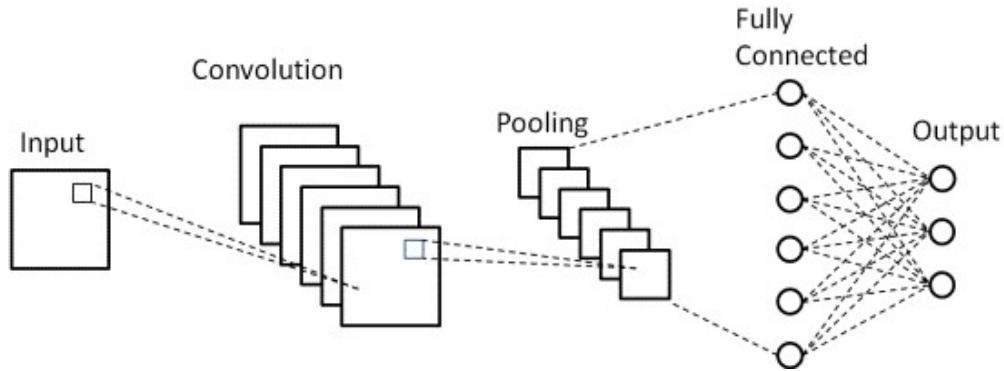


Figura 3.19: Estructura de una red convolucional.

Las distintas capas se agrupan en tres grupos:

- La capa de entrada, por la que recibiremos los datos en nuestra red.
- La parte de extracción de características, compuesta las capas de convolución y las capas de pooling.
- La parte de clasificación, en la que se realizará la predicción. Formada por las capas totalmente conectadas y la capa de salida.

Ahora nos adentraremos en cada parte por separado.

Capa de entrada

En esta capa se introducirán y guardarán los datos sin haberlos procesado antes con nuestra red. Aceptará datos tridimensionales, generalmente con la forma ((anchura*altura),profundidad), siendo la profundidad la cantidad de canales de color.

Nuestras imágenes las trataremos como matrices con la misma arquitectura que indicamos anteriormente:

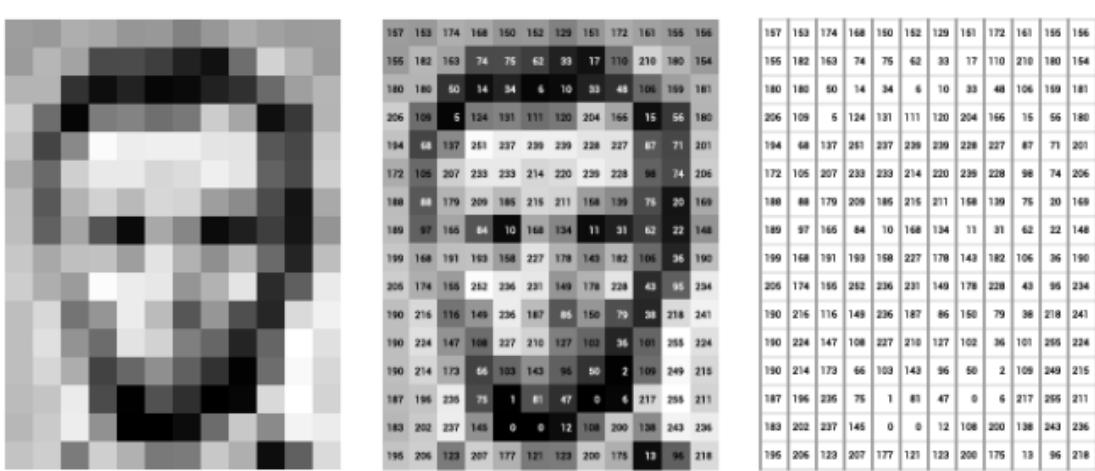


Figura 3.20: Matriz de valores de intensidad de los píxeles de una imagen con un canal (escala de grises).

Capa convolucional

Estas capas son las protagonistas en este tipo de arquitecturas. En ellas nuestra red aprenderá patrones que les servirán para poder clasificar las imágenes que reciba.

Al pasar los datos por una capa convolucional los transforma utilizando pequeños grupos de neuronas de la capa anterior, llamados **filtros**.

Profundizaremos ahora en este concepto profundizando en la convolución:

Convolución

La convolución descrita matemáticamente es una función que es la integral de sumatorio de dos funciones componentes, la cual mide lo superpuesta que se encuentra una función sobre la otra. Es importante en distintos campos, como la física o las matemáticas.

Toma como entrada la matriz de píxeles de una imagen, le aplica un **kernel**(o núcleo) convolucional y devuelve un mapa de características encontradas en una imagen.

Esta operación es conocida comúnmente como *detector de características* en las redes convolucionales. Esta capa no tiene porqué recibir una matriz de píxeles, sino que también puede recibir otros mapas de características devueltos por otras convoluciones.

Esta operación queda ilustrada en esta imagen:

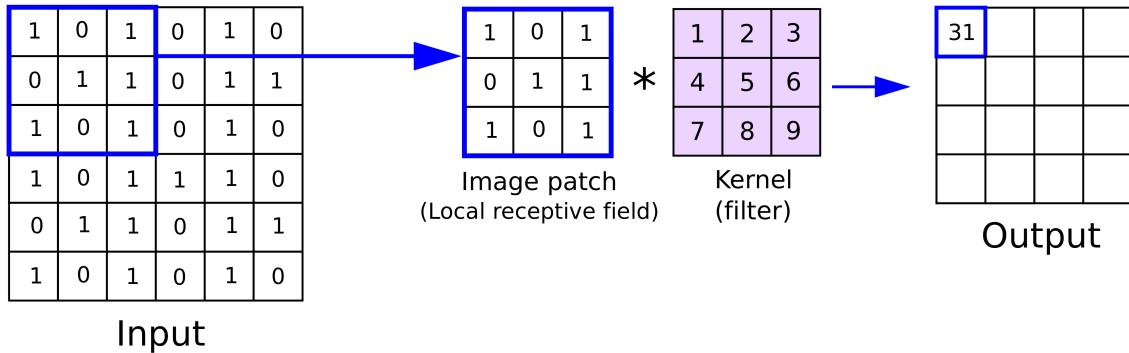


Figura 3.21: Operación de convolución.

Como se puede observar, primero recibimos una matriz de tamaño (6x6). Con un kernel de tamaño (3x3) recorreremos la matriz original, realizando una multiplicación punto a punto entre dicho kernel y las subdivisiones de la matriz original, siendo de tamaño (3x3), ya que dijimos que se realizará una multiplicación punto a punto. Dicha operación quedaría así, empezando por la matriz que iría desde (x=0,x=2) y (y=0, y=2):

$$1 * 1 + 0 * 2 + 1 * 3 + 0 * 4 + 1 * 5 + 1 * 6 + 1 * 7 + 0 * 8 + 9 * 1 = 31$$

Guardamos este valor en nuestro mapa de características, y realizaríamos la misma operación sobre la matriz (3*3) que abordaría desde (x=1,x=3) y (y=1,y=3).

El resultado de la convolución completa es un mapa de características con dimensiones (4x4), al haber usado un kernel (3x3).

Al utilizar distintos tipos de kernel se puede obtener distinta información de las imágenes, como muestra este ejemplo:

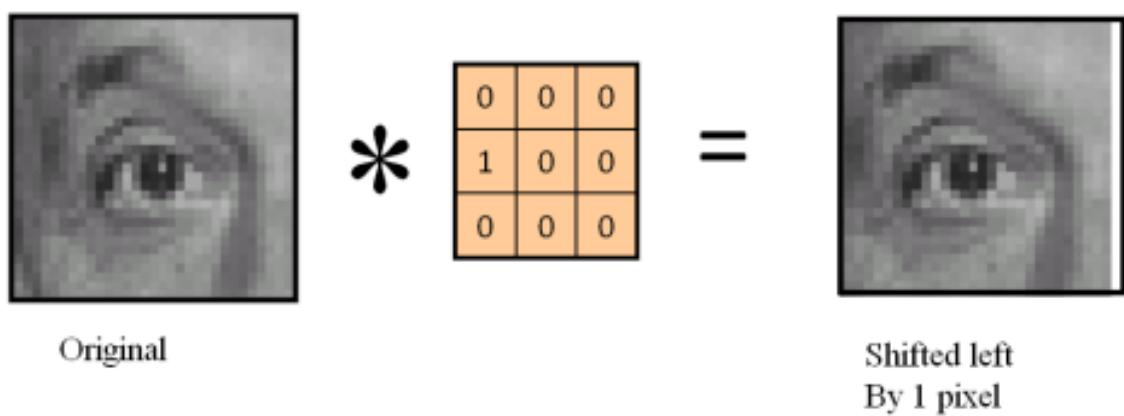
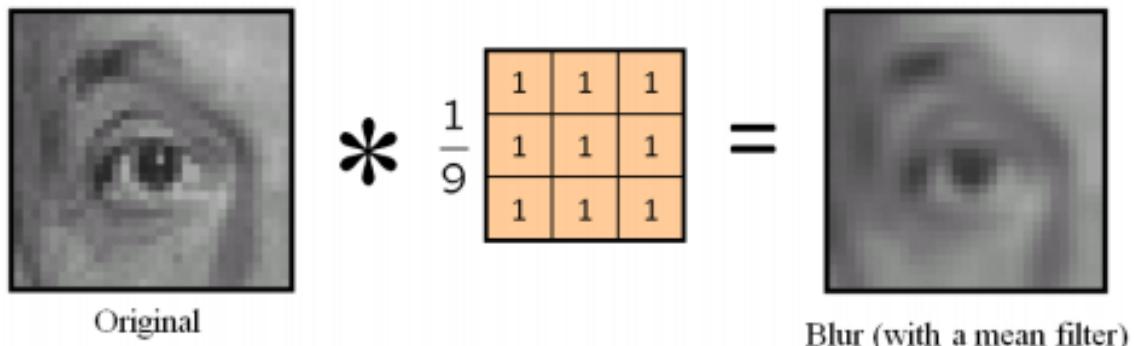


Figura 3.22: Distintos kernels con distintos mapas de características.

Cada filtro produce un mapa de características, por lo que en las redes que construiremos a lo largo del proyecto en cada capa convolucional realizaremos varios filtrados a la vez, aumentando el conocimiento de nuestra red.

Capa de pooling

Se suelen insertar entre las capas convolucionales. Se escoge estructurarlas de esta manera para progresivamente reducir el tamaño de los datos a representar. También ayudan a evitar el sobreentrenamiento.

Dependiendo de la capa de pooling se pueden utilizar dos operaciones para reducir el tamaño: *max*, el cual buscará el máximo de una porción del mapa de características recibido desde la capa convolucional entrante, o *average*, el cual calculará la media de esa porción. En nuestro caso hablaremos únicamente de *max*, puesto que es la que utilizaré para todas las redes y ofrece mejores resultados, puesto que mantiene las características importantes, mientras que *average* las difumina.

