UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN

Biblioteca
Universitas Miguel Hernández

"Redes Convolucionales. Aplicación a la clasificación de imágenes médicas"

TRABAJO FIN DE GRADO

Junio - 2023

AUTOR: Begoña Gómez Pujante

DIRECTOR: Antonio Peñalver Benavent

Biblioteca
UNIVERSITAS Miguel Hernández

Agradecimientos

Empecé esta carrera con muchas dudas, sin tener una clara visión de lo que me apasionaba y a lo que me quería dedicar en el futuro. Agradezco a esta carrera y a sus profesores por guiarme en este proceso de descubrimiento, permitiéndome ilusionarme con cada aprendizaje y encontrar mi vocación.

Cuando pienso en los agradecimientos, mis abuelos son los primeros que vienen a mi mente. Quiero expresarles mi agradecimiento por su constante apoyo y fe incondicional en mí, incluso cuando yo misma dudaba. Gracias por siempre interesaros en mi carrera, aunque solo supierais que era "algo relacionado con los ordenadores".

A mis padres, les agradezco estar siempre presentes en mis momentos de estrés académico, por estar al otro lado del teléfono después de cada examen y por madrugar cada día para llevarme. Y a mi hermano, gracias por aguantarme en mis momentos de desesperación durante las épocas de exámenes y por no enfadarse demasiado.

Esta etapa de mi vida no podría haberla superado sin Aina y Lorena. Vosotras habéis sido las mejores compañeras de carrera y ahora amigas que podría haber tenido. Muchas gracias por todas esas risas, los descansos nunca aburridos y compartir los nervios previos a cada examen. Sin duda, vosotras sois lo mejor que me llevo de esta etapa.

No solo debo agradecerles a ellas, sino también a todos mis increíbles compañeros que me han acompañado en este camino y me han brindado ayuda siempre que la he necesitado.

Agradezco a todas mis amigas, María, Paula, Abril, Claudia, Elena y Carmen, quienes siempre han estado presentes tanto en persona como a distancia, apoyándome, escuchándome y comprendiendo cuando no podía quedar o me agobiaba. Saber que con una simple llamada o mensaje estarían ahí ha sido el mejor apoyo que he podido tener.

Y, por último, pero no menos importante, quiero expresar mi agradecimiento a mi tutor Antonio Peñalver, gracias por su orientación constante y dedicación a este proyecto.

Resumen

En este trabajo de investigación, se ha abordado el estudio de las redes convolucionales y su aplicación en la clasificación de imágenes médicas. El objetivo principal ha sido adquirir un mayor conocimiento sobre esta técnica de aprendizaje automático y desarrollar un modelo de clasificación efectivo.

Para lograrlo, se ha realizado un exhaustivo estudio teórico sobre las redes convolucionales, comprendiendo su arquitectura, funcionamiento y aplicaciones en el campo de la visión por computadora. Se ha explorado la literatura científica y se han analizado diferentes enfoques y metodologías utilizadas en trabajos previos.

Posteriormente, se ha procedido a la creación de un modelo de clasificación de imágenes médicas utilizando una red convolucional. Se ha empleado un dataset de tumores cerebrales, que ha sido debidamente procesado y dividido en conjuntos de entrenamiento, validación y evaluación. A través del uso de la biblioteca PyTorch y la plataforma de desarrollo colaborativo Google Colab, se ha implementado y entrenado el modelo, ajustando los hiperparámetros y optimizando su rendimiento.

Además de la creación y entrenamiento del modelo, se ha llevado a cabo la implementación de una interfaz de usuario interactiva utilizando la plataforma Anvil. A través de esta interfaz, se ha brindado una solución intuitiva y accesible para que los usuarios puedan cargar imágenes y obtener predicciones sobre el tipo de tumor cerebral presente en ellas.

Biblioteca
UNIVERSITAS Miguel Hernández

Biblioteca
UNIVERSITAS Miguel Hernández

Índice

Índice de ilustraciones

Índice de tablas

Biblioteca
UNIVERSITAS Miguel Hernández

CAPÍTULO 1: INTRODUCCIÓN

Biblioteca
UNIVERSITAS Miguel Hernández

1.1. Introducción a la Inteligencia Artificial

Durante miles de años, hemos tratado de entender cómo pensamos; es decir, entender cómo un simple puñado de materia puede percibir, entender, predecir y manipular un mundo mucho más grande y complicado que ella misma. El campo de la Inteligencia Artificial, o IA, va más allá: no sólo intenta comprender, sino que también se esfuerza en construir entidades inteligentes [1].

En los años 50 nace la IA, cuando un grupo de pioneros de la computación comenzaron a preguntarse si se podía hacer que las computadoras pensaran [2]. Una definición concisa de la IA podría ser: el proceso de creación de sistemas y tecnologías capaces de realizar tareas que requieren inteligencia humana, como el aprendizaje, la toma de decisiones, el reconocimiento de voz y de imagen o la traducción de idiomas.

La IA abarca en la actualidad una gran variedad de subcampos, pero sus principales son el aprendizaje automático (Machine Learning) y el aprendizaje profundo (Deep Learning).

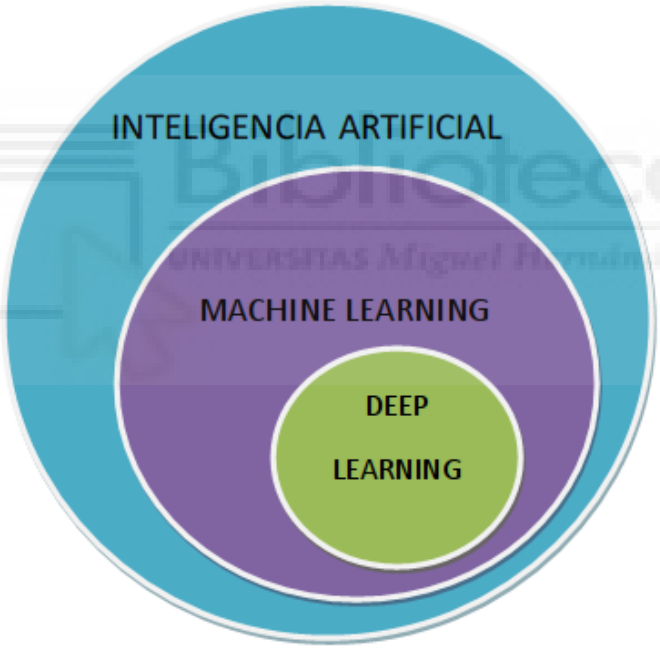


Ilustración 1: Subconjuntos de la IA

El Machine Learning o Aprendizaje automático se refiere a un amplio conjunto de técnicas informáticas que nos permiten dar a las computadoras la capacidad de aprender sin ser explícitamente programadas. Hay muchos tipos diferentes de algoritmos de Aprendizaje automático, como por ejemplo los algoritmos de agrupamiento o los árboles de decisión. Actualmente, los algoritmos más populares dentro de este campo son los de Deep Learning [2].

El Deep Learning o aprendizaje profundo es un subcampo dentro del Machine Learning, el cuál utiliza modelos de redes neuronales artificiales con múltiples capas para aprender a partir de datos. Estos modelos son capaces de aprender de manera autónoma y mejorar su rendimiento a medida que se les proporciona más información.

1.2. Introducción a las Redes Neuronales

Las redes neuronales artificiales (RNA) surgen como un intento para emular ciertas características propias de los humanos, como la capacidad de memorizar y de asociar hechos. Si se examinan con atención aquellos problemas que no pueden expresarse a través de un algoritmo, se observará que todos ellos tienen una característica en común: la experiencia. Así, parece claro que una forma de aproximarse al problema consista en la construcción de sistemas que sean capaces de reproducir esta característica humana[3].

Emular redes neuronales de manera artificial no es un desarrollo reciente, se hicieron algunos intentos antes del advenimiento de los computadores, pero su verdadero desarrollo tuvo lugar [4] en los años 40, cuando el neurólogo Warren McCulloch y el matemático Walter Pitts propusieron el primer modelo de red neuronal artificial.

Entre las décadas de los 50 y 60 el científico Frank Rosenblatt, inspirado en el trabajo de Warren McCulloch y Walter Pitts, creó el Perceptrón, la unidad desde donde nacería y se potenciarían las redes neuronales artificiales [5].

Luego de un periodo inicial de entusiasmo, las redes neuronales cayeron en un periodo de frustración y desprestigio [4] en el año 1969 debido a las limitaciones identificadas en algunas publicaciones de Marvin Minsky y Seymour Papert.
Durante esta etapa el soporte económico y computacional era muy limitado y sólo unos pocos investigadores consiguieron logros de algún nivel de importancia [4].

Este periodo llegó a su fin en el año 1982, cuando se produjo un resurgimiento de las redes neuronales gracias al redescubrimiento de una técnica llamada backpropagation (retropropagación), que permitía entrenar con éxito redes neuronales multicapa. La backpropagation junto con el descenso del gradiente se convirtieron en la columna vertebral y la potencia de las redes neuronales.

A partir de entonces, se han hecho grandes avances en el campo de las redes neuronales, y se han utilizado en una amplia gama de aplicaciones, incluyendo el reconocimiento de voz, el procesamiento de imágenes y el procesamiento del lenguaje natural.

1.2.1. La Neurona

Una neurona es una célula viva y está constituida por los mismos elementos que conforman las células biológicas. Una de las características que diferencian a las neuronas del resto de células vivas, es su capacidad de comunicación [4]. En términos generales, la neurona recibe estímulos o señales en sus entradas, y cuando la suma de estas señales alcanza un determinado nivel de activación, la neurona se dispara o activa, enviando una señal a través de su axón.

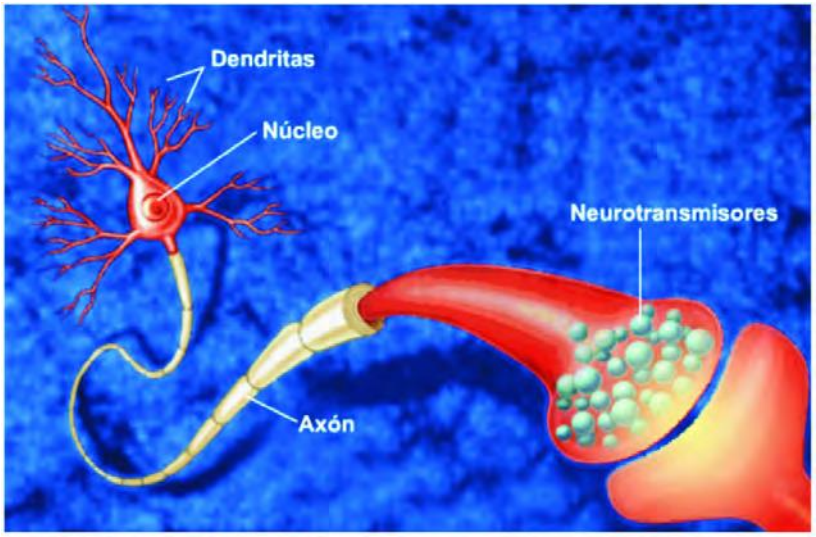


Ilustración 2: La neurona biológica

Las neuronas artificiales se basan en la estructura de la neurona biológica y emplean funciones matemáticas de valores reales para emular su funcionamiento.

Cada una de estas neuronas simples, va a tener una forma similar al siguiente diagrama [2]:

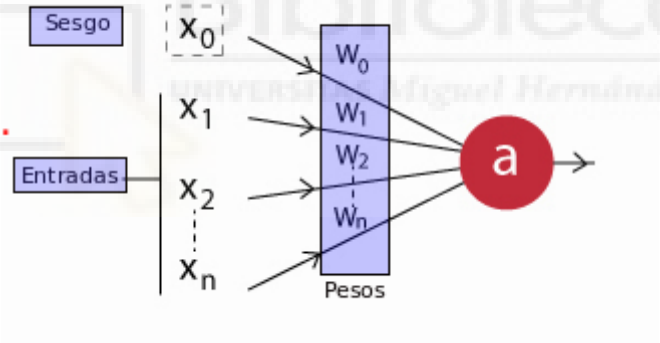


Ilustración 3: Modelo de neurona artificial

Los componentes de esta son los siguientes:

- $x1, x2, \ldots, xn$: Los datos de entrada en la neurona, los cuales también puede ser que sean producto de la salida de otra neurona de la red [2].

- x0 : La unidad de sesgo; un valor constante que se le suma a la entrada de la función de activación de la neurona. Generalmente tiene el valor 1. Este valor va a permitir cambiar la función de activación hacia la derecha o izquierda, otorgándole más flexibilidad para aprender a la neurona [2].

- $w0, w1, w2, \ldots, wn$: Los pesos relativos de cada entrada. Tener en cuenta que incluso la unidad de sesgo tiene un peso [2].

- a: La salida de la neurona. Que va a ser calculada de la siguiente forma [2]:

$$a = f(\sum_{i=0}^{n} w_i \cdot x_i)$$

Aquí f es la función de activación de la neurona. Esta función es la que les otorga tanta flexibilidad a las redes neuronales y le permite estimar complejas relaciones no lineales en los datos. Puede ser tanto una función lineal, una función logística, hiperbólica, etcétera [2].

1.2.2. La red

Existen diferentes maneras de organizar las neuronas en una red neuronal. Las redes neuronales pueden ser organizadas en capas, que son grupos de neuronas que procesan la información de forma secuencial. Hay tres tipos principales de capas en las redes neuronales:

- Capa de entrada: es la capa que recibe la información de entrada en la red, como una imagen, un texto o una señal de audio. Las neuronas de esta capa no procesan la información, simplemente la transmiten a la siguiente capa.

- Capas ocultas: son las capas intermedias entre la capa de entrada y la capa de salida. En estas capas, las neuronas procesan la información recibida de la capa anterior y generan nuevas representaciones de esta.

- Capa de salida: es la capa final de la red, que produce la salida o respuesta de la red en función de la información que ha sido procesada en las capas anteriores. La salida puede ser una clasificación, una predicción, una traducción, una imagen generada, etc.

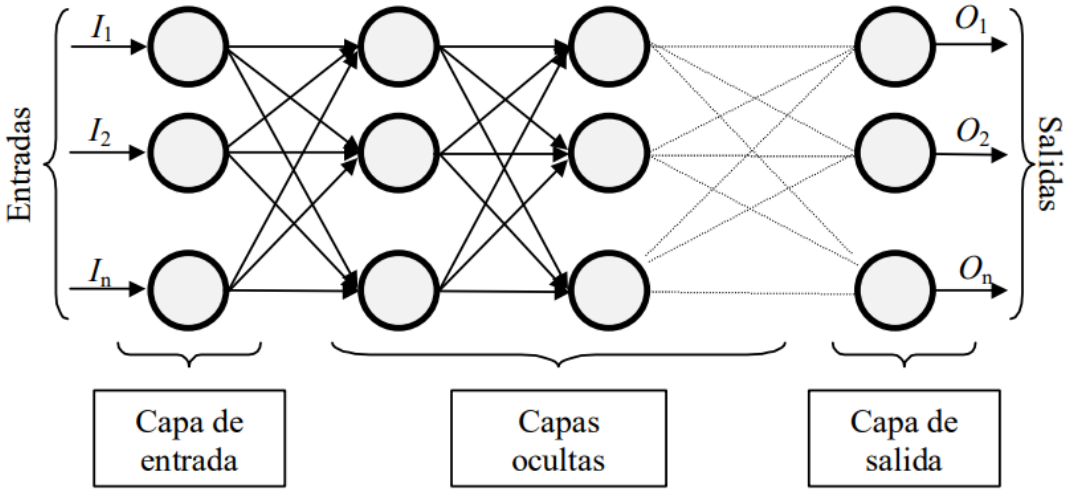


Ilustración 4: Estructura de red neuronal por capas

La cantidad de capas y neuronas en cada capa dependerá del problema que se esté abordando y la complejidad de la tarea. Las redes neuronales más simples pueden tener una sola capa oculta, mientras que las más complejas pueden tener muchas capas ocultas y millones de neuronas.

Añadir más capas a una red neuronal puede permitir aprender características más complejas y sutiles de los datos de entrada, lo que puede mejorar su rendimiento en tareas complejas. Sin embargo, agregar demasiadas capas también puede conducir a problemas de overfitting o sobreajuste (ajuste excesivo), donde la red neuronal se ajusta demasiado a los datos de entrenamiento y no generaliza bien a nuevos datos.

1.2.3. Funciones de activación

Una neurona biológica puede estar activa o inactiva; es decir, que tiene un "estado de activación". Las neuronas artificiales también tienen diferentes estados de activación; algunas de ellas solamente dos, al igual que las biológicas, pero otras pueden tomar cualquier valor dentro de un conjunto determinado [3].

La función activación calcula el estado de actividad de una neurona; transformando la entrada global (menos el umbral, Θi) en un valor (estado) de activación, cuyo rango normalmente va de 0 a 1 o de –1 a 1. Esto es así, porque una neurona puede estar totalmente inactiva (0 o –1) o activa (1) [3].

La función de activación ayuda a introducir la no linealidad en la salida de una neurona artificial. Sin una función de activación, la salida de una neurona estaría limitada a una transformación lineal de la entrada, lo que significa que la neurona simplemente produciría una combinación lineal de las entradas sin realizar ninguna operación no lineal adicional.

La introducción de una función de activación no lineal permite a la neurona realizar operaciones no lineales en las entradas y producir salidas no lineales en respuesta. Esto es importante para una amplia gama de aplicaciones de aprendizaje automático, ya que muchos fenómenos del mundo real, como las relaciones entre las variables, no son lineales.

Existen varios tipos de funciones de activación utilizados en redes neuronales, entre los cuales se incluyen:

- **Escalonada**

  La función de activación escalonada mapea cualquier valor de entrada a una salida binaria, que generalmente es 0 o 1. La función escalonada se puede expresar como:

  $$f(x) = 1 \text{ si } x > 0$$
  $$0 \text{ si } x \leq 0$$

  Esta función tiene un umbral en $x = 0$. Si la entrada es mayor que 0, la salida es 1, de lo contrario, la salida es 0. La función escalonada es una función simple pero limitada en su capacidad para modelar funciones complejas. Debido a su simplicidad, se utiliza principalmente en redes neuronales con dos clases de salida, donde se quiere clasificar una entrada como perteneciente a una clase o a otra.

Es importante destacar que la función de activación escalonada no es diferenciable en $x = 0$, lo que puede limitar su uso en algunos algoritmos de aprendizaje automático que requieren la derivada de la función de activación, como el algoritmo de backpropagation utilizado para entrenar redes neuronales.

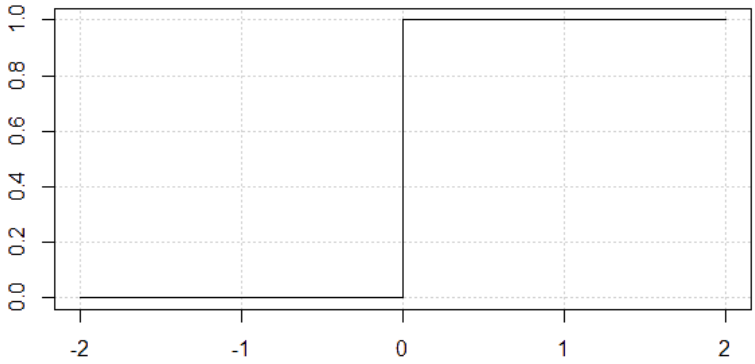


Ilustración 5: Representación gráfica de la función de activación escalonada

- **Sigmoide**

La función de activación sigmoidea (también llamada función logística) toma cualquier valor real como entrada y genera un valor en el rango (0, 1). Se calcula de la siguiente manera: [6]

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

donde x es el valor de salida de la neurona. A continuación, se puede ver el gráfico de la función sigmoidea cuando la entrada se encuentra en el rango (-10, 10) [6]:

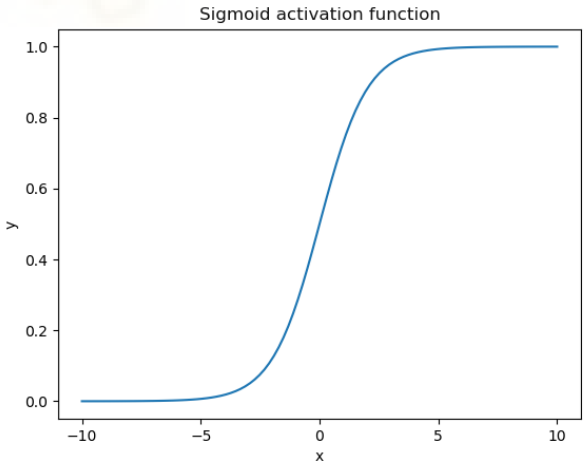


Ilustración 6: Representación gráfica de la función de activación sigmoide

Como era de esperar, la función sigmoidea no es lineal y limita el valor de una neurona en el rango pequeño de (0, 1). Cuando el valor de salida es cercano a 1, la neurona está activa y permite el flujo de información, mientras que un valor cercano a 0 corresponde a una neurona inactiva [6].

Además, una característica importante de la función sigmoidea es el hecho de que tiende a empujar los valores de entrada a cualquiera de los extremos de la curva (0 o 1) debido a su forma de S. En la región cercana a 0, si se cambia ligeramente el valor de entrada, los cambios respectivos en la salida son muy grandes y viceversa. Para entradas inferiores a -5, la salida de la función es casi cero, mientras que para entradas superiores a 5, la salida es casi uno [6].

Finalmente, la salida de la función de activación sigmoidea puede interpretarse como una probabilidad ya que se encuentra en el rango [0, 1] [6].

- **Tangente hiperbólica (TANH)**

Otra función de activación que es común en el Deep Learning es la función hiperbólica tangente, simplemente denominada función tanh. Se calcula de la siguiente manera [6]:

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

Se observa que la función tanh es una versión desplazada y estirada del sigmoide. A continuación, se puede ver su gráfico cuando la entrada está en el rango (-10, 10) [6]:

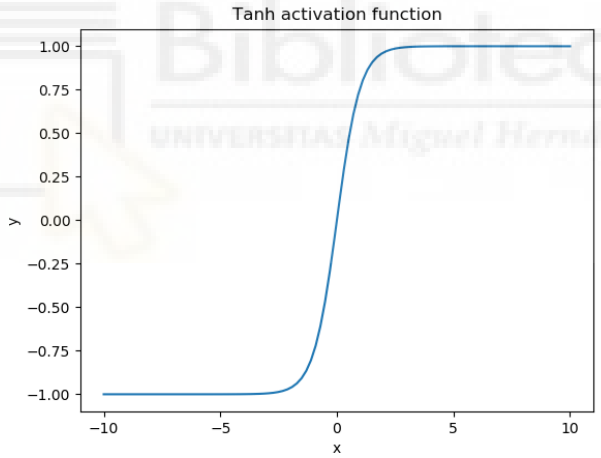


Ilustración 7:Representación gráfica de la función de activación tanh

El rango de salida de la función tanh es (-1, 1) y presenta un comportamiento similar al de la función sigmoide. La principal diferencia es el hecho de que la función tanh empuja los valores de entrada a 1 y -1 en lugar de 1 y 0 [6].

El uso de esta función da como resultado valores más altos de gradiente durante el entrenamiento y actualizaciones más altas en los pesos de la red que la función de activación sigmoide. Entonces, si se quieren gradientes fuertes y grandes pasos de aprendizaje, se usará la función de activación tanh [6].

Otra diferencia es que la salida de tanh es simétrica alrededor de cero, lo que conduce a una convergencia más rápida [6].

- **ReLU, o unidad lineal rectificada**

  La función de activación ReLU (Unidad Lineal Rectificada) devuelve 0 si la entrada es negativa y la entrada misma si es positiva. Al utilizar esta función en redes neuronales, se obtiene una tasa de cálculo más rápida que otras funciones de activación como la sigmoide y la tangente hiperbólica (tanh).

  Además, la función ReLU evita problemas de difusión de gradiente que ocurren cuando se usan funciones de activación saturadas. La difusión de gradiente se refiere a la disminución de la magnitud del gradiente durante el proceso de backpropagation, lo que puede llevar a problemas de aprendizaje lento o incluso a la desaparición del gradiente. Al ser una función no saturada, la función ReLU no sufre de estos problemas y permite una propagación eficiente del gradiente.

  Matemáticamente se expresa como:

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

  Y su representación gráfica es la siguiente:

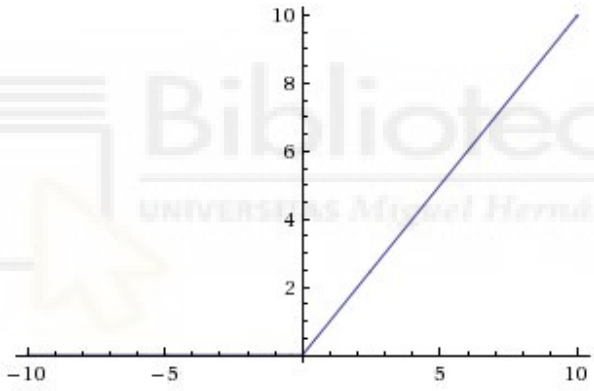


Ilustración 8: Representación gráfica de la función de activación ReLU

- **Swish**

  La función de activación Swish [7] es una extensión de la función de activación SILU que se propuso en el artículo "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning" [8]. La fórmula de SILU es $f(x) = x * \text{sigmoid}(x)$, donde $\text{sigmoid}(x) = 1 / (1 + e\text{^} - x)$. La ligera modificación que se hace en la fórmula Swish es la adición de un parámetro β entrenable, lo que la convierte en $f(x) = x * \text{sigmoid}(\beta x)$ [9].

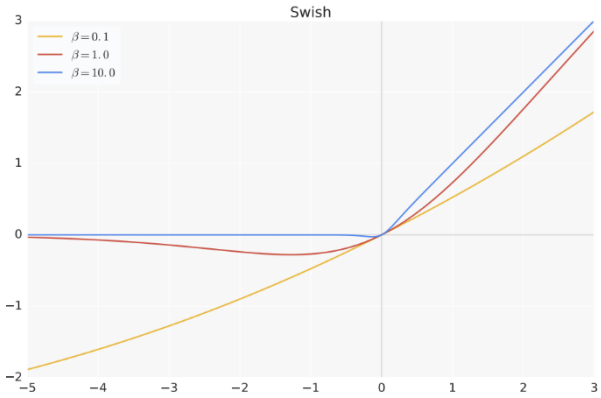


Ilustración 9: Representación gráfica de la función de activación Swish

Esta función surge como una mejora a la función de activación ReLU, que tiene limitaciones en la transmisión de ciertos gradientes negativos. Swish es una función suave y continua que permite la propagación de gradientes negativos y tiene un parámetro entrenable que permite una mejor afinación de la función de activación para una mayor propagación de información y gradientes más suaves, lo que facilita la optimización y generalización en redes neuronales profundas.

- **SineReLU**

SineReLU [10] es una función de activación alternativa a la función de activación ReLU, que se basa en una combinación de la función sinusoidal y la función ReLU, y se define como:

$$f(x) = \max(0, x) + \alpha * \sin(\max(0, x))$$

Esta función de activación surge como una alternativa a la función de activación ReLU con la intención de mejorar algunas de sus limitaciones, como el problema de "Dying ReLU". El problema se produce cuando los gradientes fluyen hacia valores cercanos a cero y, por lo tanto, los pesos no se actualizan durante el entrenamiento de la red neuronal [10]. La función SineReLU ayuda a resolver este problema al permitir que los gradientes fluyan suavemente a través de la función, lo que facilita la actualización de los pesos durante el entrenamiento, tal y como se puede observar en la siguiente gráfica:

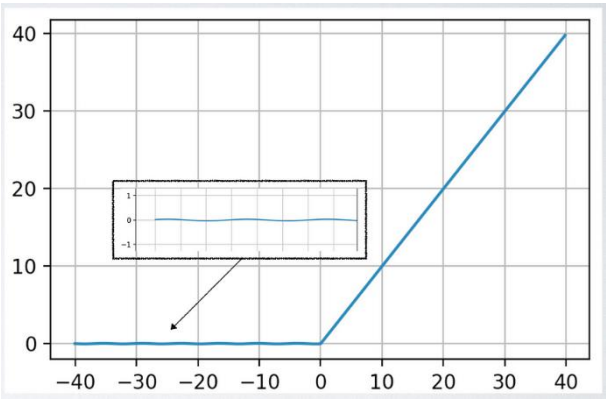


Ilustración 10: Representación gráfica de la función de activación SineReLU

La idea detrás de SineReLU es combinar la no linealidad suave y continua de la función sinusoidal con la no linealidad escalonada de ReLU, lo que permite la propagación de gradientes y la activación de un mayor número de neuronas. Además, SineReLU tiene un parámetro de ajuste que permite ajustar el grado de no linealidad de la función, lo que facilita la optimización de la red neuronal y la mejora del rendimiento en tareas de clasificación y reconocimiento de imágenes.

