Universidad de La Laguna

Máster Universitario en Ciberseguridad e Inteligencia de Datos

Estudio y desarrollo de modelos de aprendizaje profundo aplicados al diagnóstico de enfermedades oculares

Study and development of deep learning models applied to the diagnosis of eye diseases.

Julio Daniel Pérez Cabrera

Tutores:

José Marcos Moreno Vega

Carlos Pérez González

En La Laguna, a fecha de 05/07/2023.

Estudio y desarrollo de modelos de aprendizaje profundo aplicados al diagnóstico de enfermedades oculares

D. José Marcos Moreno Vega, con N.I.F. 42.841.047-M Catedrático de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. Carlos Javier Pérez González, con N.I.F. 45.452.719-G Profesor titular de Universidad adscrito al Departamento de Matemáticas, Estadística e Investigación Operativa de la Universidad de La Laguna, como cotutor.

C E R T I F I C A (N)

Que la presente memoria titulada:

Estudio y desarrollo de modelos de aprendizaje profundo aplicados al diagnóstico de enfermedades oculares

ha sido realizada bajo su dirección por D. Julio Daniel Pérez Cabrera, con

N.I.F. 45.980.211.Z

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos

oportunos firman la presente en La Laguna a 05/07/2023

Estudio y desarrollo de modelos de aprendizaje profundo aplicados al diagnóstico de enfermedades oculares

Agradecimientos

Este proyecto ha sido un reto ya que suponía el adentrarme en un área que hasta este momento no había profundizado, el Deep Learning. Quiero agradecer a mis tutores Carlos y Marcos a empujarme a ir un paso más allá y trabajar con modelos más avanzados de inteligencia artificial y dar lo mejor de mí. También quiero agradecer a mis padres Cecilia y Julio, y a mi novia Tere, por estar apoyándome siempre en los proyectos y aventuras en los que me embarco, que no han sido pocos, siendo este mi tercer máster. A todos, muchas gracias.

Acknowledgments

This project has been a challenge because it meant going into an area that until now I had not delved into, Deep Learning. I want to thank my tutors Carlos and Marcos for pushing me to go a step further and work with more advanced models of artificial intelligence and give the best of me. I also want to thank my parents Cecilia and Julio, and my girlfriend Tere, for always supporting me in the projects and adventures in which I embark, which have not been few, being this my third master. To all of you, thank you very much.

Estudio y desarrollo de modelos de aprendizaje profundo aplicados al diagnóstico de enfermedades oculares

Resumen

En este proyecto se aborda el Deep Learning desde el punto de vista teórico y práctico. Empezamos hablando de las bases del Deep Learning y las partes que lo componen, sus arquitecturas más relevantes o utilizadas. Continuamos con la explicación de las herramientas más típicas para trabajar en aprendizaje profundo y finalmente llegamos al problema abordado, el cual se trata de la detección de enfermedades mediante el estudio de los ojos. En este punto, detallamos todos los pasos seguidos para conseguir el modelo posible que nos brinde el mejor rendimiento. Finalmente, comentaremos los problemas encontrados y los retos a los que nos podremos enfrentar en el futuro.

Abstract

This project deals with Deep Learning from a theoretical and practical point of view. We start by talking about the bases of Deep Learning and the parts that compose it, its most relevant or used architectures. We continue with the explanation of the most typical tools to work in deep learning and finally we come to the problem addressed, which is the detection of diseases through the study of the eyes. At this point, we detail all the steps followed to obtain the possible model that gives us the best performance. Finally, we will comment on the problems encountered and the challenges we may face in the future.

Índice

Índice de figuras

1. Introducción y conceptos básicos de Deep Learning

1.1. Objetivos y alcance del proyecto

El objetivo principal de este proyecto es explorar en profundidad el concepto de Deep Learning, partiendo de su historia y evolución, hasta llegar a sus aplicaciones actuales y potenciales. La meta es desarrollar un conocimiento sólido y comprensivo de los componentes y fundamentos matemáticos del Deep Learning. Adicionalmente, se busca examinar y entender las técnicas y pasos requeridos para aplicar con éxito esta poderosa herramienta en problemas reales.

El proyecto abordará una variedad de subtemas relacionados con el Deep Learning. Estos incluyen el estudio de la arquitectura de las redes neuronales profundas, el proceso de aprendizaje y optimización, y la forma en que estas técnicas se pueden utilizar para analizar y clasificar grandes volúmenes de datos. Una parte clave de este proyecto será el desarrollo de un modelo de Deep Learning, que deberá ser capaz de clasificar imágenes para determinar enfermedades.

En cuanto al alcance, este trabajo se limitará a lo que se ha descrito en el índice. No obstante, se buscará profundizar en cada uno de los temas tratados, utilizando ejemplos prácticos y actuales para ilustrar y reforzar los conceptos teóricos presentados. La aplicación final del modelo de Deep Learning desarrollado, si bien será en el ámbito médico, no pretende ser un sistema de diagnóstico definitivo, sino más bien una demostración de las capacidades de esta tecnología.

1.2. Redes neuronales artificiales

Las redes neuronales artificiales (ANNs) son sistemas que intentan replicar la estructura y funcionalidad de las redes neuronales biológicas. Están compuestas por nodos o "neuronas" interconectadas, permitiendo a las ANNs aprender y adaptarse mediante la experiencia, al igual que lo hacen los cerebros humanos.

Las ANNs se organizan en capas. La capa de entrada recibe los datos, las capas ocultas procesan la información, y la capa de salida proporciona la respuesta final. Cada neurona está vinculada con todas las neuronas de la capa siguiente, transmitiendo señales que son moduladas por pesos ajustados en el proceso de aprendizaje.

Un ejemplo de funcionamiento es la Red Neuronal Feedforward, donde la información fluye en una única dirección, sin bucles de retroalimentación. Las ANNs se utilizan en diversos campos, desde el reconocimiento de voz e imágenes hasta la predicción de mercados, gracias a su capacidad para emular el procesamiento de información del cerebro humano y resolver problemas complejos de manera intuitiva.

1.3. ¿Cómo se relacionan las neuronas biológicas con las redes neuronales?

Las redes neuronales artificiales, utilizadas en Deep Learning y otras áreas de Inteligencia Artificial, están inspiradas en la estructura y funcionamiento de las neuronas biológicas del

cerebro humano. Aunque las redes neuronales artificiales son simplificaciones de las complejas redes de neuronas del cerebro, presentan algunas similitudes fundamentales con el funcionamiento de las neuronas biológicas.

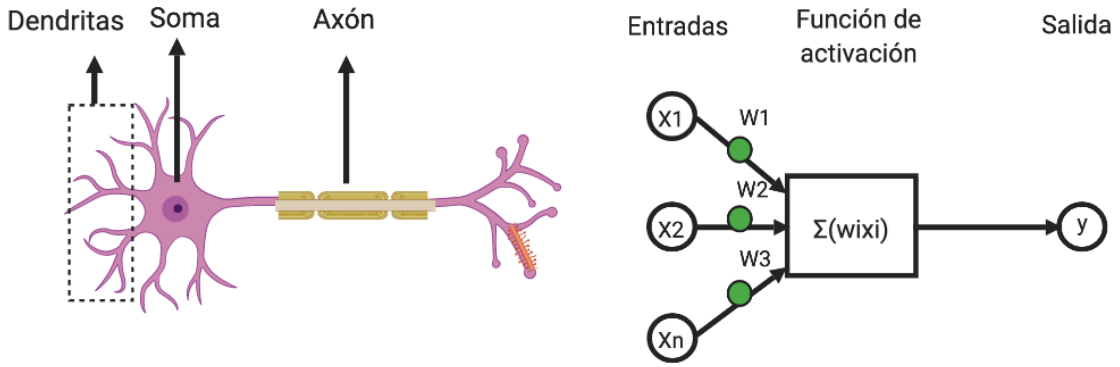


Figura 1 Comparación de neurona biológica con neurona aplicada en DL. Fuente: (García, 2019)

Cada neurona biológica tiene una estructura compuesta por un cuerpo celular, dendritas, y un axón. Las dendritas reciben señales de entrada de otras neuronas, que son integradas y procesadas en el cuerpo celular. Si el nivel de activación neuronal alcanza un umbral, la neurona dispara un impulso eléctrico a lo largo del axón, que se transmite a otras neuronas en la red. En las redes neuronales artificiales, cada neurona artificial se conecta con otras neuronas a través de conexiones que tienen pesos que se ajustan durante el entrenamiento. Cuando la entrada a una neurona artificial supera un cierto umbral, la neurona dispara una señal a las neuronas conectadas.

Otra similitud importante es que tanto las neuronas biológicas como las artificiales utilizan una función de activación para determinar si la neurona debe activarse o no en respuesta a una entrada. En el caso de las redes neuronales artificiales, esta función puede ser lineal o no lineal, y puede variar dependiendo del tipo de tarea que se esté resolviendo.

En definitiva, aunque las redes neuronales artificiales no son una reproducción exacta de las redes de neuronas del cerebro, comparten algunas similitudes fundamentales en su funcionamiento y estructura con las neuronas biológicas, lo que ha permitido el desarrollo de modelos computacionales que imitan algunas de las capacidades de procesamiento de información del cerebro humano.

1.4. ¿Qué es y cómo surge la idea del perceptrón?

El perceptrón es un modelo de red neuronal artificial que fue desarrollado por el psicólogo y matemático estadounidense Frank Rosenblatt en 1957. El perceptrón es una red neuronal simple, que consta de una o varias capas de neuronas artificiales, y se utiliza para la clasificación de patrones.

La idea detrás del perceptrón se basa en el funcionamiento de las neuronas biológicas en el cerebro. Las neuronas reciben señales eléctricas de otras neuronas y, si la señal supera un umbral, se activan y envían una señal eléctrica a otras neuronas. El perceptrón intenta imitar este proceso biológico.

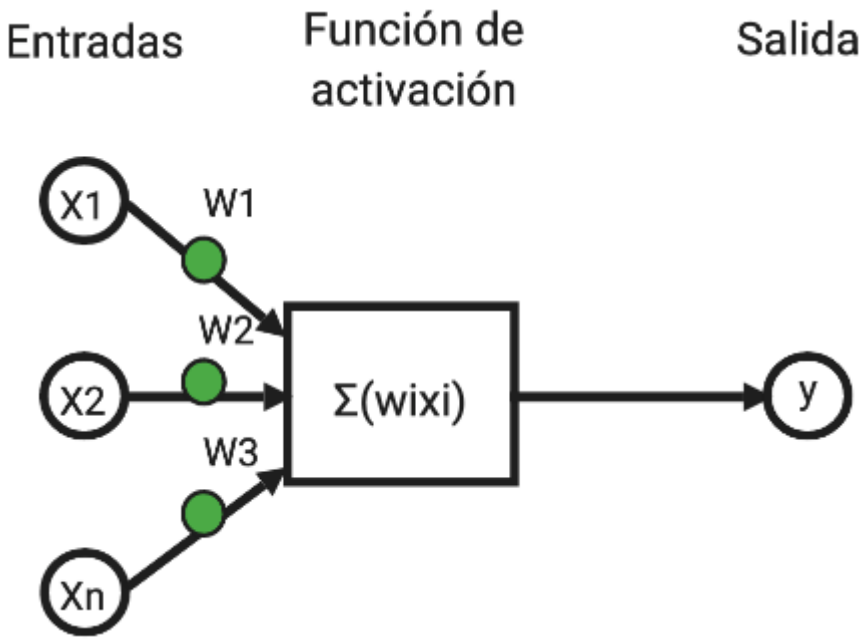


Figura 2 Estructura de neurona aplicada en Deep Learning. Fuente: (García, 2019)

Rosenblatt estaba interesado en el desarrollo de un algoritmo de aprendizaje automático que pudiera ser utilizado para la clasificación de patrones. Se inspiró en la teoría de la percepción de Gestalt, que sostiene que el cerebro organiza la información visual en patrones para interpretarla. El modelo de perceptrón consta de una capa de neuronas, donde cada neurona recibe una entrada y produce una salida binaria.

El perceptrón se entrena mediante el ajuste de los pesos de las conexiones entre las neuronas. En el entrenamiento, se presentan varios patrones de entrada, y se ajustan los pesos de las conexiones para que el perceptrón clasifique correctamente estos patrones. El proceso de entrenamiento es iterativo y se detiene cuando el perceptrón alcanza un nivel de precisión deseado.

El perceptrón fue uno de los primeros modelos de redes neuronales artificiales y sentó las bases para el desarrollo de modelos más complejos en el futuro, como las redes neuronales profundas. El perceptrón también ha sido utilizado en diversas aplicaciones, como el reconocimiento de caracteres manuscritos, la detección de spam en correos electrónicos, y la clasificación de imágenes.

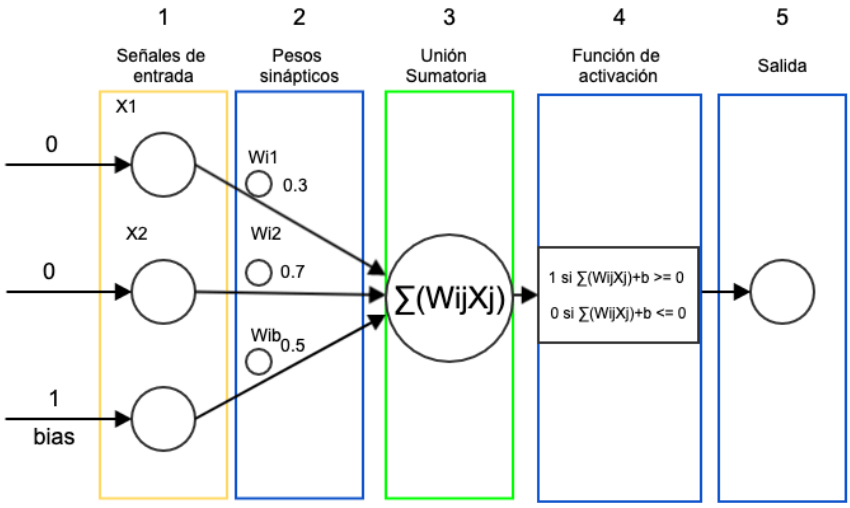


Figura 3 Perceptrón. Fuente: (García, 2019)

1.5. ¿Cómo funciona un perceptrón?

