Práctica 2: Visión artificial y aprendizaje

Objetivos

- Comprender el funcionamiento general de las redes neuronales artificiales como técnicas de aprendizaje supervisado, y en concreto los perceptrones multicapa y las redes neuronales convolucionales.
- Conocer en detalle el algoritmo de *backpropagation* y todo lo necesario para que una red neuronal pueda aprender a partir de un conjunto de datos.
- Entender cómo podemos utilizar la información de los píxeles de una imagen en visión artificial para distinguir lo que representa esa imagen.
- Conocer la librería Keras para implementar redes neuronales de forma rápida y efectiva.
- Utilizar redes neuronales para resolver un problema real de clasificación.
- Entender el papel de cada uno de los principales parámetros que configuran una red neuronal, y aprender a ajustarlos de forma efectiva y eficiente para maximizar su tasa de acierto a la vez que se minimiza el tiempo de entrenamiento.
- Utilizar Matplotlib para mostrar resultados de forma visual.
- Acelerar el flujo de trabajo utilizando *scripts* que generen de forma automática las pruebas para el ajuste de parámetros y los contenidos para el informe (resultados de clasificación, tablas, gráficas).

1) Introducción al problema a resolver y al entorno de trabajo. Normas de entrega y evaluación.

Introducción

En esta práctica trabajarás con técnicas de aprendizaje supervisado aplicadas al reconocimiento automático de imágenes en problemas reales de dificultad moderada y alta. En nuestro caso, clasificar diez tipos de vehículos y animales de la base de datos Cifar10 (https://www.cs.toronto.edu/~kriz/cifar.html). Utilizaras la librería Keras (https://keras.io/) de TensorFlow, que facilita configurar modelos de redes neuronales para problemas de clasificación y regresión.

La práctica se divide en dos partes, la primera centrada en perceptrones multi-capa (o MLP por las siglas en inglés), y la segunda en redes neuronales convolucionales (CNN). Además, cada parte se divide en dos apartados: Primero una serie de tareas relativamente sencillas, cada una centrada en uno de los parámetros clave de los modelos de redes neuronales con los que trabajarás, para que te familiarices con ellos y aprendas a ajustarlos de forma eficiente, y después una tarea más compleja, optimizar una arquitectura al completo para maximizar su tasa de acierto, en la que demostrarás las capacidades adquiridas y que conllevará la mayor parte de la nota de esa parte.

Evaluación

La implementación supone el 40% de la nota y la documentación el 60%. Ten en cuenta que **la mayor parte de la nota (80%) se obtendrá de las tareas finales de cada apartado (MLP7 y CNN3)**. Los ejercicios de la práctica que realices de la parte inicial (tanto de MLP como de CNN) contribuirán con un pequeño porcentaje (20%) ya que son más sencillos y están diseñados para introducirte en las tareas más complejas que desarrollarás más adelante.

Entorno de trabajo

En los laboratorios trabajaremos con Python en Linux. Python es un lenguaje interpretado multiplataforma muy extendido, se puede usar en todos los sistemas operativos de uso general. Puedes elegir el entorno de trabajo que prefieras (tanto sistema operativo como IDE), pero debes asegurarte de que tu código fuente funciona correctamente en Linux, en los PC de los laboratorios, porque ese es el entorno en que se evaluará la entrega.

Si no tienes experiencia con Python, se recomienda no usar Visual Studio y utilizar en su lugar el IDE Thonny (https://thonny.org/), ya instalado en los laboratorios, que da mejores ayudas a programadores de Python nóveles, y además tiene la ventaja de utilizar su propia instalación de Python, independiente de la del sistema operativo, de forma que no genera conflictos y facilita la instalación de paquetes como Tensorflow y Keras. Para instalar un paquete en Thonny ve a Herramientas → Gestionar paquetes, y ahí busca el paquete que desees instalar.

Además de Keras, en general siempre trabajaremos con Numpy (https://numpy.org/) para la manipulación eficiente de arrays, y alguna de las facilidades que nos provee Pandas (https://pandas.pydata.org/), y para el dibujado de imágenes y gráficas en pantalla utilizaremos Matplotlib (https://matplotlib.org/). También podrás utilizar cualquier librería de Python que se pueda instalar en vuestro Thonny o usando `pip --user install <librería>` en los laboratorios, pero deberás consultar primero con tu profesor/a. Como alternativa que no requiere instalación, tendrás la opción de utilizar Google Colab para realizar la práctica.

