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.
- Entender el concepto de imagen y píxel, y cómo podemos utilizar las técnicas de aprendizaje supervisado para distinguir lo que representa una imagen.
- 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.
- 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 sin desperdiciar tiempo de entrenamiento.
- Utilizar Matplotlib (o similares) 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).

Sesión 1: Introducción al problema a resolver y el entorno de trabajo. Normas de entrega y evaluación.

Introducción

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

La práctica se divide en dos partes. En la **primera parte** desarrollaremos modelos de redes neuronales con Keras para clasificar correctamente las imágenes de Cifar10. En la **segunda parte,** crearemos un dataset propio basado en las clases de Cifar10 y realizaremos las pruebas necesarias para validar la generalización de nuestro modelo.

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, y cada estudiante puede elegir el entorno de trabajo que prefiera (tanto sistema operativo como IDE), pero debes asegurarte de que tu código fuente funciona correctamente en Linux en los PC de los laboratorios usando Thonny (https://thonny.org/), 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 (ya instalado en los laboratorios), puesto 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 instar.

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.

Evaluación

La primera parte de la práctica aporta hasta 6 puntos de la nota final, mientras que la segunda parte aporta hasta 4 puntos (máximo 10 en total). Dentro de cada parte hay apartados obligatorios y optativos (solo están marcados los apartados optativos, todos los demás son obligatorios). Para aprobar la práctica, todos los apartados obligatorios deben estar realizados. Los apartados marcados como optativos no son necesarios para aprobar, pero permiten compensar la nota de apartados obligatorios que no hayan salido bien. La implementación supone el 40% de la nota y la documentación el 60%.

Esta práctica tiene **dos hitos de entrega, ambos obligatorios**. En el primer hito has de entregar la implementación de la primera parte de la práctica, que será revisada pero no contará para la calificación final: la usaremos para comprobar el desarrollo de la práctica hasta el momento, ayudar a corregir errores y solucionar problemas antes de la entrega final. El segundo hito será la entrega final, en la que se basará la calificación. Sin embargo, **si no entregas el primer hito** o si no alcanzas un mínimo de implementación de los apartados de la primera parte en esa entrega, supondrá una **penalización de hasta un 20% de la nota final**.

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 apartado de la práctica, donde pondrás las respuestas a las preguntas que se te plantean y los resultados que se piden en cada apartado, además de cualquier apunte que consideres relevante sobre el desarrollo de la práctica, pruebas extra que hayas realizado, mejoras que hayas implementado, y conclusiones extraídas de esos experimentos. Es importante que concretes y sintetices (que vayas al grano y que resumas). Si no tienes clara la respuesta a una pregunta o de qué manera debes informar sobre tus experimentos, es mejor que le pidas ayuda tu profesor/a durante las horas de prácticas o por tutoría, y no que copies parrafadas de ChatGPT que no terminas de entender, o que pongas páginas y páginas de gráficas o tablas repetitivas que te llevó mucho tiempo completar y que no te va a subir la nota. Lo importante es que tu documentación demuestre que has aprendido lo que se te pide, no que hayas hecho muchos experimentos repetitivos. Consulta a tu profesor si tienes dudas sobre esto.

La documentación **debe incluir 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). 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, 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, 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. 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, o si no se referencia en el texto donde se use, podrá penalizar hasta 1 punto**. Este requerimiento no significa que debas buscar algo que citar para rellenar (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, citar elementos que no utilizas también será motivo de penalización.

Fechas de entrega

Hito 1 (primera parte de la práctica): hasta el **30 de noviembre de 2024 a las 23:55h**.

Hito 2 (entrega final): hasta el **23 de diciembre de 2024 a las 23:55h**.

Formato de entrega

Las prácticas son individuales, no se pueden hacer en grupos y no se puede reusar código o texto de otros estudiantes. 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 utilizado por el estudiante**, tanto las implementaciones de los algoritmos requeridos como cualquier código utilizado para producir cualquier resultado que se muestre en la documentación. 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 entrega 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 tu profesor/a.

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. 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
    mlpA = probar_MLP(X_train[0].shape, ocultas=[32], activ=["sigmoid"])

    # Tarea B
    mlpB = probar_MLP(X_train[0].shape, capas=[32], activ=["sigmoid"], ep=32)
```

Puede que algunas tareas requieran varios módulos, o que reusen módulos de otras tareas con distintos parámetros (en el código de ejemplo, el módulo probar_MLP se usa para declarar, compilar, entrenar y evaluar distintos modelos de MLP). Si los nombres de tus funciones son ambiguos, utiliza comentarios para dejar claro

qué hace cada módulo. 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. En el momento de la entrega, deja comentados todos los módulos menos el último de cada tarea.

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"
from tensorflow import keras
```

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

```
(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)
```

