Minería de datos

MÁSTER UNIVERSITARIO EN INVESTIGACIÓN EN INTELIGENCIA ARTIFICIAL

Práctica 3: Aprendizaje Profundo

Jon Etxeberria San Millán

jecheverr33@alumno.uned.es

Curso: 2021-2022

Práctica 03

Contenido

1. Introducción y contextualización ... 3
1.1. Inicio Redes Convolucionales o CNN ... 4
1.2. Evolución de las CNN ... 4
1.3. Funcionamiento de las CNN ... 5
1.3.1. Pixeles y Neuronas ... 5
1.3.2. Preprocesamiento ... 5
1.3.3. Convoluciones ... 6
1.3.4. La función de Activación ... 6
1.3.5. Muestreo (subsampling) ... 7
1.3.6. Repetición de convoluciones ... 8
1.3.7. Conectar con una red neuronal "tradicional". ... 8
1.4. Entrenamiento de las redes tradicionales ... 9
1.4.1. Backpropagation ... 9
1.4.2. Descenso del gradiente ... 9
1.5. Ajuste de los modelos: underfitting, overfitting y bien ajustado ... 9
1.6. Herramientas y ajuste de parámetros para la resolución de la práctica ... 10
2. Descripción de los modelos redes diseñadas ... 10
2.1 Arquitectura de red ... 10
2.1.1 Modelo 1 ... 10
2.1.2 Modelo 2 ... 11
2.1.3 Modelo 3 ... 11
2.1.4 Modelo 4. Mejora del Modelo 3 ... 11
2.2 Ajustes de red ... 13
3. Análisis crítico de los resultados ... 13
3.1. Análisis general y de la tabla ... 14
3.2. Análisis de la gráfica ... 15
3.2.1. Accuracy (exactitud) ... 15
3.2.2. Función de pérdida (Loss function) ... 16
4. Conclusiones ... 18
Referencias ... 19

1. Introducción y contextualización

El objetivo de esta actividad es que el alumnado ponga en práctica los contenidos teóricos relativos a las redes neuronales y el aprendizaje profundo para aplicados a un dataset de granos de polen de Nueva Zelanda. Se trata de un dataset con dos variedades diferentes muy parecidas entre sí, prácticamente indistinguibles según los palinólogos. Según estos, son granos de polen monádicos, isopolares, angulaperturados, sincolpados, tectados y triangulares en vista polar.

El Deep Learning (DL) o Aprendizaje Profundo, es una variedad de Aprendizaje Automático que se basa en imitar la estructura del cerebro humano y sus redes neuronales. Las redes neuronales artificiales están especialmente indicadas para problemas complejos en los que existen multitud de casos particulares. Los algoritmos de DL se pueden describir como instancias particulares de una receta bastante simple, aplicadas para resolver un problema concreto de aprendizaje automático:

- Recopilar un conjunto de datos asociado al problema (enorme, si es posible).
- Diseñar una función de coste apropiada para el problema, también conocida como función de pérdida [loss function].
- Seleccionar un modelo de red neuronal y establecer sus hiperparámetros (tamaño, características...).
- Aplicar un algoritmo de optimización para minimizar la función de coste ajustando los parámetros de la red.

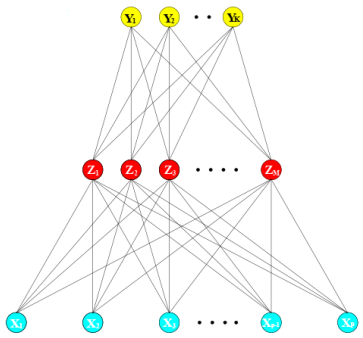


Ilustración 1: Esquema de una red neuronal feed-forward de una sola capa oculta (obtenida de [1]*)*

La gran diferencia entre el DL y el resto de las técnicas de aprendizaje es la capacidad de abstracción. Su capacidad de formar jerarquías de conceptos utilizando múltiples niveles de representación. La abstracción es la forma en que los humanos tratan con la complejidad del mundo que nos rodea. Las técnicas de DL pretenden automatizar ese proceso de abstracción, creando nuevas abstracciones sobre otras ya existentes. La topología más habitual de las redes neuronales está formada por múltiples capas, cada una de las cuales recibe como entrada la salida de la capa anterior. Las técnicas de Deep Learning parten de la idea base de que, si somos capaces de aprender con éxito múltiples niveles de representación, podremos generalizar correctamente.

Dentro de las redes neuronales hay 3 tipos que son los más usados:

1. Convolutional Neural Networks (CNN, redes neuronales convolucionales)

   Son redes neuronales artificiales que han sido diseñadas para procesar matrices estructuradas, como imágenes. Es decir, se encargan de clasificar imágenes basándose en los patrones y objetos que aparecen en ellas, por ejemplo, líneas, círculos o, incluso, ojos y caras.

   Normalmente las CNN suelen usarse en computer vision (visión artificial), ya que pueden operar con imágenes brutas (raw images) y no necesitan hacer un procesamiento previo. Esta cualidad hace que sean muy útiles para aplicaciones visuales de clasificación de imágenes, pero también procesamiento del lenguaje natural (natural language processing, NLP), ayudando con la clasificación de textos. Por ejemplo, con el procesamiento del lenguaje natural, se puede identificar si el email que se ha recibido es para finanzas, para comercial o para otro departamento.

Las redes neuronales convolucionales usan varias capas convolucionales (sobre unas 20-30 capas de media). Las capas convolucionales son un tipo especial de capa que, al apilarse unas encimas de otras, permite reconocer formas más sofisticadas.

2. Recurrent Neural Networks (RNN, redes neuronales recurrentes)

Las redes neuronales recurrentes usan datos secuenciales o datos de series de tiempo. Este tipo de redes solucionan problemas ordinales o temporales, como la traducción de idiomas, reconocimiento de voz (speech recognition), procesamiento de lenguaje natural (NLP, Natural Language Processing) y captura de imágenes. Por eso, estas redes se encuentran en tecnologías como Siri o Google translate.

Las redes neuronales recurrentes difieren de otras redes neuronales artificiales en que tienen "memoria". Es decir, las RNN toman información de inputs anteriores para influenciar los inputs y outputs actuales. Por ejemplo, al escribir con el móvil, el teclado te muestra una serie de palabras como sugerencias a partir de lo que está escrito. Esas sugerencias son las predicciones que se han hecho basándose en los caracteres que fueron escritos con anterioridad.

3. Generative Adversarial Networks (GAN, redes generativas antagónicas)

Este tipo de redes consisten en usar 2 redes neuronales artificiales y oponerlas la una a la otra (por eso se les conoce como antagónicas) para generar nuevo contenido o datos sintéticos que pueden hacerse pasar por reales. Una de las redes genera y la otra funciona como "discriminadora". La red discriminatoria (también conocida como red antagónica) ha sido entrenada para reconocer contenido real y hace de censor para que la red que genera contenido haga contenido que parezca real.