Si tu código no puede ejecutarse en los laboratorios o en Google Colab, no será evaluado, salvo acuerdo previo con el docente de tu grupo de prácticas.

Documentación

La documentación es la parte más **importante** de la práctica, supone el **60% de la nota máxima**. Tu documentación incluirá una sección para cada tarea, donde pondrás cualquier apunte que consideres relevante sobre su desarrollo, las pruebas que hayas realizado, las mejoras que hayas implementado y las conclusiones extraídas de esos experimentos. Es importante que concretes y sintetices (que vayas al grano y que resumas).

A menudo, la documentación de un experimento puede ser un poco confusa. Por ello, si tienes dudas sobre cómo presentar tus resultados, **consulta con tu profesor/a durante las horas de clase o en una tutoría**. Evita copiar información de chatbots o de fuentes que no entiendes bien. La clave no es la cantidad de texto o de gráficos, sino **la calidad y la claridad** de tu trabajo. De nada sirve llenar páginas con tablas y gráficas repetitivas. Lo que se valora es que la documentación refleje que has comprendido el experimento y has alcanzado los objetivos de aprendizaje. Lo más importante es la calidad, no la cantidad. Tu documentación debe mostrar que has aprendido lo que se te pide, no que has realizado muchos experimentos monótonos o que has usado lenguaje complicado sin entenderlo.

¡Consulta a tu profesor si tienes dudas sobre esto!

Bibliografía

La documentación debe incluir al final una sección de bibliografía con todas las referencias que utilices, o bien, estas referencias deben enlazarse en el texto donde se usen (entre paréntesis o como nota al pie). Si utilizas una sección de bibliografía, cada referencia en esa sección debe aparecer citada en el texto, en los apartados en que se ha usado. Una bibliografía mínima puede consistir solo en los apuntes de clase (aunque recomendamos que investigues más por tu cuenta). **La ausencia de bibliografía podrá penalizar hasta 1 punto**. Este requerimiento no significa que debas buscar algo que citar para rellenar, porque una buena bibliografía no da puntos, solo te permite aprender más deprisa y hacer mejor el trabajo. Si no has usado ninguna referencia externa, no debes perder el tiempo en inventarte una bibliografía. **También será motivo de penalización incluir referencias que no se usen en el texto**.

Las referencias a vídeos de plataformas como Youtube deben incluir un breve resumen de la información que se ha extraído de esa fuente.

Las consultas a ChatGPT, Claude, Bing y otros chatbots basados en grandes modelos de lenguaje son también parte de la bibliografía y deben ser referenciados. Si copias sus respuestas breves textualmente, debes marcarlas como una cita, entre comillas, dado que los resultados de un *prompt* no se pueden enlazar. Si los usas para entender conceptos o desarrollar explicaciones, en una conversación con el chatbot, no debes copiar la conversación entera en mitad de la documentación, pero puedes añadirla como un anexo al final (y citas la conversación desde el texto en que la usas), aunque será suficiente con que indiques en cada parte del texto en qué cuestiones te has apoyado en el chatbot.

Formato y fecha de entrega

Esta práctica tiene un hito intermedio, con fecha tope el 30 de noviembre de 2025, y una entrega final, hasta el 23 de diciembre de 2024 a las 23:55h. En el hito intermedio deberás entregar la parte de los MLP, como mínimo con todas las tareas completadas, y lo que puedas de la documentación. Esta entrega intermedia no se evalúa, pero la no entrega de este hito, o una entrega incompleta (en cuanto a la implementación), tendrá una penalización de hasta el 20% de la nota final de la práctica. En la entrega final deberás entregar ambas partes, ahora sí obligatoriamente con toda la documentación, y tendrás opción de mejorar lo que hubieras entregado en el hito intermedio. Ambas entregas se hacen en la misma tarea de Moodle, subiendo en cada caso un único fichero, intermedio.zip en la entrega intermedia y final.zip en la entrega final.

