PRÁCTICA 2

Jesús Ballesteros Navarro – Sistemas Inteligentes

Jesús Ballesteros Navarro
50387249E

Índice:

Nota: Para toda la práctica los datos ya están promediados y unificados, es decir, por ejemplo, en MLP2 para patience = 7 se ha realizado un total de 5 entrenamientos con ese parámetro y se ha promediado para representarlo en las gráficas de esta práctica. Se ha realizado un total de 5 iteraciones para estos promedios.

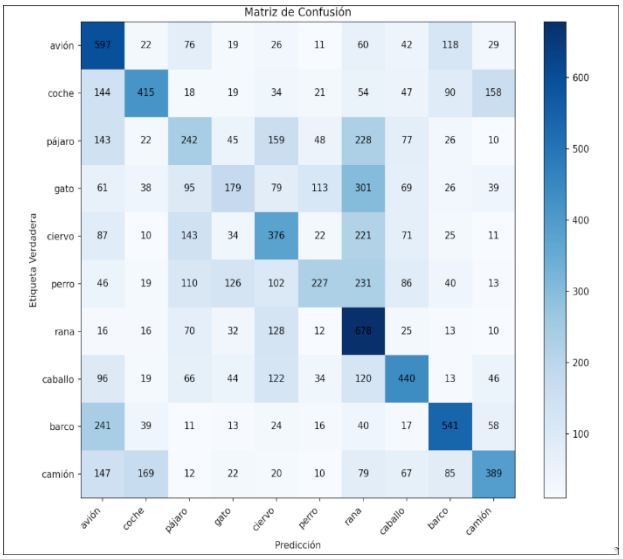
Jesús Ballesteros Navarro
50387249E

MLP 1:

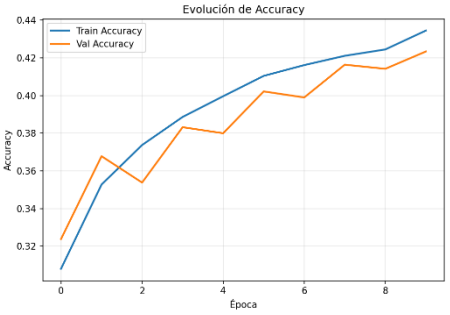
Realizando la tarea mlp1 y dibujando la gráfica lineal y matriz de confusión obtenemos los siguientes datos con este modelo simple.

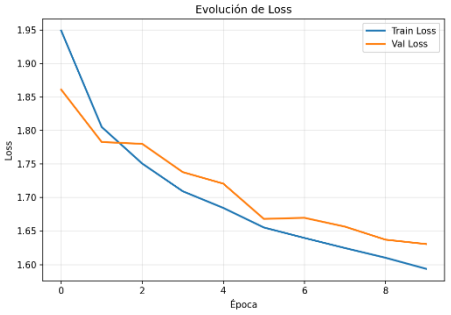
Gráficas obtenidas:

Matriz de confusión MLP1:



Gráfica línea:





Conclusión:

- Test Loss: 1.6169
- Test Accuracy: 0.4219

Como se observa en los datos, este modelo de MLP1 tiene una capacidad de acierto del 42.19%, lo que representa una mejora significativa respecto a la probabilidad aleatoria del 10%. Si bien la tasa de acierto no supera el 50%, demuestra que el modelo ha aprendido patrones útiles de las imágenes.

Por otra parte, el Test Loss de 1.6169 refleja el grado de confianza del modelo en sus predicciones. Este valor se sitúa entre un modelo aleatorio (~2.30) y un modelo bien entrenado (<1.0), indicando que hay margen de mejora, lo que se comprueba en las próximas tareas del MLP.

MLP2:

En el MLP2 se experimentará con diferentes patience para el EarlyStopping en el modelo de entrenamiento para comprobar cual es el número de épocas óptimo que debe permitir entre monitoreos para detectar cuando parar el entrenamiento.

Para este apartado voy a utilizar la clase de Keras EarlyStopping la cual que monitoriza val_loss durante el entrenamiento y detiene automáticamente cuando deja de mejorar.

Más información en su sección de la página de Keras, dentro de Callbacks API en el link que se encuentra detallado en la bibliografía sobre las apis disponibles en Keras.

En mi caso voy a probar con 5 valores para patience.

- Patience = 3: Permite evaluar si el modelo necesita más tiempo para encontrar mejoras o si converge rápidamente.

- Patience=5: Valor recomendado por defecto en la literatura y documentación de Keras. Proporciona un equilibrio entre evitar el sobreentrenamiento y permitir suficiente tiempo para la convergencia.