A esta operación la llamaremos *max pooling*. Con un filtro de tamaño (2x2) esta operación devolverá el mayor entre los 4 números posibles. Lo podemos ver ilustrado en la siguiente imagen:

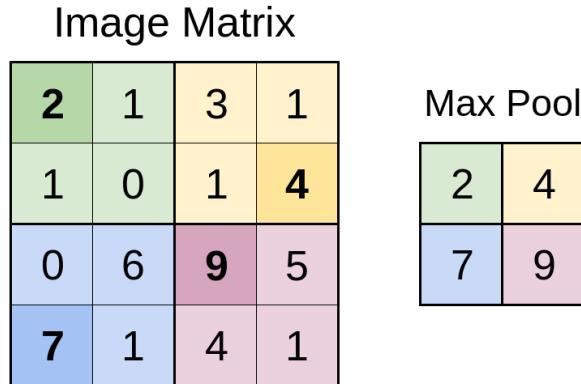


Figura 3.23: Operación de max pooling.

Para la primera ventana (2x2) esta sería la operación realizada:

$$\text{Max}(2, 1, 1, 0)$$

Y así sucesivamente con el resto, reduciendo el tamaño del mapa de características y cumpliendo su cometido.

Capa totalmente conectada

Esta será la capa que producirá la predicción de nuestros modelos. Podría ser por sí sola una red neuronal tradicional, estando todas las neuronas de una capa cualquiera conectadas a las neuronas de las capas anterior y posterior de la red. Poseen por tanto los parámetros e hiperparámetros (como pesos o bias) que una red neuronal básica podría utilizar.

Durante el entrenamiento, esta parte será la encargada de relacionar lo que ha detectado la parte de extracción de características (capas convolucionales y de pooling) con la salida esperada, ajustándose como cualquier red neuronal tradicional.

3.5. Estado del arte

Durante esta sección realizaré un repaso sobre una de las formas más comunes de elaborar modelos, "transfer learning", y las redes convolucionales más famosas del momento.

3.5.1. Transfer Learning

El transfer learning, como su nombre en español dice, significa "transferencia de aprendizaje". Con ello nos referimos al uso de redes ya entrenadas para adaptarlas a nuestros problemas. Este tipo de modelado es uno de los más cómodos para el investigador, puesto que la mayoría de pesos ya vienen entrenados, ahorrando tiempo y costes. La forma de adaptar una red entrenada se ve ilustrada en la siguiente imagen:

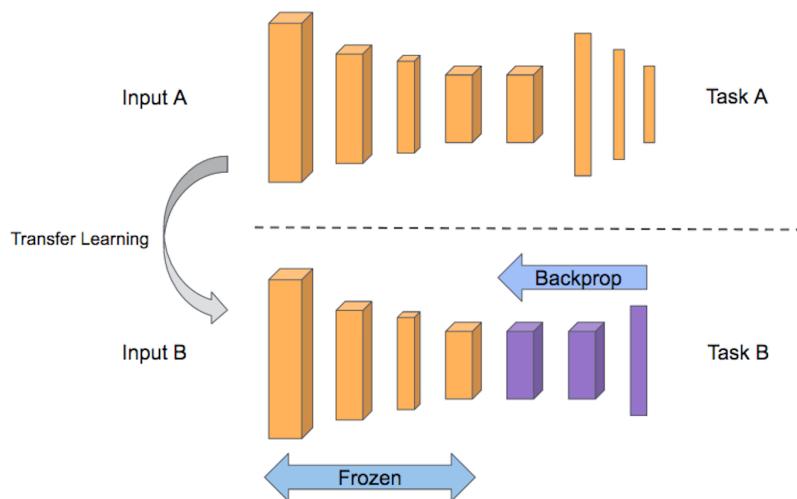


Figura 3.24: Adaptación de una red entrenada para reutilizarla.

Para poder aplicar transfer learning sobre un problema requeriremos de una serie de pasos.

- Primero debemos poseer una red ya entrenada, como se puede apreciar en la imagen superior. Lo único necesario es que esta red haya sido entrenada enfocada a un problema similar al nuestro, o que sea una red muy grande con un entrenamiento exhaustivo y cuidado, como pueden ser las redes que utilizaré para aplicar transfer learning.
- Lo segundo será congelar gran parte de la red. Con congelarla me refiero a bloquear los pesos a la hora del segundo entrenamiento, siendo este necesario para poder adaptar la red a nuestro problema.
- Después tendremos dos opciones. La primera es entrenar los pesos que no se encuentran congelados con nuestros datos, como en la imagen superior. La segunda, que es la que utilizaré en mi caso, es remover las capas totalmente conectadas de la red e insertar otras nuevas, entrenando desde cero esos pesos.

3.5.2. VGG

Este modelo de red convolucional fué presentado por Karen Simonyan y Andrew Zisserman en 2014, investigadores de la universidad de Oxford en su artículo: "Very Deep Convolutional Networks for Large-Scale Image Recognition". Su simpleza hace que sea una de las redes convolucionales más utilizadas para ImageNet. Con este dataset, en 2014 VGG16 alcanza una precisión del 92.7 % tras ser entrenada por semanas usando GPU's Titan Black de NVIDIA, ocupando el quinto lugar en las mejores predicciones del propio reto del proyecto Imagenet, "ILSVRC" (ImageNet Large Scale Visual Recognition Challenge).

Estructura

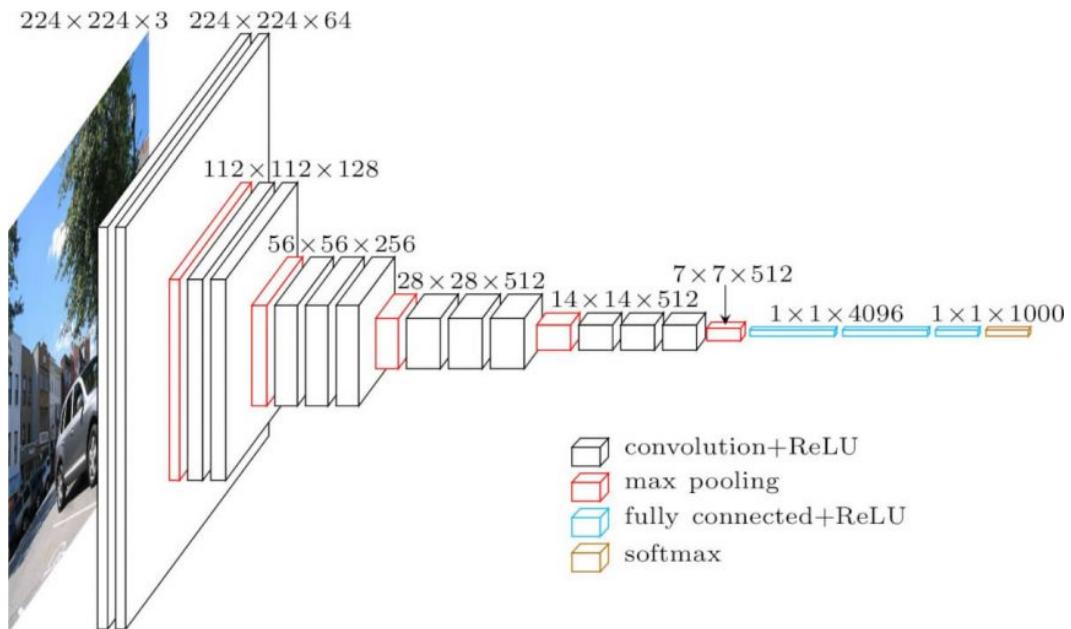


Figura 3.25: Estructura de VGG16.

La red posee:

- 2 series de 2 capas convolucionales mas 1 capa de pooling + 3 series de 3 capas convolucionales mas 1 de pooling
- 3 capas totalmente conectadas de tamaño (1x1) con 4096 canales, siendo la tercera de 1000 canales, preparada para conectarse a la capa de salida
- La capa de salida es de tipo softmax, por lo que devolverá el mayor porcentaje de predicción de todas las clases posibles, siendo esta una entre mil.

3.5.3. InceptionV3

Inception es la arquitectura desarrollada por Google y presentada en 2015. Me centraré en su tercera versión, InceptionV3 puesto que es la que he implementado.

Esta red también ha sido entrenada con Imagenet, ocupando el tercer lugar en dicha competición:

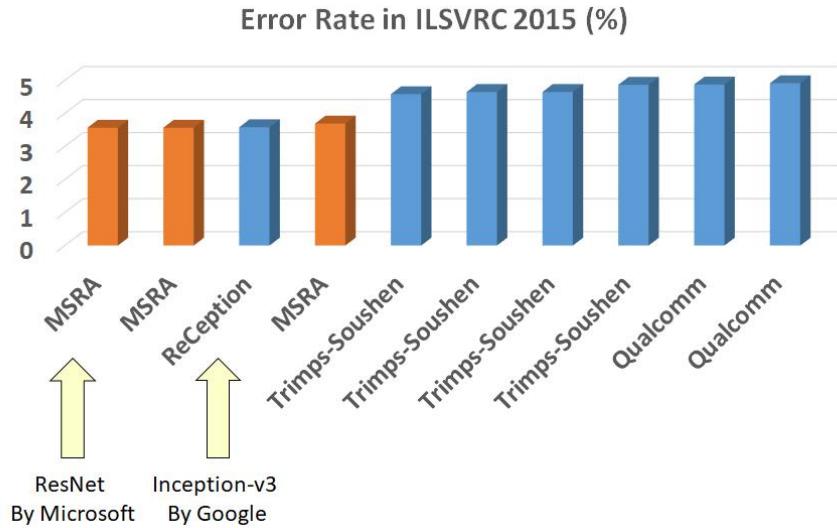


Figura 3.26: Resultados de ILSVRC 2015.

Estructura

InceptionV3 posee 42 capas de profundidad, como podemos observar en la siguiente imagen:

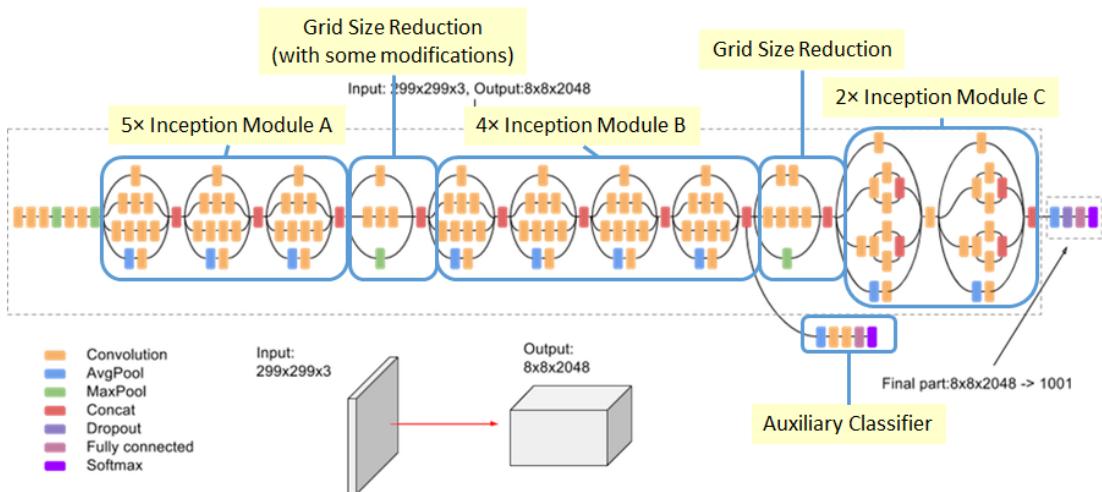
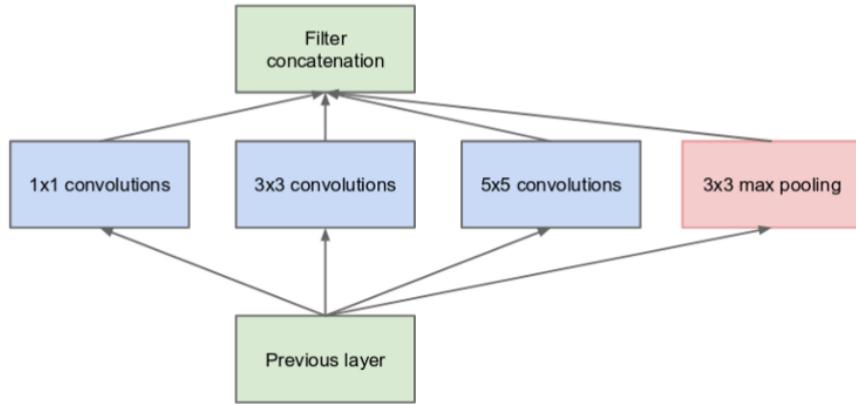


Figura 3.27: Estructura de Inceptionv3.

