Redes neuronales artificiales en el contexto de la visión artificial

TRABAJO DE FIN DE GRADO
Curso 2022-2023

FACULTAD DE CIENCIAS MATEMÁTICAS

GRADO EN INGENIERÍA MATEMÁTICA

Jaime de la Iglesia López

Directores: D. Antonio López Montes, Dª. María Teresa Benavent Merchán, D. Antonio Martinez Raya y D. José Ángel González Prieto

Madrid, 1 de julio de 2023

Resumen

En este trabajo se aborda la relación entre la visión artificial y las redes neuronales artificiales. En primer lugar se destaca la importancia de la visión artificial en diversos campos y se analizan conceptos clave como la representación digital de imágenes y la aplicación de filtros mediante convolución para elimar el ruido o realzar características relevantes. En segundo lugar se introduce el fundamento matemático de las redes neuronales, describiendo sus elementos básicos y el proceso de entrenamiento mediante backpropgation. Por último se profundiza en las redes neuronales convolucionales (CNN), destacando su capacidad para extraer características de imágenes y su eficacia en la clasificación. Finalmente, se implementa un ejemplo práctico en Matlab de tres redes convolucionales para la clasificación de imágenes.

Palabras clave: Imagen, píxel, filtro, correlación cruzada, aprendizaje supervisado, gradiente, backpropagation, pooling.

Abstract

This work addresses the relationship between computer vision and artificial neural networks. Firstly, the importance of computer vision in various fields is highlighted, and key concepts such as digital image representation and the application of filters to remove noise or enhance relevant features are analyzed. Secondly, the mathematical foundation of neural networks is introduced, describing their basic elements and the training process using backpropagation. Lastly, a deeper dive into convolutional neural networks (CNN) is conducted, emphasizing their ability to extract image features and their effectiveness in classification. Finally, a practical example is implemented in Matlab using three convolutional networks for image classification.

Key words: Image, pixel, filter, cross correlation, cross correlation, supervised learning, gradient, backpropagation, pooling.

Índice general

CAPÍTULO 1

Introducción

1.1. Motivación del trabajo

La visión artificial es uno de los campos de la informática que mayor presente y, sobre todo futuro, tienen en nuestra sociedad.

Una herramienta que ha revolucionado la visión artificial son las redes neuronales artificiales, en concreto las convolucionales o CNN (convolutional neural network). Estas redes surgen con la intención de simular la visión humana y en estos últimos años, están experimentando un desarrollo sin precedentes.

1.2. Objetivos

El objetivo principal de este trabajo es profundizar en el conocimiento de la visión artificial y de las redes neuronales artificiales, desde los fundamentos hasta las aplicaciones.

1.3. Contenido del trabajo

En primer lugar, se abordará el campo de la visión artificial, estudiando, desde el punto de vista matemático, qué es una imagen digital, qué es un filtro, los diferentes tipos de filtros que hay y cómo estos se aplican a las imágenes mediante un proceso que se denomina convolución de imágenes. Cada filtro se progamará en Matlab y se verá el resultado de su aplicación en una imagen cualquiera.

En segundo lugar, se estudiarán los conceptos matemáticos básicos y se introducirán los esquemas más sencillos de redes neuronales artificiales, sirviendo de introducción para el estudio de las CNN.

Después, se profundizará en las redes neuronales convolucionales (CNN), en su estructura y funcionamiento, y se concluirá con un ejemplo de aplicacion práctica en Matlab de clasificación de imágenes mediante CNN.

CAPÍTULO 2

Matemáticas y Visión artificial

La visión artificial se encarga de analizar y entender imágenes de forma similar a como lo hacen los humanos. Entre la multitud de aplicaciones de la visión artificial, podemos destacar las relacionadas con la robótica, medicina, automoción de vehículos, videojuegos, etc.

2.1. Imagen digital

El objeto de estudio de la visión artificial son las **imágenes digitales**. Desde el punto de vista matemático estas se representan como **matrices de píxeles**.

Las imágenes a blanco y negro son funciones discretas de dos variables $f(x, y)$ de tal manera que a cada punto $(x, y) \in N \times N$, se le asocia el valor de la intensidad luminosa en ese punto que está entre 0 y 255. Se pueden representar como matrices de píxeles I en las que cada píxel cuantifica el nivel de intensidad de la imagen en ese punto. Los valores cercanos a 0 se corresponden con tonalidades más cercanas al negro y a medida que crece se va acercando más al blanco.

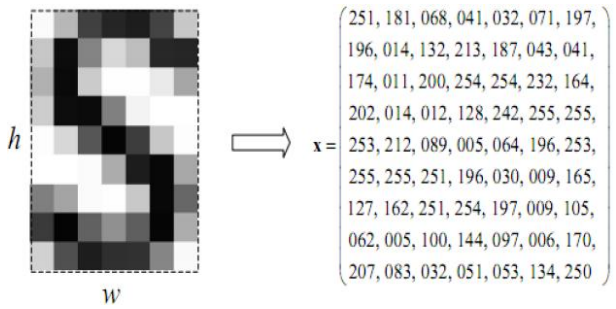


Figura 2.1: Matriz de píxeles de imagen en blanco y negro.
Fuente: link

Las imágenes a color que se tratan en el día a día tienen 3 canales de color: rojo, verde y azul (RGB). Por lo tanto, se van a representar de la siguiente manera:

$$f(x,y,z) = \begin{cases} R(x,y) & si \quad z = 0 \\ G(x,y) & si \quad z = 1 \\ B(x,y) & si \quad z = 2 \end{cases}$$

Cada una asociada al nivel de intensidad de ese color primario en cada punto (x, y). Se representa como una matriz de tres dimensiones, en la que la tercera coordenada corresponde al canal.

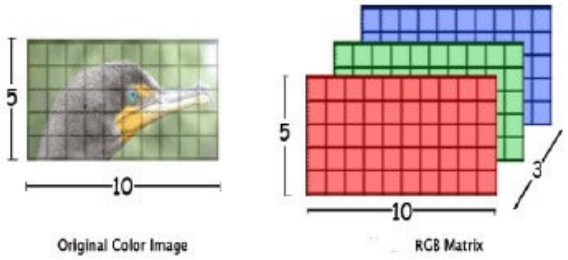


Figura 2.2: Matrices de píxeles (canales RGB) de una imagen a color.
Fuente: link

Otro concepto fundamental en la visión artificial es el de **filtro o kernel**. Son también funciones discretas que tienen asociadas matrices de tamaño reducido donde cada coeficiente (x, y) corresponde a un píxel. En cada una de las entradas de la matriz se encuentra un coeficiente cuyo valor dependerá de la máscara que se esté utilizando.

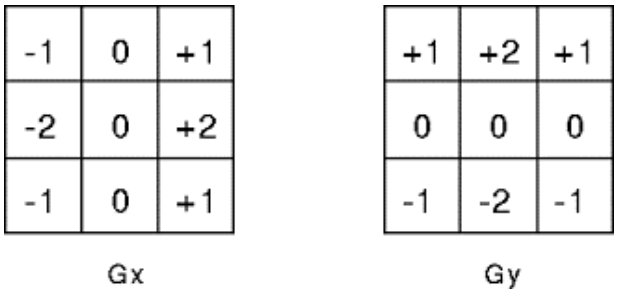


Figura 2.3: Filtros de Sobel para la detección de bordes.
Fuente: Elaboración propia.

El proceso de aplicar estos filtros a la imagen recibe el nombre de **convolución de imágenes**.

2.2. Convolución de imágenes

La convolución de imágenes se basa en una operación matemática: **la correlación cruzada**. No obstante, el nombre de convolución de imágenes lo hereda de la operación

matemática **convolución**, con la que se está más familiarizado y que guarda una estrecha relación con la correlación cruzada.

La convolución se refiere originalmente a una operación matemática que transforma dos funciones f y g en una tercera función, que en cierto sentido, representa la magnitud en la que se superponen f y una versión trasladada e invertida de g.

La **convolución** de f y g, denotado por $f * g$, se define como:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x)g(t-x)\,dx \quad \forall f, g \ \in L^2[\mathbb{R}]$$

La **correlación cruzada**, denotada como $g \star f$, se define como:

$$(g \star f)(t) = \int_{-\infty}^{\infty} g(x)f(t+x)\,dx$$

Convolución y correlación cruzada están relacionadas de la siguiente manera (se demuestra fácilmente con el cambio de variable $u = -x$):

$$f(t) * g(t) = f(-t) \star g(t)$$

Supóngase que f es la función asociada a la imagen y g la función asociada al filtro. Como son funciones discretas de dos o tres variables definidas en el dominio de los enteros (escala de grises y RGB respectivamente), interesa la versión discreta de correlación cruzada en 2D Y 3D. La operación está definida en todo el dominio de la imagen f menos en los bordes.

$$(g \star f)(i,j) = \sum_m \sum_n g(m,n)f(i+m, j+n) \quad \forall\ (i,j)\ \in\ dominio(f) \backslash bordes(f)$$

$$(g \star f)(i,j,1) = \sum_m \sum_n \sum_p g(m,n,p)f(i+m, j+n, k+p) \quad \forall\ (i,j,1)\ \in\ dominio(f) \backslash bordes(f)$$

donde la tercera variable hace referencia al canal de la imagen.

Además, como f y g, tienen asociadas matrices de píxeles I y G, estas sólo toman valores distintos de 0 en unos determinados puntos. El filtro será siempre de menor tamaño que la imagen, f, por lo que la multiplicación será 0 fuera del dominio del filtro. Por tanto m,n y p recorreran los puntos donde esté definido el filtro.

Si se considera un filtro 2D de tamaño imapar $(2a+1) \times (2b+1)$, por tanto, el filtro tiene centro y una imagen f cualquiera, la correlación cruzada entre g y f en un punto del dominio de la imagen, vendría dada por la siguiente ecuación:

$$(g \star f)(i,j) = \sum_{m=-a}^{a} \sum_{n=-b}^{b} g(m,n)f(i+m, j+n) \quad \forall\ (i,j)\ \in\ dominio(f) \backslash bordes(f)$$

De igual manera para un filtro 3D de tamaño imapar $(2a+1) \times (2b+1) \times 3$, para una imagen RGB. La correlación cruzada entre g y f en un punto del dominio de la imagen, vendría dada por:

$$(g \star f)(i,j,1) = \sum_{m=-a}^{a} \sum_{n=-b}^{b} \sum_{p=0}^{2} g(m,n,p) f(i+m, j+n, 1+p) \quad \forall\, (i,j,1) \;\in\; dominio(f) \backslash bordes(f)$$

Las dos ecuaciones anteriores definen el proceso de correlación cruzada entre una imagen f y un filtro g en un punto del dominio de la imagen.

Desde el punto de vista matricial, la operación de correlación cruzada entre f y g en un punto del dominio de f se traduce en la suma de los elementos de la matriz resultante del producto elemento a elemento entre el kernel y la región coincidente de la matriz de la imagen centrada en ese punto.

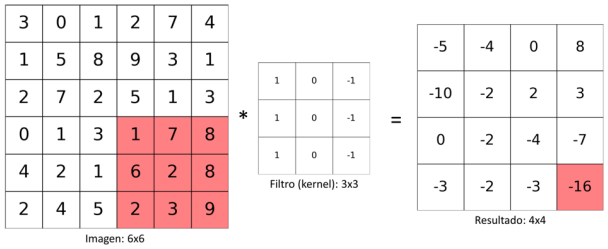


Figura 2.4: Convolución digital 2D.
Fuente: link

En la convolución de imágenes, el filtro se va a ir desplazando, recorriendo todos los píxeles, centrales de la matriz I asociada a la imagen f, calculando la correlación cruzada explicada anteriormente y obteniendo una imagen convolucionada.

Para el caso de las imágenes RGB, la convolución de imágenes 3D funciona de igual manera que la 2D. El kernel o filtro, en este caso, va a ser un cuboide de $a \times b \times 3$ que se irá desplazando en las tres dimensiones dando lugar a una imagen concolucionada en 2D.

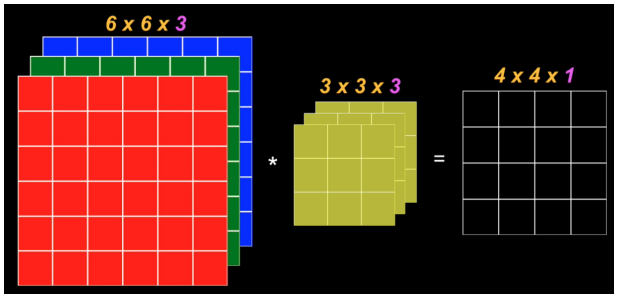


Figura 2.5: Convolución digital 3D.
Fuente: link

Es importante observar que la imagen convolucionada es de dimensión menor que la imagen original. Para evitar esto, en la práctica se utiliza el **padding o relleno**. Esto consiste simplemente en agregar pixeles con valor igual a cero a los bordes de la imagen original, de esta manera podemos controlar el tamaño del filtro y el de la salida de manera independiente.