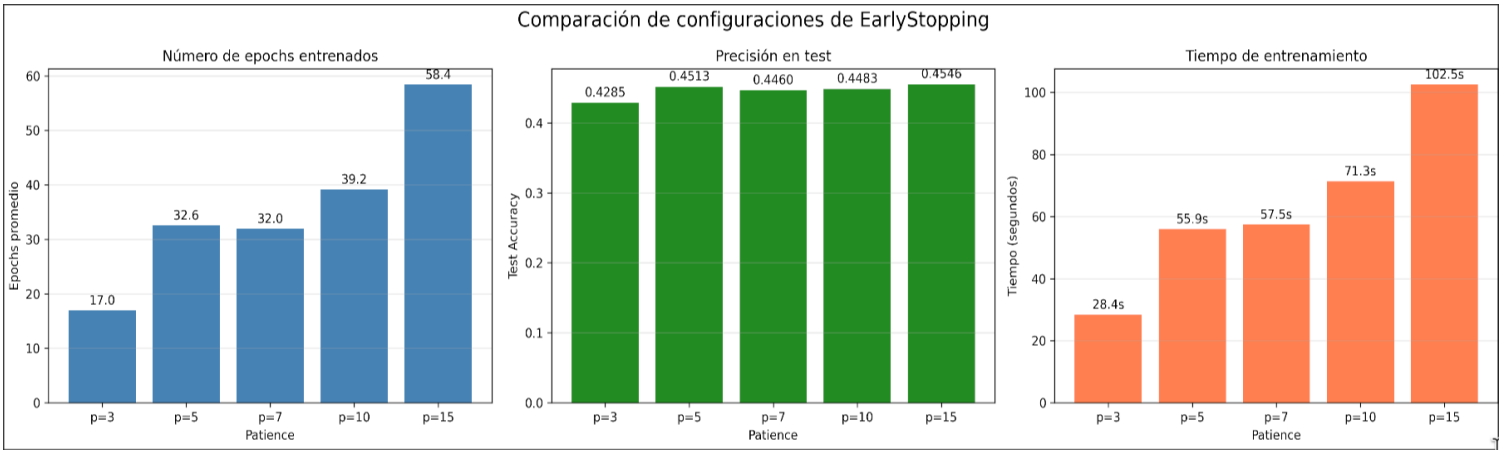
- Patience=7: Valor intermedio que permite explorar la sensibilidad del parámetro.

- Patience=10: Configuración más tolerante que da al modelo más oportunidades de recuperación. Útil para detectar si el modelo experimenta fluctuaciones temporales en val_loss que posteriormente mejoran.

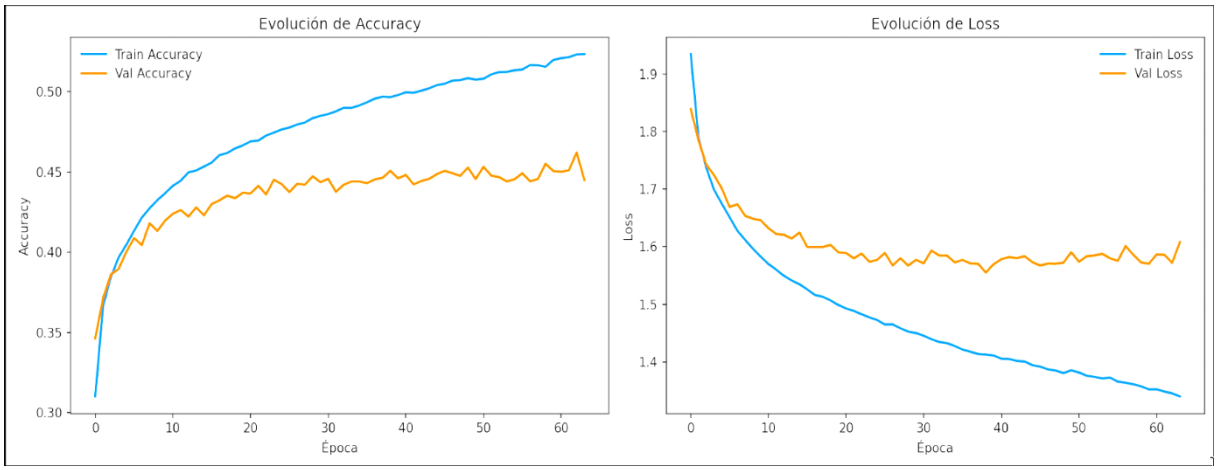
- Patience=15: El extremo superior del rango razonable. Permite verificar si valores altos de patience conducen a sobreentrenamiento.

Gráficas obtenidas:

Gráfica comparativa entre épocas:



Gráfica history sobre el mejor modelo:



Conclusión:

Como se puede observar con patience 5 es óptimo ya que tiene mejor tiempo mejor tasa de acierto y épocas muy similares a 7 patience.

Por otra parte, se aprecia que con 15 patience es ligeramente superior la tasa de acierto, pero no compensa tan ínfima diferencia con la pérdida de tiempo que hay en comparación, por lo que usaremos patience 5 para nuestro modelo optimizado.

MLP3:

En este apartado probaremos que batch size es el optimo para nuestro modelo óptimo, probaremos con diferentes tamaños de batch size.

