

REDES NEURONALES

En esta actividad vamos a utilizar una red neuronal para clasificar imágenes de dígitos del 0 al 9 escritos a mano. Para ello, utilizaremos Keras con TensorFlow.

El dataset a utilizar es MNIST, una base de datos constituida por (como no) imágenes de dígitos escritos a mano. Este dataset es ampliamente utilizado en docencia como punto de entrada al entrenamiento de redes neuronales y otros, pero también es muy utilizado en trabajos reales de investigación para el entrenamiento de imágenes. Puedes consultar más información sobre el dataset en [este enlace](#).

El código utilizado para contestar tiene que quedar claramente reflejado en el Notebook. Puedes crear nuevas celdas si así lo deseas para estructurar tu código y sus salidas. A la hora de entregar el notebook, **asegúrate de que los resultados de ejecutar tu código han quedado guardados y que son perfectamente visibles en la versión PDF que debes entregar adjunta**. Por ejemplo, a la hora de entrenar una red neuronal tiene que verse claramente un log de los resultados de cada epoch.

```
In [1]: %pip install keras  
%pip install tensorflow  
%pip install numpy  
%pip install matplotlib  
%pip install plotly  
%pip install nbformat  
%pip install pandas  
%pip install keras-flops  
%pip install seaborn  
%pip install scikit-learn
```

Requirement already satisfied: keras in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (3.6.0)
Requirement already satisfied: absl-py in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras) (2.1.0)
Requirement already satisfied: numpy in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras) (1.26.4)
Requirement already satisfied: rich in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras) (13.9.2)
Requirement already satisfied: namex in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras) (0.0.8)
Requirement already satisfied: h5py in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras) (3.12.1)
Requirement already satisfied: optree in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras) (0.13.0)
Requirement already satisfied: ml-dtypes in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras) (0.4.1)
Requirement already satisfied: packaging in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras) (24.1)
Requirement already satisfied: typing-extensions>=4.5.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from optree->keras) (4.12.2)
Requirement already satisfied: markdown-it-py>=2.2.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from rich->keras) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from rich->keras) (2.18.0)
Requirement already satisfied: mdurl=>0.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from markdown-it-py>=2.2.0->rich->keras) (0.1.2)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: tensorflow in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!0.5.1,!0.5.2,>=0.2.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (4.25.5)
Requirement already satisfied: requests<3,>=2.21.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (2.32.3)
Requirement already satisfied: setuptools in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (1.67.0)
Requirement already satisfied: tensorflowboard<2.19,>=2.18 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (3.6.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (3.12.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from astunparse>=1.6.0->tensorflow) (0.44.0)
Requirement already satisfied: rich in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras>=3.5.0->tensorflow) (13.9.2)
Requirement already satisfied: namex in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras>=3.5.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras>=3.5.0->tensorflow) (0.13.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow) (2.2.3)

Requirement already satisfied: certifi>=2017.4.17 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<2.19,>=2.18->tensorflow) (3.7)
Requirement already satisfied: tensorflow-data-server<0.8.0,>=0.7.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<2.19,>=2.18->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<2.19,>=2.18->tensorflow) (3.0.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from werkzeug>=1.0.1->tensorflow<2.19,>=2.18->tensorflow) (3.0.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from rich->keras>=3.5.0->tensorflow) (3.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from rich->keras>=3.5.0->tensorflow) (2.18.0)
Requirement already satisfied: mdurl~0.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow) (0.1.2)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: numpy in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (1.26.4)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: matplotlib in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (1.4.7)
Requirement already satisfied: numpy>=1.23 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: six>=1.5 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: plotly in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (5.24.1)
Requirement already satisfied: tenacity>=6.2.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from plotly) (9.0.0)
Requirement already satisfied: packaging in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from plotly) (24.1)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: nbformat in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (5.10.4)
Requirement already satisfied: fastjsonschema>=2.15 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from nbformat) (2.20.0)
Requirement already satisfied: jsonschema>=2.6 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from nbformat) (4.23.0)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from nbformat) (5.7.2)
Requirement already satisfied: traitlets>=5.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from nbformat) (5.14.3)
Requirement already satisfied: attrs>=22.2.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat) (24.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat) (2024.10.1)
Requirement already satisfied: referencing>=0.28.4 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat) (0.35.1)
Requirement already satisfied: rpdps-py>=0.7.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from jsonschema>=2.6->nbformat) (0.20.0)
Requirement already satisfied: platformdirs>=2.5 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from jupyter-core!=5.0.*,>=4.12->nbformat) (4.3.6)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: pandas in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (2.2.3)
Requirement already satisfied: numpy>=1.22.4 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from pandas) (2024.2)
Requirement already satisfied: six>=1.5 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: keras-flops in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (0.1.2)
Requirement already satisfied: tensorflow<3.0,>=2.2 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python

```
3.10/site-packages (from keras-flops) (2.18.0)
Requirement already satisfied: absl-py>=1.0.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (2.1.0)
Requirement already satisfied: astunparse>=1.6.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!0.5.1,!0.5.2,>0.2.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (3.4.0)
Requirement already satisfied: packaging in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (24.1)
Requirement already satisfied: protobuf!=4.21.0,!4.21.1,!4.21.2,!4.21.3,!4.21.4,!4.21.5,<6.0.0dev,>=3.20.3 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (4.25.5)
Requirement already satisfied: requests<3,>=2.21.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (2.32.3)
Requirement already satisfied: setuptools in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (1.67.0)
Requirement already satisfied: tensorflow<2.19,>=2.18 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (2.18.0)
Requirement already satisfied: keras>=3.5.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (3.6.0)
Requirement already satisfied: numpy<2.1.0,>=1.26.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (1.26.4)
Requirement already satisfied: h5py>=3.11.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (3.12.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.4.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (0.4.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<3.0,>=2.2->keras-flops) (0.37.1)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from astunparse>=1.6.0->tensorflow<3.0,>=2.2->keras-flops) (0.44.0)
Requirement already satisfied: rich in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras>=3.5.0->tensorflow<3.0,>=2.2->keras-flops) (13.9.2)
Requirement already satisfied: namex in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras>=3.5.0->tensorflow<3.0,>=2.2->keras-flops) (0.0.8)
Requirement already satisfied: optree in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from keras>=3.5.0->tensorflow<3.0,>=2.2->keras-flops) (0.13.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow<3.0,>=2.2->keras-flops) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow<3.0,>=2.2->keras-flops) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow<3.0,>=2.2->keras-flops) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from requests<3,>=2.21.0->tensorflow<3.0,>=2.2->keras-flops) (2024.8.30)
Requirement already satisfied: markdown>=2.6.8 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<2.19,>=2.18->tensorflow<3.0,>=2.2->keras-flops) (3.7)
Requirement already satisfied: tensorflow-data-server<0.8.0,>=0.7.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<2.19,>=2.18->tensorflow<3.0,>=2.2->keras-flops) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from tensorflow<2.19,>=2.18->tensorflow<3.0,>=2.2->keras-flops) (3.0.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from werkzeug>=1.0.1->tensorflow<2.19,>=2.18->tensorflow<3.0,>=2.2->keras-flops) (3.0.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from rich->keras>=3.5.0->tensorflow<3.0,>=2.2->keras-flops) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from rich->keras>=3.5.0->tensorflow<3.0,>=2.2->keras-flops) (2.18.0)
Requirement already satisfied: mdurl=>0.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from markdown-it-py>=2.2.0->rich->keras>=3.5.0->tensorflow<3.0,>=2.2->keras-flops) (0.1.2)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: seaborn in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from seaborn) (1.26.4)
Requirement already satisfied: pandas>=1.2 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from seaborn) (2.2.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from seaborn) (3.9.2)
Requirement already satisfied: contourpy>=1.0.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.3.0)
```

```

Requirement already satisfied: cycler>=0.10 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (24.1)
Requirement already satisfied: pillow>=8 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (11.0.0)
Requirement already satisfied: pyarsing>=2.3.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from pandas>=1.2->seaborn) (2024.2)
Requirement already satisfied: six>=1.5 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
Requirement already satisfied: scikit-learn in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (1.5.2)
Requirement already satisfied: numpy>=1.19.5 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from scikit-learn) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in /home/mortegad/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages (from scikit-learn) (3.5.0)
Note: you may need to restart the kernel to use updated packages.

```

```
In [2]: import os
os.environ["CUDA_VISIBLE_DEVICES"] = "-1"
import tensorflow as tf
import matplotlib.pyplot as plt

import logging
logging.getLogger('tensorflow').disabled = True
```

```

2024-10-27 18:32:45.410816: I tensorflow/core/util/port.cc:153] oneDNN custom operations are on. You may see slightly different numerical results due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2024-10-27 18:32:45.412131: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-10-27 18:32:45.413473: I external/local_xla/xla/tsl/cuda/cudart_stub.cc:32] Could not find cuda drivers on your machine, GPU will not be used.
2024-10-27 18:32:45.419598: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1730079165.428545 395641 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register factory for plugin cuDNN when one has already been registered
E0000 00:00:1730079165.431497 395641 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register factory for plugin cubLAS when one has already been registered
2024-10-27 18:32:45.441392: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.

```

Tenemos la suerte de que el dataset MNIST, el que vamos a utilizar en esta actividad, está guardado en Keras, por lo que podemos utilizarlo sin necesidad de buscar el dataset de forma externa.

```
In [3]: mnist = tf.keras.datasets.mnist
```

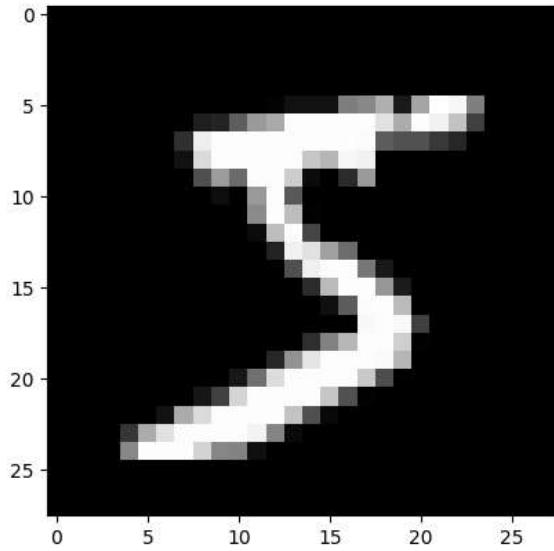
Llamar a **load_data** en este dataset nos dará dos conjuntos de dos listas, estos serán los valores de entrenamiento y prueba para los gráficos que contienen los dígitos y sus etiquetas.

Nota: Aunque en esta actividad lo veis de esta forma, también lo vais a poder encontrar como 4 variables de esta forma: training_images, training_labels, test_images, test_labels = mnist.load_data()

```
In [4]: (training_images, training_labels), (test_images, test_labels) = mnist.load_data()
```

Antes de continuar vamos a dar un vistazo a nuestro dataset, para ello vamos a ver una imagen de entrenamiento y su etiqueta o clase.

```
In [5]: import numpy as np
np.set_printoptions(linewidth=200)
plt.imshow(training_images[0], cmap="gray") # recordad que siempre es preferible trabajar en blanco y negro
#
print(training_labels[0])
print(training_images[0])
```



1. Información sobre el dataset

Una vez tenemos los datos cargados en memoria, vamos a obtener información sobre los mismos.

Pregunta 1.1 (0.25 puntos)* ¿Cuántas imágenes hay de "training" y de "test"? ¿Qué tamaño tienen las imágenes?

```
In [6]: # Conjunto de Training
print(f"Numero total de etiquetas para training:{len(training_labels)}")
print(f"Numero total de imagenes para training:{len(training_images)}")

#Conjunto de Testing
print(f"Numero total de etiquetas para testing:{len(test_labels)}")
print(f"Numero total de imagenes para testing:{len(test_images)}")

#Size de imagenes, tomo la primera y la analizo

imagen_ejemplo=training_images[0]
altura = len(imagen_ejemplo)
anchura = len(imagen_ejemplo[0])

print(f"Altura de las imagenes:{altura} pixeles")
print(f"Anchura de las imagenes:{anchura} pixeles")
```

```
Numero total de etiquetas para training:60000
Numero total de imagenes para training:60000
Numero total de etiquetas para testing:10000
Numero total de imagenes para testing:10000
Altura de las imagenes:28 pixeles
Anchura de las imagenes:28 pixeles
```

Tenemos 70000 en total, 60000 en el conjunto de training y 10000 en el conjunto de test. Considerando todas las imagenes con el mismo sizing, las imagenes sonde 28x28 pixeles.

Pregunta 1.2 *(0.25 puntos) Realizar una exploración de las variables que contienen los datos. Describir en qué consiste un example del dataset (qué información se guarda en cada imagen) y describir qué contiene la información en y.

```
In [7]: # Visualizamos Las primeras 10 imagenes (examples).
```

```
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(training_images[i], cmap='gray')
    plt.title(f"Etiqueta: {training_labels[i]}")
    plt.axis('off')
plt.show()
```

Etiqueta: 5 Etiqueta: 0 Etiqueta: 4 Etiqueta: 1 Etiqueta: 9



Etiqueta: 2 Etiqueta: 1 Etiqueta: 3 Etiqueta: 1 Etiqueta: 4



Cada example del dataset contiene: .- Un tensor 28x28 con los valores de los pixeles en escala de grises desde 0 - 255, esta matriz representa el trazo a mano de un digito. .- Una etiqueta que corresponde con el digito representado en la imagen (0-9)

2. Normalización y preprocesado de los datos

Pregunta 2.1 (0.25 puntos) Habreis notado que todos los valores numericos están entre 0 y 255. Si estamos entrenando una red neuronal, una buena practica es transformar todos los valores entre 0 y 1, un proceso llamado "normalización" y afortunadamente en Python es fácil normalizar una lista. ¿Cómo lo podemos hacer?

```
In [8]: training_images = training_images.astype('float32')/255.0
test_imagenes = test_images.astype('float32')/255.0
```

Pregunta 2.2 (0.25 puntos) Utiliza la función **reshape** de Numpy para convertir las imágenes en vectores de características de un tamaño de (N, 784). Explica con tus palabras por qué es necesario hacer esto.

```
In [9]: training_images = training_images.reshape(len(training_images),784)
test_imagenes = test_imagenes.reshape(len(test_imagenes),784)
```

En la practica vamos a entrenar un modelo MLP que toma de entrada un vector de una sola dimension, de esta forma estamos trasformando cada una de las imagenes de 28x28 pixeles en un vector de 784 valores. N en el caso de las imagenes de training es 60000 y en el de test es 10000.

Pregunta 2.3 (0.25 puntos) Para facilitar el desarrollo de la actividad, vamos a expresar las etiquetas así:

```
In [10]: training_labels = tf.keras.utils.to_categorical(training_labels)
test_labels = tf.keras.utils.to_categorical(test_labels)
```

Muestra cómo son ahora los datos, como resultado de este cambio y tambien de los realizados en las dos preguntas anteriores. Debate cómo se beneficiará la red neuronal de todos estos cambios.

```
In [11]: print(training_labels.shape)
print(training_labels[0])
```

```
(60000, 10)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Con esta transformacion estamos realizando lo que se conoce como one-hot encoding. Nuestra red neuronal tiene 10 neuronas de salida. La salida de cada una de ellas representara la probabilidad de que la imagen pasada como vector pertenezca a una de las 10 categorias. Al

representar las salidas de esta forma, mejoramos el calculo de la funcion de perdida y las metricas de validacion que vamos a emplear.

3. Creación del Modelo

Ahora vamos a definir el modelo, pero antes vamos a repasar algunos comandos y conceptos muy útiles:

- **Sequential:** Eso define una SECUENCIA de capas en la red neuronal
- **Dense:** Añade una capa de neuronas
- **Flatten:** ¿Recuerdas cómo eran las imágenes cuando las imprimiste para poder verlas? Un cuadrado, Flatten toma ese cuadrado y lo convierte en un vector de una dimensión.

Cada capa de neuronas necesita una función de activación. Normalmente se usa la función relu en las capas intermedias y softmax en la ultima capa (en problemas de clasificación de más de dos items)

- **Relu** significa que "Si $X > 0$ devuelve X , si no, devuelve 0", así que lo que hace es pasar sólo valores 0 o mayores a la siguiente capa de la red.
- **Softmax** toma un conjunto de valores, y escoge el más grande.

Pregunta 3.1 (0.5 puntos): Utilizando Keras, y preparando los datos de X e Y como fuera necesario, define y entrena una red neuronal que sea capaz de clasificar imágenes de MNIST con las siguientes características:

- Una capa de entrada del tamaño adecuado.
- Una capa oculta de 512 neuronas.
- Una capa final con 10 salidas.

```
In [12]: from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input

model = Sequential([
    Input(shape=(784,)),
    Dense(512),
    Dense(10, activation='softmax')
])
```

```
2024-10-27 18:32:48.678538: E external/local_xla/xla/stream_executor/cuda/cuda_driver.cc:152] failed call to cuInit: INTERNAL
L: CUDA error: Failed call to cuInit: UNKNOWN ERROR (303)
```

Pregunta 3.2 (0.25 puntos): ¿Crees conveniente utilizar una capa flatten en este caso? Motiva tu respuesta.

La capa Flatten tiene como objetivo la transformación de matrices de entrada multidimensionales en vectores unidimensionales. En nuestro caso ya hemos realizado esta transformación en el paso 2.2 por lo que no tiene sentido emplearla.

Respuesta a la pregunta 3.2:

Pregunta 3.3 (0.25 puntos): Utiliza la función summary() para mostrar la estructura de tu modelo.

```
In [13]: model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 512)	401,920
dense_1 (Dense)	(None, 10)	5,130

Total params: 407,050 (1.55 MB)

Trainable params: 407,050 (1.55 MB)

Non-trainable params: 0 (0.00 B)

4: Compilación y entrenamiento

Pregunta 4.1 (0.5 puntos): Compila tu modelo. Utiliza **categorical_crossentropy** como función de pérdida, **Adam** como optimizador, y monitoriza la **tasa de acierto** durante el entrenamiento. Explica qué hace cada cosa en la compilación.

```
In [14]: model.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])
```

.- Adam: estamos aplicando la función de optimización Adam al cálculo de los gradientes de cada peso. Esta optimización aplica los momentos de orden 1(m) y 2(v) a la tasa de aprendizaje ($\alpha \cdot [\beta m / \sqrt{v} + \epsilon]$). Esta técnica permite convergencias rápidas y la reducción de las oscilaciones. .- Categorical_crossentropy: emplearemos esta técnica para calcular la función de pérdida del modelo en la clasificación. Esta función evalúa la distancia entre las distribuciones de las etiquetas predichas y las reales. .- Accuracy: esta es la métrica que vamos a

emplear en la evaluacion del modelo, se calcula como el numero de predicciones correctas (TP+TN) dividido entre el numero total de predicciones (TP+TN+FN+FP).