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 100 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[:100], "Etiquetas de los primeros 1000 valores")
```

Sesión 2: MLP de Keras (definir y entrenar)

Para cada tarea de esta sección deberás crear una función que realice la carga de datos, el entrenamiento del modelo de MLP según los parámetros que tú le digas, y la evaluación de los resultados. Se recomienda crear una función `cargarCifar10()` que devuelva los cuatro conjuntos de datos de Cifar 10, con cualquier preprocesamiento que fuera necesario (ten en cuenta que en el ejemplo comentado en la sección de formato de entrega se cargan los datos pero no se preprocesan), y explicarás en la memoria en qué consiste ese preprocesamiento y por qué es necesario o conveniente. También una o varias funciones más que muestren los siguientes resultados a partir de las predicciones del modelo de esa tarea y las clases verdaderas:

1. Las gráficas 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 pérdida y la tasa de acierto finales con el conjunto de test, y el tiempo empleado en el entrenamiento (en Keras, esto lo conseguimos con la función *evaluate*, pasándole el conjunto de test), utilizando una gráfica de barras 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).

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

En la primera tarea de esta práctica 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.

El primer paso será crear un fichero .py con vuestro nombre y apellidos si no lo has hecho ya, que será el que usarás en la entrega, y programar la función para cargar los datos de Cifar10 y aplicarle cualquier preprocesamiento que sea necesario para que un MLP pueda utilizar esos datos, además de documentar la utilidad o necesidad de ese preprocesamiento.

El primer MLP que declararás ha de tener una sola capa oculta de tipo Dense con 32 neuronas y función de activación *sigmoid*, y una capa de salida con el número de neuronas que sea conveniente para nuestro problema de clasificación, función de activación *softmax*, validation_split = 0.1, y que emplee como 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`, la función de pérdida (`loss`), y las métricas que se mostrarán (`metrics`). 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, que devolverá 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 (lo veremos en la siguiente tarea).

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

En la documentación has de explicar con tus palabras, de forma que se entienda el funcionamiento concreto, qué es un MLP y en qué fundamentos se basa para "aprender". Puedes apoyarte en cualquier fuente, incluido conversaciones con chatbots, pero debes demostrar que has entendido los conceptos que utilices.

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.

Tarea B: Ajustar el valor del parámetro epochs

En esta tarea analizaremos la evolución del entrenamiento de tu modelo para detectar si no se ha entrenado suficientemente o si 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 prueba. Para ello necesitaremos ver las curvas de train_accuracy, train_loss, validation_accuracy y validation_loss, y para eso emplearemos Matplotlib. En la primera sesión vimos un ejemplo con una función plot_curva que dibuja una línea a partir de los puntos de una lista (Y), que nos puede servir para esta tarea. 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.

Explica en la documentación qué es el sobreentrenamiento, por qué puede ocurrir, por qué se debe evitar, y qué es el número de épocas en un MLP.

OPTATIVO: Utilizando *callbacks* de Keras (https://keras.io/api/callbacks/), haz que tu MLP detecte sobreentrenamiento de forma automática y corte el entrenamiento antes de alcanzar el máximo de épocas.

Tarea C: Ajustar el valor del parámetro batch_size

En esta tarea debes documentarte (y explicar en la memoria) 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 comprobarás experimentalmente los efectos que tiene sobre la velocidad y el éxito del entrenamiento de un MLP.

Entrena y evaluá varios MLP con diferentes valores de batch_size. Discute si deberías ajustar un número de epochs diferente para cada MLP. Y 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.

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.

Sesión 3: MLP de Keras (ajuste de parámetros)

Tarea D: Probar diferentes funciones de activación

Al añadir una capa Dense a un MLP debemos 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, para elegir las dos o tres que puedan ser mejores que sigmoid en el contexto de esta práctica. 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 la tarea anterior, para elegir la mejor función de activación con este problema.

Tarea E: Ajustar el número de neuronas

Entrena y evalúa varios MLP con diferente número de neuronas en su capa oculta, discute los resultados y selecciona el modelo que consiga la mejor test_acc sin desperdiciar tiempo. Explica qué efectos tiene incrementar o decrementar el número de neuronas de un MLP.

Tarea F: Optimizar un MLP de dos o más capas

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.

Sesión 4: CNN de Keras

En las siguientes tareas, muy parecidas a las anteriores, volverás a clasificar las imágenes de Cifar10 con redes neuronales, pero esta vez utilizando redes convolucionales (CNN, de *convolutional neural networks*), cuyas "neuronas" (unidades) aplican filtros de convolución 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 G: Definir, entrenar y evaluar un CNN sencillo con Keras

Para completar esta tarea 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.