Para permitir la evaluación de tu entrega, deberás crear un módulo (una función) en tu archivo Python para cada tarea de la práctica, que llamará a todas las funciones implicadas en completar esa tarea. La idea es que los profesores podamos ejecutar solo la parte que queramos de tu código, sin tener que esperar a que se entrenen

todos los clasificadores de todas las tareas. Estos módulos se podrán llamar individualmente en la última sección de tu archivo Python, utilizando el condicional `__name__ == "__main__"`. Algo como lo que sigue:

```
if __name__ == "__main__":

    ## Funciones auxiliares relevantes para la evaluación, comentadas
    #test_MLP(…)

    X_train, Y_train, X_test, Y_test = cargar_y_preprocesar_cifar10()

    # tarea A
    #MLP1 = probar_MLP(X_train, X_test, Y_train, Y_test, capas=[…], …)
    #MLP2 = probar_MLP(X_train, X_test, Y_train, Y_test, capas=[…], …)

    # Tarea B
    #CNN1 = probar_CNN(X_train, X_test, Y_train, Y_test, capas=[…], …)
    CNN2 = probar_CNN(X_train, X_test, Y_train, Y_test, capas=[…], …)
```

Puede que algunas tareas requieran varios módulos, o que reúsen módulos de otras tareas con distintos parámetros (en el código de ejemplo, el módulo probar_MLP se usa para declarar, compilar, entrenar y evaluar un modelo de MLP, definido por sus argumentos). Si los nombres de tus funciones son ambiguos, utiliza comentarios para dejar claro qué hace cada módulo. En el momento de la entrega, **deja comentados todos los módulos menos el de la última tarea**.

Las prácticas son individuales, no se pueden hacer en grupos y no se puede reusar código o texto de otros estudiantes ni de otras fuentes sin referenciarlo. La entrega se realizará **a través de Moodle**, y debe consistir en dos ficheros, **un fichero .pdf con la documentación y otro fichero .py con todo el código Python**, tanto las implementaciones de los algoritmos requeridos como el código utilizado para producir todos los resultados que muestres en la documentación (gráficas, tablas, etc.). No se puede entregar la práctica en varios ficheros de Python. Si deseas programar en varios archivos por el motivo que sea, asegúrate de juntarlo todo en un único archivo .py y comprobar que todo funciona correctamente antes de la entrega. Los dos archivos se deben llamar con el nombre completo del estudiante, usando '_' como separador entre palabras (p.ej. Juan_Carlos_Sánchez-Arjona_de_los_Rios.py y .pdf). En el caso de que creas necesario adjuntar algún otro archivo (una hoja de cálculo con resultados, un archivo de log, etc.) consulta primero a tu profesor/a.

2) Toma de contacto con Cifar 10 y las gráficas de Matplotlib

Trabajando con las imágenes de Cifar10

Vamos a cargar la base de datos de Cifar10 y familiarizarnos con la manipulación de esas imágenes. Hay muchas maneras de cargar las imágenes de Cifar10 en un programa Python. Nosotros utilizaremos la función que nos facilita Keras (https://keras.io/api/datasets/cifar10/). Para ello primero hay que importar esa librería (e instalarla si no está instalada). Al ejecutar ese `import` algunos veréis una serie de mensajes de advertencia de TensorFlow, quejándose de que no encuentra soporte para usar CUDA en la tarjeta gráfica. Quienes tengan una tarjeta NVIDIA pueden instalar el soporte para CUDA y conseguir entrenamientos de modelos de aprendizaje mucho más rápidos gracias a la paralelización, pero no es necesario en esta práctica y no afecta a la calificación. Si queremos eliminar esos mensajes de error por comodidad, podemos utilizar el siguiente código al principio de nuestro archivo .py:

```
import logging, os
logging.disable(logging.WARNING)
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "3"
```

Ahora podemos cargar las imágenes de Cifar10 y ver en qué consisten:

```
import keras

(X_train, Y_train), (X_test, Y_test) = keras.datasets.cifar10.load_data()
print(X_train.shape, X_train.dtype)
print(Y_train.shape, Y_train.dtype)
print(X_test.shape, X_test.dtype)
print(Y_test.shape, Y_test.dtype)
```

Esos print nos dicen que `X_train` es un array de Numpy de 50 000 elementos (en este caso observaciones, o *samples* en inglés) que a su vez son arrays de 32x32x3 `uint8` (unsigned int de 8 bits, números de 0 a 255), es decir, un array de 50 000 imágenes en color de 32x32 píxeles. Cada uno de esos 32x32x3=3072 bytes de una imagen es una característica de una observación. `Y_train` es otro array de 50 000 elementos `uint8` que contiene los 50 000 números asociados a cada categoría (0: avión; 1: coche; 2: pájaro, etc.), y que son los valores que querremos que nos devuelva nuestro MLP para cada imagen cuando esté bien entrenado. Y los arrays `X_test` e `Y_test` son iguales pero contienen 10 000 observaciones cada uno.

