Práctica 2.
Visión artificial y aprendizaje

Asignatura: *Sistemas inteligentes*
Alumno: **Georg Usin**
DNI/NIE: **X8174555X**

Vision artificial del T800

Índice

.....PARTE I
..... 3
Tarea A: Definir, utilizar y evaluar un MLP con Keras..... 3
Tarea B: Ajustar el valor del parámetro epochs..... 5
Tarea C: Ajustar el valor del parámetro batch_size..... 6
Tarea D: Probar diferentes funciones de activación..... 7
Tarea E: Ajustar el número de neuronas..... 8
Tarea F: Optimizar un MLP de dos o más capas..... 9
Tarea G: Definir, entrenar y evaluar un CNN sencillo con Keras..... 10
Las CNN (Convolutional Neural Network) son modelos diseñados para procesar estructuras espaciales, como imágenes. Al contrario que las MLP, las CNN explotan las propiedades espaciales y de correlación local en los datos, lo que las hace altamente eficientes y precisas en tareas de visión artificial..... 10
Observamos que el modelo con MaxPooling2D realiza mas épocas antes del sobre entrenamiento y que utilizando MaxPooling2D se obtiene un incremento considerable en precisión respecto al modelo básico. Aunque el tiempo de entrenamiento sea mayor, esto se debe a que el modelo con MaxPooling realiza muchas mas épocas..... 11
Tarea H: Ajustar el parámetro kernel_size de la CNN..... 12
Tarea I: Optimizar la arquitectura del modelo..... 13
.....PARTE II
..... 15
Tarea J: Creación de conjunto de prueba y evaluación de la generalización..... 15
..... 15
Tarea K: Resultados y experimentación..... 16
Bibliografía..... 19

PARTE I

Preprocesamiento de los datos

El preprocesamiento es un paso indispensable para preparar los datos de entrada y asegurar un rendimiento óptimo del modelo. Lo realizaremos en 3 pasos:

- Normalización: Escalamos los valores de los píxeles de [0, 255] a [0,1], con esto evitamos que los valores grandes dominen el cálculo de los gradientes y aceleramos la convergencia durante el entrenamiento.

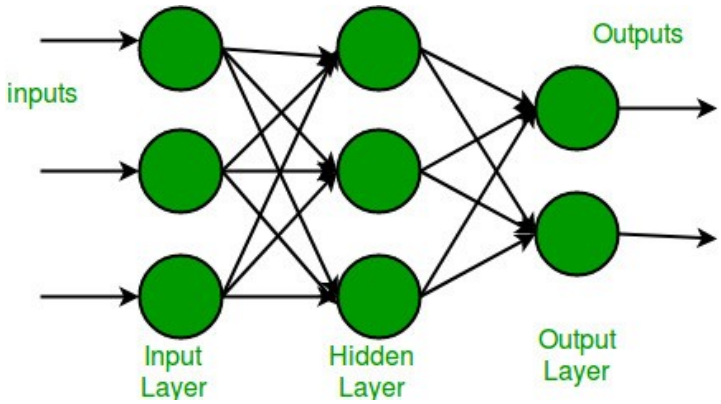
- Aplanado: Las imágenes de la base de datos CIFAR-10 tienen una estructura de 32x32x3, sin embargo, los MLP unicamente trabajan con vector unidimensionales. Por ello, “aplanamos” las imágenes y las transformamos en vectores de 3072 elementos.

- Codificación: Las etiquetas de las imágenes son representadas con valores enteros del 0 al 9. Es necesario transformarlos en vectores binarios para que la función de pérdida pueda procesarlas.

Tarea A: Definir, utilizar y evaluar un MLP con Keras

¿Que es un MLP y como aprende?

Un MLP (Multilayer Perceptron o Perceptrón multicapa) es un tipo de red neuronal artificial perteneciente a la categoría de redes feedforward. Estas redes están compuestas por capas de neuronas conectadas entre sí, donde cada neurona de una capa está conectada a todas las neuronas de la siguiente capa.



Como podemos observar en la imagen, los MLP están compuestos por 3 tipos de capas:

- Capa de Entrada: se encarga de recibir las características de entrada de los datos.
- Capas ocultas: en estas capas es donde se realiza el aprendizaje de patrones complejos.
- Capa de Salida: convierte las salidas en probabilidades, facilitando la predicción.

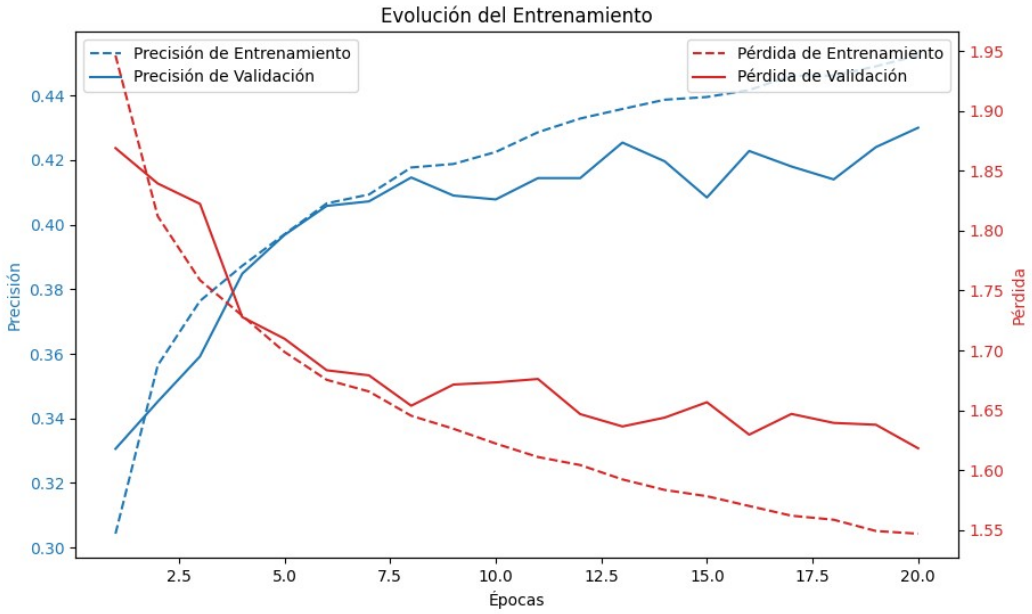
El aprendizaje de un MLP se basa en ajustar los pesos y sesgos de las “neuronas” para minimizar el error entre las predicciones del modelo y las etiquetas verdaderas. Este proceso incluye 2 procesos:

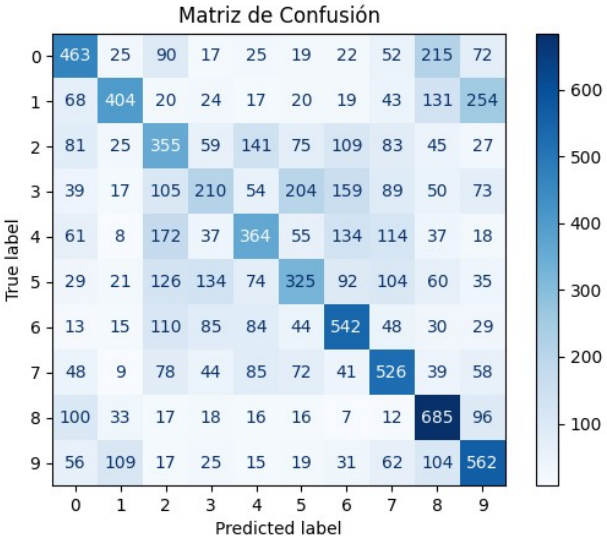
-Propagación hacia delante (Forward Propagation): las entradas atraviesan la red neuronal, capa por capa y producen una salida final. Una vez producida la salida, esta se compara con la etiqueta verdadera utilizando una función de pérdida que mide el error.