- Batch size = 16: Proporciona una excelente capacidad de exploración en el espacio de la función de pérdida. Tiende a resultar en la mejor generalización y precisión final, su único problema es la cantidad de tiempo que necesita ya que requiere de un mayor número de actualizaciones de pesos por época.

- Batch size = 32: Es el punto de referencia común que ofrece el compromiso ideal. El ruido del gradiente es suficiente para evitar estancamientos sin comprometer la estabilidad de la actualización de pesos, siendo muy utilizado por su balance.

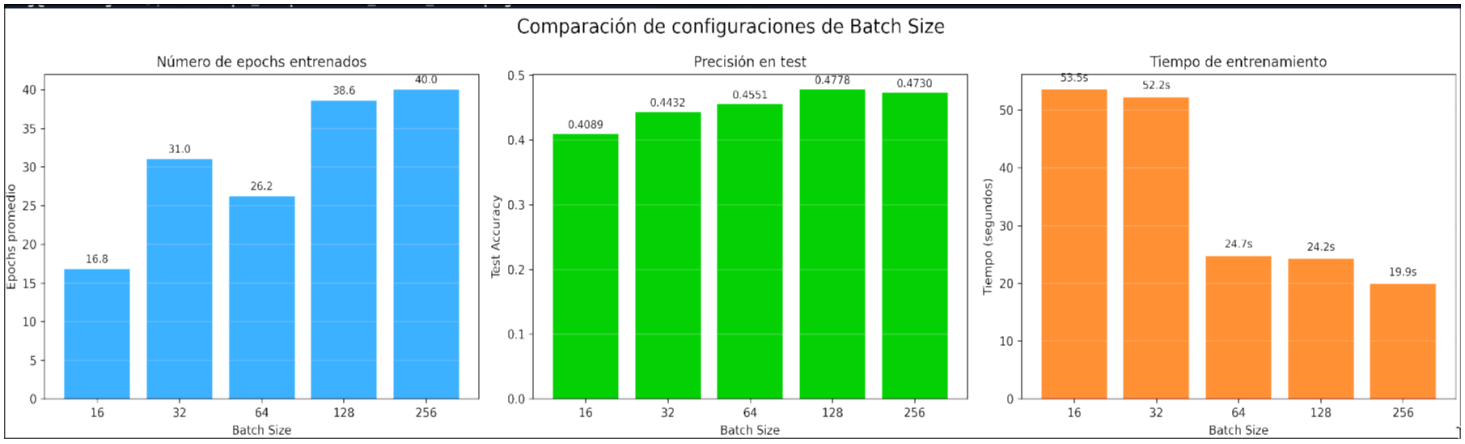
- Batch size = 64: Mantiene una buena capacidad para que el modelo encuentre óptimos profundos. La estabilidad del gradiente comienza a ser más notoria, siendo un tamaño eficiente para modelos que ya han mostrado ser robustos.

- Batch size = 128. El gradiente es muy estable y la trayectoria de entrenamiento es directa y rápida. El modelo tiene menos oportunidades de escapar de mínimos locales y puede converger a soluciones que no son las mejores para la generalización.

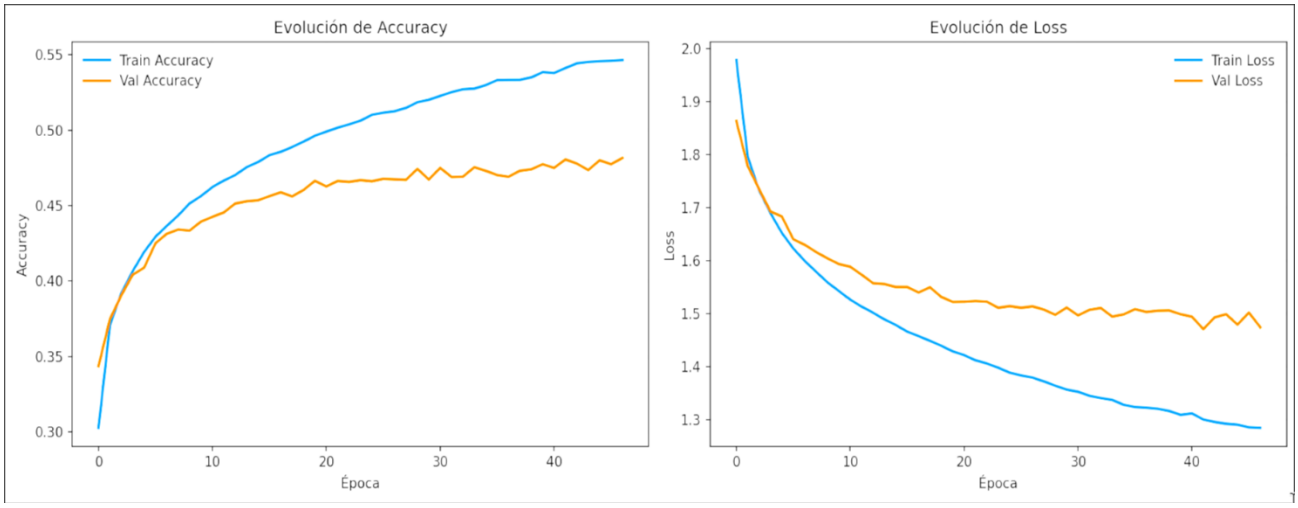
- Batch size = 256. La cantidad de bloques de imágenes a procesar es mucho menor lo que lleva a una mayor velocidad, sin embargo, la red se arriesga a quedarse estancada en soluciones subóptimas, lo que podría llevar a una menor precisión final.