Para nuestra Práctica, se utilizarán las redes Convolucionales o CNN, ya que el objetivo del trabajo es clasificar imágenes diferenciándolas en dos clases distintas.

1.1. Inicio Redes Convolucionales o CNN

Los orígenes de las redes convolutivas actuales hay que buscarlos en los modelos de redes neuronales artificiales que se propusieron, ya en los años 70, basándose en los estudios de David Hubel y Torsten Wiesel sobre el córtex visual. En la misma década en la que Hubel y Wiesel publicaron su modelo sobre la posible organización del córtex visual, un investigador japonés, Kuhiniko Fukushima, propuso un modelo de red neuronal artificial claramente bioinspirado: el cognitrón[2]. Unos años después, publicaría una versión mejorada de su modelo: el neocognitrón., que estaba formada por varias capas y realizaba reconocimiento de caracteres escritos a mano[3]. La primera red convolucional como tal se diseñó el año 1988 y se llamó LeNet5[4]. Se trataba de una arquitectura sencilla de 7 capas, dedicada a reconocer números manuscritos en los cheques de algunos bancos de Estados Unidos. Es a partir de ese momento cuando empiezan a proliferar los estudios de este tipo de redes, y en 2014 cuando de la mano de la existencia de grandes bases de datos de imágenes, y de los avances tecnológicos en computación, se convirtieron en un referente en el campo de la clasificación de imágenes gracias a sus buenos resultados.

1.2. Evolución de las CNN

Para este tipo de redes, existe un desafío anual ImageNet Large Scale Visual Recognition (ILSVRC) (J. Deng et al., 2009). En este desafío se presentan trabajos de todo el mundo, donde compiten por ver quién presenta el mejor y más preciso modelo para la clasificación de imágenes y detección de objetos sobre el conjunto de datos ImageNet[5]. De este modo, anualmente surgen nuevas redes que ofrecen mejores resultados o son mejoradas algunas ya existentes al modificar los parámetros. En la siguiente imagen se muestra la evolución de las redes aplicadas al dataset propuesto por el desafío.

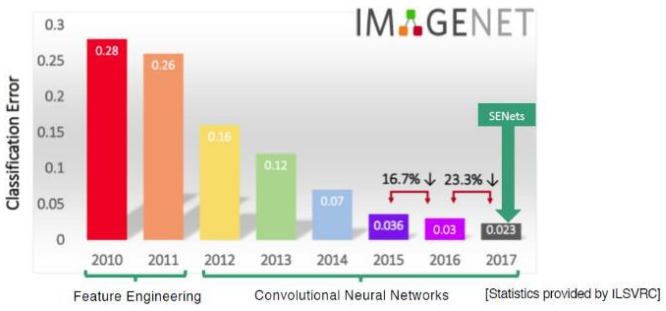


Ilustración 2: Evolución de la tasa de error en el desafío de Imagenet

Gracias a este desafío, han surgido numerosos algoritmos que han sido incorporados las librerías de programación para su aplicación sencilla por los programadores en general. Entre las redes destacan:

- **LeNet**: Desarrollada por Yann LeCun y sus colaboradores, fueron capaces de implementar una red neuronal convolucional capaz de detectar caracteres haciendo uso de los conceptos de backpropagation y feedforward [6].
- **AlexNet**: AlexNet [7] ha tenido un gran impacto en el campo de Machine Learning, y especialmente en la aplicación del aprendizaje profundo al machine vision. Tiene una arquitectura muy similar a LeNet, pero se trata de una red más profunda, con más filtros por capa y con capas convolucionales apiladas.
- **VGG**: Esta red [8] fue desarrollada por el Visual Geometry Group de la universidad de Oxford. En general existen varias redes bajo este nombre que difieren en el número de capas que poseen. Se trata de una red similar a AlexNet, pero con solo capas de convolución 3x3 y numerosos filtros.
- **GoogleNet:** GoogleNet[9] está compuesta por 22 capas y casi 12 veces menos parámetros que por ejemplo AlexNet, lo cual la hace mucho más rápida y precisa. GoogleNet se compone de sus inception layers o capas de inicio. La idea principal de éstas, es cubrir un área mayor pero también preservar una buena resolución para una pequeña muestra de información sobre las imágenes.
- **ResNet:** Las residual networks o redes residuales [10] son capaces de aprender funciones más complejas y consecuentemente conducir a un mejor rendimiento. Sin embargo, en ocasiones agregar más capas tuvo eventualmente un efecto negativo en el rendimiento final. Este fenómeno se conoce como el problema de degradación, aludiendo al hecho de que, si bien las mejores técnicas de inicialización de parámetros y la normalización de lotes permiten que las redes más profundas converjan, en ocasiones convergen a tasas de error más altas que las menos profundas.
- **DenseNet:** Las redes de convolución densas [10] pueden ser útiles para referenciar mapas futuros desde el inicio de la red. Así, cada capa del mapa de características está concatenada con la entrada de cada capa sucesiva dentro de un bloque denso. Esto permite que las capas posteriores dentro de la red aprovechen directamente las características de las capas anteriores fomentando la reutilización de características dentro de la red.

1.3. Funcionamiento de las CNN

1.3.1. Pixeles y Neuronas

Para comenzar, la red toma como entrada los pixeles de una imagen. Si tenemos una imagen con apenas 28×28 pixeles de alto y ancho, eso equivale a 784 neuronas. Y eso es si sólo tenemos 1 color (escala de grises). Si tuviéramos una imagen a color, necesitaríamos 3 canales (red, green, blue) y entonces usaríamos 28x28x3 = 2352 neuronas de entrada. Esa es nuestra capa de entrada. Para continuar con el ejemplo, supondremos que utilizamos la imagen con 1 sólo color.

1.3.2. Preprocesamiento

Antes de alimentar la red, conviene normalizar los valores. Los colores de los pixeles tienen valores que van de 0 a 255, se hará una transformación de cada pixel: "valor/255" y quedará siempre un valor entre 0 y 1.

1.3.3. Convoluciones

Una convolución consiste en tomar grupos de píxeles cercanos de la imagen de entrada e ir operando matemáticamente contra una pequeña matriz a la que se denomina *kernel* conocida como filtro. Ese kernel recorre todas las neuronas de entrada – de izquierda a derecha y de arriba hacia abajo – y genera una nueva matriz de salida, que será la nueva capa de neuronas ocultas, y que también se conoce como la matriz de activación.

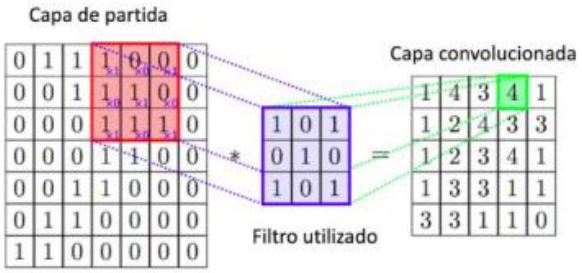


Ilustración 3: Uso de filtro (kernel) en la convolución(https://www.codificandobits.com/blog/convolucion-redes-convolucionales/)

Hay que comentar que durante las convoluciones no se aplica un solo filtro si no que habrá un conjunto de estos. Si por ejemplo se escogen 32 filtros, se obtienen en total 32 matrices de salida. A este conjunto se le denomina feature mapping .

