Open in Colab

Trabajo final redes neuronales

El presente material hace parte de la ruta de formación del talento especializado de SAPIENCIA

Los documentos que utilizaremos en la presente y proximas clases son una mezcla del trabajo de muchos profesores y académicos.

En caso de utilizar el presente contenido favor citarlo y brindar los créditos respectivos.

Entrega 1: Esteban Montoya Betancur

Link Git

Descripción trabajo final

1. Analizar el código paso a paso y entenderlo y realizar una breve explicación de como funciona.
2. Realizar módificaciones variando parámetros como la cantidad de neuronas, tipo de optimizador, funciones de activación, funciones de perdida y tamaños de entrada de la imagen.
3. A partir del entendimiento del código en el punto 1 responda las siguientes preguntas:

- ¿Cual es el objetivo de categorizar los targets o labels correspondientes a cada imagen?
- ¿En que me ayuda la normalización a la hora de entrenar los datos?

4. Realice un informe detallando los resultados obtenidos en el punto 2. El informe debe responder las siguiente preguntas:

- ¿Cómo variaron los resultados con el aumento o disminución de las neuronas?
- ¿Cómo cambia la presición del modelo propuesto, al cambiar la función de activación, que se logra observar de los resultados?
- ¿Cuál sería a su criterio la función de activación que se adapta al presente análisis ?
- ¿Cómo se comportaron los resultados de las funciones de perdidas analizadas?
- ¿Mejoraron los resultados al reducir o aumentar el tamaño de entrada de la imagen?
- ¿Cuál fue la mejor solución que logró encontrar y por qué?

Nota: El informe debe llevar los valores que probaron en el módelo y para lo cuál como minimo se deben analizar 4 optimizadores,4 funciones de activación, 4 funciones de perdida, 4 opciones de neuronas y tamaños de entrada de la imagen.
5. Concluir en que casos se debe utilizar los optimizadores,funciones de perdida, funciones de activación y tener en cuenta que se debe presentar una gráfica representativa de cada función de activación describiendo los rangos de la función y su comportamiento.

Porcentajes de calificación:

1. Punto 1 : 10%
2. Punto 2 : 10%
3. Punto 3 : 10%
4. Punto 4 : 25%
5. Punto 5 : 15%
6. Sustentación : 30%

Limitantes

Grupos máximo de 3 personas y mínimo de 2 personas

Respuesta a las preguntas

3.a Los labels se pasan a valores categóricos con el fin de poder ingresar a la capa de entrada de la red neuronal un vector de 0 y 1 en vez de un número entero, con el fin de proporcionar el input adecuado que necesita la red neuronal.

3.b La normalización a la hora de entrenar los datos ayuda en la manipulación de los valores, ya que se pasa de una matriz de dos dimensiones a un vector y los valores de este oscilarán entre 0 y 1.

Para iniciar este proyecto de redes neuronales el punto de partida es instalar las librerías TensorFlow y Keras