Las 7 primeras capas son lo que cualquiera se esperaría, pero a partir de la octava la estructura se complica.

Aparecen los módulos de Inception. Estos módulos son implementados para realizar un trabajo computacional más eficiente y para solucionar parte del sobreentrenamiento, entre otras problemáticas del deep learning. Para ello, en vez de agrupar capas convolucionales secuencialmente (como por ejemplo, con VGG), las agrupará en el mismo nivel. Esta sería una ilustración de la estructura de un módulo:



(a) Inception module, naïve version

Figura 3.28: Estructura de un módulo Inception.

De abajo a arriba, podemos observar que primero se hacen las convoluciones en paralelo, además de una capa de max pooling, y dicho resultado se concatena, enviándoselo al siguiente módulo o capa. De esta forma la red se vuelve cada vez más ancha, y por lo tanto, más barata computacionalmente hablando, puesto que las convoluciones se hacen en paralelo, sin tener que esperar a que lleguen los datos de una capa a otra.

3.5.4. ResNet

ResNet (o red neuronal residual) es la red desarrollada por Microsoft. Como podemos apreciar en la figura 3.26, ese mismo año quedó primera en el reto de Imagenet.

La estructura de esta red se basa en las células piramidales del cortex cerebral. La diferencia con el resto de neuronas es la gran cantidad de ramificaciones que poseen tanto en los axones como en las dentritas, lo que le permite comunicarse con muchas más neuronas que la neurona media.

Estructura

ResNet simula implementando conexiones que atajan capas, saltándose las que se encuentren entre la capa origen y la destino.

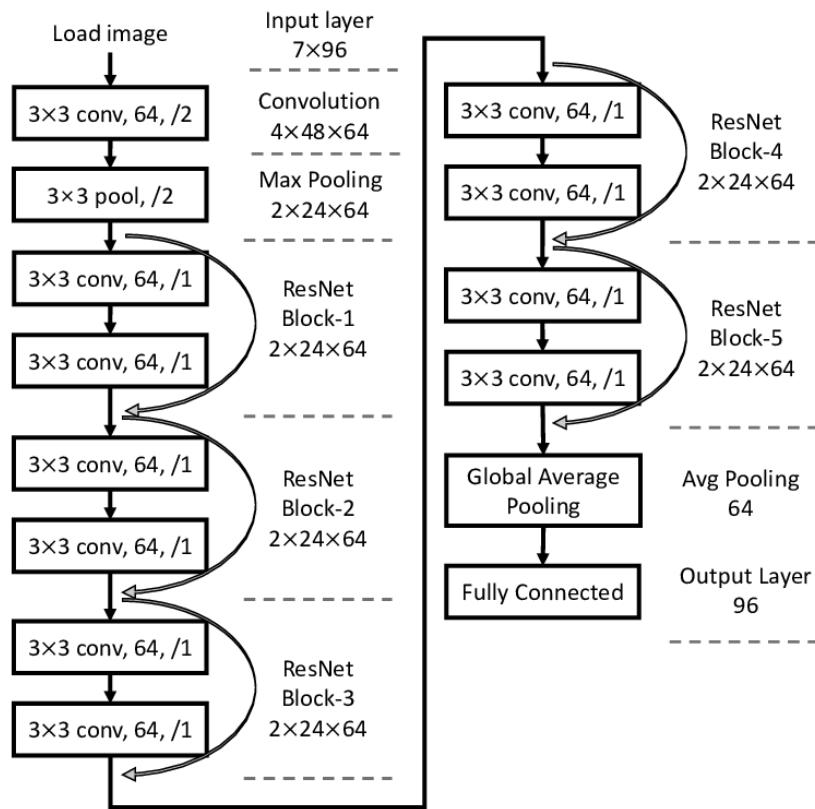


Figura 3.29: Estructura de las redes ResNet.

Una de las razones más importantes para añadir saltos entre conexiones es para mitigar el problema de saturación de precisión, donde añadir más capas a una red considerablemente profunda produce un aumento en el error durante el entrenamiento. Otra razón es que simplifica considerablemente la red, puesto que utiliza menos capas durante las primeras fases del entrenamiento, acelerando el aprendizaje. Después la red gradualmente reestablece las capas residuales para aumentar el número de características que la red pueda aprender.

Sin partes residuales, una red explora más el campo de características, lo que la hace más vulnerable a perturbaciones que distraiga a la red de fijarse en las características útiles, por lo que necesitaría más datos para reconducirse,

4. Tecnologías

4.1. Introducción

En este capítulo introduciremos las tecnologías, tanto hardware como software utilizadas para el desarrollo, tanto de la experimentación como de la aplicación web.

4.2. Hardware

4.2.1. Equipo personal

El equipo utilizado para este proyecto es un Portátil ASUS con un procesador i7-6700HQ, 16 GB de memoria RAM, con windows 10 como sistema operativo y una tarjeta gráfica NVidia GeForce GTX 960m.



Figura 4.1: Equipo utilizado para la elaboración del proyecto.

4.2.2. Google Colab



Figura 4.2: Logotipo de Google Colab.

Google Colaboratory es el entorno de desarrollo de python en la nube proporcionado por Google. Te permite ejecutar y programar en Python desde el navegador y aprovecharte de las siguientes ventajas:

- Viene configurado por defecto
- Proporciona una GPU de manera gratuita
- Facilita la compartición de contenido, así como subir el código a GitHub o enlaces él

Todo el desarrollo de las redes neuronales, desde el tratamiento del dataset hasta las pruebas de todas las redes con el conjunto de pruebas.

Para aprovechar los servicios de esta plataforma lo único necesario es tener una cuenta de Google.

4.3. Software

En esta sección mostraré el software utilizado.

4.3.1. Python3

Durante el desarrollo de este trabajo he utilizado Python3, siendo este el lenguaje de programación más cercano a mí y uno de los más utilizados para el deep learning, gracias a librerías como Keras o Pytorch.

4.3.2. NumPy

NumPy es una librería de Python que te permite utilizar vectores y matrices, además de realizar operaciones matriciales con ellos. Esta librería destaca sobre las demás porque todas sus operaciones son compiladas en C, por lo que mantiene las funcionalidades que facilita python aprovechándose de la rapidez computacional ofrecida por C.

Instalación

Para instalarlo se puede utilizar el gestor de paquetes "pip", proporcionado con Python3:

```
$ python3 -m pip install --upgrade pip  
$ python3 -m pip install --upgrade numpy
```

4.3.3. Tensorflow

Es una librería de Python que aporta funcionalidades para trabajar con tensores.

Instalación

```
$ python3 -m pip install --upgrade pip  
$ python3 -m pip install --upgrade tensorflow
```

4.3.4. Keras

Es una librería de Python que aporta funcionalidades del deep learning.

Instalación

```
$ python3 -m pip install --upgrade pip  
$ python3 -m pip install --upgrade keras
```

4.3.5. Pillow

Es una librería que aporta funcionalidades para el procesamiento de imágenes.

Instalación

```
$ python3 -m pip install --upgrade pip  
$ python3 -m pip install --upgrade Pillow
```

4.3.6. OS

Es una librería de Python, la cual ofrece la posibilidad de utilizar distintas funcionalidades propias del sistema operativo, como puede ser la manipulación de rutas locales o copiar un archivo de una carpeta a otra. Lo usaré para generar la estructura de directorios necesaria para el proyecto.

Instalación

```
$ python3 -m pip install --upgrade pip  
$ python3 -m pip install --upgrade os_sys
```

4.3.7. Flask

Flask será la librería que utilizaremos para realizar la aplicación web combinando Python con Jinja2, un compilador de plantillas HTML.

Instalación

```
$ python3 -m pip install --upgrade pip  
$ python3 -m pip install --upgrade virtualenv  
  
$ python3 -m pip install --upgrade pip  
$ python3 -m pip install --upgrade flask
```

5. Implementación y pruebas

Esta sección la dividiremos en 3 partes. La primera se compone por el preprocesado del dataset , la segunda en la creación, entrenamiento y pruebas de cada red, y la tercera es la aplicación web que utilizará las redes de la primera parte para predecir desde la aplicación las radiografías proporcionadas por el usuario.

5.1. Primera parte: preprocesado del dataset

Primero empezaremos por observar nuestro dataset.

5.2. Segunda parte: Implementación de las redes

El objetivo de nuestro proyecto es crear, entrenar y probar modelos de clasificación binaria, proporcionando un veredicto sobre si un sujeto padece de neumonía o no. Todas las redes han sido entrenadas en base a los mismos parámetros para que la competencia entre ellas sea lo más justa posible. Observaremos la implementación de las redes:

5.2.1. Primera red.

Esta será la primera red que crearemos. El propósito de esta red es mostrar el rendimiento de una red sin medidas de regularización, manteniendo una estructura simple (puesto que solo se han implementado capas de convolución, maxpooling y capas totalmente conectadas) y una efectividad considerable.

Estructura

La estructura la podemos visualizar en la siguiente imagen:

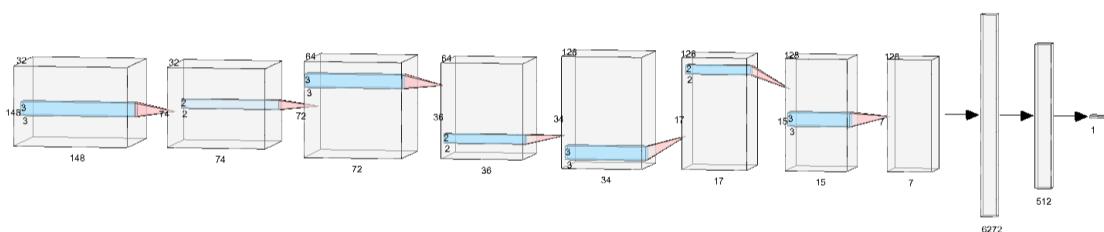


Figura 5.1: Visualización de la primera red.