Otro parámetro que se puede variar es el **stride**, el número de píxeles que el kernel se desplaza horizontal y verticalmente durante la operación de convolución. En lo anterior se ha estado considerando stride =1. Se suele aplicar en imágenes en las que los valores de píxeles cercanos son bastante similares, evitando muestrear cada píxel, aumentando la velocidad de procesamiento, pero también reduciendo el tamaño de salida. De esta manera, ajustando estos parámetros, se consigue controlar el tamaño de la imagen convolucionada.

En la siguiente sección se profundizará en los diferentes tipos de filtros que se emplean dependiendo del objetivo.

2.3. Tipos de filtros

2.3.1. Filtros de preprocesamiento

La imagen digital puede contener distintos tipos de ruido que afecten negativamente al rendimiento del sistema de visión artificial y disminuir la precisión de los resultados.

(a) Imagen original Facultad Matemáticas UCM.

(b) Imagen con ruido Facultad Matemáticas UCM.

Figura 2.6: Ruido en una imagen.
Fuente: Elaboración propia.

El objetivo de los filtros de esta fase, que se denominan **filtros de paso bajo o suavizado**, es tratar de disminuir ese ruido. Algunos de interés son:

- **Filtro de la media**: Asigna al píxel central la media de todos los pixeles incluidos en la ventana. La matriz de filtrado estaría compuesta por unos y el divisor sería

el número total de elementos en la matriz. Sus limitaciones son la preservación insuficiente de los bordes y características importantes.

Ejemplo de máscara asociada: $G = \frac{1}{9}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

- **Filtro de media ponderada**: Los elementos de la matriz de filtrado no son todos 1 sino que se da más peso a uno de ellos (generalmente el central) para obtener un resultado más parecido a la imagen original y evitar que aparezca borrosa.

  Ejemplo de máscara asociada: $G = \frac{1}{10}\begin{pmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

- **Filtro Gaussiano**: Simulan una distribución gaussiana bivariante. El valor máximo aparece en el píxel central y disminuye hacia los extremos a una velocidad inversamente proporcional a la desviación típica σ. Esto consigue suavizar la imagen sin degradar tan notablemente estructuras como puntos, líneas o bordes, a diferencia del filtro de la media.

  Ejemplo de máscara asociada: $G = \frac{1}{16}\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$

(a) Filtro de Gauus 3x3 aplicado a imagen con ruido.

(b) Filtro de la media 3x3 aplicado a imagen con ruido.

Figura 2.7: Aplicación filtro Gauss y media.
Fuente: Elaboración propia.

- **Filtro máximo o mínimo**: Estos filtros resultan útiles para eliminar el **ruido sal y pimienta**, que es la aparición dispersa de pixeles blancos (sal) y negros (pimienta).

  El filtro de máximo asigna el mayor valor dentro de una ventana ordenada de pixeles de nivel de gris, es decir, elimina los píxeles negros (ruido pimienta) y tiende a aclarar la imagen. Por el contrario, el filtro de mínimo asigna el menor valor eliminando el ruido sal y tiende a oscurecer la imagen.

(a) Imagen con ruido del tipo «sal». (b) Filtro mínimo aplicado.

Figura 2.8: Aplicación filtro mínimo.
Fuente: Elaboración propia.

En estos filtros de suavizado es imprescindible elegir con cautela el tamaño de los mismos. Un tamaño grande del filtro puede suavizar en exceso la imagen y mermar el propósito de mejorarla. Hasta ahora se han usado filtros de 3×3, se prueba ahora con uno de 18×18.

Figura 2.9: Aplicación filtro de Gauss de 18×18.
Fuente: Elaboración propia.

2.3.2. Filtros para la segmentación de la imagen

La segmentación de imágenes es el proceso por el cual, a partir de una imagen, produce otra en la que cada píxel tiene asociada una etiqueta distintiva del objeto al que pertenece. Así, una vez segmentada una imagen, se podría formar una lista de objetos consistentes en las agrupaciones de los píxeles que tengan la misma etiqueta.

En este proceso de segmentación, destacan los **filtros de detección de bordes o de paso alto**. Los bordes no dejan de ser líneas de píxeles que separan los objetos del fondo de la imagen y, por tanto, se corresponden con los puntos donde se producen discontinuidades en los valores de los píxeles adyacentes.

La mayoría de las técnicas para detectar bordes emplean operadores locales basados en distintas aproximaciones discretas de la **primera y segunda derivada** de los niveles de grises de la imagen.

Sea un punto $X \in \mathbb{R}^n$, una función diferenciable $f : \mathbb{R}^n \to \mathbb{R}$. El **gradiente** de f en X es un vector de $\mathbb{R}^n$ con la forma:

$$\nabla f(X) = \left(\frac{\partial f(X)}{\partial x_1}, ..., \frac{\partial f(X)}{\partial x_n} \right)$$

.

El gradiente de una función en un punto indica la dirección para la que hay un mayor grado de cambio en ese punto, por lo que va a ser fundamental para la detección de los bordes y la dirección de los mismos en la imagen.

Como se está trabajando con imágenes, al igual que con la correlación cruzada, interesa la aproximación discreta de las derivadas parciales en los ejes x e y. En función de las distintas aproximaciones de las derivadas se definen distintos tipos de filtros de la forma:

$$f_x = \frac{\partial f}{\partial x} = (I \odot G_x)$$

$$f_y = \frac{\partial f}{\partial y} = (I \odot G_y)$$

Donde I es la matriz de pixeles de la imagen f, G_x y G_y son los kernels asociados a los filtros que funcionan como aproximaciones discretas del operador de derivada parcial en los ejes x e y respectivamente y van a ser capaces de detectar bordes horizontales y verticales respectivamente. Y por último, $\odot$ es el producto de Hadamard de matrices[1].

En cada punto de la imagen, los resultados de las aproximaciones de los gradientes horizontal y vertical pueden ser combinados para obtener la magnitud del gradiente mediante:

$$|\nabla f(x, y)| = \sqrt{f_x^2(x, y) + f_y^2(x, y)}$$

Una práctica habitual es aproximar la magnitud del gradiente con valores absolutos:

$$|\nabla f(x, y)| = |f_x(x, y)| + |f_y(x, y)|$$

[1]El producto de Hadamard entre matrices no es más que multiplicar elemento por elemento los elementos de las dos matrices.

De esta manera se obtiene una imagen convolucionada en la que se detectan bordes en cualquier dirección.

Ejemplos de filtros de detección de bordes:

FILTRO	G_x	G_y
Roberts	$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix}$	$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$
Sobel	$\begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 1 \\ 1 & 2 & 1 \end{pmatrix}$
Prewitt	$\begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

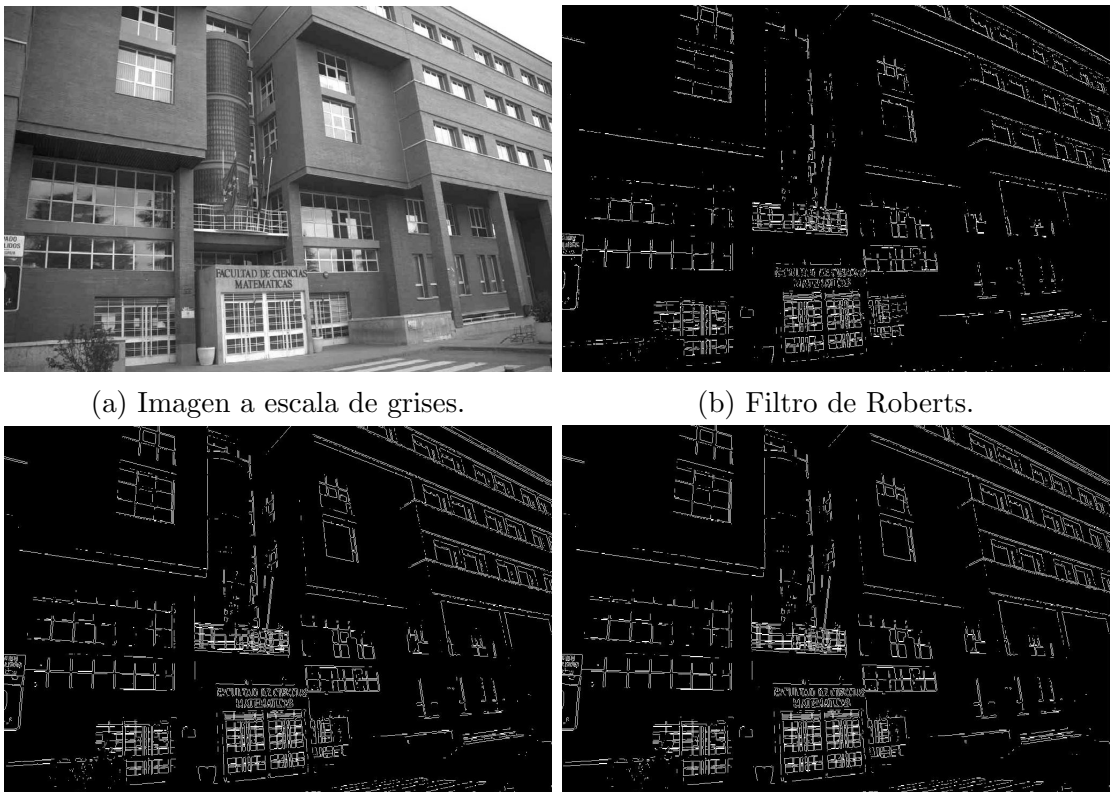
Figura 2.10: Filtros detección de bordes que usan el gradiente.
Fuente: Elaboración propia.

El gran inconveniente del **filtro de Roberts** es su extremada sensibilidad al ruido y es debido a su simplicidad. Pequeñas fluctuaciones de intensidad pueden generar respuestas significativas en la salida del filtro de Roberts, lo que lleva a una detección de bordes menos precisa.

Los **filtros de Sobel y Prewitt** funcionan ambos de forma similiar, tienen matrices asociadas más completas e involucran a más píxeles vecinos para proporcionar mayor inmunidad al ruido que el de Roberts.

Los tres detectan bordes horizontales y verticales por separado y posteriormente se combinan en una sola imagen convolucionada mediante el módulo del gradiente.

Veáse a continuación, un ejemplo de aplicación de los tres filtros mencionados anteriormente.



(a) Imagen a escala de grises. (b) Filtro de Roberts.

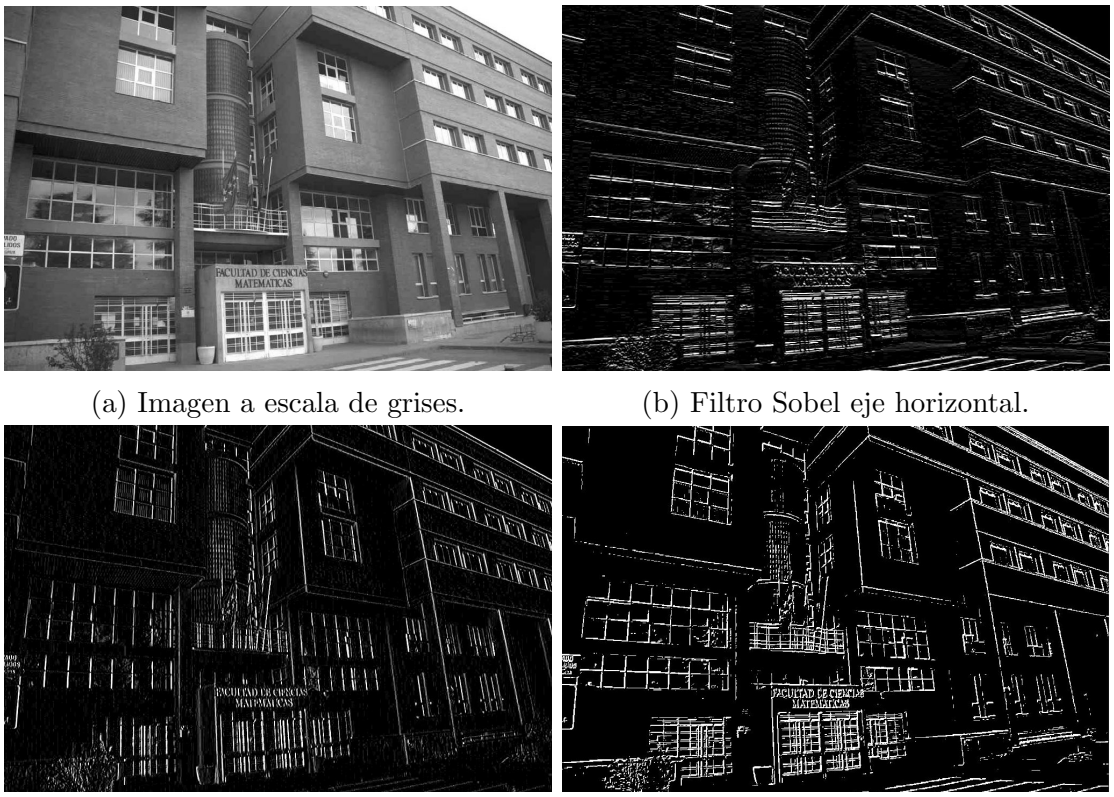
(c) Filtro de Sobel. (d) Filtro de Prewitt.

Figura 2.11: Aplicación detección de bordes.
Fuente: Elaboración propia.