Para entender cómo funciona un perceptrón, a continuación, utilizaremos el ejemplo de la figura 3 realizaremos una ejemplificación, en la que le enseñaremos al perceptrón a responder a las entradas de una compuerta AND, en donde dados 2 valores de entrada input 1, input 2, nos dé un valor de salida esperado (output).

La compuerta lógica AND realiza un producto booleano dadas dos entradas, la condición que debe cumplir la compuerta AND es definida por la siguiente tabla de verdad.

Entrada A	Entrada B	Salida
0	0	0
0	1	0
1	0	0
1	1	1

Además de las entradas, en el contexto del perceptrón, se añade un tercer valor (Bias). El valor de Bias (sesgo) es un parámetro adicional que se utiliza para ajustar la salida del perceptrón. Se trata de un término independiente que se suma a la suma ponderada de las entradas multiplicadas por los pesos.

El Bias permite controlar el umbral de activación del perceptrón. Si el valor de Bias es alto, el perceptrón tendrá una mayor probabilidad de activarse, mientras que si el valor de Bias es bajo, el perceptrón será menos propenso a activarse. Además también se utiliza para desplazar la función de activación del perceptrón, lo que puede ser útil para que el modelo se ajuste mejor a los datos y pueda aprender relaciones más complejas.

El criterio de asignación de valor de Bias atiende a observar el impacto que este valor pueda tener en el rendimiento y convergencia del modelo, pero a priori no hay un valor preestablecido.

- Caso 1: Entrada A (0), Entrada B (0) → Salida 0

Se toman 2 entradas en este caso con valor 0 para ambas, y un valor de Bias de 0.5.

Se generan los pesos aleatorios W_{i1}, W_{i2} en el rango [-1,1], W_{i1}=0.3, W_{i2}=0.7 y el valor de Bias W_{ib}=0.5.

Se realiza la suma ponderada de las entradas por los pesos:

$$suma = \big((0 \cdot 0.3) + (0 \cdot 0.7)\big) + (1 \cdot 0.5) = 0.5$$

Función de activación:

1 si suma+bias >= 0

0 si suma+bias <= 0

Para nuestro caso obtuvimos una suma de 0.5 (de acuerdo con la condición de la función de activación) por lo que nuestra salida es 0.

Salida: Como obtuvimos una salida 0 y nuestra salida esperada es 0 esta condición se cumple, y por tanto, pasaríamos con los valores de las siguientes entradas.

- Caso 2: Entrada A (0), Entrada B (1) → Salida 0

Se toman 2 entradas con valor 0 y 1. Se generarán nuevos pesos aleatorios W_{i1}, W_{i2} en el rango [-1,1], W_{i1}=0.5, W_{i2}=0.1 y para el Bias W_{ib}=0.9

Se realiza la suma ponderada de las entradas por los pesos:

$$suma = \big((0 \cdot 0.5) + (1 \cdot 0.1)\big) + (1 \cdot 0.9) = 1$$

Atendiendo a la función de activación obtuvimos una suma de 1, por lo que nuestra salida es 1.

Salida: Obtuvimos una salida 1 y nuestra salida esperada es 0, por lo que esta condición no se cumple. Entonces regresamos al paso 2 y repetimos el proceso.

Se generan nuevos pesos aleatorios W_{i1}, W_{i2} en el rango [-1,1], W_{i1}=0.1, W_{i2}=0.1, W_{ib}=0.5

Se realiza la suma ponderada de las entradas por los pesos:

$$suma = \big((0 \cdot 0.1) + (1 \cdot 0.1)\big) + (1 \cdot 0.5) = 0.6$$

Para este caso obtuvimos una suma de 0.6 (de acuerdo con la condición de la función de activación), por lo que nuestra salida es 0

Salida: Obtuvimos una salida 0 y nuestra salida esperada es 0, por lo que esta condición se cumple. Finalmente pasaríamos con los valores de las siguientes entradas.

1.6. ¿Qué son las funciones de activación?

En las redes neuronales, las funciones de activación son funciones matemáticas que se aplican a la salida de cada neurona para determinar su activación. La función de activación introduce no linealidad en la salida de la neurona, lo que permite que la red neuronal aprenda relaciones no lineales en los datos. Existen varias funciones de activación utilizadas en las redes neuronales, cada una con sus propias características y ventajas. A continuación, describimos las funciones de activación más comunes:

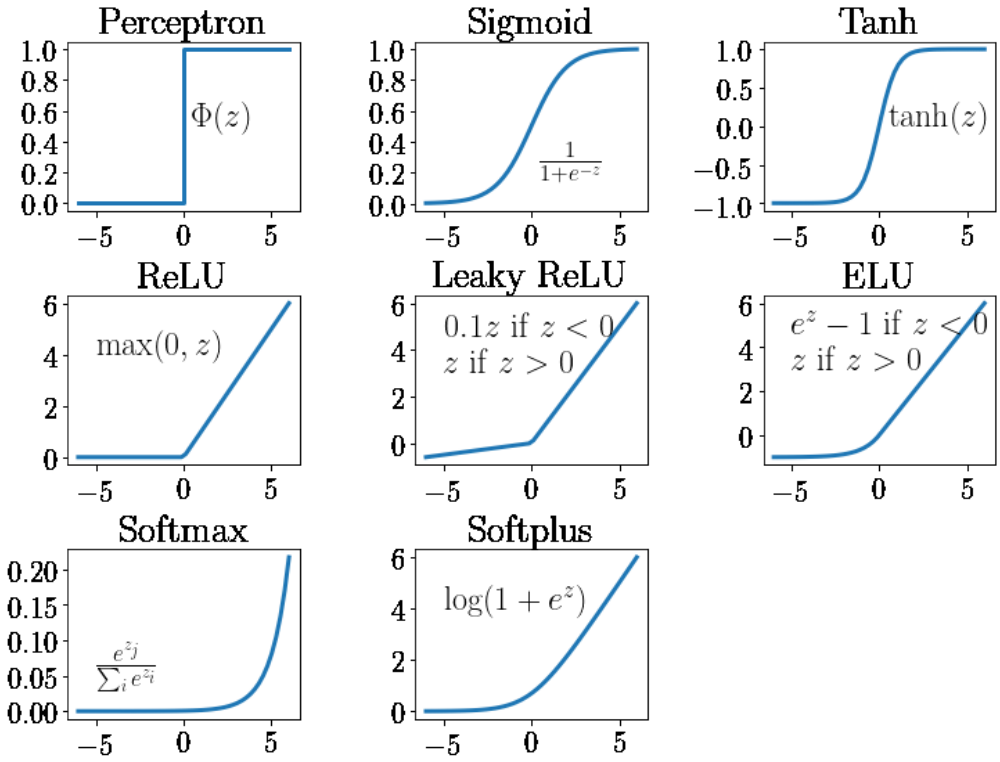


Figura 4 Funciones de activación. Fuente: (Stebner, 2020)

- Función de activación lineal:

La función de activación lineal es una función de activación muy simple que toma una entrada y devuelve la misma entrada como salida, es decir, $f(x) = x$. Debido a su simplicidad, se utiliza a menudo en las capas de salida de una red neuronal para problemas de regresión, en los que la salida de la red es un valor numérico continuo.

En los problemas de regresión, la red neuronal recibe una entrada y debe predecir un valor numérico continuo como salida. En estos casos, la función de activación lineal es útil porque permite que la red neuronal produzca valores de salida en un rango continuo sin restricciones.

Por ejemplo, en problemas de predicción de precios de viviendas, la red neuronal recibe una entrada que contiene información sobre la ubicación, el tamaño y las características de la casa, y debe predecir el precio de venta de la casa. En este caso, la función de activación lineal se puede utilizar en la capa de salida para producir una predicción numérica continua y sin restricciones.

- Función de activación sigmoide:

La función sigmoide se utiliza a menudo en redes neuronales para problemas de clasificación binaria.

En problemas de clasificación binaria, la red neuronal recibe una entrada y debe predecir si la entrada pertenece a una de las dos clases posibles. La función de activación sigmoide es útil en este caso porque su salida se puede interpretar como la probabilidad de que la entrada pertenezca a la clase positiva.

Además, la función de activación sigmoide es diferente de la función de activación lineal porque su salida está limitada en el rango de 0 a 1, lo que hace que sea útil para la

normalización de las salidas de la red neuronal. Esto puede ayudar a mejorar la estabilidad y la velocidad de convergencia de la red.

Sin embargo, la función de activación sigmoide tiene una desventaja conocida como "problema del desvanecimiento del gradiente". Este problema se produce cuando los gradientes de la función sigmoide son muy pequeños y la red neuronal tiene dificultades para aprender debido a la falta de actualizaciones de peso significativas. Por lo tanto, en redes neuronales muy profundas, se prefieren otras funciones de activación que no presentan este problema.

- Función de activación ReLU:

La función ReLU es especialmente adecuada para problemas de clasificación y regresión en los que se requiere una función de activación no lineal. También es útil para redes neuronales profundas (deep learning), ya que su derivada es fácil de calcular y no sufre del problema de desvanecimiento del gradiente.

En general, se recomienda usar la función ReLU como la función de activación predeterminada para las capas ocultas en las redes neuronales, ya que ha demostrado ser efectiva en una amplia variedad de aplicaciones de aprendizaje profundo. Además, la función ReLU también es útil cuando se desea obtener una salida esparcida, ya que establece en cero cualquier entrada negativa.

- Función de activación softmax:

La función de activación Softmax produce una distribución de probabilidad sobre las clases de salida, lo que permite que la red neuronal realice una clasificación precisa. Por lo tanto, se utiliza a menudo en problemas de clasificación en los que la salida de la red es un vector de probabilidades que indica la probabilidad de pertenencia a cada una de las clases.

Algunos ejemplos de aplicaciones en las que se utiliza la función de activación Softmax incluyen el reconocimiento de imágenes, el reconocimiento de voz y la clasificación de texto. En estas aplicaciones, la red neuronal recibe una entrada y debe predecir la clase a la que pertenece la entrada, entre varias clases posibles. Se recomienda usar la función de activación Softmax en las capas de salida de una red neuronal cuando se trata de un problema de clasificación multiclase.

1.7. El perceptrón multicapa y el algoritmo de retropropagación (Backpropagation)

La idea del perceptrón multicapa se originó en la década de 1960, cuando los científicos comenzaron a investigar redes neuronales más complejas que pudieran resolver problemas no lineales.

Uno de los pioneros en este campo fue el psicólogo Frank Rosenblatt, quien en 1958 desarrolló el perceptrón simple, una red neuronal con una sola capa de neuronas que podía resolver problemas de clasificación linealmente separables. Sin embargo, el perceptrón simple tenía limitaciones en cuanto a su capacidad para resolver problemas más complejos.

En la década de 1960, los científicos comenzaron a explorar redes neuronales más complejas, incluidas las redes neuronales multicapa. Uno de los primeros en hacerlo fue Marvin Minsky, quien junto con Seymour Papert, escribió un libro llamado "Perceptrones" en 1969, en el que

mostraban las limitaciones del perceptrón simple y proponían soluciones para mejorar su capacidad.

En particular, Minsky y Papert propusieron la idea de utilizar múltiples capas de neuronas para crear redes neuronales más complejas capaces de resolver problemas no lineales. Propusieron un algoritmo de aprendizaje para entrenar las redes neuronales multicapa, conocido como "retropropagación", que permite ajustar los pesos sinápticos de la red en función del error de salida.

A partir de la década de 1980, la retropropagación se convirtió en el algoritmo de entrenamiento más utilizado para las redes neuronales multicapa, y se convirtió en la base de muchos algoritmos de aprendizaje profundo modernos.

1.8. Funciones de pérdida y optimización

Las funciones de pérdida y de optimización son elementos cruciales en el aprendizaje de las redes neuronales. La función de pérdida cuantifica cuán lejos están las predicciones del modelo de los valores verdaderos. La elección de la función de pérdida depende del problema específico que se esté abordando. Por ejemplo, la pérdida de entropía cruzada se utiliza comúnmente para la clasificación, mientras que el error cuadrático medio se usa para la regresión.

Por otro lado, la función de optimización se usa para minimizar la función de pérdida ajustando los pesos de las neuronas en la red. La optimización es un desafío debido a la alta dimensionalidad y la naturaleza no convexa del problema. El optimizador más utilizado es el descenso del gradiente estocástico (Stochastic Gradient Descent, SGD).

Ejemplo:

Consideremos una tarea de regresión donde nuestro objetivo es predecir una variable continua. En este caso, una función de pérdida común es el error cuadrático medio (MSE).

$$\mathrm{MSE} = \frac{1}{\mathrm{N}}\sum_{\mathrm{i}=1}^{\mathrm{N}}\left(\mathrm{Y_i} - \widehat{\mathrm{Y}}_{\mathrm{i}}\right)^2$$

Donde:

- N es el número de muestras.
- Y_i es el valor real de la muestra i.
- $\underline{Y_i}$ es el valor predicho por el modelo para la muestra i.