1.2.4. Aprendizaje de una red neuronal

El aprendizaje en una red neuronal está estrechamente relacionado con la forma en que aprendemos en nuestras vidas y actividades habituales: realizamos una acción y un entrenador nos acepta o corrige para comprender cómo mejorar en una determinada tarea. De manera similar, las redes neuronales requieren de un entrenador para describir lo que debería haberse producido como respuesta a la entrada [11].

Las fases del aprendizaje de una red neuronal son las siguientes:

- **Fase 1: Propagación hacia delante (Forwardpropagation)**

  Esta fase ocurre cuando la red es expuesta a los datos de entrenamiento y estos atraviesan toda la red neuronal para que sus predicciones (etiquetas) sean calculadas. Es decir, se pasa los datos de entrada a través de la red de tal manera que todas las neuronas apliquen su transformación a la información que reciben de las neuronas de la capa anterior y la envíen a las neuronas de la siguiente capa. Cuando los datos han pasado por todas las capas y todas sus neuronas han realizado sus cálculos, se llegará a la capa final con un resultado de predicción de etiqueta para esos ejemplos de entrada [12].

- **Fase 2: Función de pérdida**

  Al entrenar una red neuronal, es necesario saber cuán lejos se está de la respuesta correcta. Para medir esto, se hace uso de una función de pérdida que calcula la diferencia entre la salida predicha y la salida esperada. El objetivo es minimizar esta pérdida, es decir, hacer que la salida predicha se acerque lo más posible a la salida esperada. Para hacer esto, los pesos y sesgos de la red se ajustan gradualmente durante el proceso de entrenamiento.

  La red neuronal pasa por varias iteraciones de entrenamiento, y en cada iteración, los pesos se ajustan para reducir la pérdida. El objetivo final es que la pérdida sea lo más cercana a 0 posible, lo que significa que el modelo es capaz de hacer predicciones precisas.

  La fórmula para calcular la pérdida es la siguiente:

$$f(m, b) = \frac{1}{N}\sum_{i=1}^{n}(y_i - (mx_i + b))^2$$

- **Fase 3: Backpropagation**

  Después de calcular la pérdida de la red neuronal, se realiza un proceso llamado propagación hacia atrás. Esto significa que se retroalimenta la información de pérdida desde la salida de la red hacia todas las neuronas que participaron directamente en la generación de la salida.

  Sin embargo, las neuronas de la capa oculta solo reciben una fracción de la señal total de pérdida, basada en la contribución relativa que cada neurona ha aportado a la salida original. Este proceso se repite capa por capa, hasta que todas las neuronas de la red han recibido una señal de pérdida que describe su contribución relativa a la pérdida total [12].

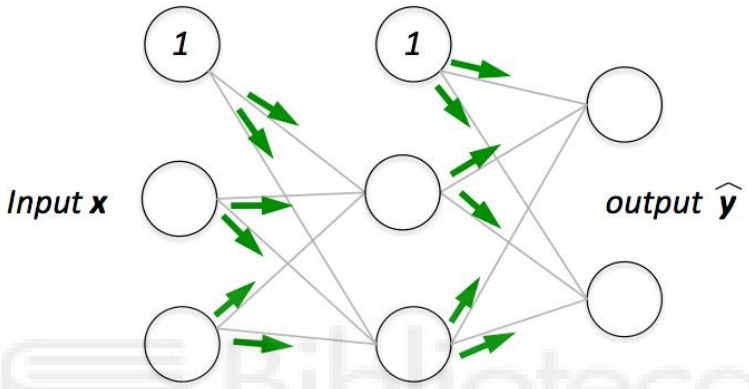


Ilustración 11: Representación gráfica del Backpropagation

Visualmente, se pueden resumir las fases del aprendizaje de una red neuronal explicadas anteriormente con la siguiente imagen:

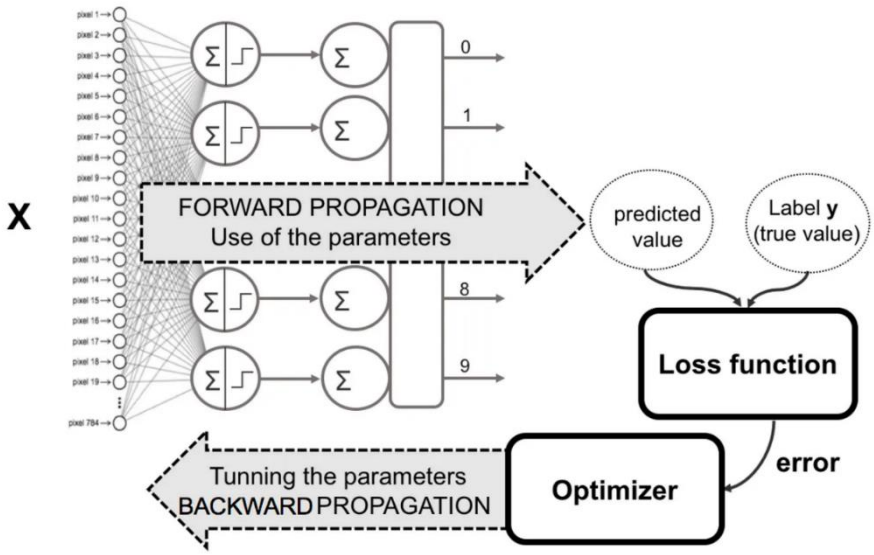


Ilustración 12: Fases del aprendizaje de una red neuronal

Una vez propagada la información del error hacia atrás en la red neuronal, se puede ajustar los pesos de las conexiones entre neuronas [12]. Lo que se intenta hacer es ajustar los pesos de la red para que las predicciones se acerquen lo más posible a las etiquetas correctas. Para lograr esto, se utiliza una técnica de optimización llamada descenso del gradiente.

El descenso del gradiente es un algoritmo de optimización utilizado comúnmente 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 [13].

Si se traza una gráfica en la que se representa a la pérdida como función de un peso sináptico w_{ij} concreto, se puede comprobar cómo los mínimos y máximos locales de la función se hallan precisamente allí donde la derivada o gradiente de la función respecto a dicho peso es nula [13]:

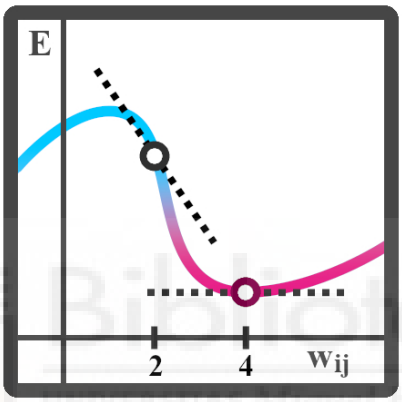


Ilustración 13: Ejemplo gráfico del descenso del gradiente

En esta ilustración se representa el valor de la derivada de la función E respecto a un peso sináptico en dos puntos distintos de la curva. Cuando el peso sináptico vale 2, la derivada de E es negativa (pendiente descendente), mientras que cuando vale 4, su derivada es cero (pendiente nula). Este último punto es el objetivo para alcanzar, puesto que en él la función E ha alcanzado un mínimo local [13].

Para que la función de pérdida se acerque a un mínimo, se debe desplazar iterativamente a través de su curva en la dirección inversa señalada por el gradiente. Si la derivada de la función en el punto es positiva (pendiente ascendente) se debe disminuir el valor del peso sináptico. Si es negativa (pendiente descendente), se debe incrementarlo [13].

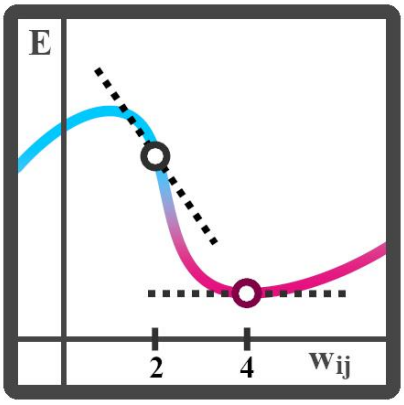


Ilustración 14: Ejemplo gráfico 2 del descenso del gradiente

En este ejemplo, dado que el gradiente es negativo en $w_{ij} = 2$, el peso sináptico debe incrementarse paso a paso hasta alcanzar el punto de gradiente nulo. Esto es lo que se conoce como descenso del gradiente [13].

1.2.5. Tipos de Redes Neuronales

Las redes neuronales se pueden clasificar según su arquitectura, que es determinada por la organización y disposición de las neuronas en la red.
A continuación, se explican brevemente algunos de estos tipos de redes:

Perceptrón simple

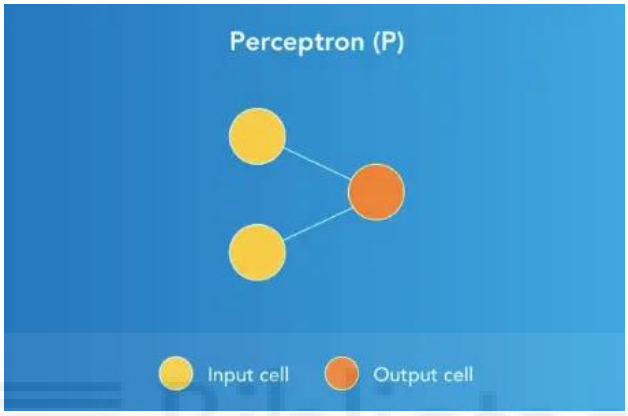


Ilustración 15: Perceptrón simple

El Perceptrón fue el primer modelo de RNA presentado a la comunidad científica por el psicólogo Frank Rosenblatt en 1958. Como es natural despertó un enorme interés en la década de los años sesenta, debido a su capacidad para aprender a reconocer patrones sencillos con una superficie de separación lineal [4].

La estructura del Perceptrón es una red monocapa pues la capa de entrada, al no realizar procesamiento alguno sobre la señal de estímulo, no se tiene en cuenta y, por tanto, esta red posee solo una capa de procesamiento que es la misma de salida. Posee conectividad total, ya que la neurona de la capa de salida está conectada a todas las unidades o neuronas de entrada [4].

Perceptrón multicapa

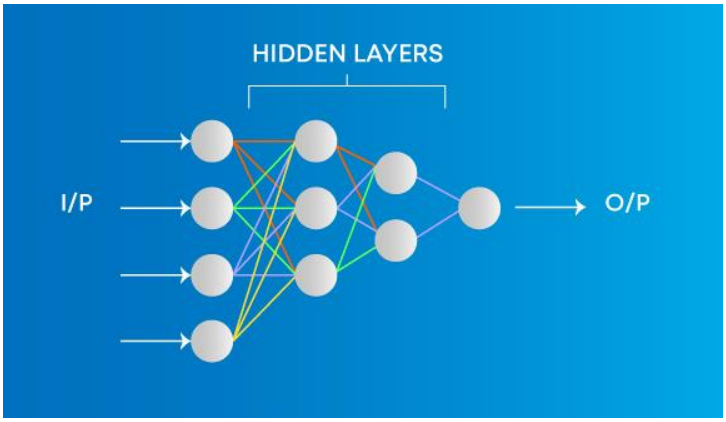


Ilustración 16: Perceptrón multicapa

Uno de los principales inconvenientes que tiene el perceptrón simple es su incapacidad para separar regiones que no sean linealmente separables. Sin embargo, Rosenblatt ya intuía que un perceptrón multicapa sí podía solucionar este problema pues, de esta manera, se podían obtener regiones de clasificación mucho más complejas, debido a una estructura que incluye, además de las capas de entrada y salida, una o más capas internas u ocultas [4].

En un perceptrón multicapa, las entradas y los pesos sinápticos se combinan y luego se procesan a través de la función de activación para producir la salida de la neurona, al igual que en el perceptrón. Pero la diferencia es que cada combinación lineal se propaga a la siguiente capa.

Cada capa alimenta a la siguiente con el resultado de su cálculo, su representación interna de los datos. Esto recorre todo el camino a través de las capas ocultas hasta la capa de salida [14].

Finalmente, la salida de la última capa se calcula y se compara con la salida deseada para ajustar los pesos de las conexiones mediante un proceso de backpropagation del error, con el fin de reducir la pérdida.

Red Neuronal Convolucional (CNN)

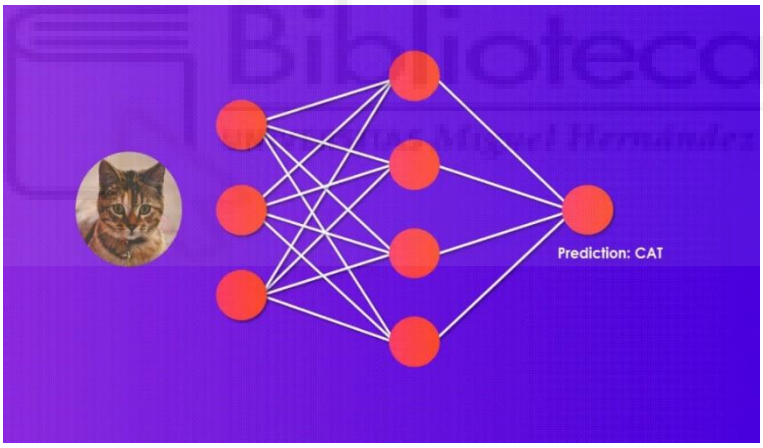


Ilustración 17: Red neuronal convolucional (CNN)

Las redes neuronales convolucionales son muy similares a las redes neuronales ordinarias como el perceptrón multicapa; se componen de neuronas que tienen pesos y sesgos que pueden aprender. Cada neurona recibe algunas entradas, realiza un producto escalar y luego aplica una función de activación. Al igual que en el perceptrón multicapa también se va a tener una función de pérdida o costo sobre la última capa, la cual estará totalmente conectada [2].

Lo que diferencia a las redes neuronales convolucionales es que suponen explícitamente que las entradas son imágenes, lo que permite codificar ciertas propiedades en la arquitectura; permitiendo ganar en eficiencia y reducir la cantidad de parámetros en la red [2].

Red Neuronal de Base Radial (RBF)

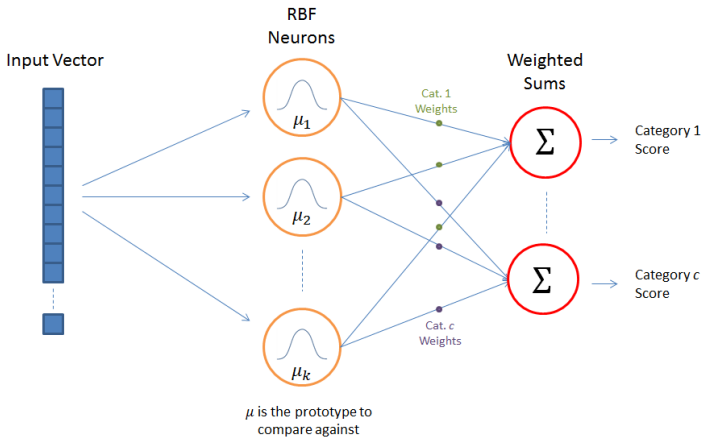


Ilustración 18: Red de base radial (RBF)

Las redes de función de base radial (RBF) son un tipo de red neuronal artificial comúnmente utilizado para problemas de aproximación de funciones. Las redes de función de base radial se distinguen de otras redes neuronales debido a su aproximación universal y su mayor velocidad de aprendizaje [15].

Una red RBF es un tipo de red neuronal de avance compuesta por tres capas, la capa de entrada, la capa oculta y la capa de salida. Cada una de estas capas tiene diferentes tareas [15]. La primera capa corresponde a las entradas de la red, la segunda es una capa oculta que consta de una serie de unidades de activación no lineal RBF y la última corresponde a la salida final de la red [16].

Las funciones de activación en las redes de función de base radial se implementan convencionalmente como funciones gaussianas [16].

Red Neuronal Recurrente (RNN)

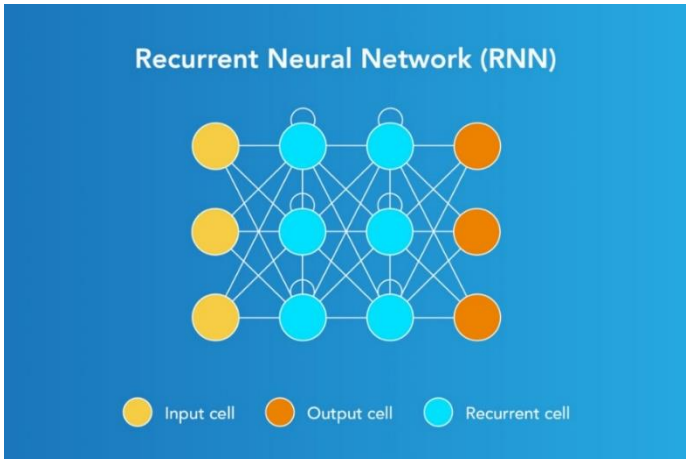


Ilustración 19: Red neuronal recurrente (RNN)

Una red neuronal recurrente es una red neuronal que está especializada en procesar una secuencia de datos con un índice de paso de tiempo que va desde 1 a τ. Para tareas que involucran entradas secuenciales, como el habla y el lenguaje, a menudo es mejor usar RNN [17].

Los RNN se llaman recurrente porque realizan la misma tarea para cada elemento de una secuencia, y la salida depende de los cálculos anteriores. Otra forma de pensar en los RNN es que tienen una "memoria" que captura información sobre lo que se ha calculado hasta el momento [17].

Red Neuronal LSTM (del inglés long short-term memory)

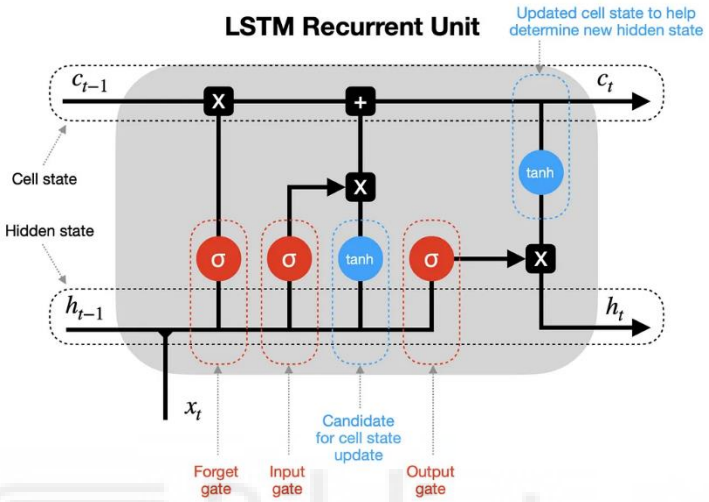


Ilustración 20: Red neuronal LSTM

Una red neuronal LSTM (Long Short-Term Memory) es un tipo especial de red neuronal recurrente (RNN) que puede manejar información a largo plazo y evitar el problema del gradiente desvaneciente en el aprendizaje de secuencias largas.

Una red neuronal LSTM típica tiene algo llamado "celda de memoria". La celda de memoria puede retener alguna información sobre la secuencia, permitiendo básicamente recordar las características importantes al principio de la secuencia que pueden tener efecto en las partes posteriores de la secuencia, en lugar de calcular la salida basándose solo en el paso de tiempo anterior [18].

Los principales componentes de LSTM son sus compuertas. Hay tres compuertas en una LSTM: la compuerta de entrada, la compuerta de olvido y la compuerta de salida. La compuerta de entrada controlará la entrada de nueva información en la celda. La compuerta de olvido controlará el contenido de la memoria, es decir, decidirá si se olvida una pieza de información para poder almacenar nueva información. La compuerta de salida controlará cuándo se utiliza la información en la salida de la celda [18].

Red Neuronal Transformer

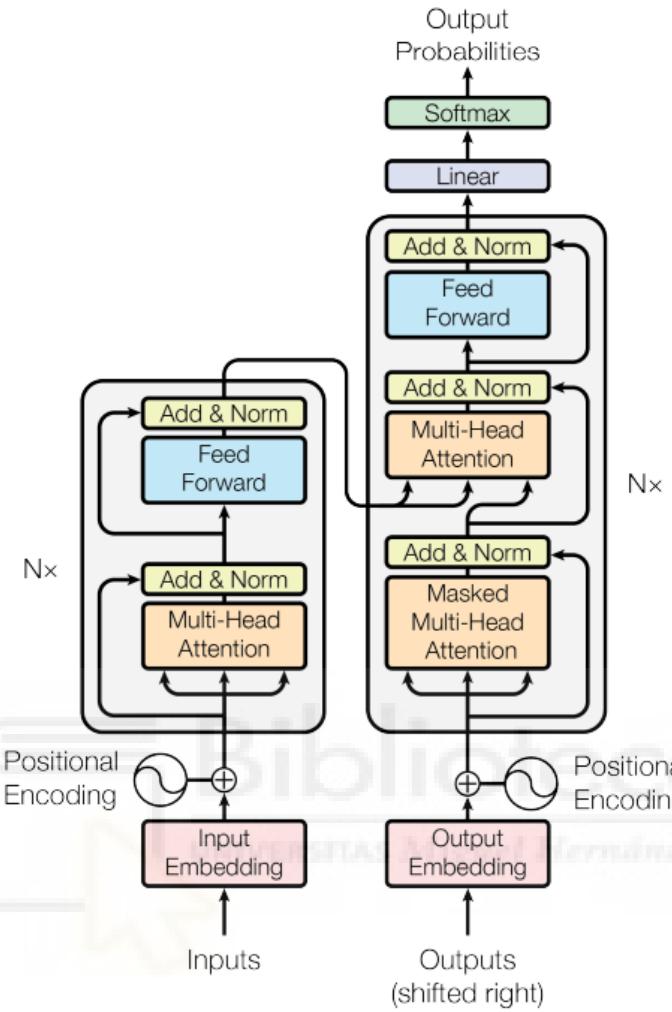


Ilustración 21: Red Neuronal Transformer

Las redes neuronales Transformer fueron introducidas en el artículo "Attention is All You Need" por Vaswani [19] como una alternativa a las RNN para tareas de procesamiento de lenguaje natural y otros problemas de secuencia. Aunque las RNN han sido durante mucho tiempo la elección predominante para el modelado de secuencias, presentan algunos problemas, como dificultad para capturar dependencias a muy largo plazo y la imposibilidad de paralelizar el cálculo secuencial, lo que hace que el entrenamiento e inferencia sean lentos.

La arquitectura Transformer funciona mediante el uso de un mecanismo de auto-atención (intra-atención), eliminando todas las operaciones recurrentes que se encuentran en el enfoque anterior [14]. Este mecanismo permite al modelo aprender a identificar patrones importantes dentro de una secuencia de entrada.

Sin embargo, dado que no se utilizan operaciones recurrentes, se debe agregar información adicional a la entrada y salida para que la red pueda entender el orden de las secuencias. Esta información adicional se llama codificación posicional y se utiliza para ayudar a la red a entender la relación entre los diferentes elementos de la secuencia.

1.3. Técnicas de clasificación de imágenes

Con el creciente uso de tecnologías, la cantidad de datos que se generan en forma de imágenes ha aumentado exponencialmente, lo que ha creado una necesidad cada vez mayor de herramientas y técnicas para analizar y procesar esta información visual.

Entre estas técnicas, la clasificación de imágenes es una de las más importantes, ya que nos permite identificar patrones y características en grandes conjuntos de imágenes, lo que a su vez nos ayuda a tomar decisiones informadas y a extraer información valiosa de los datos visuales.

Las redes convolucionales se han convertido en una de las técnicas de clasificación de imágenes más populares y efectivas en la actualidad. Sin embargo, existen otras técnicas que también pueden ser útiles para la clasificación de imágenes, como la clasificación supervisada, la clasificación no supervisada y el análisis de objetos basado en imágenes.

1.3.1. Clasificación Supervisada

La clasificación supervisada es una técnica de procesamiento de imágenes que utiliza muestras de entrenamiento previamente clasificadas para enseñar a un algoritmo de clasificación a distinguir entre diferentes clases de objetos o características en una imagen.

Los tres pasos básicos para la clasificación supervisada son:

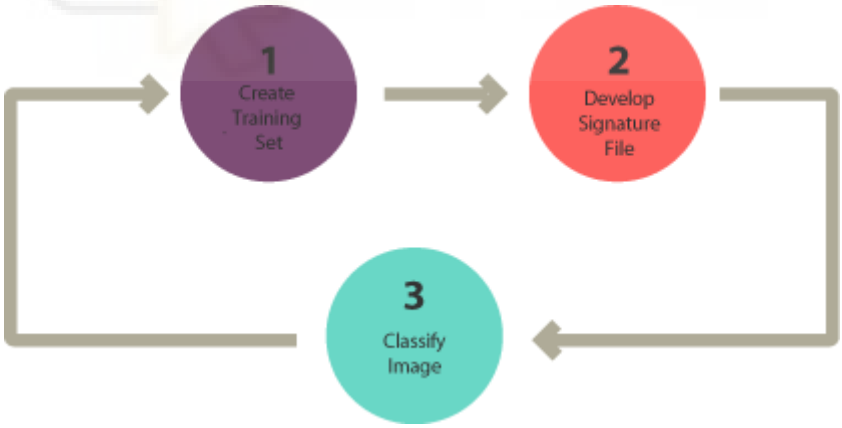


Ilustración 22: Pasos de la clasificación supervisada

- **Paso 1: Selección de áreas de entrenamiento**

  Primero se seleccionan áreas representativas en el conjunto de datos de imágenes que corresponden a las diferentes categorías que se desean clasificar. Por ejemplo, si se desea clasificar imágenes satelitales de zonas urbanas y rurales, se pueden seleccionar áreas de la imagen que contengan edificios y carreteras para las zonas urbanas, y áreas sin edificios o infraestructuras similares para las zonas rurales.

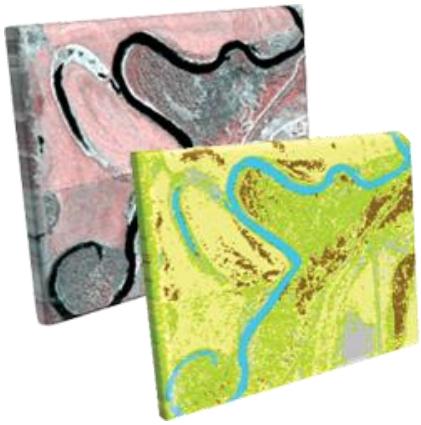


Ilustración 23: Ejemplo de selección de áreas de entrenamiento

- **Paso 2: Generación de archivo de firmas**

  A partir de las áreas de entrenamiento seleccionadas, se obtiene un archivo de firma que contiene información espectral para cada categoría [20].

  Esta información es utilizada para distinguir entre diferentes tipos de objetos o superficies en función de sus patrones únicos de reflexión de la energía electromagnética.

- **Paso 3: Clasificación**

  Finalmente, el último paso sería usar el archivo de firma para ejecutar una clasificación. A partir de aquí, se tendría que elegir un algoritmo de clasificación como [20]:

  - Máxima verosimilitud (Maximum likelihood)
  - Distancia mínima (Minimum-distance)
  - Componentes principales
  - Support Vector Machine (SVM)
  - Iso cluster

1.3.2. Clasificación No Supervisada

La clasificación no supervisada es una técnica de procesamiento de imágenes que se utiliza para agrupar píxeles en clases o clústeres sin la necesidad de tener información previa de cada una de las clases. En otras palabras, se agrupan los píxeles en función de sus propiedades espectrales y se asigna una clase a cada grupo.

Como no se necesitan muestras para la clasificación no supervisada, es una manera fácil de segmentar y entender una imagen [20].

Los dos pasos básicos para la clasificación no supervisada son [20]:

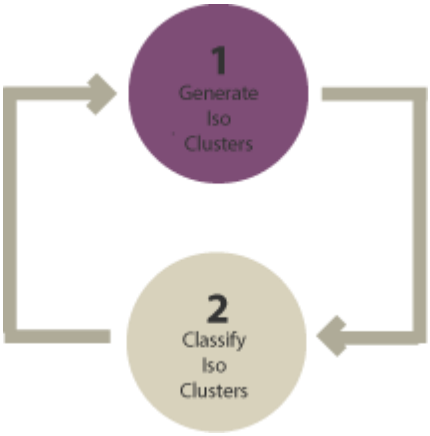


Ilustración 24: Pasos de la clasificación no supervisada

- **Paso 1: Generar clusters**

  Usando un algoritmo de clustering, se agrupan los píxeles de la imagen en grupos o clusters en base a sus propiedades espectrales.