Primero, es recomendable convertir la imagen a escala de grises. Esto se debe a que los filtros de detección de bordes se basan en la diferencia de intensidad entre los píxeles adyacentes para detectar los bordes en una imagen. Si la imagen está en color, los valores de intensidad de los píxeles se componen de diferentes canales de color (como rojo, verde y azul), lo que dificulta la detección precisa de bordes. Además, la conversión a escala de grises reduce la cantidad de datos necesarios para procesar la imagen, lo que hace que el proceso sea más rápido y eficiente.

En las imágenes de arriba se puede observar la combinación de los filtros de detección de bordes verticales y horizontales de Roberts, Sobel y Prewitt. Los bordes están resaltados en blanco puesto que se corresponden con los píxeles donde la primera derivada es más alta.

Se puede ver que los bordes están un poco saturados, esto se soluciona normalizando los filtros y se puede apreciar en el siguiente ejemplo, donde se descompone el filtro de Sobel para la detección de bordes en los ejes x e y por separado, y posteriormente se combinan como se ha estudiado:



(a) Imagen a escala de grises.

(b) Filtro Sobel eje horizontal.

(c) Filtro Sobel eje vertical.

(d) Filtro Sobel Normalizado.

Figura 2.12: Aplicación Sobel.
Fuente: Elaboración propia.

Por último, mencionar los filtros que utilizan la segunda derivada, también conocidos como **filtros de Laplace**, que también son filtros utilizados para la detección de bordes.

Por el **desarrollo de Taylor**, se consideran las siguientes aproximaciones de f en los siguientes puntos:

$$f(x+h,y) = f(x,y) + hf_x(x,y) + h^2\frac{f_{xx}(x,y)}{2} + O(h^3)$$

$$f(x-h,y) = f(x,y) - hf_x(x,y) + h^2\frac{f_{xx}(x,y)}{2} + O(h^3)$$

Si se despeja de la primera $f_x(x,y)$, considerándose la aproximación de grado 1, se tiene $f_x(x,y) = \frac{f(x+h,y)-f(x,y)}{h}$. Como se está trabajando con imágenes, se toma paso $h = 1$, y queda: $f_x(x,y) = f(x+1,y) - f(x,y)$.

Así, ya se tiene la aproximación de la diferencia finita de primer orden respecto de x de f por Taylor. Para obtener la segunda:

$$f(x+h,y) + f(x-h,y) = 2f(x,y) + h^2 f_{xx}(x,y)$$

Se despeja f_{xx} y se considera $h = 1$ de nuevo:

$$f_{xx}(x,y) = f(x+1,y) - 2f(x,y) + f(x-1,y)$$

De esta manera, se tienen de las diferencias finitas de primer y segundo orden de f respecto de f. Para y, se demuestra de la misma manera.

$$f_y(x,y) = f(x,y+1) - f(x,y)$$

$$f_{yy}(x,y) = f(x,y+1) - 2f(x,y) + f(x,y-1)$$

Considérese el operador Laplaciano:

$$\nabla^2 f(x,y) = f_{xx}(x,y) + f_{yy}(x,y)$$

Se sustituyen las aproximaciones anteriores de las segundas derivadas en el operador Laplaciano y se obtiene:

$$\nabla^2 f(x,y) = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

$$\nabla^2 f(x,y) = (I \odot L)(x,y)$$

Donde I es la matriz asociada a la imagen f, L es el kernel del filtro conocido como filtro de Laplace y $\odot$ es el producto de Hadamard:

$$L = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Figura 2.13: Aplicación filtro laplaciano.
Fuente: Elaboración propia.

Para resumir:

1. Los filtros basados en la primera derivada utilizan dos máscaras separadas, una para la detección de bordes horizontales y otra para la detección de bordes verticales. Luego, aplica estas máscaras a la imagen original para obtener dos imágenes filtradas, una para cada dirección. Finalmente, combina estas dos imágenes para obtener una imagen que resalte los bordes en ambas direcciones. Estos filtros se utilizan comúnmente para detección de bordes en imágenes de alta resolución, como imágenes médicas.

2. Por otro lado, el filtro de Laplace utiliza una sola máscara para la detección de bordes, que se basa en la segunda derivada de la intensidad de la imagen. Este filtro puede resaltar los bordes en cualquier dirección y se utiliza comúnmente en la detección de bordes en imágenes de baja resolución.

Las redes neuronales convolucionales utilizan los filtros que se han introducido en este capítulo. Antes de entrar de lleno con ellas, es necesario introducir los conceptos y estructuras más básicos de las redes neuronales tradicionales.

CAPÍTULO 3

Redes neuronales artificiales

3.1. Introducción

El **aprendizaje automático** es un subcampo de la Inteligencia Artificial que dota a los ordenadores de la capacidad de identificar patrones en datos masivos y elaborar predicciones.

Las redes neuronales artificiales pertenecen a un tipo concreto de aprendizaje automático, el **supervisado**. Estas se entrenan con un conjunto de **datos de entrenamiento**, compuestos por unos datos de entrada y su resultado deseado. Más tarde se usa un conjunto de prueba, llamado **conjunto de test**, para determinar su eficacia. Si los datos de salida son variables discretas y finitas se está ante un problema de **clasificación**, si por el contrario la salida es continua, se trata de un problema de **regresión**.

Las primeras redes neuronales artificiales surgen en los años 40 del siglo pasado, con la intención de simular el funcionamiento de las neuronas del cerebro humano, que se relacionan entre ellas, permitiendo la realización de funciones cognitivas y comportamentales complejas. Las redes neuronales artificiales van a seguir esta idea.

3.2. Estructura de una red neuronal. Perceptrón multicapa

Uno de los primeros ejemplos de red neuronal que se implementó, fue el **perceptrón**, creado por Frank Rosenblatt en la década de 1950.

El perceptrón simple es un clasificador binario que consta de una sola capa, además de la entrada y la salida. En esa capa se hará un promedio de los valores de entrada y se le aplicará una **función de activación escalón**, que produce una salida binaria en

función de si la suma de entradas ponderadas supera un umbral. Esta función de activación produce una superficie de decisión lineal que divide el espacio de entrada en dos regiones: una región para cada una de las dos clases que se están clasificando. Es la red neuronal más simple.

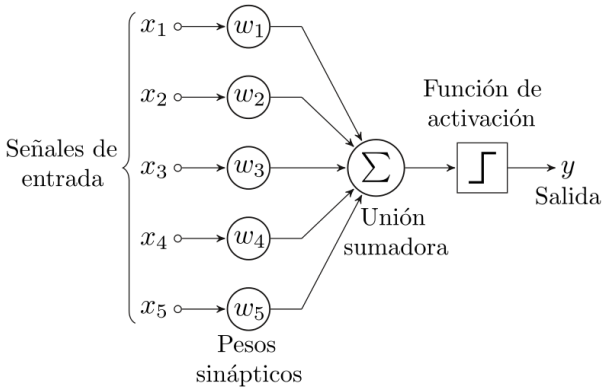


Figura 3.1: Estructura Perceptrón simple.
Fuente: link

Las neuronas, aisladas, solamente pueden separar conjuntos linealmente separables, como es el caso del perceptrón simple (función de activación escalón) o problemas no lineales muy simples en el caso de usar otras funciones de activación.

El **perceptrón multicapa** supera esta limitación, incorporando capas de neuronas ocultas (**Deep learning**) y aplicando en ellas funciones de activación no lineales. Está compuesto por una capa de entrada que recibe directamente la información del exterior, unas capas intermedias u ocultas y una capa de salida que transfiere información de la red hacia el exterior.

Cada capa está compuesta por neuronas en paralelo que se activarán o no en función de conceptos que veremos más adelante. Cada neurona recibirá la información exclusivamente de las neuronas de la capa anterior[1]. Se dirá que una red está totalmente conectada si todas las neuronas de una capa están conectadas con todas y cada una de las neuronas de la siguiente capa.

La información se transmite hacia delante (Forward Propagation) desde la capa de entrada, pasando por las capas ocultas y finalmente llegando a las capas de salida donde se realizará la tarea de clasificación o regresión.

[1]Red neuronal prealimentada, diferente de las recurrentes.

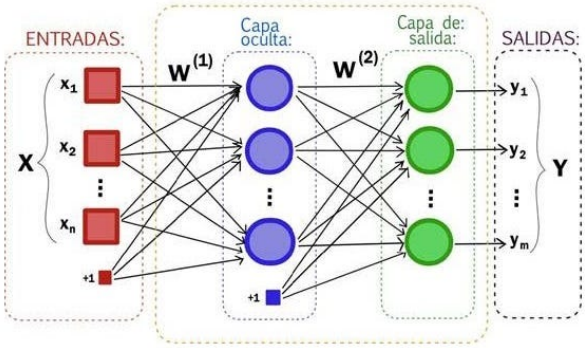


Figura 3.2: Estructura Perceptrón multicapa con una capa oculta y 3 salidas.
Fuente: link

Defínase formalmente la estructura del perceptrón multicapa y su notación:

Supóngase una red neuronal con L capas (profundidad de la red) y n_i neuronas en cada capa i (anchura de la capa i) para $i = 1, .., L$. Se tiene:

- **Datos de entrada**: $X = (x_1, \ldots, x_{n_1})^t$.
- **Datos de salida**: $Y = (y_1, \ldots, y_{n_L})^t$.
- **Pesos**: Representa la intensidad de la conexión entre dos neuronas, $w_{ji}^{[l]}$ es el peso de la conexión de la neurona $i - ésima$ de la capa $(l-1) - ésima$ con la $j - ésima$ neurona de la capa $l - ésima$. Para cada capa l, se define:

$$W_l = \begin{pmatrix} w_{11}^{[l]} & \ldots & w_{1n_{l-1}}^{[l]} \\ \ldots & \ldots & \ldots \\ w_{n_l 1}^{[l]} & \ldots & w_{n_1 n_{l-1}}^{[l]} \end{pmatrix}$$

- **Sesgo**: Controla qué tan predispuesta está la neurona a activarse independientemente de los pesos. Se representa como $b_j^{[l]}$ el sesgo de la neurona j en la capa l.

$$B_l = \left(b_1^{[l]}, \ldots, b_{n_l}^{[l]}\right)^t$$

- **Función de activación**: Determina el nivel de excitación de cada neurona. Se representa como $\sigma(x)$. Normalmente son no lineales y la importancia de estas reside, en que, sin ellas, las redes serían incapaces de resolver problemas no lineales, comportándose de forma similar a un perceptrón simple.

  Además, también ayudan a que los valores de salida estén acotados en un determinado rango. Se buscará que sean sencillas de derivar para así facilitar el entrenamiento de la red.

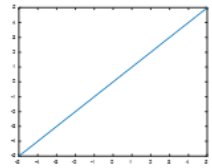
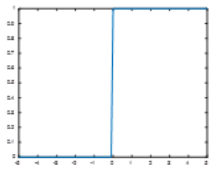
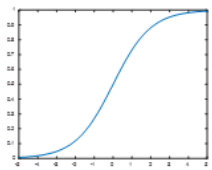
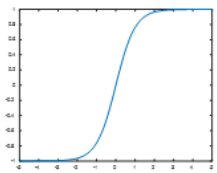
Función identidad		$\sigma(x) = x$
Función escalón		$\sigma(x) = \begin{cases} 0 \ si \ x \leq 0 \\ 1 \ si \ x > 0 \end{cases}$
Función logística		$\sigma(x) = \frac{1}{1+e^{-x}}$
Función tanh		$\sigma(x) = \tanh(x)$

Figura 3.3: Ejemplo funciones de activación.
Fuente: Elaboración propia.

La primera imagen corresponde a la frontera de decisión de una red que usa funciones de activación lineales. La segunda imagen corresponde a la de una red neuronal artificial con múltiples capas en las que se aplican funciones no lineales. Se puede observar que, en la segunda, la frontera de decisión es mucho más compleja.

Figura 3.4: Fronteras de decisión con función de activación lineal y no lineal.
Fuente: link

Una vez introducidos estos conceptos y la notación, se está listo para entender cómo funcionan las redes neuronales y cómo estas realizan sus predicciones.

3.3. Forward Propagation

Fijados unos pesos y sesgos que se ajustarán en el entrenamiento, se propagan los datos de entrada, desde la capa de entrada hacia la de salida, para elaborar la predicción de la

red neuronal.

En cada neurona se calcula su salida a partir de las salidas de las neuronas de la capa anterior. La salida a_j^l de la neurona j de la capa l se define de la siguiente forma:

$$a_j^{[l]} = \sigma\left(\sum_{i=1}^{n_{l-1}} w_{ji}^{[l]} a_i^{[l-1]} + b_j^{[l]}\right)$$

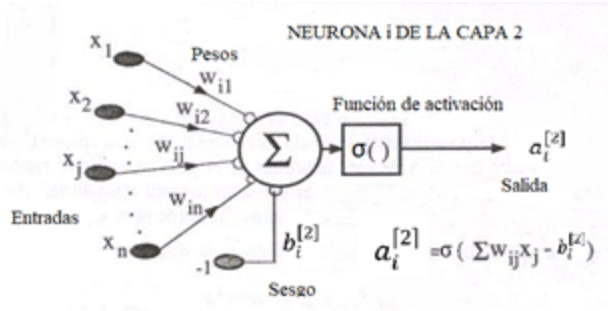


Figura 3.5: Ejemplo cálculo de activación de una neurona de la capa 2.
Fuente: Elaboración propia.