-Retro-propagación (Backpropagation): calcula los gradientes de la función de pérdida respecto a los pesos de la red y utiliza el algoritmo de descenso de gradiente (en nuestro caso utilizaremos Adam) para actualizar los pesos y reducir el error.

Este proceso se repite durante múltiples épocas, pasando por todo el conjunto de entrenamiento varias veces.

Resultados tras entrenar la red con 20 épocas.





Tarea B: Ajustar el valor del parámetro epochs

Sobre entrenamiento

El sobre entrenamiento ocurre cuando un modelo se ajusta demasiado a los datos de entrenamiento, lo que reduce su capacidad para generalizar datos nuevos y reduce el rendimiento en el conjunto de prueba. Esto lo detectamos cuando:

- La pérdida de validación val_loss comienza a aumentar.

- La precisión de validación val_accuaracy deja de mejorar o comienza a disminuir.

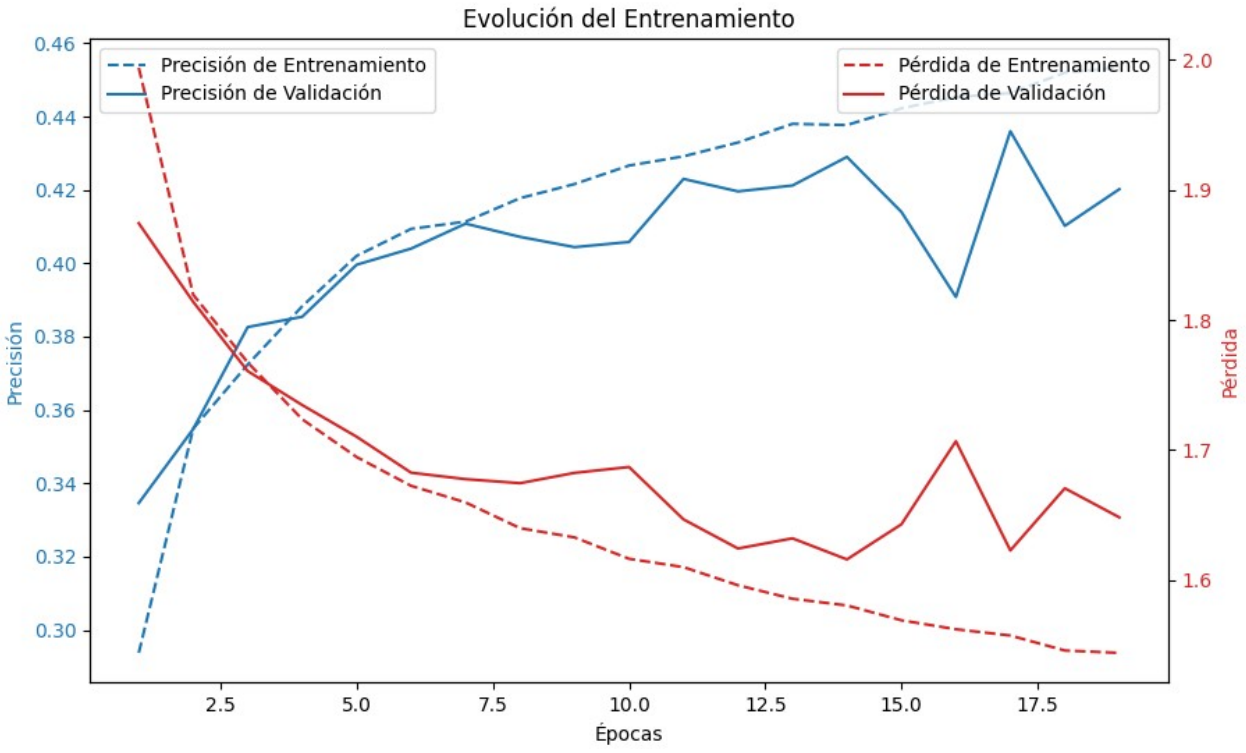
Que es el numero de épocas en MLP?

Es un parámetro que define cuántas veces pasa el modelo por el conjunto de datos completo durante el proceso de entrenamiento.

Cuando hay pocas épocas (Subentrenamiento), el modelo no tiene tiempo suficiente para aprender patrones complejos de los datos y como resultado se obtiene una baja precisión tanto en entrenamiento como en validación.

Por otro lado, cuando hay demasiadas épocas (Sobreentrenamiento), el modelo aprende en exceso los patrones específicos de los datos del entrenamiento, incluso fallos de imagen o detalles irrelevantes, como resultado se obtiene una alta precisión en entrenamiento pero baja precisión en validación.

Ejecutemos un entrenamiento con la función entrenarConDetección que finaliza el entrenamiento cuando el valor val_loss o val_accuaracy no mejora tras 5 épocas. Veamos en que época finaliza para saber cuando empieza a sobre entrenarse.



```
Epoch 19: early stopping
Restoring model weights from the end of the best epoch: 14.
Restoring model weights from the end of the best epoch: 17.
Pérdida en test: 1.6238
Tasa de acierto en test: 0.4274
```

Se produce sobre entrenamiento a partir de la época 12, por lo tanto no es recomendable realizar entrenamientos de mas de 12 épocas.