Podemos mostrar en pantalla las imágenes que hay en `X_train` e `X_test` utilizando la función `imshow` de Matplotlib. El siguiente código mostrará tres imágenes al azar del conjunto de entrenamiento:

```
import matplotlib.pyplot as plt

def show_image(imagen, titulo):
    plt.figure()
    plt.suptitle(titulo)
    plt.imshow(imagen, cmap = “Greys”)
    plt.show()

from random import sample

for i in sample(list(range(len(X_train))), 3):
    titulo = "Mostrando imagen X_train[" + str(i) + "]"
    titulo = titulo + " -- Y_train[" + str(i) + "] = " + str(Y_train[i])
    show_image(X_train[i], titulo)
```

Gráficas de los resultados con Matplotlib

Puedes probar más cosas para adquirir destreza en Python con Numpy y Matplotlib, o sacar conclusiones acerca de los datos sobre los que trabajamos. Podrías contar la cantidad de imágenes de cada categoría en el conjunto de entrenamiento, o mostrar por pantalla los primeros tres perros. Por ejemplo, el siguiente código imprime una gráfica con los valores de las etiquetas de las 20 primeras imágenes del conjunto de test:

```
def plot_curva(Y, titulo, xscale = "linear", yscale = "linear"):
    plt.title(titulo)
    plt.plot(Y)
    plt.xscale(xscale)
    plt.yscale(yscale)
    plt.show()

plot_X(Y_test[:20], "Etiquetas de los primeros 20 valores")
```

Para los ejercicios y tareas de esta práctica deberás crear varias funciones que te facilitarán el trabajo y reducirán el tamaño del código. Se recomienda crear una función `cargar_y_preprocesar_cifar10()` que devuelva los cuatro conjuntos de datos de Cifar 10, con cualquier preprocesamiento que fuera necesario (los MLP necesitan un preprocesamiento distinto de las CNN), y otra que realice el entrenamiento de un modelo de MLP (definido por los los parámetros que se le pasen) con el conjunto de entrenamiento y la evaluación de los resultados con el conjunto de test, y otras funciones auxiliares que muestren las gráficas que facilitarán el análisis de rendimiento de los modelos a partir de las predicciones del modelo y las clases verdaderas:

1. La gráfica de líneas que muestren la evolución de la pérdida y la tasa de acierto para el conjunto de entrenamiento y para el conjunto de validación durante el entrenamiento de la red (en Keras, esta información la devuelve la función *fit*, que es la que realiza el entrenamiento).

2. La gráfica de barras con el tiempo de entrenamiento y la tasa de acierto finales con el conjunto de test (en Keras, esto lo conseguimos con la función *evaluate*), para comparar resultados de varios modelos.
3. La matriz de confusión para los resultados con el conjunto de test (investiga cómo puedes mostrar esa información de manera rápida y elegante, quizás tengas que instalar alguna otra librería de Python).

3) Primera parte: tareas con MLP de Keras

Recuerda: Entrega de la primera parte hasta el **30 de noviembre de 2025 a las 23:55h**.
La **no entrega** de este hito supone hasta un **20% de penalización** en la nota final de esta práctica.

Tarea MLP1: Definir, entrenar y evaluar un MLP con Keras

En este primer ejercicio aprenderás a declarar y emplear un MLP de Keras. En un MLP se utilizan capas Dense (es decir, cada neurona de una capa está conectada a todas las neuronas de la anterior capa), que no tienen en cuenta la organización espacial de las características de una observación y que esperarán que todas las características de una misma muestra aparezcan en un único vector (ten esto en cuenta al preprocesar los datos).

El primer paso será crear un fichero .py, si no lo has hecho ya, y programar la función para cargar los datos de Cifar10 y aplicarle cualquier preprocesamiento que sea necesario para que utilizar esos datos en un MLP.