Teniendo en cuenta que:

$$a_j^{[1]} = x_j \ \forall j = 1, ..., n_1$$

$$a_j^{[L]} = \widehat{y_j} \ \forall j = 1, ..., n_L$$

De manera recursiva, se calcula la salida de la red neuronal a partir de los datos de entrada. Este es el algoritmo de **Forward Propagation**:

$$\widehat{Y} = \sigma(W_L(\sigma(W_{L-1}(...\sigma(W_2\vec{x} + B_2) + ...) + B_{L-1}) + B_L)$$

Es importante recalcar que, durante este algoritmo, los pesos y los sesgos están fijados, siendo la única variable los datos de entrada. ¿Cómo se obtienen estos pesos y sesgos? Inicialmente se toman unos pesos y sesgos cualesquiera y, mediante Forward Propagation, se calcula la salida. El método del descenso del gradiente y el algoritmo de backpropagation utilizará esta salida para actualizar los pesos por unos que minimicen una función de coste.

3.4. Aprendizaje y entrenamiento de las redes neuronales

Como ya se comentó en la introducción, las redes neuronales son algoritmos que pertenecen al **aprendizaje supervisado**. Para su entrenamiento se contará con un conjunto de datos, que dividiremos entre conjunto de entrenamiento, para ajustar los parámetros

de la red, y conjunto de test, para estudiar la eficacia de esta.

El objetivo del aprendizaje es minimizar una función de coste a partir de los datos de entrenamiento.

3.4.1. Función de coste o pérdida

La función de coste mide la diferencia entre las predicciones de una red neuronal y las salidas reales de los datos correspondientes.

El objetivo del entrenamiento de una red neuronal es minimizar la función de costo para que las predicciones de la red se aproximen lo mejor posible a las salidas reales. Existen diferentes tipos de funciones de costo que se utilizan en función del tipo de problema que se esté resolviendo. Considerando N datos de entrenamiento $\{ x_n, y_n\}, n = 1, ..., N$, algunas de las funciones de costo más comunes son:

- **Error cuadrático medio (MSE)**: Media de los errores al cuadrado entre las predicciones de la red y las salidas reales. Se utiliza en problemas de regresión.

  Siendo:

  $$C = \frac{1}{N}\sum_{i=1}^{N}\|\widehat{y}_i - y_i\|^2 = \frac{1}{N}\sum_{i=1}^{N} C_x$$

- **Entropía cruzada categórica**: Cuantifica la diferencia entre las distribuciones de probabilidad reales y las predicciones de la red. Se utiliza en problemas de clasificación multicategórica.

  $$C = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n_L} y_{ij} log\left(a_{ij}^{[L]}\right) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{n_L} y_{ij} log(\widehat{y_{ij}})$$

  Donde y_{ij} es el resultado real del dato i en la categoría j (1 si pertenece a esa categoría, 0 si no) y $a_{ij}^{[L]}$ es el resultado predicho con el dato i en la categoría j (probabilidad calculada de que pertenezca a esa categoría j).

3.4.2. Método de optimización. Desenso del gradiente

El objetivo en el aprendizaje es mejorar la red, es decir, minimizar la función de coste. El método más común para esta tarea es el del descenso del gradiente.

En el capítulo anterior ya se vió que el gradiente de una función f en el punto X se define como:

$$\nabla f(X) = \left(\frac{\partial f(X)}{\partial x_1}, ..., \frac{\partial f(X)}{\partial x_n}\right)$$

.

Para explicar el descenso del gradiente, imagínese que se está en un punto de una cordillera y se desea descender en ella porque se sabe que hay un poblado. Una manera de llegar al poblado (matemáticamente, encontrar el mínimo de una función) sería tratar de avanzar siempre cuesta abajo.

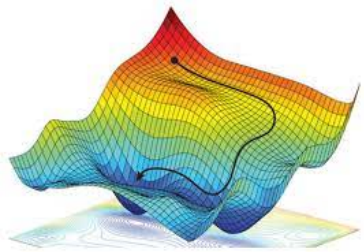


Figura 3.6: Método descenso del gradiente.
Fuente: link

El descenso del gradiente se basa en la idea de ajustar los parámetros del modelo en la dirección opuesta del gradiente de la función de costo. En otras palabras, se calcula el gradiente de la función de coste con los N datos del conjunto de entrenamiento y se actualizan los parámetros del modelo en la dirección opuesta del gradiente. Esto se repite iterativamente hasta que se alcance un mínimo local de la función de coste.

Sea $C(W, B)$ la función de coste: $C = \sum_{x=1}^{N} C_x$, donde C_x es el error para cada par de datos de entrenamiento.

Se fijan unos pesos y sesgos iniciales cualesquiera. Con el conjunto de entrenamiento se halla la función de coste y se itera de la siguiente forma hasta que la función de coste sea menor que un determinado valor:

$$w_{ji}^{[l]} = w_{ji}^{[l]} - \alpha \frac{\partial C(W, B)}{\partial w_{ji}^{[l]}}$$

$$b_{j}^{[l]} = b_{j}^{[l]} - \alpha \frac{\partial C(W, B)}{\partial b_{j}^{[l]}}$$

α es la **tasa de aprendizaje**. Un valor de α excesivamente grande hará que el método no converja, y con un valor de α muy pequeño, el coste computacional aumentará considerablemente.

Con este método, se utilizan todos los datos de entrenamiento para calcular el gradiente y así actualizar los pesos, en cada iteración. Esto puede conllevar un coste computacional elevado (si el criterio de parada es que la función de pérdida no mejore, es decir se minimice. Sin embargo, si se fija un numero de iteraciones no tiene por qué), es por ello por lo que existen diferentes variantes. La más usada en la práctica es el **Descenso de gradiente**

estocástico en mini lotes (minibatch), que utiliza un subconjunto del conjunto de entrenamiento en cada iteración seleccionado aleatoriamente.

3.4.3. Backpropagation

En la sección anterior juega un papel fundamental el concepto de gradiente. Sin embargo, calcularlo no es una tarea trivial. Hasta mediados de los 80s sólo era posible entrenar perceptrones simples debido a su elevado coste computacional. Esto supuso un parón en la investigación de las redes neuronales artificiales. Es, con la publicación del algoritmo Backpropagation de Rumelhart, Hin- ton, and Williams en 1986, cuando se produce el resurgir del desarrollo de las redes neuronales artificiales. Gracias a este algoritmo, era posible entrenar redes neuronales de múltiples capas de manera supervisada.

La idea de backpropagation es calcular el gradiente de la función de coste con respecto a cada peso mediante la regla de la cadena, calculando el gradiente de una capa cada vez, iterando hacia atrás desde la última capa para evitar cálculos redundantes en la **regla de la cadena**.

Algoritmo:

Sean N datos de entrenamiento $\{ x_n, y_n\}, n = 1, ..., N,$, y $a^{[L]}$ es el vector de salida de la red neuronal para cada dato de entrenamiento.

Considérese una función de pérdida general:

$$C = \frac{1}{N}\sum_{x=1}^{N} C(\widehat{y_x}, y_x) = \frac{1}{N}\sum_{i=1}^{N} C_x$$

Se recuerda que $\widehat{y_x} = a^{[L]} = \left(a_1^{[L]} \ldots, a_{n_L}^{[L]}\right)^t \ \forall x = 1, \ldots, N$

Para calcular las derivadas parciales de C, como es un sumatorio, se pueden calcular las derivadas parciales de cada sumando y luego sumarlas. Veáse cómo obtener las derivadas parciales de un sumando cualquiera. Es decir, el objetivo es calcular:

$$\frac{\partial C_x}{\partial w_{ji}^{[l]}} \ \ y \ \ \frac{\partial C_x}{\partial b_j^{[l]}} \ \forall l = 2, .., L, \ \forall j = 1, .., n_l, \ \forall i = 1, .., n_{l-1}.$$

Recordar antes un par de notaciones:

$$z_j^{[l]} = \sum_{i=1}^{n_{l-1}} w_{ji}^{[l]} a_i^{[l-1]} + b_j^{[l]}$$

$$a_j^{[l]} = \sigma(z_j^{[l]})$$

σ es una función de activación fácil de derivar. Por la regla de la cadena, las anteriores derivadas parciales se pueden expresar de la siguiente manera:

$$\frac{\partial C_x}{\partial w_{ji}^{[l]}} = \frac{\partial C_x}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial w_{ji}^{[l]}} = \frac{\partial C_x}{\partial z_j^{[l]}} a_i^{[l-1]}$$

$$\frac{\partial C_x}{\partial b_j^{[l]}} = \frac{\partial C_x}{\partial z_j^{[l]}} \frac{\partial z_j^{[l]}}{\partial b_j^{[l]}} = \frac{\partial C_x}{\partial z_j^{[l]}}$$

Para el propósito solo faltaría calcular $\frac{\partial C_x}{\partial z_j^{[l]}}$ para cada capa l.

Como son conocidos y_x y $C_x = C(\widehat{y_x}, y_x) = C(a^{[L]}, y_x) = C(\sigma(z^{[L]}), y_x)$. Entonces se puede calcular $\frac{\partial C_x}{\partial z_j^{[L]}}$.

Conocidos los $\frac{\partial C_x}{\partial z_j^{[L]}}$, ahora es cuando se propaga la salida hacia atrás (de ahí backpropagation) para calcular el resto de $\frac{\partial C_x}{\partial z_j^{[l]}}$ $\forall l = 2, ..., L-1$.

Por inducción, sunpóngase que se conoce $\frac{\partial C_x}{\partial z_j^{[l]}}$ y se quiere conocer $\frac{\partial C_x}{\partial z_j^{[l-1]}}$. «Abusando» del lenguaje:

$$\frac{\partial C_x}{\partial z_j^{[l-1]}} = \sum_{i=1}^{n_l} \frac{\partial C_x}{\partial z_i^{[l]}} \frac{\partial z_i^{[l]}}{\partial z_j^{[l-1]}}$$

Como se sabe que:

$$z_i^{[l]} = \sum_{j=1}^{n_{l-1}} w_{ij}^{[l-1]} a_j^{[l-1]} + b_j^{[l]} = \sum_{j=1}^{n_{l-1}} w_{ij}^{[l-1]} \sigma\left(z_j^{[l-1]}\right) + b_j^{[l]}$$

Entonces:

$$\frac{\partial C_x}{\partial z_j^{[l-1]}} = \sum_{i=1}^{n_l} \frac{\partial C_x}{\partial z_i^{[l]}} \frac{\partial \left(\sum_{p=1}^{n_{l-1}} w_{ip}^{[l-1]} \sigma(z_p^{[l-1]}) + b_p^{[l]} \right)}{\partial z_j^{[l-1]}} = \sum_{i=1}^{n_l} \frac{\partial C_x}{\partial z_i^{[l]}} \sigma'\left(z_j^{[l-1]}\right) w_{ij}^{[l]}$$

(el segundo sumatorio al derivarlo se hace 0 todas las z que no son respecto de la cual se está derivando).

Así ya se conocerían todos los $\frac{\partial C_x}{\partial z_j^{[l]}}$ y con ellos todos los $\frac{\partial C_x}{\partial w_{ji}^{[l]}}$ y $\frac{\partial C_x}{\partial b_j^{[l]}}$. Con esto ya se tendría el gradiente y se podría seguir con el método del descenso del gradiente.

En resumen, con los datos de entrenamiento y fijándose unos pesos y sesgo iniciales, se alimenta la red neuronal obteniéndose las salidas de cada neurona mediante Feedforward. Se calcula la función de coste a partir de las predicciones de la red para cada dato de entrenamiento y su salida esperada. Se calcula el gradiente mediante backpropagation y se actualizan los pesos y sesgos mediante el método del descenso del gradiente. Se repite este proceso hasta alcanzar el número de iteraciones máximo u obtener una función de

coste suficientemente pequeña.

Hay multitud de variaciones de redes neuronales artificiales como redes recurrentes (RNN), redes convolucionales (CNN), redes neuronales generativas adversarias (GAN), etc. Se profundizará, en concreto, en las convolucionales debido a su grandísima importancia en la visión artificial.

CAPÍTULO 4

Redes neuronales convolucionales

4.1. Introducción

La visión artificial y el reconocimiento de imágenes han experimentado un gran auge en los últimos años debido a los avances en el aprendizaje profundo y, en particular, en las redes neuronales convolucionales (CNN). Las redes neuronales convolucionales surgieron a partir de la necesidad de mejorar la capacidad para procesar imágenes.

Antes de la aparición de las CNN, las redes neuronales tradicionales se usaban para procesar imágenes, pero su capacidad en el caso de imágenes grandes y complejas era limitada. Esto es debido a que trataban cada píxel individualmente, por lo que solo podían clasificar con éxito imágenes que siguieran una determinada estructura. Por ejemplo, eran capaces de identificar qué número del 1 al 9 había escrito en el centro sobre un fondo negro, sin embargo, si se alteraba la localización del número o el fondo, estas redes no lo clasificaban correctamente.