1.9. Regularización y prevención del sobreajuste

La regularización y la prevención del sobreajuste son aspectos fundamentales en el ámbito del aprendizaje profundo (deep learning). En esta sección, exploraremos la importancia de estas técnicas y cómo pueden mejorar la calidad y el rendimiento de nuestros modelos.

El sobreajuste, también conocido como overfitting, ocurre cuando nuestro modelo se adapta demasiado bien a los datos de entrenamiento y pierde su capacidad de generalizar

correctamente a datos nuevos y no vistos anteriormente. Esto puede resultar en un rendimiento deficiente cuando aplicamos el modelo a situaciones reales.

Para mitigar el sobreajuste, la regularización se utiliza como una estrategia eficaz. Consiste en añadir términos adicionales a la función de pérdida del modelo con el objetivo de penalizar los coeficientes excesivamente grandes y reducir la complejidad del modelo. Esto ayuda a controlar la capacidad del modelo para adaptarse demasiado a los datos de entrenamiento y lo hace más propenso a generalizar correctamente.

Existen diferentes técnicas de regularización utilizadas en el aprendizaje profundo, como la regularización L1 y L2. La regularización L1 penaliza los coeficientes utilizando la norma L1, lo que favorece la generación de modelos dispersos con coeficientes más pequeños. Por otro lado, la regularización L2 utiliza la norma L2 para penalizar los coeficientes, lo que fomenta la generación de modelos con coeficientes más pequeños en general. Estas técnicas ayudan a evitar que ciertos coeficientes dominen el proceso de aprendizaje y contribuyan al sobreajuste.

Además de la regularización, existen otras técnicas que ayudan a prevenir el sobreajuste en el aprendizaje profundo. Una de ellas es el uso de conjuntos de datos de validación y pruebas separados del conjunto de entrenamiento. Estos conjuntos nos permiten evaluar el rendimiento del modelo en datos no vistos durante el entrenamiento y ajustar los hiperparámetros de manera adecuada.

Otra técnica comúnmente utilizada es la inclusión de capas de dropout en el modelo. El dropout consiste en desactivar aleatoriamente un porcentaje de las neuronas durante el entrenamiento, lo que ayuda a prevenir la dependencia entre las neuronas y evita que ciertas neuronas se especialicen demasiado en los datos de entrenamiento.

2. Arquitecturas de Deep Learning

El Deep Learning (Aprendizaje Profundo) es una rama de la Inteligencia Artificial que se basa en Redes Neuronales Artificiales (ANN - Artificial Neural Networks) con múltiples capas. Estas redes neuronales de aprendizaje profundo modelan y aprenden abstracciones complejas en datos a través de un proceso jerárquico, donde características de nivel simple y abstracto se aprenden a partir de los datos en las capas más bajas, y características más complejas se aprenden en capas más profundas. A continuación podemos observar la imagen representativa de los muchos tipos de redes neuronales que nos podemos encontrar.

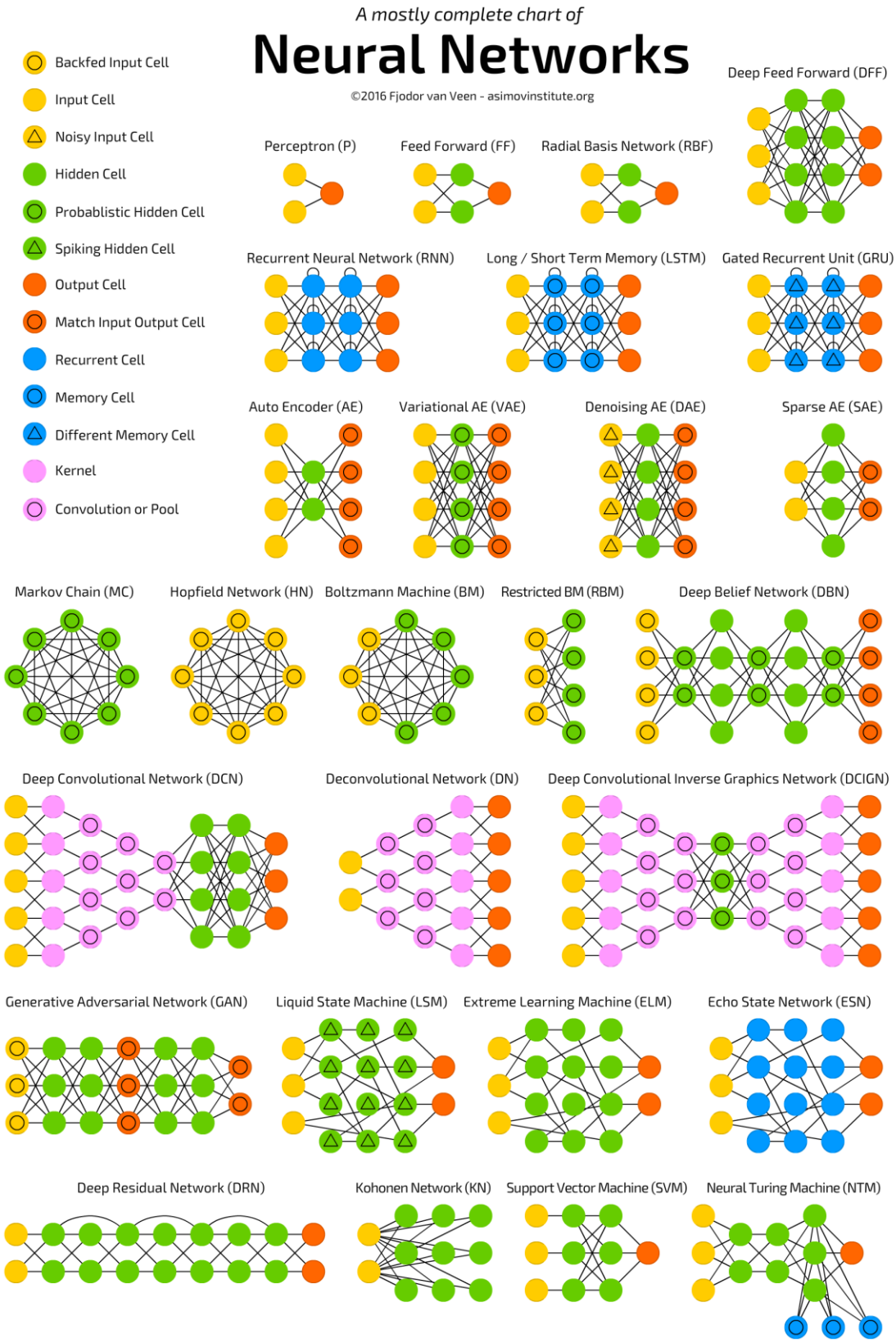


Figura 5 Diferentes tipos de redes neuronales. Fuente: (Veen, 2016)

A continuación destacaremos algunas de las más usadas:

- Redes neuronales convolucionales (CNN)

Las CNN son un tipo especial de redes neuronales diseñadas para procesar datos con una topología de cuadrícula, como una imagen, que consta de píxeles dispuestos en una estructura de ancho, alto y canales. La "convolución" en CNN se refiere a la operación matemática que se realiza. Las neuronas en la primera capa de una CNN no están conectadas a cada píxel en la imagen de entrada, sino solo a píxeles en su campo receptivo. En la segunda capa, cada neurona está conectada solo a neuronas ubicadas dentro de un pequeño rectángulo en la primera capa.

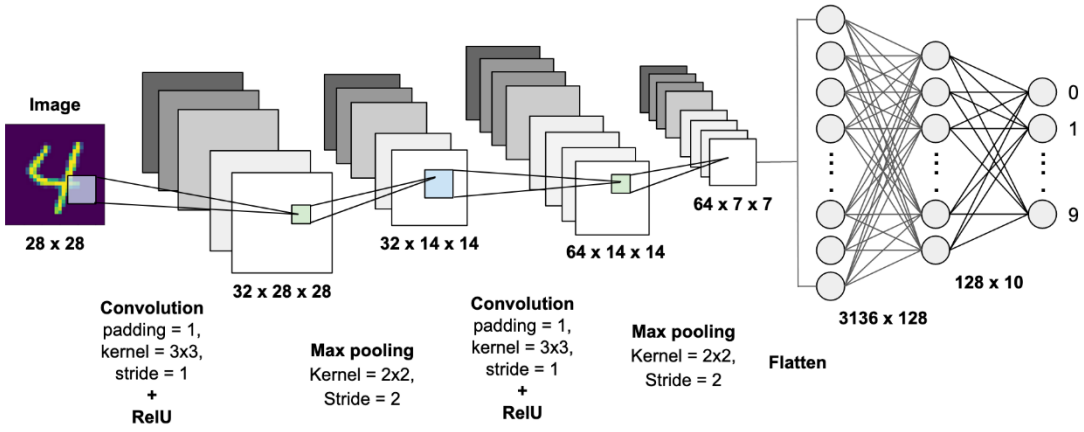


Figura 6 Estructura de CNN. Fuente: (Patel, 2019)

- Redes neuronales recurrentes (RNN)

Las RNN se utilizan principalmente para datos secuenciales. A diferencia de las redes neuronales tradicionales, las RNN tienen ciclos en su topología, lo que las hace más adecuadas para tareas que incluyen secuencias temporales y listas. Tienen la capacidad de usar su memoria interna para procesar secuencias de entradas, lo que las hace ideales para tareas como la predicción de series temporales y el reconocimiento de voz.

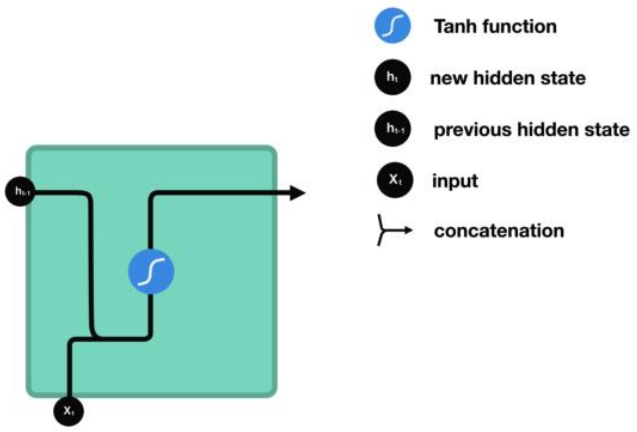
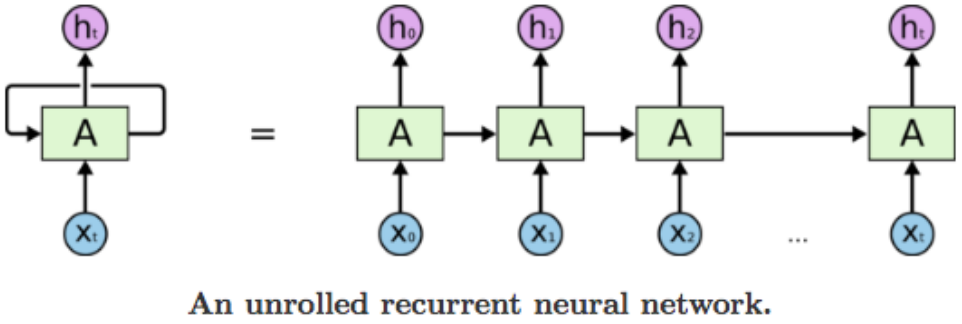


Figura 7 Estructura de RRN. Fuente: (Phi, 2018)

- Long Short-Term Memory (LSTM)

Las LSTM son una variante específica de las RNN capaces de aprender dependencias a largo plazo. Las LSTM funcionan muy bien en la práctica y ahora se utilizan ampliamente. Tienen la ventaja sobre los métodos tradicionales en que pueden evitar el problema del desvanecimiento del gradiente, que es el problema principal de las RNN.



An unrolled recurrent neural network.

Figura 8 Estructura de LSTM. Fuente: (Mittal, 2019)

- Gated Recurrent Unit (GRU)

Las GRU son una variante de las RNN, algo similar a las LSTM. Sin embargo, a diferencia de las LSTM, que utilizan un estado de celda para mantener la información, las GRU utilizan el propio estado oculto para transferir información. Esta es una diferencia importante en la arquitectura y hace que las GRU sean más eficientes en términos de memoria y computacionalmente más rápidas, debido a la menor cantidad de operaciones de tensor. Sin embargo, esto no garantiza que siempre superen a las LSTM, y la elección entre las LSTM y las GRU puede depender de la tarea específica.

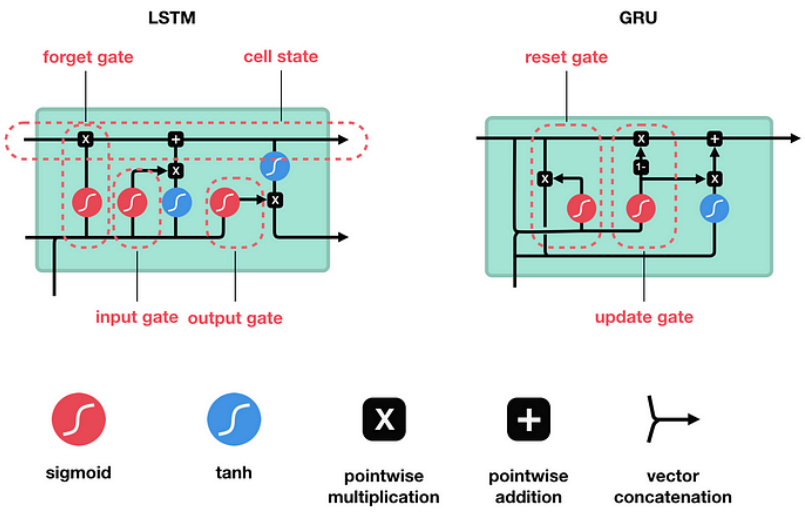


Figura 9 Estructura de GRU vs LSTM. Fuente: (Phi, 2018)

- Redes generativas adversarias (GAN)

Las Redes Generativas Adversarias (GANs) representan un interesante desarrollo en la arquitectura de Deep Learning. Básicamente, una GAN consta de dos componentes principales: un generador y un discriminador. El generador crea nuevas instancias de datos, mientras que el discriminador evalúa su autenticidad, es decir, si se asemejan a los datos de entrenamiento.