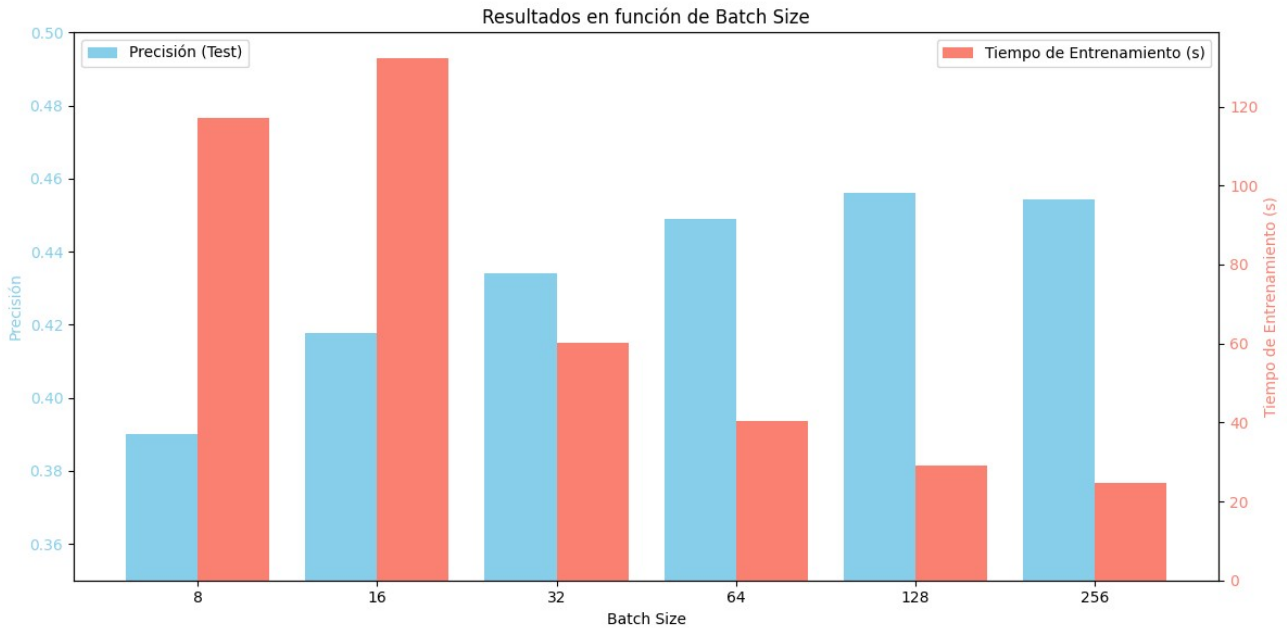
Tarea C: Ajustar el valor del parámetro batch_size

El parámetro batch_size controla el numero de muestras que procesa el modelo antes de actualizar los pesos de las neuronas. La base de datos de Cifar 10 contiene 50.000 imágenes de entrenamiento, si tenemos un batch_size de 32, el modelo divide los datos en lotes de 32 imágenes cada uno y actualiza 50.000/32 = 1,562 veces los pesos por época.

Valores pequeños de batch_size (8 o 16), supondrán un aumento de actualizaciones y una mejor cobertura del espacio de búsqueda, pero provoca un entrenamiento más lento.

Valores grandes de batch_size (64+), suponen una reducción en el numero de actualizaciones y un entrenamiento mas veloz, pero puede ser menos preciso en problemas con alta variabilidad.

Pongamos a prueba los valores 8, 16, 32, 64, 128 y 256 con 100 épocas y EarlyStopping.



Observamos que en el batch_size = 8 al graduarse los gradientes basándose en pocos ejemplos por iteración, se obtiene una menor precisión debido a una alta variabilidad en los gradientes.

También se observa que se cumple lo antes explicado, cuanto mayores sea el batch_size menor será el tiempo de entrenamiento.

Por último, el batch_size = 128 es el mejor de todos ellos, pues es el que mayor precisión tiene y la diferencia entre el tiempo de entrenamiento de este y el de 256 es muy pequeña. Por lo tanto, como lo que buscamos es la mejor precisión posible, nos quedamos con el batch_size de 128.

Tarea D: Probar diferentes funciones de activación

Las funciones de activación introducen no linealidad en un modelo MLP y controlan cómo se activan la neuronas. Analicemos algunas:

sigmoid: Rango (0, 1), es común en capas de salida para tareas de clasificación binaria. Sin embargo, satura los valores extremos y puede ser lenta para converger en redes profundas.

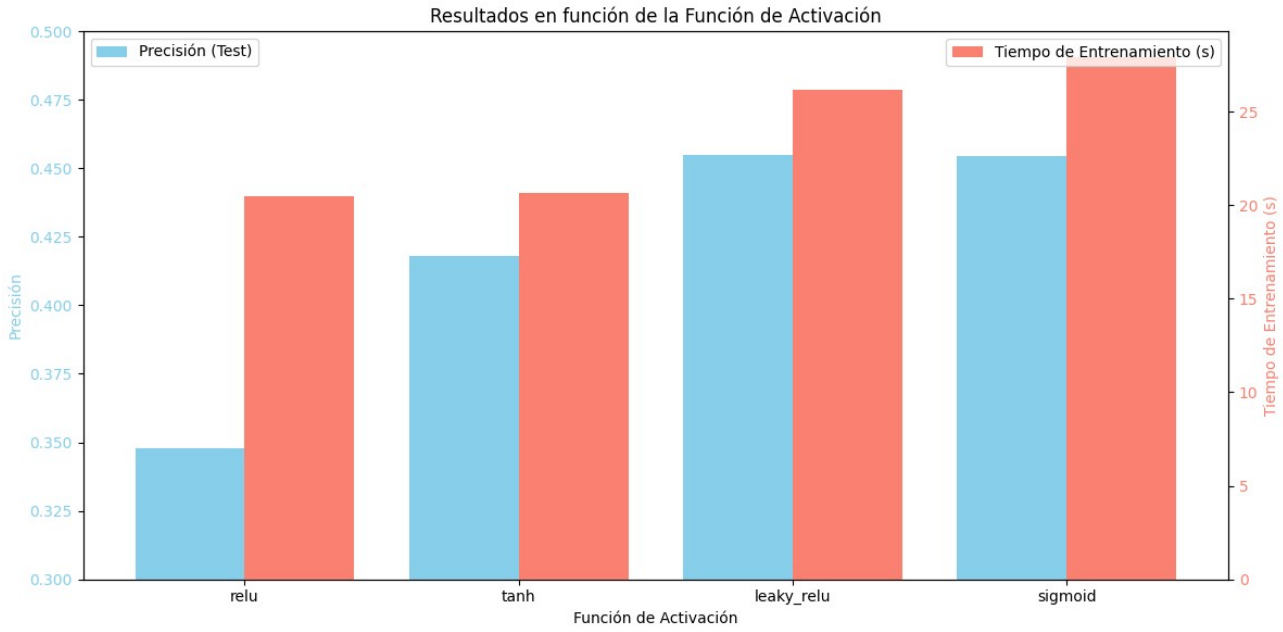
relu: Rango [0, ∞), evita la saturación y acelera la convergencia , puede provocar el problema de las neuronas muertas cuando los valores negativos se clipean a 0

leaky_relu: rango (-∞, ∞), similar a relu, pero permite un gradiente pequeño para valores negativos que mejora el problema de las neuronas muertas.

tanh: rango (-1, 1), similar a sigmoid, pero centrada en 0, lo que puede facilitar el aprendizaje en algunas arquitecturas.

softmax: rango (0, 1), pero devuleve una distribución de probabilidad para todas las clases. Se usa en la capa de salida en tareas de clasificación multiclase.

Probaremos relu, leaky_relu, tanh y sigmoid en las capas intermedias y softmax en la capa de salida. 100 épocas con EarlyStopping y batch_size = 128



La mejor función es leaky_relu, en este apartado he realizado varios entrenamientos y el ganador siempre estaba entre sigmoid y leaky_relu, lo que estaba claro es que relu era la peor de las cuatro, y tanh aun siendo mejor que relu (tampoco es difícil) no le llegaba ni a los pies a las otras dos. Por lo tanto realicé una prueba y de 20 entrenamientos 13 los ganó leaky_relu, por lo tanto, es la mejor función de activación de entre estas 4 opciones.