Pregunta 4.2 (0.5 puntos): Utiliza la función `fit()` para entrenar tu modelo. Para ayudarte en tu primer entrenamiento, utiliza estos valores:

- epochs = 5
- batch_size = 32
- validation_split = 0.25

```
In [15]: import plotly.graph_objects as go
import numpy as np
import pandas as pd

def plot_history(history, model_name):
    # Extraer la información del objeto history
    acc = history.history['accuracy']
    val_acc = history.history.get('val_accuracy')
    loss = history.history['loss']
    val_loss = history.history.get('val_loss')

    # Definir las épocas en base a la longitud de los datos
    epochs = np.arange(1, len(acc) + 1)

    # Calcular diferencias absolutas entre entrenamiento y validación
    acc_diff = np.abs(np.array(acc) - np.array(val_acc)) if val_acc is not None else None
    loss_diff = np.abs(np.array(loss) - np.array(val_loss)) if val_loss is not None else None

    # Umbral para detectar sobreajuste
    acc_threshold = 0.05 # Diferencia del 5% en precisión
    loss_threshold = 0.1 # Diferencia del 10% en pérdida

    # Detectar sobreajuste en precisión y pérdida
    acc_overfit = (acc_diff > acc_threshold) if acc_diff is not None else None
    loss_overfit = (loss_diff > loss_threshold) if loss_diff is not None else None

    # Función para encontrar intersecciones
    def encontrar_intersecciones(x, y1, y2):
        intersecciones_x = []
        intersecciones_y = []
        for i in range(1, len(x)):
            if (y1[i-1] - y2[i-1]) * (y1[i] - y2[i]) <= 0: # Cambio de signo
                intersecciones_x.append(x[i])
                intersecciones_y.append((y1[i] + y2[i]) / 2)
        return intersecciones_x, intersecciones_y

    # Encontrar intersecciones entre las curvas de accuracy
    acc_inter_x, acc_inter_y = encontrar_intersecciones(epochs, acc, val_acc)

    # Encontrar intersecciones entre las curvas de loss
    loss_inter_x, loss_inter_y = encontrar_intersecciones(epochs, loss, val_loss)

    # --- Primer gráfico: curvas de exactitud y pérdida ---
    fig_curvas = go.Figure()

    # Agregar líneas de accuracy
    fig_curvas.add_trace(go.Scatter(x=epochs, y=acc, mode='lines', name='Exactitud de Entrenamiento', line=dict(color='blue'))
    fig_curvas.add_trace(go.Scatter(x=epochs, y=val_acc, mode='lines', name='Exactitud de Validación', line=dict(color='green'))

    # Agregar intersecciones de accuracy
    fig_curvas.add_trace(go.Scatter(x=acc_inter_x, y=acc_inter_y, mode='markers', name='Intersección de Exactitud', marker=dict(color='red', size=100))

    # Agregar líneas de pérdida
    fig_curvas.add_trace(go.Scatter(x=epochs, y=loss, mode='lines', name='Pérdida de Entrenamiento', line=dict(color='red'))
    fig_curvas.add_trace(go.Scatter(x=epochs, y=val_loss, mode='lines', name='Pérdida de Validación', line=dict(color='orange'))

    # Agregar intersecciones de loss
    fig_curvas.add_trace(go.Scatter(x=loss_inter_x, y=loss_inter_y, mode='markers', name='Intersección de Pérdida', marker=dict(color='blue', size=100))

    # Configurar el primer gráfico con tamaño reducido
    fig_curvas.update_layout(
        title=f'Curvas de Exactitud y Pérdida ({model_name})',
        xaxis_title='Épocas',
        yaxis_title='Valor',
        hovermode='x unified',
        template='seaborn',
        xaxis=dict(range=[epochs[0], epochs[-1]]), # Sincronizar el rango del eje x
        width=700, # Ancho reducido
        height=500 # Altura estándar
    )

    # --- Segundo gráfico: diferencias en barras ---
    fig_diferencias = go.Figure()

    if acc_diff is not None and loss_diff is not None:
        # Trazas para las diferencias de precisión
```

```

        fig_diferencias.add_trace(go.Bar(
            x=epochs,
            y=acc_diff,
            name='Diferencia de Exactitud',
            marker_color='lightblue',
            opacity=0.6
        ))

        # Trazas para las diferencias de pérdida
        fig_diferencias.add_trace(go.Bar(
            x=epochs,
            y=loss_diff,
            name='Diferencia de Pérdida',
            marker_color='lightcoral',
            opacity=0.6
        ))

    # Configurar el segundo gráfico con tamaño reducido
    fig_diferencias.update_layout(
        title=f'Diferencias entre Entrenamiento y Validación ({model_name})',
        xaxis_title='Épocas',
        yaxis_title='Diferencia',
        hovermode='x unified',
        template='seaborn',
        xaxis=dict(range=[epochs[0], epochs[-1] + 1]), # Sincronizar el rango del eje x
        width=700, # Ancho reducido
        height=500 # Altura estándar
    )

    # Mostrar los gráficos
    fig_curvas.show()
    fig_diferencias.show()

# Almaceno los pesos iniciales para experimentos posteriores
pesos_iniciales = model.get_weights()

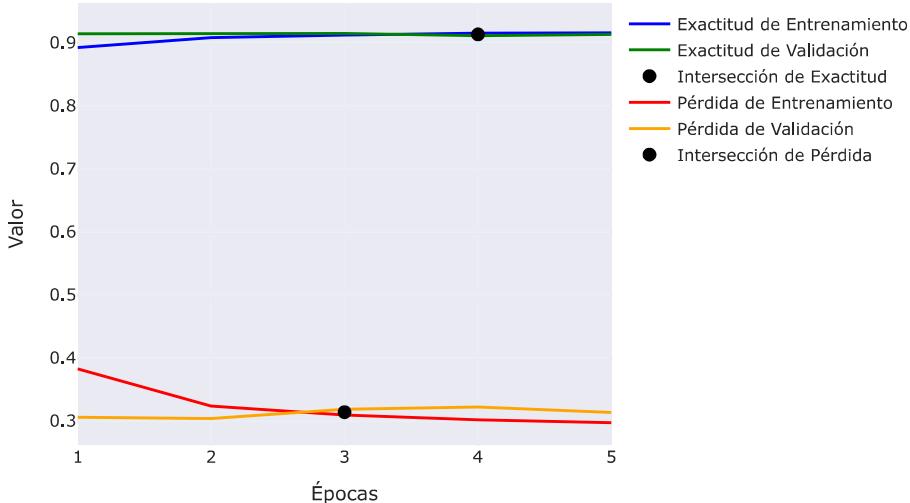
history = model.fit(training_images, training_labels,
                     epochs=5,
                     batch_size=32,
                     validation_split=0.25)

plot_history(history,"Modelo 1-512")

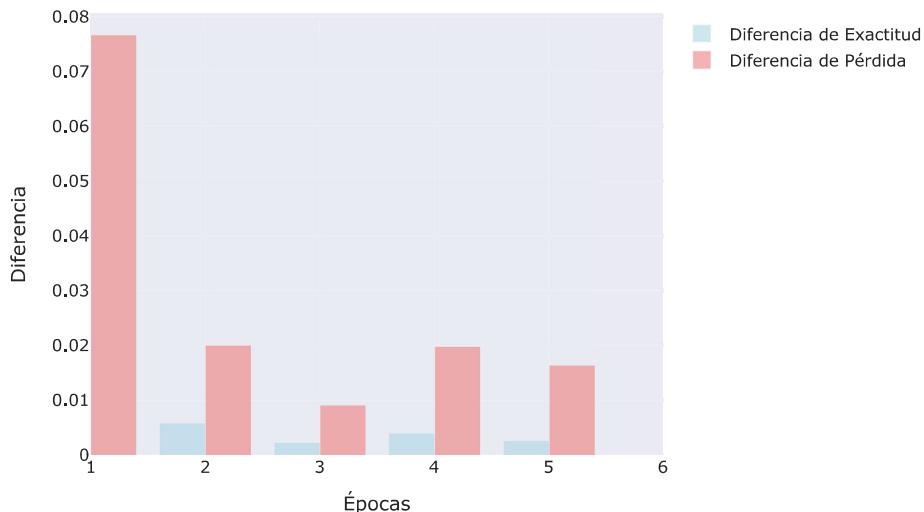
```

Epoch 1/5
1407/1407 4s 3ms/step - accuracy: 0.8665 - loss: 0.4636 - val_accuracy: 0.9139 - val_loss: 0.3058
Epoch 2/5
1407/1407 4s 3ms/step - accuracy: 0.9105 - loss: 0.3134 - val_accuracy: 0.9139 - val_loss: 0.3034
Epoch 3/5
1407/1407 6s 4ms/step - accuracy: 0.9117 - loss: 0.3121 - val_accuracy: 0.9145 - val_loss: 0.3183
Epoch 4/5
1407/1407 4s 3ms/step - accuracy: 0.9135 - loss: 0.3054 - val_accuracy: 0.9109 - val_loss: 0.3216
Epoch 5/5
1407/1407 4s 3ms/step - accuracy: 0.9178 - loss: 0.2946 - val_accuracy: 0.9129 - val_loss: 0.3134

Curvas de Exactitud y Pérdida (Modelo 1-512)



Diferencias entre Entrenamiento y Validación (Modelo 1-512)



5: Impacto al variar el número de neuronas en las capas ocultas

En este ejercicio vamos a experimentar con nuestra red neuronal cambiando el numero de neuronas por 512 y por otros valores. Para ello, utiliza la red neuronal de la pregunta 3, y su capa oculta cambia el número de neuronas:

- **216 neuronas en la capa oculta
- **1024 neuronas en la capa oculta

y entrena la red en ambos casos.

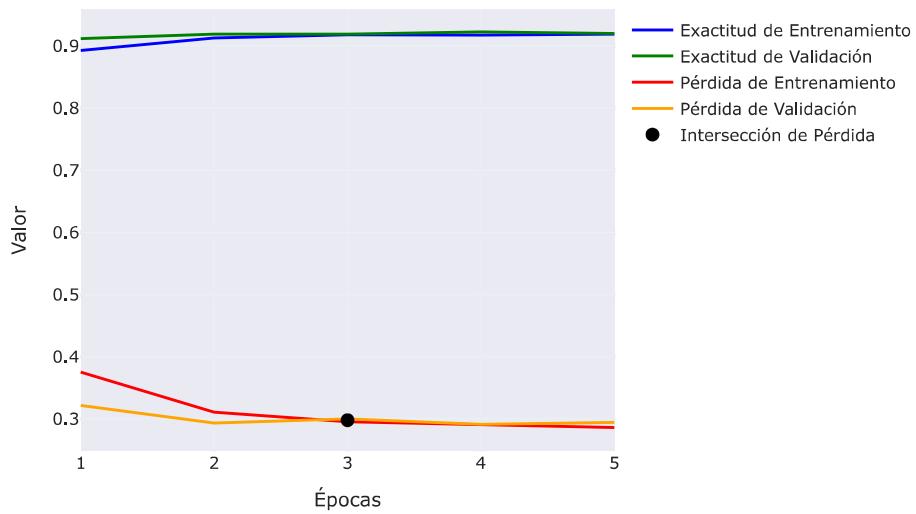
```
In [16]: model_2 = Sequential([
    Input(shape=(784,)),
    Dense(216),
    Dense(10,activation='softmax')
])
model_2.compile(optimizer='adam',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])

history_2 = model_2.fit(training_images, training_labels,
                        epochs=5,
                        batch_size=32,
                        validation_split=0.25)

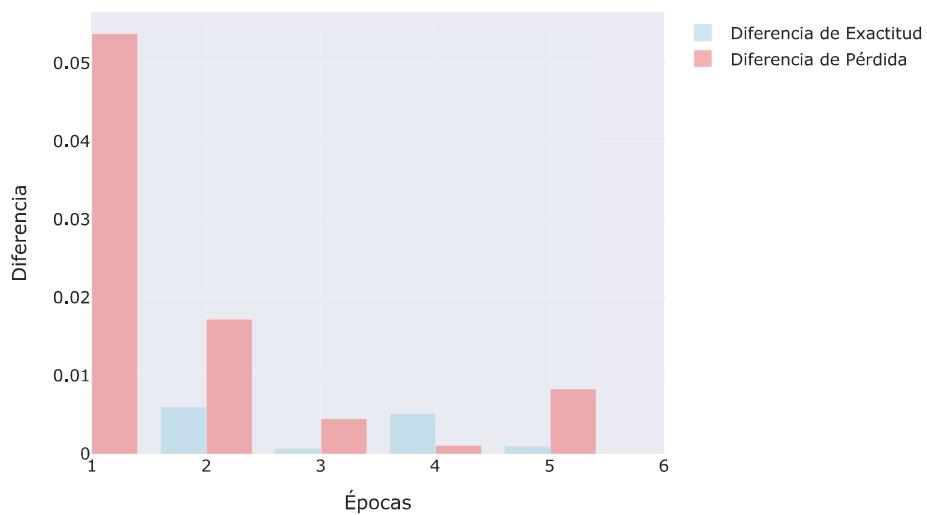
plot_history(history_2,"Modelo 2-216")
```

Epoch 1/5
1407/1407 3s 2ms/step - accuracy: 0.8563 - loss: 0.4858 - val_accuracy: 0.9118 - val_loss: 0.3216
Epoch 2/5
1407/1407 3s 2ms/step - accuracy: 0.9142 - loss: 0.3045 - val_accuracy: 0.9190 - val_loss: 0.2936
Epoch 3/5
1407/1407 3s 2ms/step - accuracy: 0.9174 - loss: 0.2974 - val_accuracy: 0.9189 - val_loss: 0.3003
Epoch 4/5
1407/1407 3s 2ms/step - accuracy: 0.9191 - loss: 0.2884 - val_accuracy: 0.9226 - val_loss: 0.2915
Epoch 5/5
1407/1407 3s 2ms/step - accuracy: 0.9215 - loss: 0.2819 - val_accuracy: 0.9200 - val_loss: 0.2946

Curvas de Exactitud y Pérdida (Modelo 2-216)



Diferencias entre Entrenamiento y Validación (Modelo 2-216)

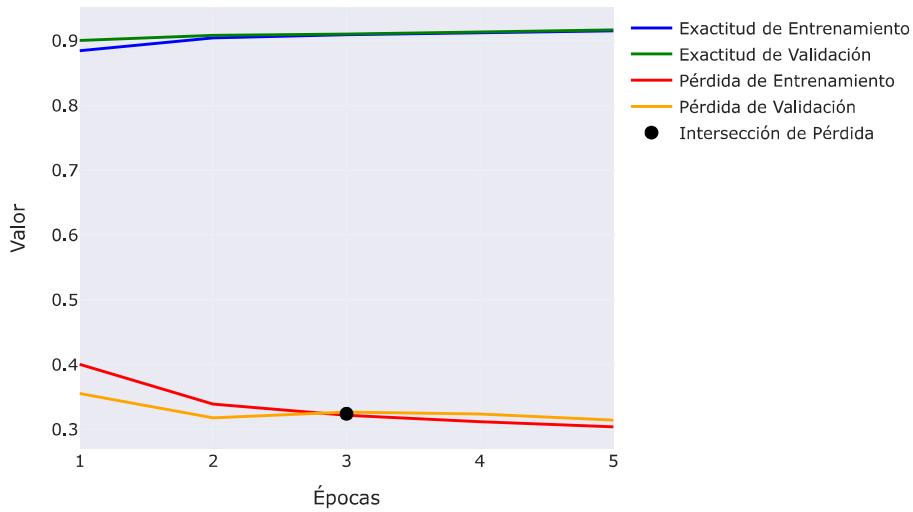


```
In [17]: model_3 = Sequential([
    Input(shape=(784,)),
    Dense(1024),
    Dense(10,activation='softmax')
])
model_3.compile(optimizer='adam',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
history_3 = model_3.fit(training_images, training_labels,
                        epochs=5,
                        batch_size=32,
                        validation_split=0.25)

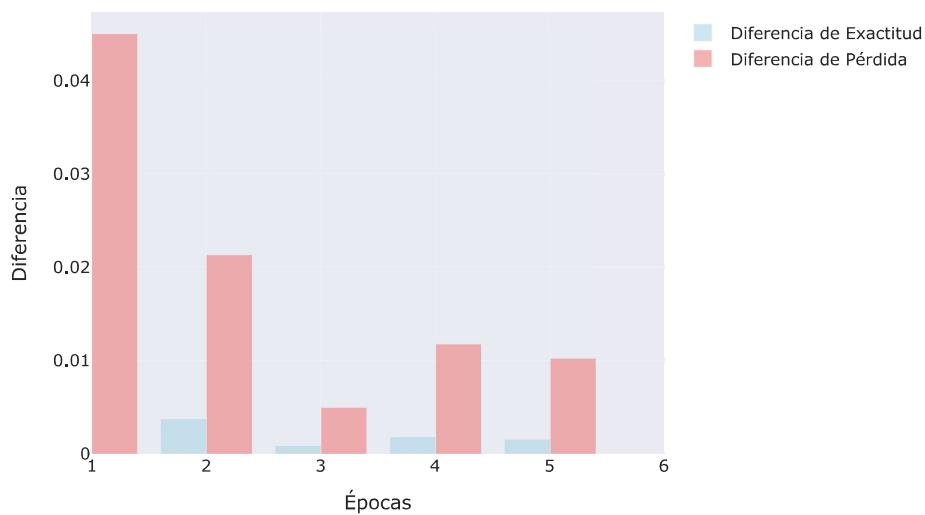
plot_history(history_3,"Modelo 3-1024")
```

Epoch 1/5
1407/1407 5s 4ms/step - accuracy: 0.8605 - loss: 0.4767 - val_accuracy: 0.9007 - val_loss: 0.3556
Epoch 2/5
1407/1407 8s 5ms/step - accuracy: 0.9054 - loss: 0.3319 - val_accuracy: 0.9084 - val_loss: 0.3179
Epoch 3/5
1407/1407 5s 4ms/step - accuracy: 0.9109 - loss: 0.3183 - val_accuracy: 0.9103 - val_loss: 0.3268
Epoch 4/5
1407/1407 5s 3ms/step - accuracy: 0.9105 - loss: 0.3146 - val_accuracy: 0.9135 - val_loss: 0.3239
Epoch 5/5
1407/1407 5s 4ms/step - accuracy: 0.9117 - loss: 0.3116 - val_accuracy: 0.9167 - val_loss: 0.3144

Curvas de Exactitud y Pérdida (Modelo 3-1024)