Este enfoque tiene aplicaciones interesantes, especialmente en la generación de imágenes realistas, la transferencia de estilos y la mejora de la resolución de las imágenes.

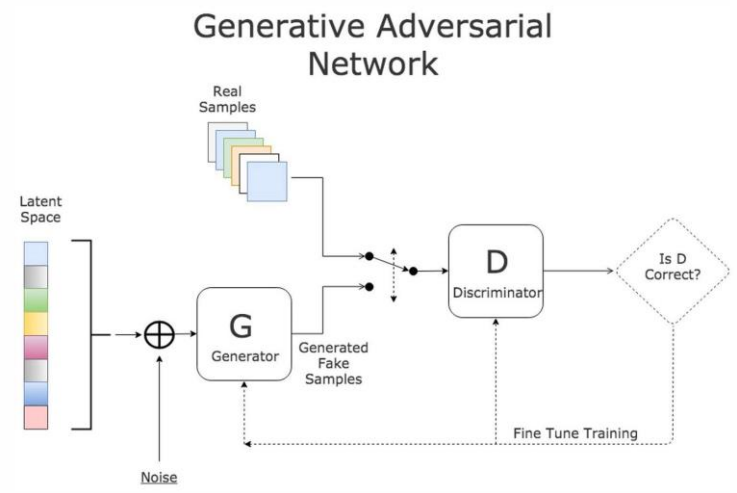


Figura 10 Estructura de GAN. Fuente: (Mosquera, 2018)

- Autoencoders y autoencoders variacionales (VAE)

Los Autoencoders son redes neuronales utilizadas para la compresión y codificación de datos. Un Autoencoder tiene dos componentes principales: un codificador que comprime la entrada en un código más pequeño, y un decodificador que reconstruye la entrada a partir del código. Los Autoencoders Variacionales (VAE) son una extensión de los Autoencoders que añaden una capa de aleatoriedad al proceso, permitiendo generar nuevas instancias de datos. Esto es útil en muchas tareas, incluyendo la generación de imágenes y la eliminación de ruido en los datos.

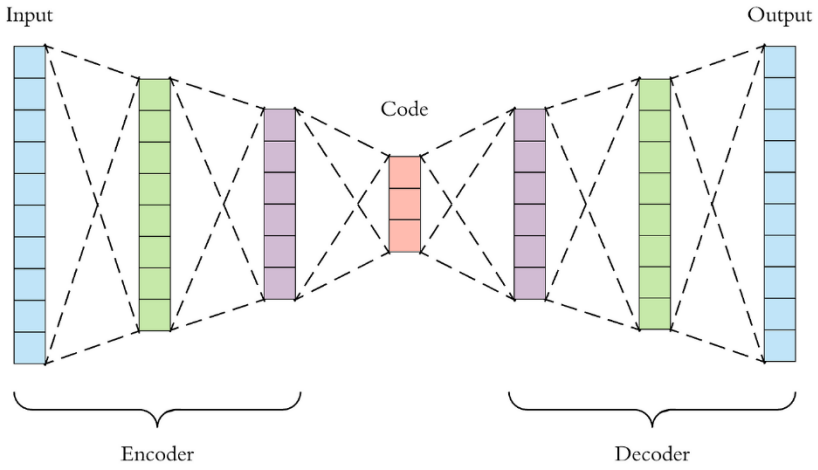


Figura 11 Estructura de VAE. Fuente: (Dertat, 2017)

- Transformers y arquitecturas derivadas

Finalmente, las arquitecturas de Transformers representan una innovación significativa en el campo del procesamiento del lenguaje natural (NLP). Los Transformers abandonan la noción de orden secuencial en favor de un enfoque basado en la atención, que permite al modelo centrarse en diferentes partes de la entrada cuando toma decisiones. Esto ha llevado a avances significativos en las tareas de NLP, incluyendo la traducción automática y la generación de texto. Un ejemplo popular de este tipo de arquitectura es BERT (Bidirectional Encoder

Representations from Transformers), que ha establecido nuevos estándares en varias tareas de NLP.

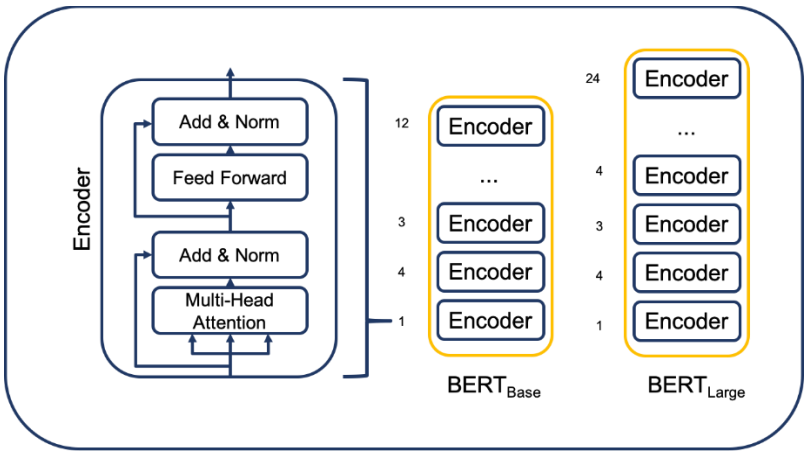


Figura 12 Estructura de Bert. Fuente: (R. Evtimov, 2020)

3. Herramientas y librerías para Deep Learning

3.1. TensorFlow

TensorFlow es una de las bibliotecas más utilizadas en el campo del Deep Learning. Desarrollada por el equipo de Google Brain, ofrece una gran flexibilidad y capacidad para trabajar con arquitecturas complejas de redes neuronales. TensorFlow permite la definición de modelos de aprendizaje profundo a un nivel bastante bajo, lo que ofrece a los usuarios un control detallado del proceso de construcción y entrenamiento del modelo. Esto, sin embargo, puede resultar en una curva de aprendizaje más empinada para aquellos que son nuevos en la disciplina.

TensorFlow también ofrece TensorFlow Lite para dispositivos móviles y embebidos y TensorFlow.js para ejecución en JavaScript. Además, proporciona un soporte sólido para la implementación en producción, lo que permite el despliegue de modelos a gran escala en diversas plataformas.

3.2. Keras

Keras es una biblioteca de Python de alto nivel que se ejecuta sobre TensorFlow, Theano y CNTK. Su enfoque principal es la facilidad de uso, por lo que es un gran punto de partida para los principiantes en el aprendizaje profundo. Keras simplifica el proceso de construcción de modelos al ofrecer módulos predefinidos para las capas comunes de una red neuronal, lo que permite una rápida prototipación.

Aunque Keras no ofrece la misma cantidad de control bajo nivel como TensorFlow, su simplicidad y su interfaz fácil de usar lo han convertido en una opción popular para los desarrolladores y los investigadores que buscan construir prototipos rápidos y experimentar con diferentes arquitecturas de redes neuronales. A partir de la versión 2.0 de TensorFlow, Keras está integrado directamente en TensorFlow como su interfaz de alto nivel, lo que permite a los usuarios aprovechar la potencia de TensorFlow con la simplicidad de Keras.

3.3. PyTorch

PyTorch, desarrollado por Facebook's AI Research lab, es otra biblioteca popular para el Deep Learning. Proporciona dos características principales: operaciones tensoriales con soporte para GPU y cálculo automático de gradientes para la construcción y el entrenamiento de redes neuronales. La diferencia clave entre PyTorch y TensorFlow/Keras es su enfoque en la "programación definida por ejecución" (Eager execution), lo que significa que las operaciones se ejecutan inmediatamente a medida que se definen en el código.

Este enfoque ofrece ventajas como una mayor flexibilidad, un fácil depurado y un diseño de red natural que resulta más intuitivo para los desarrolladores de Python. Además, PyTorch tiene un fuerte soporte para la investigación de IA, con herramientas y utilidades que facilitan la experimentación. En el aspecto de la implementación, PyTorch proporciona TorchServe para el despliegue de modelos y ONNX para la interoperabilidad con otros marcos de Deep Learning.

3.4. Comparación y selección de herramientas

La elección de la herramienta correcta para un proyecto específico de aprendizaje profundo puede ser una tarea desafiante debido a la variedad de bibliotecas disponibles. Sin embargo, podemos hacer una evaluación comparativa en función de factores como la facilidad de uso, la flexibilidad, la eficiencia y la comunidad de soporte.

- TensorFlow (TF) es conocido por su versatilidad y flexibilidad. Con su amplio soporte para diversas operaciones, TF permite el diseño de una gama variada de modelos de redes neuronales. Además, el ecosistema de TF incluye TensorFlow Lite para dispositivos móviles y de borde, y TensorFlow Extended (TFX) para la producción de ML. Aunque inicialmente TF tenía una curva de aprendizaje dura, la introducción de la API de alto nivel Keras ha simplificado significativamente el desarrollo de modelos en TensorFlow.
- Keras, que es una interfaz de alto nivel para TensorFlow, es ampliamente reconocido por su facilidad de uso. Aunque no es tan flexible como TensorFlow en su núcleo, proporciona suficiente funcionalidad para la mayoría de las tareas comunes de aprendizaje profundo. Además, la sintaxis intuitiva de Keras hace que los modelos sean más legibles y fáciles de entender.
- PyTorch, por otro lado, es amado por su diseño intuitivo y su flujo de trabajo amigable con Python. Su adopción en la comunidad de investigación ha estado creciendo rápidamente debido a su flexibilidad y la capacidad de realizar cálculos dinámicos. Sin embargo, hasta hace poco, PyTorch carecía de soporte sólido para la implementación en producción, pero esta brecha se ha reducido con la introducción de TorchServe y TorchScript.

En resumen, la elección de la herramienta depende en gran medida de las necesidades específicas del proyecto, la experiencia del equipo y el compromiso con la implementación en producción. En general, TensorFlow y PyTorch dominan el campo debido a su versatilidad y soporte para el diseño de diversos modelos. Sin embargo, Keras puede ser una opción ideal para principiantes o proyectos que no requieren una personalización extrema del modelo.

4. Caso práctico: Detección de enfermedades mediante el estudio de globos oculares usando Deep Learning

4.1. Definición del problema y objetivos

Para la elección del conjunto de datos para este proyecto hemos tenido en cuenta el criterio de la aplicabilidad y usabilidad en mundo real. Por este motivo, nos decidimos por un conjunto de datos que contiene información sobre +3000 pacientes y +7000 ojos individuales. Esta información comprende datos como la edad, género y enfermedades diagnosticadas en ese paciente a partir del estudio de sus ojos.

Las enfermedades que nos encontramos diagnosticadas en este conjunto de datos son:

- Normal (N),
- Diabetes (D),
- Glaucoma (G),
- Cataratas (C),
- Degeneración Macular relacionada con la edad (A),
- Hipertension (H),
- Miopia patológica (M),
- Otras enfermedades (O)

Con este caso de estudio sobre la mesa, definimos el objetivo de proyecto, el cuál es la detección temprana de enfermedades estudiando el fondo del globo ocular, usando modelos avanzados de inteligencia artificial, en concreto de Deep Learning.

Para ello, primero desarrollaremos un modelo que será entrenado con imágenes las cuales estarán clasificadas con las enfermedades correspondientes o en su defecto, clasificadas como un ojo normal. Una vez realizado esto, pondremos a prueba este modelo con un conjunto de evaluación, con imágenes que no haya visto anteriormente y estudiaremos su precisión para acertar, así como diferentes métricas que nos dirán como de bien o de mal ha respondido nuestro modelo.

Teniendo en cuenta lo anterior, primero nos centraremos en entrenar un modelo de clasificación con variable objetivo binaria (0's o 1's), esto es, si la imagen estudiada es normal será 0, y si posee alguna enfermedad, se le asignará un 1. Con esta premisa, nos quedarán los siguientes subconjuntos de datos:

- Ojos normales vs Resto de ojos con alguna enfermedad
- Ojos normales vs Ojos con diabetes
- Ojos normales vs Ojos con cataratas
- Ojos normales vs Ojos con glaucoma

4.2. Preprocesamiento y análisis exploratorio de los datos

El primer paso para abordar el análisis es hacer un análisis exploratorio de los datos. Este análisis consistirá en intentar extraer posibles patrones e información valiosa explorando sólo

el conjunto de datos provisto, sin entrar al modelado y entrenamiento basados en Deep Learning. Para ello utilizaremos métodos estadísticos y diferentes tipos de gráficos.