1.3.4. La función de Activación

El siguiente paso es aplicar la función de activación a la capa convolucionada. Para las capas de convolución, la función más utilizada es la llamada ReLu por Rectifier Linear Unit y consiste en una función f(x)=max(0,x). Para la última capa de unión con la capa de salida, se utiliza normalmente la función Softmax. Principales funciones de activación utilizadas en DL:

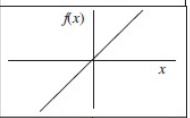
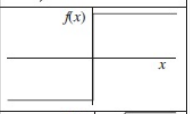
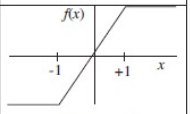
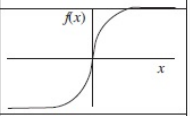
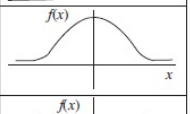
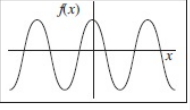
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty,+\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1,+1\}$ $\{0,+1\}$	
Lineal a tramos	$y = \begin{cases} -1, si\ x < -l \\ x, si\ +l \le x \le -l \\ +1, si\ x > +l \end{cases}$	$[-1,+1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0,+1]$ $[-1,+1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$	
Sinusidal	$y = A\ sen(\omega x + \varphi)$	$[-1,+1]$	

Ilustración 4: Funciones de activación(obtenido de [11])

- **Función activación softmax**

La función softmax es una generalización de la regresión logística que puede ser aplicada a datos continuos. Soporta sistemas de clasificación multinomial, por lo que se convierte en el recurso principal utilizado en las capas de salida de un clasificador. Esta función de activación devuelve la distribución de

probabilidad de cada una de las clases soportadas en el modelo. La función Softmax calcula la distribución de probabilidades del evento sobre 'n' eventos diferentes. En términos generales, esta función calculará las probabilidades de cada clase objetivo sobre todas las clases objetivo posibles. Más tarde, las probabilidades calculadas serán útiles para determinar la clase objetivo para las entradas dadas.

$$softmax(zi) = \frac{\exp(zi)}{\sum \exp(zj)}$$

La principal ventaja de usar Softmax es el rango de probabilidades de salida. El rango será de 0 a 1, y la suma de todas las probabilidades será igual a uno. Si la función softmax utilizada para el modelo de clasificación múltiple devuelve las probabilidades de cada clase y la clase objetivo tendrá una probabilidad alta.

- **Función activación ReLU**

La función ReLU transforma los valores introducidos anulando los valores negativos y dejando los positivos tal y como entran.

Esta función generará una salida igual a cero cuando la entrada (z) sea negativa, y una salida igual a la entrada cuando dicha esta última sea positiva.

La función de activación ReLU se ha convertido en la más usada en los modelos Deep Learning durante los últimos años, lo cual se debe principalmente a:

- La no existencia de saturación, como sí ocurre en las funciones sigmoidal y tanh. Lo anterior hace que el algoritmo del gradiente descendente converja mucho más rápidamente, facilitando así el entrenamiento.
- Es más fácil de implementar computacionalmente en comparación con las otras dos funciones, que requieren el cálculo de funciones matemáticas más complejas como la exponencial.

Se recomienda hacer uso de esta función de activación ReLU en las capas ocultas de la Red Neuronal. Esto hace que el entrenamiento sea más rápido y es mucho más probable que logre la convergencia del algoritmo del gradiente descendente.

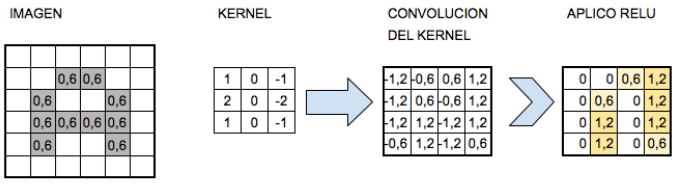


Ilustración 5: La imagen realiza una convolución con un kernel y aplica la función de activación, en este caso ReLu. Finalmente se obtiene un mapa de detección de características (obtenido de https://www.juanbarrios.com/redes-neurales-convolucionales/)

1.3.5. Muestreo (subsampling)

En este paso se toma una muestra de las neuronas más representativas antes de hacer una nueva convolución. Con el ejemplo del primer paso, tomando la imagen en blanco y negro de 28x28 píxeles 28x28pixels se tiene una primera capa de entrada de 784 neuronas y luego de la primer convolución obtenemos una capa oculta de 25.088 neuronas -que realmente son nuestros 32 mapas de características de 28×28 ((28 x 28) x 32). Si se hiciera una nueva convolución a partir de esta capa, el número de neuronas de la próxima capa requeriría un poder computacional importante. Por ello y para reducir el tamaño de la próxima capa de neuronas hacemos un muestreo preservando las características más importantes que detectó cada filtro.

Hay diversos tipos de subsampling, pero el más utilizado es el *Max-Pooling*. Aplicando un "Max-pooling" con un tamaño de 2×2, significa que recorreremos cada una de las 32 imágenes de características obtenidas anteriormente de 28x28px de izquierda-derecha, arriba-abajo, pero en vez de tomar de a 1 pixel, tomaremos de «2×2» (2 de alto por 2 de ancho = 4 pixeles) e iremos preservando el valor «más alto» de

entre esos 4 pixeles (por eso lo de «Max»). En este caso, usando 2×2, la imagen resultante es reducida «a la mitad»y quedará de 14×14 pixeles.

Después de aplicar esta técnica quedarán 32 imágenes de 14×14, pasando de 25.088 neuronas a 6272, reduciendo drásticamente el número de neuronas y almacenando todas las características de la imagen original.

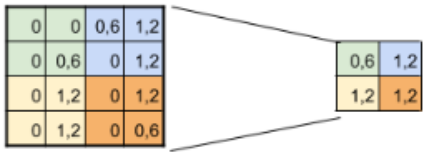


Ilustración 6: Subsampling. Aplicación de Max-Pooling de 2x2 reduciendo la salida a la mitad(obtenido de https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/)

1.3.6. Repetición de convoluciones

Una vez aplicado el pooling a la imagen, se repite este proceso hasta obtener la reducción que se estima oportuna en función de rendimiento y eficiencia.