Las CNN, por el contrario, en lugar de tener como datos de entrada un vector (cada componente es tratado individualmente), toman como datos de entrada, matrices de píxeles, donde los píxeles cercanos entre sí están más correlacionados que los más distantes. Estas redes utilizan capas de convolución para extraer características relevantes de las imágenes. Es decir, en estas capas de convolución se aplicarán filtros, como ya se mencionó en el capítulo 2, siendo los coeficientes de estos, los parámetros que se ajustarán en el entrenamiento de la red neuronal.

Una CNN tradicional se descompone en dos grandes etapas: el **feature learning**, formado por numerosos bloques convolucionales, donde se extraen las características principales de la imagen en forma de mapas de características, y la **fase de clasificación**, donde los mapas de características del último bloque convolucional se traducen en un vector que pasa a ser la entrada de un perceptrón multicapa fully connected, que realizará la

tarea de clasificación.

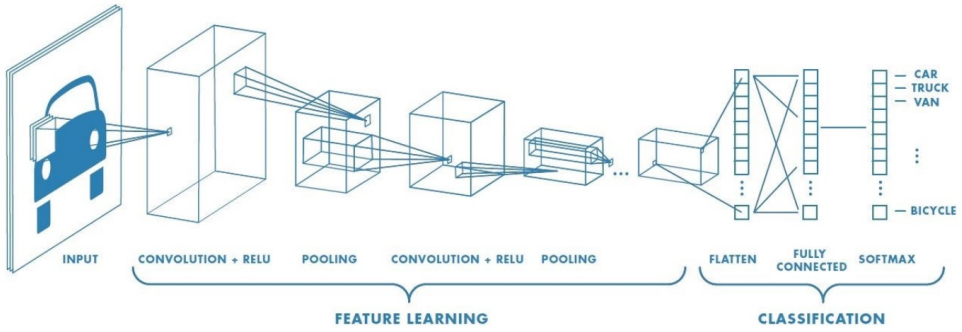


Figura 4.1: Estructura CNN.
Fuente: link

4.2. Feature Learning. Bloque convolucional

Cada bloque convolucional de la fase de feature learning va a estar compuesto, a su vez, por dos tipos de capas y en este mismo orden: **capas de convolución-relu** y **capa de pooling**.

Colocándose estos bloques uno detrás de otro se consigue ir obteniendo características cada vez más complejas (en las primeras capas se detectan bordes, en las siguientes, figuras completas, etc) de la imagen para luego su posterior clasificación.

4.2.1. Capa de convolución-ReLU

En esta capa se aplican una serie de **filtros en paralelo** a las imágenes mediante el proceso de convolución de imágenes, tal y como se explicó en el capítulo 2.Las imágenes convoluciomadas se denominan **mapas de características** y obtendrán tantos como filtros se apliquen.

Un mismo filtro sirve para extraer el mismo **rasgo o característica** en cualquier parte de la imagen. Intuitivamente, la red aprenderá filtros que se activan cuando ven algún tipo de característica visual, como un borde de alguna orientación o una mancha de algún color, como courre en las primeras capas, o eventualmente patrones completos en forma de panal o rueda, como ocurre en capas superiores de la red.

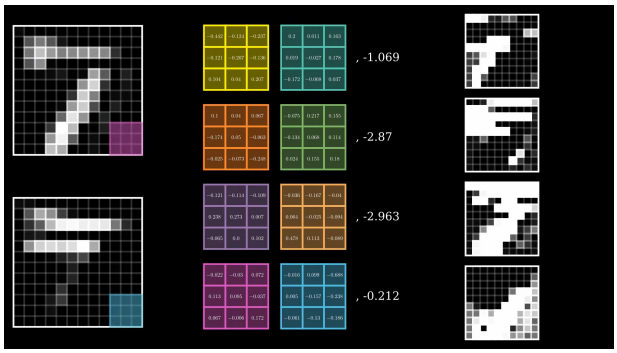


Figura 4.2: Capa convolucional intermedia con 2 mapas de características de entrada y 4 filtros $4 \times 4 \times 2$.

Fuente: link

Importante: Si en la capa l se han obtenido w mapas de características y en la capa $l+1$ se quieren aplicar k filtros, entonces se aplican k filtros de $3 \times 3 \times w$ y se obtienen k mapas de características. Además si se trabaja con imágenes en escala de grises, se tendrá como entrada un «mapa de caracteristica» por imagen, y si se trabaja con imágenes en RGB se tendrá como entrada 3 «mapas de características» por imagen, correspondientes a cada canal.

A estos mapas de características se les aplica una **función de activación ReLU** (se aplica a cada elemento de la matriz de píxeles de todos los mapas de características) (Rectified Linear Unit) que mantiene los píxeles con valores positivos y establece los valores negativos en cero, lo que permite un entrenamiento más rápido y eficaz ya que ajusta la salida a unos valores más limitados.

$$ReLU(x) = \begin{cases} x & si \quad x \geq 0 \\ 0 & si \quad x < 0 \end{cases}$$

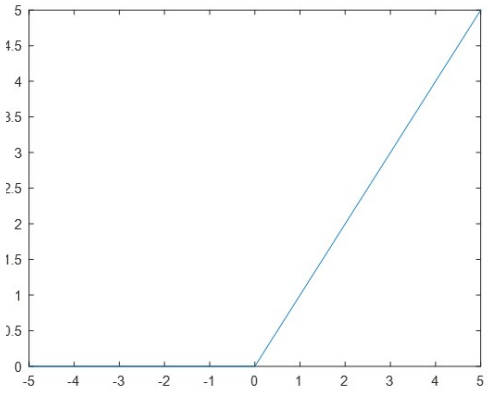


Figura 4.3: Función de activación ReLU.
Fuente: Elaboración propia.

Dependiendo de lo que se quiera clasificar, los coeficientes de estos filtros se irán ajustando automáticamente en el entrenamiento de la red neuronal. Esto supone una ventaja frente a la visión artificial tradicional, ya que estos filtros se ajustan por sí mismos y no se tienen que introducir a mano.

4.2.2. Capa de pooling

Normalmente las capas de convolución vienen seguidas por otra de agrupamiento o pooling. Esta capa ayuda a reducir la cantidad de parámetros en la red neuronal, al disminuir la dimensionalidad de la salida de la capa de convolución y también reduce el **riesgo de sobreajuste** (overfitting) que tenían las redes neuronales tradicionales al tratar con imágenes, ya que al mínimo cambio de la composición de la imagen respecto a las del conjunto de entrenamiento no la conseguía clasificar correctamente.

El overfitting se produce cuando el modelo no se puede generalizar y se ajusta demasiado al conjunto de datos de entrenamiento, funcionando correctamente con los datos de entrenamiento, pero no con datos nuevos.

Para ello se usará un filtro pool que resumirá la información de las distintas regiones de la imagen por donde se irá desplazando el filtro y, con ello reduciendo la dimensionalidad. Gracias a estos resúmenes de los píxeles próximos, las características de las imágenes son invariantes a las pequeñas traslaciones.

Tipos de pooling:

- **Max-Pooling**: Selecciona el valor máximo dentro de la ventana y lo utiliza como valor de salida. Es decir, divide la entrada en regiones y devuelve el valor máximo dentro de cada región. Es aconsejable para imágenes con fondos oscuros.

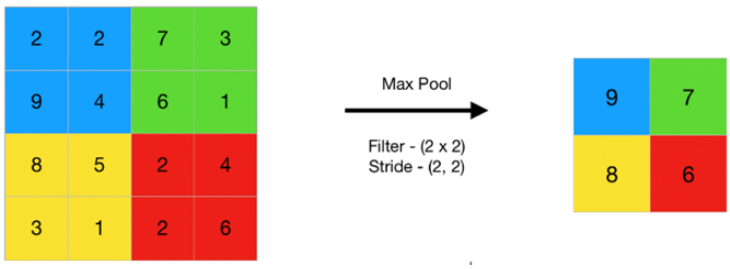


Figura 4.4: Max-Pooling.
Fuente: link

- **Average pooling**: Calcula la media aritmética de los valores dentro de la ventana y lo utiliza como valor de salida. Es decir, divide la entrada en regiones y devuelve el valor promedio de cada región. Es recomendable para preservar cracterísticas más sutiles.

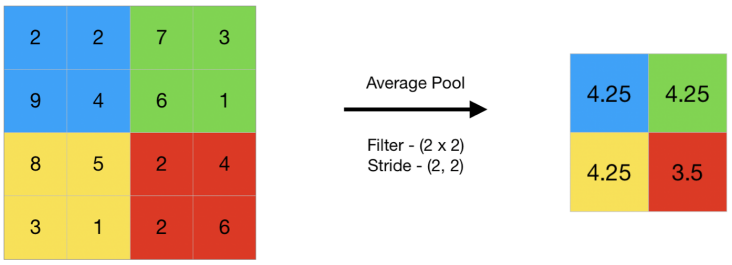


Figura 4.5: Average pooling.
Fuente: link

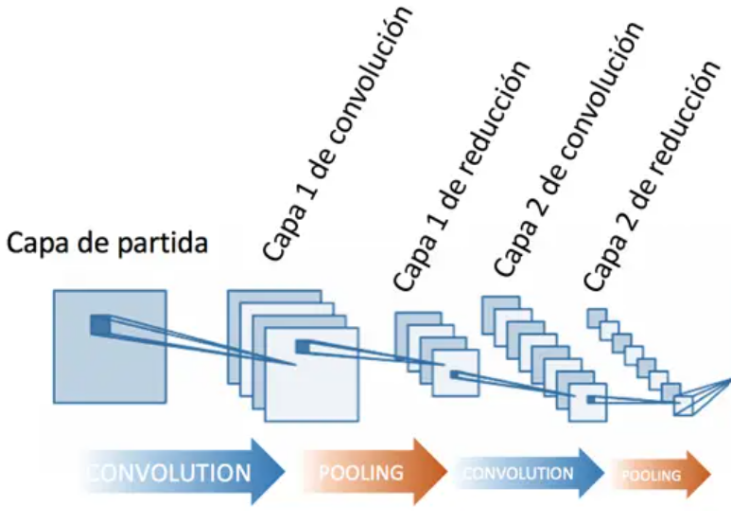


Figura 4.6: Resumen fase de feature learning.
Fuente: link

4.3. Fase de clasificación. Capas totalmente conectadas

Tras la fase de feauture learning se obtienen una serie de mapas de características que recogen la información importante de la imagen de entrada y que serán la entrada de la fase de clasificación.

La última fase de las redes neuronales convolucionales va a consistir en un perceptrón multicapa totalmente conectado como los que ya se estudió en el capítulo anterior. Este perceptrón multicapa funcionará como un clasificador, y su capa de salida tendrá tantas neuronas como categorías se quieran clasificar.

Como ya sabemos, la entrada de los perceptrones multicapa es un vector, por lo que primero es necesario transformar las matrices de los mapas de características a vectores (flattening).

Para la clasificación en la última capa del perceptrón se pueden usar distintas funciones de activación, pero la más habitual es la **Softmax**.

4.3.1. Función de activación Softmax

La función softmax es una generalización de la regresión logística. Soporta sistemas de clasificación multinomial, por lo que se convierte en el recurso principal utilizado en las capas de salida de un clasificador. La función Softmax calcula la distribución de probabilidades de las k categorías diferentes, es decir, calculará las probabilidades de que la imagen pertenezca a cada categoría y la clasificará en la que tenga mayor probabilidad esperada.

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.

$$\widehat{y_j} = a_j^{[L]} = softmax\left(z_j^{[L]}\right) = \frac{e^{z_j^{[L]}}}{\sum_{i=1}^{k} e^{z_i[L]}} \ \forall\ j = 1, \ldots, k$$

Donde L es el número de capas de la fully-connected.

Con todo esto, se está listo para abordar la parte práctica del trabajo.

4.4. Uso de las CNN para la clasificación de enfermedades de la vid

La parte práctica se llevará a cabo en MATLAB y los códigos se pueden consultar en los anexos.