In [2]:
```
pip install tensorflow
```

```
Requirement already satisfied: tensorflow in c:\users\dell\appdata\local\progr
ams\python\python310\lib\site-packages (2.14.0)
Requirement already satisfied: tensorflow-intel==2.14.0 in c:\users\dell\appda
ta\local\programs\python\python310\lib\site-packages (from tensorflow) (2.14.
0)
Requirement already satisfied: flatbuffers>=23.5.26 in c:\users\dell\appdata\l
ocal\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.
0->tensorflow) (23.5.26)
Requirement already satisfied: typing-extensions>=3.6.6 in c:\users\dell\appda
ta\local\programs\python\python310\lib\site-packages (from tensorflow-intel==
2.14.0->tensorflow) (4.8.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in c:\users\dell\appdata\lo
cal\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0
->tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-estimator<2.15,>=2.14.0 in c:\users
\dell\appdata\local\programs\python\python310\lib\site-packages (from tensorfl
```

```
ow-intel==2.14.0->tensorflow) (2.14.0)
Requirement already satisfied: tensorboard<2.15,>=2.14 in c:\users\dell\appdat
a\local\programs\python\python310\lib\site-packages (from tensorflow-intel==2.
14.0->tensorflow) (2.14.1)
Requirement already satisfied: keras<2.15,>=2.14.0 in c:\users\dell\appdata\lo
cal\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0
->tensorflow) (2.14.0)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in c:\users
\dell\appdata\local\programs\python\python310\lib\site-packages (from tensorfl
ow-intel==2.14.0->tensorflow) (0.5.4)
Requirement already satisfied: libclang>=13.0.0 in c:\users\dell\appdata\local
\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->t
ensorflow) (16.0.6)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=
4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in c:\users\dell\appdata\local\programs\pyt
hon\python310\lib\site-packages (from tensorflow-intel==2.14.0->tensorflow)
(4.24.4)
Requirement already satisfied: numpy>=1.23.5 in c:\users\dell\appdata\local\pr
ograms\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->tens
orflow) (1.26.1)
Requirement already satisfied: setuptools in c:\users\dell\appdata\local\progr
ams\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->tensorf
low) (58.1.0)
Requirement already satisfied: termcolor>=1.1.0 in c:\users\dell\appdata\local
\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->t
ensorflow) (2.3.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\dell\appdata\local\p
rograms\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->ten
sorflow) (2.0.0)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\dell\appdata\loca
l\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->
tensorflow) (1.6.3)
Requirement already satisfied: h5py>=2.9.0 in c:\users\dell\appdata\local\prog
rams\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->tensor
flow) (3.10.0)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\dell\appdata\loca
l\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->
tensorflow) (3.3.0)
Requirement already satisfied: packaging in c:\users\dell\appdata\roaming\pyth
on\python310\site-packages (from tensorflow-intel==2.14.0->tensorflow) (23.2)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\dell\appdata\lo
cal\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0
->tensorflow) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\dell\appdata\lo
cal\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0
->tensorflow) (1.59.2)
Requirement already satisfied: ml-dtypes==0.2.0 in c:\users\dell\appdata\local
\programs\python\python310\lib\site-packages (from tensorflow-intel==2.14.0->t
ensorflow) (0.2.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in c:\user
s\dell\appdata\local\programs\python\python310\lib\site-packages (from tensorf
low-intel==2.14.0->tensorflow) (0.31.0)
Requirement already satisfied: six>=1.12.0 in c:\users\dell\appdata\roaming\py
thon\python310\site-packages (from tensorflow-intel==2.14.0->tensorflow) (1.1
6.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in c:\users\dell\appdata\loc
al\programs\python\python310\lib\site-packages (from astunparse>=1.6.0->tensor
flow-intel==2.14.0->tensorflow) (0.41.3)
```

```
Requirement already satisfied: google-auth-oauthlib<1.1,>=0.5 in c:\users\dell
\appdata\local\programs\python\python310\lib\site-packages (from tensorboard<
2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (1.0.0)
Requirement already satisfied: werkzeug>=1.0.1 in c:\users\dell\appdata\local
\programs\python\python310\lib\site-packages (from tensorboard<2.15,>=2.14->te
nsorflow-intel==2.14.0->tensorflow) (3.0.1)
Requirement already satisfied: google-auth<3,>=1.6.3 in c:\users\dell\appdata
\local\programs\python\python310\lib\site-packages (from tensorboard<2.15,>=2.
14->tensorflow-intel==2.14.0->tensorflow) (2.23.3)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in c:\use
rs\dell\appdata\local\programs\python\python310\lib\site-packages (from tensor
board<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (0.7.2)
Requirement already satisfied: markdown>=2.6.8 in c:\users\dell\appdata\local
\programs\python\python310\lib\site-packages (from tensorboard<2.15,>=2.14->te
nsorflow-intel==2.14.0->tensorflow) (3.5)
Requirement already satisfied: requests<3,>=2.21.0 in c:\users\dell\appdata\lo
cal\programs\python\python310\lib\site-packages (from tensorboard<2.15,>=2.14-
>tensorflow-intel==2.14.0->tensorflow) (2.31.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in c:\users\dell\appdata
\local\programs\python\python310\lib\site-packages (from google-auth<3,>=1.6.3
->tensorboard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (5.3.2)
Requirement already satisfied: pyasn1-modules>=0.2.1 in c:\users\dell\appdata
\local\programs\python\python310\lib\site-packages (from google-auth<3,>=1.6.3
->tensorboard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (0.3.0)
Requirement already satisfied: rsa<5,>=3.1.4 in c:\users\dell\appdata\local\pr
ograms\python\python310\lib\site-packages (from google-auth<3,>=1.6.3->tensorb
oard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in c:\users\dell\appda
ta\local\programs\python\python310\lib\site-packages (from google-auth-oauthli
b<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow)
(1.3.1)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\dell\appdata\loc
al\programs\python\python310\lib\site-packages (from requests<3,>=2.21.0->tens
orboard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (2.0.7)
Requirement already satisfied: charset-normalizer<4,>=2 in c:\users\dell\appda
ta\local\programs\python\python310\lib\site-packages (from requests<3,>=2.21.0
->tensorboard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (3.3.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\dell\appdata\loc
al\programs\python\python310\lib\site-packages (from requests<3,>=2.21.0->tens
orboard<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (2023.7.22)
Requirement already satisfied: idna<4,>=2.5 in c:\users\dell\appdata\local\pro
grams\python\python310\lib\site-packages (from requests<3,>=2.21.0->tensorboar
d<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (3.4)
Requirement already satisfied: MarkupSafe>=2.1.1 in c:\users\dell\appdata\loca
l\programs\python\python310\lib\site-packages (from werkzeug>=1.0.1->tensorboa
rd<2.15,>=2.14->tensorflow-intel==2.14.0->tensorflow) (2.1.3)
Requirement already satisfied: pyasn1<0.6.0,>=0.4.6 in c:\users\dell\appdata\l
ocal\programs\python\python310\lib\site-packages (from pyasn1-modules>=0.2.1->
google-auth<3,>=1.6.3->tensorboard<2.15,>=2.14->tensorflow-intel==2.14.0->tens
orflow) (0.5.0)
Requirement already satisfied: oauthlib>=3.0.0 in c:\users\dell\appdata\local
\programs\python\python310\lib\site-packages (from requests-oauthlib>=0.7.0->g
oogle-auth-oauthlib<1.1,>=0.5->tensorboard<2.15,>=2.14->tensorflow-intel==2.1
4.0->tensorflow) (3.2.2)
Note: you may need to restart the kernel to use updated packages.

WARNING: You are using pip version 22.0.4; however, version 23.3.1 is availabl
e.
```

```
You should consider upgrading via the 'c:\Users\DELL\AppData\Local\Programs\Py
thon\Python310\python.exe -m pip install --upgrade pip' command.
```