Los bloques grises representan las capas, tanto convolucionales como de max-pooling como completamente conectadas.

Durante el preprocesado reestructuramos las imágenes para que tuvieran un tamaño de (150,150,3). La profundidad de una capa convolucional indica el número de filtros, por lo que según se adentra la imagen en la red, más patrones intentará encontrar nuestra red en cada capa, por ello decidido aumentar la profundidad en cada capa convolucional.

Después, al utilizar filtros de tamaño (2x2) en cada capa de max-pooling, las dimensiones de altura y anchura se verán reducidas a la mitad mientras que los datos se crucen con ellas.

Llegando al final de la red nos encontramos con las capas totalmente conectadas. Primero con una capa de profundidad 6072, que servirá para aplanar las capas convolucionales hasta llegar a la última capa, la cual es de una neurona y proporcionará un valor entre 0 y 1, que indicará si el sujeto está sano o enfermo, respectivamente.

Este será el sumario de la red:

Model: "sequential_3"		
Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d_12 (MaxPooling)	(None, 74, 74, 32)	0
conv2d_13 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_13 (MaxPooling)	(None, 36, 36, 64)	0
conv2d_14 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_14 (MaxPooling)	(None, 17, 17, 128)	0
conv2d_15 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_15 (MaxPooling)	(None, 7, 7, 128)	0
flatten_3 (Flatten)	(None, 6272)	0
dense_6 (Dense)	(None, 512)	3211776
dense_7 (Dense)	(None, 1)	513
<hr/>		
Total params: 3,453,121		
Trainable params: 3,453,121		
Non-trainable params: 0		

Figura 5.2: Sumario de la estructura de la primera red.

Como podemos observar, nuestra red posee **3.453.121** parámetros a entrenar.

Entrenamiento

El entrenamiento se ha realizado con 3784 imágenes de entrenamiento y 947 imágenes de validación (lo cual constituye el 80 % de nuestro dataset). El número de batches, por cada epoch será de 20, por lo que el número de pasos que observaremos en cada epoch será de 190, al haber 3784 imágenes en entrenamiento entre 20 batches

(189.2, al cual para no perder ese 0.2 sobrante de imágenes se redondeará hacia el entero superior).

Estas gráficas mostrarán los valores de precisión y pérdida durante el entrenamiento:

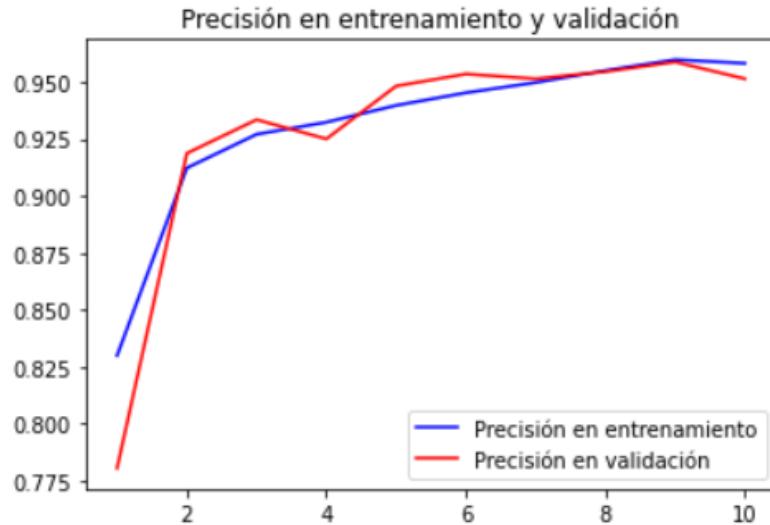


Figura 5.3: Valores de precisión durante el entrenamiento.

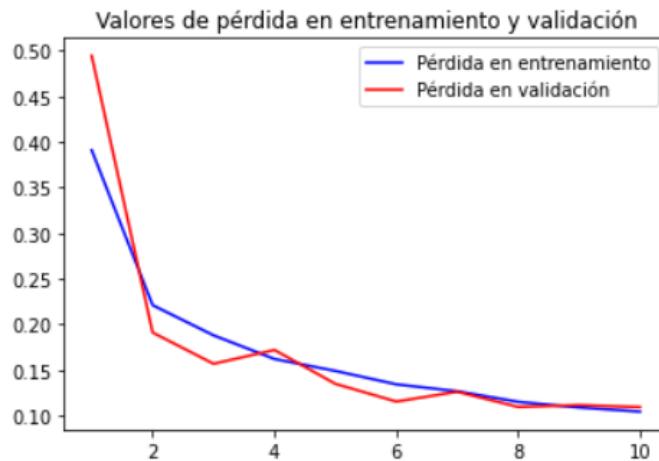


Figura 5.4: Valores de pérdida durante el entrenamiento.

Como podemos observar, nuestra red ha obtenido más del 95 % de aciertos durante el entrenamiento, lo que puede ser por una parte muy buena señal, puesto que se ha entrenado correctamente, con una curva de aprendizaje correcta, o puede ser una mala señal, porque se ha acercado considerablemente al 100 % de aciertos, síntoma de sobreentrenamiento.

Ahora comprobaremos con el conjunto de test si nuestra red ha sido sobreentrenada o ha conseguido ser un modelo tan robusto como aparenta en el entrenamiento.

Pruebas

Nuestro conjunto de pruebas posee un total de 1179 imágenes. Durante la etapa de pruebas realizaremos 20 repeticiones de todo el conjunto, y en cada repetición dividiremos el dataset en 50 batches.

Estos son los resultados:

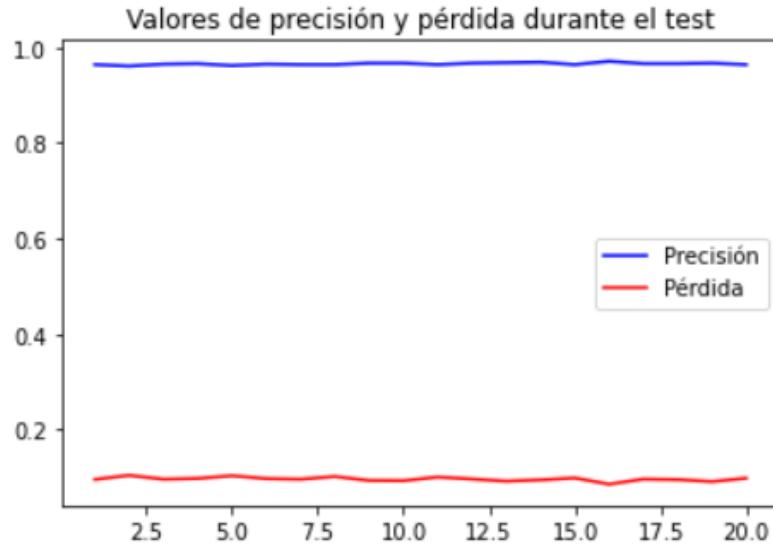


Figura 5.5: Valores de precisión y pérdida durante las pruebas.

Como podemos observar, en cada repetición hemos conseguido una precisión cercana a 1, siendo el valor medio 0.9650. Esto significa que hemos encontrado un modelo lo suficientemente bueno como para mantener un nivel de aciertos constante y alto. Este alto valor se puede deber también a la falta de imágenes, puesto este dataset puede ser muy similar en todo su conjunto, por lo que habría que realizar más pruebas con imágenes de otros dataset, o en su defecto, transformar las que ya poseemos.

De esta idea surge nuestra segunda red.

5.2.2. Segunda red. Con regularización y data augmentation

¿Qué es el data "augmentation"? es una técnica utilizada en el análisis de datos, la cual se usa para aumentar la cantidad de datos, añadiendo pequeñas modificaciones a los que ya tenemos. De esta manera nos aseguramos de que nuestros datos pierden similitud, provocando que a la red se le complique llegar al sobreentreno.

Aquí podemos observar un ejemplo de data augmentation que he utilizado para el entrenamiento de esta red:

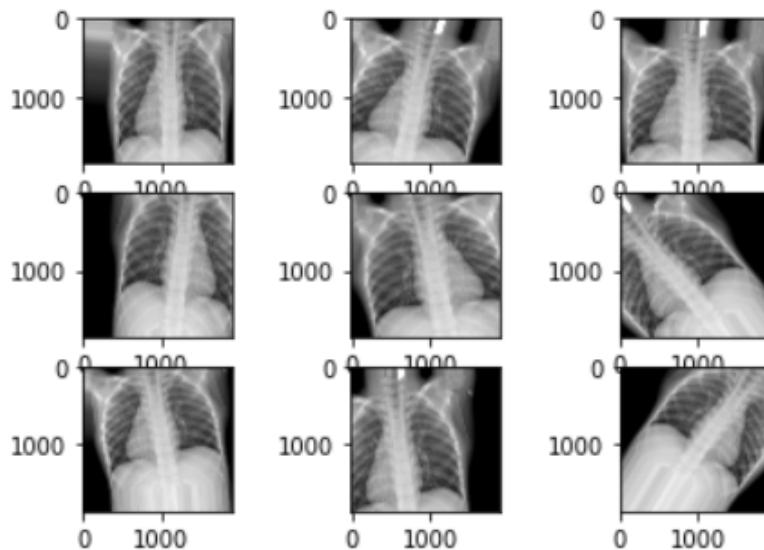


Figura 5.6: Ejemplo de data augmentation con radiografías.

Ahora proseguiremos con la estructura de nuestra red.

Estructura

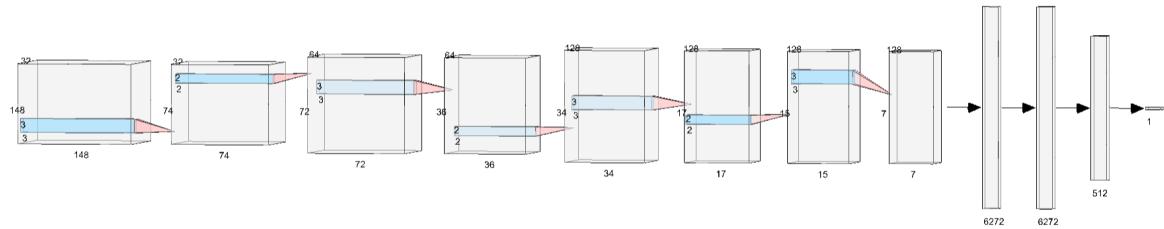


Figura 5.7: Visualización de la segunda red.

La única diferencia con la primera red es que entre las dos primeras capas totalmente conectadas nos encontramos con una capa de Dropout, añadiéndole más regularización a la red.

El sumario de esta red quedaría así:

Model: "sequential"		
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 148, 148, 32)	896
max_pooling2d (MaxPooling2D)	(None, 74, 74, 32)	0
conv2d_1 (Conv2D)	(None, 72, 72, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 36, 36, 64)	0
conv2d_2 (Conv2D)	(None, 34, 34, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 17, 17, 128)	0
conv2d_3 (Conv2D)	(None, 15, 15, 128)	147584
max_pooling2d_3 (MaxPooling2D)	(None, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dropout (Dropout)	(None, 6272)	0
dense (Dense)	(None, 512)	3211776
dense_1 (Dense)	(None, 1)	513
<hr/>		
Total params: 3,453,121		
Trainable params: 3,453,121		
Non-trainable params: 0		

Figura 5.8: Sumario de la estructura de la segunda red.