El primer MLP que declararás ha de tener una sola capa oculta de tipo Dense con 48 neuronas y función de activación *sigmoid*, una capa de salida con el número de neuronas que sea conveniente para nuestro problema de clasificación y función de activación 'softmax', y debe utilizar el optimizador `Adam`. Puedes encontrar un ejemplo en la documentación de keras.io (https://keras.io/guides/sequential_model/).

Una vez declarado el modelo, habrás de compilarlo con la función *compile*, indicando como mínimo el `optimizer`, que ha de ser `Adam`, la función de pérdida (`loss`, que será 'categorical_crossentropy') y las métricas que se mostrarán (`metrics`, que serán 'accuracy'). Una vez compilado puedes mostrar un resumen de la estructura de tu modelo llamando a `summary`.

Después tendrás que entrenarlo con el conjunto de entrenamiento utilizando la función *fit*, pasándole `validation_split`=0.1, `batch_size`=32 y `epochs`=10. Esta función devolverá, entre otros, un historial de los valores de tasa de acierto (*accuracy*) y de pérdida durante el entrenamiento, tanto del conjunto de entrenamiento como del de validación, que podrás usar para plotear la gráfica de evolución del entrenamiento.

Una vez entrenado el modelo, debes evaluarlo con el conjunto de test, para obtener una tasa de acierto y una pérdida con ese conjunto, utilizando la función `evaluate`. Como veremos más adelante, comparando esa tasa de acierto con la que obtengas con otros modelos podrás estimar qué modelo es mejor.

Tarea MLP2: Ajustar el valor del parámetro epochs

En este ejercicio analizarás la evolución del entrenamiento de tu modelo para detectar si no se ha entrenado suficientemente (es decir, podría haber alcanzado mejores resultados solo con dejarlo entrenar más tiempo) o si se ha entrenado demasiado y se ha incurrido en sobreentrenamiento. El sobreentrenamiento ocurre cuando el clasificador se ajusta tan bien a las características del conjunto de entrenamiento que pierde generalidad y obtiene peores resultados con el conjunto de test de los que conseguiría si se hubiera entrenado menos.

Necesitarás ver las curvas de *train_accuracy*, *train_loss*, *validation_accuracy* y *validation_loss*, y para eso

emplearás Matplotlib. Anteriormente viste un ejemplo con una función *plot_curva* que dibuja una línea a partir de los puntos de una lista (parámetro Y), que nos puede servir para este ejercicio. En la página oficial de Matplotlib tenemos un ejemplo más elaborado (https://matplotlib.org/stable/plot_types/basic/plot.html), para ploteados en el que X no es uniforme y que dibuja varias curvas en la misma gráfica. Idealmente, querremos plotear las curvas de *accuracy* sobre uno de los ejes verticales y las curvas de *loss* sobre el otro, para poder ajustar el rango independientemente y que se aprecie mejor la pendiente de las curvas.

Una vez tengamos esa gráfica podemos comprobar rápidamente si el entrenamiento se detuvo prematuramente, cuando todas las curvas mostraban una mejora que podía continuar, o si se entrenó demasiado tiempo, cuando las curvas de validación empezaron a empeorar. Con esa información, debes ajustar el valor del número de épocas para que el entrenamiento se detenga en los mejores valores de *val_accuracy* (y en caso de duda, los mejores valores de *val_loss*).

Ten en cuenta que el entrenamiento de estos modelos parte de estados iniciados al azar, que influyen en el éxito del entrenamiento, por lo que dos entrenamientos diferentes del mismo modelo (todos los parámetros iguales) darán resultados diferentes, a veces muy diferentes. Por ello es necesario realizar varios entrenamientos independientes y analizar los resultados promedio, e idealmente descartar resultados *outlier*. En esta práctica sobra con que hagas cinco repeticiones de cada entrenamiento (guardas cada *history*, los promedias al final y eso es lo que ploteas), pero ten en cuenta que para conseguir robustez estadística en un entorno profesional se necesitarían más (o muchas más) repeticiones, dependiendo de la complejidad del problema y de lo bien ajustado que estén el resto de parámetros para evitar quedar atrapado en mínimos locales demasiado pronto.