Gráficas obtenidas:

Gráfica comparativa entre Batch size:



Gráfica history sobre el mejor modelo:



Conclusión:

Como se observa en la gráfica de barras, el ganador en este caso para ser elegido como mejor batch size para el modelo óptimo es 128 ya que tiene la mejor probabilidad de acierto (47,78%) en uno de los mejores tiempos (24,25 seg) aunque bien es cierto que ha entrenado en el segundo número mayor de épocas no es algo relativamente negativo viendo su correspondencia a las otras dos variables.

Observando la segunda gráfica vemos que el valor de precisión y perdida se estabilizan positivamente respecto a sus definiciones a partir de las 30 épocas.

MLP4:

En este apartado probaremos que función de activación es óptima para los parámetros obtenidos de los MLP anteriores.

Las funciones de activación de Keras se encuentran definidas y explicadas en su página oficial dentro del subapartado Layer activations de Layers API dentro de la documentación de las apis.

Los diferentes tipos de funciones de activación que se probarán son las siguientes.

- Activación = Sigmoid: Comprime cualquier entrada numérica a un valor entre 0 y 1. Se utiliza para introducir no linealidad y es esencial en la clasificación binaria para representar probabilidades. Es bastante lenta debido a su elevado coste computacional.

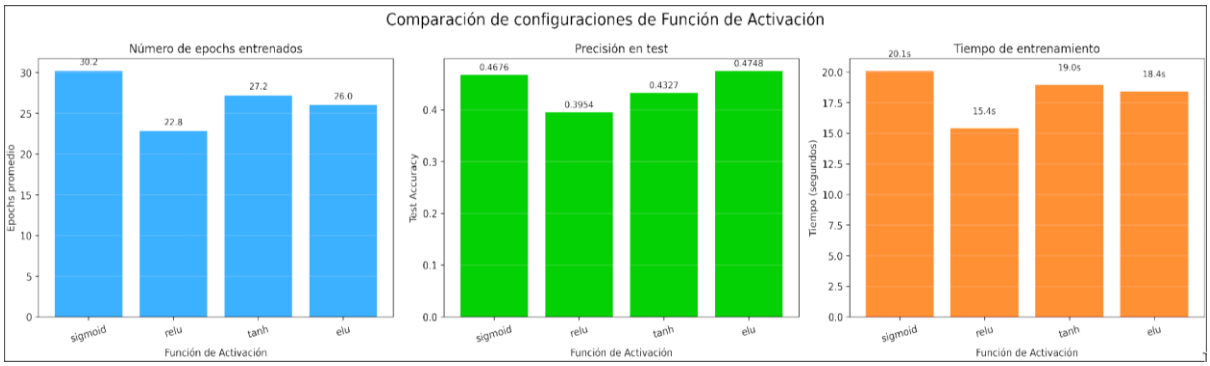
- Activación = ReLU: Rectifica la señal. Si la entrada es positiva, la deja pasar sin cambios. Si la entrada es negativa o cero, la salida es cero. Simple y veloz, ayudando a acelerar el entrenamiento

- Activación = Tanh: Comprime cualquier entrada numérica a un valor de salida entre -1 y 1. Se usa como alternativa a la Sigmoid para centrar las activaciones alrededor de cero, lo que ayuda a la estabilidad del optimizador.