  Se pueden crear 8, 20 o 42 grupos. Menos clusters tienen más píxeles similares dentro de los grupos. Pero más clusters aumentan la variabilidad dentro de los grupos [20].

  Algunos de los algoritmos de clustering más comunes son K-means e ISODATA [20].

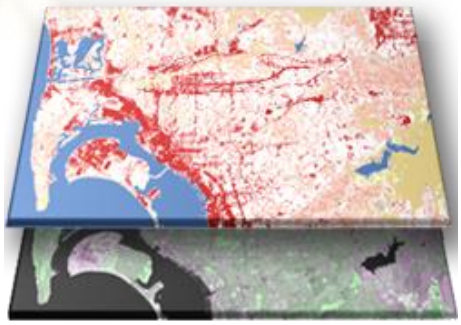


Ilustración 25: Ejemplo de generación de clusters

- **Paso 2: Asignar clases**

  El siguiente paso es asignar manualmente clases de cobertura del suelo a cada grupo. Por ejemplo, si desea clasificar vegetación y no vegetación, puede seleccionar aquellos grupos que los representen mejor [20].

- **Paso 3: Clasificación Basada en Objetos (OBIA)**

  La clasificación basada en objetos (OBIA) es una técnica de clasificación de imágenes que implica la segmentación de la imagen en objetos con diferentes formas y tamaños, en lugar de trabajar con píxeles individuales.

A continuación, se describen los pasos más comunes en el proceso de la clasificación basada en análisis de objetos de imágenes:

1. **Segmentación multirresolución**

   El primer paso es segmentar la imagen agrupando píxeles contiguos y similares para formar objetos. Si los objetos se segmentan con precisión y las características de los objetos son lo suficientemente distintas, los objetos resultantes pueden ser tan significativos que la clasificación se vuelve automática.

   Por ejemplo, los resultados de segmentación a continuación resaltan edificios [20].

Ilustración 26: Ejemplo de segmentación multirresolución

   Los 2 algoritmos de segmentación más comunes son [20]:

   - Segmentación multiresolución en eCognition
   - La herramienta de desplazamiento medio del segmento en ArcGIS

2. **Seleccionar áreas de entrenamiento**

3. **Definir estadísticas**

   Las estadísticas se definen para cada objeto segmentado y se utilizan para la descripción y clasificación de estos.

4. **Clasificación**

   La clasificación se realiza mediante diferentes métodos, como la forma, textura, espectral, contexto geográfico o clasificación de vecino más cercano (NN), que utiliza las estadísticas definidas para cada objeto con el fin de compararlos con las áreas de entrenamiento y asignar una clase a cada objeto.

CAPÍTULO 2: ESTADO DE LA CUESTIÓN

Biblioteca
UNIVERSITAS Miguel Hernández

2.1. Introducción Redes Convolucionales

El cerebro humano procesa imágenes sin darnos cuenta, pero las máquinas necesitan saber cómo representar una imagen para poder leerla.
Cada imagen es un arreglo de puntos (un píxel) dispuestos en un orden especial. Si cambia el orden o el color de un píxel, la imagen también cambiaría [21].

La máquina divide la imagen en una matriz de píxeles y almacena el color de cada píxel en una ubicación.

Se debe preservar la disposición espacial en ambas direcciones horizontal y vertical, y se puede usar una matriz de peso para tomar los píxeles juntos.

Esto es exactamente lo que hace una red neuronal convolucional. Una red neuronal convolucional toma una imagen de entrada y la procesa con matrices de peso para extraer características específicas sin perder la información de su disposición espacial.

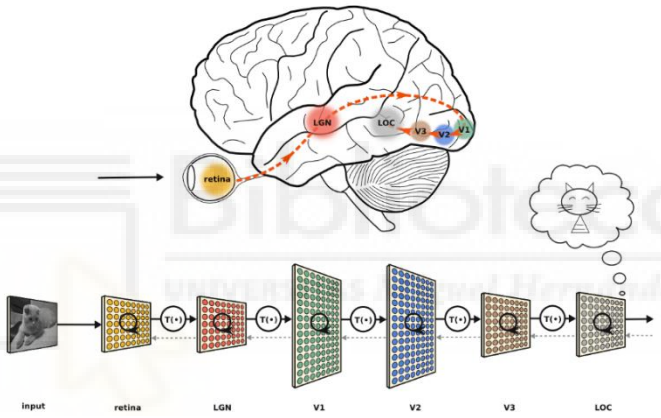


Ilustración 27: Procesamiento de imagen en el cerebro y red convolucional

1.4. Evolución de las CNN

1.4.1. Células simples y complejas

En 1959, David Hubel y Torsten Wiesel descubrieron células simples y complejas. Según su estudio, para el reconocimiento de patrones visuales, se utilizan dos tipos de células [22]. Una célula simple responde a bordes y barras de orientaciones particulares, como esta imagen [23]:

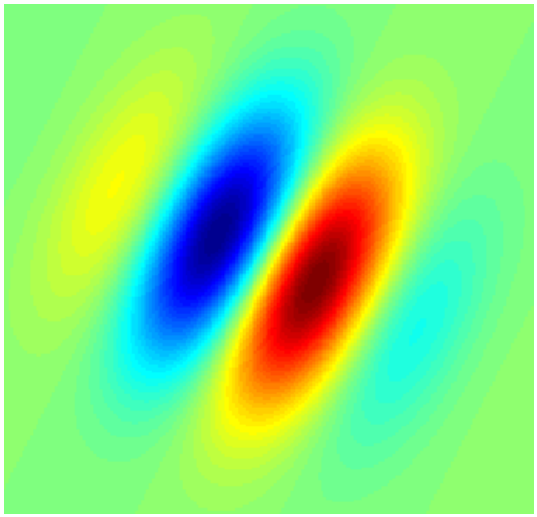


Ilustración 28: Célula simple

Una célula compleja también responde a bordes y barras de orientaciones particulares, pero es diferente de una célula simple en que estos bordes y barras pueden desplazarse por la escena y la célula seguirá respondiendo. Por ejemplo, una célula simple puede responder solo a una barra horizontal en la parte inferior de una imagen, mientras que una célula compleja podría responder a barras horizontales en la parte inferior, media o superior de una imagen. Esta propiedad de las células complejas se llama "invarianza espacial" [23].

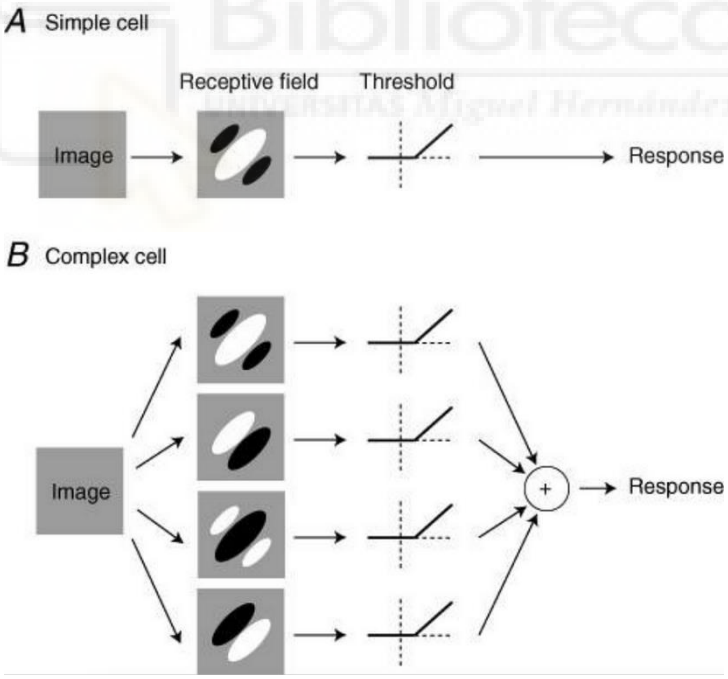


Ilustración 29: Diferencia entre célula simple y compleja

1.4.2. El Neocognitron de Kunihiko Fukushima

En la década de los 80, el Dr. Kunihiko Fukushima se inspiró en el trabajo de Hubel y Wiesel sobre células simples y complejas, y propuso el modelo "neocognitron". El modelo neocognitron incluye componentes denominados "células S" y "células C" [23].

Las células S se encuentran en la primera capa del modelo, y están conectadas a las células C que se encuentran en la segunda capa del modelo. La idea general es capturar el concepto "de simple a complejo" y convertirlo en un modelo computacional para el reconocimiento de patrones visuales [23].

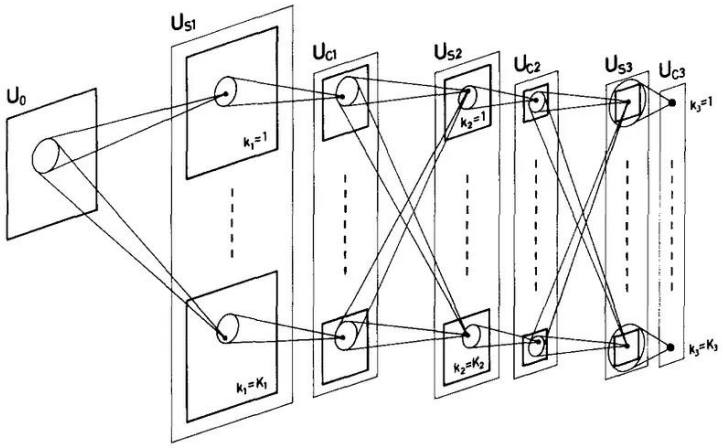


Ilustración 30: El modelo Neocognitron

1.4.3. El LeNet-5 de Yann LeCun

Aunque el trabajo de Fukushima fue muy influyente en el campo emergente de la IA, la primera aplicación moderna de las redes neuronales convolucionales fue implementada en la década de los 90 por Yann LeCun y otros en su artículo "Aprendizaje basado en gradientes aplicado al reconocimiento de documentos", que probablemente sea el artículo de IA más popular de la década de los 90 (citado en 34378 artículos) [23].

En el artículo, Yann LeCun entrenó una red neuronal convolucional con el conjunto de datos MNIST de dígitos escritos a mano [23].

La idea detrás de la LeNet-5 de Yann LeCun era mejorar el modelo Neocognitron de Kunihiko Fukushima. En lugar de utilizar células artificiales simples y complejas para identificar patrones visuales, LeCun propuso el uso de redes neuronales convolucionales para identificar características más complejas a partir de características más simples.

LeCun entrenó la LeNet-5 con el conjunto de datos MNIST de dígitos escritos a mano siguiendo los siguientes pasos:

- Proporcionar al modelo una imagen de ejemplo;
- Pedir al modelo que prediga la etiqueta;
- Actualizar los ajustes del modelo comparando el resultado de la predicción y el valor de la etiqueta real;
- Repetir este proceso hasta alcanzar los ajustes opcionales del modelo donde se minimice la pérdida [23].

La implementación de LeCun estableció los estándares para las aplicaciones de visión por computadora y procesamiento de imágenes actuales [23].

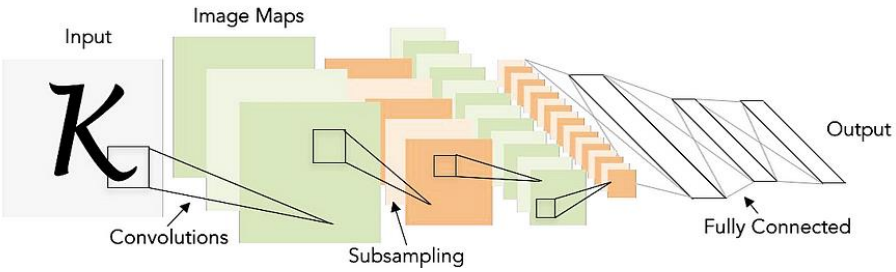


Ilustración 31: Arquitectura del LeNet-5

1.4.4. Desde la década de los 90 en adelante

Después del modelo LeNet-5, se desarrollaron y mejoraron muchos otros modelos de redes neuronales convolucionales. Uno de los primeros y más influyentes fue el modelo AlexNet, que ganó la competencia ImageNet Large Scale Visual Recognition Challenge (ILSVRC) en 2012. Este modelo utilizó una arquitectura más profunda y una mayor cantidad de capas de convolución y neuronas que el modelo LeNet-5.

Similar a MNIST, ImageNet es un conjunto de datos públicos y gratuitos de imágenes y sus correspondientes etiquetas verdaderas. En lugar de centrarse en dígitos escritos a mano etiquetados de 0 a 9, ImageNet se centra en imágenes naturales, o imágenes del mundo, etiquetadas con una variedad de descriptores que incluyen "anfibio", "mueble" y "persona" [23].

A lo largo de los últimos años, se han propuesto y perfeccionado muchos otros modelos, como VGGNet, GoogLeNet, ResNet y DenseNet, que han mejorado continuamente la precisión y eficiencia de las redes neuronales convolucionales en una variedad de aplicaciones de visión artificial, como la detección y clasificación de objetos, reconocimiento de rostros, y segmentación de imágenes, entre otros.

En la siguiente imagen se muestra una línea temporada con la evolución de las redes convolucionales:

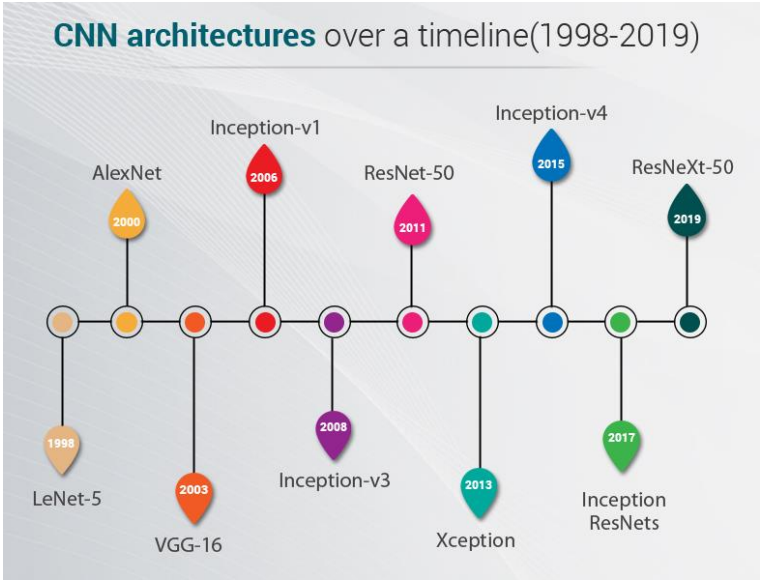


Ilustración 32: Evolución de las CNN

1.5. Arquitectura de una CNN

En general, las redes neuronales convolucionales se construyen con una estructura que contendrá 3 tipos distintos de capas [2]:

- Una capa convolucional, que es la que le da el nombre a la red.
- Una capa de reducción o de pooling, la cual va a reducir la cantidad de parámetros al quedarse con las características más comunes.
- Una capa clasificadora totalmente conectada, la cual nos va a dar el resultado final de la red.

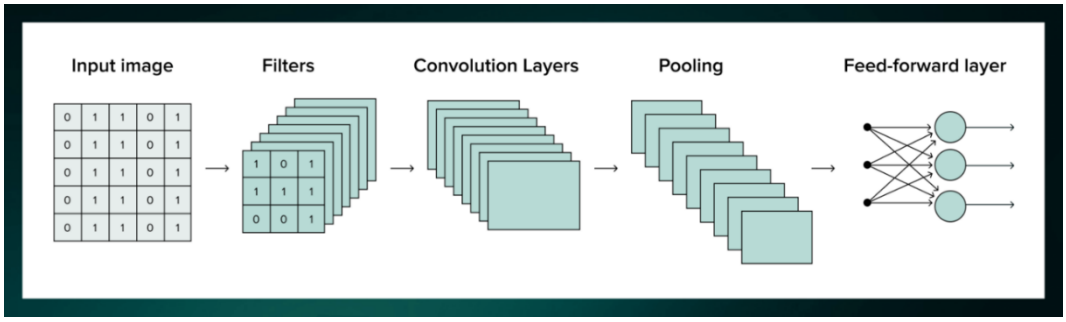


Ilustración 33: Capas de una red convolucional

1.5.1. Capa Convolucional

La capa convolucional es el componente más importante de cualquier arquitectura CNN. Esta capa es la encargada de aplicar la operación de convolución a la imagen de entrada para encontrar los patrones que más tarde permitirán clasificarla.
Para ello, se define:

- **El número de filtros/kernels a aplicar a la imagen**: el número de matrices por las que se van a convolucionar las imágenes de entrada [24].

  Un kernel o filtro se puede describir como una cuadrícula de valores o números discretos, donde cada valor se conoce como el peso de ese kernel [25].

  Durante el inicio del proceso de entrenamiento de un modelo CNN, todos los pesos de un kernel se asignan con números aleatorios (también hay diferentes enfoques disponibles para inicializar los pesos) [25].

  Luego, con cada época de entrenamiento, los pesos se ajustan y el kernel aprende a extraer características significativas. En la siguiente imagen se muestra un filtro 2D [25].

0	1
-1	2

Ilustración 34: Ejemplo de un kernel 2x2

- **El tamaño de estos filtros**: 99% de las veces son cuadrados, de 3×3, 5×5, etcétera [24].

A continuación, se puede ver el esquema general en el que se realiza la convolución de la matriz que contiene los datos de la imagen con un kernel o filtro, lo que proporciona un mapa de activación 2D para cada filtro.

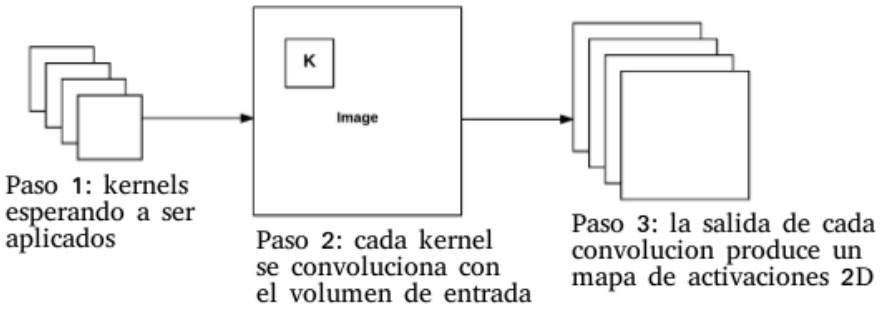


Ilustración 35: Esquema general de convolución de una imagen

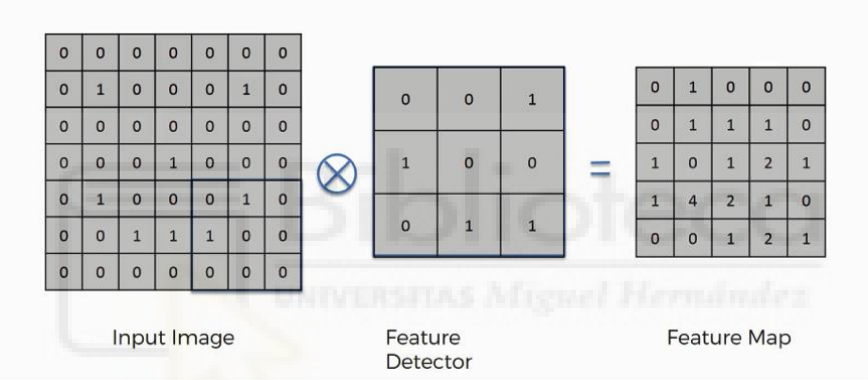


Ilustración 36: Operación de convolución

Durante la convolución, se multiplican los valores de cada elemento de la matriz de entrada con los valores correspondientes del kernel. Si el resultado de esta multiplicación es igual o mayor que 1, se mantiene ese valor, de lo contrario se eliminan.
La convolución sigue la siguiente fórmula [26]:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau$$

Lo que pretende es mostrar cómo una función modifica la forma de la otra. En la siguiente imagen, se generaron datos que fueron modificados usando el filtro para crear un mapa de activación. De esta manera, a través de la utilización de diferentes filtros, se generan muchos mapas de activación que, en conjunto, forman la capa de convolución [26].

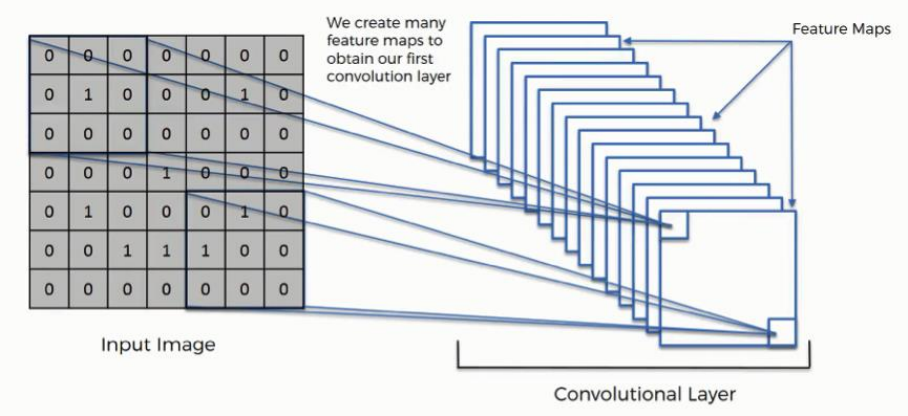


Ilustración 37: Capa convolucional

1.5.2. Capa de Pooling

Una limitación de la salida del mapa de características (o mapa de activación) de las capas convolucionales es que registran la posición precisa de las características en la entrada. Esto significa que pequeños movimientos en la posición de la característica en la imagen de entrada darán como resultado un mapa de características diferente. Esto puede ocurrir con el recorte, la rotación, el desplazamiento y otros cambios menores en la imagen de entrada [27].

Un enfoque común para abordar este problema es usar una capa de pooling.

La incorporación de la capa de agrupamiento (pooling layers) en las redes neuronales convolucionales tiene como objetivo lograr la invariancia a cambios en la posición o condiciones de iluminación, la robustez al desorden y la compacidad de la representación [28].

En general, la capa de agrupamiento combina las características vecinas que se encuentran en la misma región de la imagen o conjunto de datos para formar una característica más general y representativa. Esta característica puede ser más fácilmente utilizada por la red para reconocer patrones y características en los datos de entrada.

Por lo tanto, la capa de agrupamiento reduce la dimensionalidad del conjunto de características y ayuda a la red a ser más eficiente en la extracción de características importantes en los datos de entrada.

Los dos tipos de pooling de la arquitectura de red más comunes son [24]:

- **max-pooling:** calcula el máximo de los elementos [24].

  Este método toma el valor más alto de cada submatriz del mapa de activación y lo forma en una matriz separada. Al hacer esto, se asegura de que las características a aprender se mantengan limitadas en número, al tiempo que se preservan las características clave de cualquier imagen. El agrupamiento máximo se realiza generalmente utilizando un filtro de 2x2 [26].

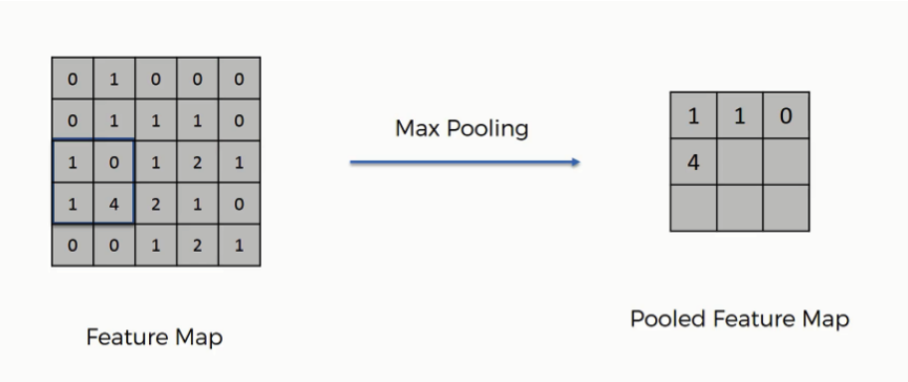


Ilustración 38: Paso de un ejemplo de agrupamiento máximo

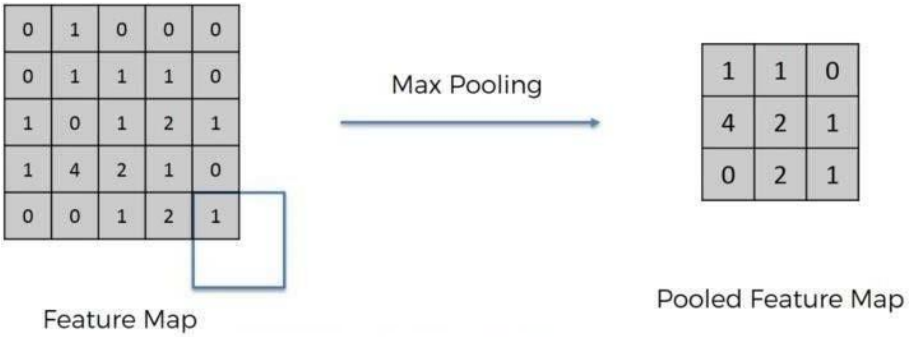


Ilustración 39: Resultado de un ejemplo de agrupamiento máximo

- **averag-pooling:** calcula la media de los elementos [24].

1.5.3. Capa Totalmente Conectada

Por lo general, la última parte de cada arquitectura de red neuronal convolucional utilizada para clasificación consiste en capas completamente conectadas, donde cada neurona de una capa está conectada con cada neurona de su capa anterior. La última capa de capas completamente conectadas se utiliza como capa de salida (clasificador) de la arquitectura de la CNN [25].

Las capas completamente conectadas son un tipo de red neuronal artificial de alimentación directa (ANN) y siguen el principio de la red neuronal de perceptrón multicapa tradicional (MLP). Estas capas reciben la entrada de la última capa de convolución o agrupamiento, que está en forma de un conjunto de métricas o mapas de características. Los mapas de características mencionados se aplanan para crear un vector que se introduce en las capas totalmente conectadas para producir la salida final de la CNN, como se muestra en la siguiente figura [25].

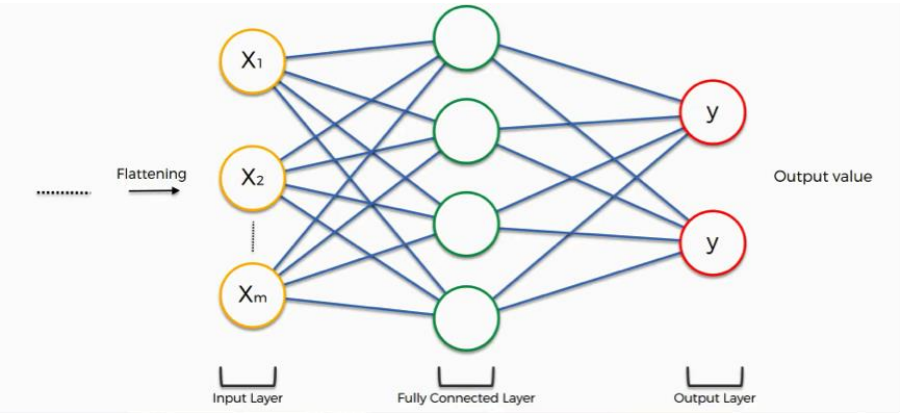


Ilustración 40: Arquitectura de las capas completamente conectadas

1.6. Aplicaciones de CNN

Las redes neuronales convolucionales son una herramienta poderosa en la IA y tienen una amplia variedad de aplicaciones prácticas en diversas industrias y campos.

En la era del Big Data, a diferencia de los enfoques tradicionales, una CNN es capaz de aprovechar una gran cantidad de datos para lograr resultados prometedores [29].

Dos de las principales aplicaciones de las CNN son el procesamiento del lenguaje natural y la clasificación de imágenes. Esta última es en la que se va a enfocar este trabajo.

1.6.1. Procesamiento del Lenguaje Natural (Natural Language Processing o NLP)

Las Redes Neuronales Convolucionales se aplican tradicionalmente en el campo de la Visión por Computadora. Sin embargo, en los últimos años también se han aplicado ampliamente en el campo del procesamiento del lenguaje natural, donde han demostrado ser útiles en numerosas tareas, como la categorización de texto, el análisis semántico, la recuperación de consultas de búsqueda, la modelización de oraciones, la clasificación y predicción de texto, entre otras [30].

Los modelos más utilizados y efectivos para procesar secuencias de datos (como el lenguaje natural) están construidos en base a redes convolucionales complejas o recurrentes [19].

Las redes recurrentes usan la posición de los símbolos en las secuencias de entrada y salida para realizar cálculos. Al alinear estas posiciones en pasos de tiempo, se genera una secuencia de estados ocultos ht, que dependen del estado oculto anterior ht-1 y la entrada en la posición t [19].