Como podemos observar, el número de parámetros a entrenar en esta segunda red sigue siendo el mismo que en la primera, pero comprobaremos ahora, con añadirle data augmentation y una capa de Dropout habremos añadido la suficiente regularización como para asegurarnos de no conseguir sobreentrenamiento.

Entrenamiento

Hay que tener en cuenta que la técnica de data augmentation **sólo** se debe de aplicar en el conjunto de entrenamiento, mientras que los de validación y pruebas no se deben de tocar.

Estas gráficas representan los valores de precisión y pérdida durante el entrenamiento:

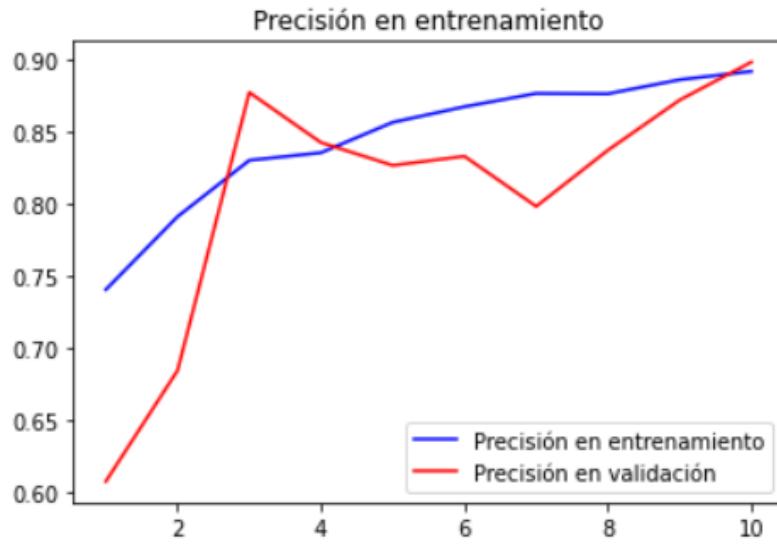


Figura 5.9: Valores de precisión durante el entrenamiento.



Figura 5.10: Valores de pérdida durante el entrenamiento.

Como podemos observar, los valores de precisión y pérdida en validación no son tan lineares, lo que demuestra que la regularización se está aplicando correctamente. Como podemos observar, la regularización provoca que no lleguemos a valores de precisión por encima del 90 %, obteniendo como valor más alto 0.8928.

Para esta red en concreto, 10 epoch parecen ser pocos, puesto que el aplicar data augmentation y una capa de Dropout es bastante regularización añadida.

Ahora comprobemos cómo se ha desenvuelto en el conjunto de pruebas:

Pruebas

Las pruebas se realizarán en las mismas condiciones que en la primera red. 1179 imágenes en 20 iteraciones con 50 batches por iteración:

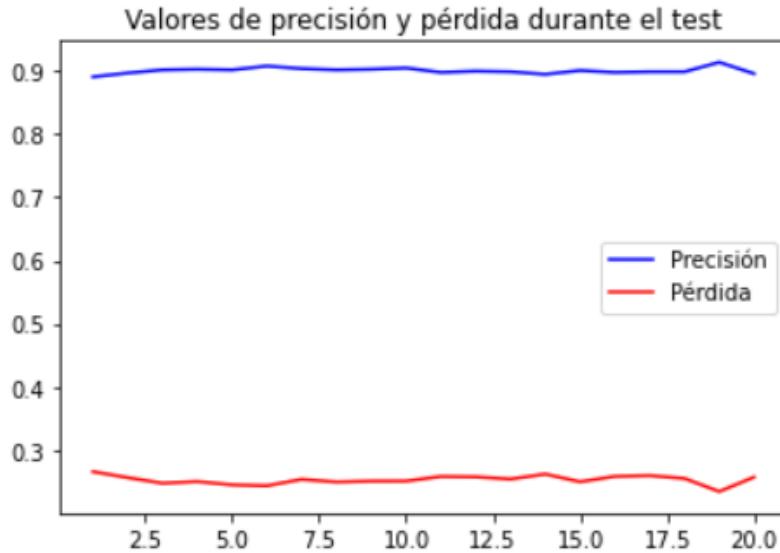


Figura 5.11: Valores de precisión y pérdida durante las pruebas.

Se puede observar que se ha conseguido un modelo estable, el cual llega a alcanzar durante bastantes iteraciones un poco más del 90 % de acierto. Debido a que la precisión se aleja lo suficiente del 100 %, podemos decir con seguridad que este modelo ha evitado el sobreentrenamiento, aunque podría mejorar la precisión con un poco más de entrenamiento.

5.2.3. Tercera red: VGG16 + Transfer Learning

Nuestra tercera red será la red VGG16 aplicándole Transfer Learning.

Estructura

Para ello tomaremos todas las capas de dicha red **menos** las capas totalmente conectadas.

En paralelo, creamos una red totalmente conectada, la cual conectaremos a la red VGG16 sin sus capas totalmente conectadas de primeras.

block3_conv1 (Conv2D)	(None, 37, 37, 256)	295168
block3_conv2 (Conv2D)	(None, 37, 37, 256)	590080
block3_conv3 (Conv2D)	(None, 37, 37, 256)	590080
block3_pool (MaxPooling2D)	(None, 18, 18, 256)	0
block4_conv1 (Conv2D)	(None, 18, 18, 512)	1180160
block4_conv2 (Conv2D)	(None, 18, 18, 512)	2359808
block4_conv3 (Conv2D)	(None, 18, 18, 512)	2359808
block4_pool (MaxPooling2D)	(None, 9, 9, 512)	0
block5_conv1 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv2 (Conv2D)	(None, 9, 9, 512)	2359808
block5_conv3 (Conv2D)	(None, 9, 9, 512)	2359808
block5_pool (MaxPooling2D)	(None, 4, 4, 512)	0
flatten (Flatten)	(None, 8192)	0
dense (Dense)	(None, 512)	4194816
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 1)	513
=====		
Total params: 18,910,017		
Trainable params: 6,555,137		
Non-trainable params: 12,354,880		

Figura 5.12: Sumario de la estructura de la tercera red.

Como podemos observar, nuestra red tiene 18.910.017 parámetros, de los cuales 6.555.137 podremos entrenar, lo cual aligerará la fase de entrenamiento y la carga computacional. Los parámetros ya entrenados fueron entrenados con el dataset de Imagenet.

En cuanto a la red adherida a VGG16, tiene esta estructura:

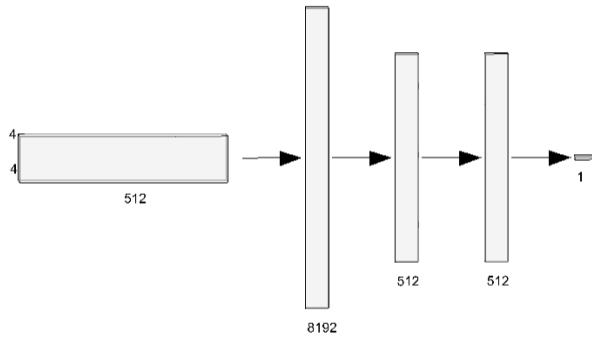


Figura 5.13: Red neuronal acoplada a VGG16.

La primera capa que podemos ver es la última propia de VGG16. La segunda es una capa que aplanará la red. Como la última capa de max pooling tenía una profundidad de 32, al añadirle la capa aplanadora esta también deberá tener una profundidad de 32. También le he añadido una capa de Dropout, puesto que al haber tantos parámetros entrenados me ha parecido necesario añadir un poco de regularización, aunque sea en las capas de la red creada por mí.

Entrenamiento

Esta red, al igual que la primera, ha sido entrenada con las 3784 imágenes de entrenamiento y las 947 de validación, con 10 epochs de 20 batches cada uno.

El resultado del entrenamiento es el siguiente, observando los valores de pérdida y precisión en cada epoch:

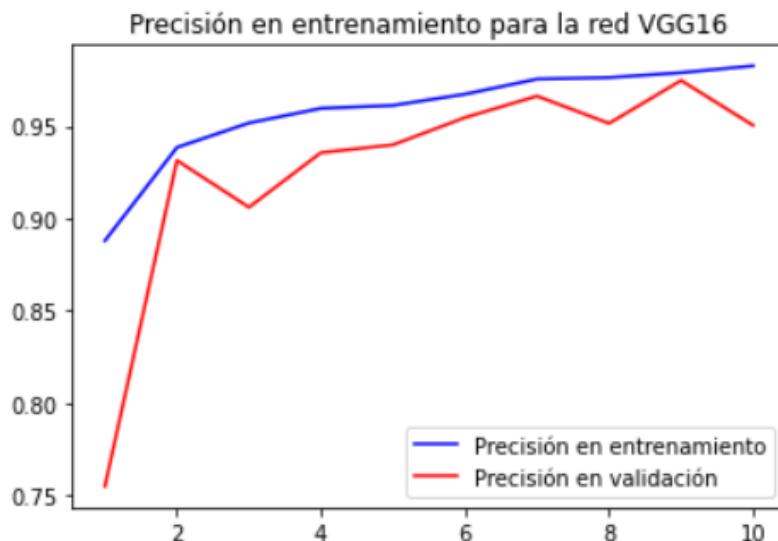


Figura 5.14: Valores de precisión durante el entrenamiento de VGG16.

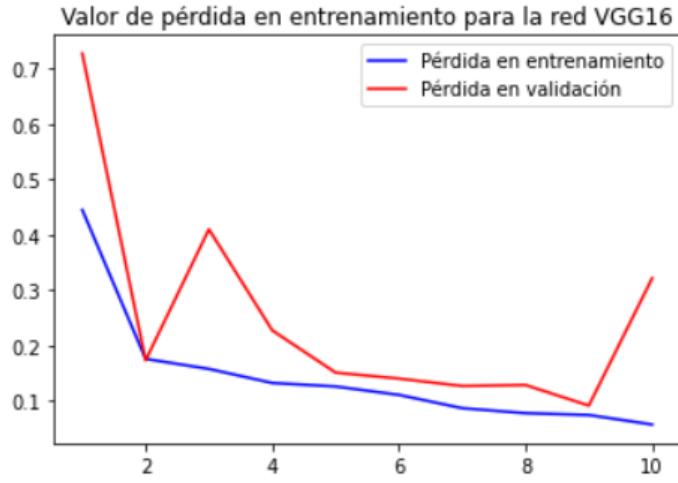


Figura 5.15: Valores de pérdida durante el entrenamiento de VGG16.

Podemos observar que el valor de precisión ha ido aumentando de forma constante durante el entrenamiento, alcanzando rápidamente un valor de 0.97, incluso llegando a alcanzar un 0.985 de precisión, por lo que podemos encontrarnos en un caso de sobreentrenamiento. Otro indicio es la rapidez en la convergencia, puesto que en 6 iteraciones ya había alcanzado un valor superior a 0.97.