Diferencias entre Entrenamiento y Validación (Modelo 3-1024)



```
In [18]: def plot_comparacion_modelos(historiales, nombres_modelos, resultados_prueba):
    # Verificar que las listas tengan el mismo tamaño
    #if len(historiales) != len(nombres_modelos) or len(historiales) != len(resultados_prueba):
    #    raise ValueError("El número de historiales, nombres de modelos y resultados de prueba debe coincidir.")

    # Inicializar listas para almacenar los valores de la última época
    exactitud_entrenamiento = []
    exactitud_validacion = []
    perdida_entrenamiento = []
    perdida_validacion = []
    exactitud_prueba = []
    perdida_prueba = []

    # Recorrer los historiales y extraer los valores de la última época y los resultados de prueba
    for historial, resultado_prueba in zip(historiales, resultados_prueba):
        prueba_perdida, prueba_exactitud = resultado_prueba[0], resultado_prueba[1]

        exactitud = historial.history['accuracy']
        exactitud_val = historial.history.get('val_accuracy')
        perdida = historial.history['loss']
        perdida_val = historial.history.get('val_loss')

        ultima_epoca = len(exactitud) - 1

        # Guardar los valores de la última época y redondearlos al tercer decimal
        #exactitud_entrenamiento.append(round(exactitud[ultima_epoca], 3))
        #exactitud_validacion.append(round(exactitud_val[ultima_epoca], 3) if exactitud_val is not None else 'N/A')
```

```

#perdida_entrenamiento.append(round(perdida[ultima_epoca], 3))
#perdida_validacion.append(round(perdida_val[ultima_epoca], 3) if perdida_val is not None else 'N/A')

# Guardar los valores de la última época y redondearlos al tercer decimal
exactitud_entrenamiento.append(round(np.mean(exactitud), 3))
exactitud_val.append(round(np.mean(exactitud_val), 3) if exactitud_val is not None else 'N/A')
perdida_entrenamiento.append(round(np.mean(perdida), 3))
perdida_val.append(round(np.mean(perdida_val), 3) if perdida_val is not None else 'N/A')
# Guardar los valores de precisión y pérdida en prueba
exactitud_prueba.append(round(prueba_exactitud, 3))
perdida_prueba.append(round(prueba_perdida, 3))

df = pd.DataFrame({
    'Modelo':nombres_modelos,
    'Media(Exactitud_entrenamiento)': exactitud_entrenamiento,
    'Media(Exactitud_validacion)': exactitud_val,
    'Media(Exactitud_prueba)': exactitud_prueba,
    'Media(Perdida_entrenamiento)': perdida_entrenamiento,
    'Media(Perdida_validacion)': perdida_val,
    'Media(Perdida_prueba)':perdida_prueba
})
)

# Crear el gráfico de barras
fig = go.Figure()

# Añadir las barras en el orden especificado
fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=exactitud_entrenamiento,
    name='Exactitud de Entrenamiento',
    marker_color='blue',
    text=exactitud_entrenamiento,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=exactitud_val,
    name='Exactitud de Validación',
    marker_color='green',
    text=exactitud_val,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=perdida_entrenamiento,
    name='Pérdida de Entrenamiento',
    marker_color='red',
    text=perdida_entrenamiento,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=perdida_val,
    name='Pérdida de Validación',
    marker_color='orange',
    text=perdida_val,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=exactitud_prueba,
    name='Exactitud de Prueba',
    marker_color='purple',
    text=exactitud_prueba,
    textposition='outside',
    hoverinfo='none'
))

fig.add_trace(go.Bar(
    x=nombres_modelos,
    y=perdida_prueba,
    name='Pérdida de Prueba',
    marker_color='darkred',
    text=perdida_prueba,
    textposition='outside',
    hoverinfo='none'
))

```

```

# Configurar el layout del gráfico
fig.update_layout(
    title='Comparativa de Modelos',
    barmode='group',
    xaxis_title='Modelos',
    yaxis_title='Valor',
    template='seaborn',
    width=1200,
    height=600
)

# Mostrar el gráfico
fig.show()
return df

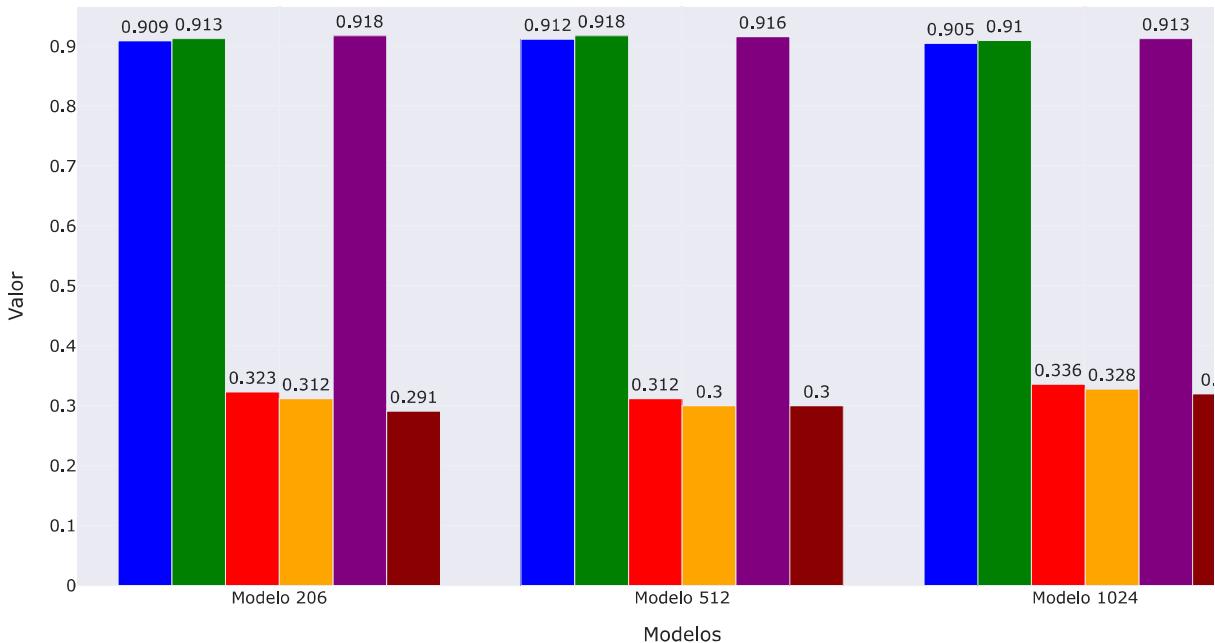
# Crear listas de histories, nombres de modelos y tests.
historias = [history, history_2, history_3]
modelos = ["Modelo 206", "Modelo 512", "Modelo 1024"]

tests = [model_2.evaluate(test_imagenes,test_labels), model.evaluate(test_imagenes,test_labels),model_3.evaluate(test_imagenes)]
# Llamar a la función para mostrar el gráfico
df=plot_comparacion_modelos(historias, modelos,tests)
df

313/313 ━━━━━━ 0s 770us/step - accuracy: 0.9053 - loss: 0.3308
313/313 ━━━━ 0s 964us/step - accuracy: 0.9042 - loss: 0.3361
313/313 ━━━━ 0s 1ms/step - accuracy: 0.9029 - loss: 0.3531

```

Comparativa de Modelos



	Modelo	Media(Exactitud_entrenamiento)	Media(Exactitud_validacion)	Media(Exactitud_prueba)	Media(Perdida_entrenamiento)	Media(Perdida_prueba)
0	Modelo 206	0.909	0.913	0.918	0.323	0.323
1	Modelo 512	0.912	0.918	0.916	0.312	0.312
2	Modelo 1024	0.905	0.910	0.913	0.336	0.336

```

In [19]: import seaborn as sns

def calcular_flops(modelo, numero_inputs):
    total_flops = 0
    for layer in modelo.layers:
        if isinstance(layer, tf.keras.layers.Dense):
            # Obtener unidades de entrada
            if hasattr(layer, 'input_shape') and layer.input_shape is not None:
                units_in = layer.input_shape[-1]
            elif layer.weights:

```

```

        units_in = layer.weights[0].shape[0]
    else:
        units_in = numero_inputs # Proporciona el número de entradas del modelo

    # Obtener unidades de salida utilizando 'layer.units'
    units_out = layer.units

    # Calcular FLOPs: multiplicaciones y sumas
    flops = 2 * units_in * units_out

    # Incluir las operaciones de bias si la capa las utiliza
    if layer.use_bias:
        flops += units_out

    total_flops += flops
else:
    print(f"La capa {layer.name} no es una capa Dense y no se incluye en el cálculo.")
return (total_flops/1024)/1024

def comparativa_costos(modelos, nombres_modelos, numero_inputs):
    # Inicializar listas para almacenar los datos
    parametros = []
    flops_modelos = []

    # Evaluar cada modelo
    for modelo in modelos:
        # Número de parámetros
        parametros.append(modelo.count_params()/1000)
        # Calcular FLOPs
        total_flops = calcular_flops(modelo,numero_inputs)
        flops_modelos.append(total_flops)
        # Medir tiempo de inferencia y precisión en prueba

        # Crear el DataFrame
        datos = {
            'Modelo': nombres_modelos,
            'Parámetros': parametros,
            'MFLOPS': flops_modelos,
        }
        df = pd.DataFrame(datos)

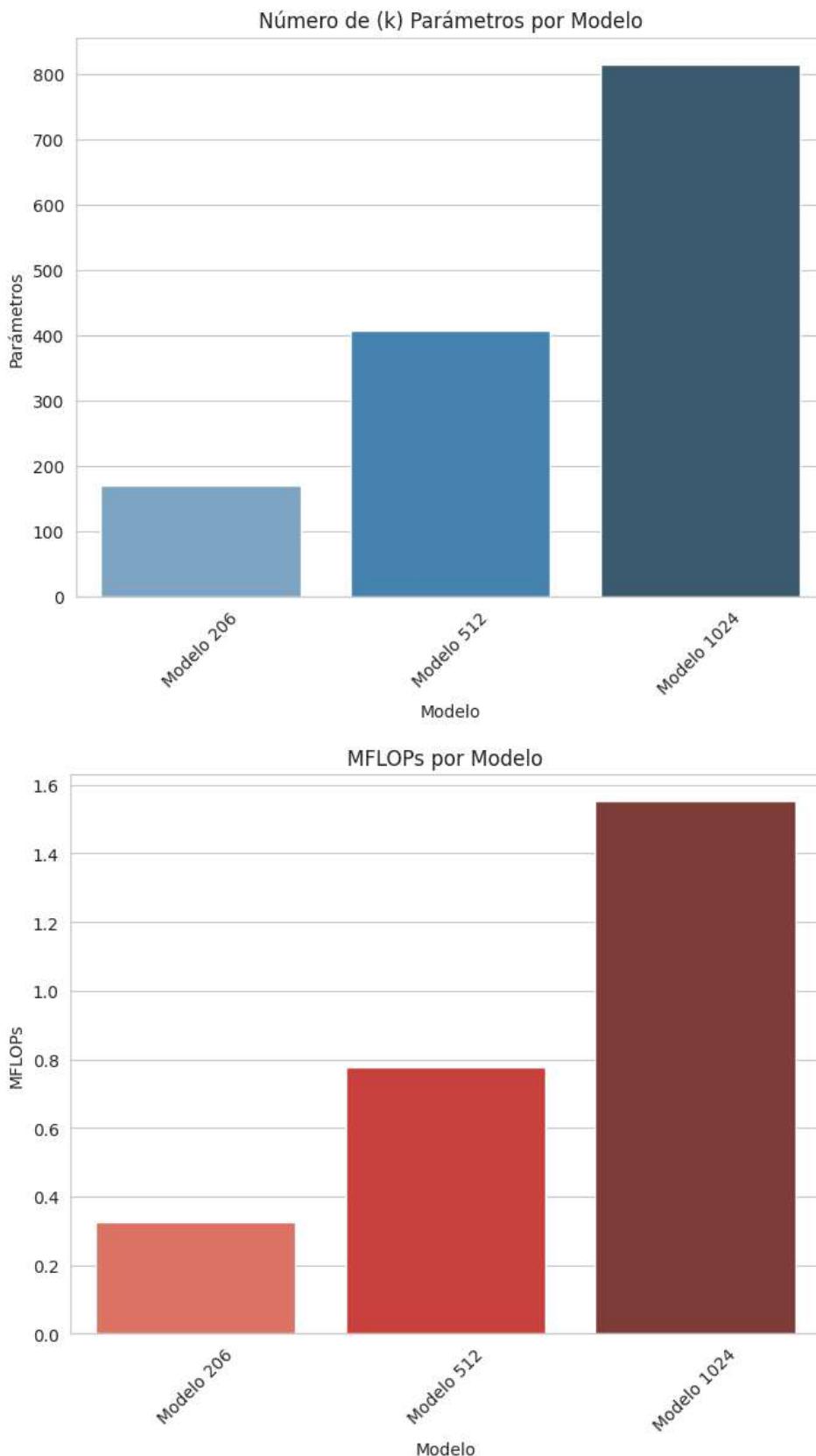
    # Establecer el estilo
    sns.set_style('whitegrid')

    # Diagrama de barras para 'Parámetros'
    plt.figure(figsize=(8, 6))
    sns.barplot(x='Modelo', y='Parámetros', hue='Modelo', data=df, palette='Blues_d', legend=False)
    plt.title('Número de (k) Parámetros por Modelo')
    plt.xticks(rotation=45)
    plt.show()

    # Diagrama de barras para 'FLOPs'
    plt.figure(figsize=(8, 6))
    sns.barplot(x='Modelo', y='MFLOPS', hue='Modelo', data=df, palette='Reds_d', legend=False)
    plt.title('MFLOPs por Modelo')
    plt.xticks(rotation=45)
    plt.show()

df=comparativa_costos([model_2,model,model_3],modelos,10)

```



Pregunta 5.1 (0.5 puntos): ¿Cuál es el impacto que tiene la red neuronal?

Al aumentar el numero de neuronas en la capa densa, estamos dando a la red mas capacidad para aprender patrones de problemas complejos. Sin embargo, este aumento de la capacidad de aprendizaje puede conllevar en ocasiones que los modelos con mas parametros sobreajusten los casos de entrenamiento teniendo finalmente un rendimiento menor de cara a su uso con casos no vistos. MNIST con digitos no es un problema muy complejo, de hecho los resultados con 206 neuronas en la capa oculta segun la tabla de comparativas alcanza un **exactitud** en test (prueba) mayor que los otros dos. En contraste a la perdida, es tambien el modelo 206 el que menor **perdida** tiene situandose. Esto situa al modelo con 206 como el mejor para este problema. Hemos analizado tambien las diferencias entre exactitud y perdida durante el entrenamiento y podemos identificar que el modelo 206 es el que presenta un aprendizaje mas robusto, con menores oscilaciones. Por contra los otros dos modelos si presentan cambios considerables en la funcion de perdida. Deberiamos modificar el

numero de capas para lograr un aprendizaje de casos complejos como los digitos 4 y 9 o 1 y 7 donde una sola capa no se esta revelando suficiente.

De forma adicional, y como podemos ver en la tabla de numero de parametros, los parametros han ido duplicandose con cada modelo, al igual de las operaciones requeridas para su solucion. En este caso en el modelo mas costoso llegamos casi a 1.6 MFLOPS. Este es un impacto importante a tener en cuenta cuando pongamos el modelo en produccion.

6: Número de neuronas de la capa de salida

Considerad la capa final, la de salida de la red neuronal de la pregunta 3.

Pregunta 6.1 (0.25 puntos): ¿Por qué son 10 las neuronas de la última capa?

Pregunta 6.2 (0.25 puntos): ¿Qué pasaría si tuvieras una cantidad diferente a 10?

Por ejemplo, intenta entrenar la red con 5, para ello utiliza la red neuronal de la pregunta 1 y cambia a 5 el número de neuronas en la última capa.

```
In [38]: model_4 = Sequential([
    Input(shape=(784,)),
    Dense(512),
    Dense(5,activation='softmax')
])

model_4.compile(optimizer='adam',
                 loss='categorical_crossentropy',
                 metrics=['accuracy'])
history_4 = model_4.fit(training_images, training_labels,
                        epochs=5,
                        batch_size=32,
                        validation_split=0.25)
```

Epoch 1/5

```
ValueError                                     Traceback (most recent call last)
Cell In[38], line 10
      1 model_4 = Sequential([
      2     Input(shape=(784,)),
      3     Dense(512),
      4     Dense(5,activation='softmax')
      5 ])
      6 model_4.compile(optimizer='adam',
      7                 loss='categorical_crossentropy',
      8                 metrics=['accuracy'])
--> 10 history_4 = model_4.fit(training_images, training_labels,
      11                 epochs=5,
      12                 batch_size=32,
      13                 validation_split=0.25)

File ~/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages/keras/src/utils/traceback_utils.py:122, in filter_traceback.<locals>.error_handler(*args, **kwargs)
    119     filtered_tb = _process_traceback_frames(e.__traceback__)
    120     # To get the full stack trace, call:
    121     # `keras.config.disable_traceback_filtering()`
--> 122     raise e.with_traceback(filtered_tb) from None
    123 finally:
    124     del filtered_tb

File ~/workspace/my-universe/ai-data-science/.venv/lib/python3.10/site-packages/keras/src/backend/tensorflow/nn.py:587, in categorical_crossentropy(target, output, from_logits, axis)
    585 for e1, e2 in zip(target.shape, output.shape):
    586     if e1 is not None and e2 is not None and e1 != e2:
--> 587         raise ValueError(
    588             "Arguments `target` and `output` must have the same shape. "
    589             "Received: "
    590             f"target.shape={target.shape}, output.shape={output.shape}"
    591         )
    593 output, from_logits = _get_logits(
    594     output, from_logits, "Softmax", "categorical_crossentropy"
    595 )
    596 if from_logits:
```

ValueError: Arguments `target` and `output` must have the same shape. Received: target.shape=(None, 10), output.shape=(None, 5)

.- Las neuronales de la capa de salida, representan la distibucion de la probabilidad de que una imagen pertenezca a una de las 10 clases incluidas en el set de datos. La funcion de softmax se encarga de asegurar que la suma de todas las probabilidades sume correctamente 1.

.- Las funciones de softmax y categorical_crossentropy no pueden ser calculadas. Los vectores de target tienen una dimension de 10 en lugar de 5. Quedan 5 distribuciones sin poder ser asignadas (comparadas con el valor esperado). .- La propia librería detecta el error en los parametros de configuracion.

7: Aumento de epoch y su efecto en la red neuronal

En este ejercicio vamos a ver el impacto de aumentar los epoch en el entrenamiento. Usando la red neuronal de la pregunta 3:

Pregunta 7.1 (0.25 puntos)

- Intentad 15 epoch para su entrenamiento, probablemente obtendras un modelo con una pérdida mucho mejor que el que tiene 5.

Pregunta 7.2 (0.25 puntos)

- Intenta ahora con 30 epoch para su entrenamiento.

Pregunta 7.3 (0.25 puntos)

- ¿Qué está pasando en la pregunta anterior? Explica tu respuesta y da el nombre de este efecto si lo conoces.