Una vez comprendido cómo ajustar las épocas manualmente, podemos utilizar un *callback* de Keras (https://keras.io/api/callbacks/) llamado *EarlyStopping*, para que nos haga ese trabajo de forma automática, detectando el sobreentrenamiento durante el propio entrenamiento y parándolo antes de alcanzar el máximo de épocas. Los *callbacks* son funciones que se ejecutan durante el entrenamiento para conseguir ciertos efectos. Es recomendable utilizar este *callback* para todo el resto de tareas, pero ten en cuenta que si no lo configuras correctamente tus entrenamientos pararán antes de alcanzar su óptimo o perderán tiempo innecesariamente. Debes mostrar en esta tarea los experimentos que hagas para ajustar los parámetros de *EarlyStopping*.

Tarea MLP3: Ajustar el valor del parámetro batch_size

En este ejercicio debes documentarte sobre qué controla el parámetro *batch_size* en el algoritmo de entrenamiento de un MLP (p.ej.: https://stats.stackexchange.com/questions/153531/what-is-batch-size-in-neural-network) y comprobar experimentalmente los efectos que tiene sobre la velocidad y el éxito del entrenamiento de un MLP.

Primero entrena y evaluá varios MLP con diferentes valores de batch_size. Comprueba si deberías ajustar un número de epochs diferente para cada MLP. Después utiliza Matplotlib para crear **una** gráfica de barras que muestre el *test accuracy* y el tiempo de entrenamiento de todos los modelos que hayas probado. Atención, una sola gráfica con los resultados de todos los modelos, no una gráfica por cada modelo. A partir de esa gráfica, has de seleccionar el valor de *batch_size* que consiga la mejor tasa de acierto sin emplear más tiempo del necesario.

Tarea MLP4: Probar diferentes funciones de activación

Como ya hemos visto, al añadir una capa Dense a un MLP podemos indicar una función de activación (https://keras.io/api/layers/activations/). Documéntate sobre las recomendaciones de uso de cada función y el rango de valores que pueden devolver, además de sobre qué tecnicas de inicialización de pesos (https://keras.io/api/layers/initializers/) son las más adecuadas para cada una. Esto último es importante para evitar los problemas de gradiente desvaneciente, gradiente explosivo y neuronas muertas (y además te ahorrarás mucho tiempo de entrenamiento con funciones inadecuadas o cuyos pesos no han sido iniciados correctamente).

Elige entonces las dos o tres combinaciones de función de activación más técnica de iniciación de pesos que puedan ser mejores en el contexto de esta práctica que *sigmoid* con iniciación aleatoria. Después, entrena y

evalúa varios MLP con esas funciones de activación, y *plotea* los resultados de tasa de acierto y tiempo de entrenamiento como en el ejercicio anterior, para elegir la mejor combinación de función de activación y técnica de iniciación para este problema y modelo de red neuronal.

Tarea MLP5: Ajustar el número de neuronas

Ahora entrena y evalúa varios MLP con diferente número de neuronas en su capa oculta. Analiza los resultados para poder seleccionar el modelo que consiga la mejor *test_acc* sin desperdiciar tiempo. Verás que a partir de cierto número de neuronas el tiempo de entrenamiento sigue aumentando sin mejorar la tasa de acierto final (o incluso empeorando), por lo que no siempre más es mejor.

Tarea MLP6: Ajustar el número de capas y de neuronas por capa

Añade una o más capas Dense a tu modelo y vuelve a ajustar todos los parámetros, de la forma en que has aprendido en las tareas anteriores, para mejorar la tasa de acierto o reducir el tiempo de entrenamiento sin empeorar la tasa de acierto. Prueba también el efecto que tendrá en los resultados distintas formas de distribuir las neuronas totales entre las capas (p.ej., con 96 neuronas en total, podrías probar [96] (una capa de 96), [48+48] (dos capas de 48), [32+64], [64+32], [32+32+32], [16+32+48], etc.).

Tarea MLP7: Optimizar la arquitectura de un MLP

En esta tarea tu objetivo es mejorar lo máximo posible la tasa de acierto de test de un MLP sin emplear más de 1000 neuronas ni más de 6 capas Dense. Para ello deberás valerte de las conocimientos adquiridos en los ejercicios anteriores.