Ahora comprobaremos con nuestro conjunto de pruebas la validez de este modelo.

5.2.4. Pruebas

Las predicciones que hemos realizado han sido sobre nuestro conjunto de test, el cual consta de 1179 imágenes, sobre las que iteraremos 20 veces, con 50 batches en cada iteración.

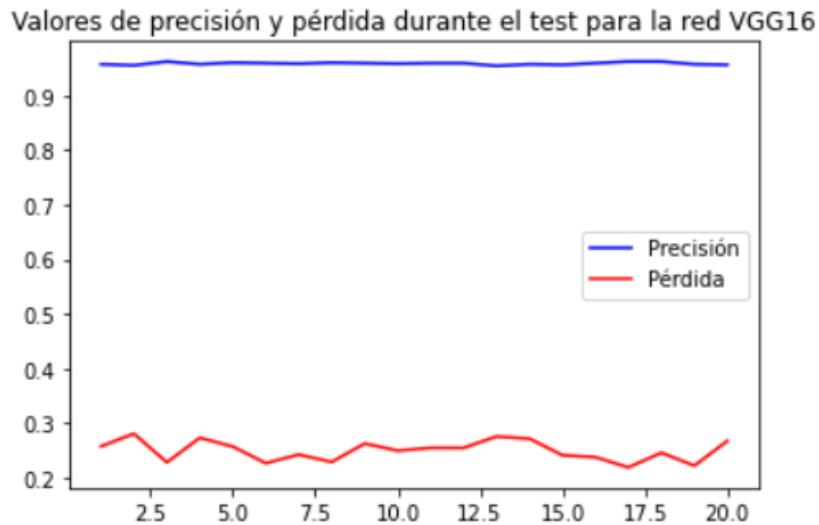


Figura 5.16: Valores de precisión y pérdida durante las pruebas de VGG16.

La red ha predicho correctamente la gran mayoría de imágenes del conjunto de

pruebas como hemos podido observar, obteniendo una precisión del 0.95 de manera constante. Aparentemente no ha habido problemas de sobreentreno, puesto que la red ha alcanzado unos valores de precisión óptimos y se han mantenido constantes sin importar cuantas veces se repitan las mismas predicciones.

5.2.5. Cuarta red: InceptionV3 + Transfer Learning

Nuestra cuarta red será la red Inception en su tercera versión, aplicándole Transfer Learning.

Estructura

Aplicándole la misma técnica de Transfer Learning que a la red VGG16, obtendremos esta estructura:

activation_91 (Activation)	(None, 3, 3, 384)	0	batch_normalization_91[0][0]
activation_92 (Activation)	(None, 3, 3, 384)	0	batch_normalization_92[0][0]
batch_normalization_93 (BatchNo)	(None, 3, 3, 192)	576	conv2d_97[0][0]
activation_85 (Activation)	(None, 3, 3, 320)	0	batch_normalization_85[0][0]
mixed9_1 (Concatenate)	(None, 3, 3, 768)	0	activation_87[0][0] activation_88[0][0]
concatenate_1 (Concatenate)	(None, 3, 3, 768)	0	activation_91[0][0] activation_92[0][0]
activation_93 (Activation)	(None, 3, 3, 192)	0	batch_normalization_93[0][0]
mixed10 (Concatenate)	(None, 3, 3, 2048)	0	activation_85[0][0] mixed9_1[0][0] concatenate_1[0][0] activation_93[0][0]
flatten_1 (Flatten)	(None, 18432)	0	mixed10[0][0]
dense_2 (Dense)	(None, 512)	9437696	flatten_1[0][0]
dropout_1 (Dropout)	(None, 512)	0	dense_2[0][0]
dense_3 (Dense)	(None, 1)	513	dropout_1[0][0]
=====			
Total params:	31,240,993		
Trainable params:	9,438,209		
Non-trainable params:	21,802,784		

Figura 5.17: Sumario de la estructura de la cuarta red.

Esta sin lugar a dudas es la red más larga de todas. Posee 31.240.993 de parámetros, de los cuales podremos entrenar la cantidad de 9.438.209. Para entender porqué la cantidad de parámetros a entrenar es tan grande primero debemos observar la estructura de la red acoplada a InceptionV3:

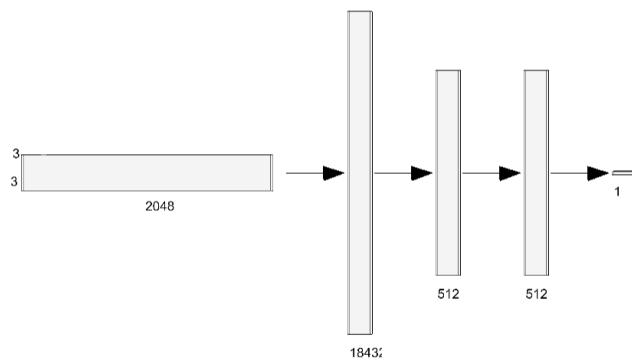


Figura 5.18: Red neuronal acoplada a InceptionV3.

La primera capa que podemos observar es la última capa no entrenable de InceptionV3. Al poseer unas dimensiones de (3,3,2048) la capa aplanadora realizará esta operación para poder conectarla a una capa totalmente conectada: $3 * 3 * 2048 = 18432$, provocando que la profundidad de la capa aplanadora sea de 18432. Esto provoca que en la capa totalmente conectada existan $18432 * 512$ conexiones entre la capa aplanadora y la capa actual, más $512 * 1$ conexiones entre la capa actual y la capa de Dropout. Además, es necesario añadir las 513 conexiones provocadas por la unión entre la capa de Dropout y la capa de salida de profundidad 1, al ser un clasificador binario.

La suma de todos estos parámetros sería: $18432 * 512 + 512 * 1 + 513 = 9,438,209$ parámetros a entrenar.

Por ello, el entrenamiento de InceptionV3 será más lento y no llegará a converger tan rápido.

5.2.6. Entrenamiento

El entrenamiento de esta red ha sido el mismo que el de las demás exceptuando la segunda red. Hacemos uso de las 3784 imágenes de entrenamiento y 947 de validación, con 10 epochs de 20 batches cada uno.

Las siguientes gráficas muestran los resultados en cada iteración, mostrando los valores de precisión y pérdida durante el entrenamiento:

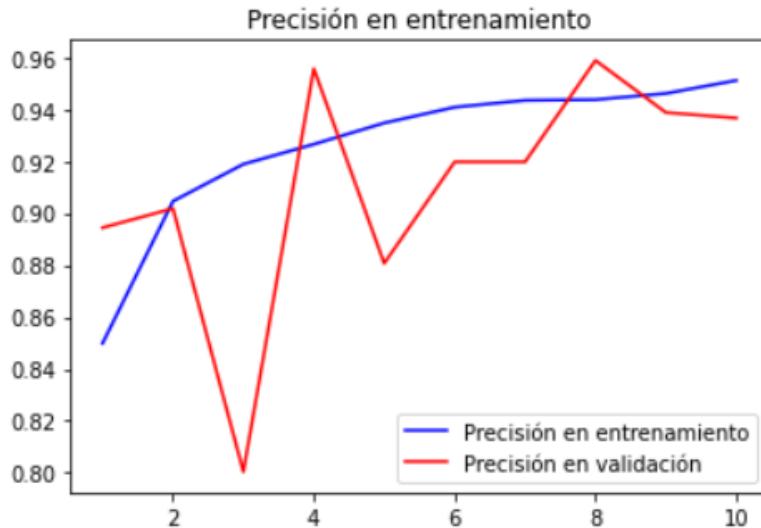


Figura 5.19: Valores de precisión durante el entrenamiento de InceptionV3.



Figura 5.20: Valores de pérdida durante el entrenamiento de InceptionV3.

Como advertimos anteriormente, el entrenamiento no ha convergido tan rápido debido al alto número de parámetros a entrenar. La precisión en validación lo confirma, siendo menos consistente en las primeras iteraciones y alineándose poco a poco con la precisión en entrenamiento.

Ahora comprobaremos que es lo suficientemente consistente como para ofrecer una buena predicción sobre el conjunto de test.

5.2.7. Pruebas

Ahora comprobaremos la eficacia del modelo. Para ello utilizaremos el conjunto de prueba, con 1179 imágenes y realizaremos la predicción 20 veces sobre todas ellas, con 50 batches por iteración.

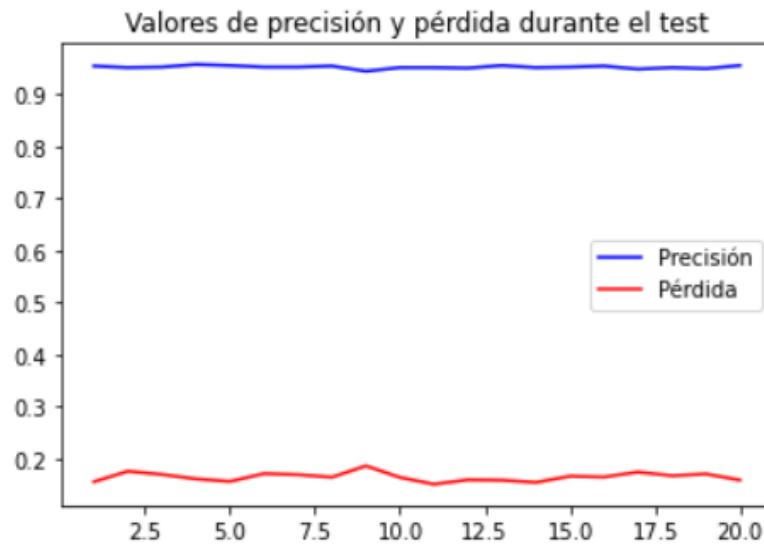


Figura 5.21: Valores de precisión y pérdida durante las pruebas de InceptionV3.

Como podemos observar, el modelo es lo suficientemente estable como para ser correcto. La precisión durante la mayoría de iteraciones ha rondado el 95 % de acierto, lo cual me ha sorprendido gratamente, al alcanzar menores valores de precisión durante el entrenamiento.

Aun así, 0.95 de precisión se acerca bastante a 1, por lo que habría que añadirle algo más de regularización.

5.2.8. Quinta red: ResNet + Transfer Learning

Nuestra última red será una combinación de ResNet aplicándole Transfer Learning.