In [3]:
```
pip install keras
```

```
Requirement already satisfied: keras in c:\users\dell\appdata\local\programs\p
ython\python310\lib\site-packages (2.14.0)
Note: you may need to restart the kernel to use updated packages.
WARNING: You are using pip version 22.0.4; however, version 23.3.1 is availabl
e.
You should consider upgrading via the 'c:\Users\DELL\AppData\Local\Programs\Py
thon\Python310\python.exe -m pip install --upgrade pip' command.
```

Se importan las principales librerías para trabajar en ciencia de datos tales como: Numpy, Pandas y Matplotlib, además, algunas funciones importantes de keras para la manipulación del conjunto de datos MNIST.

In [4]:
```
import numpy as np
from keras import layers, models
from keras.utils import to_categorical
from keras.datasets import mnist
import matplotlib.pyplot as plt
import pandas as pd
```

Una vez importadas las librerías, se procede a obtener el dataset de mnist, que consiste en matrices numéricas equivalentes a imágenes de 28 x 28 pixeles, en las cuales, cada entrada de la matriz corresponde a un pixel y su valor oscila entre 0 y 255, asociado a los valores de la escala de grises, que finalmente conforman un dígito escrito a mano. Este dataset está dividido en cuatro partes, que consisten en los datos de entrenamiento y sus etiquetas; y en los datos de prueba y sus etiquetas.