En esta tarea además puedes incluir diferentes mejoras sobre el entrenamiento de las redes con el objetivo de aumentar su capacidad de generalización, acelerar su convergencia o mejorar su tasa de acierto. Algunas de las mejoras que podrías incluir son las siguientes:

- Métodos de regularización: https://keras.io/api/layers/regularization_layers/
- Aumento de datos: https://keras.io/api/layers/preprocessing_layers/image_augmentation/
- Normalización y estandarización de datos: https://keras.io/api/layers/normalization_layers/

Las mejoras que propongas deberán estar fundamentadas, implementadas y evaluadas convenientemente para que puedan ser tenidas en consideración. Ten en cuenta que ciertas técnicas no serán efectivas al mezclarlas con otras, o que necesitarán de otras técnicas para extraer todo su potencial. Cuantas más mejoras pruebes y documentes (aunque no ayuden a mejorar la tasa de acierto final) más nota tendrás en esta tarea, pero en general se puede alcanzar la nota máxima probando con tres de las mejora listadas arriba.

Es recomendable que te documentes antes de iniciar la experimentación, para apoyarte en resultados previos de otros investigadores (todo bien referenciado en tu bibliografía) y poder dirigir mejor tus tiempo y esfuerzos. Pero ten en cuenta que en muchos de los modelos que podrás encontrar en la literatura del arte no tienen esa limitación de número de filtros: no puedes utilizar los modelos grandes tal cuál porque tardarán demasiado, pero sí puedes sacar ideas de su arquitectura y las técnicas auxiliares empleadas.

Se valora que el código esté correctamente comentado y que sea autoexplicativo (evitar nombres de funciones o variables que no se entiendan). No incluyas fragmentos de código en la documentación salvo que sea imprescindible para explicar algo y que creas que no es suficiente con comentarlo en el código.

Una vez realizado el estudio y la experimentación con los modelos de MLP que decidas, debes documentar bien todo tu trabajo, mostrando las gráficas relevantes y comentando los resultados. Para comprender mejor los resultados de los modelos generados, se deberán incluir matrices de confusión que permitan visualizar las categorías sobre las que se confunde.

Finalmente, deberás indicar cuál es la mejor configuración obtenida, comparando los resultados en forma de

tabla resumen, justificando adecuadamente dicha decisión, y desarrollando en tus conclusiones por qué crees que una arquitectura de MLP es mejor que otras.

4) Segunda parte: tareas con CNN de Keras

Recuerda: Entrega final hasta el **23 de diciembre de 2024 a las 23:55h**.

En los siguientes ejercicios, parecidos a los anteriores, volverás a clasificar las imágenes de Cifar10 con redes neuronales, pero esta vez utilizando redes convolucionales, cuyas "neuronas" (o unidades) aplican filtros de convolución (https://en.wikipedia.org/wiki/Kernel_(image_processing)) a las imágenes de entrada, permitiendo la extracción de características de más alto nivel, por lo que están mejor adaptadas a problemas de visión artificial.

Tarea CNN1: Definir, entrenar y evaluar un CNN sencillo con Keras

Para completar este ejercicio necesitarás documentarte sobre las CNN y buscar ejemplos de implementación utilizando Keras. Encontrarás varios en la propia web de Keras (p.ej. https://keras.io/examples/vision/mnist_convnet/). Ten en cuenta que las capas de una CNN utilizan la información espacial y de color, por lo que las imágenes no se deben preprocesar de la misma manera que con las capas Dense de un MLP: deben mantener su distribución espacial intacta.

Para empezar a experimentar, declara una CNN con dos bloques de convolución compuestos cada uno de una capa Conv2D con filtros (*kernel*) de 3x3 y activación ReLU con inicializador He Normal, seguida de una capa de MaxPooling2D con filtros de 2x2. El primer bloque ha de tener 16 filtros de convolución y el segundo 32. Al final de la red debes añadir una capa oculta Dense de 100 neuronas, y finalmente, como capa de salida otra capa Dense con función de activación softmax. Deja el resto de parámetros en sus valores por defecto pero ajusta *epochs* como hiciste en ejercicios anteriores (apóyate en gráficas para comprobar que tu *EarlyStopping* está bien configurado, puede que requiera valores distintos a los que necesitaban los MLP).