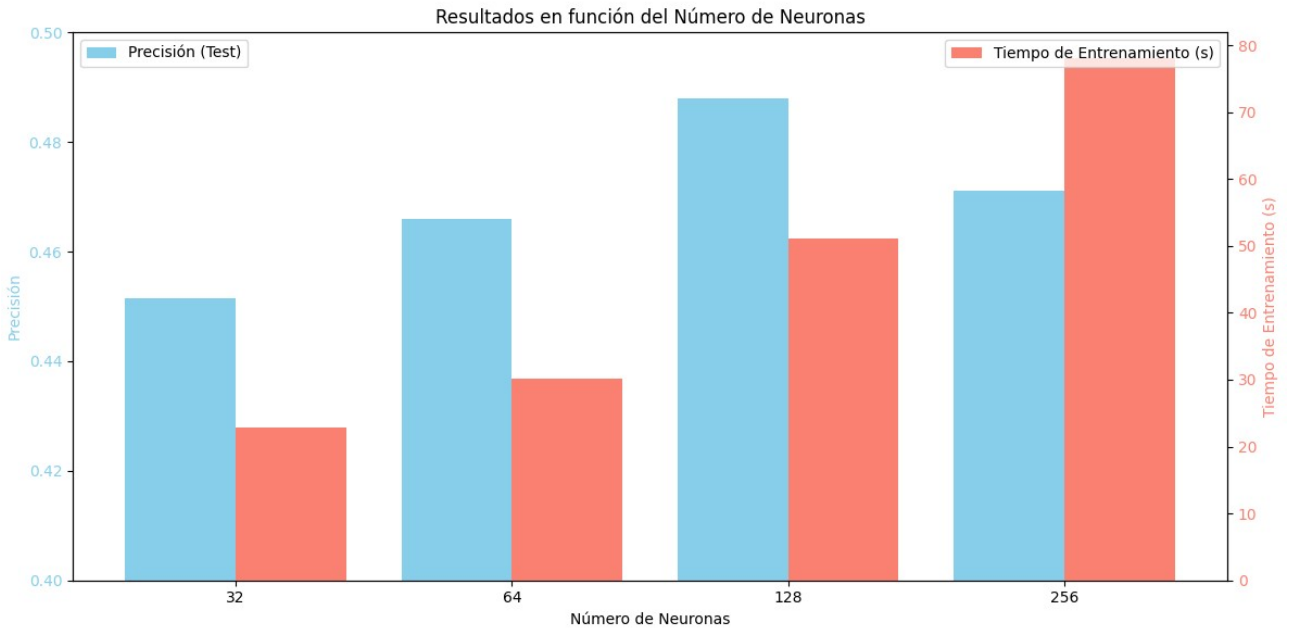
Tarea E: Ajustar el número de neuronas

Primero, veamos que sucede al aumentar o reducir el número de neuronas de la capa oculta.

Al aumentar el número de neuronas se obtiene una mayor capacidad de aprendizaje y un mejor rendimiento ante tareas completas. Sin embargo, el riesgo de sobre ajuste, el tiempo de entrenamiento y la complejidad computacional son mayores.

Al reducir el numero de neuronas se consigue un menor riesgo de sobre ajuste y un menor tiempo de entrenamiento. Pero, tendrá una capacidad limitada de aprendizaje y un peor rendimiento en las tareas complejas. Es decir, obtenemos todo lo contrario que al aumentar el numero de neuronas.

Probaremos con 32, 64, 128 y 256 neuronas en la capa oculta. 100 épocas con EarlyStopping, batch_size = 128 y leaky_relu en la capa oculta y softmax en la capa de salida.

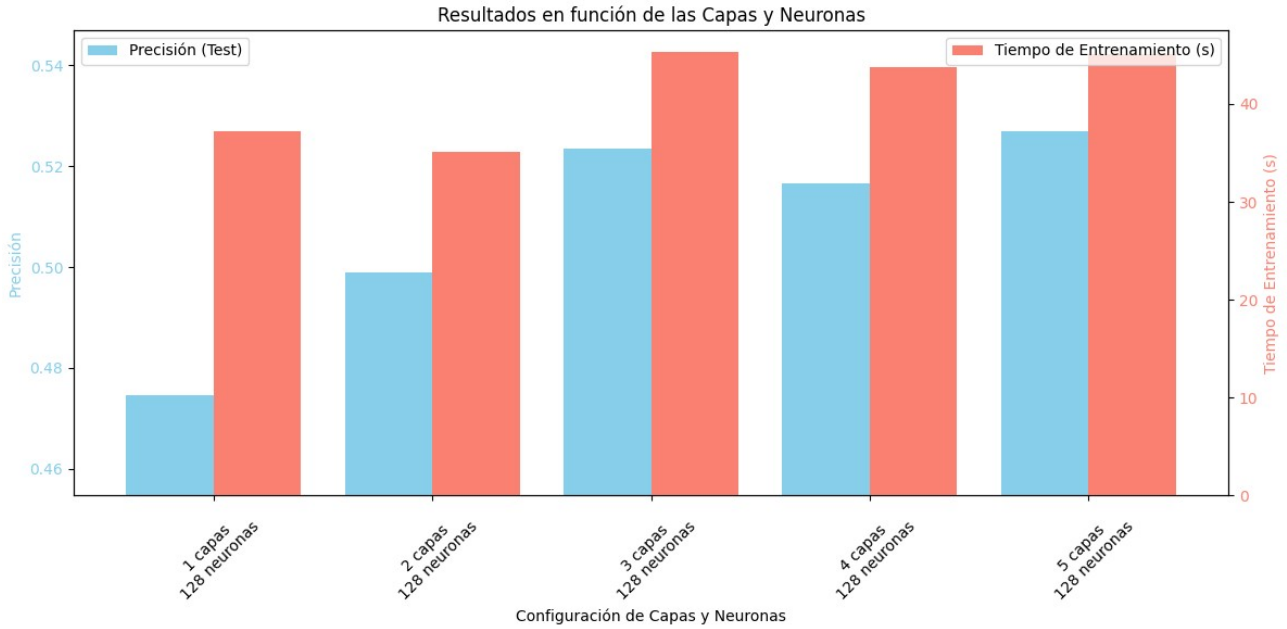


Observamos que la mejor opción en cuanto a precisión es 128 neuronas en la capa oculta.

Tarea F: Optimizar un MLP de dos o más capas

Cuando añadimos más capas ocultas a un MLP, le damos la capacidad de aprender representaciones más complejas de los datos, mejorando así la capacidad de aprendizaje y la precisión en tareas complejas. Como todo en la vida, nada es perfecto y al aumentar el numero de capas ocultas también se aumenta el riesgo de sobre ajuste, el tiempo de entrenamiento y el la complejidad computacional.

Probaremos con 1, 2 ,3, 4 y 5 capas ocultas. 100 épocas con EarlyStopping, batch_size = 128, leaky_relu en la capa oculta y softmax en la capa de salida y 128 neuronas.