A continuación observaremos las figuras 13 y 14, que representan la frecuencia de pacientes varones y mujeres con alguna enfermedad o no, distribuidos por edad:

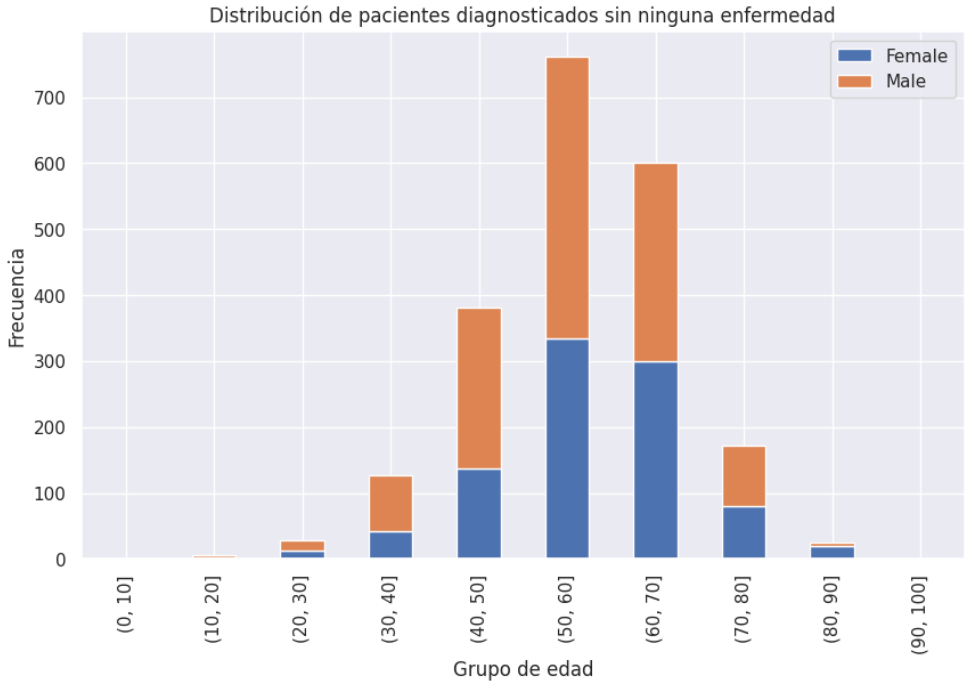


Figura 13 Distribución de pacientes sin ninguna enfermedad. Fuente: Elaboración propia

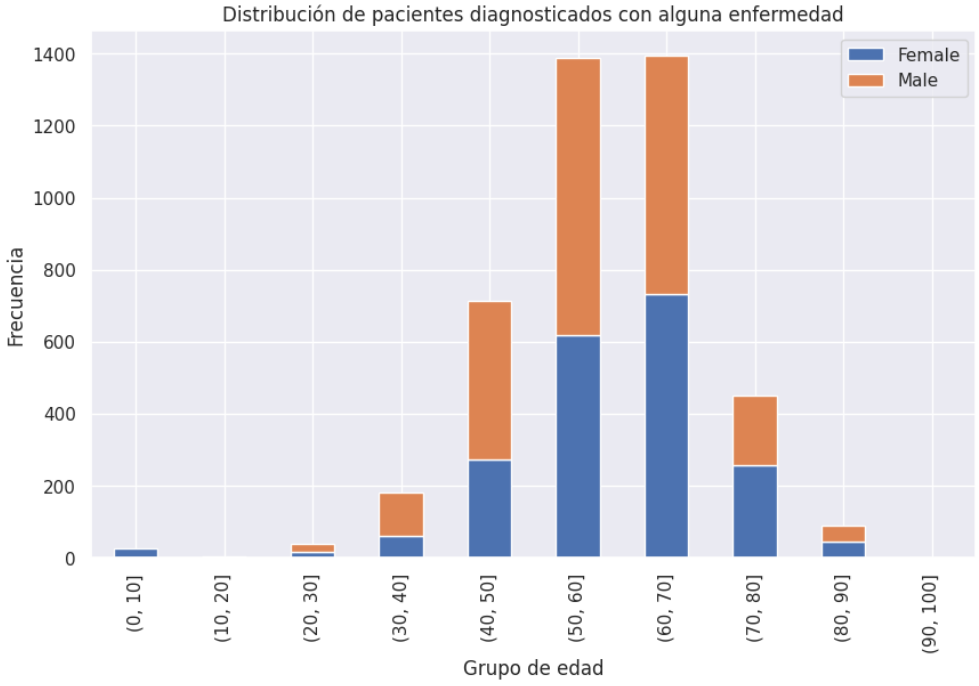


Figura 14 Distribución de pacientes con alguna enfermedad. Fuente: Elaboración propia

Podemos obtener la siguiente lectura:

- En todos los rangos de edad tenemos más hombres que mujeres. En el rango de 50-70 años de pacientes con ojos con enfermedad, los hombres mantienen el número de casos pero en las mujeres sigue creciendo los casos hasta los 70 años.
- Las pacientes mujeres a pesar de tener una avanzada edad no se les ha diagnosticado ninguna enfermedad, cuando este hecho no ocurre en hombres.
- Es curioso que a las edades más tempranas, sólo se hayan recogido mujeres con enfermedad, lo que podrían ser casos aislados, ya que a esas edades no se espera ningún deficiencia a nivel ocular, salvo las que se desarrollen al nacer o por genética.

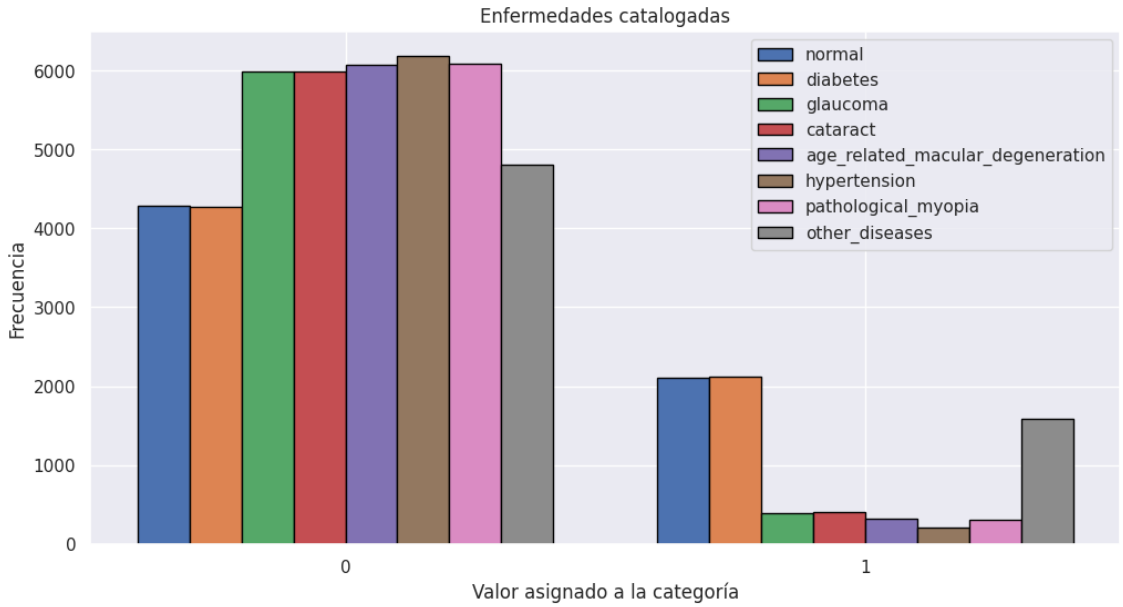


Figura 15 Frecuencia de enfermedades diagnosticadas. Fuente: Elaboración propia

En el gráfico de la figura 15 vemos las enfermedades más frecuentes, siendo la diabetes la principal enfermedad diagnosticada, seguida del resto de enfermedades, que están al mismo nivel, y sin contar a aquellas enfermedades que no están catalogadas.

Por último, estudiamos las enfermedades diferenciando por ojos en las figuras 16 y 17:

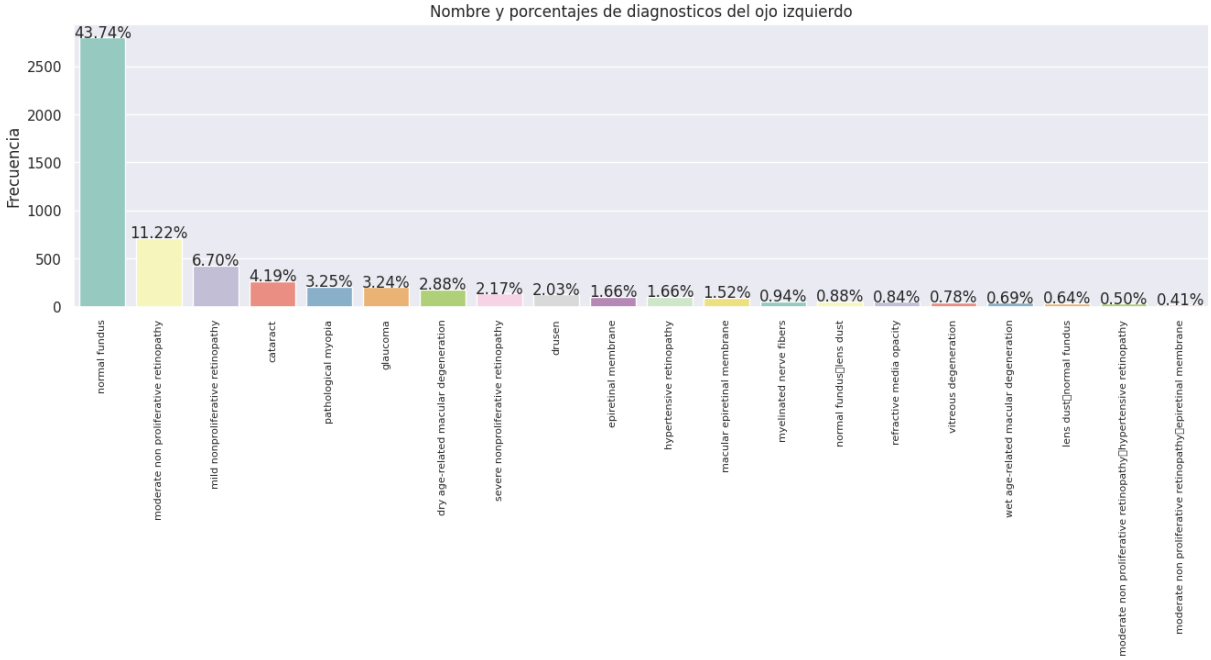


Figura 16 Frecuencia en % de enfermedades diagnosticadas en ojo izquierdo. Fuente: Elaboración propia

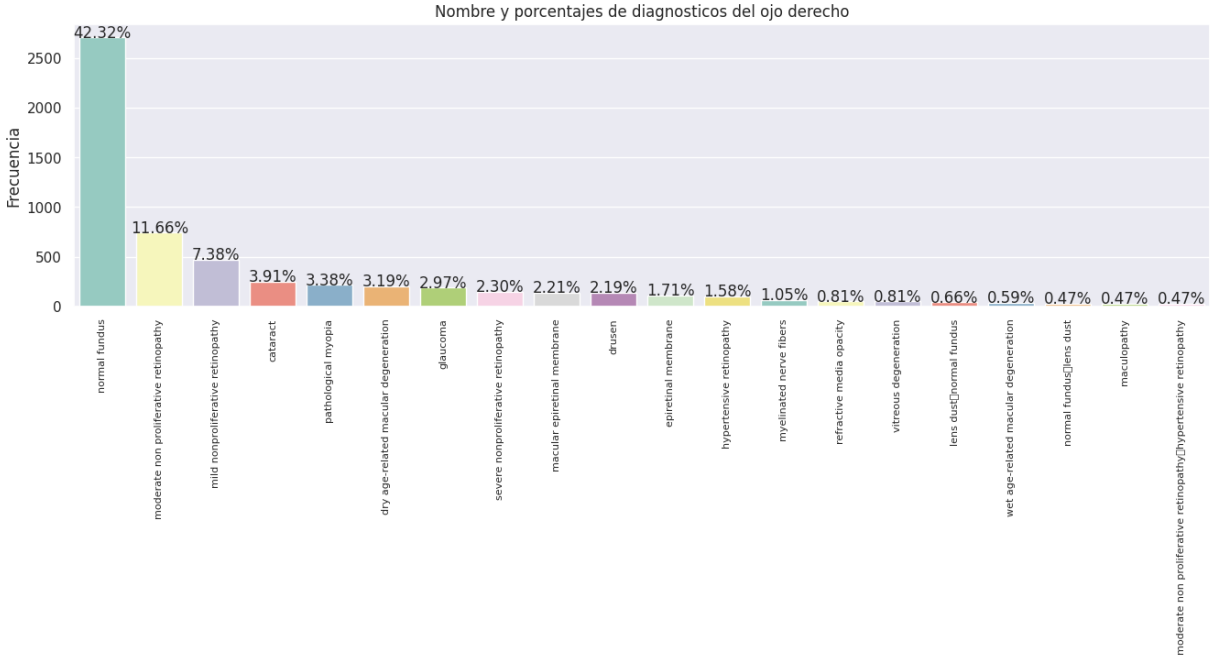


Figura 17 Frecuencia en % de enfermedades diagnosticadas en ojo derecho. Fuente: Elaboración propia

En este análisis no observamos ninguna diferencia significativa entre ambos ojos por lo que descartamos que las enfermedades estén referenciadas a un ojo en concreto.