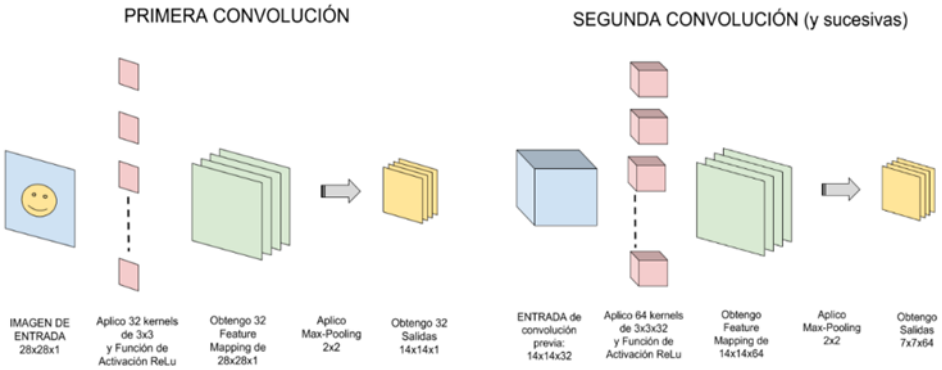


Ilustración 7: Proceso de aplicación sucesiva de convoluciones

1.3.7. Conectar con una red neuronal "tradicional".

Como punto de finalización se toma la última capa oculta a la que se le realizó el proceso de subsampling, que se dice que es tridimensional y se procesa para que deje de serlo pasando a ser una capa de neuronas tradicionales. Entonces a esta última se le aplica la función de activación Softmax que conecta contra la capa de salida final que tendrá la cantidad de neuronas correspondientes con las clases que estamos clasificando.

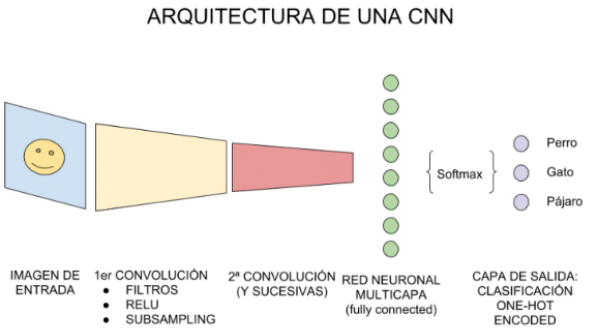


Ilustración 8: Conexión de red convolucional con una red tradicional

Las salidas al momento del entrenamiento tendrán el formato conocido como "one-hot-encoding" en el que para perros y gatos será: [1,0] y [0,1], para coches, aviones o barcos será [1,0,0]; [0,1,0];[0,0,1]. La función Softmax se encarga de trasladar dichos valores a probabilidad a las neuronas de salida.

1.4. Entrenamiento de las redes tradicionales

1.4.1. Backpropagation

Este método de entrenamiento[12] consiste en un método de cálculo que emplea un ciclo de propagación. Una vez aplicado un patrón a la entrada de la red como estímulo, este se propaga desde la primera capa a través de las capas siguientes de la red hasta generar una salida. La señal de salida se compara con la salida esperada y se calcula una señal de error para cada una de las salidas.

En las CNN, se debe ajustar el valor de los pesos de los distintos kernels. Esto es una gran ventaja al momento del aprendizaje pues como se constató cada kernel es de un tamaño reducido, en nuestro ejemplo en la primera convolución es de tamaño de 3×3, eso son sólo 9 parámetros que debemos ajustar en 32 filtros dan un total de 288 parámetros. En comparación con los pesos entre dos capas de neuronas "tradicionales": una de 748 y otra de 6272 en donde están todas interconectarlas con todas y eso equivaldría a tener que entrenar y ajustar más de 4,5 millones de pesos (sólo para 1 capa).

1.4.2. Descenso del gradiente

El descenso del gradiente es el método estándar utilizado para minimizar la función de pérdida o error, esto es, reducir la diferencia entre el resultado obtenido y el que se busca obtener. En este algoritmo los pesos sinápticos de la red son modificados hasta que dicha función alcanza un mínimo. El gradiente es la generalización vectorial de la derivada, es un vector de tantas dimensiones como la función y cada dimensión contiene la derivada parcial en dicha dimensión:

$$\nabla f = [\partial f / \partial x_1, \partial f / \partial x_2, \dots \partial f / \partial x_n] \quad \text{(Función 1)}$$

El gradiente ∇f_x es el vector que contiene la información de cuanto crece la función en un punto especifico x por cada dimensión de nuestra función de forma independiente. Lo siguiente una vez conocida la dirección de descenso, es determinar el tamaño de paso. Para poder controlar la proporción del paso utilizaremos un parámetro k de tal forma que el paso a dar se cuantifica: $-k\nabla$. k es un parámetro de tunning del método "gradient descent" al que se denomina "learning rate". Dada una posición x se puede avanzar a una nueva posición que dependerá del paso que demos, tamaño y dirección:

$$x' = x - k\nabla f(x) \quad \text{(Función 2)}$$

1.5. Ajuste de los modelos: underfitting, overfitting y bien ajustado

Las principales causas al obtener malos resultados en Machine Learning son el *overfitting* o el *underfitting* de los datos. Cuando entrenamos nuestro modelo intentamos "hacer encajar" -fit en inglés- los datos de entrada entre ellos y con la salida.

Si los datos de entrenamiento son muy pocos la máquina no será capaz de generalizar el conocimiento y estará incurriendo en *underfitting*. El modelo no es capaz de extraer conocimiento y obtener un rendimiento aceptable en el dataset de entrenamiento, y también en los datos de validación.

Por el contrario, el *overfitting*, se produce cuando el modelo se aprende los datos de train perfectamente(demasiado bien), con buen resultado en el entrenamiento, pero mal o menor rendimiento con los datos de validación.

Se dice que un modelo está *bien ajustado* cuando aprende bien los datos de entrenamiento, y además obtiene buenos resultados al generalizar los datos al entrenamiento y a la validación.

Ilustración 9: Gráficos representativos de los 3 posibles ajustes del modelo (obtenido de https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/)

1.6. Herramientas y ajuste de parámetros para la resolución de la práctica

Para la resolución de esta práctica se han seguido la directrices marcadas, realizando el desarrollo en Python aplicando las librerías de Keras[13] y TensorFlow[14]. Se han utilizado también otras librerías de Python, Pandas especializada en el manejo y análisis de estructuras de datos; Numpy especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos. Sklearn es un conjunto de rutinas escritas en Python para hacer análisis predictivo, que incluyen clasificadores, ...

Se ha utilizado Google Colab como entorno de programación, así como Kaggle Notebook en determinado momentos para pruebas específicas. Al principio intenté trabajar Enel IDE Pycharm, pero tuve bastante problemas con la descarga e instalación de las librerías necesarias para realizar la práctica.

Además he leído en profundidad la teoría propuesta para poder interiorizarla ([15], [1]). Gracias a [15] he podido profundizar y comprender con base matemática la parte aplicación de la técnica de pooling a un dataset de imágenes. En [1] también se aprende la base matemática que está detrás del diseño de las redes neuronales y el funcionamiento de las mismas.

Respecto a la calibración de los algoritmos y las modificaciones de los hierparámetros aplicados a cada modelos, se explica en detalle en la página 10 en el punto “Descripción de modelos”. .