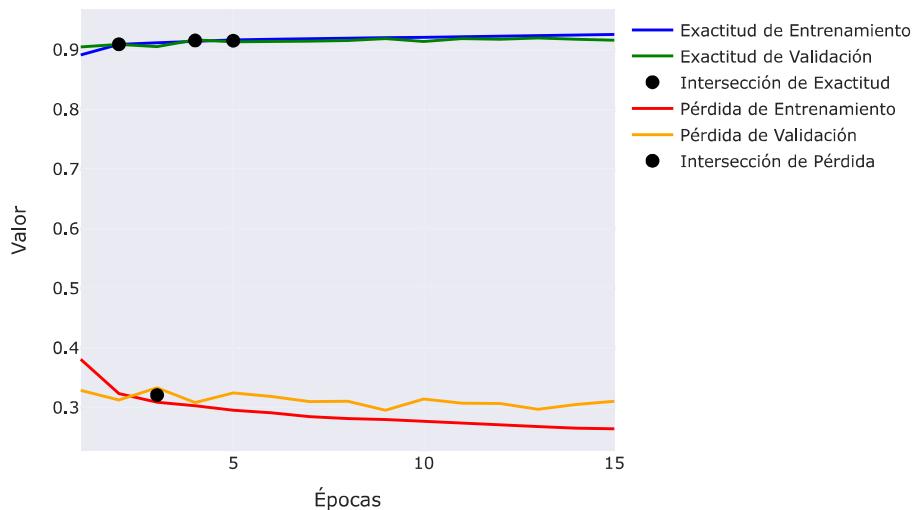
```
In [21]: model_15e = Sequential([
    Input(shape=(784,)),
    Dense(512),
    Dense(10,activation='softmax')
])

model_15e.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
history_15e = model_15e.fit(training_images, training_labels,
                            epochs=15,
                            batch_size=32,
                            validation_split=0.25)

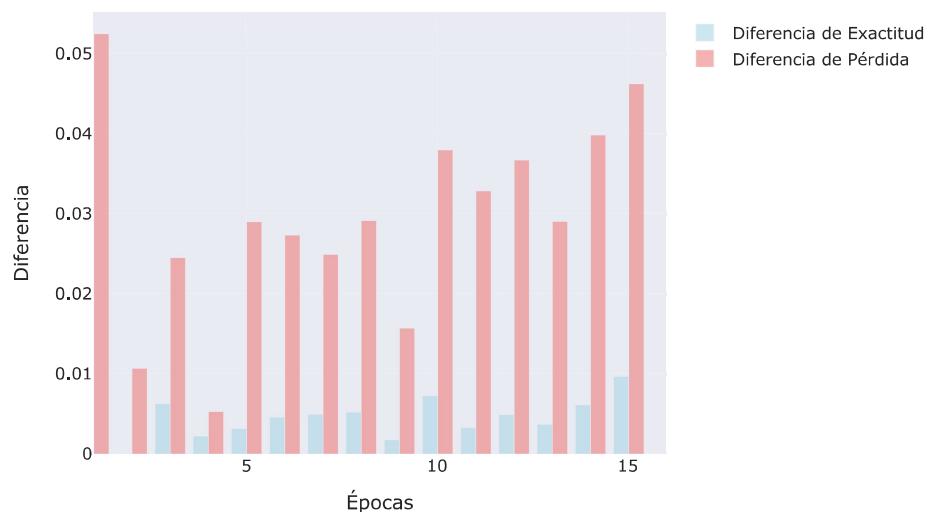
plot_history(history_15e,"Modelo 15e")
```

Epoch 1/15
1407/1407 4s 3ms/step - accuracy: 0.8617 - loss: 0.4623 - val_accuracy: 0.9049 - val_loss: 0.3289
Epoch 2/15
1407/1407 4s 3ms/step - accuracy: 0.9118 - loss: 0.3214 - val_accuracy: 0.9093 - val_loss: 0.3128
Epoch 3/15
1407/1407 4s 3ms/step - accuracy: 0.9104 - loss: 0.3123 - val_accuracy: 0.9053 - val_loss: 0.3333
Epoch 4/15
1407/1407 7s 5ms/step - accuracy: 0.9138 - loss: 0.2980 - val_accuracy: 0.9167 - val_loss: 0.3086
Epoch 5/15
1407/1407 4s 3ms/step - accuracy: 0.9148 - loss: 0.2954 - val_accuracy: 0.9137 - val_loss: 0.3244
Epoch 6/15
1407/1407 4s 3ms/step - accuracy: 0.9187 - loss: 0.2882 - val_accuracy: 0.9134 - val_loss: 0.3185
Epoch 7/15
1407/1407 4s 3ms/step - accuracy: 0.9208 - loss: 0.2744 - val_accuracy: 0.9143 - val_loss: 0.3099
Epoch 8/15
1407/1407 4s 3ms/step - accuracy: 0.9225 - loss: 0.2761 - val_accuracy: 0.9156 - val_loss: 0.3105
Epoch 9/15
1407/1407 4s 3ms/step - accuracy: 0.9218 - loss: 0.2722 - val_accuracy: 0.9189 - val_loss: 0.2955
Epoch 10/15
1407/1407 4s 3ms/step - accuracy: 0.9194 - loss: 0.2799 - val_accuracy: 0.9139 - val_loss: 0.3144
Epoch 11/15
1407/1407 4s 3ms/step - accuracy: 0.9228 - loss: 0.2684 - val_accuracy: 0.9187 - val_loss: 0.3072
Epoch 12/15
1407/1407 7s 5ms/step - accuracy: 0.9244 - loss: 0.2666 - val_accuracy: 0.9179 - val_loss: 0.3071
Epoch 13/15
1407/1407 4s 3ms/step - accuracy: 0.9250 - loss: 0.2628 - val_accuracy: 0.9197 - val_loss: 0.2972
Epoch 14/15
1407/1407 4s 3ms/step - accuracy: 0.9253 - loss: 0.2586 - val_accuracy: 0.9177 - val_loss: 0.3055
Epoch 15/15
1407/1407 4s 3ms/step - accuracy: 0.9257 - loss: 0.2605 - val_accuracy: 0.9163 - val_loss: 0.3105

Curvas de Exactitud y Pérdida (Modelo 15e)



Diferencias entre Entrenamiento y Validación (Modelo 15e)



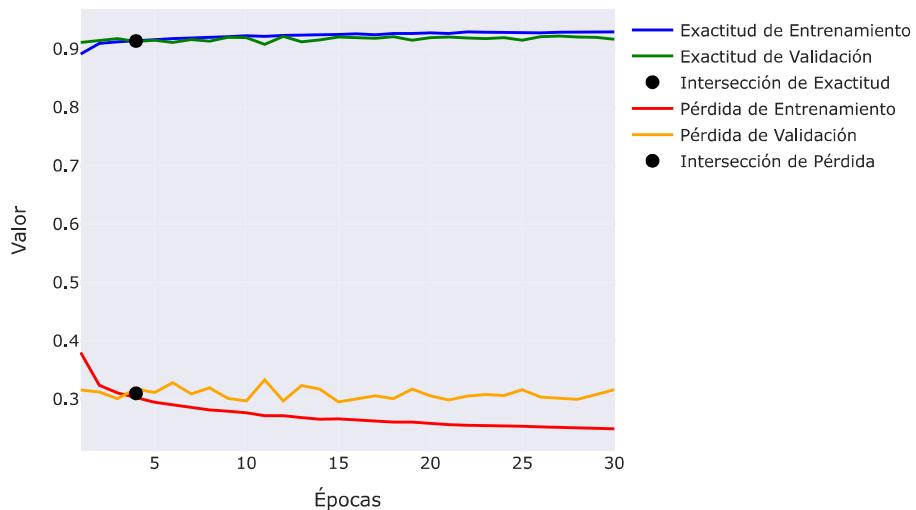
```
In [22]: model_30e = Sequential([
    Input(shape=(784,)),
    Dense(512),
    Dense(10,activation='softmax')
])

model_30e.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
history_30e = model_30e.fit(training_images, training_labels,
                            epochs=30,
                            batch_size=32,
                            validation_split=0.25)

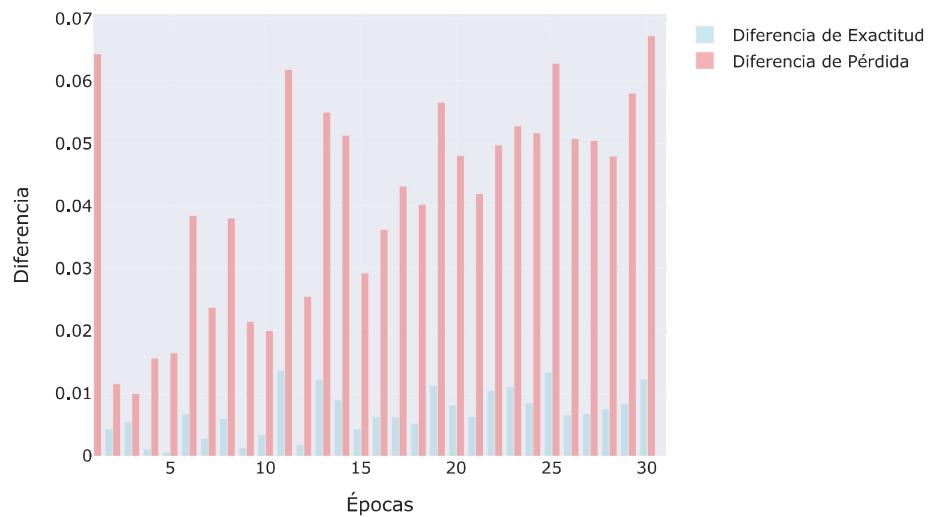
plot_history(history_30e,"Modelo 30e")
```

Epoch 1/30
1407/1407 4s 3ms/step - accuracy: 0.8655 - loss: 0.4539 - val_accuracy: 0.9110 - val_loss: 0.3155
Epoch 2/30
1407/1407 4s 3ms/step - accuracy: 0.9065 - loss: 0.3277 - val_accuracy: 0.9135 - val_loss: 0.3123
Epoch 3/30
1407/1407 4s 3ms/step - accuracy: 0.9122 - loss: 0.3077 - val_accuracy: 0.9174 - val_loss: 0.3009
Epoch 4/30
1407/1407 4s 3ms/step - accuracy: 0.9142 - loss: 0.2969 - val_accuracy: 0.9129 - val_loss: 0.3177
Epoch 5/30
1407/1407 7s 5ms/step - accuracy: 0.9156 - loss: 0.2951 - val_accuracy: 0.9149 - val_loss: 0.3110
Epoch 6/30
1407/1407 4s 3ms/step - accuracy: 0.9201 - loss: 0.2817 - val_accuracy: 0.9109 - val_loss: 0.3285
Epoch 7/30
1407/1407 4s 3ms/step - accuracy: 0.9213 - loss: 0.2782 - val_accuracy: 0.9158 - val_loss: 0.3090
Epoch 8/30
1407/1407 4s 3ms/step - accuracy: 0.9199 - loss: 0.2838 - val_accuracy: 0.9133 - val_loss: 0.3195
Epoch 9/30
1407/1407 4s 3ms/step - accuracy: 0.9199 - loss: 0.2774 - val_accuracy: 0.9196 - val_loss: 0.3011
Epoch 10/30
1407/1407 4s 3ms/step - accuracy: 0.9244 - loss: 0.2698 - val_accuracy: 0.9193 - val_loss: 0.2967
Epoch 11/30
1407/1407 4s 3ms/step - accuracy: 0.9216 - loss: 0.2653 - val_accuracy: 0.9077 - val_loss: 0.3332
Epoch 12/30
1407/1407 5s 3ms/step - accuracy: 0.9237 - loss: 0.2732 - val_accuracy: 0.9215 - val_loss: 0.2971
Epoch 13/30
1407/1407 7s 5ms/step - accuracy: 0.9268 - loss: 0.2627 - val_accuracy: 0.9124 - val_loss: 0.3235
Epoch 14/30
1407/1407 4s 3ms/step - accuracy: 0.9235 - loss: 0.2657 - val_accuracy: 0.9154 - val_loss: 0.3171
Epoch 15/30
1407/1407 4s 3ms/step - accuracy: 0.9267 - loss: 0.2608 - val_accuracy: 0.9206 - val_loss: 0.2954
Epoch 16/30
1407/1407 4s 3ms/step - accuracy: 0.9272 - loss: 0.2609 - val_accuracy: 0.9195 - val_loss: 0.3002
Epoch 17/30
1407/1407 4s 3ms/step - accuracy: 0.9260 - loss: 0.2527 - val_accuracy: 0.9180 - val_loss: 0.3058
Epoch 18/30
1407/1407 4s 3ms/step - accuracy: 0.9266 - loss: 0.2558 - val_accuracy: 0.9209 - val_loss: 0.3009
Epoch 19/30
1407/1407 4s 3ms/step - accuracy: 0.9283 - loss: 0.2503 - val_accuracy: 0.9150 - val_loss: 0.3170
Epoch 20/30
1407/1407 4s 3ms/step - accuracy: 0.9302 - loss: 0.2492 - val_accuracy: 0.9191 - val_loss: 0.3060
Epoch 21/30
1407/1407 7s 5ms/step - accuracy: 0.9297 - loss: 0.2460 - val_accuracy: 0.9201 - val_loss: 0.2984
Epoch 22/30
1407/1407 4s 3ms/step - accuracy: 0.9305 - loss: 0.2474 - val_accuracy: 0.9187 - val_loss: 0.3051
Epoch 23/30
1407/1407 4s 3ms/step - accuracy: 0.9258 - loss: 0.2594 - val_accuracy: 0.9177 - val_loss: 0.3080
Epoch 24/30
1407/1407 4s 3ms/step - accuracy: 0.9289 - loss: 0.2537 - val_accuracy: 0.9194 - val_loss: 0.3062
Epoch 25/30
1407/1407 4s 3ms/step - accuracy: 0.9286 - loss: 0.2486 - val_accuracy: 0.9151 - val_loss: 0.3162
Epoch 26/30
1407/1407 4s 3ms/step - accuracy: 0.9283 - loss: 0.2562 - val_accuracy: 0.9210 - val_loss: 0.3034
Epoch 27/30
1407/1407 4s 3ms/step - accuracy: 0.9307 - loss: 0.2456 - val_accuracy: 0.9219 - val_loss: 0.3018
Epoch 28/30
1407/1407 4s 3ms/step - accuracy: 0.9288 - loss: 0.2491 - val_accuracy: 0.9206 - val_loss: 0.2996
Epoch 29/30
1407/1407 4s 3ms/step - accuracy: 0.9274 - loss: 0.2466 - val_accuracy: 0.9196 - val_loss: 0.3082
Epoch 30/30
1407/1407 7s 5ms/step - accuracy: 0.9294 - loss: 0.2512 - val_accuracy: 0.9167 - val_loss: 0.3163

Curvas de Exactitud y Pérdida (Modelo 30e)



Diferencias entre Entrenamiento y Validación (Modelo 30e)

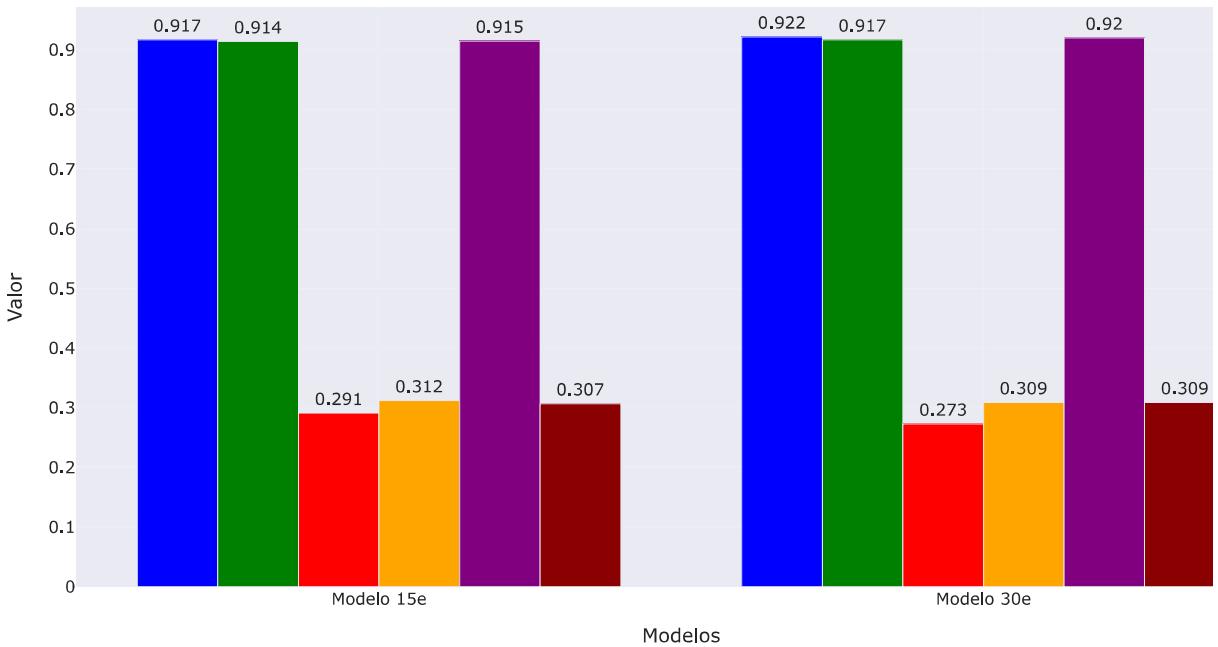


```
In [23]: # Crear Listas de histories, nombres de modelos y tests.
historias_3 = [history_15e, history_30e]
modelos_3 = [ "Modelo 15e", "Modelo 30e"]
tests_3 = [ model_15e.evaluate(test_imagenes,test_labels),model_30e.evaluate(test_imagenes,test_labels)]
```

Llamar a la función para mostrar el gráfico
plot_comparacion_modelos(historias_3, modelos_3,tests_3)

```
313/313 ━━━━━━━━ 0s 951us/step - accuracy: 0.9045 - loss: 0.3464  
313/313 ━━━━━━━━ 0s 927us/step - accuracy: 0.9089 - loss: 0.3447
```

Comparativa de Modelos



	Modelo	Media(Exactitud_entrenamiento)	Media(Exactitud_validacion)	Media(Exactitud_prueba)	Media(Perdida_entrenamiento)	Media(Perdida_validacion)
0	Modelo 15e	0.917	0.914	0.915	0.291	0.312
1	Modelo 30e	0.922	0.917	0.92	0.273	0.309

.- Efectivamente la perdida en prueba de ambos entrenamientos es menor que la perdida inicial del modelo de 5 epochs, sin embargo observamos claramente el sobreajuste de los modelos al ser entrenados con mas epochs. Esto se puede verificar al analizar las diferencias entre perdida en entrenamiento y perdida de validacion. Como se puede observar en el grafico de diferencias, estos valores presentan una divergencia creciente. Los modelos "memorizan" los casos, pero generalizan incorrectamente.