Una vez analizado el conjunto de datos, podemos enmarcar el problema, tal y como se presenta en el conjunto de datos disponible, de varias formas diferentes, como por ejemplo:

- Un problema de clasificación multiclase multietiqueta, en el que cada instancia (imagen) puede contener varias clases (por ejemplo, una imagen que pertenezca tanto a la clase Diabetes como a la clase Otros).
- Un problema de clasificación multiclase, en el que cada instancia puede pertenecer a una única clase (utilizando las etiquetas finales (objetivo) como clases, por ejemplo).
- Un problema de clasificación binaria, en el que sólo las instancias que pertenecen a 2 clases son muestras del conjunto de datos (por ejemplo, sólo las instancias Cataratas y Normal).

Estos enfoques son posibles porque el propio conjunto de datos nos da una variable target en la que sólo hay una enfermedad posible, pero a su vez para cada ojo, nos provee de una columna por enfermedad, y existen casos en los que hay varias de estas categorías con valor 1 para el mismo ojo.

En nuestro caso, hemos optado por desarrollar un modelo de clasificación binaria enfocándonos en los ojos sanos y estudiándolos frente a otras enfermedades, de manera unitaria.

4.2.1. Carga y preprocesamiento del conjunto de datos

En esta sección, se detallará el proceso de preprocesamiento de datos realizado para el desarrollo del modelo de detección de enfermedades oculares. El objetivo principal es analizar y modelar datos médicos relacionados con enfermedades oculares, utilizando imágenes de los ojos y su correspondiente etiqueta de diagnóstico.

El conjunto de datos utilizado en este proyecto se encuentra en el archivo CSV "full_df.csv", el cual contiene información relevante sobre las imágenes y sus diagnósticos asociados. Para comenzar, se carga el conjunto de datos y se realiza un preprocesamiento inicial para adecuarlo a nuestras necesidades.

El primer paso consiste en cargar el archivo CSV y realizar algunas transformaciones en los nombres de las columnas para facilitar su manipulación. A continuación, se utiliza la función ***pd.get_dummies()*** para convertir las etiquetas de diagnóstico en variables binarias, lo que nos permitirá realizar clasificación multiclase si es necesario.

La primera opción de procesamiento de datos consiste en realizar una comparación entre ojos normales y ojos con diferentes enfermedades. Este enfoque nos permitirá desarrollar un modelo de clasificación binaria para detectar si un ojo presenta una enfermedad ocular o si es considerado normal.

Para llevar a cabo esta opción, se implementan las funciones ***is_normal()*** y ***is_XXX()***, donde XXX representa el nombre de una enfermedad específica, como catarata, diabetes o glaucoma. Estas funciones se utilizan para asignar valores binarios a las columnas correspondientes en el conjunto de datos, indicando si una determinada enfermedad está presente en el ojo o no.

Una vez que se han asignado los valores binarios adecuados, se realiza una separación de las imágenes según su categoría: ojos normales y ojos con enfermedades. Se selecciona una cantidad específica de imágenes de cada categoría para equilibrar el conjunto de datos y evitar sesgos.

4.2.2. Creación del conjunto de datos

Después de realizar el preprocesamiento de los datos, se procede a crear el conjunto de datos para su posterior uso en el desarrollo del modelo. Para ello, se utiliza la función ***create_dataset()***, que toma como entrada la categoría de las imágenes (normal o con enfermedad) y su etiqueta correspondiente (0 para normal y 1 para con enfermedad).

Dentro de esta función, se carga cada imagen y se realiza un redimensionamiento a un tamaño específico (en este caso, 224x224 píxeles) utilizando la biblioteca OpenCV. A continuación, las imágenes redimensionadas y sus etiquetas se agregan al conjunto de datos.

Finalmente, se mezcla aleatoriamente el conjunto de datos para evitar cualquier sesgo en el orden de las imágenes y se devuelve el conjunto de datos creado.

4.2.3. Ejecución del código

El código proporcionado implementa todas las etapas descritas anteriormente para las diferentes opciones de procesamiento de datos. Para ejecutar el código, es necesario cargar el archivo CSV "full_df.csv" y especificar la ruta de las imágenes correspondientes.

Se generan diferentes conjuntos de datos según la opción elegida (normal vs. enfermedad específica), los cuales se almacenan en variables como *dataset_normal, dataset_cataract*, etc.

4.3. Diseño e implementación del modelo

Después de realizar un análisis de datos y obtener conclusiones a partir de gráficos, es el momento de procesar los datos antes de entrenar el modelo. Para ello, he creado una función llamada **create_model** que me permite construir el modelo y aplicar diferentes configuraciones según mis necesidades.

La función **create_model** acepta varios parámetros, siendo el primero **model_to_apply**, que me permite especificar el modelo base que utilizaré. En este caso, he implementado dos opciones: "vgg19" e "inception_V3". Estos modelos pre-entrenados han demostrado buen rendimiento en tareas de clasificación de imágenes, por lo que los utilizo como punto de partida.

Si el modelo seleccionado es "vgg19", cargo el modelo VGG19 pre-entrenado utilizando **tf.keras.applications.VGG19**. Establezco los pesos en "imagenet" y especifico que no quiero incluir la capa superior, ya que agregaré mis propias capas personalizadas. Además, establezco el tamaño de entrada de la imagen según mis requisitos. A continuación, congelo todas las capas del modelo VGG19 para evitar que se modifiquen durante el entrenamiento. Luego, construyo mi modelo secuencial y agrego el modelo VGG19 como una capa utilizando **model.add(vgg)**. Para permitir la entrada del modelo VGG19 a las capas personalizadas, utilizo **model.add(tf.keras.layers.Flatten())** para aplanar los datos.

En el caso de seleccionar el modelo "inception_V3", sigo un proceso similar. Cargo el modelo InceptionV3 pre-entrenado utilizando **tf.keras.applications.inception_v3.InceptionV3**. Al igual que antes, congelo todas las capas del modelo InceptionV3 y construyo mi modelo secuencial. Agrego el modelo InceptionV3 como una capa utilizando **model.add(inception)**. Nuevamente, utilizo **model.add(tf.keras.layers.Flatten())** para aplanar los datos.

Además de la elección del modelo base, la función **create_model** también me permite configurar otras opciones, como el dropout, la regularización y el gamma. El dropout es una técnica de regularización que ayuda a prevenir el sobreajuste al apagar aleatoriamente algunas

neuronas durante el entrenamiento. Si se especifica un valor mayor a cero para **dropout_rate**, agrego una capa de dropout a mi modelo utilizando **model.add(tf.keras.layers.Dropout(dropout_rate))**.

En cuanto a la regularización, tengo la opción de elegir entre "l1" y "l2". Estas son técnicas de regularización que penalizan los pesos del modelo para evitar el sobreajuste. Si **regularization** es igual a "l1", agrego una capa Dense con una función de activación "relu" y una regularización L1 utilizando **kernel_regularizer=tf.keras.regularizers.l1()**. De manera similar, si **regularization** es igual a "l2", agrego una capa Dense con regularización L2.

El parámetro **gamma** me permite configurar el gamma para la inicialización de los pesos de la capa Dense. Si **gamma** es igual a "scale", agrego una capa Dense con una función de activación "relu", una regularización L2 y una inicialización de pesos "glorot_normal". Además, establezco un sesgo inicial constante en 1.0 utilizando **bias_initializer=tf.keras.initializers.Constant(value=1.0)**.

Finalmente, agrego una capa Dense de salida con una función de activación "sigmoid" para la clasificación binaria de las imágenes.

Para la compilación del modelo, utilizo el optimizador Adam con una tasa de aprendizaje de 0.001. Como estoy realizando una clasificación binaria, utilizo la pérdida **BinaryCrossentropy** y las métricas de precisión (**accuracy**), área bajo la curva (**auc**), precisión (**prec**) y área bajo la curva de precisión y recuperación (**prc**).

4.3.1. Modelos preentrenados

4.3.1.1. VGG19

El modelo preentrenado VGG19 es una arquitectura de red neuronal convolucional (CNN) que ha sido entrenada en una gran cantidad de imágenes de la base de datos ImageNet. Fue desarrollado por el Visual Geometry Group (VGG) en la Universidad de Oxford.

El modelo VGG19 se compone de un total de 19 capas, incluyendo capas convolucionales, capas de agrupación y capas totalmente conectadas. Estas capas se organizan en bloques, y cada bloque está compuesto por una serie de capas convolucionales seguidas por una capa de agrupación. La arquitectura VGG19 es conocida por su simplicidad y consistencia en el tamaño de los filtros convolucionales y las capas de agrupación.

Las capas internas más importantes del modelo VGG19 son:

1. Convolutional Layers: Estas capas se encargan de extraer características de las imágenes de entrada mediante filtros convolucionales. En el caso del modelo VGG19, las capas convolucionales están compuestas principalmente por filtros de tamaño 3x3 con una función de activación ReLU. Estas capas realizan la tarea de detectar características de bajo y alto nivel en las imágenes.
2. Pooling Layers: Después de cada bloque de capas convolucionales, se aplica una capa de agrupación (también conocida como capa de submuestreo o pooling). La capa de agrupación reduce la dimensionalidad de las características extraídas por las capas convolucionales, lo que ayuda a reducir la cantidad de parámetros en el modelo y a aumentar la eficiencia computacional. En el caso del modelo VGG19, se utiliza la técnica de agrupación por máximo (max pooling) con ventanas de tamaño 2x2 y un desplazamiento de 2.

3. Fully Connected Layers: Al final de la arquitectura, se añaden capas totalmente conectadas que realizan la tarea de clasificación. Estas capas reciben como entrada las características extraídas por las capas convolucionales y las capas de agrupación y generan las probabilidades de clasificación para las diferentes categorías. En el caso del modelo VGG19, se utilizan capas densas con una función de activación ReLU.

Las capas internas del modelo VGG19 se pueden visualizar en forma de diagrama de capas., tal como se muestra en la figura 18:

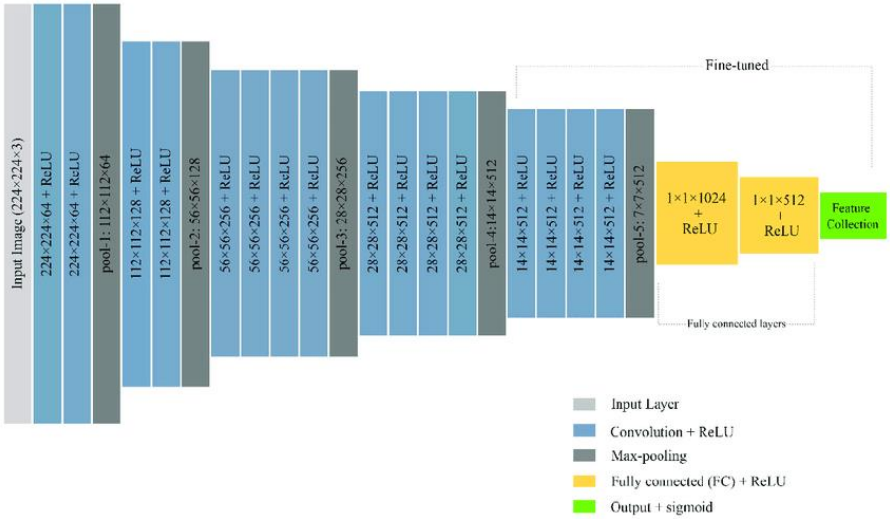


Figura 18 Estructura de modelo preentrenado VGG19. Fuente: (Rahman, 20)

4.3.1.2. InceptionV3

El modelo pre-entrenado InceptionV3 es una reconocida arquitectura de red neuronal convolucional (CNN) que ha sido ampliamente utilizada para tareas de clasificación de imágenes. Fue desarrollado por el equipo de Google Brain y ha logrado resultados impresionantes en diversos desafíos de visión por computadora.

InceptionV3 se caracteriza por su estructura profunda y el uso de un módulo único llamado módulo Inception. Este módulo está diseñado para capturar información en diferentes escalas espaciales utilizando capas convolucionales paralelas con diferentes tamaños de filtros (1x1, 3x3, 5x5) y operaciones de pooling. Estas ramas paralelas se concatenan para formar una representación más rica de los datos de entrada.

Las capas internas más importantes del modelo InceptionV3 incluyen:

1. Convolutional Layers: InceptionV3 comienza con varias capas convolucionales que realizan la extracción local de características. Estas capas aplican un conjunto de filtros aprendibles a la imagen de entrada, detectando diferentes patrones y características.
2. Inception Modules: El módulo Inception es el componente clave de InceptionV3. Consiste en múltiples ramas convolucionales paralelas con diferentes tamaños de filtros, seguidas de una operación de concatenación. Esto permite que la red capture características a diferentes escalas y aprenda representaciones diversas. El módulo Inception reduce el costo computacional utilizando convoluciones 1x1 para la reducción de dimensionalidad antes de aplicar convoluciones más grandes.
3. Pooling Layers: InceptionV3 incorpora capas de pooling para reducir las dimensiones espaciales de los mapas de características y extraer la información más relevante. Por

lo general, se utiliza el max pooling con una ventana de 3x3 y un paso de 2 en InceptionV3.

4. Fully Connected Layers: Hacia el final de la red, se emplean capas completamente conectadas para la clasificación. Estas capas toman las características extraídas y las mapean a las etiquetas de clase correspondientes. En InceptionV3, las capas completamente conectadas finales están seguidas de una función de activación softmax para generar probabilidades de clase.