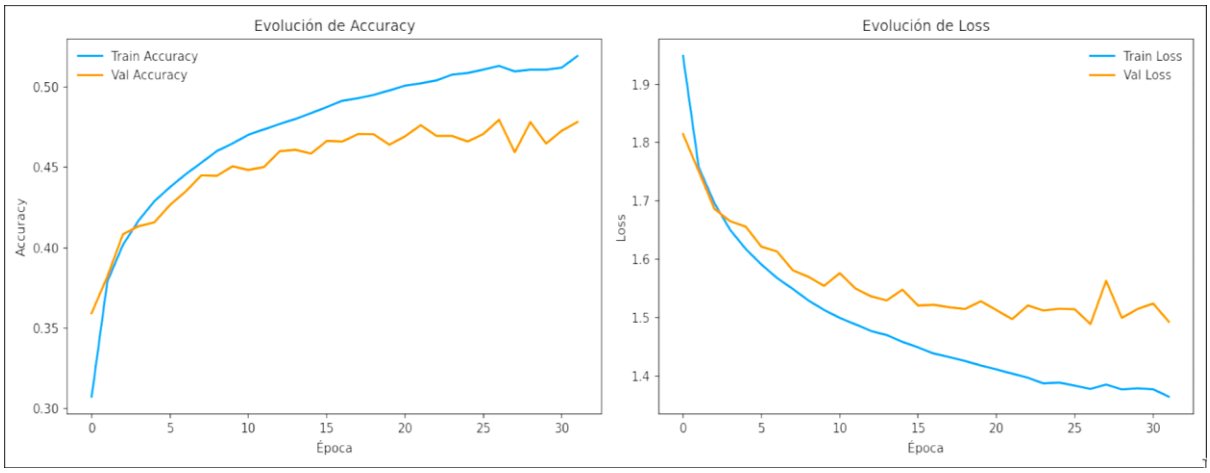
- Activación = ELU: Combina la rectificación para valores positivos, con una curva exponencial suave para valores negativos. Ayuda a que la media de las activaciones se mantenga cerca de cero, mitigando el problema del gradiente desvaneciente y facilitando la convergencia.

Gráficas obtenidas:

Gráfica comparativa entre funciones de activación:



Gráfica history sobre el mejor modelo:



Conclusión:

Como se observa la más eficiente en este caso es elu con un tiempo medio no muy variado con las otras, pero una precisión notablemente mayor al resto de funciones. Cuenta además con una media de épocas por entrenamiento relativamente baja comparado con el resto de funciones.

Respecto a la segunda gráfica vemos que los valores se estabilizan más a partir de 20-25 épocas.

MLP5:

En este apartado se probará que número de neuronas es optima para usar en las capas ocultas del modelo.

Se probarán con las siguientes cantidades de neuronas:

- Nº neuronas = 24: Valor muy baja, se puede utilizar para probar modelos muy simples o comprobar el margen de mejora del modelo, suele causar cuellos de botella graves en el modelo debido a esta limitación a la hora de procesar los datos.

- Nº neuronas = 48: Valor muy limitada, se utiliza en casos de prueba en los que el objetivo sea comprobar datos sin mucha profundidad, es decir pruebas rápidas o iniciales de un proyecto relacionado con el modelo. Gasta poco tiempo y recursos.

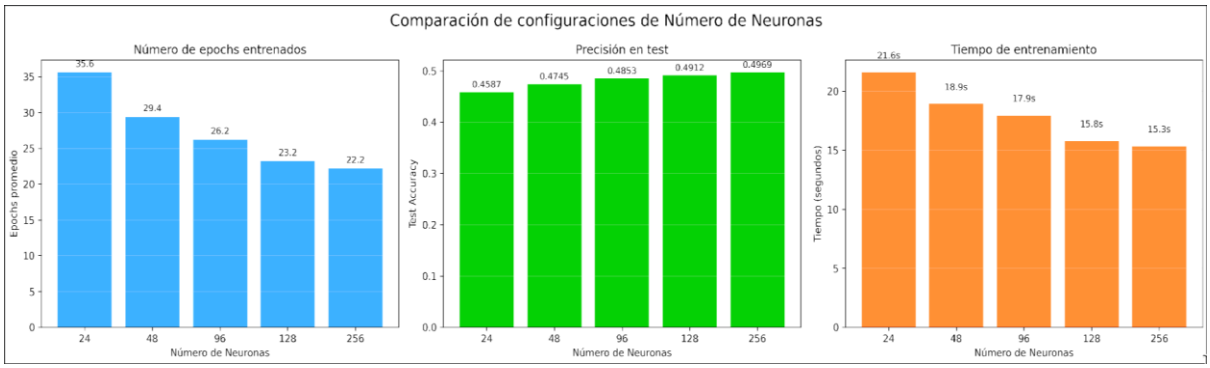
- Nº neuronas = 96: Valor adecuado y balanceado el cual funciona en modelos sin gran complejidad. Ofrece suficiente capacidad para modelar relaciones complejas sin aumentar drásticamente el riesgo de sobreajuste

- Nº neuronas = 128: Valor óptimo y muy común, ya que aprovecha la eficiencia del cómputo binario. Ofrece un rendimiento sólido mediante una capacidad alta que puede aprender relaciones complejas sin aumentar drásticamente el costo computacional, siempre que se use una buena regularización.