2. Descripción de los modelos redes diseñadas

2.1 Arquitectura de red

2.1.1 Modelo 1

Tal y como se pide en el enunciado, “por una red neuronal sin capa oculta, equivalente a una regresión binomial logística, que reciba las imágenes y las clasifique”. Se trata de la red más simple, desde la cuál iniciar las pruebas de añadir capas y hacer la comparación de mejoras de rendimiento. En esta estructura, la primera capa recibe las entradas de imágenes de polen, para en la segunda capa definir si se trata de un grano de polen de Tipo A o de Tipo B.

Para la primera capa se ha elegido una “flatten”, que recibe la entradas de imágenes. La segunda es de tipo “Dense” que es la encargada de volver el Tipo de grano de polen gracias a la función de activación de tipo “Softmax”, que devuelve un arreglo de dos posibilidades que entre ambos suman 1. Así, cada uno de los nodos, contienen una aproximación de la probabilidad de que la imagen con la que se está trabajando pertenezca a una de las dos clases de polen. Las características de la Red se presentan en la siguiente *Tabla 1*: Características de la arquitectura aplicada al Modelo 1.

Tabla 1: Características de la arquitectura aplicada al Modelo 1

Layer (Type)	Output Shape	Parameters	Activación
Flatten	(None, 30000)	0	
Dense	(None, 2)	60002	softmax

2.1.2 Modelo 2

Para el Segundo modelo el enunciado pide que sea igual que el modelo 1 pero con una o más capas ocultas. Bajo estas premisas se ha diseñado una red de 2 capas ocultas además de las 2 capas que ya tenía el Modelo 1 . La primera capa es una capa "flatten" igual que en el Modelo 1. La primera capa oculta que recibe las imágenes de la capa "flatten", es una capa "dense" de 400 unidades y función de activación "ReLu". La segunda capa oculta de la Red es también de tipo "dense", con 100 unidades, y función de activación "ReLu". La última y cuarta capa es idéntica a la del primer modelo. Se trata de una capa "Dense" con 2 unidades de salida. El diseño de la red se muestra en la *Tabla* 2: Características de la arquitectura aplicada al Modelo 2.

Tabla 2: Características de la arquitectura aplicada al Modelo 2

Layer (Type)	Output Shape	Parameters	Activación
Flatten	(None, 30000)	0	
Dense	(None, 400)	12000400	ReLu
Dense	(None, 100)	40100	ReLu
Dense	(None, 2)	202	softmax

2.1.3 Modelo 3

Para el Modelo 3 ya se pide una Red Convolucional propiamente. Hasta ahora no se habían utilizado capas de convolución. En este caso, la primera capa es una capa "conv2D". Los parámetros aplicados a esta capa son: para los filtros se ha seleccionado 16, tamaño de kernel de 3x3 y un paso en el avance de (1,1). En la salida de esta capa se encuentra una capa "flatten" que se usa para "aplanar" la entrada, es decir, unidimensionalizar la entrada multidimensional. Así, las salidas se conectan a otra capa oculta de tipo "Dense" de 128 unidades y función de activación de tipo "ReLu". La capa de salida se trata de una capa "Dense" con función de activación "softmax" y 2 unidades de salida. El diseño de la red puede observarse en la *Tabla 3*: Características de la arquitectura aplicada al Modelo 3

Tabla 3: Características de la arquitectura aplicada al Modelo 3

Layer (Type)	Output Shape	Parameters	Activación
Conv2D	(None, 100, 100, 16)	0	ReLu
Flatten	(None, 16000)	0	
Dense	(None, 128)	20480128	ReLu
Dense	(None, 2)	258	softmax

2.1.4 Modelo 4. Mejora del Modelo 3

Tomando como base el Modelo 3, se han realizado pruebas modificando los parámetros de la red diseñada. Los parámetros modificados y probados, han sido : "keras.layers.Conv2D(***filtros***, ***kernel_size***, ***strides =(1, 1)***":

- Filtros: Se trata del número de filtros de los que aprenderán las capas convolucionales. Se han probado los parámetros [8,16,32,64].
- Tamaño del kernel (kernel_size): Se han probado las dimensiones comunes que incluyen "1 × 1, 3 × 3, 5 × 5 y 7 × 7", y se p asan como tuplas.
- Zancadas: Número entero o tupla / lista de 2 enteros, que especifica el "paso" de la convolución junto con la altura y el ancho del volumen de entrada. Se han probado los valores [(1,1), (2,2), (3,3)].

Además de los parámetros se han hecho pruebas añadiendo más capas de convolución. El diseño final seleccionado consta de 2 capas de convolución "Conv2D", a las que se han añadido a cada una capa de pooling "MaxPooling2D", y también capas de "Dropout" del 15% para solucionar el overfitting.

El diseño de la red final, ha sido la de utilizar los parámetros: 16, 32 para los filtros en las capas de convolución, con un tamaño de kernel de (3x3) para ambas y una zancada de (1,1). Las capas resultantes pueden verse en la siguiente *Tabla 4: Características de la arquitectura aplicada al Modelo 4 (Modelo 3 mejorado)*.

Tabla 4: Características de la arquitectura aplicada al Modelo 4 (Modelo 3 mejorado)

Layer (Type)	Output Shape	Parameters	Activación
Conv2D	(None, 100, 100, 16)	448	ReLu
MaxPooling2D	(None, 50, 50, 16)	0	
Dropout	(None, 50, 50, 16)	0	
Conv2D	(None, 50, 50, 32)	4640	ReLu
MaxPooling2D	(None, 25, 25, 32)	0	
Dropout	(None, 25, 25, 32)	0	
Flatten	(None, 20000)	0	
Dense	(None, 64)	1280064	ReLu
Dense	(None, 2)	130	softmax

2.1.5 Modelo 5. Red preentrenada VGG16

El modelo VGG es una red neuronal convolucional propuesta por K. Simonyan y A. Zisserman, de la Universidad de Oxford, que adquirió notoriedad al ganar el Desafío de Reconocimiento Visual a Gran Escala de ImageNet (ILSVRC) en 2014. El modelo alcanzó una precisión del 92,7% en Imagenet, que es una de las puntuaciones más altas logradas. ImageNet es una enorme base de datos con más de 14 millones de imágenes etiquetadas en más de 1000 clases, a partir de 2014.

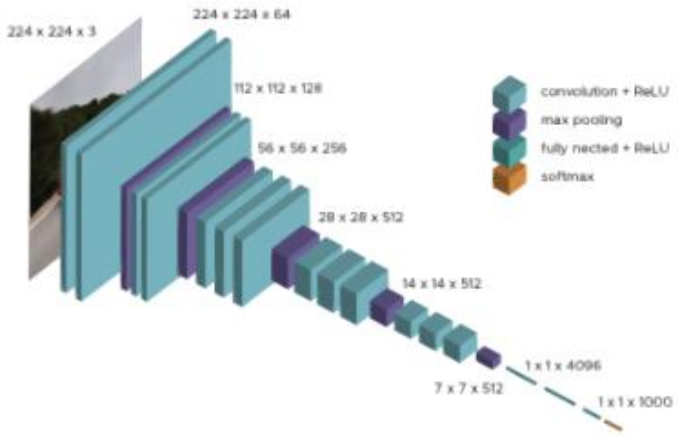


Ilustración 10: Estructura del algoritmo VGG16 (obtenido de https://datascientest.com/es/vgg-que-es-este-modelo-daniel-te-lo-cuenta-todo)

El modelo sólo requiere un preprocesamiento específico que consiste en restar a cada píxel el valor RGB medio, calculado en el conjunto de entrenamiento. Durante el entrenamiento del modelo, el input a la primera capa de convolución es una imagen RGB de tamaño 224 x 224. Para todas las capas de convolución, el núcleo de convolución es de tamaño 3×3: la dimensión más pequeña para capturar las nociones de arriba, abajo, izquierda/derecha y centro.