4.4.1. Objetivo y dataset utilizado

Se ha escogido el dataset de Kaggle *Grapevine Disease Images*:
`https://www.kaggle.com/datasets/piyushmishra1999/plantvillage-grape/`

Grape Dataset			
Fig	**Class Name**	**Cause of disease**	**Total samples**
1	Black Rot	Fungus	1180
2	Esca		1383
3	Leaf blight		1076
4	Healthy	--------	423

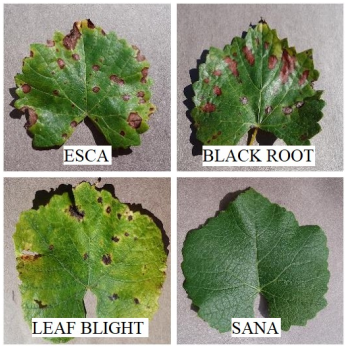


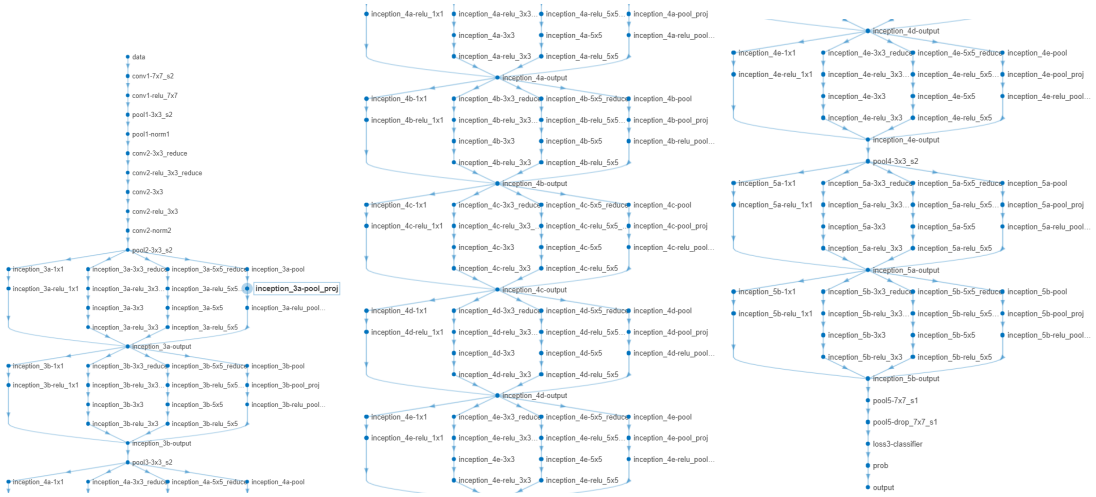
Figura 4.7: Contenido del Dataset
Fuente: Elaboración propia.

Este dataset contiene 4062 imágenes, todas de 256x256 píxeles de hojas de la vid sanas o infectadas por hongos: 423 de ellas son **sanas**, 1180 con **Podredumbre negra** (Black Rot), 1383 con **Esca** y 1076 con **Tizón** (leaf blight).

Como ya se puede leer en el título de la sección, el objetivo será clasifcar imáganes de hojas de la vid en las cuatro categorías que se han mencionado anteriormente. Para ello se hará uso de las CNN «Googlenet» y «Resnet50» de Matlab, y una más simple propgramada desde cero.

4.4.2. Googlenet

Googlenet [21] es una red neuronal convolucional, introducida por el ingeniero de Google Christian Szegedy en 2014, que revolucionó la clasificación de imagenes debido a su profundidad y ahorro de recursos computancionales respecto a las demás redes del momento. Esta red se ha ido actualizando con los años y en la actualidad es capaz de clasificar hasta 1000 categorías distintas.



(a) Parte 1. (b) Parte 2. (c) Parte 3.

Figura 4.8: Arquitectura Googlenet.
Fuente: Elaboración propia.

Su arquitectura consta con 144 capas y es algo más compleja que las estudiadas en teoría ya que involucra una nueva arquitectura denominada **módulos Inception**. La idea detrás del módulo Inception es la de mejorar la eficiencia computacional y la precisión de la red neuronal convolucional, utilizando múltiples capas convolucionales (con filtros de diferentes tamaños) y combinándolas en paralelo, en lugar de ejecutarlas siempre secuencialmente. De esta manera, la red puede capturar características de diferentes tamaños en la imagen de entrada, lo que puede ayudar a mejorar la precisión de la clasificación y reducir el sobreajuste.

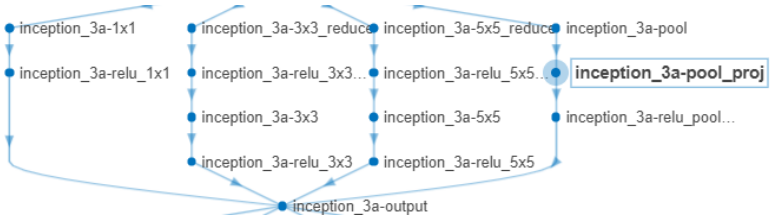
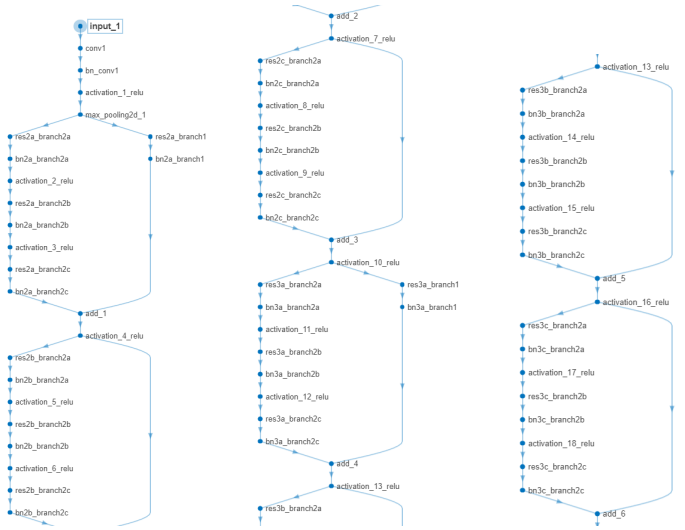


Figura 4.9: Módulo Inception 3a de Googlenet.
Fuente: Elaboración propia.

Esta estructura junto con nuevas técnicas de optimización la convierte en una red «poderosísima» para la clasificación y detección de objetos en imágenes, llegando a ganar concursos como el «Large-Scale Visual Recognition Challenge» (ILSVRC14).

4.4.3. Resnet50

Resnet50 [22] es una de las variantes más populares de las redes ResNet. Las CNN ResNet introducen una mejora en el aprendizaje de la red utilizando conexiones residuales en sus capas. En lugar de apilar simplemente las capas convolucionales una encima de la otra, ResNet utiliza bloques residuales que permiten que la información fluya a través de «atajos» desde las capas anteriores hasta las capas posteriores. Esto soluciona problemas como el desvanecimiento del gradiente que realentizan el entrenamiento.



(a) Parte 1. (b) Parte 2. (c) Parte 3.

Figura 4.10: Parte de la Arquitectura Resnet50.
Fuente: Elaboración propia.

4.4.4. CNN programada desde cero

Constará de 15 capas: Una de entrada para matrices de 256×256, 3 bloques convolucionales intermedios (formados a su vez, por una capa convolucional, una de normalización,

una de ReLu y una de Maxpooling), una capa fully-connected con 4 neuronas (se quiere clasificar 4 categorías) y una capa de clasificación softmax.

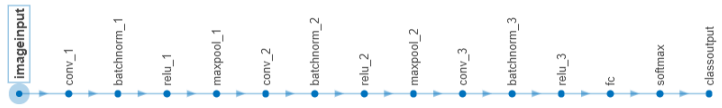


Figura 4.11: Arquitectura CNN hecha desde cero.
Fuente: Elaboración propia.

4.4.5. Procedimiento

Primero de todo, se ha separado el conjunto de datos en conjunto de etrenamiento (70 %), conjunto de validación (20 %) y conjunto de prueba (10 %).

Googlenet y Resnet50

Se modificó la resolución de las imágenes del dataset ya que las redes Googlenet y Resnet50, por defecto, aceptan como entrada matrices de $244 \times 244 \times 3$ y las imágenes de nuestro dataset se corresponden con matrices $256 \times 256 \times 3$.

Después, se ha modificado ligeramente la arquitectura de las redes, ya que ambas están diseñadas para clasificar hasta 1000 categorías distintas y en este caso sólo se necesitan 4 (Sana, Podredumbre negra, Esca y Tizón). Para esto sólo se ha tenido que cambiar las capas 142 y 144 en el caso de Googlenet y las capas 175 y 177 en el caso de Resnet50.

CNN programada desde cero

Se programó capa por capa, especificándose en cada una de ellas, el tamaño y número de filtros, el padding, el stride etc.

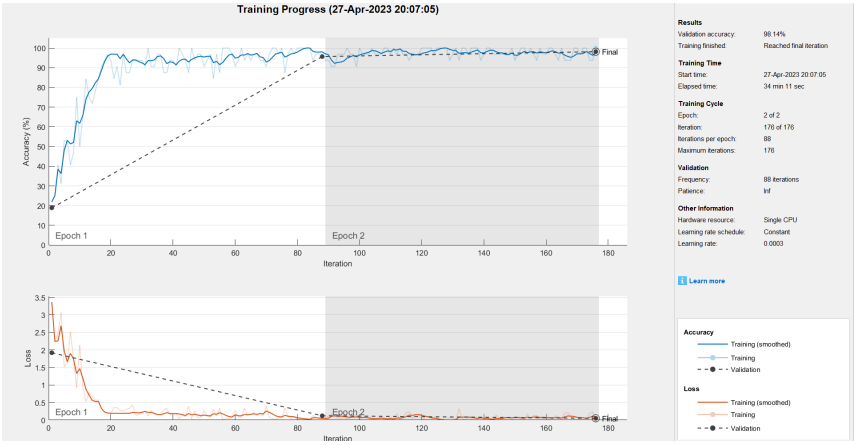
Entrenamiento

Por último, se ha entrenado a las tres redes con el conjunto de entrenamiento y validación especificándose los siguientes parámetros:

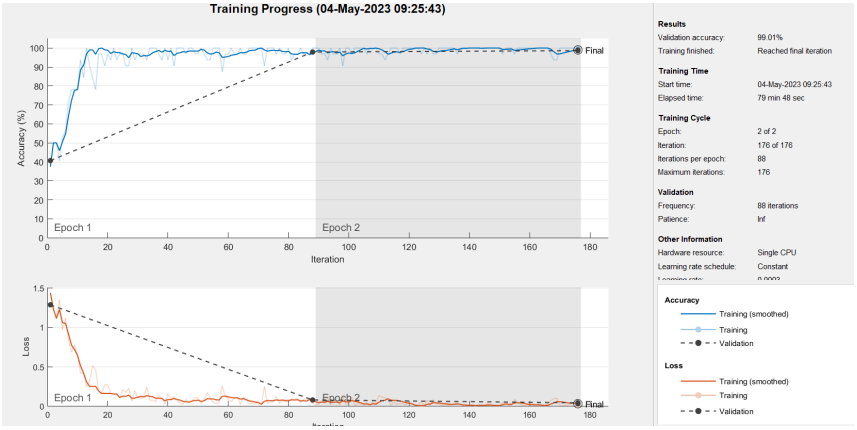
- **Método de optimización**: «sgdm» (stochastic gradient descent with momentum).
- «**MiniBatchSize**» (tamaño del minibatch): 32
- «**InitialLearnRate**» (tasa de aprendizaje inicial): $3 \cdot 10^{-4}$
- «**Shuffle**»: Se mezcla el conjunto de entranamiento y validación en cada época.

- «**MaxEpochs**»: 2 (número de «pasadas» por el conjunto de entrenamiento completo).

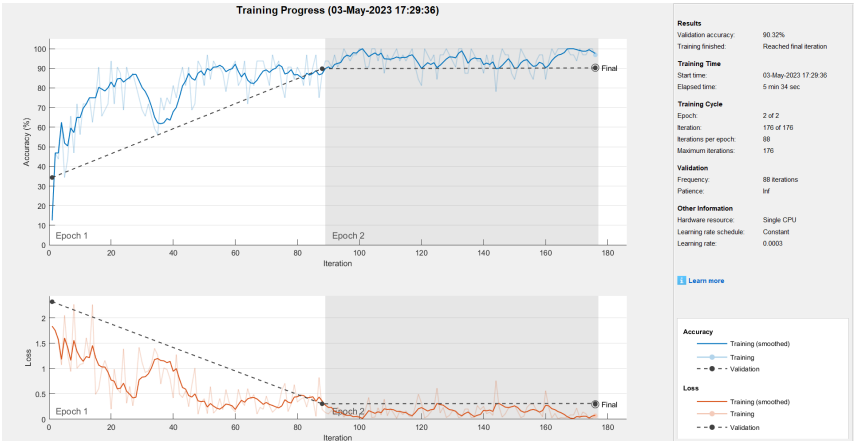
Se puede ver el progreso del entrenamiento de las tres redes en las siguientes imágenes:



(a) Entrenamiento Googlenet.



(b) Entrenamiento Resnet.



(c) Entrenamiento CNN hecha desde cero.

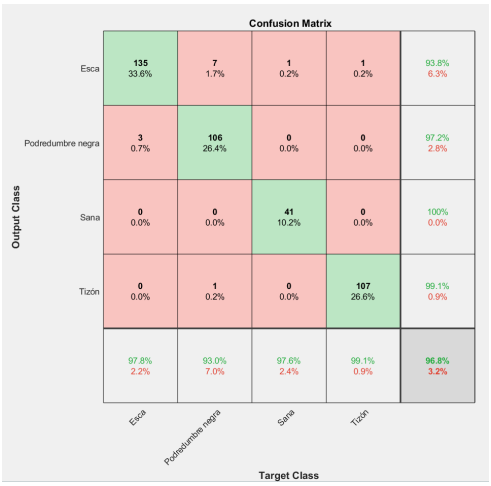
Figura 4.12: Entrenamiento. Fuente: Elaboración propia.