A partir del dataset de mnist, es necesario esquematizar los datos para poder correr el modelo, como primera medida se necesita hacer un reshape de los datos de entrenamiento para cambiar la forma de los datos, es decir, cada imagen al ser una matriz de 28 x 28, no puede ser ingresada como input de una red neuronal de esta forma, sino que debe ser una fila de 784 entradas, es decir, pasar la matriz a un vector que represente la imagen. Esto se almacenará en la variable `train_data_flattened` . Adicionalmente, como los datos de mnist se almacenan como arreglos de numpy, se convertirá a un DataFrame de pandas y se almacenará en la variable `train_data_df`

In [5]:
```
(train_data, train_labels), (test_data, test_labels) =mnist.load_data()
# Aplana las imágenes y conviértelas en un DataFrame
train_data_flattened = train_data.reshape(train_data.shape[0], -1)
train_data_df = pd.DataFrame(train_data_flattened)
```

Las próximas líneas son comprobaciones que sirven para ilustrar lo realizado en la casilla anterior y para explorar un poco mejor los datos, no son indispensables para el entendimiento del código en general.

In [6]:
```
type(train_data_flattened)
```

Out[6]:
```
numpy.ndarray
```

In [7]:
```
train_data
```

Out[7]:
```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
```

```
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]],

      [[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]],

      [[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]],

      ...,

      [[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]],

      [[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]],

      [[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]]], dtype=uint8)
```