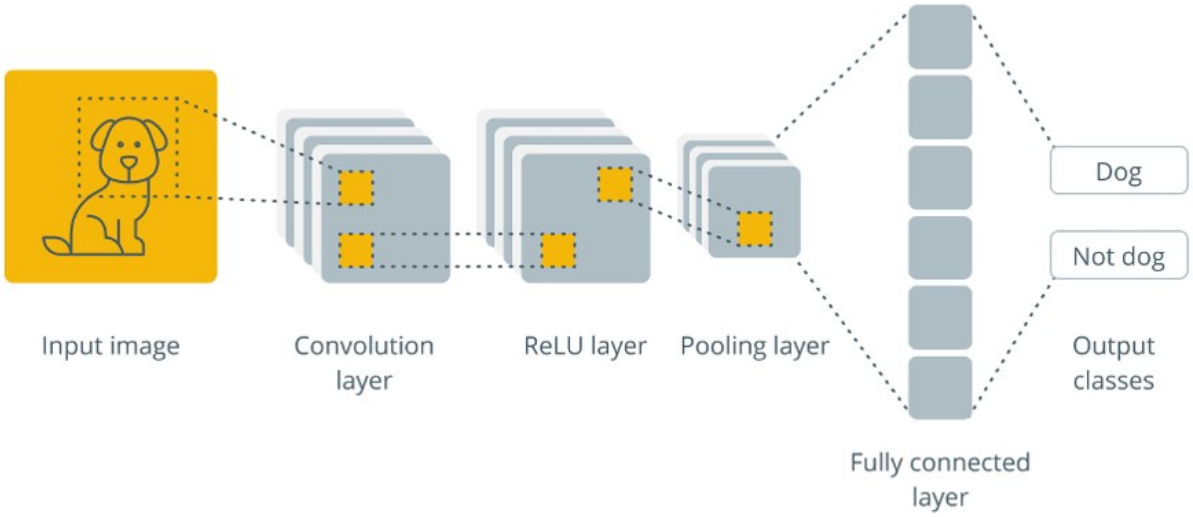


Vemos que el modelo con 5 capas es el que mayor precisión y tiempo de entrenamiento tiene. La mejora en precisión no es demasiado notable, pero la diferencia de tiempo de entrenamiento tampoco lo es. Por ello nos quedamos con el modelo de 5 capas.

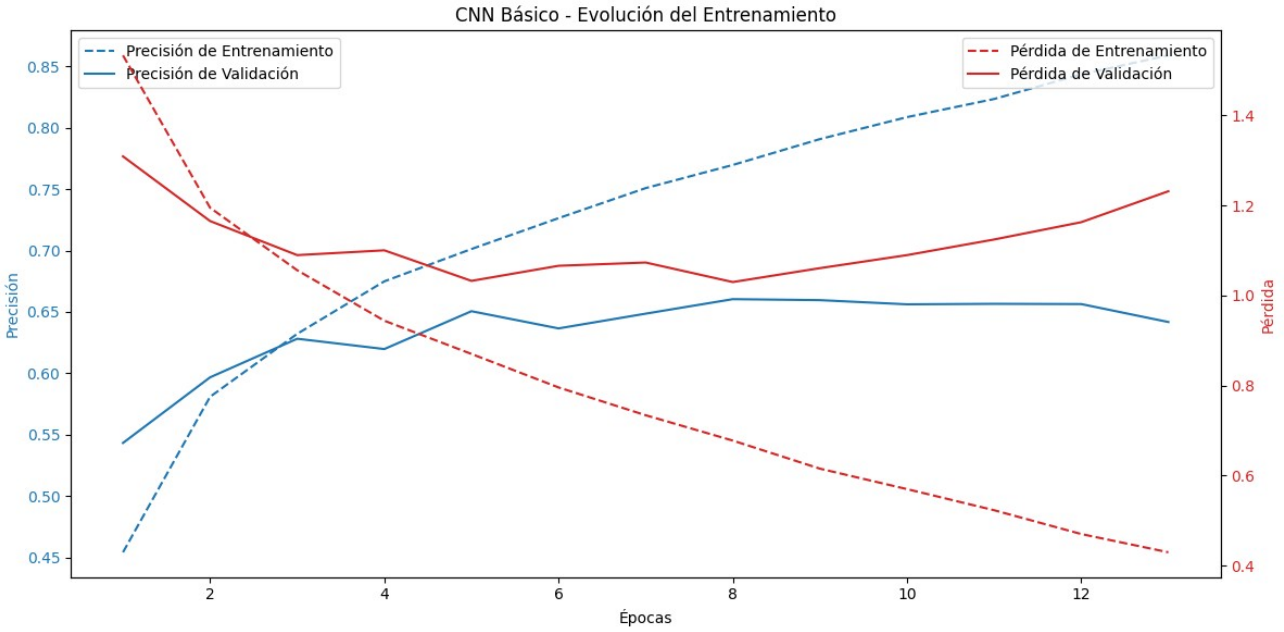
Recapitulando: utilizaremos 5 capas ocultas, 128 neuronas, el valor batch_size será 128 y utilizaremos leaky_relu para las capas ocultas y softmax para la capa de salida.

Tarea G: Definir, entrenar y evaluar un CNN sencillo con Keras

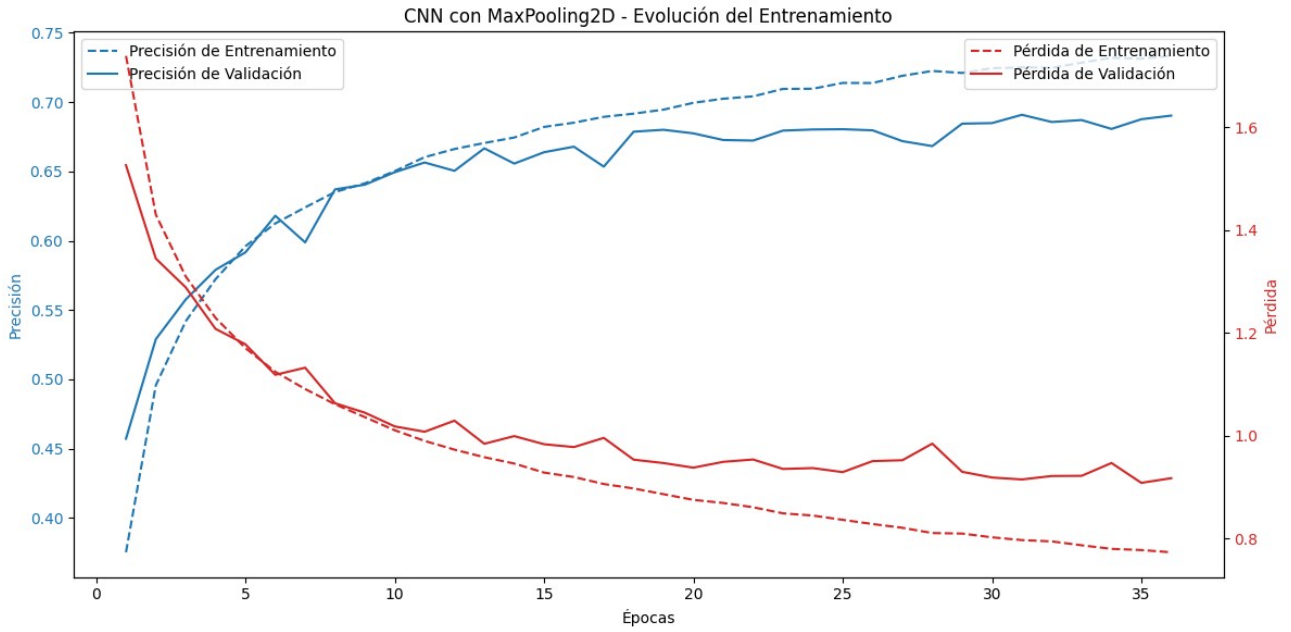
Las CNN (Convolutional Neural Network) son modelos diseñados para procesar estructuras espaciales, como imágenes. Al contrario que las MLP, las CNN explotan las propiedades espaciales y de correlación local en los datos, lo que las hace altamente eficientes y precisas en tareas de visión artificial.