8: Early stop

En el ejercicio anterior, cuando entrenabas con epoch extras, tenías un problema en el que tu pérdida podía cambiar. Puede que te haya llevado un poco de tiempo esperar a que el entrenamiento lo hiciera, y puede que hayas pensado "¿no estaría bien si pudiera parar el entrenamiento cuando alcance un valor deseado?", es decir, una precisión del 85% podría ser suficiente para ti, y si alcanzas eso después de 3 epoch, ¿por qué sentarte a esperar a que termine muchas más épocas? Como cualquier otro programa existen formas de parar la ejecución

A partir del código de ejemplo, hacer una nueva función que tenga en cuenta la perdida (loss) y que pueda parar el código para evitar que ocurra el efecto secundario que vimos en el ejercicio 5.

```
In [24]: class myCallback(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        if(logs is not None) and (logs.get('loss') < 0.20):
            print("\n Alcanzada una perdida menor al 20%, se cancela el entrenamiento!!")
            self.model.stop_training = True
```

Pregunta 8.1. *(0.75 puntos)*: Consulta la documentación de Keras y aprende cómo podemos utilizar Early stop en nuestros modelos.

```
In [39]: callback=myCallback()

model_early_stop = Sequential([
    Input(shape=(784,)),
    Dense(512),
    Dense(10,activation='softmax')
])

model_early_stop.compile(optimizer='adam',
                        loss='categorical_crossentropy',
                        metrics=['accuracy'])
```

```

history_early_stop = model_early_stop.fit(training_images, training_labels,
                                         epochs=30,
                                         batch_size=32,
                                         validation_split=0.25,
                                         callbacks=[callback])

Epoch 1/30
1407/1407 4s 3ms/step - accuracy: 0.8616 - loss: 0.4669 - val_accuracy: 0.9127 - val_loss: 0.3125
Epoch 2/30
1407/1407 4s 3ms/step - accuracy: 0.9088 - loss: 0.3225 - val_accuracy: 0.9197 - val_loss: 0.2986
Epoch 3/30
1407/1407 7s 5ms/step - accuracy: 0.9122 - loss: 0.3049 - val_accuracy: 0.9152 - val_loss: 0.3092
Epoch 4/30
1407/1407 4s 3ms/step - accuracy: 0.9155 - loss: 0.3012 - val_accuracy: 0.9157 - val_loss: 0.3117
Epoch 5/30
1407/1407 4s 3ms/step - accuracy: 0.9183 - loss: 0.2924 - val_accuracy: 0.9162 - val_loss: 0.3127
Epoch 6/30
1407/1407 4s 3ms/step - accuracy: 0.9175 - loss: 0.2890 - val_accuracy: 0.9129 - val_loss: 0.3165
Epoch 7/30
1407/1407 4s 3ms/step - accuracy: 0.9162 - loss: 0.2878 - val_accuracy: 0.9152 - val_loss: 0.3051
Epoch 8/30
1407/1407 4s 3ms/step - accuracy: 0.9210 - loss: 0.2775 - val_accuracy: 0.9179 - val_loss: 0.3137
Epoch 9/30
1407/1407 4s 3ms/step - accuracy: 0.9220 - loss: 0.2794 - val_accuracy: 0.9199 - val_loss: 0.3009
Epoch 10/30
1407/1407 4s 3ms/step - accuracy: 0.9246 - loss: 0.2601 - val_accuracy: 0.9165 - val_loss: 0.3047
Epoch 11/30
1407/1407 4s 3ms/step - accuracy: 0.9240 - loss: 0.2705 - val_accuracy: 0.9123 - val_loss: 0.3188
Epoch 12/30
1407/1407 6s 4ms/step - accuracy: 0.9259 - loss: 0.2607 - val_accuracy: 0.9143 - val_loss: 0.3048
Epoch 13/30
1407/1407 4s 3ms/step - accuracy: 0.9226 - loss: 0.2653 - val_accuracy: 0.9185 - val_loss: 0.3135
Epoch 14/30
1407/1407 4s 3ms/step - accuracy: 0.9258 - loss: 0.2601 - val_accuracy: 0.9177 - val_loss: 0.3043
Epoch 15/30
1407/1407 4s 3ms/step - accuracy: 0.9241 - loss: 0.2645 - val_accuracy: 0.9217 - val_loss: 0.3016
Epoch 16/30
1407/1407 4s 3ms/step - accuracy: 0.9261 - loss: 0.2651 - val_accuracy: 0.9174 - val_loss: 0.3041
Epoch 17/30
1407/1407 4s 3ms/step - accuracy: 0.9250 - loss: 0.2608 - val_accuracy: 0.9161 - val_loss: 0.3065
Epoch 18/30
1407/1407 4s 3ms/step - accuracy: 0.9292 - loss: 0.2537 - val_accuracy: 0.9179 - val_loss: 0.3164
Epoch 19/30
1407/1407 4s 3ms/step - accuracy: 0.9276 - loss: 0.2528 - val_accuracy: 0.9146 - val_loss: 0.3137
Epoch 20/30
1407/1407 4s 3ms/step - accuracy: 0.9291 - loss: 0.2473 - val_accuracy: 0.9200 - val_loss: 0.3061
Epoch 21/30
1407/1407 8s 5ms/step - accuracy: 0.9289 - loss: 0.2474 - val_accuracy: 0.9227 - val_loss: 0.3014
Epoch 22/30
1407/1407 4s 3ms/step - accuracy: 0.9288 - loss: 0.2470 - val_accuracy: 0.9166 - val_loss: 0.3108
Epoch 23/30
1407/1407 4s 3ms/step - accuracy: 0.9284 - loss: 0.2469 - val_accuracy: 0.9188 - val_loss: 0.3062
Epoch 24/30
1407/1407 4s 3ms/step - accuracy: 0.9293 - loss: 0.2462 - val_accuracy: 0.9180 - val_loss: 0.3054
Epoch 25/30
1407/1407 4s 3ms/step - accuracy: 0.9283 - loss: 0.2566 - val_accuracy: 0.9189 - val_loss: 0.3095
Epoch 26/30
1407/1407 4s 3ms/step - accuracy: 0.9301 - loss: 0.2415 - val_accuracy: 0.9159 - val_loss: 0.3183
Epoch 27/30
1407/1407 4s 3ms/step - accuracy: 0.9277 - loss: 0.2532 - val_accuracy: 0.9151 - val_loss: 0.3141
Epoch 28/30
1407/1407 4s 3ms/step - accuracy: 0.9309 - loss: 0.2464 - val_accuracy: 0.9178 - val_loss: 0.3116
Epoch 29/30
1407/1407 6s 4ms/step - accuracy: 0.9299 - loss: 0.2440 - val_accuracy: 0.9168 - val_loss: 0.3143
Epoch 30/30
1407/1407 4s 3ms/step - accuracy: 0.9300 - loss: 0.2457 - val_accuracy: 0.9187 - val_loss: 0.3084

```

9. Unidades de activación

En este ejercicio, vamos a evaluar la importancia de utilizar las unidades de activación adecuadas. Como hemos visto en clase, funciones de activación como sigmoid han dejado de utilizarse en favor de otras unidades como ReLU.

Pregunta 9.1 *(0.75 puntos)*: Utilizando la red realizada en el ejercicio 3, escribir un breve análisis comparando la utilización de unidades sigmoid y ReLU (por ejemplo, se pueden comentar aspectos como velocidad de convergencia, métricas obtenidas...). Explicar por qué pueden darse estas diferencias. Opcionalmente, comparar con otras activaciones disponibles en Keras.

Pista: Usando redes más grandes se hace más sencillo apreciar las diferencias. Es mejor utilizar al menos 3 o 4 capas densas.

```

In [26]: model_5_ReLU = Sequential([
    Input(shape=(784,)),
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])

```

```

model_5_ReLU.compile(optimizer='adam',
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

history_5_ReLU = model_5_ReLU.fit(training_images, training_labels,
                                   epochs=5,
                                   batch_size=32,
                                   validation_split=0.25,
                                   )
plot_history(history_5_ReLU,"Modelo ReLU")

model_5_Sigmoidal = Sequential([
    Input(shape=(784,)),
    Dense(512,activation='sigmoid'),
    Dense(10,activation='softmax')
])

model_5_Sigmoidal.compile(optimizer='adam',
                           loss='categorical_crossentropy',
                           metrics=['accuracy'])

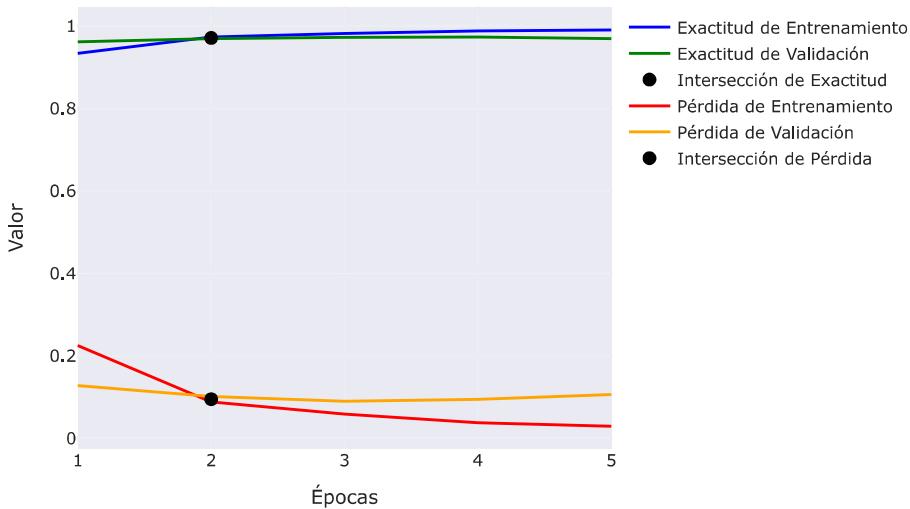
history_5_Sigmoidal = model_5_Sigmoidal.fit(training_images, training_labels,
                                              epochs=5,
                                              batch_size=32,
                                              validation_split=0.25
                                              )

plot_history(history_5_Sigmoidal,"Modelo Sigmoidal")

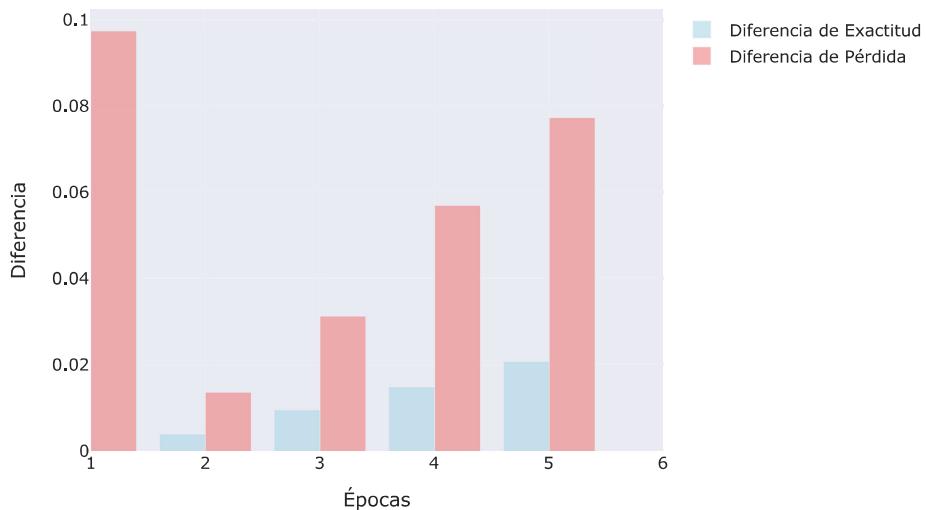
```

Epoch 1/5
1407/1407 7s 5ms/step - accuracy: 0.8896 - loss: 0.3754 - val_accuracy: 0.9617 - val_loss: 0.1282
Epoch 2/5
1407/1407 4s 3ms/step - accuracy: 0.9721 - loss: 0.0914 - val_accuracy: 0.9692 - val_loss: 0.1019
Epoch 3/5
1407/1407 4s 3ms/step - accuracy: 0.9838 - loss: 0.0524 - val_accuracy: 0.9725 - val_loss: 0.0901
Epoch 4/5
1407/1407 4s 3ms/step - accuracy: 0.9892 - loss: 0.0358 - val_accuracy: 0.9732 - val_loss: 0.0948
Epoch 5/5
1407/1407 4s 3ms/step - accuracy: 0.9915 - loss: 0.0269 - val_accuracy: 0.9695 - val_loss: 0.1067

Curvas de Exactitud y Pérdida (Modelo ReLU)



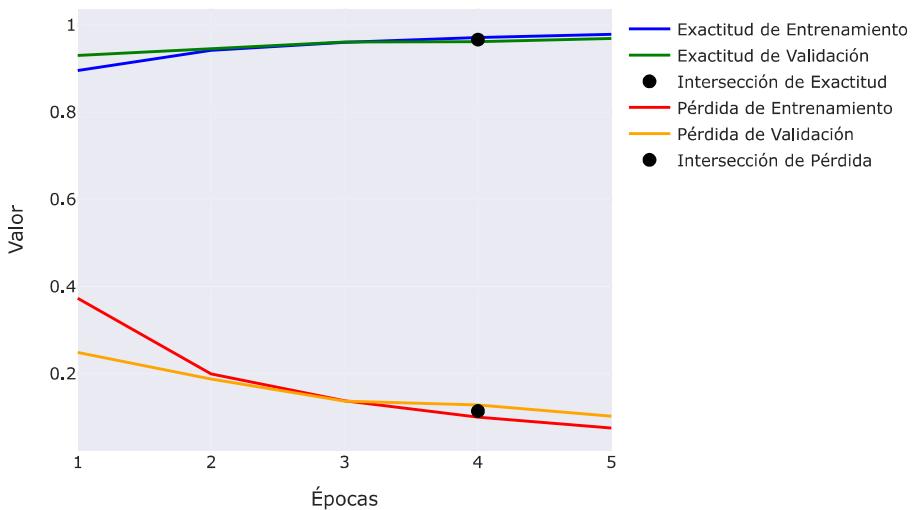
Diferencias entre Entrenamiento y Validación (Modelo ReLU)



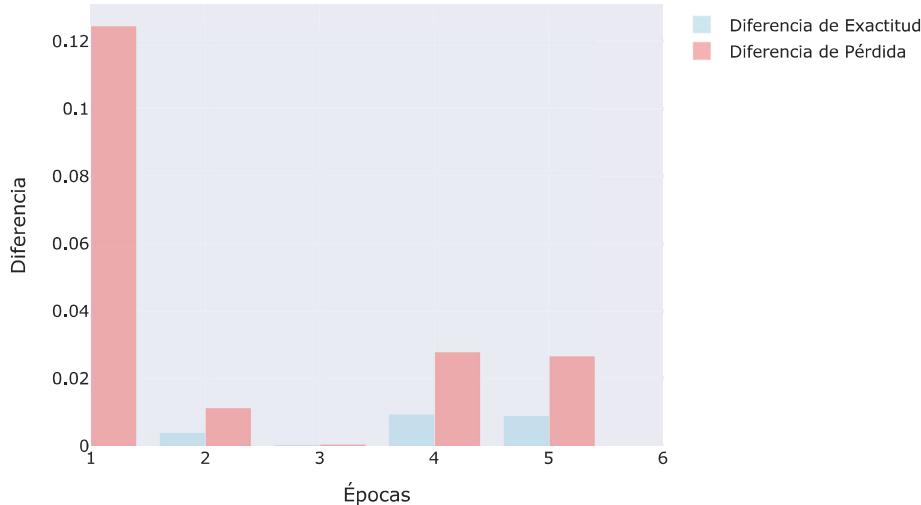
```

Epoch 1/5
1407/1407 4s 3ms/step - accuracy: 0.8406 - loss: 0.5756 - val_accuracy: 0.9299 - val_loss: 0.2489
Epoch 2/5
1407/1407 4s 3ms/step - accuracy: 0.9394 - loss: 0.2117 - val_accuracy: 0.9456 - val_loss: 0.1880
Epoch 3/5
1407/1407 4s 3ms/step - accuracy: 0.9580 - loss: 0.1465 - val_accuracy: 0.9605 - val_loss: 0.1369
Epoch 4/5
1407/1407 7s 5ms/step - accuracy: 0.9704 - loss: 0.1036 - val_accuracy: 0.9613 - val_loss: 0.1284
Epoch 5/5
1407/1407 4s 3ms/step - accuracy: 0.9785 - loss: 0.0731 - val_accuracy: 0.9689 - val_loss: 0.1023
  
```

Curvas de Exactitud y Pérdida (Modelo Sigmoidal)



Diferencias entre Entrenamiento y Validación (Modelo Sigmoidal)



```
In [27]: def normalizar_perdida(historia, metrica='loss'):
    valores = np.array(historia.history[metrica])
    perdida_inicial = valores[0]
    valores_normalizados = valores / perdida_inicial # Escalar con respecto a la pérdida inicial
    return valores_normalizados

def calcular_velocidad_convergencia_normalizada(historia, metrica='loss'):
    valores_normalizados = normalizar_perdida(historia, metrica)
    velocidad_convergencia = np.abs(np.diff(valores_normalizados))
    velocidad_promedio = np.mean(velocidad_convergencia)
    return velocidad_convergencia, velocidad_promedio

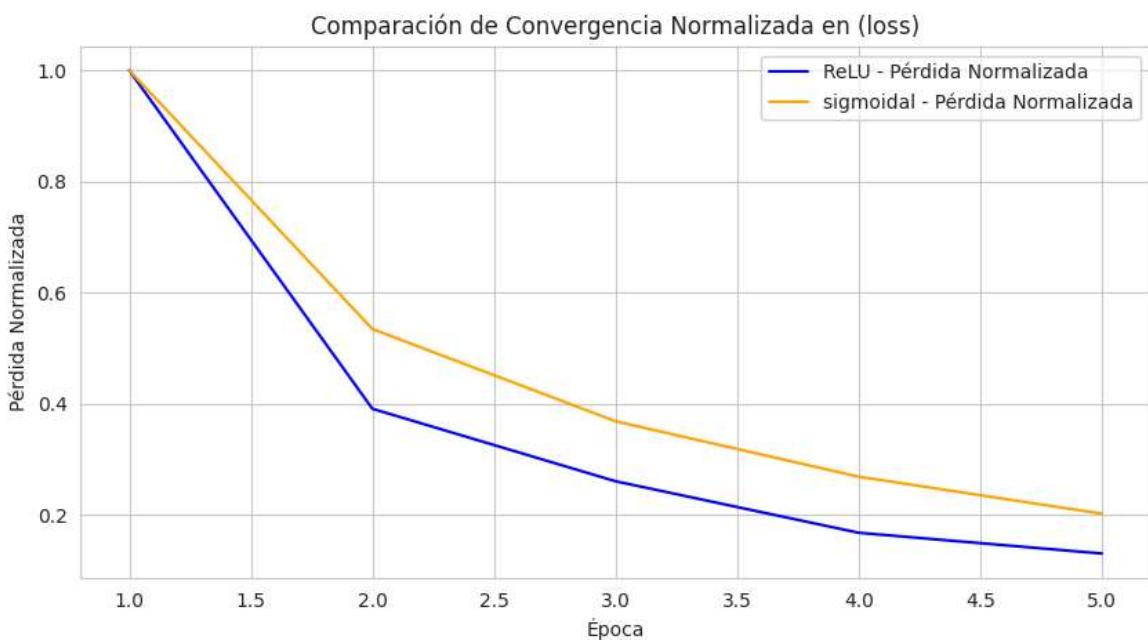
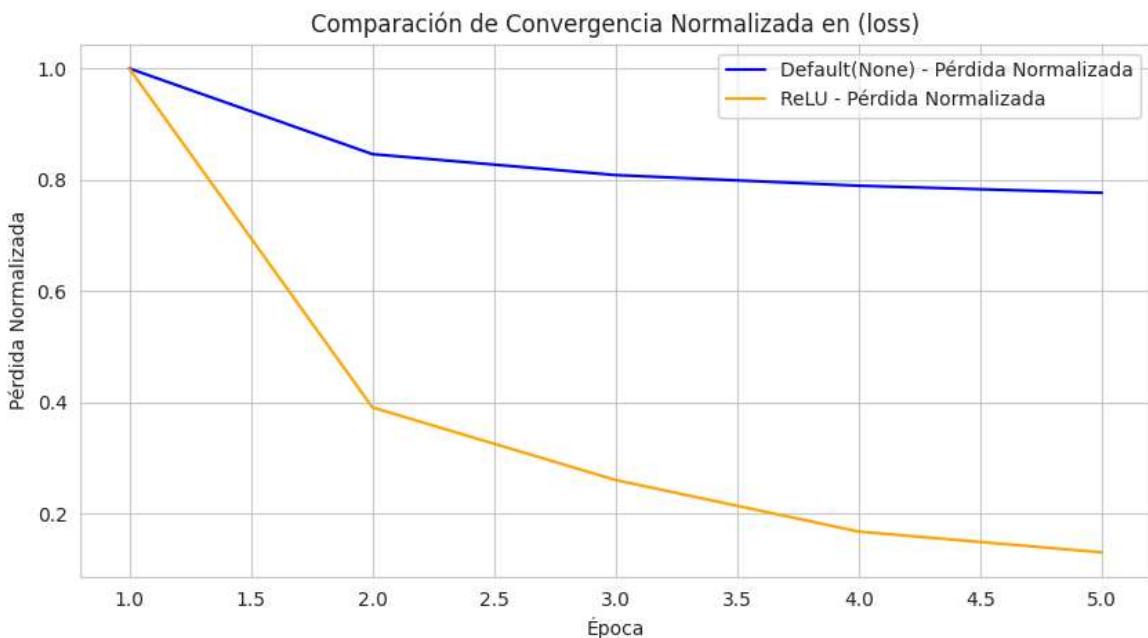
def comparar_convergencia_normalizada(nombre_modelo_1, historia1, nombre_modelo_2, historia2, metrica='loss'):
    # Obtener las pérdidas normalizadas
    valores_normalizados1 = normalizar_perdida(historia1, metrica)
    valores_normalizados2 = normalizar_perdida(historia2, metrica)

    # Calcular la velocidad de convergencia normalizada para ambos modelos
    velocidad_convergencia1, _ = calcular_velocidad_convergencia_normalizada(historia1, metrica)
    velocidad_convergencia2, _ = calcular_velocidad_convergencia_normalizada(historia2, metrica)

    # Crear un rango de épocas para el eje x
    épocas = np.arange(1, len(valores_normalizados1) + 1)

    # Graficar las pérdidas normalizadas y las velocidades de convergencia en un solo gráfico
    plt.figure(figsize=(10, 5))
    plt.plot(epocas, valores_normalizados1, label=f"{nombre_modelo_1} - Pérdida Normalizada", color='blue')
    plt.plot(epocas, valores_normalizados2, label=f"{nombre_modelo_2} - Pérdida Normalizada", color='orange')
    plt.title(f"Comparación de Convergencia Normalizada en ({metrica})")
    plt.xlabel('Época')
    plt.ylabel('Pérdida Normalizada')
    plt.legend()
    plt.grid(True)
    plt.show()

comparar_convergencia_normalizada("Default(None)",history,"ReLU",history_5_ReLU)
comparar_convergencia_normalizada("ReLU",history_5_ReLU,"sigmoidal",history_5_Sigmoidal)
```



Hemos comparado la velocidad de convergencia entre las funciones de activacion, ReLU, Sigmoidal y sin funcion especifica (es el default de Keras donde simplemente tenemos linealidad). Hemos observado que ReLU es mas rapida que Sigmoidal. La Sigmoidal trabaja en el rango 0-1, lo que puede hacer que las derivadas puedan volverse muy pequenas para valores extremos. Esto influye en la propagacion del gradiente ralentizando la actualizacion de los pesos. Tambien hemos comparado ReLU con linealidad y comprobamos que ReLU es mucho mas rapida. Para realizar todos estos calculos, hemos tenido que normizar los datos ya que los puntos de inicio de la perdida eran muy diferentes.