Tarea CNN2: Ajustar el parámetro *kernel_size* de la CNN

Ahora debes entrenar varios modelos de tu CNN con diferente tamaño de filtros en las capas Conv2D, analizar sus resultados apoyándote en gráficas y seleccionar el mejor. Ten en cuenta que a mayor tamaño del filtro, la red tendrá mayor capacidad de detección de características grandes con menos capas Conv2D, pero que su cálculo y entrenamiento es más costoso.

Tarea CNN3: Optimizar la arquitectura de un CNN

Esta tarea es análoga a la tarea MLP7. De nuevo tu objetivo es optimizar la arquitectura de una CNN para maximizar su tasa de acierto empleando unos recursos limitados: No debes utilizar más de 32 filtros en las capas Conv2D antes del primer *MaxPooling2D* y no más de 128 filtros en el resto de capas *Conv2D*. Para desarrollar esta tarea, sigue el mismo guión que en la tarea MLP7, con el añadido de que aquí puedes probar otros métodos de agrupación (*pooling*) aparte de *MaxPooling*: https://keras.io/api/layers/pooling_layers/.

Aunque no se incluyen en este enunciado como ejercicios separados, para obtener los mejores resultados de tu CNN necesitarías ajustar el batch_size, las funciones de activación, el número de capas de convolución y la distribución de los filtros entre capas, además del número y tamaño de capas Dense finales. Reconfigurar es necesario porque lo que funciona bien con un modelo no tiene por qué funcionar bien con otro. Deberás documentar esa experimentación en esta tarea, pero con concisión.

Finalmente, además de comentar por qué unas arquitecturas de CNN son mejores que otras, comenta también las ventajas de CNN frente a MLP y viceversa.

5) Consideraciones finales

- En el hito intermedio deberás entregar la parte de los MLP, como mínimo con todas las tareas completadas, en un fichero llamado intermedio.zip, y lo que puedas de la documentación, antes del 30 de noviembre de 2025 a las 23:55h.
- Esta entrega intermedia no se evalúa, pero la no entrega de este hito, o una entrega incompleta (en cuanto a la implementación), tendrá una penalización de hasta el 20% de la nota final de la práctica.
- En la entrega final deberás entregar ambas partes, en un fichero final.zip, ahora sí obligatoriamente con toda la documentación, donde podrás mejorar lo que hubieras entregado en el hito intermedio.
- La implementación supone el 40% de la nota y la documentación el 60%.
- **La mayor parte de la nota (80%) se obtendrá de las tareas finales de cada apartado (MLP7 y CNN3)**. Los ejercicios de la práctica que realices de la parte inicial (tanto de MLP cómo de CNN) contribuirán con un pequeño porcentaje (20%) ya que son más sencillos y están diseñados para introducirte en las tareas más complejas que desarrollarás más adelante.
- Las prácticas son individuales, no se pueden hacer en grupos y no se puede reusar código o texto de otros estudiantes ni de otras fuentes sin referenciarlo. La entrega se realizará **a través de Moodle**, y debe consistir en dos ficheros, **un fichero .pdf con la documentación y otro fichero .py con todo el código Python**, tanto las implementaciones de los algoritmos requeridos como el código utilizado para producir todos los resultados que muestres en la documentación (gráficas, tablas, etc.).
- No se puede entregar la práctica en varios ficheros de Python. Si deseas programar en varios archivos por el motivo que sea, asegúrate de juntarlo todo en un único archivo .py y comprobar que todo funciona correctamente antes de la entrega.
- Los dos archivos de la práctica se deben llamar con tu nombre completo, usando '_' como separador entre palabras (p.ej. Juan_Carlos_Sánchez-Arjona_de_los_Rios.py y .pdf). En el caso de que creas necesario adjuntar algún otro archivo (una hoja de cálculo con resultados, un archivo de log, etc.) consulta primero a tu profesor/a.

AVISO IMPORTANTE

No cumplir cualquiera de las normas de entrega descritas al principio de este documento puede suponer un suspenso de la práctica.

Las prácticas son individuales. No se pueden hacer en pareja o grupos.

Cualquier código o texto copiado supondrá un suspenso de la práctica para todos los estudiantes implicados y **se informará a la dirección de la EPS para la toma de medidas oportunas** (Reglamento para la Evaluación de Aprendizajes de la Universidad de Alicante, BOUA 9/12/2015, y documento de Actuación ante copia en pruebas de evaluación de la EPS).