Capas Conv2D: son el núcleo de una CNN, estas capas aplican filtros sobre la entrada para extraer características relevantes, como bordes, texturas o incluso formas. Generan mapas de características que destacan patrones específicos en diferentes partes de la imágenes, como las orejas en punta de los gatos o la forma de la cabeza de un caballo. En la imagen de arriba se ve claramente, es la Convolution Layer.



Capas MaxPooling2D: estas capas reducen las dimensiones espaciales de los mapas de características generados por las Conv2D, esto ayuda al modelo a simplificar las representaciones aprendidas, conservar la información relevante y mejorar la eficiencia.




```
Resultados Comparativos:
Modelo Básico - Precisión en test: 0.6498, Tiempo: 219.67 segundos
Modelo con MaxPooling2D - Precisión en test: 0.6795, Tiempo: 315.97 segundos
```

Observamos que el modelo con MaxPooling2D realiza mas épocas antes del sobre entrenamiento y que utilizando MaxPooling2D se obtiene un incremento considerable en precisión respecto al modelo básico. Aunque el tiempo de entrenamiento sea mayor, esto se debe a que el modelo con MaxPooling realiza muchas mas épocas.

MLP es mas sencillo y mas rápido, pero es menos preciso que CNN

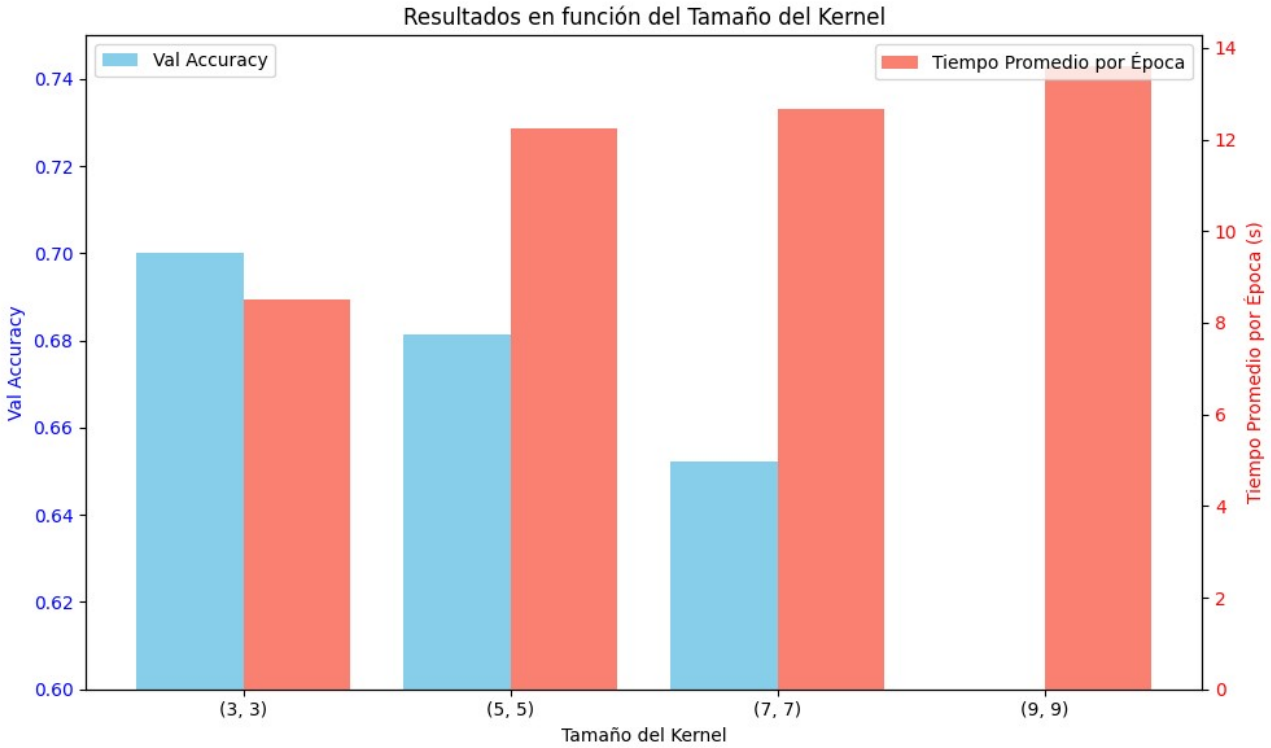
Tarea H: Ajustar el parámetro kernel_size de la CNN

El kernel size (tamaño del filtro) en una CNN define el área o ventana que se desplaza sobre la imagen para detectar características. Por ejemplo, un kernel de 3x3 examina bloques de 3 píxeles de altura y 3 píxeles de ancho.

Los kernels pequeños como 3x3 o 5x5 detectan características pequeñas y detalles finos, mientras que los kernels más grandes como 7x7 o 9x9 detectan características más grandes y globales.

Sin embargo, los kernels grandes requieren más cálculos, ya que cada filtro analiza más píxeles en cada paso. Por lo tanto, kernels grandes tienen un mayor costo computacional.

Los kernels pequeños son el estándar porque ofrecen una mejor relación precisión-eficiencia, mientras que los kernels grandes se utilizan para analizar imágenes de alta resolución.



En la gráfica observamos que el mejor kernel size es 3x3, ya que las imágenes de CIFAR-10 que utilizamos tienen una resolución de 32x32 píxeles.

Tarea I: Optimizar la arquitectura del modelo

Que podría mejorar la precisión de una CNN?

- Aumentar el número de capar Conv2D permite que el modelo aprenda características más complejas y abstractas, añadiremos 3 o más capas Conv2D, con un número creciente de filtros.

- MaxPooling y Reducción de Dimensionalidad, el pooling reduce la dimensión de la imagen, manteniendo las características más importantes, previene el sobreajueste, añadiremos MaxPooling2D después de cada capa Conv2D.

- BatchNormalization, acelera el entrenamiento y reduce el riesgo de quedar atrapado en mínimos locales, aumenta la estabilidad de la red, añadiremos BatchNormalization despues de Conv2D.

- Dropout, durante el entrenamiento, apaga aleatoriamente neuronas, lo que fuerza al modelo a aprender de manerma mas robusto y reduce sobreajuste, añadiremos Dropout despues de capas Dense o Flatten.

- Aumentar Filtros en las capas Conv2D, más filtros permiten que el modelo capture diferentes tipos de características en la misma imagen, probaremos 32, 64, 128 y 256 filtros.