Tabla 5: Características de la arquitectura aplicada al Modelo VGG16

Layer (Type)	Output Shape	Parameters	Activación
VGG16	(None, 3, 3, 512)	14714688	ReLu
GlobalAveragePooling2D	(None, 512)	0	
Dense	(None, 2)	1026	softmax

2.2 Ajustes de red

Antes de entrenar los modelos diseñados, hay que definir una serie de ajustes para el entrenamiento. En los ajustes a través de la función compile(), hay que definir una serie de hiperparámetros:

```
# Compilar el modelo
modelo3.compile(optimizer='adam',
                loss='sparse_categorical_crossentropy',
                metrics=['accuracy'])
```

Ilustración 11: función compile() para ajustar el modelo con 3 parámetros a seleccionar

Como se observa en *Ilustración 11*, son 3 los hiperparámetros que hay que seleccionar en el ajuste del modelo.

Como optimizador se ha utilizado ***Adam***, que realmente se llama Adaptive Moment Estimation (Estimación Adaptativa de Momentos), el cual es, en pocas palabras, RMSProp con momentum. Adam se da cuenta de los beneficios de AdaGrad y RMSProp. Adam no solo calcula la tasa de aprendizaje de parámetros adaptativos en función del valor medio del primer momento como el algoritmo RMSProp, sino que también hace un uso completo del valor medio del segundo momento del gradiente (es decir, la varianza no centrada). Adam es un optimizador nuevo(presentado por Diederik Kingma de OpenAI y Jimmy Ba de la Universidad de Toronto en su artículo de ICLR de 2015), construido en base a sus predecesores, por tanto, podremos esperar que su rendimiento sea superior. Según sus autores, este optimizador presenta varias ventajas frente al resto: fácil de implementar, computacionalmente eficiente, tiene pequeños requisitos de memoria, cambio de escala invariante a diagonal de los gradientes, muy adecuado para problemas que son grandes en términos de datos y / o parámetros, apropiado para objetivos no estacionarios, apropiado para problemas con pendientes muy ruidosas o escasas, los hiperparámetros tienen una interpretación intuitiva y normalmente requieren pocos ajustes. Por estas ventajas se ha seleccionado este optimizador.

El hiperparámetro seleccionado para la función de pérdida ha sido ***sparse categorical crossentropy***. La selección de este hiperparámetro se ha realizado en base a que sólo hay una clasificación de 2 tipo de imágenes excluyentes una con la otra. Una imagen (tipo de polen) sólo puede estar en una categoría. Así, se puede acelerar la ejecución del entrenamiento y ahorrar memoria. La función "los" evalúa la desviación entre las predicciones realizadas por la red neuronal y los valores reales de las observaciones utilizadas durante el aprendizaje. Se ha descartado el "categorical crossentropy", porque: se utiliza "sparse categorical crossentropy" cuando las clases se excluyen mutuamente (por ejemplo, cuando cada muestra pertenece exactamente a una clase) y la "categorical crossentropy" cuando una muestra puede tener varias clases o las etiquetas son probabilidades blandas (como [0.5, 0.3, 0.2]).

Para la métrica se ha seleccionado ***Accuracy(Exactitud)***. A través de este hiperparámetro se mide el porcentaje de casos que el modelo ha acertado. Esta métrica crea dos variables locales, total y count , que se utilizan para calcular la frecuencia con la que y_pred coincide con y_true . Esta frecuencia se devuelve finalmente como binary accuracy : una operación idempotente que simplemente divide el total por el count . En términos estadísticos, la exactitud está relacionada con el sesgo de una estimación. También se conoce como Verdadero Positivo (o "True positive rate").

3. Análisis crítico de los resultados

Una vez diseñadas las arquitecturas y los hiperparámeteros de entrenamiento , se analizan los resultados. Se han diseñado 4 modelos de redes, tal y como se piden en el enunciado. El primer modelo, una red neuronal sin capa oculta, que equivale a la regresión binomial logística. El segundo modelo, añade 2 capas ocultas a la arquitectura. El tercero ya incorpora una capa convolucional como entrada, y el cuarto consiste en aplicar más capas convolucionales, así como la mejora de hiperparámetros y capas de "pooling y dropout".

El entrenamiento de las redes se ha realizado con 50 épocas (epochs) y se ha utilizado el método "callback" de la librería de Keras para almacenar los mejores resultados obtenidos para le modelo entrenado.

3.1. Análisis general y de la tabla

A la hora de realizar las pruebas hay varios elementos que se han observado. El tiempo de entrenamiento necesario para las redes está muy relacionado con la arquitectura del modelo, y el número de capas que poseen. Cuanto más capas tiene un modelo, más tarde en tiempo de ejecución el entrenamiento y más memoria RAM usa en realizar el entrenamiento completo.

Aunque se ha trabajado con Google Colab en la nube, el consumo de memoria RAM de modelos con varias capas como por ejemplo el Modelo 4 se eleva con respecto a los anteriores, y su tiempo de entrenamiento es mucho más elevado, llegando a la media hora de tiempo necesario. Sin embargo, el primer modelo no tarda más de 5-8 minutos.

Los resultados obtenidos para los diferentes modelos diseñados se presentan en la siguiente *Tabla 6 .*

Tabla 6: Tabla de resultados obtenidos con las redes CNN

Modelo	Arquitectura de Red	Pesos entrenables	Pesos totales	Train acc.	Valid. acc.	Test acc.
Red1	Flatten + Dense	60.002	60.002	90,42	81,77	86,25
Red2	Flatten + 3*(Dense)	12.040.702	12.040.702	100	93,27	93,3333
Red3	Conv2D + Flatten + 2*(Dense)	20.480.834	20.480.834	100	93,71	93,89
Red4 (mejora de red 3)	2*(Conv2D + MaxPooling2D + Dropout)+ Flatten + 2*(Dense)	1.285.282	1.285.282	99,17	95,31	94,58
Red VGG16	VGG16 + GlobalAveragePooling2D + Dense	1.026	14.715.714	89,79	89,06	88,74

De la *Tabla 6:* Tabla de resultados obtenidos con las redes CNN puede observarse que los valores de Training son siempre superiores a los de la validación y test en todos los modelos. Se trata de un indicador de que puede haber "overfitting". Según puede observarse en los resultados, la incorporación de Redes de Convolución mejora el resultado de las pruebas, algo que eras esperable ya que el dataset consta de imágenes como datos. Aun así, la Red2 ha obtenido muy buenos resultados, por encima del 90% en validación, y además con poca exigencia computacional. Entre la Red3 y Red4, se observa que las mejoras aplicadas por la Red4 hacen que su rendimiento obtenido en las pruebas sea superior, y además parece que empieza a reducir el overfitting que se ha detectado en la experimentación. El modelo VGG16, aunque no ha alcanzado los valores de validación y test de otros modelos, ha hecho un entrenamiento sin overfitting con una gráfica en donde los resultados de entrenamiento y validación están totalmente alineados.