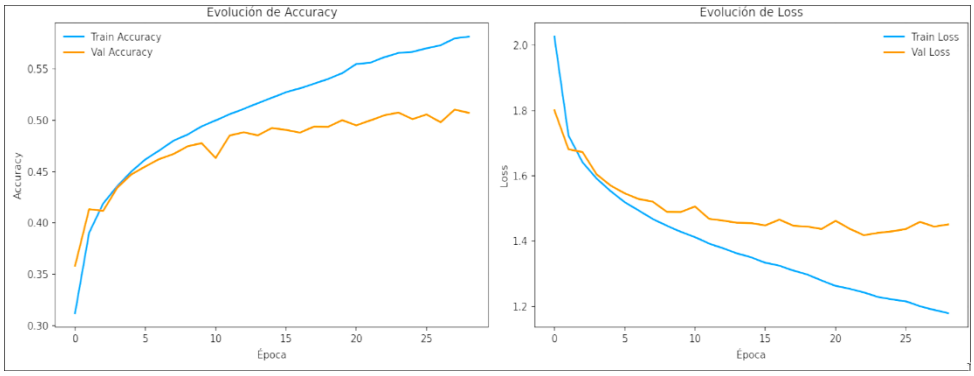
- Nº neuronas = 256: Valor muy alto. Se utiliza en modelos diseñados para problemas de grandes complejidades en los que se requiera la máxima capacidad posible de aprendizaje. Su principal inconveniente es el riesgo de que el modelo aprenda ruido o particularidades del conjunto de entrenamiento.

Gráficas obtenidas:

Gráfica comparativa entre nº de neuronas:



Gráfica history sobre el mejor modelo:



Conclusión:

Como se aprecia en la gráfica comparativa, la mejor cantidad de neuronas para el modelo es e 256 ya que es el que menos tiempo tarda y mayor índice de acierto tiene, el único inconveniente que se puede recalcar, observado en la gráfica es el riesgo de aprender detalles no importantes y de dar por hecho datos que sean incorrectos ya que como se ve en la práctica ha necesitado muy pocas épocas para entrenarse.

En la segunda gráfica observamos que los valores de acierto y loss se regulan a partir de las 15 épocas.

MLP6:

En este apartado revisaremos cual debería ser la arquitectura del método para que sea lo más optimo posible, es decir, su cantidad de capas ocultas y neuronas en estas capas.
Se probará con las siguientes arquitecturas:

1 capa oculta:

- Arquitectura = [128]: Priorizando la velocidad de entrenamiento.
- Arquitectura = [256]: Busca un rendimiento ligeramente superior a 128, aunque con mayor riesgo de sobreajuste y coste computacional.

2 capas ocultas:

- Arquitectura = [128, 64]: Arquitectura clásica donde la capacidad disminuye con la profundidad. La primera capa aprende características complejas y a la segunda capa comprime y refinar esa representación.

- Arquitectura = [256, 128]: Arquitectura ancha y profunda. La primera capa se ocupa de una máxima extracción de características de bajo nivel y la segunda refina esas características y crea una representación más abstracta y robusta. Al usar tantas neuronas tiene riesgo de sobreajuste.

- Arquitectura = [128, 128]: Prueba si la profundidad es más importante que la compresión. La primera capa extrae características iniciales y la segunda realiza una segunda pasada de refinamiento lo que le permite a la red profundizar en la imagen, tomando los patrones simples que encontró la Capa 1 y volviéndolos más complejos y abstractos.

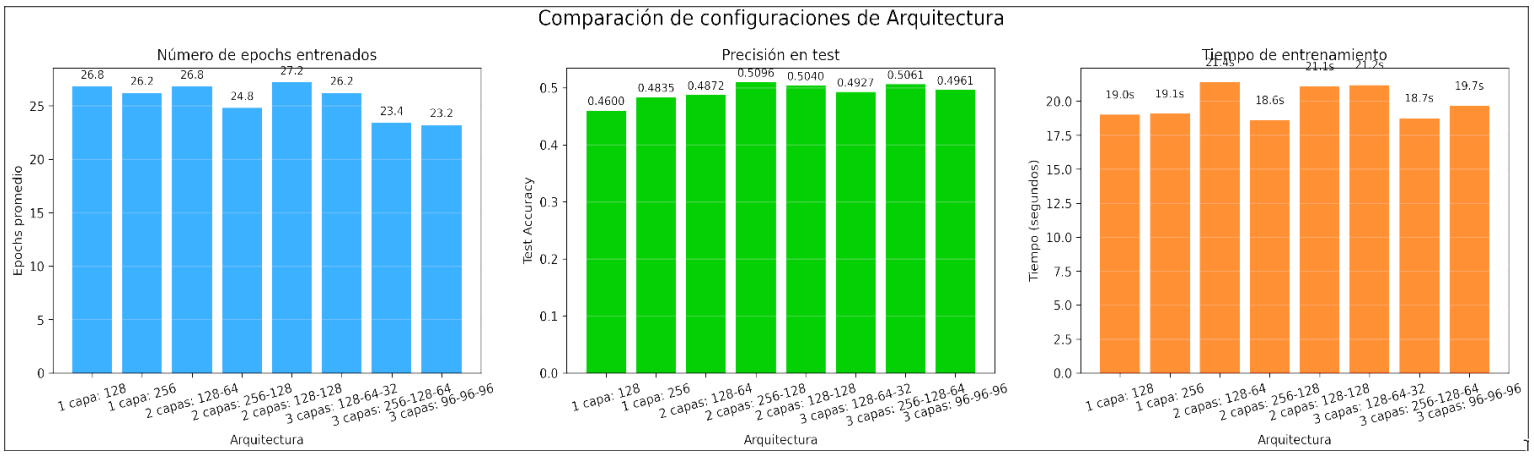
3 capas ocultas:

- Arquitectura = [128, 64, 32]: La primera capa realiza una extracción inicial, la segunda realiza una abstracción y compresión de la extracción inicial y la tercera se encarga de destilar la información útil en la representación más compacta posible antes de la salida.