Sin embargo, esta naturaleza inherentemente secuencial impide la paralelización dentro de los ejemplos de entrenamiento, lo cual se vuelve crítico en longitudes de secuencia más largas, ya que las limitaciones de memoria limitan el agrupamiento de ejemplos [19].

Por este motivo, los mecanismos de atención son fundamentales para modelos de transducción y modelado de secuencias, ya que permiten la modelización de dependencias sin importar su distancia. Sin embargo, su uso generalmente es en conjunto con redes recurrentes [19].

En contraste, el Transformer propone una arquitectura de modelo que rechaza la recurrencia y en su lugar depende completamente de un mecanismo de atención para establecer dependencias globales entre la entrada y la salida. El Transformer permite una paralelización significativamente mayor y puede alcanzar un nuevo estado de vanguardia en calidad de traducción después de ser entrenado [19].

Hoy en día, las redes Transformer se han convertido en una de las arquitecturas de modelo más populares en el procesamiento de lenguaje natural y la visión por computadora. Esto se debe a su capacidad de modelar dependencias globales de forma eficiente y paralela. Los modelos populares de lenguaje natural basados en Transformers incluyen ChatGPT, BERT, GPT-4, T5 y RoBERTa, entre otros.

A continuación, se explicará brevemente el reconocimiento de voz y la categorización de texto.

1.6.1.1. Reconocimiento de Voz

El reconocimiento de voz, o conversión de voz a texto, es la capacidad de una máquina o programa para identificar palabras habladas en voz alta y convertirlas en texto legible [31].

En condiciones normales, el reconocimiento de voz se basa en el análisis del espectro de la señal de voz en el tiempo y la frecuencia, lo que implica una gran diversidad en las señales de voz debido a los diferentes hablantes y entornos. Sin embargo, al utilizar la idea de las redes neuronales convolucionales, se trata la señal de voz como una imagen y se utiliza una red convolucional profunda, que es ampliamente utilizada en la visión por computadora, para identificarla.

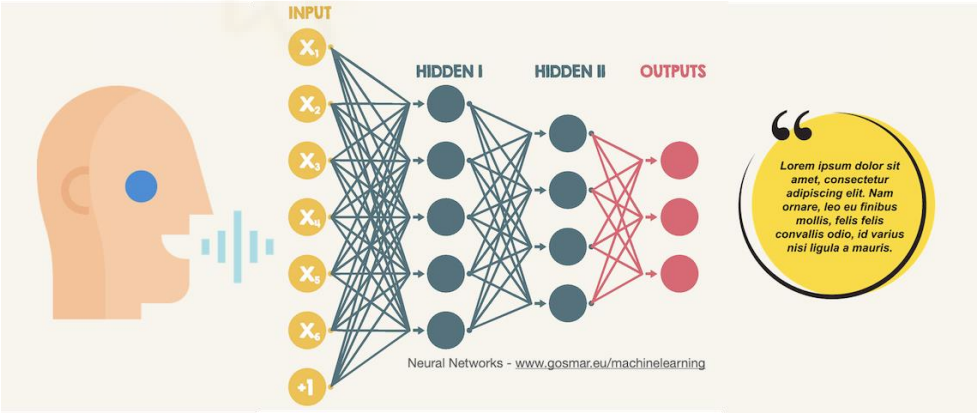


Ilustración 41: Funcionamiento del reconocimiento de voz con CNN

Los sistemas de reconocimiento de voz tienen muchas aplicaciones, como asistentes virtuales, servicio al cliente, aplicaciones médicas, transcripción en tiempo real de procedimientos judiciales, reconocimiento de emociones y comunicación manos libres.

Estas aplicaciones permiten el uso de la voz para interactuar con dispositivos y servicios, y pueden mejorar la accesibilidad y la eficiencia en diversos contextos.

1.6.1.2. Clasificación de Texto

La categorización de texto es la tarea de asignar automáticamente clases predefinidas a documentos escritos en lenguaje natural [30].

Se estudian muchos tipos de categorización de texto, cada uno de los cuales se ocupa de diferentes tipos de documentos y clases, como la categorización de temas para descubrir los temas mencionados (por ejemplo, deportes, educación, política), la detección de spam y la clasificación de sentimientos para detectar el sentimiento general en reseñas de productos o películas [30].

En la clasificación de texto, las CNN se utilizan principalmente para tareas como categorización de temas o análisis de sentimientos. Esto se debe a que las principales características de las CNN, como la invariancia a la ubicación y la composicionalidad local, no se aplican en el procesamiento de lenguaje natural. La posición de una palabra en una oración es muy importante, y las palabras cercanas no necesariamente tienen el mismo significado o están conectadas.

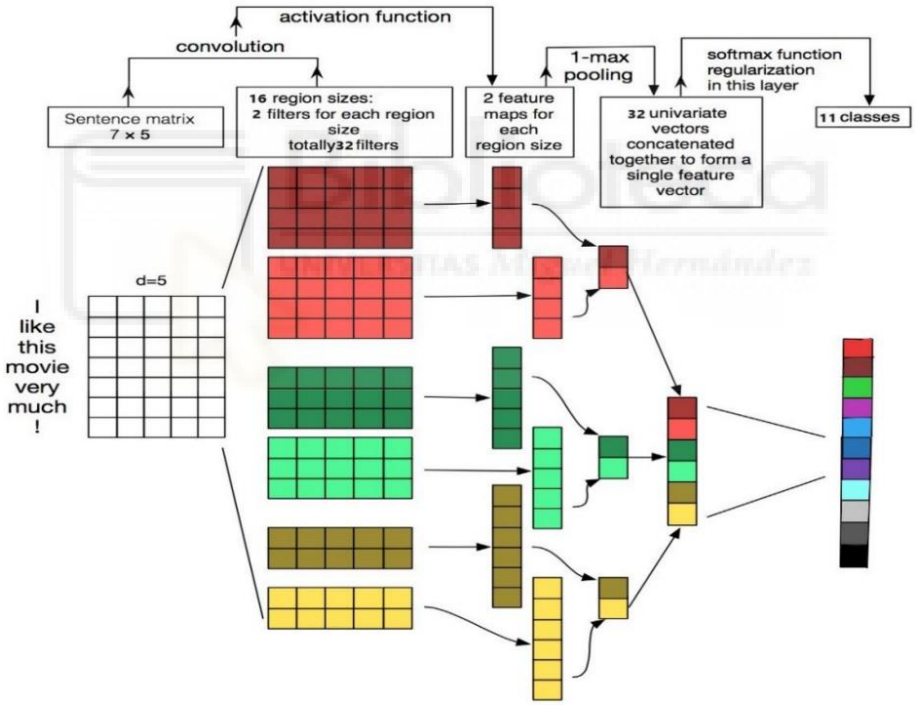


Ilustración 42:Funcionamiento de la Categorización de Texto con CNN

1.6.2. Clasificación de Imágenes

El uso principal de las redes neuronales convolucionales es el reconocimiento y la clasificación de imágenes. También es el único caso de uso que involucra los frameworks más avanzados (especialmente en el caso de la imagen médica) [32].

El proceso de categorización de imágenes a través de CNN se logra mediante el uso de algoritmos de aprendizaje supervisado y no supervisado, y se aplica en campos como la etiquetación de imágenes, la búsqueda visual y los motores de recomendación.

Grandes empresas como Facebook, Google y Amazon utilizan esta tecnología para mejorar la experiencia del usuario y hacer recomendaciones de productos basados en criterios visuales.

1.6.2.1. Reconocimiento Facial

El reconocimiento facial es una subcategoría del reconocimiento de imágenes que trata imágenes más complejas, como rostros humanos o de animales [32].

La distinción entre el reconocimiento de imágenes y el reconocimiento facial se basa en la complejidad operativa: la capa adicional de trabajo requerida [32].

La forma y las características del rostro se reconocen primero, seguido del reconocimiento básico del objeto. Luego se examinan las características del rostro para determinar su identidad, y se calcula la percepción de apariencia de una persona en particular.

El reconocimiento facial se utiliza en plataformas de redes sociales como Facebook para las redes sociales y el entretenimiento. Los ejemplos más destacados son los filtros de Snapchat y simplificar el proceso a menudo dudoso de etiquetar personas en fotos.

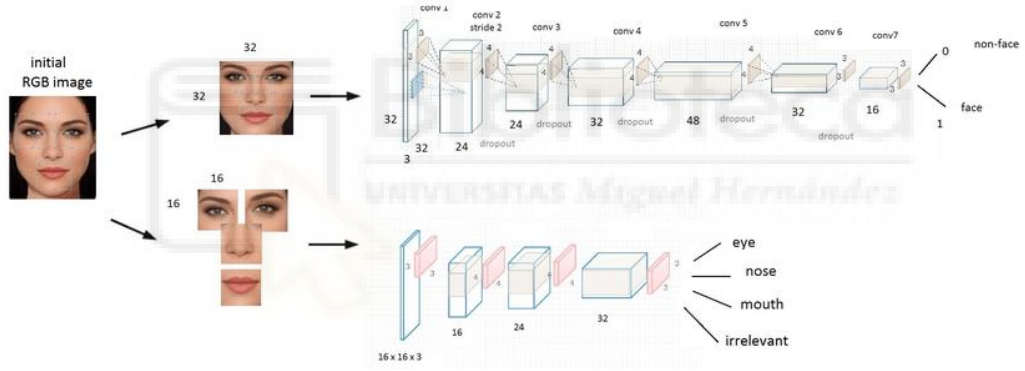


Ilustración 43: Proceso de Reconocimiento Facial

1.6.2.2. Procesamiento de Imágenes Médicas

La inteligencia artificial en imágenes médicas es un desarrollo reciente que tiene el potencial de revolucionar el campo. La capacidad de la IA para aprender y hacer predicciones puede ayudar a los médicos a diagnosticar enfermedades de manera más temprana y precisa [33].

Por ejemplo, los médicos utilizan algoritmos de aprendizaje profundo para diagnosticar enfermedades a partir de imágenes médicas como radiografías y tomografías computarizadas. Además, los científicos están trabajando en la creación de robots que puedan ayudar a los enfermeros en el cuidado de los pacientes [33].

La IA puede ayudar a reducir el número de falsos positivos y negativos en las imágenes médicas, lo que puede llevar a diagnósticos y planes de tratamiento más precisos. Aunque existen preocupaciones sobre cuántos datos necesita la IA para ser efectiva, los beneficios que esta tecnología aporta a la comunidad médica son mayores [33].

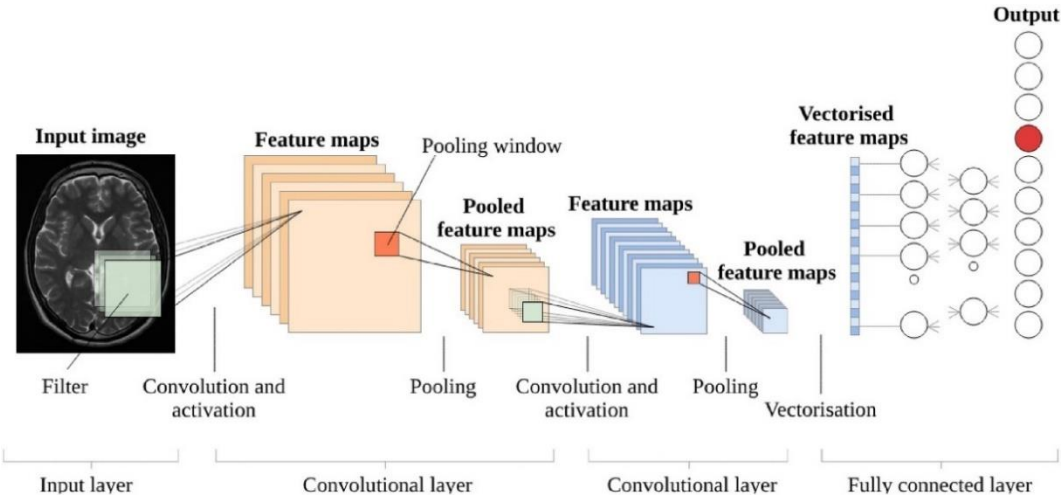


Ilustración 44: Funcionamiento del procesado de imágenes médicas

Biblioteca
UNIVERSITAS Miguel Hernández

CAPÍTULO 3: OBJETIVOS

Biblioteca
UNIVERSITAS Miguel Hernández

3.1. Organización de los objetivos

En este apartado se van a definir los objetivos a alcanzar con este proyecto.
Dichos objetivos se dividen en diferentes categorías que son las siguientes:

- **Generales**: son los objetivos que se pretenden alcanzar con el trabajo, los resultados que se esperan de este proyecto.
- **Técnicos**: se refieren a los conocimientos y habilidades técnicas que se pretenden adquirir o aplicar durante el desarrollo del trabajo.
- **Didácticos**: son aquellos que se pretenden alcanzar con el trabajo y que pueden ser útiles para el desarrollo de proyectos relacionados con el tema en cuestión.
- **Personales**: son los objetivos que el autor del proyecto se ha planteado como metas personales que desea alcanzar a través del desarrollo del proyecto. Estos pueden estar relacionados con su formación académica, su desarrollo profesional o su desarrollo personal.

3.1.1. Objetivos generales

- Implementar y entrenar una red neuronal convolucional utilizando un conjunto de datos de entrenamiento y ajustando los parámetros para optimizar el rendimiento.
- Desarrollar un sistema de ayuda al diagnóstico que integre la técnica de redes neuronales convolucionales para mejorar la precisión y eficiencia en la detección de neumonías en las radiografías.
- Diseñar e implementar una interfaz gráfica de usuario para interactuar con la red neuronal convolucional y mostrar los resultados en tiempo real.

3.1.2. Objetivos Técnicos

- Diseñar y desarrollar una arquitectura de red neuronal convolucional que sea capaz de identificar las características relevantes en las radiografías de tórax para detectar la presencia de neumonía.

3.1.3. Objetivos Didácticos

- Preparar un entorno de trabajo en Google Colab para el desarrollo de modelos de redes neuronales convolucionales.
- Instalar y configurar las bibliotecas de Python necesarias para trabajar con redes convolucionales, como TensorFlow [34], Keras [35] o PyTorch [36], en Google Colab.
- Instalar y utilizar las bibliotecas de Python necesarias para el análisis de datos y la visualización de resultados en la tarea de clasificación de imágenes con redes neuronales convolucionales, tales como Matplotlib, NumPy o Pandas en Google Colab.
- Descargar y cargar conjuntos de datos relevantes para la tarea de clasificación de imágenes, como CIFAR-10, ImageNet o Kaggle en Google Colab.

3.1.4. Objetivos Personales

- Aprender los conceptos fundamentales de la programación en Python, incluyendo variables, funciones, estructuras de control y tipos de datos, necesarios para trabajar con redes neuronales convolucionales.
- Adquirir habilidades para trabajar con herramientas de programación en Python, como TensorFlow, PyTorch, o Keras, para desarrollar y entrenar redes neuronales convolucionales
- Desarrollar habilidades técnicas en la creación de aplicaciones web utilizando la plataforma Anvil y aplicar estos conocimientos en la implementación de una interfaz de usuario intuitiva para la red convolucional.
- Adquirir experiencia en la implementación y automatización de tareas de preprocesamiento de datos en Google Colab, para mejorar la eficiencia en el procesamiento de grandes conjuntos de datos utilizados en el entrenamiento de redes neuronales convolucionales.

3.1.5. Lo que no es objetivo

Los siguientes puntos no son objetivos de este trabajo:

- Cubrir todas las posibles aplicaciones de las redes neuronales convolucionales en lugar de enfocarse en una o varias aplicaciones específicas y profundizar en ellas para poder analizarlas en detalle.
- Asegurar que el modelo de red neuronal convolucional tenga una precisión del 100% en todos los casos posibles.
- Realizar pruebas exhaustivas y experimentos con todas las posibles arquitecturas de redes neuronales, así como con diferentes bibliotecas.
- Utilizar el modelo de red neuronal convolucional desarrollado como una herramienta única y definitiva para el diagnóstico de neumonías, sin tener en cuenta otras pruebas diagnósticas y la opinión de especialistas.

CAPÍTULO 4: HIPÓTESIS DE TRABAJO

Biblioteca
UNIVERSITAS Miguel Hernández

En este capítulo, se presentarán las diversas herramientas y tecnologías utilizadas en la parte práctica de este trabajo.

4.1. Red neuronal

4.1.1. Lenguaje

Python es un lenguaje de programación interpretado y de alto nivel, diseñado para ser fácil de leer y escribir. Fue creado en 1991 por Guido van Rossum y es ampliamente utilizado en diferentes campos, incluyendo la ciencia de datos, IA, desarrollo web, automatización de tareas y más.

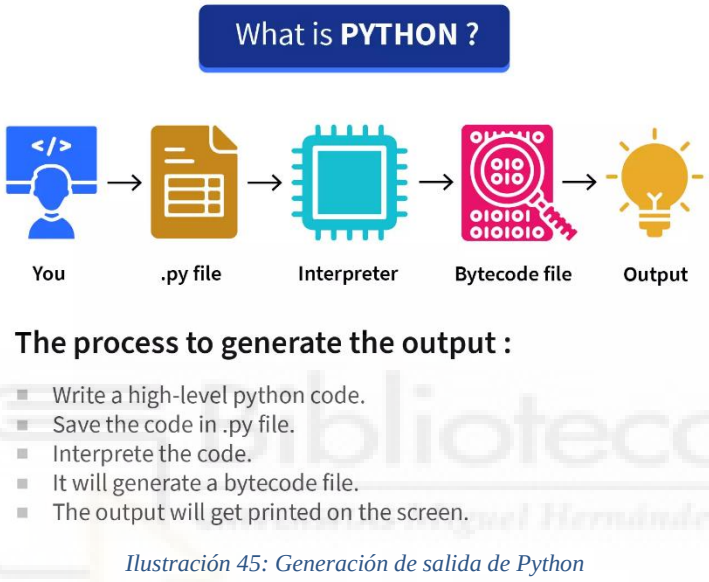


Ilustración 45: Generación de salida de Python

Este es uno de los lenguajes de programación más populares y utilizados en la actualidad, tanto en la industria como en la academia. Algunas de las razones de su relevancia son [37]:

- Sintaxis clara y legible
- Multiplataforma
- Orientado a objetos
- Código abierto
- Fácil de aprender
- Integración y adaptación
- Soporte para GUI
- Programación de alto nivel
- Amplia biblioteca estándar

Se trata, por tanto, de un lenguaje de programación muy versátil que, en los últimos años se ha abierto camino, sobre todo, en Data Science, tanto para el análisis de datos como para el desarrollo de algoritmos de aprendizaje automático [37].

Para el desarrollo de la red neuronal se han utilizado diversas librerías de Python que facilitan su implementación, entre ellas se encuentran las que se mencionan a continuación.

4.1.1.1. NumPy

NumPy es el paquete fundamental para la computación científica en Python. Es una biblioteca de Python que proporciona un objeto de array multidimensional, varios objetos derivados (como arrays y matrices enmascaradas) y una variedad de rutinas para operaciones rápidas en arrays [38].

El objeto fundamental en NumPy es el ndarray, que almacena elementos de datos de un solo tipo y permite operaciones eficientes en términos de rendimiento gracias a que gran parte del código está compilado en lenguajes de bajo nivel como C o Fortran, lo que las hace mucho más rápidas que las operaciones en listas regulares de Python.

Ilustración 46: Características de NumPy

4.1.1.2. Matplotlib

Matplotlib es un paquete de Python para la visualización de datos. Permite la creación fácil de varios tipos de gráficos, incluyendo gráficos de línea, dispersión, barras, cajas y radiales, con gran flexibilidad para estilos refinados y anotaciones personalizadas. El versátil módulo de artistas permite a los desarrolladores definir prácticamente cualquier tipo de visualización [39].

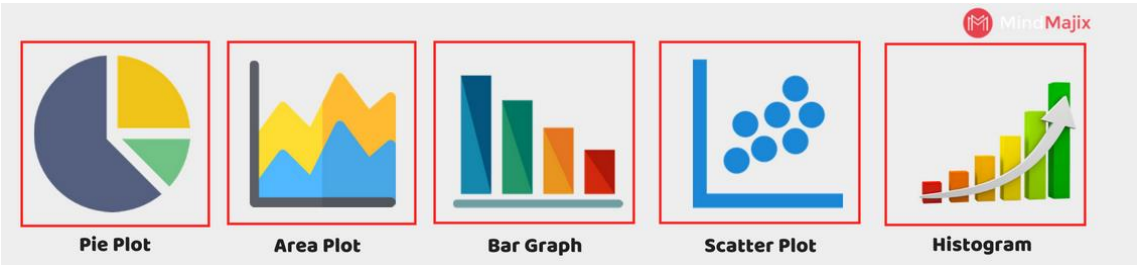


Ilustración 47: Gráficas disponibles en Matplotlib

4.1.1.3. Pandas

Pandas es un paquete de Python que proporciona estructuras de datos rápidas, flexibles y expresivas diseñadas para hacer que trabajar con datos "relacionales" o "etiquetados" sea fácil e intuitivo [40]. Introduce dos nuevos tipos de datos en Python: Series y DataFrame. El DataFrame representa la hoja de cálculo completa o los datos rectangulares, mientras que la Serie es una única columna del DataFrame. Es una herramienta útil para trabajar con datos en diferentes formatos, como tablas de Excel, archivos CSV y bases de datos SQL.

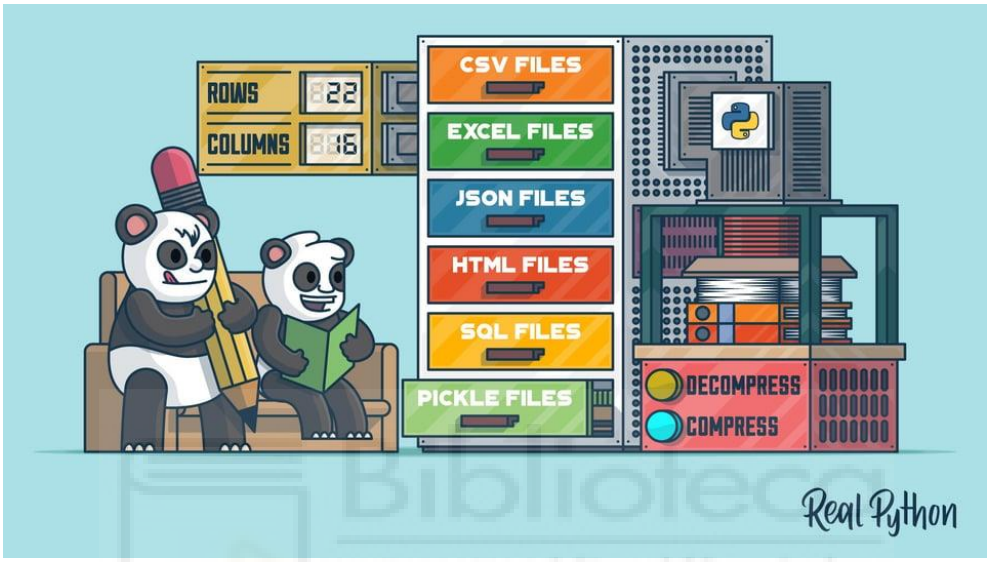


Ilustración 48: Funcionamiento de Pandas

4.1.1.4. Scikit-learn

Scikit-learn es una biblioteca de aprendizaje automático de código abierto para el lenguaje de programación Python. Este paquete proporciona una amplia selección de herramientas de aprendizaje automático supervisado y no supervisado para problemas de clasificación, regresión, reducción de la dimensionalidad y agrupamiento. Además, se enfoca en la facilidad de uso, el rendimiento y la accesibilidad para usuarios con diferentes perfiles.

Ilustración 49: Paquete Scikit-learn

4.1.1.5. PyTorch

PyTorch es una biblioteca para programas en Python que facilita la construcción de proyectos de aprendizaje profundo. Se enfatiza en la flexibilidad y permite que los modelos de aprendizaje profundo se expresen en un lenguaje Python idiomático [41].

Una de las características clave de PyTorch es su estructura de datos central, el tensor, que es un arreglo multidimensional similar a los arreglos NumPy y permite la realización de operaciones matemáticas aceleradas en hardware dedicado, lo que facilita el diseño y entrenamiento de arquitecturas de redes neuronales.

Otra característica distintiva de PyTorch es su capacidad para trabajar con grafos de cómputo dinámicos, lo que permite una mayor flexibilidad en la construcción de modelos y reduce la cantidad de código necesario para implementar y modificar modelos de aprendizaje profundo.

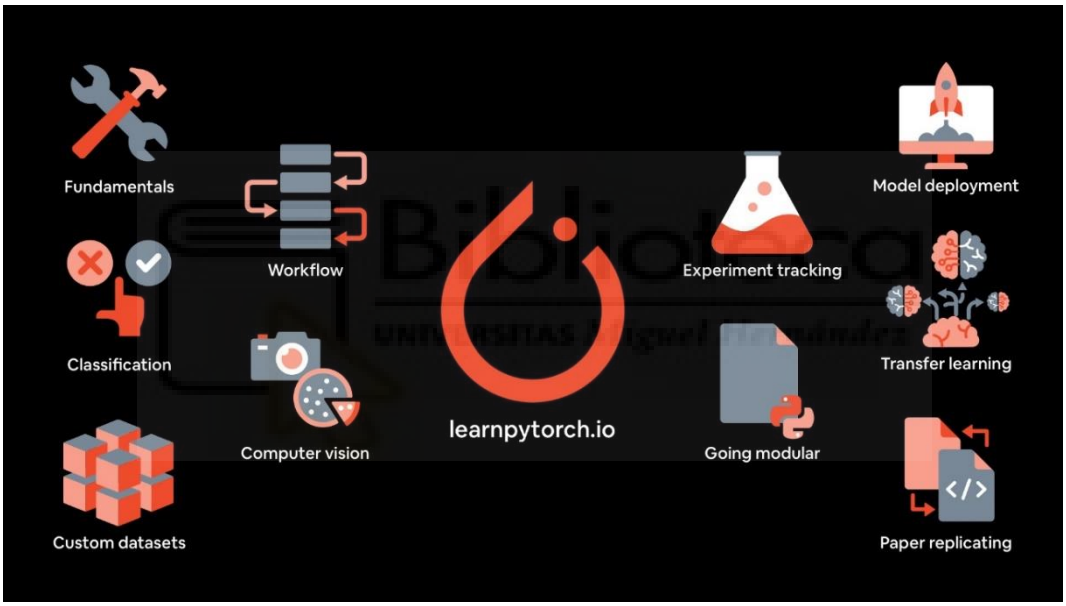


Ilustración 50: Funcionalidades de PyTorch

Además del paquete fundamental Torch, PyTorch tiene varios paquetes adicionales que extienden su funcionalidad. En particular, dos de los paquetes adicionales más relevantes de PyTorch que han sido utilizados en este proyecto son:

- **Torchvision** consiste en conjuntos de datos populares, arquitecturas de modelos y transformaciones de imágenes comunes para la visión por computadora [42].

- **Torchmetrics** es una colección de métricas de aprendizaje automático para modelos distribuidos y escalables de PyTorch, y una API fácil de usar para crear métricas personalizadas [43].

- **Torchinfo** proporciona información complementaria a la que se obtiene con "print(tu_modelo)" en PyTorch, similar a model.summary() en Tensorflow, para visualizar el modelo y facilitar la depuración de la red [44].

4.1.1.6. MLxtend

MLxtend es una biblioteca que implementa una variedad de algoritmos y utilidades esenciales para el aprendizaje automático y la minería de datos [45]. Esta biblioteca ofrece algoritmos de selección de características secuenciales, generalización apilada y minería de patrones frecuentes, así como funciones de visualización para inspeccionar el rendimiento predictivo de diferentes subconjuntos de características.

Ilustración 51: Biblioteca MLxtend

4.1.1.7. Requests

Requests es una biblioteca HTTP elegante y sencilla para Python, que permite enviar solicitudes HTTP/1.1 de manera fácil. No hay necesidad de agregar manualmente cadenas de consulta a las URLs o codificar los datos POST. La conexión keep-alive y el pooling de conexiones HTTP son automáticos gracias a urllib3 [46].