Para visualizar las capas internas y la estructura general del modelo InceptionV3, podemos observar la figura 19:

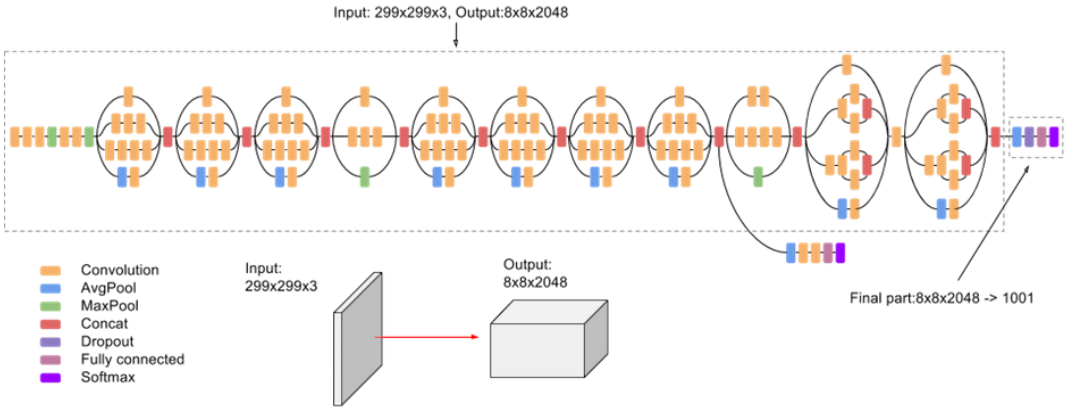


Figura 19 Estructura de modelo preentrenado InceptionV3. Fuente: (Google Cloud Documentación, s.f.)

4.3.2. Arquitectura, optimizador y función de pérdida utilizados

En primer lugar, hemos optado por utilizar una arquitectura secuencial debido a su simplicidad y flexibilidad. La arquitectura secuencial nos permite construir nuestro modelo de manera secuencial, agregando capas una tras otra. Esto es especialmente útil en nuestro caso, ya que podemos diseñar fácilmente una estructura de capas que se adapte a las características y complejidad de nuestro problema de clasificación de ojos.

Además, la arquitectura secuencial es compatible con una amplia variedad de capas disponibles en TensorFlow, lo que nos brinda la libertad de seleccionar las capas más apropiadas para nuestro problema. Por ejemplo, podemos incorporar capas convolucionales para extraer características visuales relevantes de las imágenes de los ojos y capas completamente conectadas para la etapa de clasificación. La arquitectura secuencial también nos permite añadir capas de dropout para evitar el sobreajuste y mejorar la generalización del modelo.

En cuanto a la elección de la función de pérdida, hemos optado por BinaryCrossEntropy (entropía cruzada binaria) debido a que nuestro problema de clasificación es binario: enfermedad/ojo sano. La entropía cruzada binaria es una función de pérdida adecuada para problemas de clasificación binaria, ya que penaliza de manera efectiva la diferencia entre las predicciones del modelo y las etiquetas reales. Nos permite evaluar la capacidad del modelo para distinguir entre clases y ajustar los pesos de manera apropiada durante el entrenamiento.

Por último, hemos seleccionado el optimizador Adam para el entrenamiento de nuestro modelo. Adam es un algoritmo de optimización popular y eficiente que combina los beneficios de los métodos de descenso de gradiente estocástico (SGD) y de momentum. Adam adapta automáticamente la tasa de aprendizaje a medida que se entrena el modelo, lo que permite un entrenamiento más rápido y una convergencia más estable. Además, Adam también realiza ajustes basados en los momentos de primer y segundo orden de los gradientes, lo que ayuda a evitar los problemas de aprendizaje lento o estancamiento en los mínimos locales.

Como conclusión, podemos decir que hemos escogido la arquitectura secuencial para nuestro modelo debido a su simplicidad y flexibilidad, BinaryCrossEntropy como función de pérdida para nuestro problema de clasificación binaria de ojos y el optimizador Adam para el entrenamiento eficiente y estable del modelo. Estas decisiones están respaldadas por su idoneidad y éxito demostrado en problemas similares de clasificación de imágenes médicas.

Los modelos generados en nuestro código por lo general tendrán la estructura que se muestra en la figura 20, salvo cuando se añadan más capas:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 5, 5, 2048)        21802784

 flatten (Flatten)           (None, 51200)             0

 dense (Dense)               (None, 1)                 51201

=================================================================
Total params: 21,853,985
Trainable params: 51,201
Non-trainable params: 21,802,784
_________________________________________________________________
```

Figura 20 Ejemplo de modelo DL de tipo secuencial. Fuente: Elaboración propia

4.3.3. Aplicando RandomizedSearchCV

RandomizedSearchCV es una técnica de búsqueda de hiperparámetros que se basa en la evaluación exhaustiva de diferentes combinaciones de parámetros dentro de un rango predefinido. Su objetivo es encontrar la mejor configuración de parámetros que maximice el rendimiento del modelo. En nuestro caso, hemos utilizado RandomizedSearchCV en combinación con el estimador KerasClassifier para buscar los mejores hiperparámetros para nuestro modelo de aprendizaje profundo.

El uso de KerasClassifier como estimador nos permite utilizar modelos de Keras en conjunción con RandomizedSearchCV. Esto nos brinda la flexibilidad de utilizar la potencia y la versatilidad de las bibliotecas de aprendizaje profundo de Keras, al tiempo que nos beneficiamos de las capacidades de búsqueda de hiperparámetros de RandomizedSearchCV.

Al utilizar RandomizedSearchCV, hemos definido una distribución de parámetros que queremos explorar. Esta distribución de parámetros nos permite especificar los rangos o los conjuntos de valores que queremos probar para cada hiperparámetro. Al hacerlo, estamos agregando sentido y estructura a la búsqueda de hiperparámetros, lo que nos ayuda a centrarnos en las configuraciones más prometedoras y a evitar la exploración ciega de todo el espacio de parámetros.

Durante la ejecución de RandomizedSearchCV, se evalúan diferentes combinaciones de parámetros mediante validación cruzada. Cada combinación de parámetros se entrena y evalúa en múltiples divisiones del conjunto de datos, lo que nos proporciona una estimación confiable del rendimiento del modelo para esa configuración en particular. Al finalizar la búsqueda, RandomizedSearchCV nos devuelve la mejor configuración de parámetros encontrada, la cual podemos utilizar para construir nuestro modelo final con los mejores hiperparámetros.

En nuestro caso aplicaremos esta técnica para dar con la mejor combinación de capas para nuestro modelo final, añadiendo u obviando capas de dropout, gamma o de regularización.

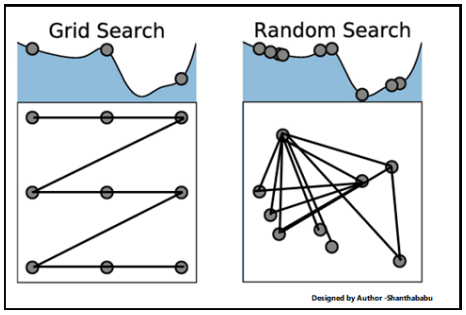


Figura 21 Comparación de GridSearch vs RandomSearch. Fuente: (Pandian, 2022)

4.4. Entrenamiento y validación del modelo

Una vez realizadas todas las configuraciones de los modelos, nuestro enfoque ha sido el realizar diversas pruebas con todas estas opciones, usando ambos modelos preentrenados y aplicando primero el modelo sin RandomizedSearchCV y posteriormente aplicándolo, y así para cada enfermedad. A continuación uno de los reportes generados al realizar el entrenamiento y validación de un modelo:

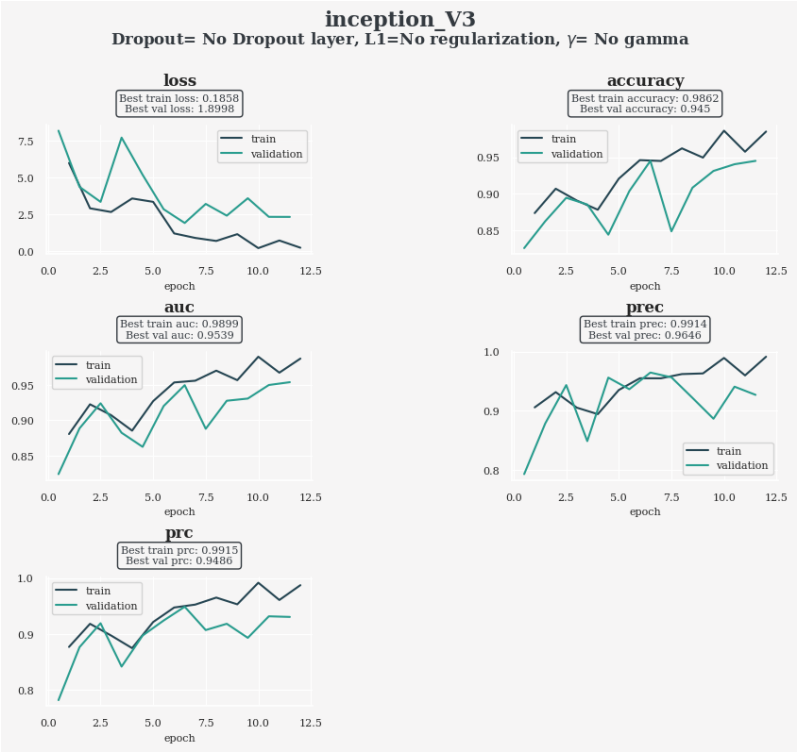
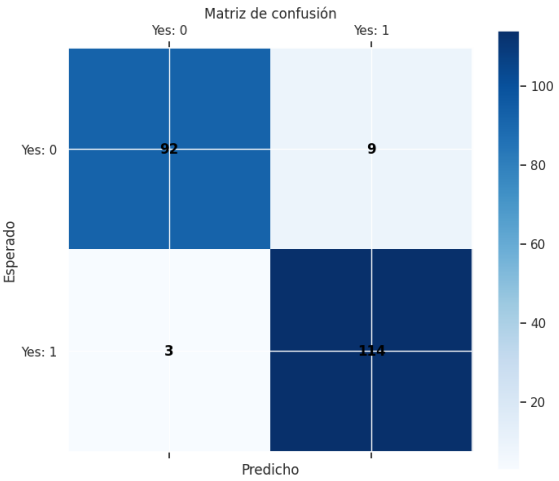


Figura 22 Ejemplo de resultados y métricas de entrenamiento de un modelo. Fuente: Elaboración propia

En este caso vemos el rendimiento calculado por el propio modelo para el caso de Ojos normales vs Ojos con cataratas, aplicando diferentes métricas como la curva AUC, el Accuracy o la Precision.

- AUC (Area Under the Curve): Es una métrica utilizada para evaluar la calidad del modelo en un problema de clasificación binaria. Representa el área bajo la curva ROC (Receiver Operating Characteristic) y proporciona una medida del rendimiento del modelo en términos de la capacidad de distinguir entre las clases positiva y negativa. Un valor de AUC cercano a 1 indica un mejor rendimiento, ya que significa que el modelo tiene una alta tasa de verdaderos positivos y una baja tasa de falsos positivos.
- PRC (Precision-Recall Curve): Es una métrica utilizada también en problemas de clasificación binaria. Representa la relación entre la precisión (precision) y la recuperación (recall) del modelo a medida que se varía el umbral de clasificación. La curva PRC proporciona información sobre cómo se equilibran la precisión y la recuperación en diferentes umbrales de clasificación. Un área bajo la curva PRC más grande indica un mejor rendimiento del modelo.
- Loss (Pérdida): La pérdida es una métrica que representa la discrepancia entre las predicciones del modelo y los valores reales del conjunto de datos. El objetivo del entrenamiento del modelo es minimizar esta pérdida. En modelos de deep learning, se utilizan diferentes funciones de pérdida dependiendo del tipo de problema, como la BinaryCrossEntropy para clasificación binaria. Un valor más bajo de pérdida indica un mejor ajuste del modelo a los datos.
- Accuracy (Precisión): La precisión es una métrica que representa la proporción de predicciones correctas realizadas por el modelo en relación con el número total de predicciones. Es una medida simple y común para evaluar el rendimiento de los modelos de clasificación. Sin embargo, en casos de conjuntos de datos desequilibrados, donde una clase tiene muchas más muestras que la otra, la precisión puede no ser suficiente para evaluar correctamente el modelo.
- Precision (Precisión): La precisión es una métrica que mide la proporción de predicciones positivas correctas realizadas por el modelo en relación con el número total de predicciones positivas. Es una medida útil para evaluar la capacidad del modelo de evitar falsos positivos. Una alta precisión indica que el modelo tiene una baja tasa de falsos positivos.

Posteriormente, nosotros aplicamos **nuestro propio cálculo del rendimiento** usando el reporte de clasificación (Classification Report), la matriz de confusión y aplicando nuestro propio valor de corte de lo que consideremos un acierto o no, usando el valor **cut_off_value**:




```
-----------------------------------
Classification Report
-----------------------------------
              precision    recall  f1-score   support