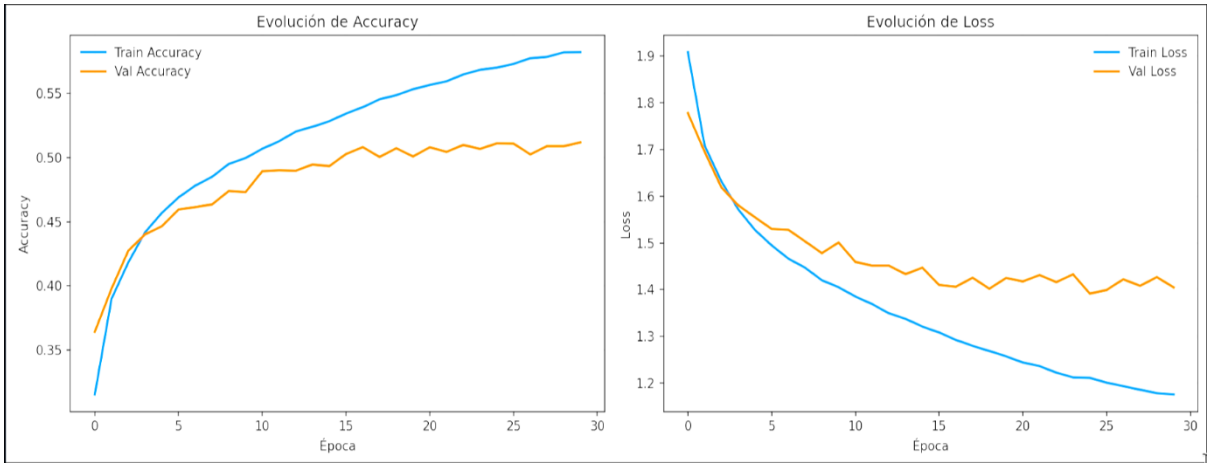
- Arquitectura = [256, 128, 64]: La primera capa se encarga de una extracción de máxima capacidad, la segunda se encarga de la combinación de patrones y la tercera realiza una representación final muy abstracta y compleja. Se prueba el límite de la jerarquía.

- Arquitectura = [96, 96, 96]: Las tres capas trabajan con la misma capacidad para extraer patrones jerárquicos. Se busca ver si la profundidad constante es más eficiente que la compresión.

Gráficas obtenidas:



Gráfica history sobre el mejor modelo:



Conclusión:

Como se puede observar en la gráfica comparativa, la mejor opción es la arquitectura de dos capas [256, 128] ya que tiene mayor índice de acierto en el menor tiempo comparado con el resto de los modelos y entrenado en una cantidad de épocas relativamente bajo. Por lo que se concluye que un modelo con dos capas de profundidad moderada y alta capacidad esa lo óptimo.

En la segunda gráfica se observa que se regulan los datos a partir de poco más de 10 épocas de entrenamiento lo cual es un aumento considerable de optimización.

MLP7:

Usando los resultados obtenidos en las anteriores tareas del MLP vamos a construir un modelo óptimo con las siguientes características:

- Patience = 5: Ofrece el mejor equilibrio entre detener el sobreajuste y permitir que el modelo converja. Proporciona el margen adecuado para que val_loss demuestre una tendencia real de estancamiento o divergencia.