3.2. Análisis de la gráfica

3.2.1. Accuracy (exactitud)

Observando las gráficas en ***¡Error! No se encuentra el origen de la referencia.***, se puede comprobar el fenómeno de overfitting. Por el comportamiento general de estas gráficas se observa que son el característico cuando un modelo presenta overfitting. Por un lado, la exactitud de los datos de entrenamiento aumenta linealmente con las epochs, hasta alcanzar casi el 100%, mientras que la exactitud de los datos de validación se detiene alrededor del 80-90% y a partir de aquí se mantiene constante a lo largo de las epochs.

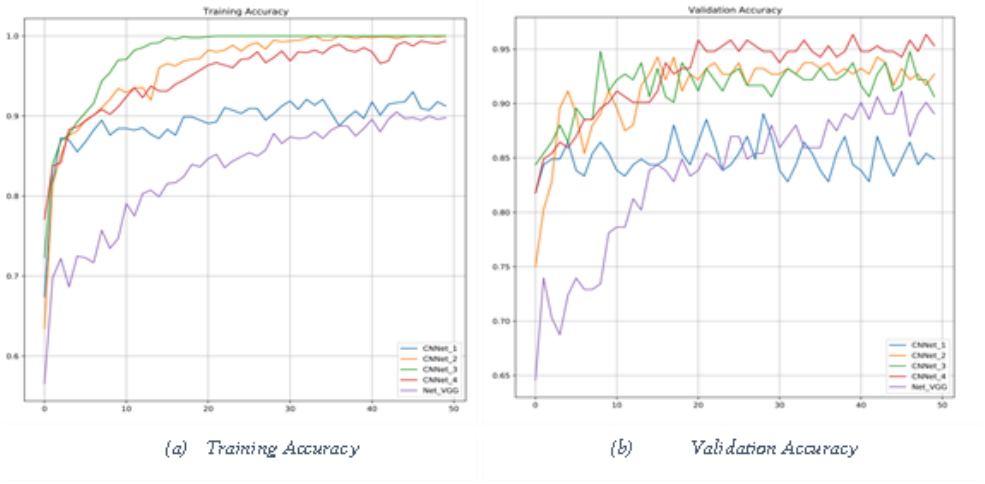


Ilustración 12: Training y Validation Accuracy. Gráficas obtenidas de la exactitud de los modelos

Se puede ver como a partir de la epoch 15, la exactitud de training ya empiezas a acercarse a 100%, y por otro lado la validación empieza a estancarse. Así, podemos entender que debemos combatir el overfitting del modelo aplicando alguna de las técnicas existentes para ello: i) uso de modelos más simples ii) regularización iii) Remover "features" irrelevantes iv) detener el entrenamiento antes de tiempo. El modelo 4 sí que aplica alguna de estas técnicas, como es el Dropout, y parece que el overfitting se presenta más tarde, sobre el epoch 30.

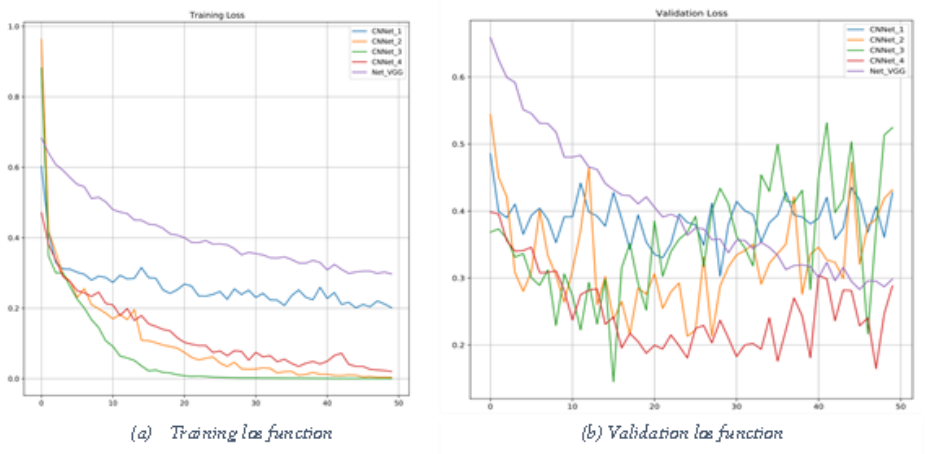
A pesar de la existencia de overfitting, los valores obtenidos en la validación son buenos, ya que quitando el Modelo1, el resto todos están por encima del 90% de exactitud.

Destaca la irregularidad de los resultados de la gráfica de validación. El Modelo 1 destaca sobre el resto teniendo una variación de resultados entre epochs muy alta, y además sin lograr mejorar manteniéndose estacionario. Esto suele suceder cuando el modelo cree haber encontrado un óptimo global, y sin embargo se encuentra en un mínimo local. Normalmente esto se debe a que el gradiente es 0, y ya no se puede salir aplicando el gradiente, se debe aplicar el hiperparámetro momentum que busca el mínimo global.

El modelo VGG16 a pesar de no haber llegado a los valores tan altos como otros modelos, alcanza ya 90% en alguna época y si hubiera estado aplicando la función earlyStopping se hubiera detenido en ese valor. Sin embargo, al querer hacer una gráfica comparativa de todas se ha dejado que siga hasta la época 50. Sin embargo, puede observarse como aún mantiene una tendencia al alza, y seguramente con más épocas, hubiera superado los valores del resto, que ya no incrementan sus resultados.

3.2.2. Función de pérdida (Loss function)

Según los resultados que pueden observarse en ***¡Error! No se encuentra el origen de la referencia.***, la menor pérdida se consigue con la CNNet3, que además es la que más pendiente pronunciada adquiere en el Training. La CNNet4 (mejora de CNNet3) tiene una pendiente menos pronunciada que la 3, y en la validación tarda más epochs en perder el ajuste óptimo. Igual que se observaba en las funciones de exactitud, se puede observar el overfitting a partir de 10-20 epochs.



(a) Training loss function *(b) Validation loss function*

Ilustración 13: Training y Validation Loss functions

La CNNet1 es la que obtiene peores resultados, tanto en entrenamiento como en validación. Se intuye que pueda sufrir de underfitting. No son resultados sorprendentes ya que la arquitectura de la red es muy simple, equivaliendo a una simple regresión binomial logística.