La imagen de arriba muestra el gráfico de cómo se ha desempeñado el entrenamiento

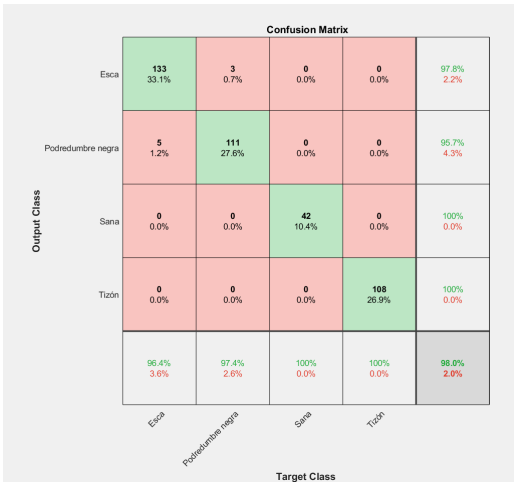
de las tres redes.

Se ha especificado un máximo de 2 etapas de entrenamiento para las tres. Cada etapa ha realizado 88 iteraciones para entrenarlas, de tal manera que en los gráficos superiores se puede ver cómo va mejorando la precisión conforme se va aumentando el número de iteraciones de entrenamiento. Una vez entrenada y determinada la red, se aplica a los datos de validación y se obtiene una precisón de 98.14 %, 99.01 % y 90.32 % para Googlenet, Resnet50 y la CNN hecha desde cero respectivamente. Por el contrario, en los gráficos inferiores, se puede ver cómo disminuye la función de pérdida en función de las iteraciones, convergiendo a cero.

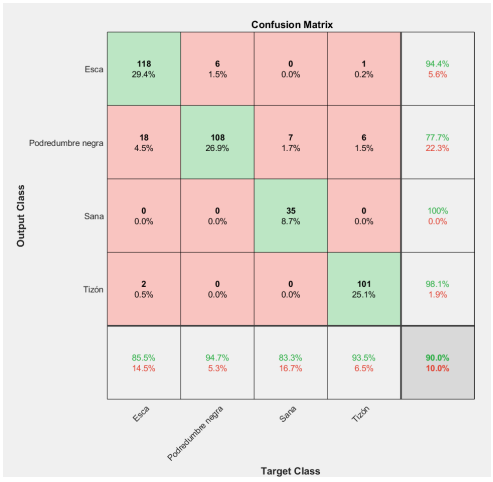
4.4.6. Resultados



(a) Matriz de confusión de la clasificación con Googlenet.



(b) Matriz de confusión de la clasificación con Resnet50.



(c) Matriz de confusión de la clasificación con la CNN hecha desde cero.

Figura 4.13: Resultados.
Fuente: Elaboración propia.

Una de las principales preocupaciones era el desbalanceo de la clase «Sanas», habiendo la mitad de muestras de esta clase frente a cada una de las demás. Esto puede afectar el proceso de generalización de la información y perjudicar la clasificación de las clases minoritarias. En este supuesto caso, una solución habría sido el «oversampling», creándose más imágenes ficticias de la clase minoritaria a partir de las ya existentes, mediante rotaciones, zoom, cambio en la iluminación, etc. Finalmente, el desbalanceo no ha supuesto ningún problema, ya que como se puede observar, se han conseguido precisiones superiores al 90 % para la clase «Sanas».

Cabe destacar que el entrenamiento de las redes Googlenet y Resnet50, podría haberse finalizado antes, ya que desde la primera época, los modelos se ajustaban perfectamente al problema. No obstante, se han especificado los mismos criterios de entrenamiento para las 3 y así poder comparar rendimientos mejor.

Se puede observar que las redes Googlenet y Resnet50, tardan mucho más tiempo en entrenarse que la CNN programada desde cero. Esto es debido a que las dos primeras son mucho más complejas y profundas, y por tanto, se tienen que ajustar muchos más parámetros que en la tercera. Gracias a esta profundidad y complejidad también se obtienen precisiones mayores, como era de esperar.

En definitiva, si se prioriza la precisión a la velocidad del entrenamiento, las redes Googlenet y Resnet50 son la mejor opción, ya que obtienen resultados muy cercanos al 100 %. Por otro lado, si se busca un «mix» entre buen rendimiento y velocidad, la CNN programada desde cero también es una muy buena alternativa. Si bien es cierto que para problemas más complejos, esta última podría «quedarse corta», se ha visto que para problemas comunes de clasificación puede ser una herramienta de gran utildiad.

CAPÍTULO 5

Conclusiones finales

Este trabajo ha proporcionado una comprensión profunda de la intersección entre los campos de visión artificial e inteligencia artificial, destacando el papel crucial de las redes neuronales convolucionales en el procesamiento y análisis de imágenes.

Además de explorar los conceptos básicos de la visón artificial, se ha introducido el fundamento matemático de las redes neuronales, incluyendo conceptos clave como las neuronas, las conexiones ponderadas, las funciones de activación y la retropropagación. Se ha profundizado específicamente en las redes neuronales convolucionales (CNN), estudiando su estructura y resaltando su capacidad para extraer características relevantes de las imágenes y su eficacia en la clasificación de imágenes.

La implementación, en la práctica, de tres redes convolucionales para clasificar imágenes ha evidenciado el potencial y la aplicabilidad de esta tecnología en la resolución de problemas reales. Se ha comparado el rendimiento entre dos redes convolucionales avanzadas y una sencilla y poco profunda, demostrando que las redes más simples también pueden servir de gran utilidad y lograr un gran desempeño en tareas de clasifición comunes.

En resumen, este TFG ha proporcionado una visión integral de cómo las redes neuronales artificiales y la visión artificial se complementan entre sí y ha permitido comprender el papel fundamental de las redes convolucionales en el análisis y clasificación de imágenes. Este trabajo abre nuevas puertas para futuras investigaciones y aplicaciones en el campo de la visión artificial, como la detección de objetos dentro de una imagen o la segmentacion de videos, destacando el potencial ilimitado de las redes neuronales artificiales en este ámbito en constante evolución.

APÉNDICE A

Aplicación filtros de suavizado

A.1. Filtro media y Gauss

[25]

```
% Filtro de suavizado media y Gaussiano
% Cargamos la imagen
im=imread('facultad_mates.jpg');
% Introducimos ruido
fn = imnoise(im,'salt & pepper', 0.05);

% GAUSS
r = fn (:,:,1) ;
g = fn (:,:,2) ;
b = fn (:,:,3) ;

gaussian = fspecial("gaussian", [3,3],4) ;

r = imfilter(r,gaussian);
g = imfilter(g,gaussian);
b = imfilter(b,gaussian);

g1 = cat(3,r,g,b);

% MEDIA
media=imfilter(fn, fspecial('average'));

figure;
imshow(im);
title("Imagen original");

figure;
imshow(fn);
```

```
29 title ("Imagen con ruido");
30
31 figure;
32 imshow(g1);
33 title ("Aplicando Gauss 3x3");
34
35 figure;
36 imshow(media);
37 title ("Aplicando media 3x3");
```

A.2. Filtro mínimo

```
1  % Carga la imagen
2  im = imread('facultad_mates.jpg');
3  % Anadimos ruido de tipo sal
4  img = imnoise(im,'gaussian', 0.4);
5
6
7  % Muestra la imagen original
8  figure;
9  imshow(img);
10 title('Imagen con ruido de tipo sal');
11
12 % Define el tamano del filtro
13  filtro_size  = 3;
14
15 % Separa los canales RGB
16 R = img(:,:,1);
17 G = img(:,:,2);
18 B = img(:,:,3);
19
20 % Aplica el filtro minimo (elimina la sal)
21 % a cada canal
22 R_min = ordfilt2(R, 1, ones( filtro_size ));
23
24 G_min = ordfilt2(G, 1, ones( filtro_size ));
25
26 B_min = ordfilt2(B, 1, ones( filtro_size ));
27
28 % Crea la imagen RGB filtrada
29 min_img = cat(3, R_min, G_min, B_min);
30
31 figure;
32 imshow(min_img);
33 title(sprintf('Filtro minimo de %d x %d', ...
34       filtro_size , filtro_size ));
```

APÉNDICE B

Aplicación filtros de detección de bordes

```
% Cargamos la imagen
im=imread('facultad_mates.jpg');

% Convertimos a escala de grises
I=rgb2gray(im);
figure,imshow(I);

% Filtros detectores de bordes
B1 = edge(I,"Roberts");
B2 = edge(I,"Sobel");
B3 = edge(I,"Prewitt");

% Representaciones de las imagenes
% Crear la figura con cuatro subimagenes
figure;
subplot(2,2,1),subimage(I), title ...
    ('Imagen original en escala de grises');
axis off

subplot(2,2,2),subimage(B1),title('Filtro de Roberts');
axis off

subplot(2,2,3),subimage(B2),title('Filtro de Sobel');
axis off

subplot(2,2,4),subimage(B3),title('Filtro de Prewitt');
axis off

% Filtro de Sobel paso a paso
% Sobel direccionales
SX = [-1,0,1;-2,0,2;-1,0,1];
SY = [-1,-2,-1;0,0,0;1,2,1];

```

```
34 B5 = imfilter(I,SX);
35 B6 = imfilter(I,SY);
36
37 % Juntamos ambos y normalizamos para obtener el filtro de Sobel
38 % Segunda parte: calculamos el valor total del gradiente
39 S = abs(B5)+ abs(B6);
40
41 % Valor maximo del gradiente
42 Vmax = max(max(S));
43
44 % Normalizamos el gradiente a 255
45 SN = (S/Vmax)*255;
46 SN = uint8(SN);
47
48 figure;
49 subplot(2,2,1),subimage(I),...
50       title('Imagen original en escala de grises');
51 axis off
52
53 subplot(2,2,2),subimage(B5),title('Sobel Horizontal');
54 axis off
55
56 subplot(2,2,3),subimage(B6),title('Sobel Vertical');
57 axis off
58
59 subplot(2,2,4),subimage(SN),title('Sobel');
60 axis off
61
62
63 % Creamos el filtro laplaciano a mano:
64 L =[0,1,0;1,-4,1;0,1,0];
65 B7 = imfilter(I,L);
66 figure,imshow(B7);
```

APÉNDICE C

Uso de las CNN para la clasificacion de enfermedades de la vid

C.1. Googlenet

```
1
2  Dataset = imageDatastore("Dataset", ...
3      "IncludeSubfolders", true, ...
4       "LabelSource", "foldernames");
5
6  % La carpeta Dataset tiene sub carpetas,
7  % de ahi includesubfolders = true
8  % Ademas el nombre de la etiqueta,
9  % sera el nombre de la subcarpeta a la que
10 % pertenezca
11 % Dividimos el conjunto de datos en datos de
12 % entrenamento, validacion y test
13
14 [Training_Dataset, Validation_Dataset ,...
15     Test_Dataset] = ...
16     splitEachLabel(Dataset,0.7, 0.2, 0.1);
17
18 % Cargamos la red GoogleNet sin entrenar y
19 % la guardamos en la variable net
20 net = googlenet;
21 % Veamos la estructura de googlenet,
22 % vemos que la entrada son matrices de
23 % 224x224x3, sin embargo todas nuestras
24 % imagenes del dataset, son de
25 % 2556x256 pixeles, por tanto seran
26 % matrices de 256x256x3.
27 analyzeNetwork(net)
28
29 % Cambiemos el tamano de las imagenes
30 % del dataset para que no den problemas
```

```
% Todas las imagenes de entrada son RGB,
% asi que solo nos tenemos que
% precupar del tamano de las 2 primeras dimensiones
Input_Layers_Size = net.Layers(1).InputSize(1:2);
Resized_Training_Image = augmentedImageDatastore...
    (Input_Layers_Size,  ...
     Training_Dataset);
Resized_Validation_Image = augmentedImageDatastore...
    (Input_Layers_Size,  ...
     Validation_Dataset);
Resized_Test_Image = augmentedImageDatastore...
    (Input_Layers_Size,  ...
     Test_Dataset);

% Si nos fijamos en la arquitectura de nuevo
% de la red googlenet, podemos
% ver que en la capa fully conected(142) y en
% la capa de salida (144)) esta
% disenada para clasificar 1000 categorias
% diferentes,pero en nuestro trabajo solo
% estamos interesados en clasificar 4, asi
% que esto tambien lo tenemos que modificar

Feature_Learner = net.Layers(142);
Output_Classifier = net.Layers(144);

% Guardar el numero de categorias,
% en este caso 4, pero lo hacemos general
Number_of_Classes = numel...
    (categories(Training_Dataset.Labels));

% Creamos una nueva capa fully conected
% adaptada a nuestro trabajo
New_Feature_Learner = fullyConnectedLayer...
    (Number_of_Classes, ...
    "Name", "Grape Leaf Disease Learner", ...
    "WeightLearnRateFactor", 10, ...
    "BiasLearnRateFactor", 10);

% Creamos una nueva capa de clasificacion
% adaptada a nuestro trabajo
New_Classifier_Layer =classificationLayer("Name", ...
    "Grape Leaf Disease Classifier");

% Actualizamos las dos capas que hemos creado
% dentro de la estrcutura de la red Googlenet,
% para ello creamos un nuevo layer graph, para
% crear nuestra propia red a partir de googlenet,
%almacenada en la variable net

Layer_Graph = layerGraph(net);
New_Layer_Graph = replaceLayer(Layer_Graph,...
    Feature_Learner.Name, New_Feature_Learner);
New_Layer_Graph = replaceLayer(New_Layer_Graph, ...
```

```
85      Output_Classifier.Name, ...
86      New_Classifier_Layer);
87  analyzeNetwork(New_Layer_Graph)
88
89  % Entrenemos la nueva red
90  Size_of_Minibatch = 32;
91  Validation_Frequency = floor(numel(...
92      Resized_Training_Image.Files)/Size_of_Minibatch);
93
94  % sgdm es descenso del gradiente estocastico
95  % por minibatches
96  % minibatch ya sabemos lo que es,
97  % subconjunto del entrenamiento para
98  % actualizar los pesos
99  % una epoca es un ciclo de entrenamiento
100 % completo en todo el conjunto de entrenamiento
101 % shuffle en cada epoca, para no coger
102 % siempre los mismos minibatches
103
104 Training_Options = trainingOptions("sgdm", ...
105     "MiniBatchSize", Size_of_Minibatch, ...
106     "MaxEpochs", 2, ...
107     "InitialLearnRate", 3e-4, ...
108     "Shuffle","every-epoch", ...
109     "ValidationData", Resized_Validation_Image, ...
110     "ValidationFrequency", Validation_Frequency, ...
111     "Verbose", false, "Plots", "training-progress");
112
113 net = trainNetwork(Resized_Training_Image, ...
114     New_Layer_Graph, Training_Options);
115
116 % Probemos la red con el conjunto de test
117 [Label_Predicted, Probability]= ...
118      classify (net,Resized_Test_Image);
119
120 % Categorias reales del dataset de test
121 Test_Labels = Test_Dataset.Labels;
122
123 % Matriz de confusion
124 figure
125 plotconfusion(Test_Labels,Label_Predicted)
126
127 % Mostremos algunas pocas predicciones
128 numImages = 9;
129 idx = randperm(numel(Resized_Test_Image.Files),...
130     numImages);
131
132 figure
133 tiledlayout("flow")
134 for i = 1:numImages
135      nexttile
136     imshow(Resized_Test_Image.Files{idx(i)});
137      title(char(Test_Labels(idx(i))) + ...
138          " Predicted as "+ ...
```

```
139         char(Label_Predicted(idx(i))))
140 end
```