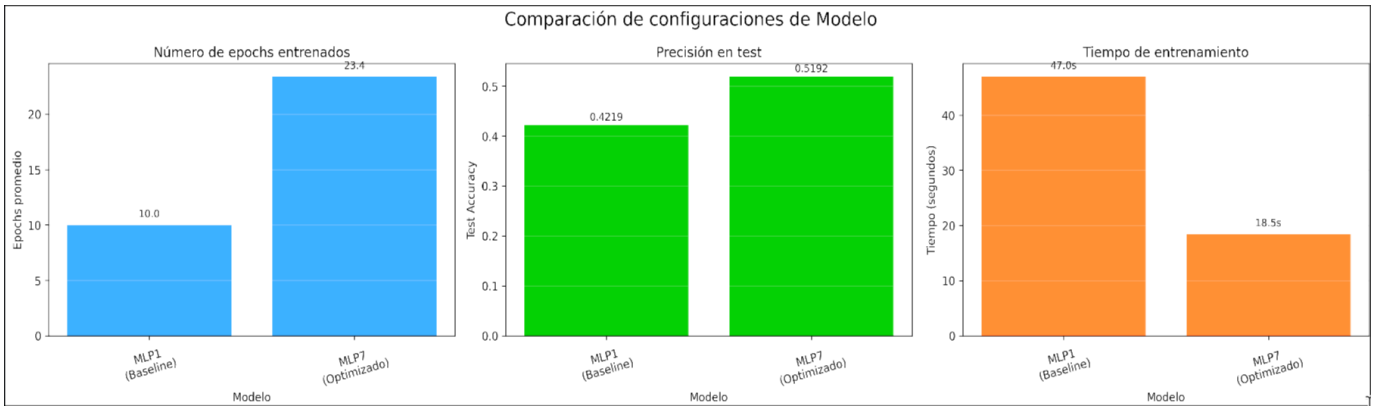
- Batch size = 128: Mostró ser la configuración que logra la mejor precisión final mientras mantiene una alta eficiencia de cómputo. Proporciona un gradiente más estable que los lotes pequeños, lo que acelera la convergencia, sin caer en el riesgo de *estancamiento* en el que caía el Bach size de 256.

- Función de activación = elu: Combina la eficiencia de ReLU con una curva exponencial suave para valores negativos. Esto ayuda a que el promedio de las activaciones se mantenga cerca de cero, lo que mitiga el gradiente desvaneciente y resulta en la convergencia más estable y precisa.

- Arquitectura = [256,128]: Confirma que el problema requiere profundidad moderada y alta capacidad jerárquica. La primera capa se encarga de la máxima extracción de características iniciales, y la segunda capa se encarga de refinarlas y comprimirlas, logrando una representación abstracta que optimiza la precisión.

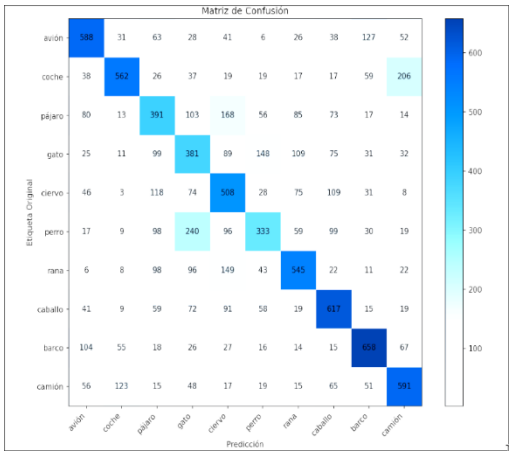
Gráficas obtenidas:

Gráfica comparativa entre el modelo del MLP1 y el modelo optimizado del

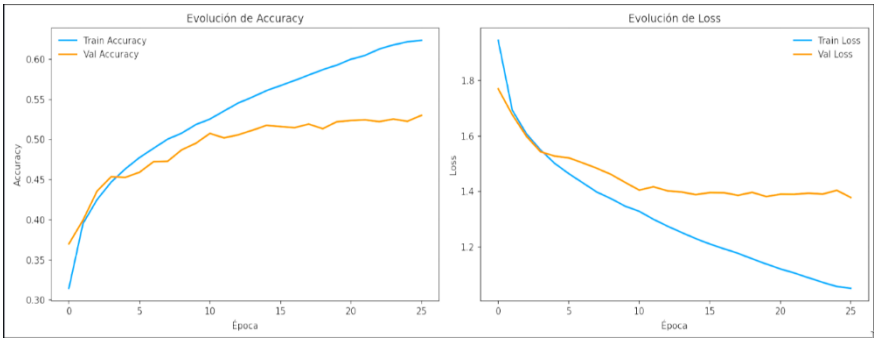


MLP7:

Matriz de confusión del modelo optimizado:



Gráfica history sobre el modelo optimizado:



Conclusión:

Como se observa en la primera gráfica en la que se compara el modelo optimizado con el modelo inicial, el modelo optimizado es completamente superior, teniendo una tasa de acierto casi 10% mayor al modelo inicial y en un tiempo que representa un cuarto de lo que tarda el modelo inicial.

Utiliza más épocas para entrenarse, pero no se excede en la cantidad que necesita.

Observando la gráfica de valores de aciertos y loss vemos que ambos valores se regulan a partir de poco más de 10 épocas.

Por último, de acuerdo a la matriz de confusión podemos observar que su índice de acierto es superior al de fallo corroborando el 51% de acierto que tiene el modelo optimizado

Bibliografía:

Documentación oficial Keras: https://keras.io/api/ y https://keras.io/guides/
Contienen todas las guias y apis de keras usadas en el código

Documentación oficial matplot: https://matplotlib.org/stable/plot_types/

Claude – Sonet 4.5

Gemini - Pro 3.0