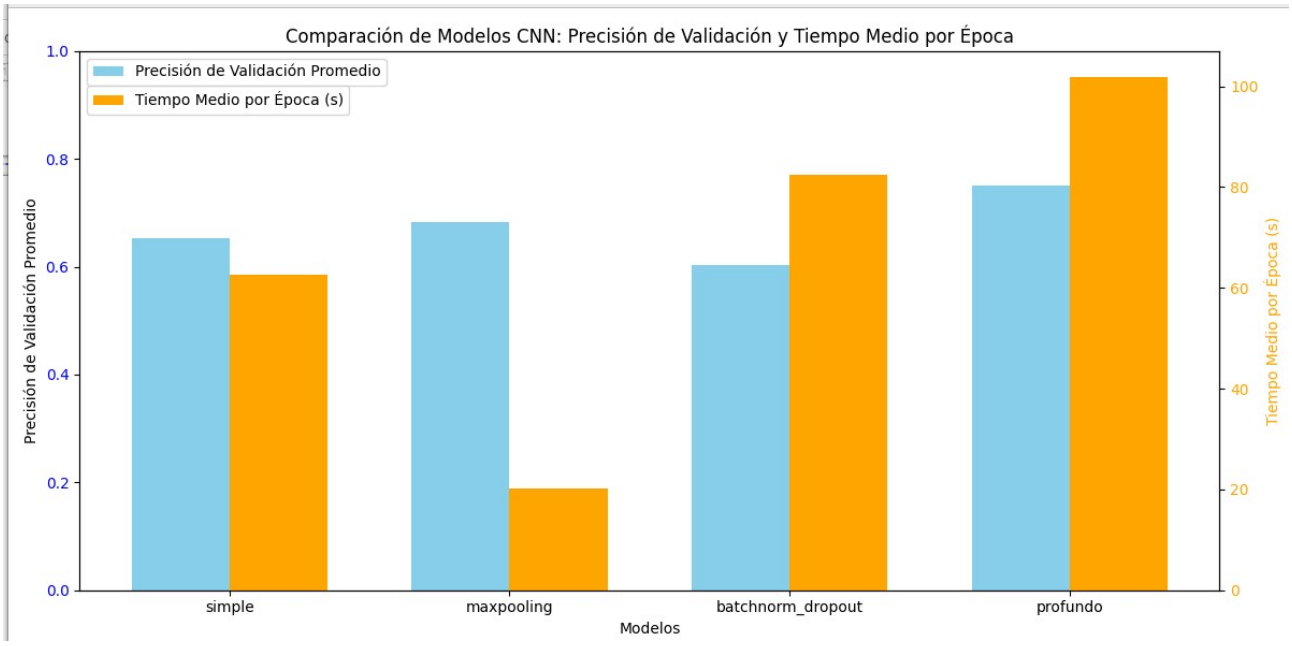
Modelos utilizados;

Modelo 'simple': Utiliza 2 capas Conv2D con 32 y 64 filtros. No utiliza MaxPooling, BatchNormalization ni Dropout.

Modelo 'maxpooling': Añade capas de MaxPooling2D para reducir la dimensionalidad después de cada Conv2D.

Modelo 'batchnorm_dropout': Usa BatchNormalization después de las capas Conv2D y Dropout para regularizar el modelo y prevenir el sobreajuste.

Modelo 'profundo': Tiene más capas Conv2D, aumenta progresivamente los filtros (32, 64, 128) y aplica MaxPooling y Dropout para regularizar el modelo y mejorar su capacidad de generalización.

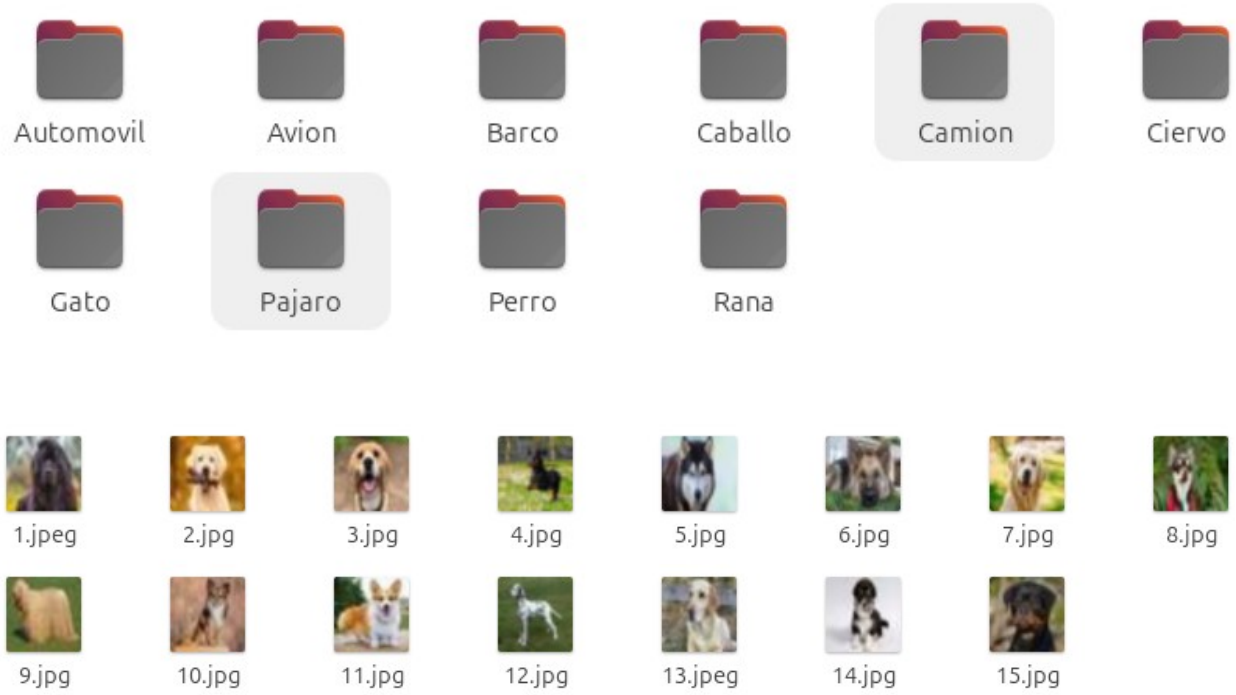


En mi caso, considero que un tiempo de entrenamiento es excesivo si supera los 5 minutos, lo que hace que el modelo **maxpooling** sea la mejor opción para mis necesidades actuales. Aunque tiene menor precisión en comparación con el modelo **profundo**, el tiempo medio por época es significativamente más bajo. Este ahorro de tiempo es crucial dado que mi hardware no tiene la capacidad necesaria para ejecutar modelos más complejos de manera eficiente. El hardware de este ordenador es tan deplorable que no mueve ni el Skyrim que es del 2011.

Si tuviera acceso a un equipo con mejor rendimiento, podría considerar el modelo profundo como una opción viable, ya que se ejecutaría más rápido y ofrecería una mayor precisión. Sin embargo, dadas mis limitaciones actuales, la selección de **maxpooling** es una decisión equilibrada entre eficiencia y desempeño, priorizando el tiempo de entrenamiento sin comprometer excesivamente la calidad.