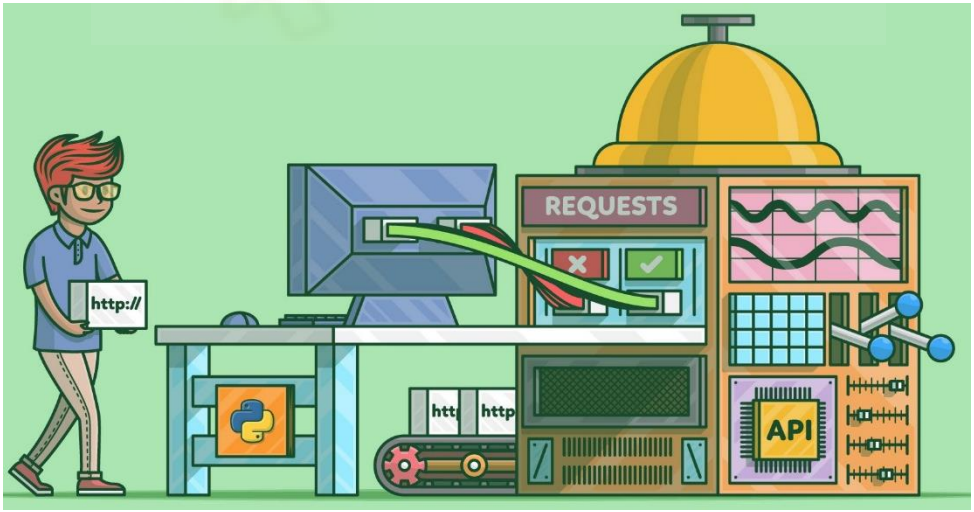


Ilustración 52: Funcionamiento de la biblioteca Requests

4.1.1.8. tqdm

tqdm es una biblioteca de Python que proporciona una barra de progreso visual durante la iteración de un bucle para indicar el progreso del proceso. También proporciona algunas características adicionales, como la capacidad de pausar y reanudar la barra de progreso y personalizar el aspecto de la barra de progreso.

4.1.1.9. timeit

La biblioteca timeit de Python es una herramienta que proporciona una forma sencilla de cronometrar pequeños fragmentos de código Python. Tiene tanto una Interfaz de línea de comandos como una invocable [47].

4.1.1.10. Pathlib

Pathlib es un módulo de la biblioteca estándar de Python que proporciona una forma orientada a objetos para manejar rutas de sistema de archivos y manipular archivos. Es decir, ofrece una manera más sencilla y legible de manejar rutas de archivos y directorios en comparación con el uso de cadenas de texto.

4.1.1.11. split-folders

split_folders es una biblioteca de Python que proporciona una forma sencilla de dividir un conjunto de datos en conjuntos separados de entrenamiento, validación y prueba. Se puede utilizar para dividir automáticamente una carpeta de imágenes en carpetas separadas para cada uno de estos conjuntos [48].

Esta biblioteca toma como entrada un directorio que contiene el conjunto de datos original y la proporción de división deseada entre los conjuntos de entrenamiento, validación y prueba. Luego, la biblioteca crea subdirectorios para cada conjunto y mueve las imágenes a estos subdirectorios según la proporción especificada [48].

4.1.2. Entorno de desarrollo

En este proyecto se ha utilizado Google Colab como entorno de desarrollo para la implementación y entrenamiento de redes convolucionales. La elección de este entorno se basa en varias ventajas que ofrece, como la posibilidad de utilizar GPUs de manera gratuita.

El aprendizaje profundo requiere una gran cantidad de cálculos, y a menudo supera la capacidad de un procesador convencional. Por lo tanto, el uso de una unidad de procesamiento gráfico (GPU) se vuelve necesario para acelerar los cálculos y reducir el tiempo de entrenamiento. En este contexto, Google Colab ofrece una solución.

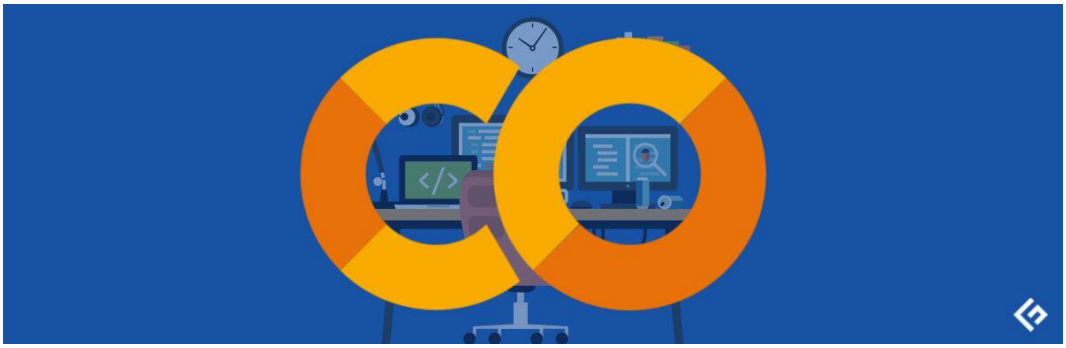


Ilustración 53: Logo de Google Colab

Google Colaboratory, o Google Colab, es un servicio en línea ofrecido por Google que permite a los usuarios escribir y ejecutar código en lenguaje Python en un entorno de desarrollo basado en la nube, con la posibilidad de utilizar recursos de hardware como CPU, GPU y TPU de manera gratuita.

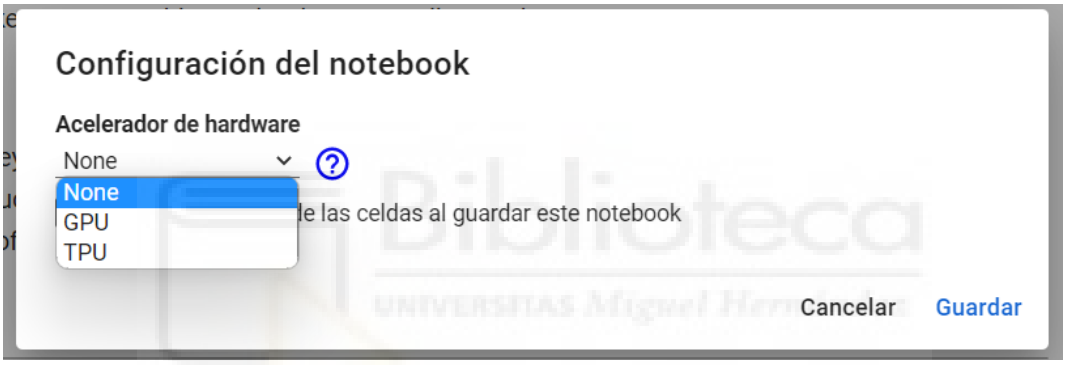


Ilustración 54: Tipos de ejecución en Google Colab

Este framework proporciona 12,72 GB de RAM y 358,27 GB de espacio en disco duro en una sola ejecución. Cada ejecución dura 12 horas, después de lo cual se restablece y el usuario debe establecer una conexión de nuevo [49].

Además, Google Colab permite descargar conjuntos de datos directamente en Google Drive desde Kaggle. Kaggle genera una clave API y proporciona un archivo. json con la clave en su interior [49]. Esto facilita el acceso a grandes conjuntos de datos para el entrenamiento de modelos.

4.2. Aplicación web

La implementación de la interfaz de la red convolucional se ha realizado mediante la utilización de la plataforma Anvil. Se ha escogido esta plataforma debido a su facilidad de uso y capacidad para implementar aplicaciones web sin la necesidad de configurar servidores o administrar infraestructuras complejas.

Además, Anvil permite la integración de la aplicación con otras herramientas y plataformas en la nube, como Google Colab. Esto permite utilizar GPUs de manera gratuita y, por lo tanto, reducir los costos de la ejecución de la red convolucional.

Ilustración 55: Logo de Anvil

Anvil es una potente plataforma en línea que permite construir aplicaciones web utilizando únicamente Python. Viene con una implementación en dos clics, autenticación de usuarios integrada, bases de datos fáciles de usar y muchas otras características [50]. También permite implementar la aplicación directamente en la nube, eliminando la necesidad de configurar un servidor para alojarla.

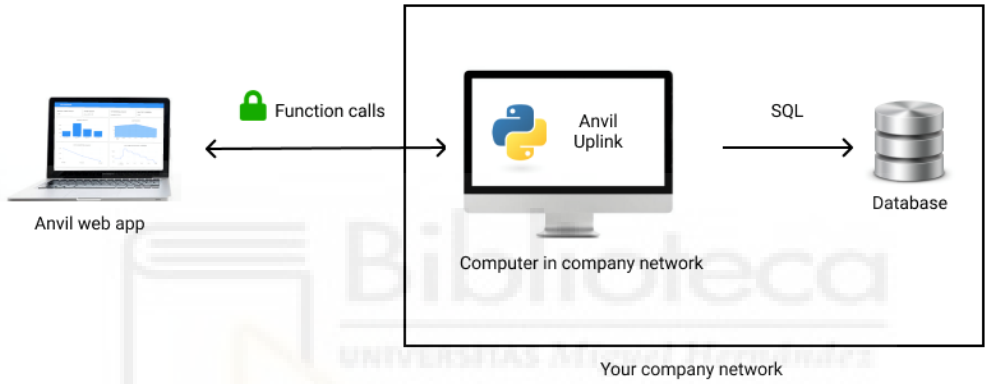


Ilustración 56: Conexión entre la aplicación web de Anvil con una SQL exterior

La plataforma en la nube Anvil se utiliza para desarrollar la parte frontal de la aplicación web y almacenarla, mientras que la plataforma en la nube de Google aloja el backend secundario del sistema. Anvil Cloud actúa como intermediario para cifrar los datos subidos por el usuario y transferirlos entre el navegador del usuario y la nube de Anvil y entre la nube de Anvil y la nube de Google. La comunicación entre Anvil y Google Colab se logra mediante el uso de la API de Anvil Uplink [51].

CAPÍTULO 5: METODOLOGÍA Y RESULTADOS

Biblioteca
UNIVERSITAS Miguel Hernández

En este capítulo se presenta la metodología utilizada y los resultados obtenidos en la implementación de la red convolucional, así como la interfaz desarrollada para su utilización.

5.1. Planificación del proyecto

La gestión de proyectos implica la aplicación de procesos, métodos, habilidades, conocimientos y experiencia para alcanzar objetivos específicos según los criterios de aceptación dentro de los parámetros acordados. Se trata de planificar, organizar, coordinar y controlar los recursos disponibles para alcanzar los objetivos del proyecto en términos de tiempo, costo, calidad y alcance [52]. La planificación del proyecto en el contexto de este trabajo es esencial para asegurar su éxito.

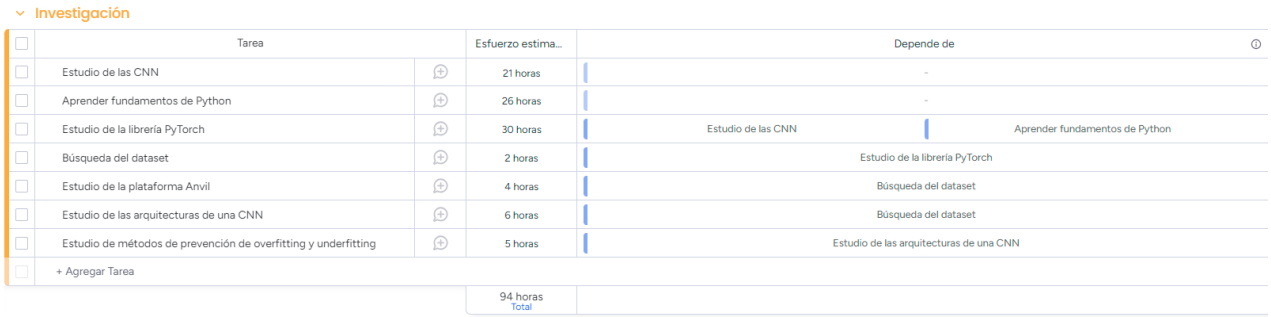
En este trabajo, se han empleado diversas técnicas de gestión de proyectos para garantizar una planificación eficiente y un seguimiento efectivo del proyecto. Dos técnicas clave que se han utilizado han sido el Desglose de Tareas (WBS, por sus siglas en inglés) y el Diagrama de Gantt.

5.1.1. Desglose de Tareas (WBS)

El desglose de tareas es una herramienta utilizada en la gestión de proyectos para descomponer el trabajo en componentes más pequeños y manejables. Consiste en una representación jerárquica y estructurada de las tareas y entregables que deben completarse en un proyecto.

Para la realización de este proyecto se ha optado por dividir las tareas en las fases de investigación, diseño, desarrollo y documentación. Cada una de estas fases representa una etapa clave en el proceso de trabajo. Dentro de cada fase, se han identificado y definido las tareas especificadas en la siguiente foto.

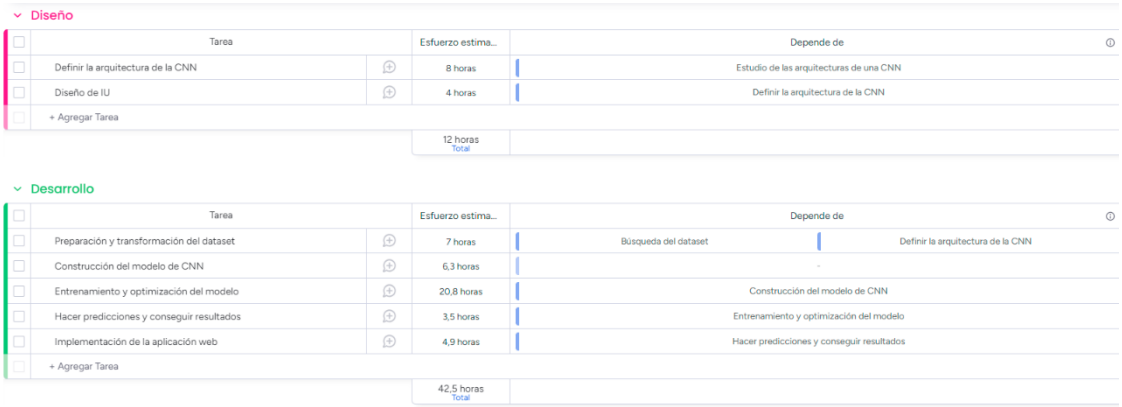
Además, se ha establecido una secuencia lógica entre las tareas mediante el uso de tareas precedentes. Estas tareas precedentes indican que la finalización de una o varias tareas es un requisito previo para comenzar la(s) siguiente(s) tarea(s).



Investigación

Tarea	Esfuerzo estima...	Depende de
Estudio de las CNN	21 horas	-
Aprender fundamentos de Python	26 horas	-
Estudio de la librería PyTorch	30 horas	Estudio de las CNN, Aprender fundamentos de Python
Búsqueda del dataset	2 horas	Estudio de la librería PyTorch
Estudio de la plataforma Anvil	4 horas	Búsqueda del dataset
Estudio de las arquitecturas de una CNN	6 horas	Búsqueda del dataset
Estudio de métodos de prevención de overfitting y underfitting	5 horas	Estudio de las arquitecturas de una CNN
+ Agregar Tarea		
	94 horas Total	

Ilustración 57: Desglose de tareas del proyecto 1



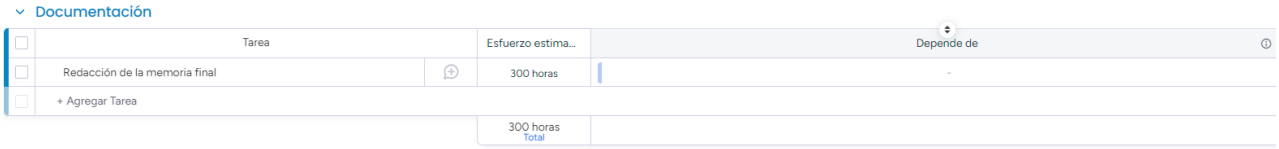
Diseño

Tarea	Esfuerzo estima...	Depende de
Definir la arquitectura de la CNN	8 horas	Estudio de las arquitecturas de una CNN
Diseño de IU	4 horas	Definir la arquitectura de la CNN
+ Agregar Tarea		
	12 horas Total	

Desarrollo

Tarea	Esfuerzo estima...	Depende de
Preparación y transformación del dataset	7 horas	Búsqueda del dataset, Definir la arquitectura de la CNN
Construcción del modelo de CNN	6,3 horas	-
Entrenamiento y optimización del modelo	20,8 horas	Construcción del modelo de CNN
Hacer predicciones y conseguir resultados	3,5 horas	Entrenamiento y optimización del modelo
Implementación de la aplicación web	4,9 horas	Hacer predicciones y conseguir resultados
+ Agregar Tarea		
	42,5 horas Total	

Ilustración 58: Desglose de tareas del proyecto 2



Documentación

Tarea	Esfuerzo estima...	Depende de
Redacción de la memoria final	300 horas	-
+ Agregar Tarea		
	300 horas Total	

Ilustración 59: Desglose de tareas del proyecto 3

5.1.2. Diagrama de Gantt

El diagrama de Gantt, muy usado en la gestión de proyectos, es un gráfico de barras horizontales que se usa para ilustrar el cronograma de un proyecto, programa o trabajo. Es una forma de visualizar la programación de tu proyecto, de dar seguimiento a los logros y de estar siempre familiarizado con el cronograma de tu trabajo [53].

Las tareas detalladas en el apartado anterior, junto con el esfuerzo estimado en horas a cada una de ellas, se muestran en el siguiente diagrama de Gantt. Cada tarea se representa como una barra horizontal en el gráfico, donde la longitud de la barra indica la duración estimada de la tarea.

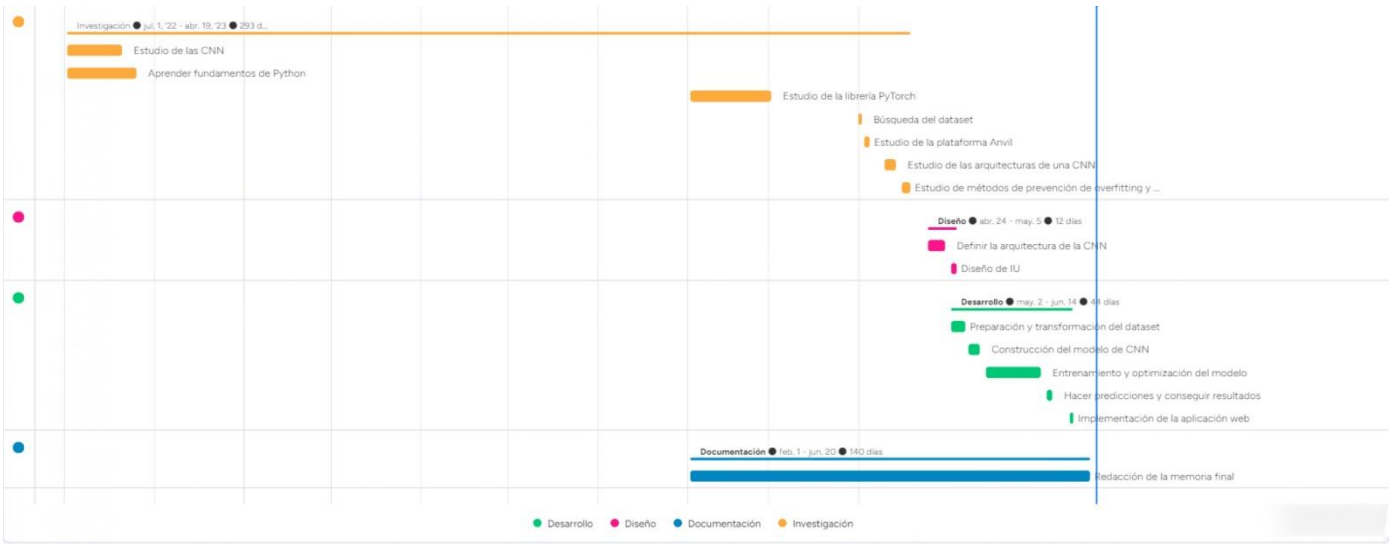


Ilustración 60: Diagrama de Gantt

5.2. Desarrollo / Desarrollo de la red neuronal

5.2.1. Dataset

El dataset utilizado en este proyecto, conocido como "Brain Tumor Classification (MRI)" [54], fue obtenido de la plataforma Kaggle. Kaggle es una reconocida comunidad en línea que alberga conjuntos de datos y competencias de aprendizaje automático.

Características del dataset

El dataset Brain Tumor MRI incluye una colección de imágenes de resonancia magnética (MRI) del cerebro, específicamente enfocadas en la detección y clasificación de tumores cerebrales.

Este dataset contiene un total de 3264 imágenes de resonancia magnética del cerebro humano y se divide en dos partes principales: entrenamiento (training dataset) y prueba (testing dataset). El conjunto de entrenamiento comprende 2870 imágenes, mientras que el conjunto de prueba consta de 394 imágenes.

Ambos conjuntos se clasifican en cuatro categorías, correspondientes a los diferentes tipos de tumores cerebrales: glioma, meningioma, no tumor y pituitary.

La división en entrenamiento y prueba permite utilizar el conjunto de datos de entrenamiento para entrenar y ajustar el modelo de detección y clasificación de tumores cerebrales, mientras que el conjunto de datos de prueba se utiliza para evaluar y medir el rendimiento y precisión del modelo.

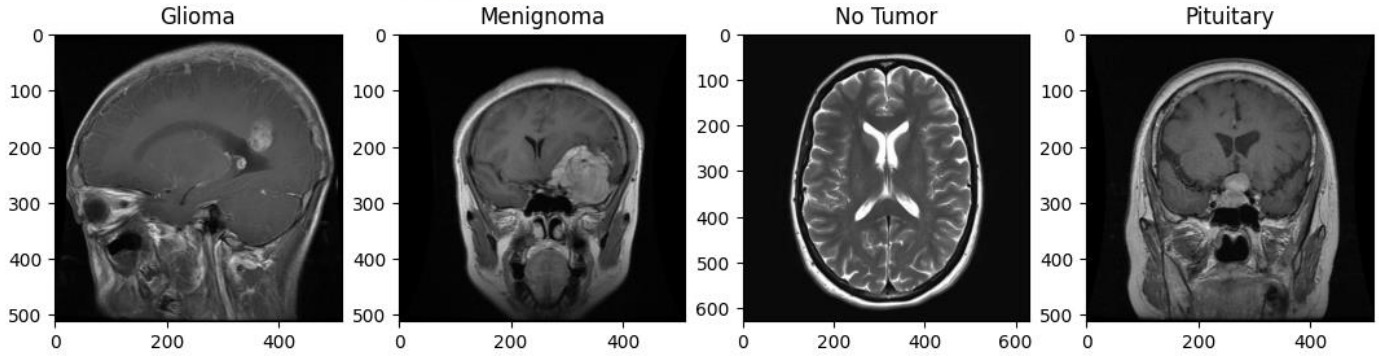


Ilustración 61: Ejemplos de las 4 categorías del dataset

5.2.2. Métricas

En un problema de clasificación, solo hay cuatro posibles resultados al aplicar el clasificador a cualquier instancia. Estos resultados son [55]:

- Verdaderos positivos (VP): Se refiere a las imágenes de resonancia magnética que son correctamente identificadas y clasificadas como tumores cerebrales de acuerdo con su tipo.

- Verdaderos negativos (VN): Representa el número de imágenes que son clasificadas correctamente como "no tumor".

- Falsos positivos (FP): Se refiere a las instancias que son incorrectamente clasificadas como tumores cerebrales. Esto sucede cuando el clasificador etiqueta erróneamente una muestra como perteneciente a un tipo específico de tumor cerebral, cuando en realidad no lo es.

- Falsos negativos (FN): Hace referencia a las imágenes de resonancia magnética que son incorrectamente clasificadas como "no tumor" cuando, en realidad, presentan un tumor cerebral. En estos casos, el clasificador no logra detectar correctamente la presencia de un tumor cerebral.

Estos resultados se representan en la matriz de confusión, una de las técnicas más importantes para resumir el rendimiento de la clasificación de imágenes. También se conoce como matriz de errores, un diseño tabular que permite visualizar el rendimiento de las predicciones y las etiquetas verdaderas en el modelo de clasificación. En el análisis predictivo, cada fila de la matriz de confusión representa las instancias de una clase real, y cada columna representa una instancia en la clase predicha y viceversa. Los elementos diagonales de la matriz de errores representan predicciones de clasificación correctas, y los elementos no diagonales representan predicciones incorrectas de clasificación de imágenes [56].

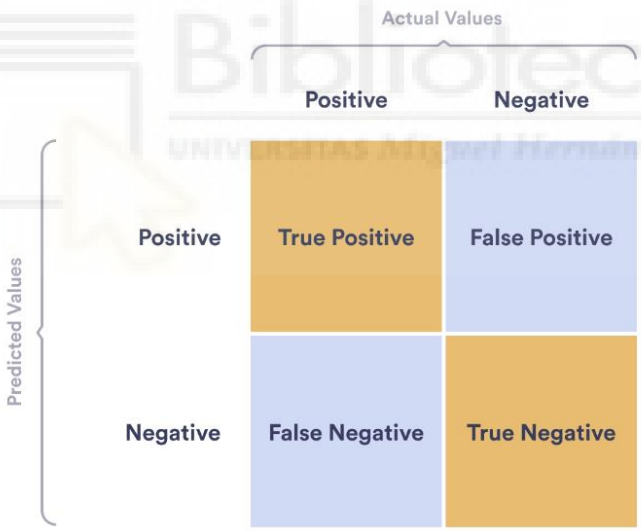


Ilustración 62: Matriz de confusión

Las métricas utilizadas para calcular el rendimiento del modelo propuesto incluyen exactitud, precisión, exhaustividad y valor-F.

1. **Exactitud (Accuracy)**

   La exactitud mide la proporción de muestras clasificadas correctamente sobre el total de muestras. Se calcula utilizando la siguiente fórmula:

$$Accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

2. **Precisión (Precision)**

La precisión mide la proporción de muestras clasificadas como positivas que son realmente positivas. Se calcula con la siguiente fórmula:

$$Precision = \frac{VP}{VP + FP}$$

3. **Exhaustividad (Sensitivity o Recall)**

La sensibilidad representa la proporción de todos los ejemplos positivos que se dividen y mide la capacidad del clasificador para identificar casos positivos [55]. Se calcula utilizando la fórmula indicada a continuación:

$$Recall = \frac{VP}{VP + FN}$$

4. **Valor-F (F1-score)**

La medida F1 es una métrica que combina la precisión y la sensibilidad para proporcionar una evaluación más completa del rendimiento de un modelo de clasificación. Se calcula haciendo uso de la siguiente fórmula:

$$F1 - score = \frac{2 \times Precision \times Recall}{Recall + Precision}$$

5.2.3. Preprocesamiento de imágenes y Aumento de datos (Data Augmentation)

Preprocesamiento de imágenes

El preprocesamiento de imágenes se refiere a las técnicas y pasos aplicados a las imágenes antes de ser utilizadas en un modelo o algoritmo de visión por computadora. Consiste en manipular y transformar las imágenes con el objetivo de mejorar su calidad, reducir el ruido, corregir imperfecciones y adaptarlas para su posterior análisis.

Algunas de las técnicas de preprocesamiento utilizadas en este trabajo son las siguientes:

- Redimensionamiento: Todas las imágenes del conjunto de datos se redimensionaron a un tamaño común de 224 × 224 píxeles.

- Transformación a Tensor: Las imágenes se convirtieron en tensores, una técnica común en PyTorch, que las representa en el formato (C * H * W) [57].

- Normalización: Se aplicó la normalización a los tensores de las imágenes para mejorar la preparación de los datos. Esta técnica ajusta los valores de los píxeles a un rango específico, lo que facilita el entrenamiento del modelo.

Data Augmentation

El aumento de datos es un conjunto de técnicas utilizadas para aumentar la cantidad de datos en un modelo de aprendizaje automático mediante la adición de copias ligeramente modificadas de datos existentes o la creación de datos sintéticos a partir de datos existentes. Ayuda a suavizar el modelo de aprendizaje automático y reducir el overfitting de los datos [58].

Las técnicas de Data Augmentation utilizadas en este proyecto son las siguientes:

- Random Horizontal Flip: Esta técnica se utiliza para realizar una inversión horizontal aleatoria en la imagen proporcionada con una probabilidad dada [59].

- Random Rotation: La imagen de origen se rota aleatoriamente en sentido horario o antihorario por un cierto número de grados, cambiando la posición del objeto en el marco [60].

- Random Resized Crop: Es un tipo de aumento de datos de imágenes en el que se realiza un recorte de tamaño aleatorio del tamaño original y una relación de aspecto aleatoria de la relación de aspecto original. Este recorte se redimensiona finalmente al tamaño especificado [61].

- Color Jitter: Esta técnica aplica pequeñas perturbaciones o variaciones aleatorias en los valores de color de una imagen, como el brillo, contraste, saturación y tono.

- Random Apply: Aplica de manera aleatoria una lista de transformaciones con una probabilidad dada [62].

- RandAugment: Es un método que selecciona aleatoriamente una combinación de transformaciones de imagen con igual probabilidad para cada imagen en un minibatch.