Estructura

Estas son las últimas capas de lo que sería nuestra quinta red:

conv5_block3_1_bn	(BatchNormali	(None, 5, 5, 512)	2048	conv5_block3_1_conv[0][0]
conv5_block3_1_relu	(Activation	(None, 5, 5, 512)	0	conv5_block3_1_bn[0][0]
conv5_block3_2_conv	(Conv2D)	(None, 5, 5, 512)	2359808	conv5_block3_1_relu[0][0]
conv5_block3_2_bn	(BatchNormali	(None, 5, 5, 512)	2048	conv5_block3_2_conv[0][0]
conv5_block3_2_relu	(Activation	(None, 5, 5, 512)	0	conv5_block3_2_bn[0][0]
conv5_block3_3_conv	(Conv2D)	(None, 5, 5, 2048)	1050624	conv5_block3_2_relu[0][0]
conv5_block3_3_bn	(BatchNormali	(None, 5, 5, 2048)	8192	conv5_block3_3_conv[0][0]
conv5_block3_add	(Add)	(None, 5, 5, 2048)	0	conv5_block2_out[0][0] conv5_block3_3_bn[0][0]
conv5_block3_out	(Activation)	(None, 5, 5, 2048)	0	conv5_block3_add[0][0]
flatten_3	(Flatten)	(None, 51200)	0	conv5_block3_out[0][0]
dense_6	(Dense)	(None, 64)	3276864	flatten_3[0][0]
dropout_3	(Dropout)	(None, 64)	0	dense_6[0][0]
dense_7	(Dense)	(None, 1)	65	dropout_3[0][0]
<hr/>				
Total params: 26,864,641				
Trainable params: 3,276,929				
Non-trainable params: 23,587,712				

Figura 5.22: Sumario de la estructura de la quinta red.

Podemos observar que el total de parámetros asciende a 26.864.641, de los cuales 3.276.929 serán entrenables. De las tres redes que hemos elaborado con Transfer Learning, esta es la que menos parámetros tendremos que entrenar. Como en las dos anteriores redes, visualizaremos la red que acoplaremos a la red ya entrenada:

Dado que las dimensiones de la última capa propia de ResNet son (5,5,2048), la profundidad de la capa de aplanamiento deberá de ser de $5 * 5 * 2048 = 51200$. El número de conexiones entre la primera capa totalmente conectada y la capa aplanadora, 3.276.800, mas 64 conexiones entre la capa totalmente conectada y la capa de Dropout. Después se le añadirían las 64 conexiones entre la última capa y la capa de Dropout, mas la conexión de salida. Lo que en total haría 3.276.929 parámetros entrenables.

Ahora comprobaremos cómo ha rendido durante el entrenamiento:

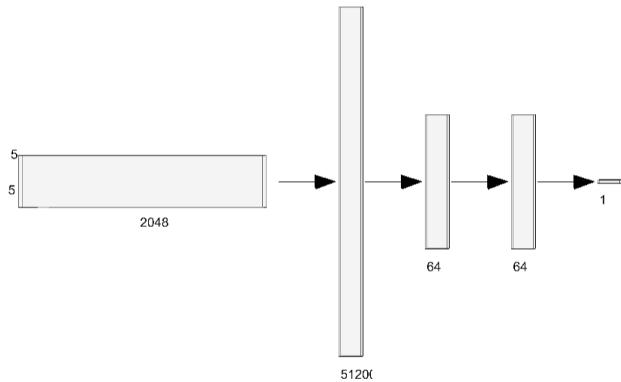


Figura 5.23: Red neuronal acoplada a ResNet.

Entrenamiento

Al igual que todas las demás redes exceptuando la segunda, hemos utilizado nuestro conjunto de entrenamiento de 3784 imágenes sin modificar y las 947 imágenes del conjunto de validación, con 10 epochs de 20 batches cada uno.

Mostraremos a continuación los valores de precisión y pérdida en cada iteración:

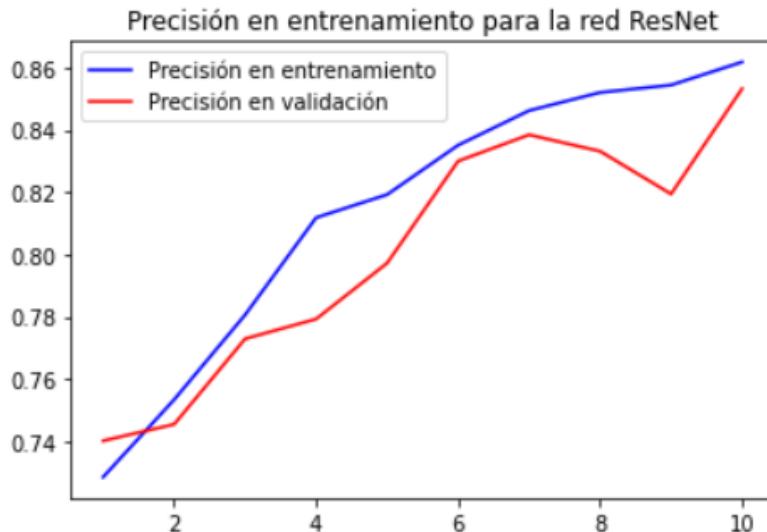


Figura 5.24: Valores de precisión durante el entrenamiento de ResNet.

Como podemos observar, con 10 epoch esta red no ha sido capaz de converger, puesto que la curva sigue siendo muy ascendente, aun siendo de las tres modeladas con Transfer Learning la que menos parámetros tiene. No obstante, ha sido constante en el aprendizaje hasta alcanzar un valor de precisión del 85 % en el último epoch.

Ahora realizaremos las pruebas pertinentes para poner a prueba el modelo.

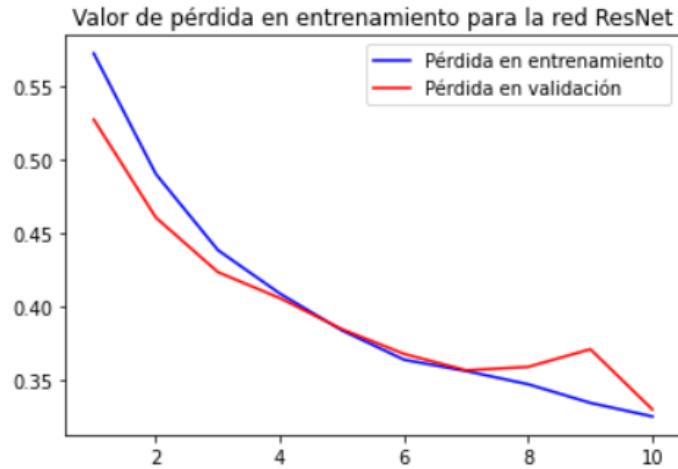


Figura 5.25: Valores de pérdida durante el entrenamiento de ResNet.

Pruebas

El conjunto de pruebas utilizado ha sido el mismo para todos los modelos, siendo este un conjunto formado por 1179 imágenes. Realizaremos 20 repeticiones sobre todo el conjunto, con 50 batches por iteración.

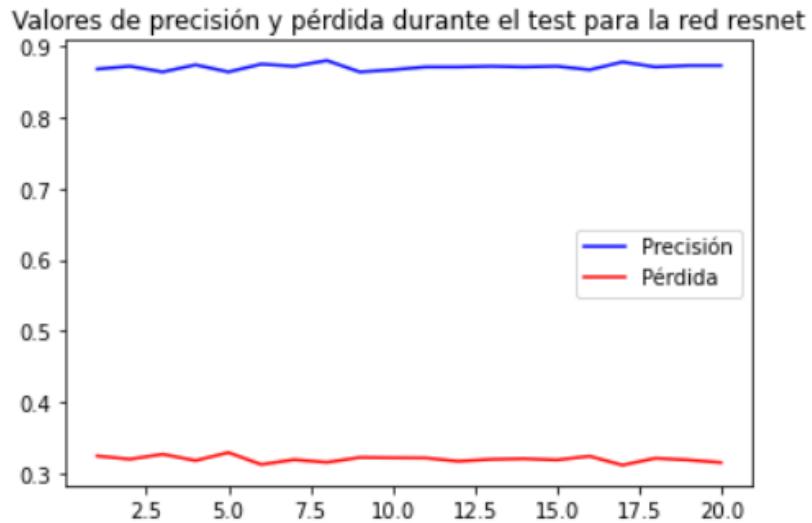


Figura 5.26: Valores de precisión y pérdida durante las pruebas de ResNet.

Se puede observar que los valores tanto de precisión como de pérdida han sido constantes a lo largo de las iteraciones, por lo que podemos decir que aunque el modelo no sea muy bueno (puesto que no llega al 90 % de acierto) es bastante fiable y constante.

A este modelo, según vimos en el entrenamiento le haría falta más epochs, puesto que todavía no ha conseguido converger y mejoraría los resultados en las pruebas.

5.3. Tercera parte: Aplicación Web

En esta sección mostraré la aplicación web realizada para este proyecto. Esta web realizará una predicción sobre una imagen que aportaría el usuario, pudiendo elegir cualquiera de las cinco redes elaboradas durante el proyecto.

Mostraré un diagrama de flujo de dicha aplicación:

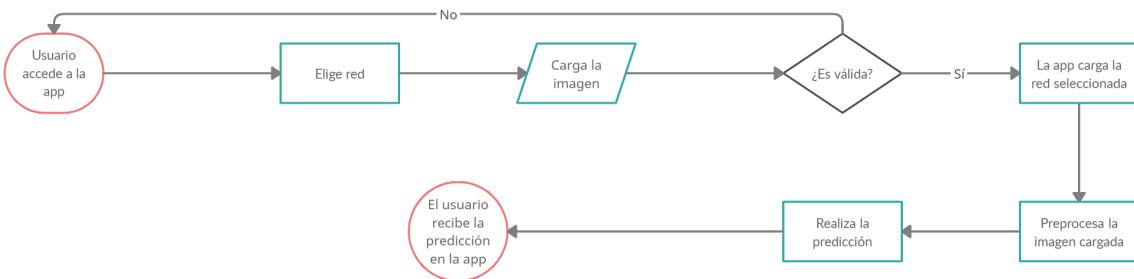


Figura 5.27: Diagrama de flujo de la aplicación web.

Primero, un usuario entraría en la aplicación web:

Figura 5.28: Primera vista de la aplicación web.

La estructura de la web conforma una cabecera, la cual pregunta por la red a utilizar, un pie en el que indico mi autoría de la aplicación y un formulario.

Dicho formulario consta de dos partes:

- La selección de una de las cinco redes anteriormente modeladas:

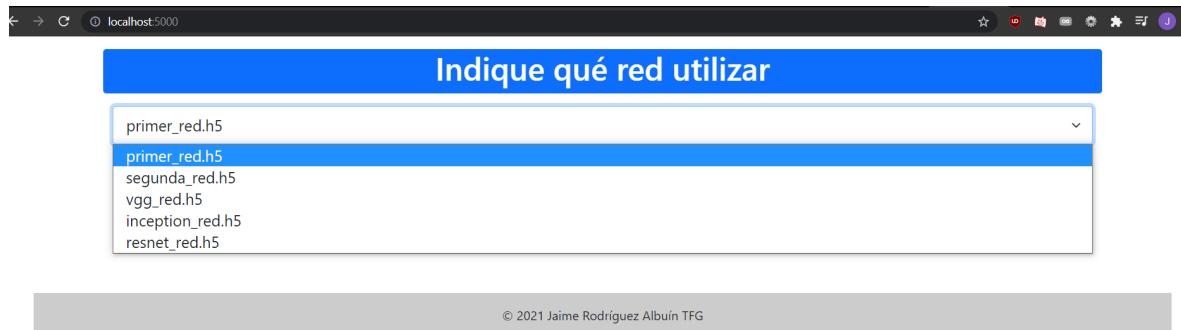


Figura 5.29: Selección con las cinco redes disponibles.