10. Inicialización de parámetros

En este ejercicio, vamos a evaluar la importancia de una correcta inicialización de parámetros en una red neuronal.

Pregunta 10.1 *(0.75 puntos): Partiendo de una red similar a la del ejercicio anterior (usando ya ReLUs), comentar las diferencias que se aprecian en el entrenamiento al utilizar distintas estrategias de inicialización de parámetros. Para ello, inicializar todas las capas con las siguientes estrategias, disponibles en Keras, y analizar sus diferencias:

- Inicialización con ceros.
- Inicialización con una variable aleatoria normal.
- Inicialización con los valores por defecto de Keras para una capa Dense (estrategia *glorot uniform*)

In [28]:

```
from tensorflow.keras import initializers

# Inicialización con ceros
model_5_ReLU_ceros = Sequential([
    Input(shape=(784,)),
    Dense(512, activation='relu', kernel_initializer=initializers.Zeros()),
```

```

        Dense(10, activation='softmax', kernel_initializer=initializers.Zeros())
    ])

model_5_ReLU_ceros.compile(optimizer='adam',
                           loss='categorical_crossentropy',
                           metrics=['accuracy'])

history_5_ReLU_ceros = model_5_ReLU_ceros.fit(training_images, training_labels,
                                               epochs=5,
                                               batch_size=32,
                                               validation_split=0.25)

plot_history(history_5_ReLU_ceros,"Modelo-5-ReLU-ceros")

# Inicialización con una variable aleatoria normal
model_5_ReLU_aleatoria = Sequential([
    Input(shape=(784,)),
    Dense(512, activation='relu', kernel_initializer=initializers.RandomNormal(mean=0., stddev=0.05)),
    Dense(10, activation='softmax', kernel_initializer=initializers.RandomNormal(mean=0., stddev=0.05))
])

model_5_ReLU_aleatoria.compile(optimizer='adam',
                               loss='categorical_crossentropy',
                               metrics=['accuracy'])

history_5_ReLU_aleatoria = model_5_ReLU_aleatoria.fit(training_images, training_labels,
                                                       epochs=5,
                                                       batch_size=32,
                                                       validation_split=0.25)
plot_history(history_5_ReLU_aleatoria,"Modelo-5-ReLU-aleatoria")

# Inicialización con una variable aleatoria normal
model_5_ReLU_glorot = Sequential([
    Input(shape=(784,)),
    Dense(512, activation='relu', kernel_initializer=initializers.glorot_uniform()),
    Dense(10, activation='softmax', kernel_initializer=initializers.glorot_uniform())
])

model_5_ReLU_glorot.compile(optimizer='adam',
                            loss='categorical_crossentropy',
                            metrics=['accuracy'])

history_5_ReLU_glorot = model_5_ReLU_glorot.fit(training_images, training_labels,
                                                 epochs=5,
                                                 batch_size=32,
                                                 validation_split=0.25)
plot_history(history_5_ReLU_glorot,"Modelo-5-ReLU-glorot")

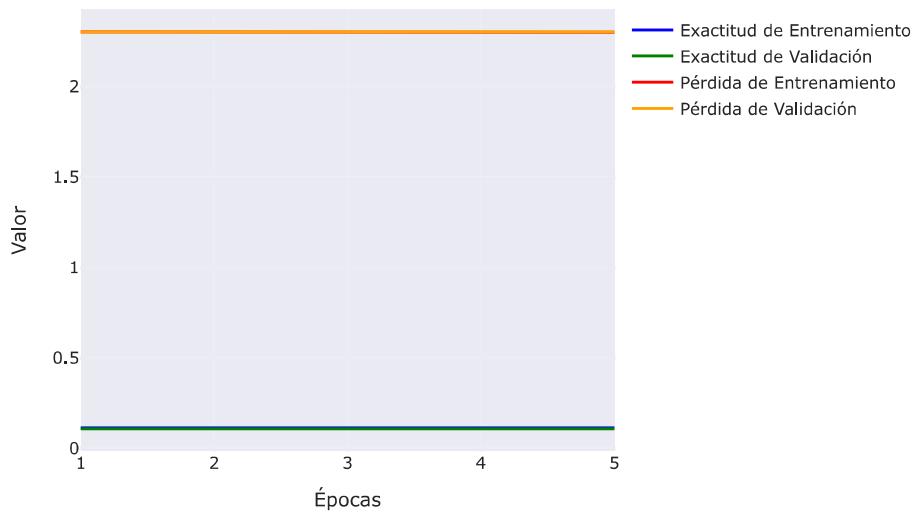
```

```

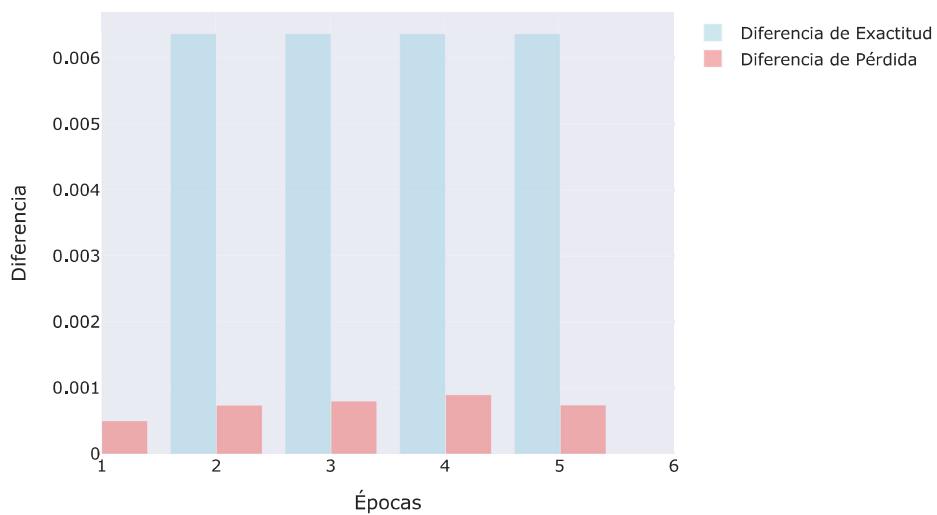
Epoch 1/5
1407/1407 4s 3ms/step - accuracy: 0.1125 - loss: 2.3019 - val_accuracy: 0.1076 - val_loss: 2.3019
Epoch 2/5
1407/1407 4s 3ms/step - accuracy: 0.1156 - loss: 2.3009 - val_accuracy: 0.1076 - val_loss: 2.3019
Epoch 3/5
1407/1407 4s 3ms/step - accuracy: 0.1151 - loss: 2.3008 - val_accuracy: 0.1076 - val_loss: 2.3019
Epoch 4/5
1407/1407 4s 3ms/step - accuracy: 0.1134 - loss: 2.3013 - val_accuracy: 0.1076 - val_loss: 2.3020
Epoch 5/5
1407/1407 4s 3ms/step - accuracy: 0.1149 - loss: 2.3009 - val_accuracy: 0.1076 - val_loss: 2.3019

```

Curvas de Exactitud y Pérdida (Modelo-5-ReLU-ceros)

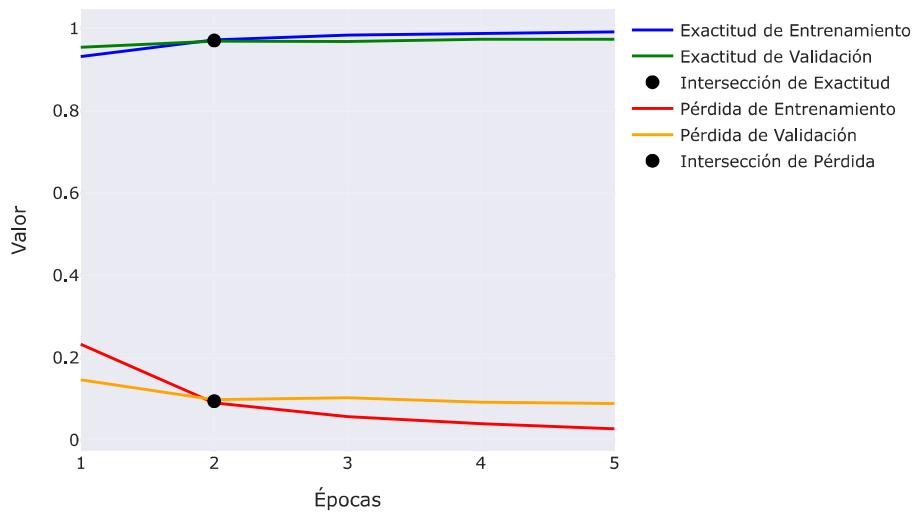


Diferencias entre Entrenamiento y Validación (Modelo-5-ReLU-ceros)

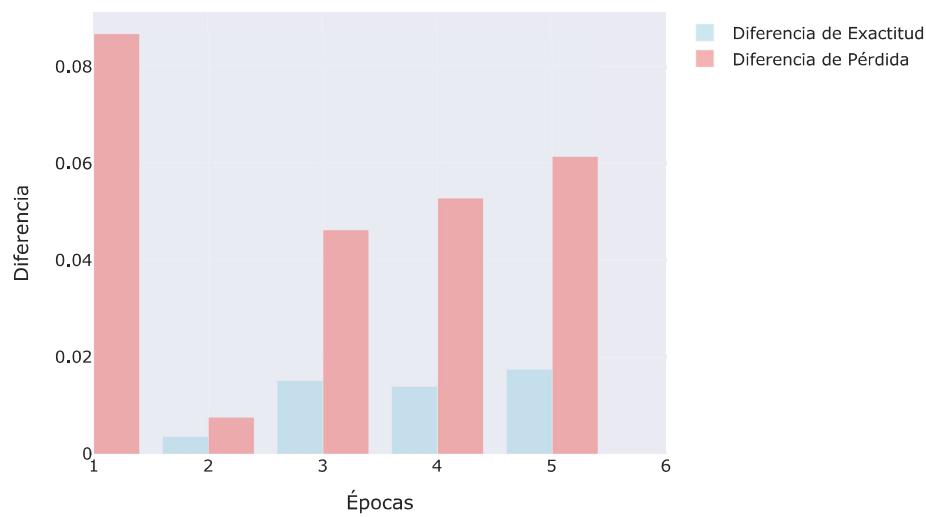


```
Epoch 1/5
1407/1407 4s 3ms/step - accuracy: 0.8846 - loss: 0.3967 - val_accuracy: 0.9544 - val_loss: 0.1459
Epoch 2/5
1407/1407 7s 5ms/step - accuracy: 0.9720 - loss: 0.0913 - val_accuracy: 0.9688 - val_loss: 0.0982
Epoch 3/5
1407/1407 4s 3ms/step - accuracy: 0.9850 - loss: 0.0520 - val_accuracy: 0.9684 - val_loss: 0.1026
Epoch 4/5
1407/1407 4s 3ms/step - accuracy: 0.9886 - loss: 0.0383 - val_accuracy: 0.9735 - val_loss: 0.0922
Epoch 5/5
1407/1407 4s 3ms/step - accuracy: 0.9927 - loss: 0.0242 - val_accuracy: 0.9740 - val_loss: 0.0884
```

Curvas de Exactitud y Pérdida (Modelo-5-ReLU-aleatoria)



Diferencias entre Entrenamiento y Validación (Modelo-5-ReLU-aleatoria)

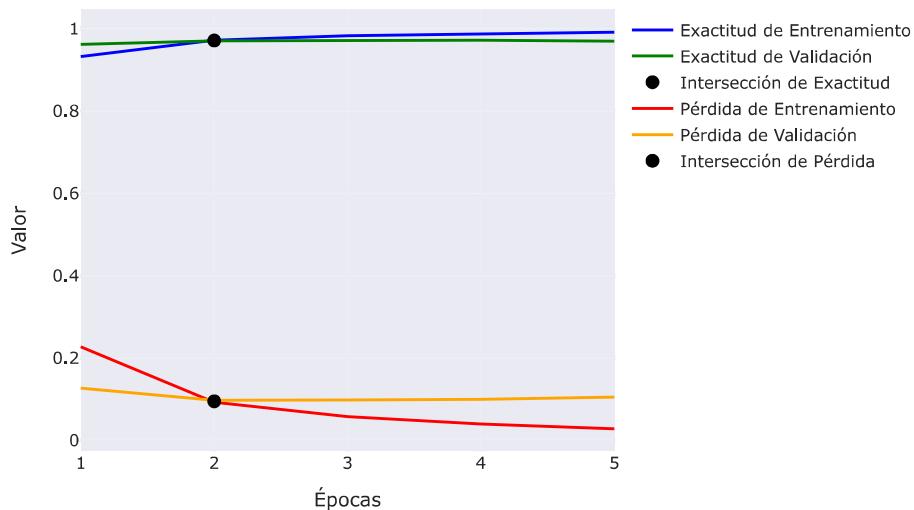


```

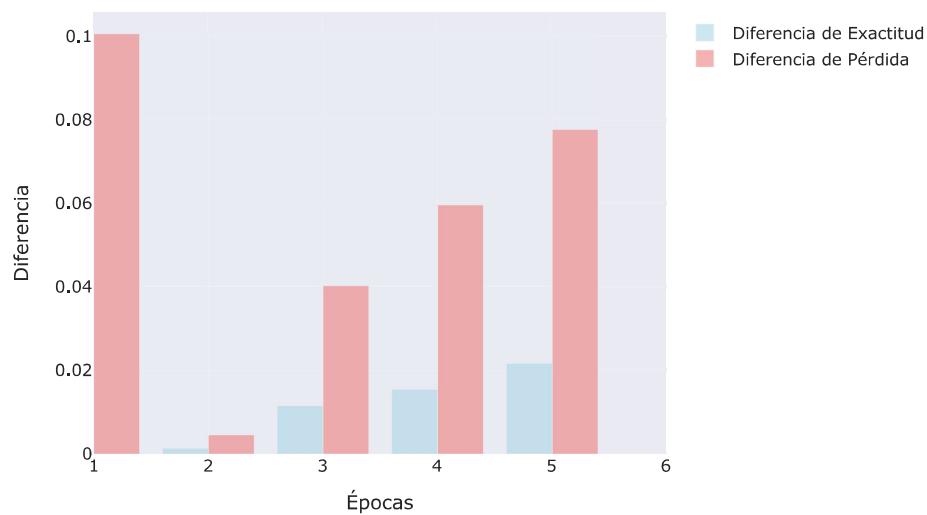
Epoch 1/5
1407/1407 4s 3ms/step - accuracy: 0.8881 - loss: 0.3825 - val_accuracy: 0.9616 - val_loss: 0.1271
Epoch 2/5
1407/1407 4s 3ms/step - accuracy: 0.9706 - loss: 0.0973 - val_accuracy: 0.9706 - val_loss: 0.0972
Epoch 3/5
1407/1407 4s 3ms/step - accuracy: 0.9843 - loss: 0.0542 - val_accuracy: 0.9712 - val_loss: 0.0976
Epoch 4/5
1407/1407 4s 3ms/step - accuracy: 0.9879 - loss: 0.0379 - val_accuracy: 0.9719 - val_loss: 0.0993
Epoch 5/5
1407/1407 7s 5ms/step - accuracy: 0.9912 - loss: 0.0293 - val_accuracy: 0.9697 - val_loss: 0.1054

```

Curvas de Exactitud y Pérdida (Modelo-5-ReLU-glorot)



Diferencias entre Entrenamiento y Validación (Modelo-5-ReLU-glorot)

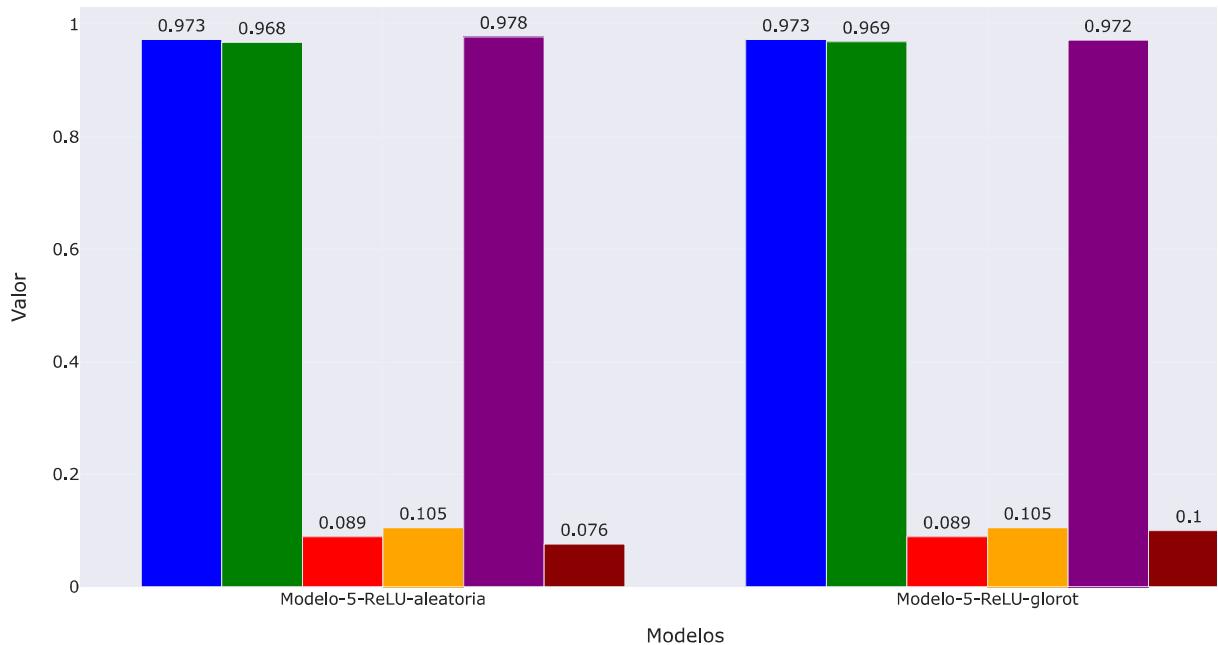


```
In [29]: # Crear Listas de histories, nombres de modelos y tests.
historias_4 = [history_5_ReLU_aleatoria, history_5_ReLU_glorot]
modelos_4 = [ "Modelo-5-ReLU-aleatoria", "Modelo-5-ReLU-glorot"]
tests_4 = [ model_5_ReLU_aleatoria.evaluate(test_imagenes,test_labels),model_5_ReLU_glorot.evaluate(test_imagenes,test_labels)

# Llamar a la función para mostrar el gráfico
plot_comparacion_modelos(historias_4, modelos_4,tests_4)

313/313 ━━━━━━ 0s 973us/step - accuracy: 0.9742 - loss: 0.0868
313/313 ━━━━━━ 0s 927us/step - accuracy: 0.9683 - loss: 0.1091
```