C.2. Resnet50

```
1  Dataset = imageDatastore("Dataset",...
2      "IncludeSubfolders", true,  ...
3       "LabelSource", "foldernames");
4
5  [Training_Dataset, Validation_Dataset ,...
6      Test_Dataset] = ...
7      splitEachLabel(Dataset,0.7,  0.2,  0.1);
8
9  net = resnet50;
10 analyzeNetwork(net)
11
12 Input_Layers_Size = net.Layers(1).InputSize(1:2);
13 Resized_Training_Image = ...
14     augmentedImageDatastore(Input_Layers_Size, ...
15      Training_Dataset);
16 Resized_Validation_Image = ...
17     augmentedImageDatastore(Input_Layers_Size, ...
18      Validation_Dataset);
19 Resized_Test_Image = ...
20     augmentedImageDatastore(Input_Layers_Size, ...
21      Test_Dataset);
22
23 Feature_Learner = net.Layers(175);
24 Output_Classifier  = net.Layers(177);
25
26 Number_of_Classes = numel(...
27     categories(Training_Dataset.Labels));
28
29 New_Feature_Learner = ...
30     fullyConnectedLayer(Number_of_Classes, ...
31     "Name", "Grape Leaf Disease Learner", ...
32     "WeightLearnRateFactor", 10, ...
33     "BiasLearnRateFactor", 10);
34
35 New_Classifier_Layer =classificationLayer("Name", ...
36     "Grape Leaf Disease  Classifier");
37
38 Layer_Graph = layerGraph(net);
39 New_Layer_Graph = replaceLayer(Layer_Graph,...
40     Feature_Learner.Name, ...
41     New_Feature_Learner);
42 New_Layer_Graph = replaceLayer(New_Layer_Graph,...
43     Output_Classifier.Name, ...
44     New_Classifier_Layer);
45 analyzeNetwork(New_Layer_Graph)
```

```

Size_of_Minibatch = 32;
Validation_Frequency = floor(numel(...
    Resized_Training_Image.Files)/Size_of_Minibatch);


Training_Options = trainingOptions("sgdm", ...
    "MiniBatchSize", Size_of_Minibatch, ...
    "MaxEpochs", 2, ...
    "InitialLearnRate", 3e-4, ...
    "Shuffle","every-epoch", ...
    "ValidationData", Resized_Validation_Image, ...
    "ValidationFrequency", Validation_Frequency, ...
    "Verbose", false , "Plots", "training-progress");

net = trainNetwork(Resized_Training_Image,...
    New_Layer_Graph, Training_Options);


[Label_Predicted, Probability]= ...
     classify (net,Resized_Test_Image);

Test_Labels = Test_Dataset.Labels;


figure
plotconfusion(Test_Labels,Label_Predicted)

numImages = 9;
idx = randperm(numel(Resized_Test_Image.Files),numImages);

figure
tiledlayout ("flow")
for i = 1:numImages
    nexttile
    imshow(Resized_Test_Image.Files{idx(i)});
     title (char(Test_Labels(idx(i))) + " Predicted as "+ ...
         char(Label_Predicted(idx(i))))
end
```

C.3. CNN hecha por mi

[26]

```

Dataset = imageDatastore("Dataset",...
    "IncludeSubfolders", true, ...
     "LabelSource", "foldernames");

[Training_Dataset, Validation_Dataset, Test_Dataset] = ...
```

```
    splitEachLabel(Dataset,0.7, 0.2, 0.1);

% Para este caso, programemos nosotros una
% sencilla paso por paso. Sera una
% CNN con tres bloques covolucionales,
% cada uno de ellos formado por una
% capa de convolucion, otra de relu y otra de max pooling

net = [
    % Definimos la entrada de tamano 256x256,
    % que es el tamano que tienen
    % todas las imagenes del dataset
    imageInputLayer([256 256 3])

    % Primer bloque convolucional
    % 8 filtros de 3x3 con padding
    convolution2dLayer(3,8,"Padding","same")

    % Normaliza los datos para la relu
    batchNormalizationLayer

    % Func de activacion relu, para solo mostrar
    % las caracteristiicas que nos interesan
    reluLayer

    % El maxpooling es de 2x2 y stride de 2
    maxPooling2dLayer(2,"Stride",2)

    % Segundo bloque convolucional
    convolution2dLayer(3,16,"Padding","same")
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2,"Stride",2)

    % Tercer bloque convolucional
    convolution2dLayer(3,32,"Padding","same")
    batchNormalizationLayer
    reluLayer

    % Fase fully connected, con 4 neuronas,
    % ya que hay 4 categorias
    fullyConnectedLayer(4)

    % Funcion softxmax que te da prob de pertenecer
    % a cada una de las 10 clases
    softmaxLayer
    classificationLayer
];

analyzeNetwork(net)

Size_of_Minibatch = 32;
Validation_Frequency = floor(numel(...
    Training_Dataset.Files)/Size_of_Minibatch);
```

```
61
62 % Como sigue, igual que en las otras redes
63 option = trainingOptions("sgdm", ...
64     "MiniBatchSize", Size_of_Minibatch, ...
65     "MaxEpochs", 2, ...
66     "InitialLearnRate", 3e-4, ...
67     "Shuffle","every-epoch", ...
68     "ValidationData", Validation_Dataset, ...
69     "ValidationFrequency", Validation_Frequency, ...
70     "Verbose", false, "Plots", "training-progress");
71
72 net = trainNetwork(Training_Dataset, net, option);
73
74 [Label_Predicted, Probability]= classify (net,Test_Dataset);
75
76 Test_Labels = Test_Dataset.Labels;
77
78
79 figure
80 plotconfusion(Test_Labels,Label_Predicted)
81
82 numImages = 9;
83 idx = randperm(numel(Test_Dataset.Files),numImages);
84
85 figure
86 tiledlayout ("flow")
87 for i = 1:numImages
88     nexttile
89     imshow(Test_Dataset.Files{idx(i)});
90     title (char(Test_Labels(idx(i))) + " Predicted as "+ ...
91         char(Label_Predicted(idx(i))))
92 end
```

Índice de figuras

Referencias

[1] R.C. Gonzalez y R.E.Woods. *Digital Image Processing* (3a edición). Pearson Prentice Hall, 2007.

[2] R. Klette. *Concise Computer Vision*. Springer, 2014.

[3] R. Jain, R. Kasturi, B.G. Schunck. *Machine Vision, Chapter 5. Edge Detection (pp. 140-185)*. McGraw-Hill, Inc., ISBN 0-07-032018-7, 1995.

[4] A. González, F.J. Martínez de Pisón, A.V. Pernía Espinoza, F. Alba, M. Castejón, J. Ordieres y E. Vergara, *Técnicas y algoritmos básicos de la visión artificial*. Universidad de la Rioja: Servicio de publicaciones, 2006.

[5] J.F. Vélez, A.B. Moreno, A. Sánchez, y J. L. Esteban, *Visión por Computador* (2da edición). RA-MA, 2003.

[6] A.R. Paguay y P.R. Urgilés. *Recuperación de imágenes mediante extracción de Blobs aplicando el operador laplaciano de Gauss*. Universidad Politécnica Salesiana, 2012.

[7] C. F. Higham y D. J. Higham, *Deep learning: An introduction for applied mathematicians*, SIAM REVIEW, Vol. 61, No. 4, pp. 860–891, 2018.

[8] I. Goodfellow, Y. Bengio, y A. Courville. *Deep learning*. MIT press, 2016. `http://www.deeplearningbook.org`

[9] E. F. Caicedo y J. A. López. *Una aproximación práctica a las redes neuronales artificiales*. Alianza Editorial, 2009.

[10] *Tipos de imagen. MATLAB y Simulink. MathWorks*. `https://es.mathworks.com/help/matlab/creating_plots/image-types.html`

[11] *Procesamiento en el dominio espacial*. (Parte 2). Universidad de Sevilla. `http://asignatura.us.es/imagendigital/Tema2-ParteII_Filtros.pdf`

[12] *El concepto de la convolución en gráficos, para comprender las CNN*. J. Cuartas, 2021. https://josecuartas.medium.com/el-concepto-de-la-convoluci%C3%B3n-en-gr%C3%A1ficos-para-comprender-las-convolutional-neural-networks-cnn-519d2eee009c

[13] Correlación Cruzada. https://es.wikipedia.org/wiki/Correlaci%C3%B3n_cruzada

[14] *Padding, strides, max-pooling y stacking en las Redes Convolucionales*. https://www.codificandobits.com/blog/padding-strides-maxpooling-stacking-redes-convolucionales/

[15] *A Comprehensive Tutorial to learn Convolutional Neural Networks from Scratch*. https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

[16] *¿Qué son las redes neuronales convolucionales?*. https://es.mathworks.com/discovery/convolutional-neural-network-matlab.html

[17] *Introducción al aprendizaje automático. Cap 05.7. Redes Neuronales Convolucionales*. Grado Biotecnología UPM. https://dcain.etsin.upm.es/~carlos/bookAA/05.7_RRNN_Convoluciones_CIFAR_10_INFORMATIVO.html

[18] *Función de coste – Redes neuronales*. D. Calvo, 2018. https://www.diegocalvo.es/funcion-de-coste-redes-neuronales/

[19] *Filtrar imágenes*. https://es.mathworks.com/help/images/linear-filtering.html?s_tid=CRUX_lftnav

[20] L.S. Acosta. *Detección de bordes en una imagen*. Universidad de Jaén, 2015. http://www4.ujaen.es/~satorres/practicas/practica3_vc.pdf

[21] *GoogLeNet. Un artículo de La Máquina Oráculo*. https://lamaquinaoraculo.com/computacion/googlenet/

[22] *Tipos de arquitecturas de redes convolucionales* by KeepCoding. https://keepcoding.io/blog/tipos-arquitecturas-redes-convolucionales/

[23] *Grapevine Disease Images*. https://www.kaggle.com/datasets/piyushmishra1999/plantvillage-grape

[24] far1din. (13 de septiembre de 2022). *Visualizing Convolutional Neural Networks — Layer by Layer* [Archivo de Vídeo]. Youtube. https://www.youtube.com/watch?v=JboZfxUjLSk

[25] Nuruzzaman Faruqui. (6 de octubre de 2020). *Lesson 33: Gaussian Filter* [Archivo de Vídeo]. Youtube. https://www.youtube.com/watch?v=tlfmOOaA5ZU

[26] Atecnea. (25 de febrero de 2021). *Programar Red Neuronal Convolucional (CNN) en Matlab desde cero* [Archivo de Vídeo]. Youtube. https://www.youtube.com/watch?v=2YCUOrvgE9M

[27] Nuruzzaman Faruqui. (10 de mayo de 2021). *Fruit Classification using GoogleNet Convolutional Neural Network (CNN)* [Archivo de Vídeo]. Youtube. `https://www.youtube.com/watch?v=58-1KmsIEcQ`