- La carga de una imagen desde el equipo local:

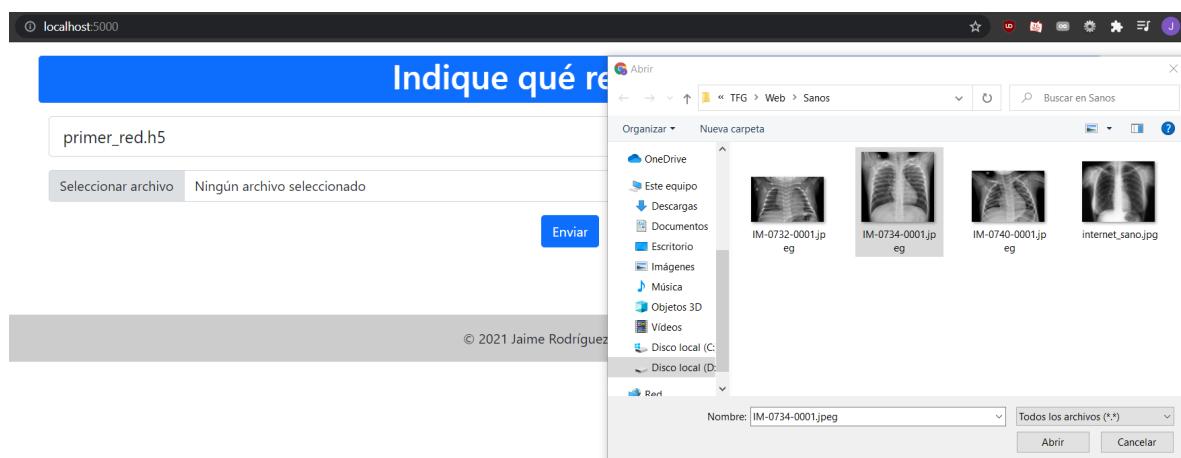
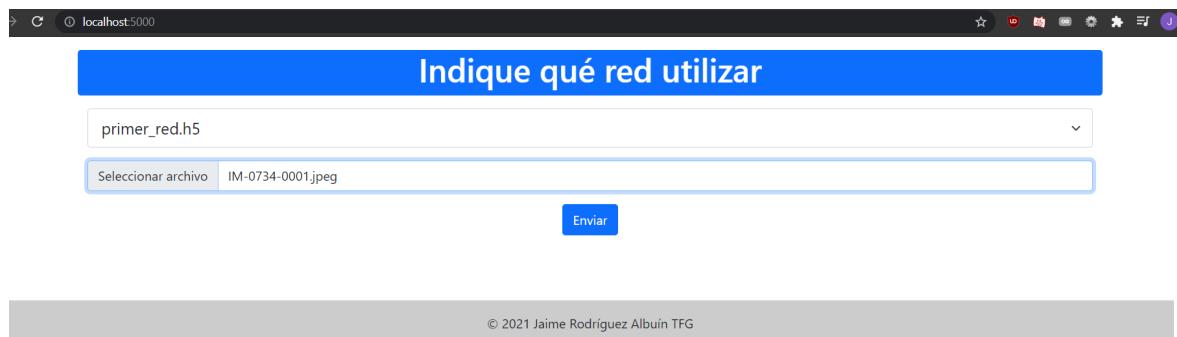


Figura 5.30: Carga de una imagen en la aplicación web.

Una vez todo seleccionado quedaría así dicho formulario:



The screenshot shows a browser window with the URL 'localhost:5000'. The main content is a form with a blue header bar containing the text 'Indique qué red utilizar'. Below the header is a dropdown menu with the value 'primer_red.h5'. Underneath it is a file input field showing the path 'Seleccionar archivo IM-0734-0001.jpeg'. At the bottom of the form is a blue 'Enviar' button. The footer of the page contains the copyright notice '© 2021 Jaime Rodríguez Albuín TFG'.

Figura 5.31: Formulario con la red seleccionada y la imagen cargada.

Lo siguiente sería pulsar el botón de enviar.

Una vez pulsado debemos esperar unos segundos a que la aplicación procese la imagen y la red seleccionada realice la predicción.

Una vez la aplicación termine devolverá las siguientes pantallas, dependiendo del resultado de la predicción.

- En caso de que la red indique que la radiografía es de una persona **sana**:

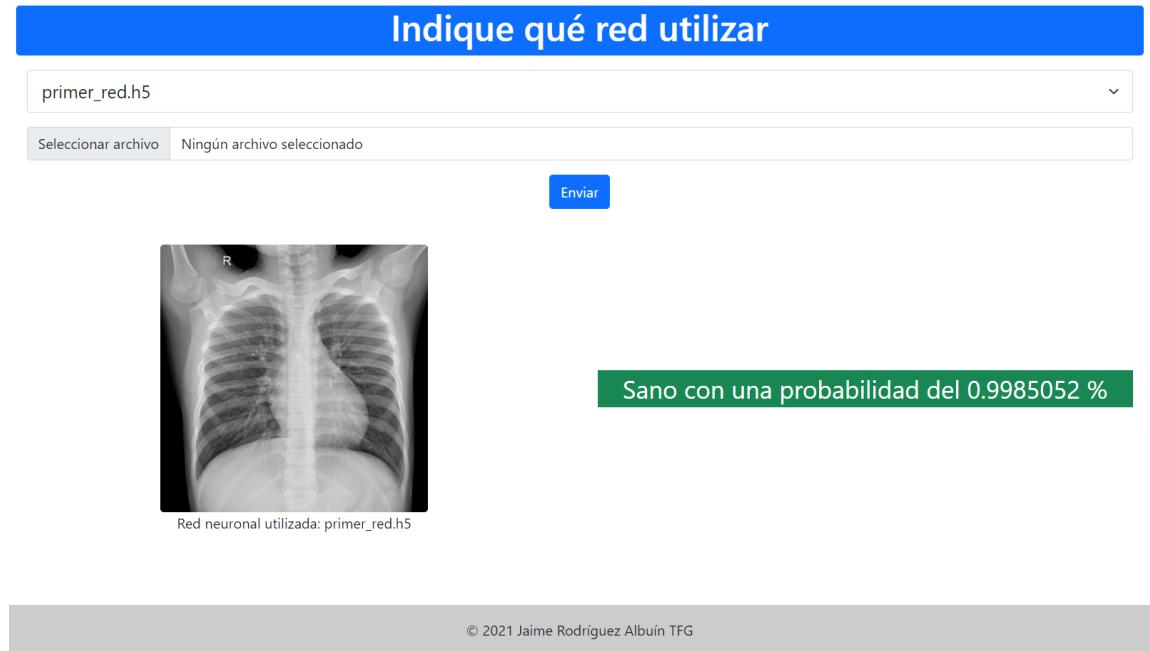


Figura 5.32: Resultado de una predicción con un paciente sano.

- En caso de que la red indique que la radiografía es de una persona **enferma**:

Indique qué red utilizar

primer_red.h5

Enviar

Seleccionar archivo Ningún archivo seleccionado



Enfermo con una probabilidad del 0.9628794 %

Red neuronal utilizada: primer_red.h5

© 2021 Jaime Rodríguez Albuín TFG

Figura 5.33: Resultado de una predicción con un paciente enfermo.

6. Manual de usuario

En este capítulo realizaremos una guía paso a paso, tanto para echar a andar la experimentación como para utilizar la aplicación web.

6.1. Experimentación

Antes de echar a correr

6.2. Página web

En este capítulo concluimos que...

7. Pruebas

7.1. Introducción

En este capítulo explicaremos...

7.2. Conclusiones

En este capítulo concluimos que...

8. Conclusiones

9. Bibliografía

- [1] Fernando Sancho Caparrini. Redes neuronales: una visión superficial, 2020. URL <https://www.cs.us.es/~fsancho/?e=72>.
- [2] François Chollet. Deep learning with python. 2018.
- [3] Nueva York. Cornell, universidad de Ithaca. Curso 1114, convolución., 2013. URL https://www.cs.cornell.edu/courses/cs1114/2013sp/sections/S06_convolution.pdf.
- [4] Universidad de Oregón. Transparencias sobre el perceptrón., 2020. URL <http://classes.engr.oregonstate.edu/eecs/spring2018/cs519-400/slides/week2-perceptron.pdf>.
- [5] Universidad de Sevilla. Página principal de la escuela técnica de ingeniería informática, 2020. URL <https://www.informatica.us.es>.
- [6] Universidad de Zagreb. Artificial neural networks, 2020. URL https://www.fer.unizg.hr/_download/repository/AI_12_ArtificialNeuralNetworks.pdf.
- [7] Deepai. Inception module., 2020. URL <https://deepai.org/machine-learning-glossary-and-terms/inception-module>.
- [8] Josh Patterson Adam Gibson. Deep learning: A practitioner's approach. 2017.
- [9] Google. Te damos la bienvenida a colaboratory., 2021. URL <https://colab.research.google.com/notebooks/welcome.ipynb?hl=es>.
- [10] Universidad de Birmingham John A. Bullinaria. Biological neurons and neural networks, artificial neurons, 2015. URL https://www.cs.bham.ac.uk/~pxt/NC/12_JB.pdf.
- [11] Keras. Documentación de keras., 2021. URL <https://keras.io/api/#models-api>.
- [12] Sung Kim. Filtros convolucionales en matlab., 2013. URL <http://kiwi.bridgeport.edu/cpeg585/ConvolutionFiltersInMatlab.pdf>.
- [13] Neurohive. Vgg16 - convolutional network for classification and detection., 2018. URL <https://neurohive.io/en/popular-networks/vgg16/>.
- [14] NumPy. Documentación de numpy., 2021. URL <https://numpy.org/doc/>.
- [15] OS. Documentación de os., 2021. URL <https://docs.python.org/3/library/os.html>.
- [16] PIL. Documentación de pillow., 2021. URL <https://pillow.readthedocs.io/en/latest/index.html>.

- [17] Python. Página web de python., 2021. URL <https://www.python.org/>.
- [18] Sebastian Ruder. An overview of gradient descent optimization algorithms., 2017. URL <https://arxiv.org/pdf/1609.04747.pdf>.
- [19] Wikipedia. Stochastic gradient descent., 2021. URL https://en.wikipedia.org/wiki/Stochastic_gradient_descent#AdaGrad.
- [20] Wikipedia. Entrada de wikipedia sobre la entropía cruzada., 2021. URL https://en.wikipedia.org/wiki/Cross_entropy.
- [21] Wikipedia. Data augmentation., 2021. URL https://en.wikipedia.org/wiki/Data_augmentation.
- [22] Wikipedia. Dimensionality reduction, 2021. URL https://en.wikipedia.org/wiki/Dimensionality_reduction.
- [23] Wikipedia. Residual neural network., 2021. URL https://en.wikipedia.org/wiki/Residual_neural_network.
- [24] Wikipedia. Supervised learning, 2021. URL [https://en.wikipedia.org/wiki/Supervised_learning#:~:text=Supervised%20learning%20\(SL\)%20is%20the,a%20set%20of%20training%20examples](https://en.wikipedia.org/wiki/Supervised_learning#:~:text=Supervised%20learning%20(SL)%20is%20the,a%20set%20of%20training%20examples).