In [8]:
```
train_data_df.head(10)
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

	0	1	2	3	4	5	6	7	8	9	...	774	775	776	777	778	779	780	781	782	783
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

10 rows × 784 columns

In [9]:
```
train_data_df.shape
```

Out[9]:
```
(60000, 784)
```

In [10]:
```
train_data.shape
```

Out[10]:
```
(60000, 28, 28)
```

In [11]:
```
train_data_df[689]
```

Out[11]:
```
0         0
1         0
2        96
3         0
4        42
        ..
59995     0
59996     0
59997     0
59998     0
59999     0
Name: 689, Length: 60000, dtype: uint8
```

A continuación, se ejemplifica mejor el contenido del dataset mnist, observando un elemento aleatorio de su componente de entrenamiento, en este caso el número 689. Primero, se observa la matriz 28x28, donde sus primeros elementos son valores en 0, que representarían casillas en blanco y hacia el centro va tomando diversos valores hasta alrededor de 255, que a mayor número, son más oscuros. Estos valores representan el color de cada pixel, que a su vez conforman un dígito. A través del método `imshow` de matplotlib, se puede visualizar de forma gráfica el contenido de estas matrices, y finalmente, en la variable de etiquetas se encuentra almacenado el valor, interpretado previamente, de cada dígito, en este caso, correspondiente al valor de 0.

In [12]:
```
train_data[689]
```

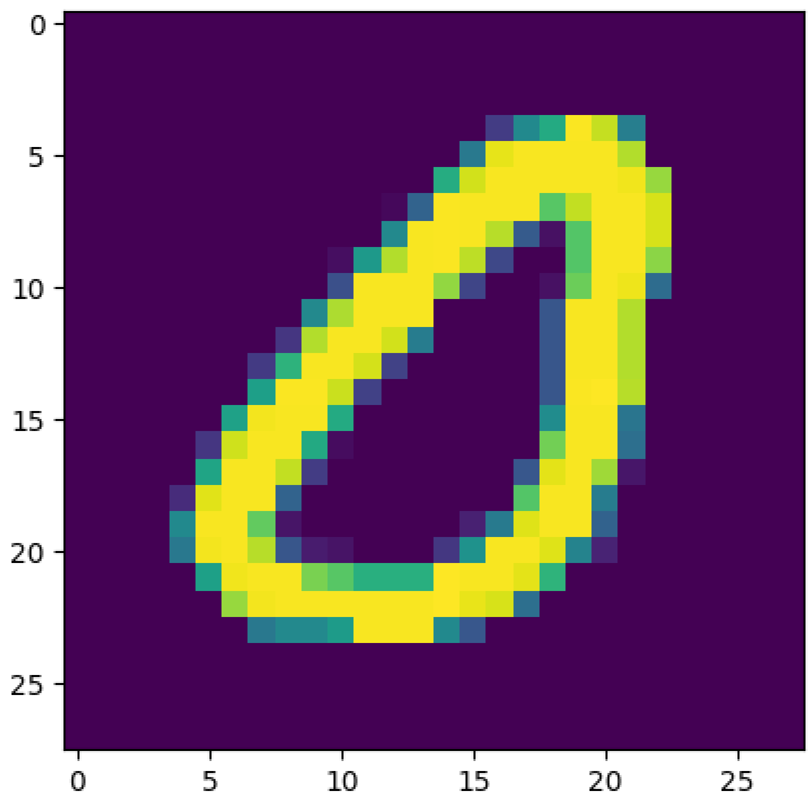
Out[12]:
```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

```
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,  45, 121, 156, 254, 233, 109,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0, 103, 246, 253, 253, 253, 253, 226,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0, 158, 238, 253, 253, 253, 253, 253, 250, 216,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   5,
  81, 254, 253, 253, 253, 189, 232, 253, 253, 240,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 120,
 253, 254, 253, 228,  73,  11, 187, 253, 253, 240,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   9, 138, 226,
 253, 254, 230,  55,   0,   0, 187, 253, 253, 211,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  62, 253, 253,
 253, 214,  53,   0,   0,  11, 198, 253, 249,  90,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,   0, 120, 224, 253, 253,
 253,   0,   0,   0,   0,  68, 253, 253, 226,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,   0,  40, 226, 253, 253, 238,
 107,   0,   0,   0,   0,  68, 253, 253, 226,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0,  43, 165, 253, 253, 239,  50,
   0,   0,   0,   0,   0,  68, 253, 253, 226,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0,   0, 144, 254, 254, 235,  49,   0,
   0,   0,   0,   0,   0,  68, 254, 255, 228,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,   0, 146, 251, 253, 253, 156,   0,   0,
   0,   0,   0,   0,   0, 124, 253, 253,  99,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0,  40, 237, 253, 253, 155,   8,   0,   0,
   0,   0,   0,   0,   0, 201, 253, 253,  93,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0, 149, 253, 253, 232,  45,   0,   0,   0,
   0,   0,   0,   0,  69, 245, 253, 219,  15,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,  32, 244, 253, 253,  81,   0,   0,   0,   0,
   0,   0,   0,   0, 187, 253, 253, 107,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0, 121, 253, 253, 194,  14,   0,   0,   0,   0,
   0,   0,  23, 104, 243, 253, 253,  80,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0, 102, 251, 253, 228,  68,  19,  13,   0,   0,
   0,  41, 130, 253, 253, 243, 114,  25,   0,   0,   0,   0,   0,
   0,   0],
[  0,   0,   0,   0,   0, 145, 250, 253, 253, 204, 189, 161, 161,
 161, 255, 253, 253, 245, 166,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
```

```
       [  0,   0,   0,   0,   0,   0, 216, 251, 253, 253, 253, 253, 253,
        253, 255, 247, 240,  93,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0, 102, 120, 120, 140, 253, 253,
        253, 121,  69,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0]], dtype=uint8)
```

In [13]:
```
plt.imshow(train_data[689])
```

Out[13]:
```
<matplotlib.image.AxesImage at 0x1ffa6a6d570>
```



In [14]:
```
train_labels[689]
```

Out[14]:
```
0
```

A la hora de crear modelos con Keras, existen tres opciones: El modelo secuencial, la API funcional y modelo subclase. El modelo secuencial, que será utilizado en este caso, es ideal para redes neuronales que consisten en un sólo tensor de entrada y uno sólo de salida. Dado que este es el caso del presente ejercicio y que los otros modelos son para otros casos específicos, se procederá a usar el modelo secuencial. Para esto se define el modelo en la variable `model` .