Comparativa de Modelos



	Modelo	Media(Exactitud_entrenamiento)	Media(Exactitud_validacion)	Media(Exactitud_prueba)	Media(Perdida_entrenamiento)	Media
0	Modelo-5-ReLU-aleatoria	0.973	0.968	0.978	0.089	
1	Modelo-5-ReLU-glorot	0.973	0.969	0.972	0.089	

11. Optimizadores

Problema 11.1 *(0.75 puntos)*: Partiendo de una red similar a la del ejercicio anterior (utilizando la mejor estrategia de inicialización observada), comparar y analizar las diferencias que se observan al entrenar con varios de los optimizadores vistos en clase, incluyendo SGD como optimizador básico (se puede explorar el espacio de hiperparámetros de cada optimizador, aunque para optimizadores más avanzados del estilo de RMSprop es buena idea dejar los valores por defecto provistos por Keras).

```
In [30]: model_5_ReLU_sgd = Sequential([
    Input(shape=(784,)),
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])

model_5_ReLU_sgd.compile(optimizer='sgd',
                        loss='categorical_crossentropy',
                        metrics=['accuracy'])

history_5_ReLU_sgd = model_5_ReLU_sgd.fit(training_images, training_labels,
                                         epochs=5,
                                         batch_size=32,
                                         validation_split=0.25)

plot_history(history_5_ReLU_sgd, "Modelo-5-ReLU-sgd")

model_5_ReLU_RMSprop = Sequential([
    Input(shape=(784,)),
    Dense(512, activation='relu'),
    Dense(10, activation='softmax')
])

model_5_ReLU_RMSprop.compile(optimizer='RMSprop',
                            loss='categorical_crossentropy',
                            metrics=['accuracy'])
```

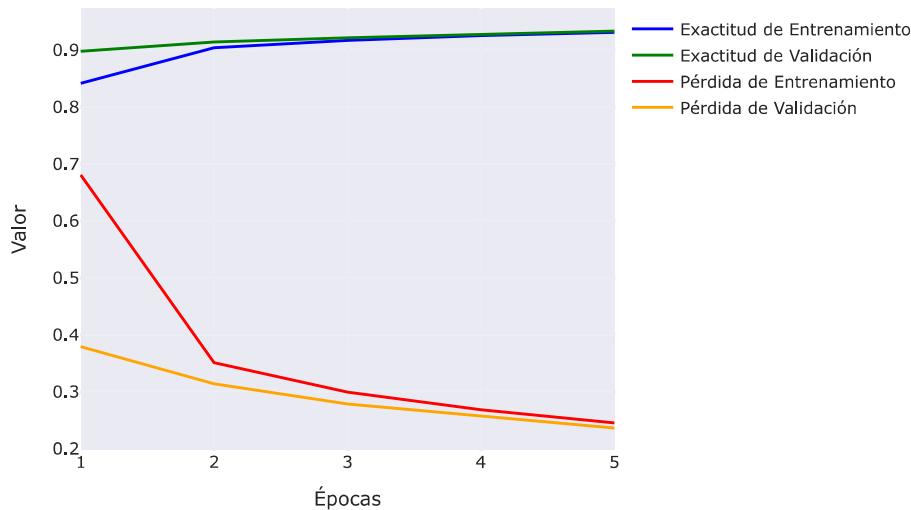
```

history_5_ReLU_RMSprop = model_5_ReLU_RMSprop.fit(training_images, training_labels,
    epochs=5,
    batch_size=32,
    validation_split=0.25)
plot_history(history_5_ReLU_RMSprop, "Modelo-5-ReLU-RMSprop")

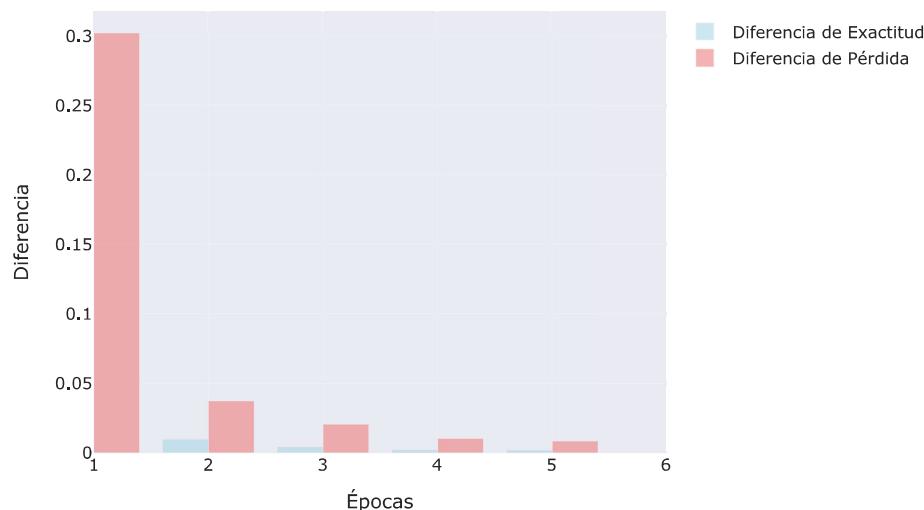
```

Epoch 1/5
1407/1407 3s 2ms/step - accuracy: 0.7543 - loss: 1.0586 - val_accuracy: 0.8979 - val_loss: 0.3791
Epoch 2/5
1407/1407 2s 2ms/step - accuracy: 0.9020 - loss: 0.3692 - val_accuracy: 0.9141 - val_loss: 0.3138
Epoch 3/5
1407/1407 2s 2ms/step - accuracy: 0.9153 - loss: 0.3068 - val_accuracy: 0.9215 - val_loss: 0.2788
Epoch 4/5
1407/1407 2s 2ms/step - accuracy: 0.9243 - loss: 0.2754 - val_accuracy: 0.9279 - val_loss: 0.2580
Epoch 5/5
1407/1407 2s 2ms/step - accuracy: 0.9308 - loss: 0.2474 - val_accuracy: 0.9336 - val_loss: 0.2365

Curvas de Exactitud y Pérdida (Modelo-5-ReLU-sgd)

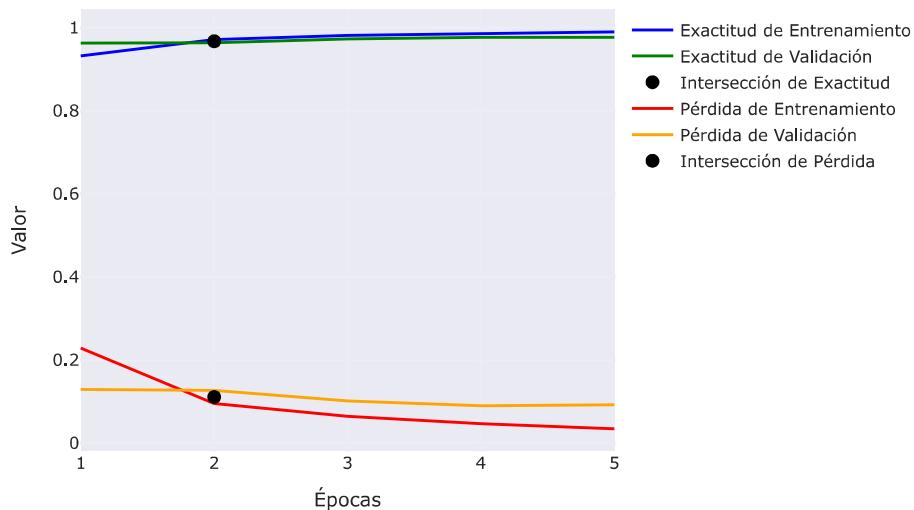


Diferencias entre Entrenamiento y Validación (Modelo-5-ReLU-sgd)

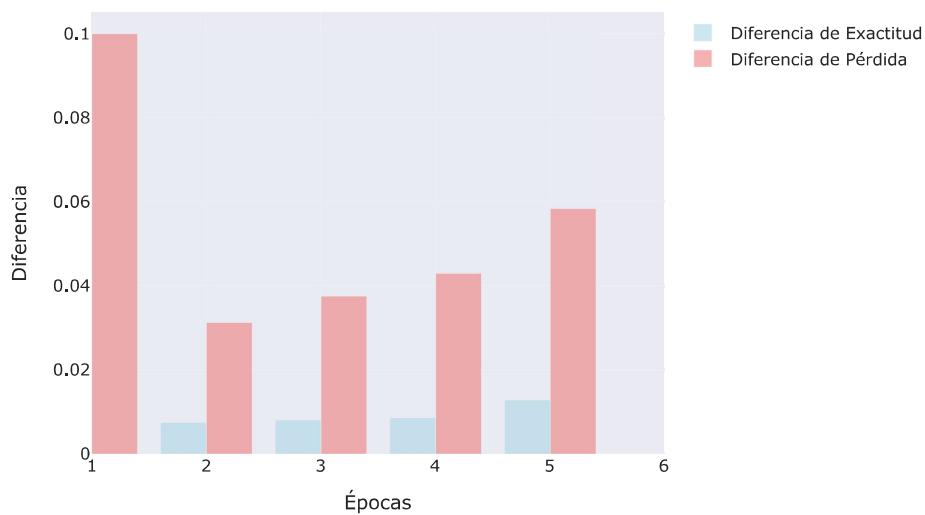


Epoch 1/5
1407/1407 4s 3ms/step - accuracy: 0.8901 - loss: 0.3708 - val_accuracy: 0.9633 - val_loss: 0.1288
Epoch 2/5
1407/1407 4s 3ms/step - accuracy: 0.9715 - loss: 0.0935 - val_accuracy: 0.9635 - val_loss: 0.1266
Epoch 3/5
1407/1407 4s 3ms/step - accuracy: 0.9817 - loss: 0.0623 - val_accuracy: 0.9731 - val_loss: 0.1016
Epoch 4/5
1407/1407 4s 3ms/step - accuracy: 0.9873 - loss: 0.0435 - val_accuracy: 0.9771 - val_loss: 0.0898
Epoch 5/5
1407/1407 7s 5ms/step - accuracy: 0.9903 - loss: 0.0333 - val_accuracy: 0.9770 - val_loss: 0.0924

Curvas de Exactitud y Pérdida (Modelo-5-ReLU-RMSprop)



Diferencias entre Entrenamiento y Validación (Modelo-5-ReLU-RMSprop)

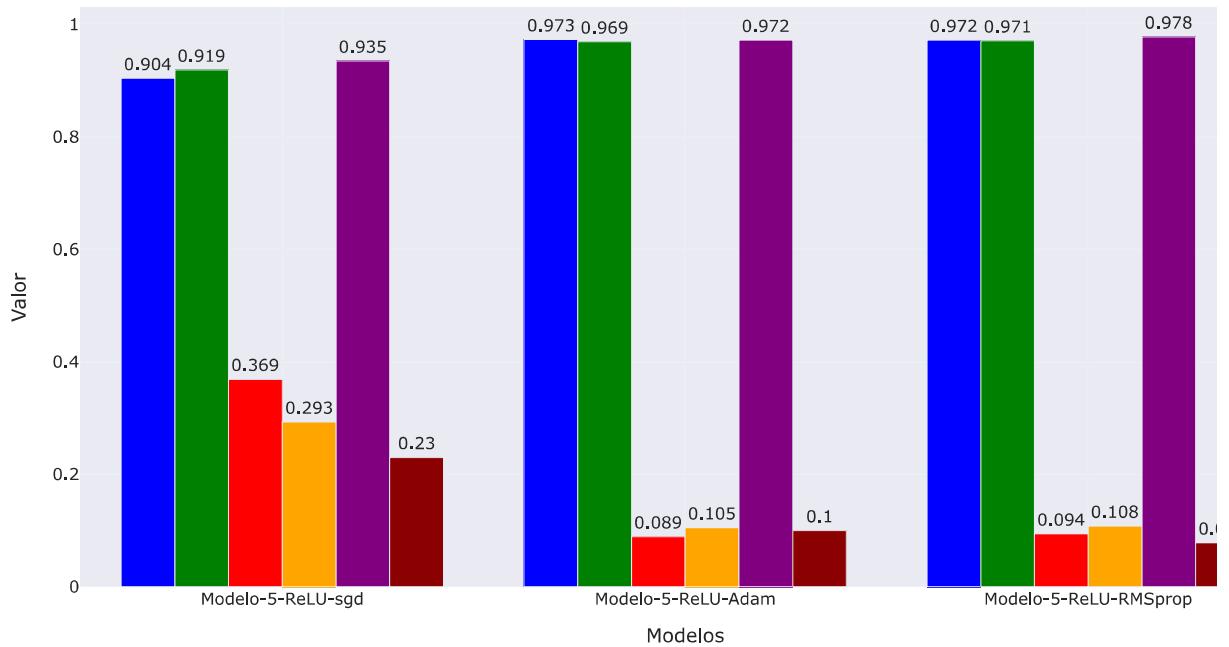


```
In [31]: # Crear Listas de histories, nombres de modelos y tests.
historias_6 = [history_5_ReLU_sgd, history_5_ReLU_glorot, history_5_ReLU_RMSprop]
modelos_6 = [ "Modelo-5-ReLU-sgd", "Modelo-5-ReLU-Adam", "Modelo-5-ReLU-RMSprop"]
tests_6 = [ model_5_ReLU_sgd.evaluate(test_imagenes,test_labels), model_5_ReLU_glorot.evaluate(test_imagenes,test_labels), mo

# Llamar a la función para mostrar el gráfico
plot_comparacion_modelos(historias_6, modelos_6, tests_6)

313/313 ━━━━━━ 0s 931us/step - accuracy: 0.9251 - loss: 0.2662
313/313 ━━━━━━ 0s 929us/step - accuracy: 0.9683 - loss: 0.1091
313/313 ━━━━ 0s 1ms/step - accuracy: 0.9750 - loss: 0.0929
```

Comparativa de Modelos



	Modelo	Media(Exactitud_entrenamiento)	Media(Exactitud_validacion)	Media(Exactitud_prueba)	Media(Perdida_entrenamiento)	Media(Perdida_prueba)
0	Modelo-5-ReLU-sgd	0.904	0.919	0.935	0.369	0.293
1	Modelo-5-ReLU-Adam	0.973	0.969	0.972	0.089	0.105
2	Modelo-5-ReLU-RMSprop	0.972	0.971	0.978	0.094	0.108

Estoy comparando SGD, ADAM y RMSprop. Ahora voy a comparar ReLU con Adam, pero modificando Adam con los parametros de learning_rate=0.0005 (que es la mitad de la default), y el batch size=16 (que es la mitad del actual). Con estos parametros tratamos de logran el aumento de la exactitud.

```
In [32]: from tensorflow.keras.optimizers import Adam

model_6_ReLU = Sequential([
    Input(shape=(784,)),
    Dense(512, activation='relu', kernel_initializer=initializers.glorot_uniform(), bias_initializer=initializers.glorot_uniform()),
    Dense(10, activation='softmax', kernel_initializer=initializers.glorot_uniform(), bias_initializer=initializers.glorot_uniform())
])

optimizador=Adam(learning_rate=0.0005)

model_6_ReLU.compile(optimizer,
                      loss='categorical_crossentropy',
                      metrics=['accuracy'])

history_6_ReLU = model_6_ReLU.fit(training_images, training_labels,
                                    epochs=5,
                                    batch_size=16,
                                    validation_split=0.25)

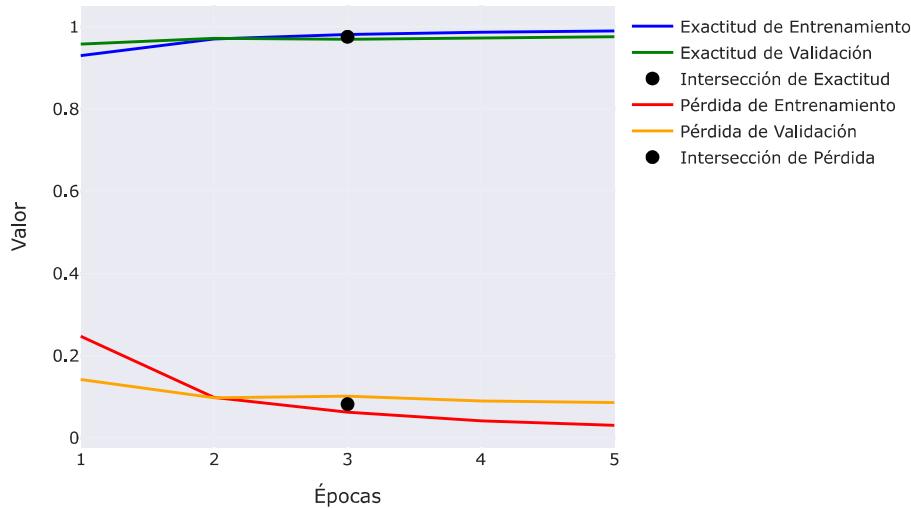
plot_history(history_6_ReLU, "Modelo-6-ReLU-batch-20")
```

```

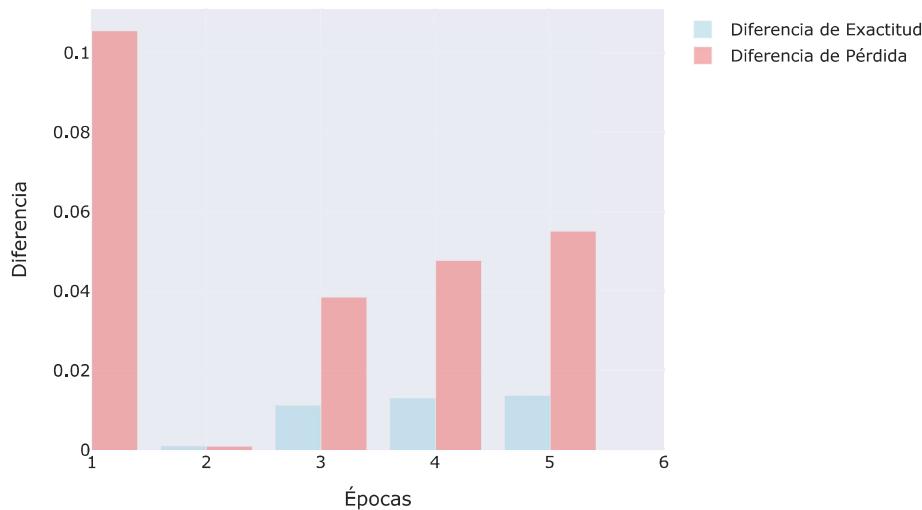
Epoch 1/5
2813/2813 8s 3ms/step - accuracy: 0.8829 - loss: 0.4154 - val_accuracy: 0.9581 - val_loss: 0.1418
Epoch 2/5
2813/2813 7s 2ms/step - accuracy: 0.9706 - loss: 0.1014 - val_accuracy: 0.9721 - val_loss: 0.0977
Epoch 3/5
2813/2813 7s 2ms/step - accuracy: 0.9825 - loss: 0.0617 - val_accuracy: 0.9701 - val_loss: 0.1015
Epoch 4/5
2813/2813 7s 3ms/step - accuracy: 0.9874 - loss: 0.0412 - val_accuracy: 0.9739 - val_loss: 0.0897
Epoch 5/5
2813/2813 10s 4ms/step - accuracy: 0.9915 - loss: 0.0276 - val_accuracy: 0.9763 - val_loss: 0.0862