La CNNet2 tiene un comportamiento similar a las CNNet3 y 4, con una pendiente pronunciada acercándose a 0 en Testing, y overfitting en validación a partir de las epochs 10-20.

Hay que comentar que en la validación las gráficas presentan la misma irregularidad que en exactitud. Observando las gráficas, la CNNet2 es la que más variabilidad presenta entre epochs de resultados.

Ya que el modelo VGG16 no presenta el problema de overfitting, puede observarse como aunque en train no obtiene buenos resultados, en validation sigue descendiendo la recta y alcanza ya al Modelo 4 que por overfitting está casi al mismo valor, pero en dirección ascendente la pendiente.

3.2.3. Gráfica del Modelo VGG16

Tras aplicar la red preentrenada VGG16 creo que merece ilustrar la gráfica obtenida, ya que, al contrario del resto de modelos, presenta un entrenamiento casi perfecto, con poca variabilidad y con valores de train y validation cercanos en toda la curva.

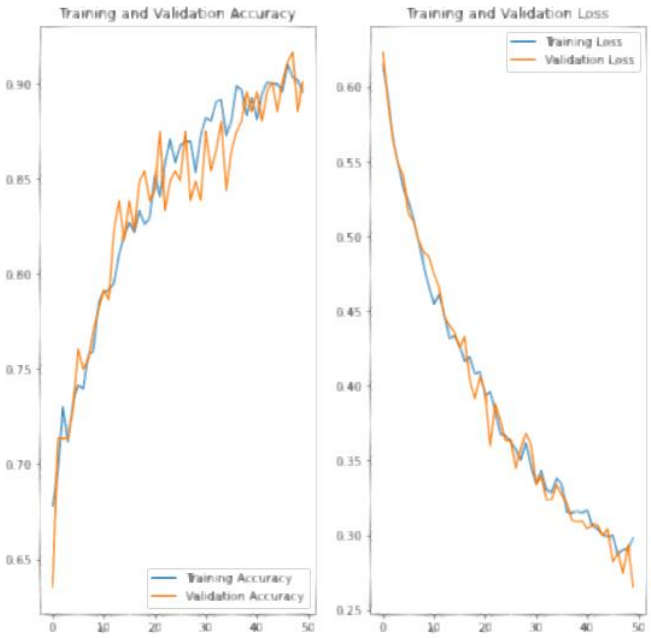


Ilustración 14: Gráficos de Accuracy y Loss function de la red VGG16

Pude observarse en las dos gráficas como el test y la validación siguen en valores muy cercanos en todas las épocas. Se diría que es un caso claro de modelo bien ajustado. Además, si se le hubieran aplicado más épocas, puede observarse como todavía los resultados hubieran sido mejores, ya que mantiene la pendiente correcta la gráfica en ambos casos. Seguramente, esta red al ser mucho más pesada que las otras aplicadas, necesitará de más épocas para llegar a su punto óptimo de entrenamiento.

4. Conclusiones

En general la práctica me ha resultado muy interesante, productiva y con la que se aprende mucho. A nivel teórico, me he quedado con ganas de seguir avanzando en la parte matemática que se recoge en los documentos propuestos. Creo que profundizar en estos aspectos puede ser clave para poder trabajar con un conocimiento más sólido en este campo. Respecto al IDE utilizado, parece que los NoteBooks (Kaggel y Google Colab) ofrecen un rendimiento de trabajo bueno, y no dan problemas con la descarga de librerías.

En cuanto los resultados de las pruebas, como era de esperar, los mejores resultados entre las redes diseñadas manualmente se han obtenido en la CNNet4, tras aplicar mejoras de capas y dropout. Así se ha conseguido suavizar los efectos del overfitting que estaban sufriendo las redes anteriores. La red VGG16 a pesar de no haber llegado a los valores de validación de las otras, su modelo bien ajustado con más épocas hubiera alcanzado mejores resultados tanto en "accuracy" como en "loss".

La CNNet1 es la que peores resultados ha presentado en las pruebas realizadas, algo que era de esperar por la simpleza de la arquitectura. Se podría afirmar que es una red que para este dataset sufre de underfitting.

Las CNNet3 y 4 son las que mejores resultados han tenido en general. Era algo que se intuía, ya que son las redes que adoptan las redes convolucionales que son optimizan la clasificación de imágenes. Las mejoras realizadas en el Modelo4 respecto la anterior en dropout y más capas han suavizado el problema del overfitting, y han permitido obtener mejores resultados finales en exactitud.

Para los modelos se han hecho otra serie de pruebas con diferentes optimizadores, ya que por ejemplo el SGD también se recomienda en diferentes fuentes de información a los que se ha acudido. Sin embargo, al final, tras probar varios (SGD, Adagrad, Adadelta…) los mejores resultados se han obtenido con Adam. También hay que comentar que los resultados en exactitud de las redes varían de prueba a prueba, pudiendo en determinados casos ofrecer un mejor rendimiento un modelo sobre otro.

Las redes preentrenadas como la VGG16 parece que son una muy buena opción para implementar ya modelos sin tener que hacer pruebas y diseños propios. Para aprender y estudiar está bien, pero quizás para empresas que buscan rapidez y buenos resultados a corto plazo, este tipo de modelos son una buena solución para sus necesidades.

Para trabajos futuros, convendría aplicar más técnicas para la mejora de resultados. En esta práctica no se ha aplicado "data augmentation" para aumentar el dataset, "data cleaning", técnicas de regularización (L1, L2…), aplicar Early Stopping…

Referencias

[1] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*. New York, NY: Springer New York, 2009.

[2] K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biol. Cybern.*, vol. 20, no. 3, pp. 121–136, 1975, doi: 10.1007/BF00342633.

[3] K. Fukushima, "A neural network for visual pattern recognition," *Computer (Long. Beach. Calif).*, vol. 21, no. 3, pp. 65–75, 1988, doi: 10.1109/2.32.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998, doi: 10.1109/5.726791.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255, doi: 10.1109/CVPR.2009.5206848.

[6] Y. LeCun *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989, doi: 10.1162/neco.1989.1.4.541.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," 2012, Accessed: Jun. 23, 2022. [Online]. Available: http://code.google.com/p/cuda-convnet/.

[8] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, Sep. 2014, doi: 10.48550/arxiv.1409.1556.

[9] C. Szegedy *et al.*, "Going Deeper with Convolutions," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June-2015, pp. 1–9, Sep. 2014, doi: 10.48550/arxiv.1409.4842.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, pp. 770–778, Dec. 2015, doi: 10.48550/arxiv.1512.03385.

[11] J. R. Hilera González and V. J. Martínez Hernando, *Redes neuronales artificiales : fundamentos, modelos y aplicaciones*. Madrid: RA-MA, 1995.

[12] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. USA: Prentice Hall PTR, 1998.

[13] "Keras: the Python deep learning API." https://keras.io/ (accessed Jun. 23, 2022).

[14] "TensorFlow." https://www.tensorflow.org/ (accessed Jun. 23, 2022).

[15] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*, vol. 26. Cambridge Massachusetts: The MIT Press, 2016.