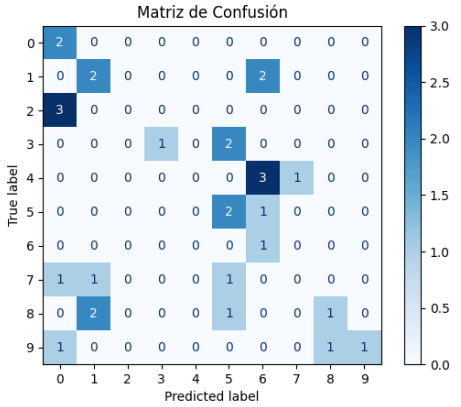
PARTE II

Tarea J: Creación de conjunto de prueba y evaluación de la generalización

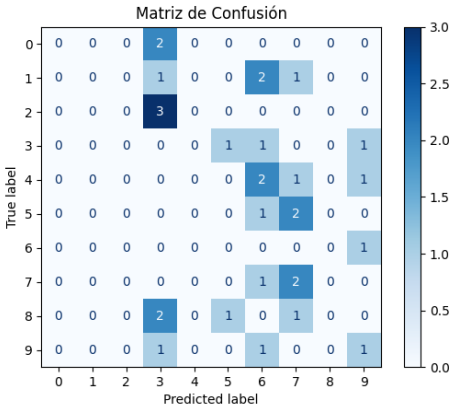


Tarea K: Resultados y experimentación

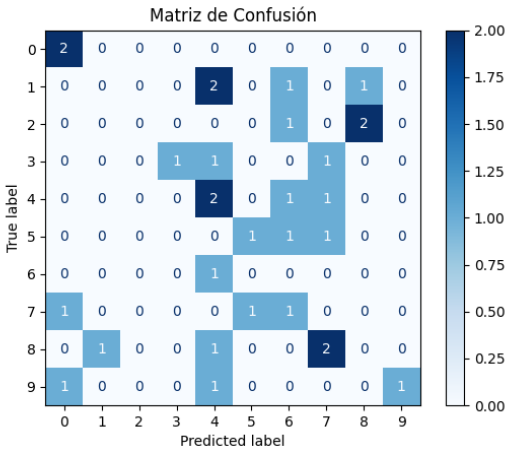
capas_ocultas=(128, 64), activacion='relu', optimizador='adam', lr=0.001, batch_size=32, kernel_size=(3, 3), epochs=100



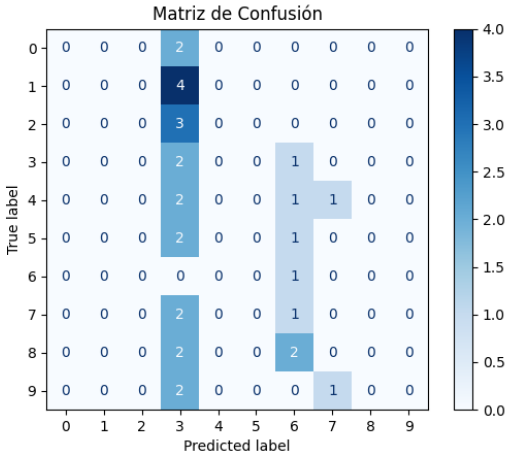
capas_ocultas=(128, 32), activacion='relu', optimizador='adam', lr=0.001, batch_size=16, kernel_size=(3, 3), epochs=100



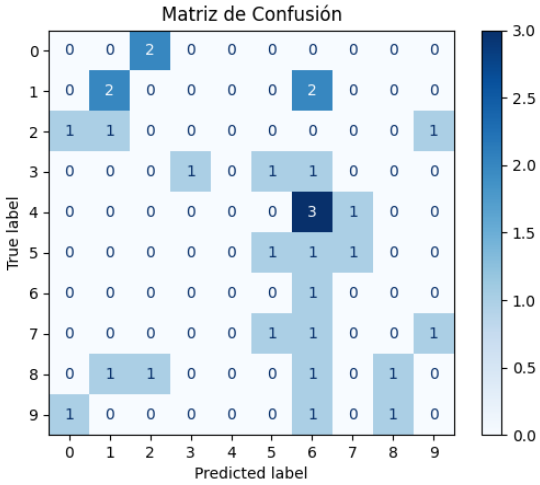
capas_ocultas=(128, 64), activacion='leaky_relu', optimizador='adam', lr=0.001, batch_size=32, kernel_size=(3, 3), epochs=100



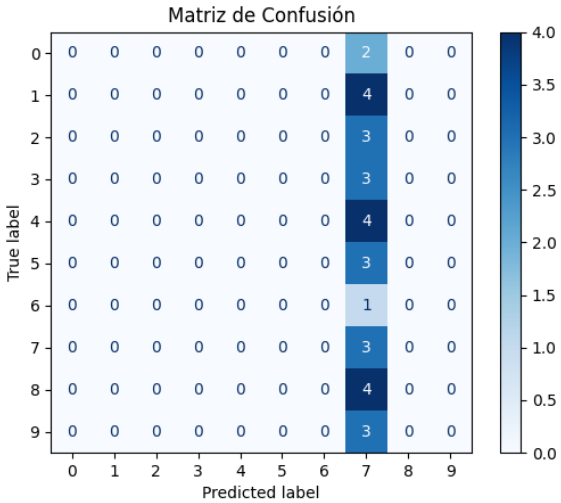
capas_ocultas=(128, 64), activacion='leaky_relu', optimizador='sgd', lr=0.001, batch_size=32, kernel_size=(3, 3), epochs=100



capas_ocultas=(128, 64), activacion='leaky_relu', optimizador='adam', lr=0.001, batch_size=128, kernel_size=(3, 3), epochs=100



capas_ocultas=(128, 64), activacion='leaky_relu', optimizador='adam’, lr=0.01, batch_size=128, kernel_size=(3, 3), epochs=100



Capas ocultas	Activacion	Optimizador	Learning Rate	Batch size	Kernel_size	Precision	Tiempo por epoca (s)
128,64	relu	adam	0,001	32	(3, 3)	0,3333	0,14
128,32	relu	adam	0,001	16	(3, 3)	0,1667	0,18
128,64	leaky_relu	adam	0,001	32	(3, 3)	0,3333	0,14
128,64	leaky_relu	sgd	0,001	32	(3, 3)	0,0833	0,16
128,64	leaky_relu	adam	0,001	128	(3, 3)	0,5	0,13
128,64	leaky_relu	adam	0,01	128	(3, 3)	0,25	0,2

Los mejores resultados se obtienen con la configuración: capas_ocultas=(128, 64), activacion='leaky_relu', optimizador='adam’, lr=0.001, batch_size=128, kernel_size=(3, 3), epochs=100

Bibliografía

Configuracion inicial MLP: **https://keras.io/examples/vision/mlp_image_classification/**

Fuente de la imagen de CNN que lo explica: **https://blog.stackademic.com/deep-learning-convolutional-neural-networks-cnns-ce48fb7fdf57**

Funciones de activación: **https://keras.io/api/layers/activation_layers/**

Gran ayuda para generar las gráficas: **https://chatgpt.com/**