5.2.4. Arquitectura de la red

La arquitectura de la red convolucional utilizada está inspirada en la reconocida red VGG-16. Sin embargo, se ha simplificado para adaptarse a los requisitos específicos del problema abordado. A continuación, se describe detalladamente la estructura de la red convolucional:

- Capa de entrada: Esta capa recibe las imágenes de entrada y establece las dimensiones iniciales del tensor de entrada.

- Bloques convolucionales: La red consta de 5 bloques convolucionales. Cada bloque convolucional realiza las siguientes operaciones secuenciales:

    - Capa convolucional: Esta capa realiza operaciones de convolución en los datos de entrada para extraer características relevantes.

- Capa ReLU: Después de cada capa convolucional, se aplica una función de activación ReLU para introducir la no linealidad en la red y mejorar la capacidad de aprendizaje.

- Capa de max pooling: Esta capa reduce la resolución espacial de los mapas de características, seleccionando el valor máximo en cada vecindario de la imagen.

- Capa clasificadora: Después de los bloques convolucionales, se utiliza una capa clasificadora para producir la salida final. La capa clasificadora consta de las siguientes capas:

    - Capa Flatten: Esta capa convierte los mapas de características en un vector unidimensional, para ser procesados por capas completamente conectadas.

    - Funciones de activación ReLU y capas de Dropout: introducen no linealidad y regularización en la red, respectivamente.

    - Capa Lineal: utiliza una función de activación específica, la función softmax para problemas de clasificación multiclase como el presente. Esta función transforma las salidas de la red en una distribución de probabilidades para cada clase objetivo.

Además de la estructura de la red, se deben considerar los hiperparámetros puesto que controlan el comportamiento general del modelo. Su objetivo es minimizar la función de pérdida predefinida y obtener mejores resultados para un conjunto de datos específico. Algunos de los hiperparámetros a ajustar son el número de neuronas y épocas, la función de activación softmax, la tasa de aprendizaje y el optimizador [56].

Layer (type (var_name))	Input Shape	Output Shape	Param #	Trainable
BrainTumorV0 (BrainTumorV0)	[32, 3, 224, 224]	[32, 4]	--	True
├─Sequential (block_1)	[32, 3, 224, 224]	[32, 64, 112, 112]	--	True
│ └─Conv2d (0)	[32, 3, 224, 224]	[32, 64, 224, 224]	1,792	True
│ └─ReLU (1)	[32, 64, 224, 224]	[32, 64, 224, 224]	--	--
│ └─MaxPool2d (2)	[32, 64, 224, 224]	[32, 64, 112, 112]	--	--
├─Sequential (block_2)	[32, 64, 112, 112]	[32, 64, 56, 56]	--	True
│ └─Conv2d (0)	[32, 64, 112, 112]	[32, 64, 112, 112]	36,928	True
│ └─ReLU (1)	[32, 64, 112, 112]	[32, 64, 112, 112]	--	--
│ └─MaxPool2d (2)	[32, 64, 112, 112]	[32, 64, 56, 56]	--	--
├─Sequential (block_3)	[32, 64, 56, 56]	[32, 64, 28, 28]	--	True
│ └─Conv2d (0)	[32, 64, 56, 56]	[32, 64, 56, 56]	36,928	True
│ └─ReLU (1)	[32, 64, 56, 56]	[32, 64, 56, 56]	--	--
│ └─MaxPool2d (2)	[32, 64, 56, 56]	[32, 64, 28, 28]	--	--
├─Sequential (block_4)	[32, 64, 28, 28]	[32, 64, 14, 14]	--	True
│ └─Conv2d (0)	[32, 64, 28, 28]	[32, 64, 28, 28]	36,928	True
│ └─ReLU (1)	[32, 64, 28, 28]	[32, 64, 28, 28]	--	--
│ └─MaxPool2d (2)	[32, 64, 28, 28]	[32, 64, 14, 14]	--	--
├─Sequential (block_5)	[32, 64, 14, 14]	[32, 64, 7, 7]	--	True
│ └─Conv2d (0)	[32, 64, 14, 14]	[32, 64, 14, 14]	36,928	True
│ └─ReLU (1)	[32, 64, 14, 14]	[32, 64, 14, 14]	--	--
│ └─MaxPool2d (2)	[32, 64, 14, 14]	[32, 64, 7, 7]	--	--
├─Sequential (classifier)	[32, 64, 7, 7]	[32, 4]	--	True
│ └─Flatten (0)	[32, 64, 7, 7]	[32, 3136]	--	--
│ └─ReLU (1)	[32, 3136]	[32, 3136]	--	--
│ └─Dropout (2)	[32, 3136]	[32, 3136]	--	--
│ └─Linear (3)	[32, 3136]	[32, 64]	200,768	True
│ └─ReLU (4)	[32, 64]	[32, 64]	--	--
│ └─Dropout (5)	[32, 64]	[32, 64]	--	--
│ └─Linear (6)	[32, 64]	[32, 4]	260	True

Total params: 350,532
Trainable params: 350,532
Non-trainable params: 0
Total mult-adds (G): 22.57

Input size (MB): 19.27
Forward/backward pass size (MB): 1095.06
Params size (MB): 1.40
Estimated Total Size (MB): 1115.73

Ilustración 63: Arquitectura base de la CNN

5.2.5. Overfitting y Underfitting

En el proceso de entrenamiento del modelo, se realizan modificaciones en base a las curvas de pérdida para abordar los problemas de subajuste (underfitting) y sobreajuste (overfitting).

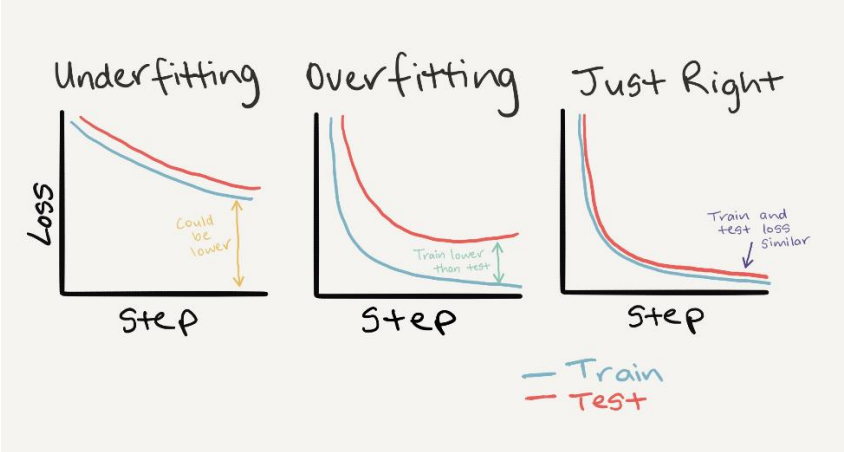


Ilustración 64: Ejemplos de curvas de pérdida

El underfitting ocurre cuando un modelo no es capaz de hacer predicciones precisas basadas en los datos de entrenamiento y, por lo tanto, no tiene la capacidad de generalizar correctamente en nuevos datos. Los modelos con underfitting tienden a tener un rendimiento deficiente tanto en los conjuntos de entrenamiento como en los conjuntos de prueba [63].

Para abordar este problema, se pueden aplicar técnicas como:

- Agregar más capas o neuronas
- Ajustar los hiperparámetros
- Usar Transfer Learning: El conocimiento de un modelo de aprendizaje automático previamente entrenado se aplica a un problema diferente pero relacionado [64]. Esta técnica ayudar a mitigar problemas de overfitting y underfitting.
- Entrenar durante más tiempo

Un modelo se considera overfitting cuando tiene un excelente desempeño en los datos de entrenamiento, pero no logra generalizar correctamente en los datos de validación. Esto se debe a que el modelo aprende demasiado de los datos de entrenamiento, incluyendo el ruido y patrones específicos, lo cual afecta negativamente su rendimiento en nuevos datos [63].

Algunas de las técnicas comunes utilizadas para superar el overfitting incluyen:

- Regularización: Se refiere a una variedad de técnicas para simplificar tu modelo. Por ejemplo, se puede agregar un parámetro de penalización en una regresión (regularización L1 y L2)
- Usar Capas Dropout: Este tipo de capas desactivan aleatoriamente algunas neuronas, lo que evita que el modelo dependa demasiado de ellas y promueve la generalización del aprendizaje.
- Data augmentation

- Early stopping: Se refiere a detener el entrenamiento del modelo antes de que comience a sobreajustarse a los datos de entrenamiento, evitando así un deterioro en su capacidad de generalización [63].
- Learning Rate Decay: Es una técnica ampliamente utilizada para entrenar redes neuronales modernas. Comienza con una tasa de aprendizaje alta y luego la reduce varias veces. Se ha observado empíricamente que esto ayuda tanto a la optimización como a la generalización [65].

5.2.6. Entrenamiento de la red

En cada entrenamiento, se proporcionará una descripción detallada de la estructura de la red neuronal utilizada, incluyendo información sobre las capas y su disposición. Además, se presentarán los valores específicos de los hiperparámetros ajustados en la red, como la tasa de aprendizaje, el tamaño del lote y otros parámetros relevantes. Esto permitirá tener una visión completa de cómo se ha diseñado y configurado la red neuronal en cada entrenamiento, brindando una mayor comprensión de su funcionamiento y los resultados obtenidos.

5.2.6.1. Entrenamiento 1: Modelo base

Durante este primer entrenamiento, la CNN se ha entrenado utilizando la arquitectura mencionada previamente sin aplicar ninguna técnica de Data Augmentation, únicamente empleando técnicas de preprocesamiento de imágenes.

Los hiperparámetros utilizados son los siguientes:

Hiperparámetro	Valor
Input layer shape	3
Output layer shape	4
Capas ocultas	4
Neuronas por capa oculta	64
Función de activación capa oculta	ReLU
Función de activación capa salida	Softmax
Función de pérdida	CrossEntropyLoss
Oprimizador	Adam
Learning rate	0,001
Batch size	32
Epochs	100

Tabla 1: Hiperparámetros de Entrenamiento 1

Resultados obtenidos

En las gráficas obtenidas de este modelo de red convolucional, se observa la evolución de las funciones de pérdida (loss) y exactitud (accuracy) a medida que se incrementan los epochs. En particular, se ha identificado un fenómeno de overfitting a medida que transcurren los epochs.

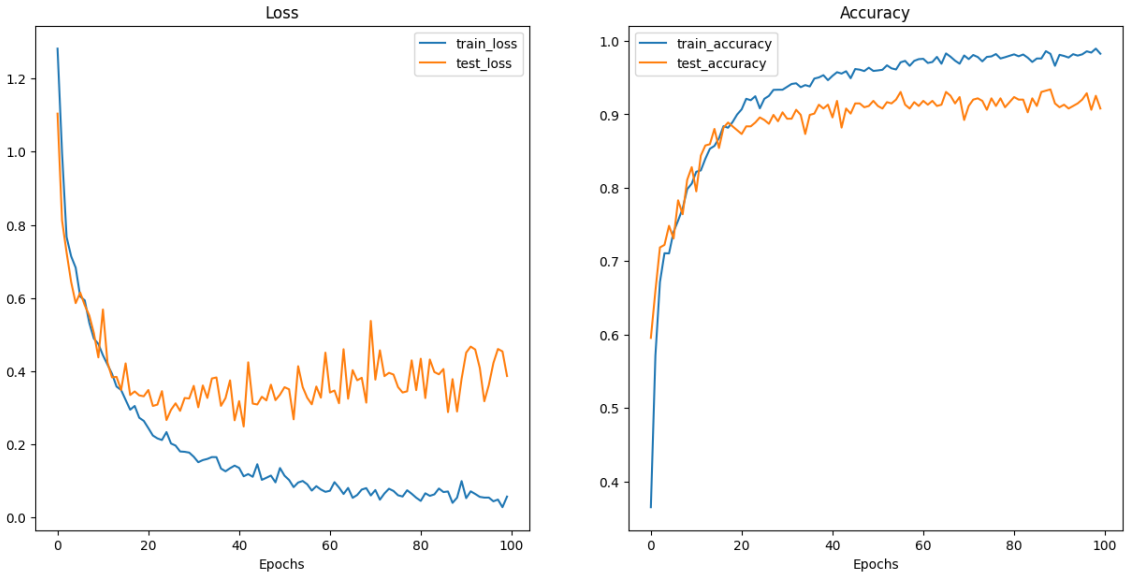


Ilustración 65: Gráficas del Entrenamiento 1

En la gráfica de pérdida, se puede ver que la pérdida en el conjunto de entrenamiento disminuye constantemente, mientras que, en el conjunto de validación, la pérdida inicialmente sigue una tendencia similar pero luego comienza a aumentar. Esta discrepancia entre las líneas de entrenamiento y validación sugiere que el modelo está sobreajustándose a los datos de entrenamiento y no logra generalizar adecuadamente en nuevos datos.

En la gráfica de exactitud, se evidencia una tendencia similar. La exactitud en el conjunto de entrenamiento aumenta a medida que se incrementan los epochs, pero en el conjunto de validación, después de un cierto punto, la exactitud se estanca o incluso disminuye. Estos patrones revelan que el modelo no está generalizando correctamente y está demasiado ajustado a los datos de entrenamiento.

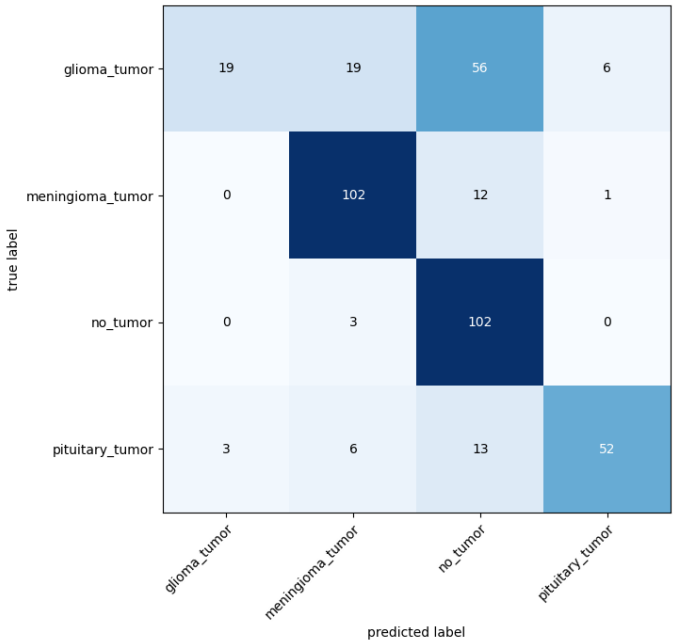


Ilustración 66: Matriz de confusión del Entrenamiento 1

```
print(f"Accuracy: {acc_value:.4f}")
print(f"Precision: {precision_value:.4f}")
print(f"Recall: {recall_value:.4f}")
print(f"F1 Score: {f1_value:.4f}")
```

```
Accuracy: 0.6878
Precision: 0.7717
Recall: 0.6878
F1 Score: 0.6980
```

Ilustración 67: Métricas del Entrenamiento 1

Al examinar la matriz de confusión, se observa que el modelo tiene dificultades para distinguir adecuadamente los tumores de glioma y tiende a clasificarlos incorrectamente como no_tumor o meningioma_tumor en la mayoría de los casos. Sin embargo, muestra un mejor desempeño en la clasificación de los tumores de meningioma y no_tumor, con una exactitud relativamente alta.

5.2.6.2. Entrenamiento 2: Modelo con Data Augmentation

En este segundo entrenamiento, se ha aplicado la técnica de Data Augmentation a la red. El objetivo es mejorar el rendimiento y la capacidad de generalización del modelo al introducir variaciones y diversidad en los datos de entrenamiento.

Se ha utilizado la arquitectura descrita anteriormente, pero esta vez se han aplicado técnicas de aumento de datos adaptadas específicamente al problema de clasificación de tumores cerebrales. Estas técnicas incluyen transformaciones a las imágenes de entrenamiento, como rotaciones o cambios de escala y la realización de un eliminado aleatorio, como ya se mencionó en el apartado Data Augmentation.

Los hiperparámetros utilizados han sido los mismos que en el entrenamiento anterior para mantener la consistencia en la comparación de resultados.

Resultados obtenidos

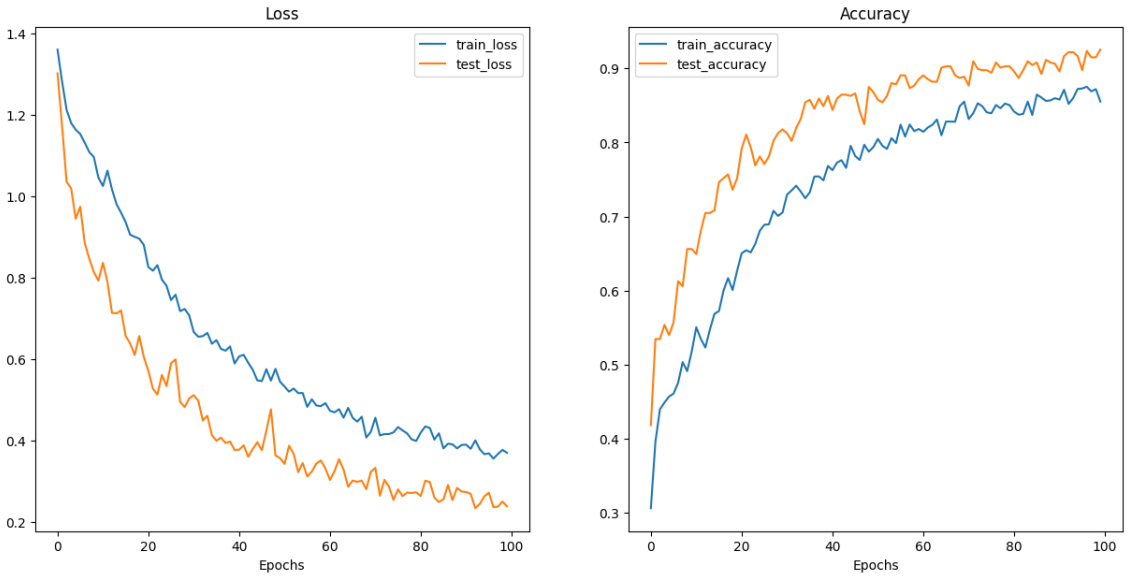


Ilustración 68: Graficas del Entrenamiento 2

Las gráficas muestran que, al principio del entrenamiento, la pérdida en la validación es menor que en el entrenamiento, y la exactitud en la validación es mayor que en el entrenamiento. Sin embargo, alrededor del epoch 60, no se observa una mejora significativa en ninguna de las dos métricas en ambos conjuntos de datos.

El análisis de las gráficas de pérdida y exactitud muestra una diferencia notable en comparación con el entrenamiento anterior. El modelo actual exhibe un mejor desempeño en los datos de validación en comparación con los datos de entrenamiento. Esto puede ser interpretado de diferentes maneras: como una mejor generalización del modelo en datos no vistos durante la validación, o como un posible sobreajuste en los datos de entrenamiento que limita su capacidad para generalizar en nuevos datos.

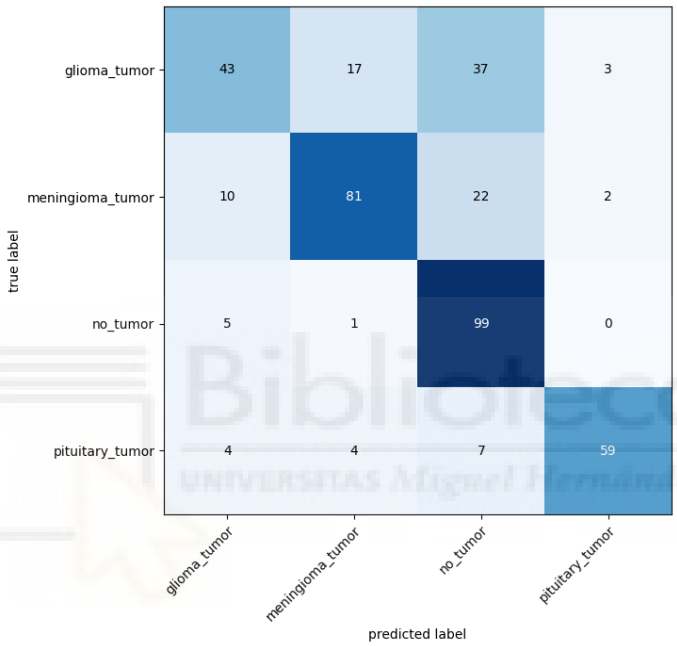


Ilustración 69: Matriz de confusión del Entrenamiento 2

```
print(f"Accuracy: {acc_value:.4f}")
print(f"Precision: {precision_value:.4f}")
print(f"Recall: {recall_value:.4f}")
print(f"F1 Score: {f1_value:.4f}")
```

```
Accuracy: 0.7186
Precision: 0.7505
Recall: 0.7186
F1 Score: 0.7157
```

Ilustración 70: Métricas del Entrenamiento 2

La matriz de confusión muestra mejoras significativas en la clasificación de glioma_tumor y pituitary_tumor. Además, la exactitud de este modelo ha aumentado a 71,86%, superando el 68,78% del entrenamiento anterior. Estos resultados reflejan un progreso prometedor en la capacidad del modelo para distinguir entre diferentes tipos de tumores cerebrales.

5.2.6.3. Entrenamiento 3: Modelo con mejoras de código

En el segundo entrenamiento, se han realizado mejoras significativas en el modelo mediante la adición de capas de Batch Normalization 2D en cada bloque convolucional y un bloque convolucional adicional a la arquitectura existente.

La capa BatchNorm2 es una técnica de normalización que se aplica después de la capa de convolución en la red neuronal. Su objetivo principal es mejorar la estabilidad y acelerar el proceso de entrenamiento al normalizar las activaciones intermedias en cada bloque convolucional. Esto ayuda a mitigar los efectos negativos de gradientes desvanecientes o explosivos, y facilita un aprendizaje más rápido y eficiente.

Las nuevas adiciones buscan mejorar la precisión y rendimiento del modelo. Al aplicar normalización y aumentar la profundidad de la red, se espera reducir el sesgo y mejorar la capacidad de generalización, lo que resulta en una mejor clasificación de los tumores.

Los hiperparámetros utilizados son los siguientes:

Hiperparámetro	Valor
Input layer shape	3
Output layer shape	4
Capas ocultas	5
Neuronas por capa oculta	64
Función de activación capa oculta	ReLU
Función de activación capa salida	Softmax
Función de pérdida	CrossEntropyLoss
Oprimizador	Adam
Learning rate	0,001
Batch size	32
Epochs	60

Tabla 2: Hiperparámetros de Entrenamiento 2

Resultados obtenidos

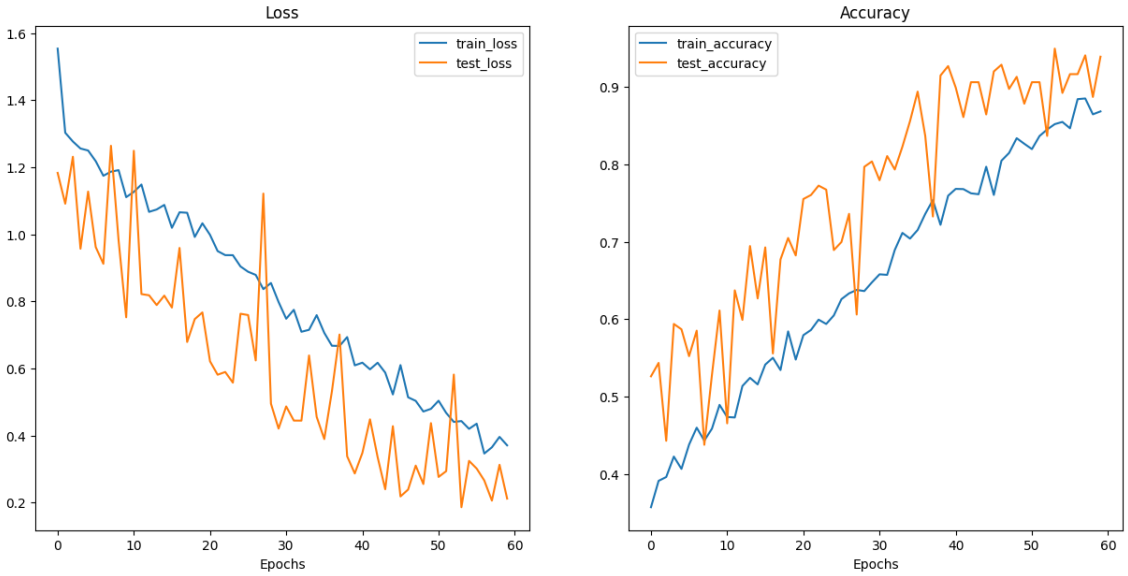


Ilustración 71: Gráficas del Entrenamiento 3

En este entrenamiento, se observan picos en las gráficas de pérdida y exactitud durante la validación, evidenciando fluctuaciones en el desempeño del modelo en diferentes momentos. Un pico significativo ocurre alrededor del epoch 30, con aumento de la pérdida y disminución de la exactitud, posiblemente debido a sobreajuste u obstáculos en el entrenamiento. La línea de pérdida en el entrenamiento es más estable que en la validación, sugiriendo un aprendizaje más constante, pero dificultades en la generalización de los datos de validación. Aunque la validación muestra mejores resultados, podrían estar influenciados por el sobreajuste o características específicas del conjunto de datos utilizado.

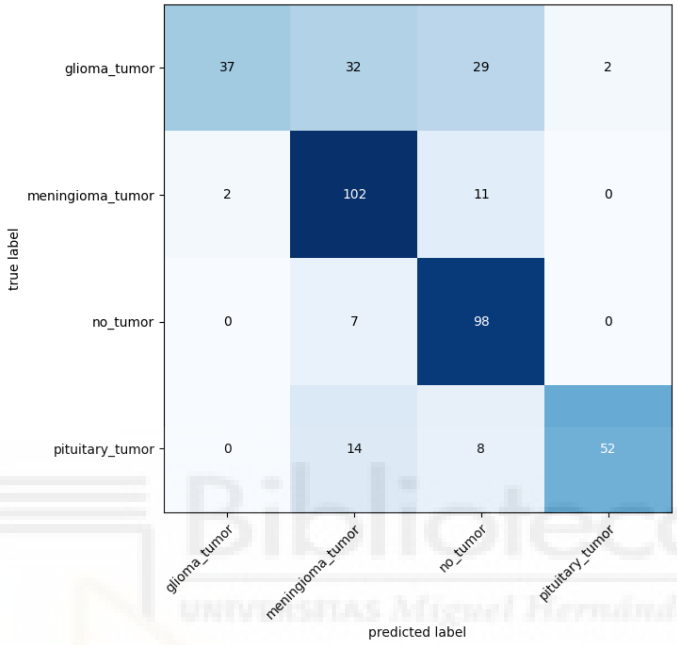


Ilustración 72: Matriz de confusión del Entrenamiento 3

```
18
19 print(f"Accuracy: {acc_value:.4f}")
20 print(f"Precision: {precision_value:.4f}")
21 print(f"Recall: {recall_value:.4f}")
22 print(f"F1 Score: {f1_value:.4f}")
```

```
Accuracy: 0.7232
Precision: 0.8102
Recall: 0.7232
F1 Score: 0.7335
```

Ilustración 73: Métricas del Entrenamiento 3

La matriz de confusión revela cambios sutiles en la clasificación de glioma_tumor y pituitary_tumor en comparación con los entrenamientos anteriores. Para glioma_tumor, se ha notado una ligera disminución en la precisión de la clasificación, indicando un rendimiento ligeramente inferior. En el caso de pituitary_tumor, también ha mostrado un ligero deterioro, pasando de predecir correctamente 59 casos a solo 51. Es importante destacar que la predicción de meningioma_tumor ha mostrado una mejora significativa.

En general, el modelo ha logrado un pequeño incremento en su exactitud, pasando del 71,86% anterior al 72,32%, lo que sugiere un rendimiento general mejorado en la clasificación de las diferentes clases.

5.2.6.4. Entrenamiento 4: Modelo con Regularization Decay

Durante el entrenamiento previo, se observó que la función de pérdida del modelo mostraba fluctuaciones significativas, lo que sugiere que las neuronas relacionadas tenían pesos excesivamente altos. Esta situación señala que el modelo tenía una complejidad elevada, lo que aumenta el riesgo de overfitting.

Para abordar este problema, se ha implementado la técnica de Regularization Decay. Esta técnica ayuda a reducir la complejidad del modelo restringiendo los valores de las neuronas, evitando que sean excesivamente grandes. Al limitar el valor de las neuronas, se logra mitigar en cierta medida la aparición del overfitting [66].

Además, se ha prolongado el tiempo de entrenamiento al aumentar el número de epochs. Esta extensión permite al modelo aprender patrones más complejos y ajustarse mejor a los datos.