```

Curvas de Exactitud y Pérdida (Modelo-6-ReLU-batch-20)



Diferencias entre Entrenamiento y Validación (Modelo-6-ReLU-batch-20)



```

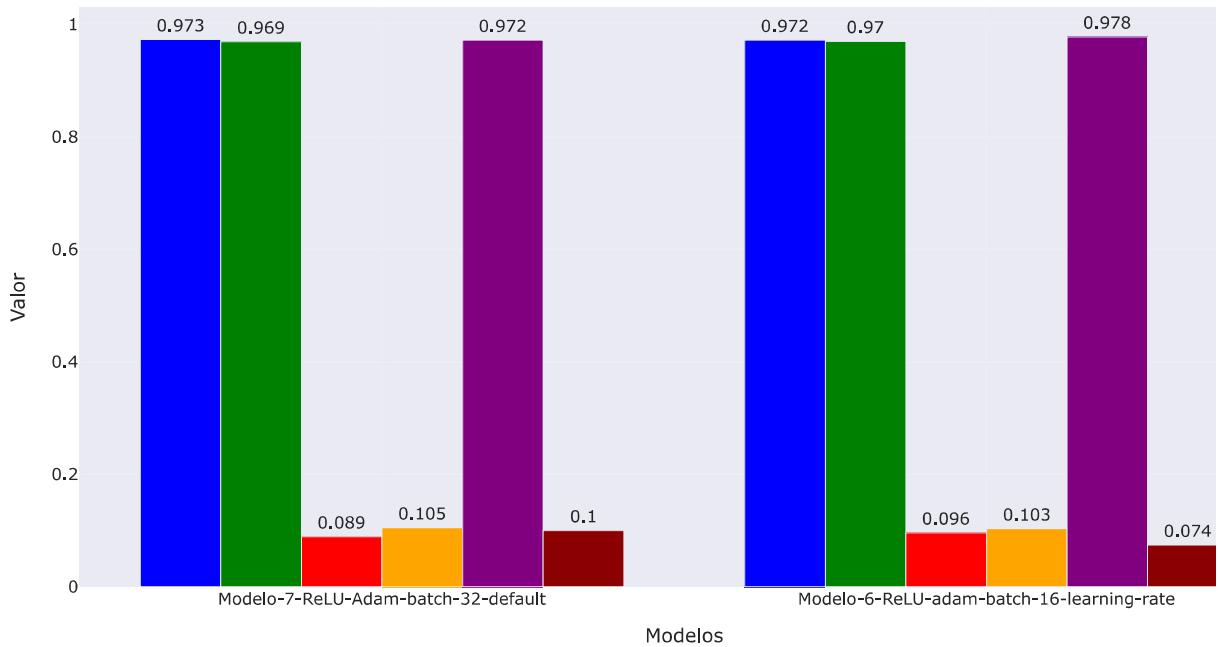
In [33]: # Crear listas de histories, nombres de modelos y tests.
historias_7 = [history_5_ReLU_glorot,history_6_ReLU]
modelos_7 = [ "Modelo-7-ReLU-Adam-batch-32-default","Modelo-6-ReLU-adam-batch-16-learning-rate"]
tests_7 = [ modelo_5_ReLU_glorot.evaluate(test_imagenes,test_labels), modelo_6_ReLU.evaluate(test_imagenes,test_labels) ]

# Llamar a la función para mostrar el gráfico
plot_comparacion_modelos(historias_7, modelos_7,tests_7)

313/313 0s 1000us/step - accuracy: 0.9683 - loss: 0.1091
313/313 0s 969us/step - accuracy: 0.9739 - loss: 0.0872

```

Comparativa de Modelos



	Modelo	Media(Exactitud_entrenamiento)	Media(Exactitud_validacion)	Media(Exactitud_prueba)	Media(Perdida_entrenamiento)	Media
0	Modelo-7-ReLU-Adam-batch-32-default	0.973	0.969	0.972	0.089	0.089
1	Modelo-6-ReLU-adam-batch-16-learning-rate	0.972	0.970	0.978	0.096	0.096

12. Regularización y red final (1.25 puntos)

Problema 12.1 *(2 puntos)*: Entrenar una red final que sea capaz de obtener una accuracy en el validation superior al 95%. Para ello, combinar todo lo aprendido anteriormente y utilizar técnicas de regularización para evitar overfitting. Algunos de los elementos que pueden tenerse en cuenta son los siguientes.

- Número de capas y neuronas por capa
- Optimizadores y sus parámetros
- Batch size
- Unidades de activación
- Uso de capas dropout, regularización L2, regularización L1...
- Early stopping (se puede aplicar como un callback de Keras, o se puede ver un poco "a ojo" cuándo el modelo empieza a caer en overfitting y seleccionar el número de epochs necesarias)
- Batch normalization

Si los modelos entrenados anteriormente ya se acercan al valor requerido de accuracy, probar distintas estrategias igualmente y comentar los resultados.

Explicar brevemente la estrategia seguida y los modelos probados para obtener el modelo final, que debe verse entrenado en este Notebook. No es necesario guardar el entrenamiento de todos los modelos que se han probado, es suficiente con explicar cómo se ha llegado al modelo final.

```
In [34]: ## Tu modelo y comentarios de texto aquí  
## Puedes incluir tantas celdas como quieras  
## pero recuerda visualizar la gráfica con la tasa de acierto  
## así como utilizar la función predict() para tu última evaluación
```

```
## (deberás consultar La documentación de Keras para entender la función)
## No olvides utilizar celdas de Markdown para texto
```

Con los resultados de los distintos apartados y experimentando con distintas configuraciones he llegado a este modelo:

.- He optado por dividir la capa densa en dos de 216 neuronas de forma que el modelo pueda aprender patrones mas complejos y patrones jerarquicos. Mis primeros modelos con una sola capa presentaron alto casos de fallo con digitos como 4 y 9. Al agregar mas capas favorezco el aprendizaje jerarquico. .- Dados los buenos resultados en convergencia de ReLU, la he seleccionado como la funcion de activacion de las capas ocultas. Ya que estamos empleando categorical_crossentropy la ultima capa es softmax para que distrituya de una forma eficiente las probabilidad en la capa final. .- Vimos que tenidamos un problema con la tendencia al sobreajuste, motivo por el que he agregado capas de dropout al 20%. .- El empleo de un learning rate de menor valor al por defecto tambien contribuyo a mejorar el entrenamiento, lo he dejado en 0.0001. .- He agregado early stop sobre la perdida en validacion para impedir aplicar demasiado entrenamiento. .- La inicializacion con valores aleatorios se mostro ligeramente mejor que glorot, asi que la he aplicado tambien.

```
In [35]: from tensorflow.keras.optimizers import Adam
from tensorflow.keras.layers import Dropout

modelo_final = Sequential([
    Input(shape=(784,)),
    Dense(512, activation='relu',kernel_initializer=initializers.glorot_uniform()),
    Dropout(0.3),
    Dense(128, activation='relu',kernel_initializer=initializers.glorot_uniform()),
    Dropout(0.3),
    Dense(10, activation='softmax',kernel_initializer=initializers.glorot_uniform())
])

optimizador=Adam(learning_rate=0.0005)

modelo_final.compile(optimizer=optimizador,
                      loss='categorical_crossentropy',
                      metrics=['accuracy']
                     )

modelo_final.summary()

class Callback_final(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs=None):
        if(logs is not None) and (logs.get('loss')< 0.1):
            print("\n Alcanzada una perdida menor al 20%, se cancela el entrenamiento!!")
            self.model.stop_training = True

callback=Callback_final()

history_final = modelo_final.fit(training_images, training_labels,
                                   epochs=15,
                                   batch_size=16,
                                   validation_split=0.25,
                                   callbacks=[callback]
                                  )

plot_history(history_final, "Modelo-Final")
```

Model: "sequential_14"

Layer (type)	Output Shape	Param #
dense_28 (Dense)	(None, 512)	401,920
dropout (Dropout)	(None, 512)	0
dense_29 (Dense)	(None, 128)	65,664
dropout_1 (Dropout)	(None, 128)	0
dense_30 (Dense)	(None, 10)	1,290

Total params: 468,874 (1.79 MB)

Trainable params: 468,874 (1.79 MB)

Non-trainable params: 0 (0.00 B)

Epoch 1/15

2813/2813 8s 3ms/step - accuracy: 0.8389 - loss: 0.5259 - val_accuracy: 0.9567 - val_loss: 0.1382

Epoch 2/15

2813/2813 8s 3ms/step - accuracy: 0.9569 - loss: 0.1437 - val_accuracy: 0.9688 - val_loss: 0.1018

Epoch 3/15

2813/2813 8s 3ms/step - accuracy: 0.9686 - loss: 0.0994 - val_accuracy: 0.9703 - val_loss: 0.0952

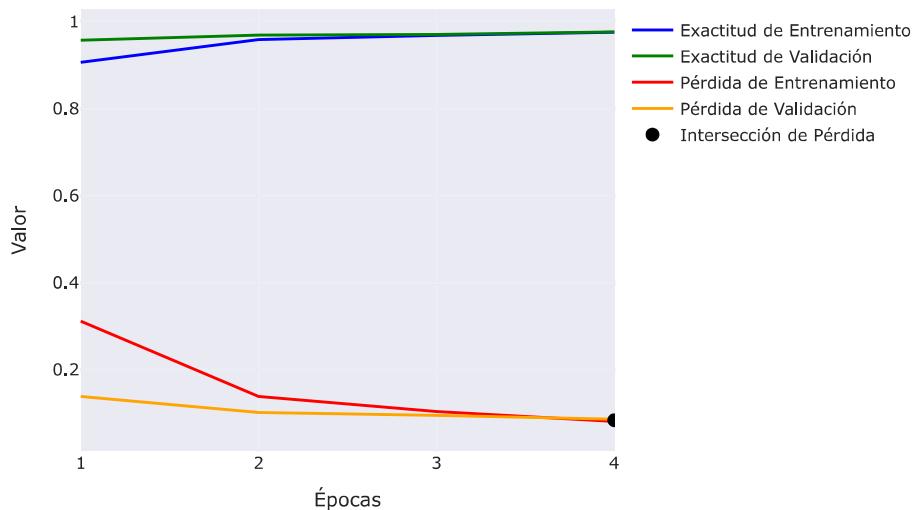
Epoch 4/15

2810/2813 0s 4ms/step - accuracy: 0.9743 - loss: 0.0832

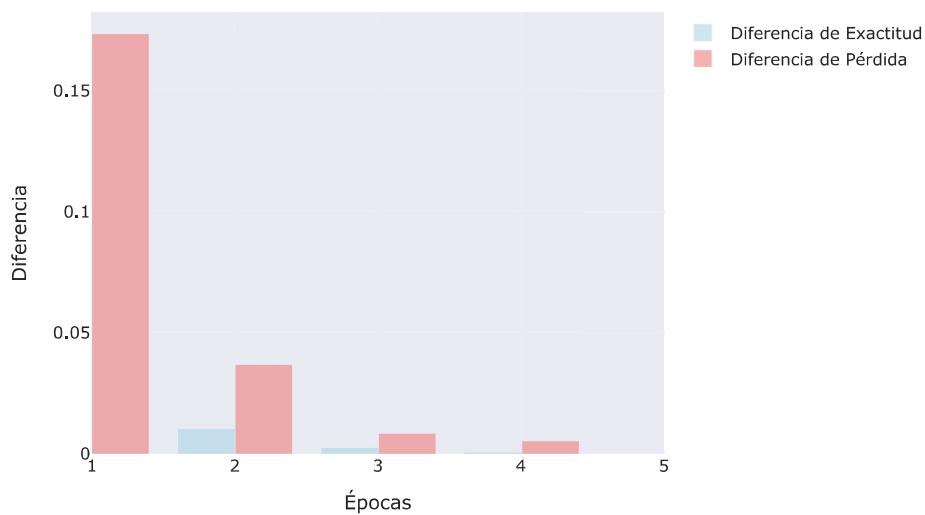
Alcanzada una perdida menor al 20%, se cancela el entrenamiento!!

2813/2813 11s 4ms/step - accuracy: 0.9743 - loss: 0.0832 - val_accuracy: 0.9761 - val_loss: 0.0865

Curvas de Exactitud y Pérdida (Modelo-Final)



Diferencias entre Entrenamiento y Validación (Modelo-Final)

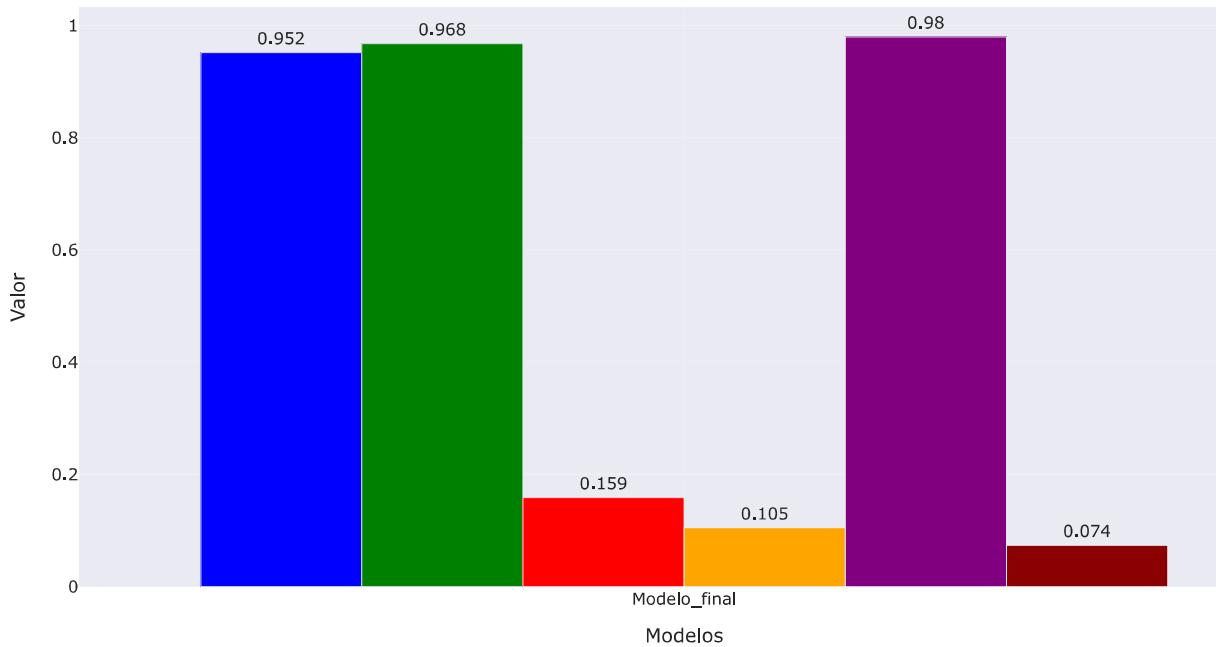


```
In [36]: # Crear Listas de histories, nombres de modelos y tests.
historias_final = [history_final]
modelos_final = [ "Modelo_final" ]
tests_final = [modelo_final.evaluate(test_imagenes,test_labels)]

# Llamar a la función para mostrar el gráfico
plot_comparacion_modelos(historias_final, modelos_final,tests_final)

313/313 ━━━━━━━━ 0s 1ms/step - accuracy: 0.9770 - loss: 0.0829
```

Comparativa de Modelos



```
Out[36]:
```

Modelo	Media(Exactitud_entrenamiento)	Media(Exactitud_validacion)	Media(Exactitud_prueba)	Media(Perdida_entrenamiento)	M
0 Modelo_final	0.952	0.968	0.98	0.159	

```
In [41]:
```

```
from sklearn.metrics import confusion_matrix, classification_report

predicciones = modelo_final.predict(test_imagenes)
clases_predichas = np.argmax(predicciones, axis=1)

test_labels_aux = np.argmax(test_labels, axis=1) if test_labels.ndim > 1 and test_labels.shape[1] > 1 else test_labels

matriz_confusion = confusion_matrix(test_labels_aux, clases_predichas)

mask = matriz_confusion == 0

plt.figure(figsize=(8, 6))
sns.heatmap(matriz_confusion, annot=True, fmt='d', cmap='YlGnBu', mask=mask, cbar=False, linewidths=0.5)
plt.title("Matriz de Confusión")
plt.xlabel("Clase Predicha")
plt.ylabel("Clase Verdadera")
plt.show()

reporte_clasificacion = classification_report(test_labels_aux, clases_predichas)
print("Reporte de Clasificación:\n", reporte_clasificacion)
```

313/313 ━━━━━━━━ 0s 1ms/step

Matriz de Confusión

	0	1	2	3	4	5	6	7	8	9	
0	969	1	1	1	4	1	1	1	2		
1		1119	7	1	1	2	1	4			
2	2		1015	5	1	2	5	2			
3			3	986	5	4	3	9			
4	1		4	952	6	3	2	14			
5	1		4	1	877	5	1	2	1		
6	3	2		3	8	940		2			
7		2	6	2	1		1010	1	6		
8	1		4	8	3	5	2	4	942	5	
9	2	3	1	2	5	3	1	5	1	986	
	0	1	2	3	4	5	6	7	8	9	

Reporte de Clasificación:

	precision	recall	f1-score	support
0	0.99	0.99	0.99	980
1	0.99	0.99	0.99	1135
2	0.98	0.98	0.98	1032
3	0.98	0.98	0.98	1010
4	0.99	0.97	0.98	982
5	0.97	0.98	0.98	892
6	0.98	0.98	0.98	958
7	0.98	0.98	0.98	1028
8	0.98	0.97	0.97	974
9	0.96	0.98	0.97	1009
accuracy		0.98	0.98	10000
macro avg	0.98	0.98	0.98	10000
weighted avg	0.98	0.98	0.98	10000

.- Por la matriz de confusión, vemos que todos los dígitos tienen una elevada probabilidad de ser reconocidos (0=99% - 9=96%). .. La sensibilidad es también elevada, el valor que presenta menor valor es 4=0.97, solo el 3% de los casos son mal clasificados para este dígito. .. El índice de F-score es bueno para todos los dígitos, los casos con cierta leve confusión con 4,8 y 9.