           0       0.97      0.91      0.94       101
           1       0.93      0.97      0.95       117

    accuracy                           0.94       218
   macro avg       0.95      0.94      0.94       218
weighted avg       0.95      0.94      0.94       218
```

Figura 23 Classification Report y Matriz de confusión obtenidos en cada prueba. Fuente: Elaboración propia

Al final del reporte podremos ver ejemplos de como nuestro modelo ha etiquetado cada imagen:

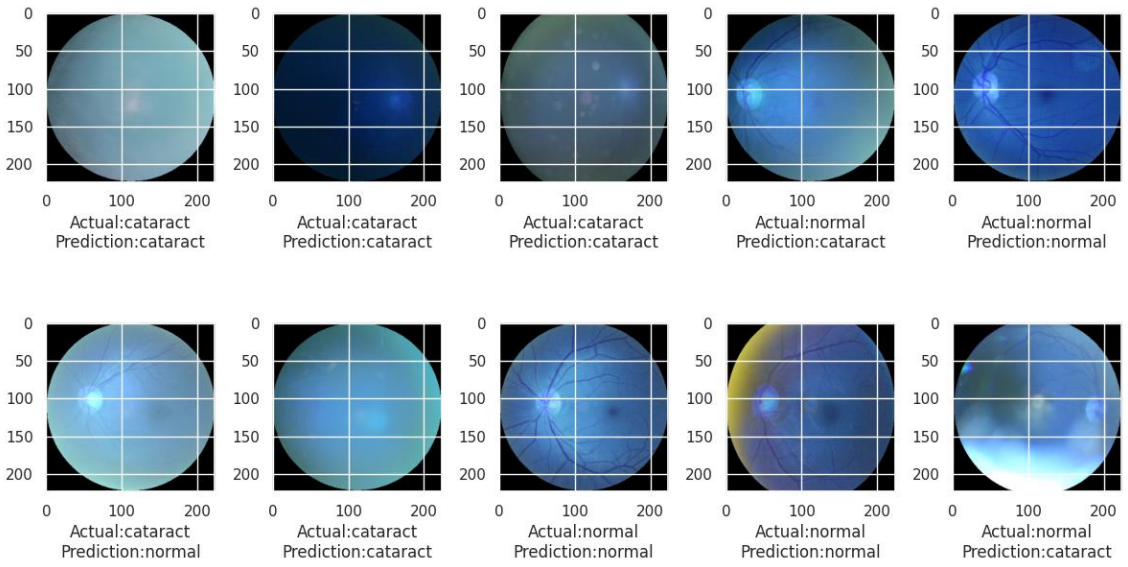


Figura 24 Resultados de predicción del modelo usando el conjunto de test. Fuente: Elaboración propia

4.5. Evaluación de resultados y conclusiones

Una vez realizadas todas las pruebas con las combinaciones, vemos la tabla de resultados para cada caso de clasificación. Debemos aclarar que los valores aquí recogidos tras nuestros propios cálculos de rendimiento, tal como describimos en la sección anterior:

Modelo preentrenado		Métricas / Configuración	Normal vs Resto	Normal vs Cataratas	Normal vs Diabetes	Normal vs Glaucoma
VGG19	No RScv	Accuracy	0,65	0,98	0,97	0,99
		Precision	0,66	0,98	0,97	0,99
		F1-Score	0,64	0,98	0,97	0,99
	Si RScv	Accuracy	-	0,96	0,95	0,99
		F1-Score	-	0,97	0,93	0,99
		Precision	-	0,96	0,97	0,99
		L1	-	No	No	No
		L2	-	Si	No	Si
		Gamma	-	No	No	No
		Dropout	-	0,3	0,3	0,3
InceptionV3	No RScv	Accuracy	0,56	0,9037	0,84	0,88
		Precision	0,59	0,90	0,87	0,91
		F1-score	0,54	0,90	0,83	0,88
	Si RScv	Accuracy	0,54	0,84	0,45	0,94
		F1-Score	0,55	0,84	0,32	0,94
		Precision	0,50	0,87	0,73	0,94
		L1	No	No	No	No
		L2	No	Si	Si	No
		Gamma	No	No	No	No
		Dropout (valor)	0,3	0,1	0,1	0,1

Analizando los resultados obtenidos, en general se observa un mejor rendimiento clasificatorio usando el modelo preentrenado VGG19 para todos los escenarios planteados. Usando la técnica de la búsqueda aleatoria por validación cruzada, notamos que se consigue una mejora a pesar de que se añaden capas de regularización.

Como conclusión final podemos decir que:

- Para todas las tareas de clasificación propuestas en este proyecto, el mejor modelo preentrenado es VGG19.
- Valorando el usando de la técnica RandomizedSearchCV para la búsqueda aleatoria de parámetros para nuestro modelo, podemos asegurar que no mejoramos en todas las pruebas realizadas a excepción de la prueba Normal vs Glaucoma, donde frente al prueba donde no usamos esta técnica. Aún así, el modelo que hace uso de VGG19 nos sigue dando un mejor rendimiento.

5. Problemas encontrados, recursos utilizados y retos para el futuro

En cuanto a los problemas encontrados, en primer momento realicé la instalación del entorno en un ordenador personal, con una Nvidia RTX3060Ti, la cual es una GPU dedicada y además procesador Intel i5 13th Generation y 16 GB de RAM, por lo que se supone es un ordenador más que apto para este tipo de tareas. Sin embargo, al avanzar en el proyecto, me empezó a dar fallos de apago de ordenador, desbordamiento de memoria RAM y demás problemas derivados de la demanda de recursos de este proyecto, por lo que, en mitad del proyecto decidí replicar el entorno de programación en Google Colab y hacer uso de los recursos en la nube. Aún así, seguía teniendo problemas similares en cuanto a limitación de la RAM necesaria para seguir modelando y entrenando mis modelos, así como también limitación de la memoria de la tarjeta gráfica.

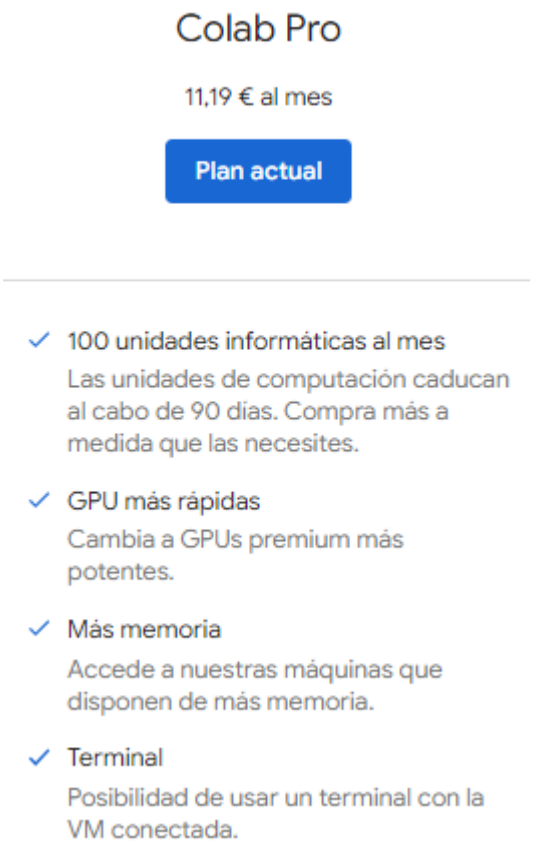


Figura 25 Plan usado de Google Colab para acometer el proyecto. Fuente: Elaboración propia

Finalmente, me decidí a aplicar al programa Colab Pro, en la que tengo diversas gráficas con una memoria más que suficiente (hasta 40 GB) para poder realizar este proyecto con garantías, así como también RAM, hasta 85 GB RAM. De esta manera evitaba cortes y caídas en el kernel de ejecución actual, así como el acumulado de datos de las variables y modelos, y la sobre carga de la memoria en la GPU.

Además, llegando a la recta final del proyecto, nos topamos con otro error, relacionado con la prueba Normal vs Resto (VGG19), donde en general habíamos conseguido realizarla con éxito a

pesar de obtener errores y warnings durante la ejecución, pero finalmente no hemos podido reejecutarla, obteniendo problemas relacionados con los tensores de la tarjeta gráfica.

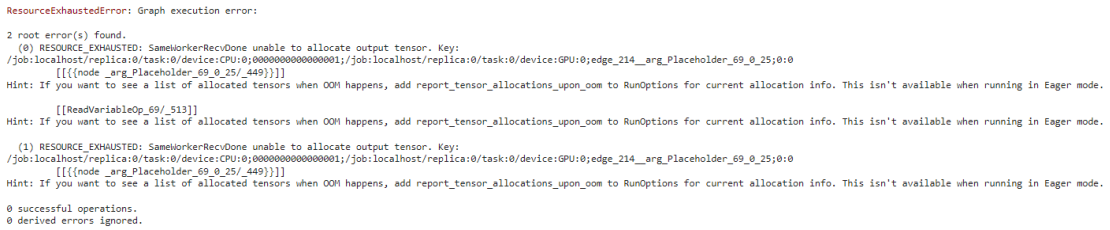


Figura 26 Error obtenido frecuentemente durante el entrenamiento por desbordamiento de recursos. Fuente: Elaboración propia.

5.1. Retos futuros

Como ideas de continuación de este proyecto se puede proponer el desarrollo de un modelo clasificador que consiga clasificar varias enfermedades a la vez en función del ojo que esté estudiando. De esta forma sería un modelo más aplicable al problema real de detección de las anomalías y a la vez de mejorar mejorar la eficiencia en el proceso, pues pasaríamos de 4 modelos para detectar 4 enfermedades a un modelo para detectarlas todas.

También se podría trabajar en el modelo desarrollado que clasifica entre los ojos catalogados como normales y los que tienen algún tipo de enfermedad, pues a pesar de contar con más muestras para entrenamiento, no se ha conseguido que este modelo en concreto rinda al nivel de los otros modelos desarrollados.

Por último, se puede seguir experimentando con el valor de corte **cutOff_value,** para seguir afinando los modelos y conseguir aún un mejor rendimiento en todos ellos, estableciendo un grado de seguridad aún mayor sobre lo que el modelo puede considerar un acierto y lo que no.

6. Código de desarrollo del proyecto

A continuación se indica una URL del repositorio donde se encuentra todo el código comentado, paquetes utilizados y en general todo el material desarrollo para la realización del proyecto.

Repositorio

7. Bibliografía

Dertat, A. (04 de Octubre de 2017). *Towardsdatascience*. Recuperado el 04 de Julio de 2023, de Towardsdatascience: https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

García, U. (Junio de 2019). *FutureLab*. Obtenido de FutureLab: https://futurelab.mx/redes%20neuronales/inteligencia%20artificial/2019/06/25/intro-a-redes-neuronales-pt-1/

Google Cloud Documentación. (s.f.). Recuperado el 04 de Julio de 2023, de https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=es-419

Google DeepMind. (2020). Obtenido de AlphaGo - The Movie | Full award-winning documentary: https://www.youtube.com/watch?v=WXuK6gekU1Y&ab_channel=GoogleDeepMind

Hsu, F.-H. (2022). *Behind Deep Blue: Building the Computer That Defeated the World Chess Champion.* Princeton University Press.

Keras. (2023). *Keras Documentation*. Obtenido de https://keras.io/

Matplotlib. (2023). *Matplotlib Documentation*. Obtenido de https://matplotlib.org/

McCorduck, P. (2004). *Machines Who Think: A personal inquiry into the history and prospects of artificial intelligence.* A K Peters Ltd.

Mittal, A. (12 de Octubre de 2019). *Medium*. Recuperado el 04 de Julio de 2023, de Medium: https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e

Mosquera, D. G. (1 de Febrero de 2018). *Medium*. Recuperado el 04 de Julio de 2023, de Medium: https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and-tensorflow-cb03cdcdba0f

Numpy. (2023). *Numpy Documentation*. Obtenido de https://numpy.org/

Pandas. (2023). *Pandas Documentation*. Obtenido de https://pandas.pydata.org/

Pandian, S. (22 de Febrero de 2022). *Analytics Vidhya*. Recuperado el 04 de Julio de 2023, de Analytics Vidhya: https://www.analyticsvidhya.com/blog/2022/02/a-comprehensive-guide-on-hyperparameter-tuning-and-its-techniques/

Patel, K. (8 de Septiembre de 2019). *Towardsdatascience.* Recuperado el 04 de Julio de 2023, de https://towardsdatascience.com/mnist-handwritten-digits-classification-using-a-convolutional-neural-network-cnn-af5fafbc35e9

Phi, M. (14 de Septiembre de 2018). *Towardsdatascience*. Recuperado el 04 de Julio de 2023, de https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

PyPi. (2023). *PyPi OpenCV Repository*. Obtenido de https://pypi.org/project/opencv-python/

Python. (2023). *Python OS package Documentation*. Obtenido de https://docs.python.org/3/library/os.html

Python. (2023). *Python Pathlib Repository*. Obtenido de https://docs.python.org/3/library/pathlib.html

Python. (2023). *Python Random Package Repository*. Obtenido de https://docs.python.org/es/3/library/random.html

R. Evtimov, M. F. (7 de February de 2020). *Humboldt-wi.github - Universidad de Berlín*. Recuperado el 04 de Julio de 2023, de https://humboldt-wi.github.io/blog/research/information_systems_1920/bert_blog_post/

Rahman, M. M. (Julio de 20). *ResearchGate*. Recuperado el 04 de Julio de 2023, de ResearchGate: https://www.researchgate.net/figure/Illustration-of-fine-tuned-VGG19-pre-trained-CNN-model_fig1_342815128

Scikit-learn. (2023). *Scikit-learn Documentation*. Obtenido de https://scikit-learn.org/

Seaborn. (2023). *Seaborn Pydata*. Obtenido de https://seaborn.pydata.org/

Stebner, A. (2020). *ResearchGate*. Obtenido de https://www.researchgate.net/figure/Common-activation-functions-in-artificial-neural-networks-NNs-that-introduce_fig7_341310767

Stuart J. Russell, N. (2016). *Artificial Intelligence: A modern approach.* Prentice Hall.

Tensorflow - Google. (2023). *Tensorflow Documentation*. Obtenido de https://www.tensorflow.org/?hl=es-419

Veen, F. v. (2016). *asimovinstitute.org*. Obtenido de asimovinstitute.org: https://miro.medium.com/v2/resize:fit:3000/1*cuTSPlTq0a_327iTPJyD-Q.png