Los hiperparámetros utilizados son los siguientes:

Hiperparámetro	Valor
Input layer shape	3
Output layer shape	4
Capas ocultas	5
Neuronas por capa oculta	64
Función de activación capa oculta	ReLU
Función de activación capa salida	Softmax
Función de pérdida	CrossEntropyLoss
Oprimizador	Adam
Learning rate	0,001
Batch size	32
Epochs	85

Tabla 3: Hiperparámetros de Entrenamiento 3

Resultados obtenidos

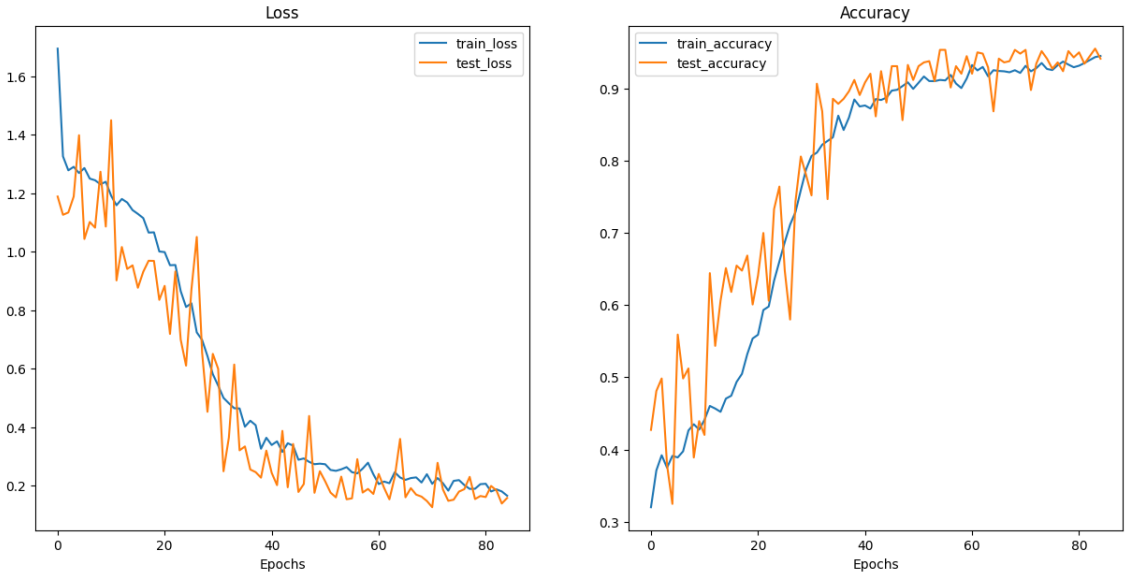


Ilustración 74: Gráficas del Entrenamiento 4

En las gráficas de los y accuracy se observa que los picos, aunque aún están presentes, no alcanzan la misma altura que antes. Esto muestra que el modelo ha logrado una mayor estabilidad en el desempeño durante el entrenamiento.

Una diferencia notable es que las líneas correspondientes al entrenamiento y la validación están más cercanas entre sí en términos de valores de loss y accuracy. Esto sugiere que el modelo ha aprendido a generalizar mejor, es decir, a tener un desempeño más equilibrado tanto en los datos de entrenamiento como en los de validación. La reducción de la brecha entre las líneas de training y validation indica una menor discrepancia en el rendimiento del modelo en ambos conjuntos de datos.

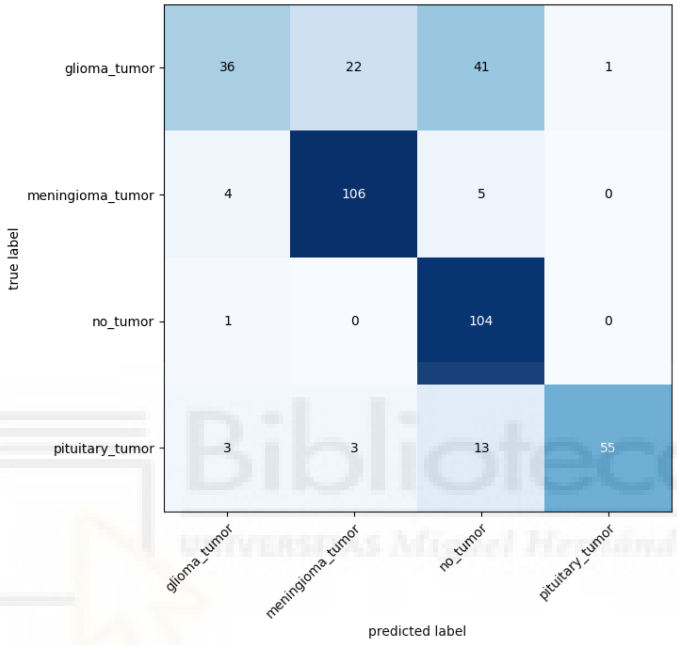


Ilustración 75: Matriz de confusión del Entrenamiento 4

```
19 print(f"Accuracy: {acc_value:.4f}")
20 print(f"Precision: {precision_value:.4f}")
21 print(f"Recall: {recall_value:.4f}")
22 print(f"F1 Score: {f1_value:.4f}")
```

```
Accuracy: 0.7539
Precision: 0.8119
Recall: 0.7539
F1 Score: 0.7640
```

Ilustración 76: Métricas del Entrenamiento 4

En la matriz de confusión, se evidencia que la predicción de glioma_tumor se mantiene constante, mientras que meningioma_tumor, no_tumor y pituitary_tumor han experimentado mejoras en sus predicciones. Estos cambios han llevado a un aumento en el accuracy general del modelo, que ha pasado del 72,32% al 75,39%.

5.2.6.5. Entrenamiento adicional: Modelo con Transfer Learning

En este entrenamiento adicional, se ha explorado la técnica de Transfer Learning y su aplicación en este problema de clasificación. El Transfer Learning es un método de aprendizaje automático en el que un modelo desarrollado para una tarea en concreto se puede usar como punto de partida para el desarrollo de otro modelo con otra tarea distinta de la primera [67]. Esta técnica permite aprovechar el conocimiento adquirido por el modelo en tareas anteriores para mejorar la generalización y obtener mejores resultados en una tarea específica.

Se ha utilizado el Transfer Learning con el objetivo de mejorar las predicciones del problema de clasificación. Para ello, se han empleado dos modelos pre-entrenados ampliamente utilizados en el campo de la visión por computadora: VGG16 y EfficientNet.

Estos modelos se destacan por su capacidad para lograr un alto rendimiento con una cantidad relativamente baja de parámetros, lo que los hace adecuados para aplicaciones con recursos limitados.

Los hiperparámetros utilizados son los siguientes:

Hiperparámetro	Valor
Output layer shape	4
Función de pérdida	CrossEntropyLoss
Oprimizador	Adam
Learning rate	0,001
Batch size	32
Epochs VGG16	9
Epochs EfficientNet	20

Tabla 4: Hiperparámetros de Entrenamiento adicional

Resultados obtenidos con VGG16

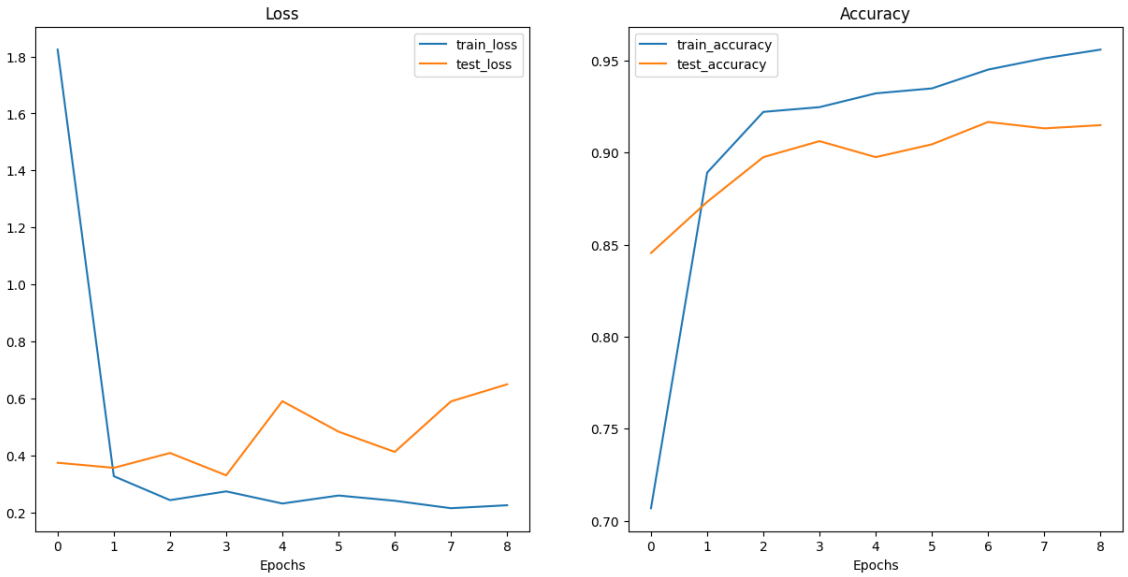


Ilustración 77: Gráficas del modelo VGG16

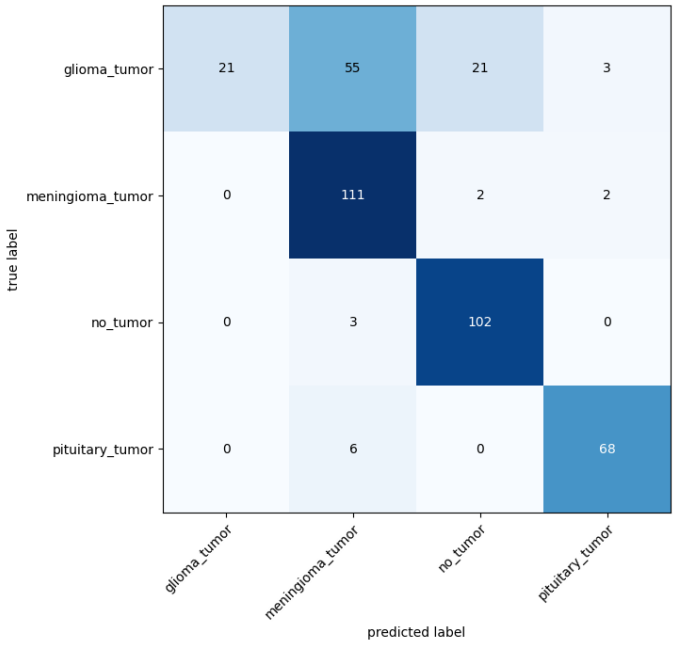


Ilustración 78: Matriz de confusión del modelo VGG16

```
19 print(f"Accuracy: {acc_value:.4f}")
20 print(f"Precision: {precision_value:.4f}")
21 print(f"Recall: {recall_value:.4f}")
22 print(f"F1 Score: {f1_value:.4f}")

Accuracy: 0.7664
Precision: 0.8454
Recall: 0.7664
F1 Score: 0.7665
```

Ilustración 79: Métricas del modelo VGG16

Resultados obtenidos con EfficientNet

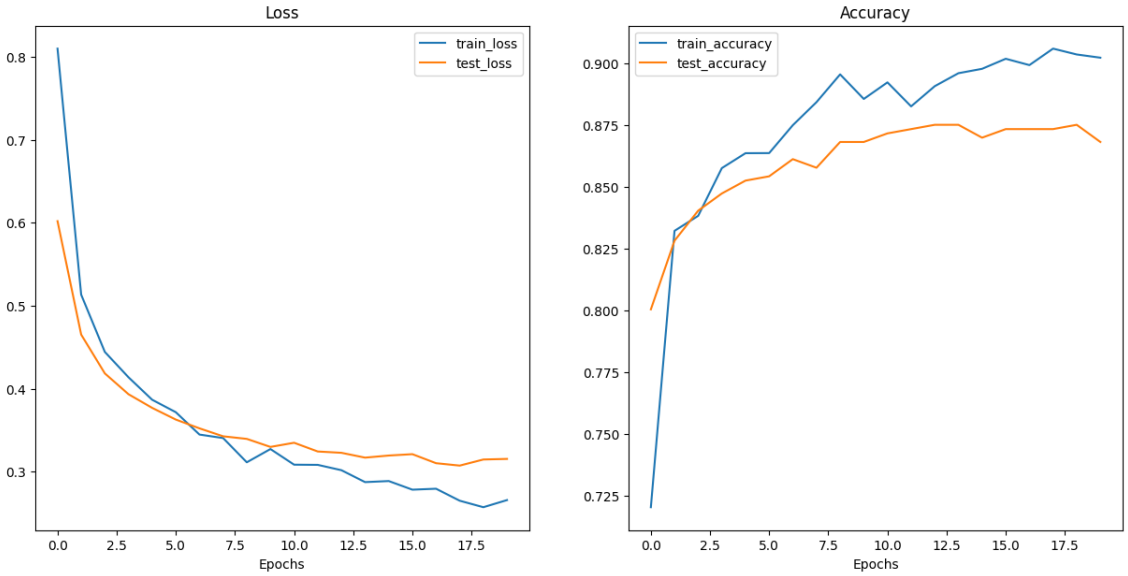


Ilustración 80: Gráficas del modelo EfficientNet

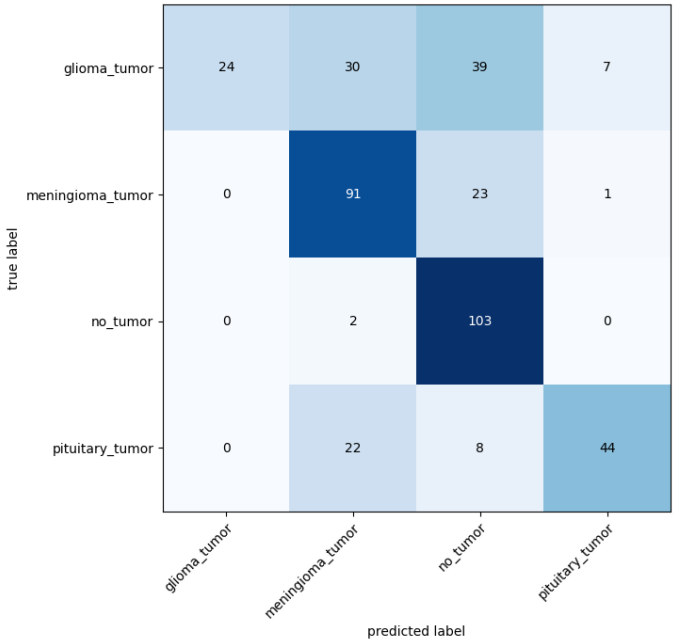


Ilustración 81: Matriz de confusión del modelo EfficientNet

```
19 print(f"Accuracy: {acc_value:.4f}")
20 print(f"Precision: {precision_value:.4f}")
21 print(f"Recall: {recall_value:.4f}")
22 print(f"F1 Score: {f1_value:.4f}")

Accuracy: 0.6517
Precision: 0.7673
Recall: 0.6517
F1 Score: 0.6650
```

Ilustración 82: Métricas del modelo EfficientNet

El análisis de los resultados de los modelos VGG16 y EfficientNet entrenados mediante Transfer Learning revela algunas observaciones importantes. En primer lugar, el modelo EfficientNet presenta un menor nivel de exactitud en comparación con los entrenamientos anteriores, con un accuracy del 65,17%. Esto muestra que el modelo no está logrando clasificar correctamente algunas clases, particularmente la clase glioma_tumor. Además, se observa en la matriz de confusión que el modelo tiene dificultades en la clasificación de la clase pituitary_tumor, detectando 22 casos como meningioma_tumor de los 74.

Por otro lado, el modelo VGG16 muestra un accuracy ligeramente mayor en comparación con los entrenamientos anteriores, con un valor del 76,64%. Sin embargo, al examinar la matriz de confusión, se evidencia que el modelo también tiene dificultades en la clasificación de la clase glioma_tumor. Se detectaron 55 casos de glioma_tumor como meningioma_tumor de los 100 totales, mientras que solo clasificó correctamente 21 casos.

Estos resultados sugieren que el conocimiento transferido de los modelos preentrenados (VGG16 y EfficientNet) no fue suficiente para capturar las características relevantes y complejas de los tumores cerebrales en este conjunto de datos específico.

5.2.7. Reutilización de la red con otro dataset

El dataset utilizado para reutilizar el modelo del Entrenamiento 4 es "CT KIDNEY DATASET: Normal-Cyst-Tumor and Stone" [68]. El conjunto de datos consta de imágenes de tomografía computarizada (CT) de pacientes con diagnósticos de 4 clases diferentes, incluyendo: Cyst (quiste), Normal, Stone y Tumor.

Estas clases representan diferentes condiciones y hallazgos relacionados con los riñones. El conjunto de datos contiene un total de 12,446 imágenes únicas, con 3,709 casos de quistes, 5,077 normales, 1,377 de piedras y 2,283 casos de tumores renales.

Los hiperparámetros utilizados son los siguientes:

Hiperparámetro	Valor
Output layer shape	4
Función de pérdida	CrossEntropyLoss
Oprimizador	Adam
Learning rate	0,001
Batch size	32
Epochs	5

Tabla 5: Hiperparámetros de la reutilización de la red

Resultados obtenidos

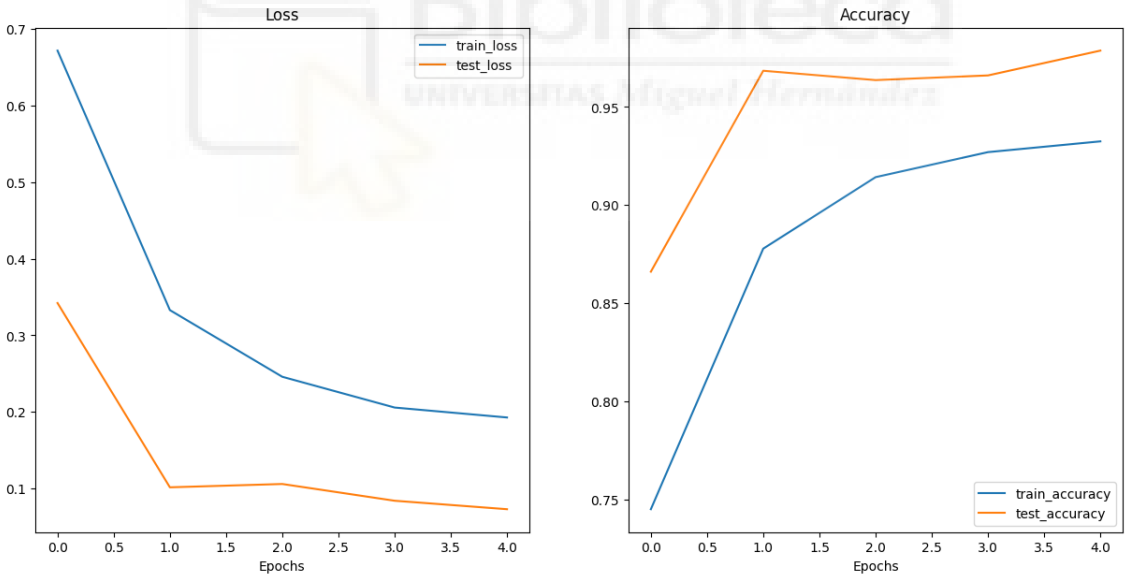


Ilustración 83: Gráficas del modelo reutilizado

Al examinar las gráficas de loss y accuracy, se evidencia que los datos de validación presentan un desempeño superior en comparación con los datos de entrenamiento. Esto indica que el modelo generaliza bien en datos no vistos previamente. Sin embargo, es importante destacar que, a partir del epoch 2, no se observaron mejoras significativas en los resultados. Esto sugiere que el modelo alcanzó su capacidad máxima de aprendizaje en este nuevo conjunto de datos y no pudo mejorar aún más su rendimiento.

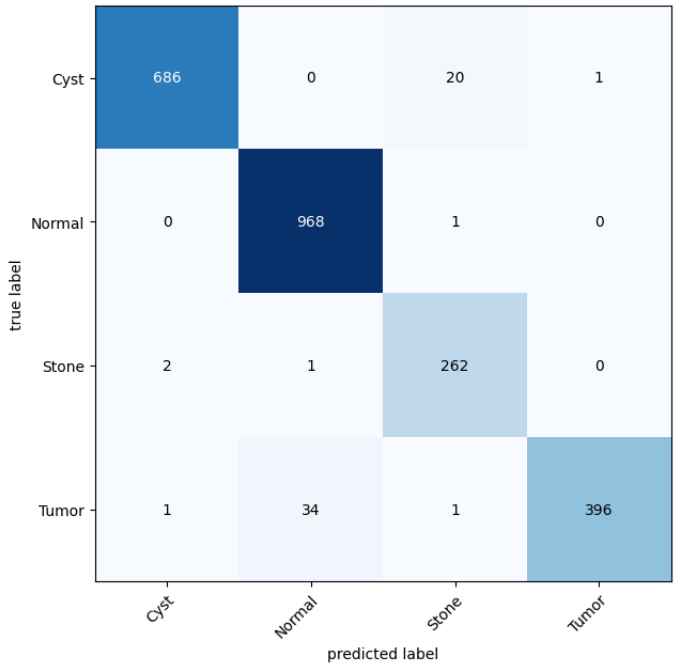


Ilustración 84: Matriz de confusión del modelo reutilizado

```
print(f"Accuracy: {acc_value:.4f}")
print(f"Precision: {precision_value:.4f}")
print(f"Recall: {recall_value:.4f}")
print(f"F1 Score: {f1_value:.4f}")

Accuracy: 0.9687
Precision: 0.9702
Recall: 0.9687
F1 Score: 0.9743
```

Ilustración 85: Métricas del modelo reutilizado

La matriz de confusión revela que, en la mayoría de los casos, las predicciones del modelo fueron correctas para todas las clases del dataset de riñón. Esto implica que el modelo fue capaz de captar patrones significativos en las imágenes de riñón y emplearlos de manera efectiva para realizar clasificaciones precisas.

Los resultados obtenidos al utilizar este modelo en el nuevo dataset son notables, alcanzando una exactitud de 96,87%. Este rendimiento supera significativamente la exactitud del 75,39% lograda en el conjunto de datos de tumores cerebrales, con el cual se entrenó el modelo inicialmente.

La discrepancia en los resultados puede atribuirse a diversas razones. En primer lugar, las imágenes de riñón pueden poseer características más distintivas y fáciles de clasificar en comparación con las imágenes de tumores cerebrales. Además, el dataset de riñón puede presentar una distribución de clases más equilibrada, lo que favorece un aprendizaje más efectivo por parte del modelo. Asimismo, es posible que la arquitectura y las características del modelo se adapten mejor a los patrones presentes en los datos del dataset de riñón.

5.2.8. Análisis de resultados y discusión

Entrenamiento	Accuracy	Precision	Recall	F1-Score	Tiempo (sg)
1	0,6878	0,7717	0,6878	0,6980	1602,606
2	0,7186	0,7505	0,7186	0,7157	1025,463
3	0,7232	0,8102	0,7232	0,7335	1110,822
4	0,7539	0,8119	0,7539	0,7640	1816,770
VGG16	0,7664	0,8454	0,7664	0,7665	174,666
EfficientNet	0,6517	0,7673	0,6517	0,6650	253,082

Tabla 6: Métricas de la evaluación de los modelos

El análisis de los resultados obtenidos revela mejoras progresivas en la exactitud de la clasificación a medida que se realizan los diferentes entrenamientos. Se observa un aumento gradual en el accuracy, la precisión, el recall y el F1-Score a lo largo de los entrenamientos.

El primer entrenamiento muestra una exactitud moderada, seguida de mejoras en el segundo y tercer entrenamiento. Sin embargo, es en el cuarto entrenamiento donde se alcanza el mayor nivel de exactitud, con valores superiores en todas las métricas evaluadas.

Las métricas de precision, recall y F1-Score también mostraron mejoras en los diferentes entrenamientos. Estas son importantes para evaluar la capacidad del modelo de predecir correctamente las clases positivas y negativas, así como el equilibrio entre precisión y exhaustividad. A medida que avanzaban los entrenamientos, se observó un aumento en la precisión y el recall, lo que implica una mejora en la calidad de las predicciones del modelo.

La incorporación de los modelos pre-entrenados VGG16 y EfficientNet en el análisis también resultó relevante. El modelo VGG16 demostró un buen desempeño, con altos valores de accuracy y métricas consistentes, lo que evidencia su capacidad para aprender patrones relevantes en los datos y realizar clasificaciones precisas. Por otro lado, el modelo EfficientNet obtuvo un rendimiento levemente inferior, con valores más bajos en las métricas de precision, recall y F1-Score en comparación con los otros modelos utilizados.

Es necesario tener en cuenta que el rendimiento obtenido en este proyecto pudo estar limitado por la disponibilidad limitada de datos de entrenamiento. La clasificación precisa de tumores cerebrales es una tarea compleja que requiere una cantidad considerable de datos de alta calidad y diversidad para capturar todas las variaciones y patrones relevantes. En este caso, la cantidad o calidad de los datos de entrenamiento podría haber sido insuficiente para alcanzar un rendimiento óptimo.

5.3. Desarrollo de Aplicación Web

Para el desarrollo de la aplicación web destinada a la detección de tumores cerebrales, se utilizó una combinación de las plataformas Anvil y Google Colab. Esta aplicación permite a los usuarios cargar una imagen de resonancia magnética del cerebro y obtener predicciones sobre la presencia y clasificación del tumor.

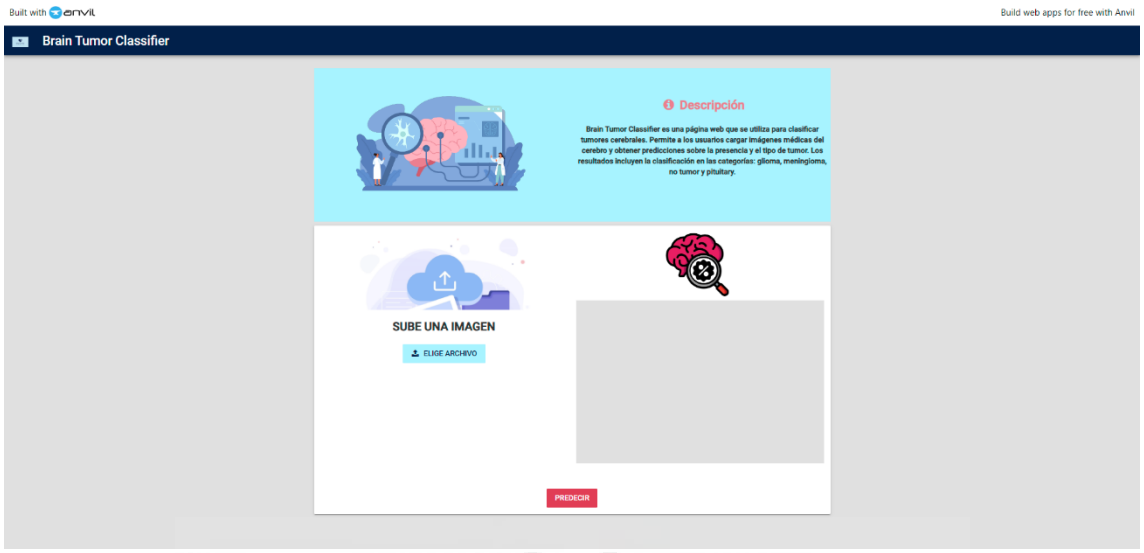


Ilustración 86: Inicio de la aplicación web

5.3.1. Conexión entre Google Colab y Anvil

La conexión entre Google Colab y Anvil se establece para permitir la comunicación entre la aplicación web en Anvil y la notebook en Google Colab. Esta conexión bidireccional se logra mediante la implementación de funciones tanto en el lado de Anvil como en el lado de Google Colab.

Google Colab

En el lado de Google Colab, se utiliza la función anvil.server.connect("your-uplink-key"), donde "your-uplink-key" es la clave de enlace específica de la aplicación. Esta función permite enviar y recibir datos entre la notebook y la aplicación.