Para empezar a experimentar, declara una CNN con dos capas Conv2D de 16 y 32 filtros respectivamente, ambas con activación ReLu y filtros de 3x3, y como capa de salida una capa Dense con función de activación softmax. Deja el resto de parámetros en sus valores por defecto. Como hiciste en tareas anteriores, apóyate en gráficas para ajustar el valor adecuado de épocas si no utilizas *callbacks* de *Early Stopping*.

Ahora añade una capa MaxPooling2D con tamaño de filtro 2x2 después de cada capa Conv2 (similar al código en el ejemplo enlazado al principio de esta página), ajusta el número de épocas y compara los resultados entre ambos modelos de CNN (con y sin capas MaxPooling2d). En la memoria, argumenta cuál es mejor, y explica también los fundamentos de funcionamiento de una CNN, y en qué consiste una capa Conv2D y una capa MaxPooling2D.

Tarea H: Ajustar el parámetro kernel_size de la CNN

Como en el resto de tareas, ahora debes entrenar varios modelos de tu CNN con diferente tamaño de filtros en las capas Conv2D, analiza sus resultados apoyándote en gráficas y selecciona 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 es más costoso.

OPTATIVO: Existe una relación entre el tamaño de los datos de entrada y de los filtros que influye en la capacidad de detección de características de la red. Vuelve a discutir si siempre es conveniente o necesario incluir capas MaxPooling2D, y demuestra tus conclusiones apoyándote en tu experimentación.

Tarea I: Optimizar la arquitectura del modelo

En esta tarea tu objetivo es mejorar la tasa de acierto de test sin emplear un tiempo de entrenamiento excesivo (lo que es excesivo o no dependerá de ti, razónalo en la documentación). Prueba a declarar nuevos modelos de CNN con diferente número de capas Conv2D y Dense, con diferente número de neuronas o filtros. 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.

Recuerda: Entrega de la primera parte hasta el **30 de noviembre de 2024 a las 23:55h**.

La **no entrega** de este hito supone hasta un **20% de penalización** en la entrega final.

Sesión 5: Crear dataset propio para validar la generalización

En la primera parte de la práctica hemos trabajado con librerías que traen implementados muchos métodos y clases útiles para entrenar redes neuronales, lo que ahorra mucho tiempo de implementación y depuración.

En cuanto a los datos, hemos utilizado el dataset Cifar10, ya preparado por Keras para ser utilizado por las redes neuronales que hemos definido.

En esta segunda parte de la práctica deberás crear un conjunto de datos propio con las mismas categorías que el dataset original, adaptarlo para poder realizar predicciones sobre las imágenes y comprobar, de manera más realista, la capacidad de generalización de los modelos obtenidos en la primera parte de la práctica.

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

En esta tarea se propone que generes un conjunto de imágenes de prueba para evaluar la generalización de la red. Este conjunto de imágenes debe tener el mismo formato que el dataset Cifar10, por lo que serán imágenes de color de 32x32 píxeles. Deberás recopilar 15 imágenes por cada una de las 10 categorías del dataset: avión, automóvil, pájaro, gato, ciervo, perro, rana, caballo, barco y camión.

Una vez construido el dataset, deberás asegurarte de realizar el preprocesamiento necesario para que se cargue adecuadamente en la red. Keras utiliza la librería PIL para la manipulación de imágenes, por lo que deberás utilizar las instrucciones adecuadas para realizar el *downsampling* de las imágenes.

Sesión 6: Experimentación sobre el dataset propio

En esta sesión vas a realizar la experimentación necesaria para comprobar la capacidad de generalización real de los modelos entrenados con Cifar10, aplicándolos a las imágenes de tu dataset.

Tarea K: Resultados y experimentación

En esta tarea deberás realizar una batería de pruebas sobre los distintos parámetros configurables de la red, que ya has probado en la primera parte de la práctica, y deberás calcular las métricas correspondientes de precisión para evaluar su desempeño sobre el dataset generado.

Algunos de los parámetros que deberás analizar son los siguientes:

- Tamaño y profundidad de las capas ocultas.
- Funciones de activación de las capas ocultas.
- Optimizadores y learning rate del entrenamiento.
- Número de iteraciones y tamaño del batch.
- Tamaño de los filtros en capas Conv2D.

Una vez realizado el estudio, mostrando gráficas y comentando los resultados, deberás indicar cuál es la mejor configuración obtenida, comparando los resultados en forma de tabla resumen, y justificando adecuadamente dicha decisión.

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.

Tarea L (OPTATIVA): Mejoras para favorecer la generalización

En esta tarea puedes incluir diferentes mejoras sobre el entrenamiento de las redes con el objetivo de aumentar su capacidad de generalización.

Algunas de las mejoras a incluir en esta sección pueden ser las siguientes:

- Métodos de agrupación (*pooling*): https://keras.io/api/layers/pooling_layers/
- 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 los datos: https://keras.io/api/layers/normalization_layers/
- Técnicas de inicialización inteligente de los pesos de la red: https://keras.io/api/layers/initializers/

Las mejoras que propongas deberán estar fundamentadas, implementadas y evaluadas convenientemente para que puedan ser tenidas en consideración.

Sesión 7: Resolución de dudas y revisión de la documentación

En esta sesión resolveremos dudas que puedan quedar sobre la implementación de cualquiera de los apartados antes de la entrega final, y repasaremos las cuestiones que se han de tratar y las que no en la documentación.

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).