Además, se crea una función de predicción que se encarga de clasificar la imagen cargada por el usuario en la página de Anvil. Esta función utiliza el modelo de red convolucional entrenado para realizar la predicción y devuelve los resultados a la aplicación web a través de la conexión establecida.

```python
@anvil.server.callable
def predict_brain_tumor(img):

  with anvil.media.TempFile(img) as file_path:
    image = cv2.imread(file_path)

  target_image = PIL.Image.fromarray(image)

  # Create transform pipleine to resize image
  custom_image_transform = transforms.Compose([ transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])])

  # Transform target image
  custom_image_transformed = custom_image_transform(target_image)

  loaded_model.eval()
  with torch.inference_mode():
      # Add an extra dimension to the image
      target_image = custom_image_transformed.unsqueeze(dim=0)

      # Make a prediction on image with an extra dimension and send it to the target device
      target_image_pred = loaded_model(target_image.to(device))

      # Convert logits -> prediction probabilities (using torch.softmax() for multi-class classification)
      target_image_pred_probs = torch.softmax(target_image_pred, dim=1)

      # Convert prediction probabilities -> prediction labels
      target_image_pred_label = torch.argmax(target_image_pred_probs, dim=1)

      image_pred_class = class_names[target_image_pred_label]
      prob_pred_class = target_image_pred_probs.max()
      prob_pred_class = prob_pred_class.item()

  return image_pred_class, prob_pred_class
```

Ilustración 87: Función de predicción en Google Colab

Anvil

En el lado de Anvil, se crean 2 funciones principales:

1. La primera función se encarga de cargar la imagen seleccionada por el usuario en la página de Anvil y guardar la fuente de dicha imagen en una variable para su posterior envío a Google Colab.

```python
def elegir_file_change(self, file, **event_args):
  """This method is called when a new file is loaded into this FileLoader"""
  self.img_tumor.source = self.elegir_file.file
  pass
```

Ilustración 88: Función de visualizar foto cargada en Anvil

2. La segunda función se activa al hacer clic en el botón de “Predecir” en la interfaz de usuario de Anvil.

   Esta función llama a la función de predicción en Google Colab, pasando la imagen cargada por el usuario. A continuación, se reciben los resultados de la predicción, que incluyen la clase correspondiente al tumor y su probabilidad asociada. Estos resultados se visualizan en la interfaz de usuario de Anvil.

```python
def btn_predecir_click(self, **event_args):
  """This method is called when the button is clicked"""
  # Call the Google Colab function and pass it the image file
  tumor_categoria, tumor_probabilidad = anvil.server.call('predict_brain_tumor', self.elegir_file.file)

  # If a category is returned set our tumor class
  if tumor_categoria and tumor_probabilidad:
    self.clase_tumor.visible = True
    self.probabilidad.visible = True
    self.img_searching.visible = False

    # Set tumor class
    self.clase_tumor.text = tumor_categoria

    # Set tumor probability with formatting
    #self.probabilidad.text = f"Probabilidad: {tumor_probabilidad:.4f}"
    self.probabilidad.text = f"Probabilidad: {tumor_probabilidad*100:.2f}"
```

Ilustración 89: Función del botón "Predecir" en Anvil

Para asegurar que la conexión entre Anvil y Google Colab se mantenga activa y que la aplicación web esté siempre en funcionamiento, se utiliza la función anvil.server.wait_forever() en Google Colab. Esta función permite que la aplicación web siga ejecutándose en Anvil, esperando la interacción de los usuarios y respondiendo a las solicitudes de predicción.

Es importante destacar que si la función anvil.server.wait_forever() no está en ejecución, la página web dejará de funcionar correctamente y los usuarios no podrán interactuar con la aplicación ni realizar predicciones.

5.3.2. Flujo de la Aplicación Web

El flujo de la aplicación comienza con el usuario subiendo la foto a través de la interfaz de usuario proporcionada por Anvil. Una vez cargada la imagen, esta se visualiza en la pantalla para que el usuario pueda verificar que se haya cargado correctamente.

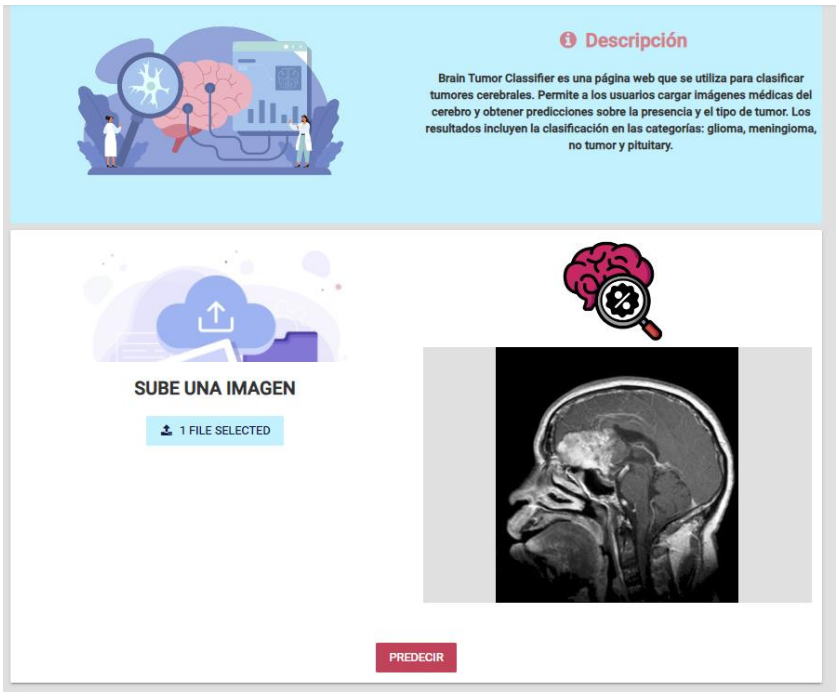


Ilustración 90: Visualizar imagen cargada en la aplicación web

Luego, el usuario puede hacer clic en el botón "Predecir" para que la aplicación realice la clasificación del tumor en la imagen. Para llevar a cabo esta tarea, la aplicación se comunica con el backend de Google Colab, donde se encuentra el modelo de aprendizaje profundo entrenado.

Una vez que se realiza la predicción, los resultados se devuelven a la aplicación web y se muestran al usuario. Esto incluye la clasificación del tumor y la probabilidad asociada a esa clasificación. Por ejemplo, la aplicación puede mostrar que se detectó un glioma con una probabilidad del 85%.

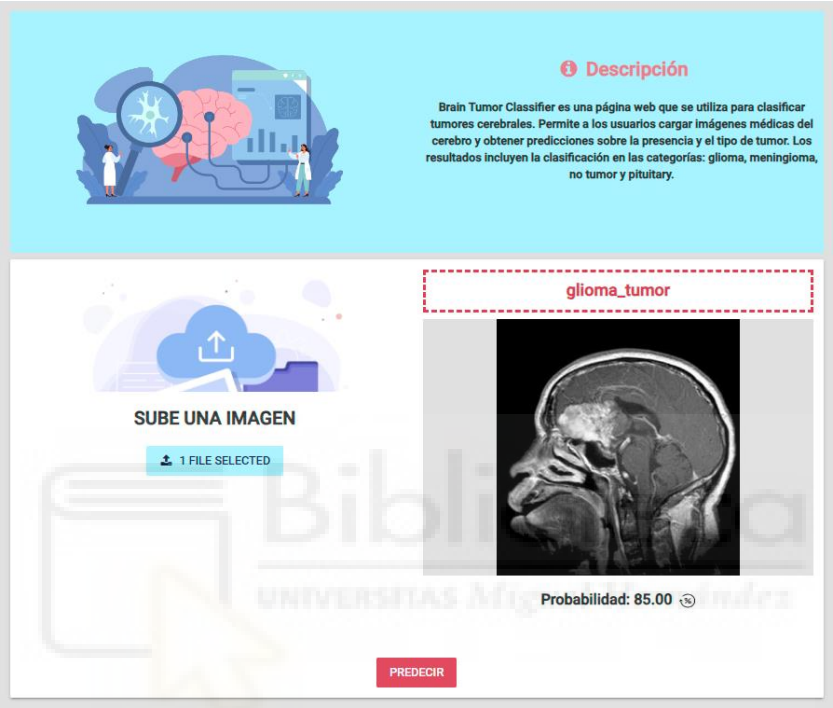


Ilustración 91: Predicción en la aplicación web

La interfaz de usuario de la aplicación también proporciona una experiencia amigable al mostrar la imagen original junto con la clasificación y la probabilidad en un formato fácil de entender. Además, se implementó un botón de predecir para permitir que el usuario realice múltiples predicciones con diferentes imágenes.

CAPÍTULO 6: CONCLUSIONES Y DESARROLLOS FUTUROS

Biblioteca
UNIVERSITAS Miguel Hernández

6.1. Conclusiones

Este proyecto ha sido una experiencia invaluable, brindando la oportunidad de adquirir conocimientos fundamentales en el campo de la inteligencia artificial y las redes neuronales convolucionales. A nivel personal, ha sido un proceso de crecimiento significativo en términos prácticos y técnicos, permitiendo el desarrollo de habilidades de resolución de problemas y adaptabilidad.

Durante el desarrollo de este proyecto, se ha tenido la oportunidad de aprender y aplicar diversas tecnologías y herramientas. En primer lugar, se ha adquirido un sólido dominio del lenguaje de programación Python, el cual se ha revelado como una herramienta poderosa y versátil para implementar algoritmos de aprendizaje automático y procesamiento de imágenes. Además, se ha trabajado con la biblioteca PyTorch, que ofrece una amplia gama de funcionalidades para la construcción y entrenamiento de redes neuronales convolucionales.

No obstante, durante la implementación práctica del trabajo, surgieron obstáculos que plantearon desafíos adicionales. En un principio, el dataset utilizado presentaba inconvenientes que impactaban la calidad de las predicciones obtenidas. Durante el proceso de entrenamiento del modelo de red convolucional, se lograron resultados alentadores, exhibiendo una precisión cercana al 90%. Sin embargo, al evaluar el modelo con imágenes del conjunto de datos de evaluación, se observó que las predicciones no eran correctas, incluso para imágenes del conjunto de datos de entrenamiento.

Ante esta situación, se planteó la posibilidad de que el problema residiera en el dataset en sí. Para investigar esta hipótesis, se decidió probar con otro dataset, en este caso el " Brain Tumor Classification (MRI)". Esta decisión implicó un reinicio en la etapa de preprocesamiento de datos y ajuste de parámetros del modelo, pero resultó ser crucial para determinar si el problema radicaba en el modelo planteado o en la calidad de los datos.

Después de realizar las pruebas con el nuevo dataset, se obtuvieron resultados más alentadores. El modelo de red convolucional mostró una mejora significativa en las predicciones, lo que indicaba que el modelo en sí no era el problema, sino la calidad y la idoneidad del dataset utilizado inicialmente.

Además, se encontraron dificultades en el uso de Google Colab, debido al tiempo que tardaba cada ejecución y a las limitaciones en el uso de GPU, lo que resultó en un tiempo de desarrollo mayor al estimado y la imposibilidad de hacer más pruebas de las deseadas.

A pesar de estos inconvenientes, se logró superar los desafíos y completar con éxito el desarrollo del modelo de red convolucional y la creación de la aplicación web con Anvil. El aprendizaje de Anvil ha sido una experiencia valiosa que ha permitido adquirir habilidades adicionales en el desarrollo de aplicaciones web y simplificado la implementación de una interfaz al código desarrollado en Google Colab.

6.2. Futuros trabajos

En futuros trabajos, se recomienda poner un mayor énfasis en la recopilación exhaustiva de datos, asegurando que sean representativos de la diversidad de casos clínicos y tipos de tumores cerebrales. Esto ayudará a mejorar la capacidad de generalización del modelo y a obtener predicciones más precisas en situaciones del mundo real.

Es recomendable explorar diversas técnicas para mejorar el rendimiento y la precisión del modelo de clasificación de tumores cerebrales. Además de la optimización de hiperparámetros y el uso de técnicas de Transfer Learning con otros modelos pre-entrenados distintos a VGG16 y EfficientNet, se pueden considerar otras estrategias.

Por ejemplo, el empleo de Ensemble Learning, que consiste en combinar múltiples modelos, puede contribuir a mejorar la calidad y robustez de las predicciones. También es válido explorar técnicas avanzadas de Data Augmentation, las cuales permiten generar variaciones y aumentar la diversidad del conjunto de datos de entrenamiento, lo que puede llevar a una mejora en la capacidad de generalización del modelo.

Asimismo, el aprendizaje semi-supervisado es una opción interesante para aprovechar datos no etiquetados y mejorar la precisión del modelo. Esta técnica combina datos etiquetados y no etiquetados en el proceso de entrenamiento, lo que puede ayudar a descubrir patrones ocultos y mejorar la capacidad de predicción.

Por último, se puede investigar el uso de redes neuronales generativas para generar imágenes sintéticas que se añadan al conjunto de datos de entrenamiento. Esto ampliaría la diversidad y representatividad de los datos, lo que podría mejorar el desempeño y la capacidad de clasificación del modelo.

Estas técnicas adicionales ofrecen oportunidades prometedoras para mejorar tanto el rendimiento como la precisión de los modelos de clasificación de tumores cerebrales, y son aspectos interesantes que considerar en futuros trabajos de investigación.

CAPÍTULO 7: BIBLIOGRAFÍA

Biblioteca
UNIVERSITAS Miguel Hernández

[1] Stuart J. Russell y Peter Norvig, *Inteligencia artificial : un enfoque moderno*. Pearson Prentice Hall, 2004.
[2] Raúl E. López Briega, «Deep Learning - Libro online de IAAR», 2017. https://iaarbook.github.io/deeplearning/ (accedido 1 de marzo de 2023).
[3] C. Alberto Ruiz Marta Susana Basualdo Autor y D. Jorge Matich, «Cátedra: Informática Aplicada a la Ingeniería de Procesos-Orientación I Redes Neuronales: Conceptos Básicos y Aplicaciones», 2001.
[4] Eduardo Francisco Caicedo Bravo y Jesús Alfonso López Sotelo, «Una aproximación práctica a las redes neuronales artificiales», 2009.
[5] Na8, «Breve Historia de las Redes Neuronales Artificiales | Aprende Machine Learning», 2018. https://www.aprendemachinelearning.com/breve-historia-de-las-redes-neuronales-artificiales/ (accedido 20 de febrero de 2023).
[6] Panagiotis Antoniadis, «Activation Functions: Sigmoid vs Tanh | Baeldung on Computer Science», 2022. https://www.baeldung.com/cs/sigmoid-vs-tanh-functions (accedido 16 de marzo de 2023).
[7] P. Ramachandran, B. Zoph, y Q. V Le Google Brain, «Searching for Activation Functions», *6th International Conference on Learning Representations, ICLR 2018 - Workshop Track Proceedings*, oct. 2017, Accedido: 27 de junio de 2023. [En línea]. Disponible en: https://arxiv.org/abs/1710.05941v2
[8] S. Elfwing, E. Uchibe, y K. Doya, «Sigmoid-weighted linear units for neural network function approximation in reinforcement learning», *Neural Networks*, vol. 107, pp. 3-11, 2018, doi: 10.1016/j.neunet.2017.12.012.
[9] D. Misra, «The Swish Activation Function», 2021. https://blog.paperspace.com/swish-activation-function/ (accedido 25 de abril de 2023).
[10] W. Rodrigues, «SineReLU — An Alternative to the ReLU Activation Function», 4 de mayo de 2018. https://wilder-rodrigues.medium.com/sinerelu-an-alternative-to-the-relu-activation-function-e46a6199997d (accedido 25 de abril de 2023).
[11] FUTURISM, «How Do Artificial Neural Networks Learn?», 2015. https://futurism.com/how-do-artificial-neural-networks-learn (accedido 20 de febrero de 2023).
[12] Jordi TORRES.AI, «Learning Process of a Deep Neural Network», 2020. https://towardsdatascience.com/learning-process-of-a-deep-neural-network-5a9768d7a651 (accedido 20 de febrero de 2023).
[13] Rubén Rodríguez Abril, «El descenso del gradiente • Un artículo de La Máquina Oráculo», 2020. https://lamaquinaoraculo.com/computacion/el-descenso-del-gradiente/ (accedido 23 de febrero de 2023).
[14] Carolina Bento, «Multilayer Perceptron Explained with a Real-Life Example and Python Code: Sentiment Analysis», 2021. https://towardsdatascience.com/multilayer-perceptron-explained-with-a-real-life-example-and-python-code-sentiment-analysis-cb408ee93141 (accedido 27 de febrero de 2023).
[15] A. Sharif Ahmadian, «Numerical Modeling and Simulation», *Numerical Models for Submerged Breakwaters*, 2016, Accedido: 15 de marzo de 2023. [En línea]. Disponible en: https://www.sciencedirect.com/topics/engineering/radial-basis-function-network
[16] H. Faris, I. Aljarah, y S. Mirjalili, «Evolving Radial Basis Function Networks Using Moth-Flame Optimizer», *Handbook of Neural Computation*, ene. 2017, Accedido: 17 de marzo de 2023. [En línea]. Disponible en: https://www.sciencedirect.com/topics/engineering/radial-basis-function-network

[17] Javaid Nabi, «Recurrent Neural Networks (RNNs)», 2019. https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85 (accedido 27 de febrero de 2023).
[18] S. Hiriyannaiah, A. M. D. Srinivas, G. K. Shetty, S. G.M., y K. G. Srinivasa, «A computationally intelligent agent for detecting fake news using generative adversarial networks», *Hybrid Computational Intelligence*, pp. 69-96, ene. 2020, doi: 10.1016/B978-0-12-818699-2.00004-4.
[19] A. Vaswani *et al.*, «Attention Is All You Need», *Adv Neural Inf Process Syst*, vol. 2017-December, pp. 5999-6009, jun. 2017, Accedido: 4 de mayo de 2023. [En línea]. Disponible en: https://arxiv.org/abs/1706.03762v5
[20] GISGeography, «Image Classification Techniques in Remote Sensing», 2022. https://gisgeography.com/image-classification-techniques-remote-sensing/ (accedido 1 de marzo de 2023).
[21] Analytics Vidhya, «Convolutional Neural Networks (CNN) from Scratch», 2020. https://courses.analyticsvidhya.com/courses/take/convolutional-neural-networks-cnn-from-scratch/texts/10844923-what-is-a-neural-network (accedido 15 de septiembre de 2022).
[22] Orhan G. Yalçın, «The Brief History of Convolutional Neural Networks», 2021. https://towardsdatascience.com/the-brief-history-of-convolutional-neural-networks-45afa1046f7f (accedido 3 de marzo de 2023).
[23] Rachel Draelos, «The History of Convolutional Neural Networks», 2019. https://glassboxmedicine.com/2019/04/13/a-short-history-of-convolutional-neural-networks/ (accedido 5 de marzo de 2023).
[24] Redacción KeepCoding, «Arquitectura típica de una red neuronal convolucional», 6 de enero de 2023. https://keepcoding.io/blog/arquitectura-tipica-red-neuronal-convolucional/ (accedido 6 de marzo de 2023).
[25] A. Ghosh, A. Sufian, F. Sultana, A. Chakrabarti, y D. De, «Fundamental concepts of convolutional neural network», en *Intelligent Systems Reference Library*, Springer, 2019.
[26] A. Ajit, K. Acharya, y A. Samanta, *A Review of Convolutional Neural Networks*. 2020. Accedido: 5 de marzo de 2023. [En línea]. Disponible en: https://www.pyimagesearch.com/2018/11/12/yolo-object-
[27] J. Brownlee, «A Gentle Introduction to Pooling Layers for Convolutional Neural Networks», 2019. https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (accedido 9 de marzo de 2023).
[28] D. Yu, H. Wang, P. Chen, y Z. Wei, «Mixed pooling for convolutional neural networks», en *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, Springer Verlag, 2014.
[29] Z. Li, F. Liu, W. Yang, S. Peng, y J. Zhou, «A Survey of Convolutional Neural Networks: Analysis, Applications, and Prospects», *IEEE Trans Neural Netw Learn Syst*, vol. 33, n.º 12, dic. 2022, Accedido: 20 de marzo de 2023. [En línea]. Disponible en: https://ieeexplore.ieee.org/document/9451544
[30] S. Sakib, A. Jawad, A. Jawad Kabir, y H. Ahmed, «An Overview of Convolutional Neural Network: Its Architecture and Applications», 2019, doi: 10.20944/PREPRINTS201811.0546.V4.
[31] B. Lutkevich, «What is Speech Recognition?», 2021. https://www.techtarget.com/searchcustomerexperience/definition/speech-recognition (accedido 14 de marzo de 2023).

[32] P. Sharma, «Applications of Convolutional Neural Networks (CNN)», 2021. https://www.analyticsvidhya.com/blog/2021/10/applications-of-convolutional-neural-networkscnn/ (accedido 14 de marzo de 2023).
[33] Z. Rguibi, A. Hajami, D. Zitouni, A. Elqaraoui, y A. Bedraoui, «CXAI: Explaining Convolutional Neural Networks for Medical Imaging Diagnostic», *Electronics 2022, Vol. 11, Page 1775*, vol. 11, jun. 2022, doi: 10.3390/ELECTRONICS11111775.
[34] «TensorFlow». https://www.tensorflow.org/?hl=es-419 (accedido 29 de mayo de 2023).
[35] «Keras: Deep Learning for humans». https://keras.io/ (accedido 29 de mayo de 2023).
[36] «PyTorch». https://pytorch.org/ (accedido 29 de mayo de 2023).
[37] T. School, «Características principales de Python: ¿cuáles son?», 7 de marzo de 2022. https://www.tokioschool.com/noticias/caracteristicas-principales-de-python/ (accedido 25 de abril de 2023).
[38] NumPy Developers, «What is NumPy? — NumPy v1.24 Manual», 14 de noviembre de 2009. https://numpy.org/doc/stable/user/whatisnumpy.html (accedido 1 de mayo de 2023).
[39] A. K. Y. Yim, C. Y. L. Chung, y A. C. S. Yu, *Matplotlib for Python developers : effective techniques for data visualization with Python*. Packt Publishing Ltd, 2018. Accedido: 26 de abril de 2023. [En línea]. Disponible en: https://books.google.es/books?hl=es&lr=&id=G99YDwAAQBAJ&oi=fnd&pg=PP1&dq=matplotlib+python&ots=tyd2ox7PZg&sig=piSd5TqfoXImSiWLF4keJkQSZG8#v=onepage&q=matplotlib%20python&f=false
[40] W. McKinney, «pandas: powerful Python data analysis toolkit», 2012, Accedido: 1 de mayo de 2023. [En línea]. Disponible en: http://pandas.pydata.org/pandas-docs/version/0.7.3/pandas.pdf
[41] E. Stevens, L. Antiga, y T. Viehmann, *Deep Learning with PyTorch*. Manning Publications Co., 2020. Accedido: 26 de abril de 2023. [En línea]. Disponible en: https://isip.piconepress.com/courses/temple/ece_4822/resources/books/Deep-Learning-with-PyTorch.pdf
[42] Torch Contributors, «torchvision — Torchvision main documentation», 2017. https://pytorch.org/vision/stable/index.html (accedido 3 de mayo de 2023).
[43] Lightning AI, «TorchMetrics — PyTorch Lightning 2.0.2 documentation», 2021. https://lightning.ai/docs/pytorch/stable/ecosystem/metrics.html (accedido 2 de mayo de 2023).
[44] «torchinfo · PyPI», 14 de mayo de 2023. https://pypi.org/project/torchinfo/ (accedido 22 de junio de 2023).
[45] S. Raschka, «MLxtend: Providing machine learning and data science utilities and extensions to Python's scientific computing stack», 2018, doi: 10.21105/joss.00638.
[46] K. Reitz, «Requests: HTTP for Humans™ — Requests 2.29.0 documentation», 2016. https://docs.python-requests.org/en/latest/ (accedido 3 de mayo de 2023).
[47] Python Software Foundation, «timeit — documentación de Python - 3.11.3», 2023. https://docs.python.org/es/3/library/timeit.html (accedido 3 de mayo de 2023).
[48] Md Younus Ahamed, «split_folders for train test val split of images», marzo de 2023. https://www.kaggle.com/code/shuvostp/split-folders-for-train-test-val-split-of-images (accedido 22 de junio de 2023).

[49] P. Kanani y M. Padole, «Deep Learning to Detect Skin Cancer using Google Colab», vol. 8, n.º 6, ago. 2019, doi: 10.35940/ijeat.F8587.088619.
[50] Product Hunt, «Anvil - Product Information, Latest Updates, and Reviews 2023», 2023. https://www.producthunt.com/products/anvil-2 (accedido 5 de mayo de 2023).
[51] M. N. M. Sehmi, M. F. A. Fauzi, W. S. H. M. W. Ahmad, y E. W. L. Chan, «PancreaSys: An Automated Cloud-Based Pancreatic Cancer Grading System», *Frontiers in Signal Processing*, vol. 2, 2022, doi: 10.3389/frsip.2022.833640.
[52] APM, «What is project management?», 2011. https://www.apm.org.uk/resources/what-is-project-management/ (accedido 29 de mayo de 2023).
[53] J. Martins, «Diagrama de Gantt: qué es y cómo crear uno con ejemplos», 12 de septiembre de 2022. https://asana.com/es/resources/gantt-chart-basics (accedido 30 de mayo de 2023).
[54] Sartaj, «Brain Tumor Classification (MRI)», 2020. https://www.kaggle.com/datasets/sartajbhuvaji/brain-tumor-classification-mri (accedido 20 de junio de 2023).
[55] Y. Ma y Y. Luo, «Bone fracture detection through the two-stage system of Crack-Sensitive Convolutional Neural Network», *Inform Med Unlocked*, vol. 22, p. 100452, ene. 2021, doi: 10.1016/J.IMU.2020.100452.
[56] C. Srinivas *et al.*, «Deep Transfer Learning Approaches in Performance Analysis of Brain Tumor Classification Using MRI Images», *J Healthc Eng*, vol. 2022, 2022, doi: 10.1155/2022/3264367.
[57] S. Varsheni R, «10 PyTorch Transformations for Data Scientists», 22 de abril de 2021. https://www.analyticsvidhya.com/blog/2021/04/10-pytorch-transformations-you-need-to-know/ (accedido 31 de mayo de 2023).
[58] S. Ray, «What Is Data Augmentation?», 27 de noviembre de 2021. https://medium.com/lansaar/what-is-data-augmentation-3da1373e3fa1 (accedido 31 de mayo de 2023).
[59] GeeksforGeeks, «Python PyTorch – RandomHorizontalFlip() Function», 10 de junio de 2022. https://www.geeksforgeeks.org/python-pytorch-randomhorizontalflip-function/ (accedido 8 de junio de 2023).
[60] J. Solawetz, «Why and How to Implement Random Rotate Data Augmentation», 24 de junio de 2020. https://blog.roboflow.com/why-and-how-to-implement-random-rotate-data-augmentation/ (accedido 8 de junio de 2023).
[61] Papers With Code, «Random Resized Crop Explained». https://paperswithcode.com/method/randomresizedcrop (accedido 20 de junio de 2023).
[62] PyTorch, «RandomApply — Torchvision main documentation». https://pytorch.org/vision/main/generated/torchvision.transforms.RandomApply.html (accedido 20 de junio de 2023).
[63] T. P., «Overfitting and underfitting in machine learning», 17 de octubre de 2022. https://www.superannotate.com/blog/overfitting-and-underfitting-in-machine-learning (accedido 1 de junio de 2023).
[64] N. Donges, «What Is Transfer Learning? A Guide for Deep Learning», 12 de septiembre de 2022. https://builtin.com/data-science/transfer-learning (accedido 1 de junio de 2023).
[65] K. You, M. Long, J. Wang, y M. I. Jordan, «How Does Learning Rate Decay Help Modern Neural Networks?», ago. 2019, Accedido: 19 de junio de 2023. [En línea]. Disponible en: https://arxiv.org/abs/1908.01878v2

[66] G. Li, X. Jian, Z. Wen, y J. Alsultan, «Algorithm of overfitting avoidance in CNN based on maximum pooled and weight decay», *Applied Mathematics and Nonlinear Sciences*, vol. 7, n.º 2, 2022, doi: 10.2478/amns.2022.1.00011.
[67] Tokio School, «¿Qué es y cómo funciona el transfer learning?», 5 de julio de 2022. https://www.tokioschool.com/noticias/transfer-learning/ (accedido 22 de junio de 2023).
[68] Md Nazmul Islam y Md Humaion Kabir Mehedi, «CT KIDNEY DATASET: Normal-Cyst-Tumor and Stone», 2021. https://www.kaggle.com/datasets/nazmul0087/ct-kidney-dataset-normal-cyst-tumor-and-stone (accedido 22 de junio de 2023).

Biblioteca
UNIVERSITAS Miguel Hernández


UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE
ESCUELA POLITÉCNICA SUPERIOR DE ELCHE
GRADO EN INGENIERÍA INFORMÁTICA EN TECNOLOGÍAS DE LA INFORMACIÓN
UNIVERSITAS Miguel Hernández
TRABAJO FIN DE GRADO
JUNIO - 2023
"NORMAS DE ESTILO DEL TRABAJO FIN DE GRADO"
AUTOR: BEGOÑA GÓMEZ PUJANTE
DIRECTOR: ANTONIO PEÑALVER BENAVENT