Una vez creado el modelo, se agregan las capas necesarias para definir la red neuronal. Esto se puede hacer como un atributo al definir el modelo o con el método `add` después de definirlo. En este caso se usó el método `add` .

En el presente caso, se crea una red neuronal densa con una capa oculta **?** de 512 neuronas y una función de activación ReLU, cuya capa de entrada es de tamaño 784 (28*28), que corresponde a la cantidad de pixeles de cada imagen. Su capa de salida tiene un tamaño de 10, el cual corresponde a los dígitos entre 0-9, que son las salidas posibles de cada imagen. Su función de activación es softmax.

In [15]:
```
model = models.Sequential()
model.add(layers.Dense(512,activation='relu', input_shape=(28*28,)))
model.add(layers.Dense(10,activation='softmax'))
```

Con la arquitectura del modelo ya definida, se realiza la compilación, donde se especifica el optimizador a utilizar en el entrenamiento, la función de pérdida y las métricas con que se va a medir el desempeño del entrenamiento, en este caso la exactitud y la precisión.

In [16]:
```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy', 'Precision'])
```

A continuación, se obtendrá un resumen del modelo, donde se muestran las dos capas y su respectivo número de parámetros a analizar. En el caso de la primera capa, corresponde a 401.920 parámetros que se obtienen de multiplicar las 512 neuronas por los 784 pixeles de entrada

In [17]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 512)               401920

 dense_1 (Dense)             (None, 10)                5130

=================================================================
Total params: 407050 (1.55 MB)
Trainable params: 407050 (1.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Las siguientes líneas de código darán inicio al **pre procesamiento en los datos antes de entrenar el modelo de red neuronal.** Básicamente lo que se hace es tranformar la matriz en un vector, donde cada entrada de este representan los píxeles de la imagen (matriz), luego, convierten los valores a tipo números de punto flotante y se procede con su normalización, es decir, dividir cada valor de píxel por 255 para tenerlos escalados al rango (0,1). Por último, se repiten ambos procedimientos pero con los datos de prueba.

In [18]:
```
x_train = train_data_df
#x_train = train_data.reshape((60000,28*28))
x_train = x_train.astype('float32')/255

x_test = test_data.reshape((10000,28*28))
x_test = x_test.astype('float32')/255
```

In [19]:
```
x_train[0]
```

Out[19]:
```
0        0.0
1        0.0
2        0.0
3        0.0
4        0.0
        ...
59995    0.0
59996    0.0
59997    0.0
59998    0.0
59999    0.0
Name: 0, Length: 60000, dtype: float32
```

El dataset de mnist, inicialmente proporciona las etiquetas como un número entero entre 0 y 9, asociado a los dígitos numéricos de las imágenes. Sin embargo, para ingresar estas etiquetas a la red neuronal, es necesario convertir estos valores en un vector categórico, correspondiente a los valores entre 0 y 9, es decir [0,1,2,3,4,5,6,7,8,9], para lo cual, la etiqueta correspondiente será un 1 según la posición y el resto de valores corresponderán a 0. Un ejemplo de lo anterior, sería el de la posición `train_label[689]` , cuya etiqueta es 0. Al convertir a categórico, sería el vector [1,0,0,0,0,0,0,0,0,0].

A continuación se hará dicha conversión y se almacenará en las variables `y_train` y `y_test` , la primera para las etiquetas de entrenamiento y la segunda para las etiquetas de prueba:

In [20]:
```
y_train = to_categorical(train_labels)
y_test =to_categorical(test_labels)
```

In [21]:
```
train_labels[0]
```

Out[21]:
```
5
```

In [22]:
```
y_train[0]
```

Out[22]:
```
array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)
```

A continuación, se importa TensorBoard, que sirve para realizar visualizaciones de los datos a entrenar

In [23]:
```
from tensorflow.keras.callbacks import TensorBoard
```

In [24]:
```
tensorboardDenso = TensorBoard(log_dir ='logs/denso')
```

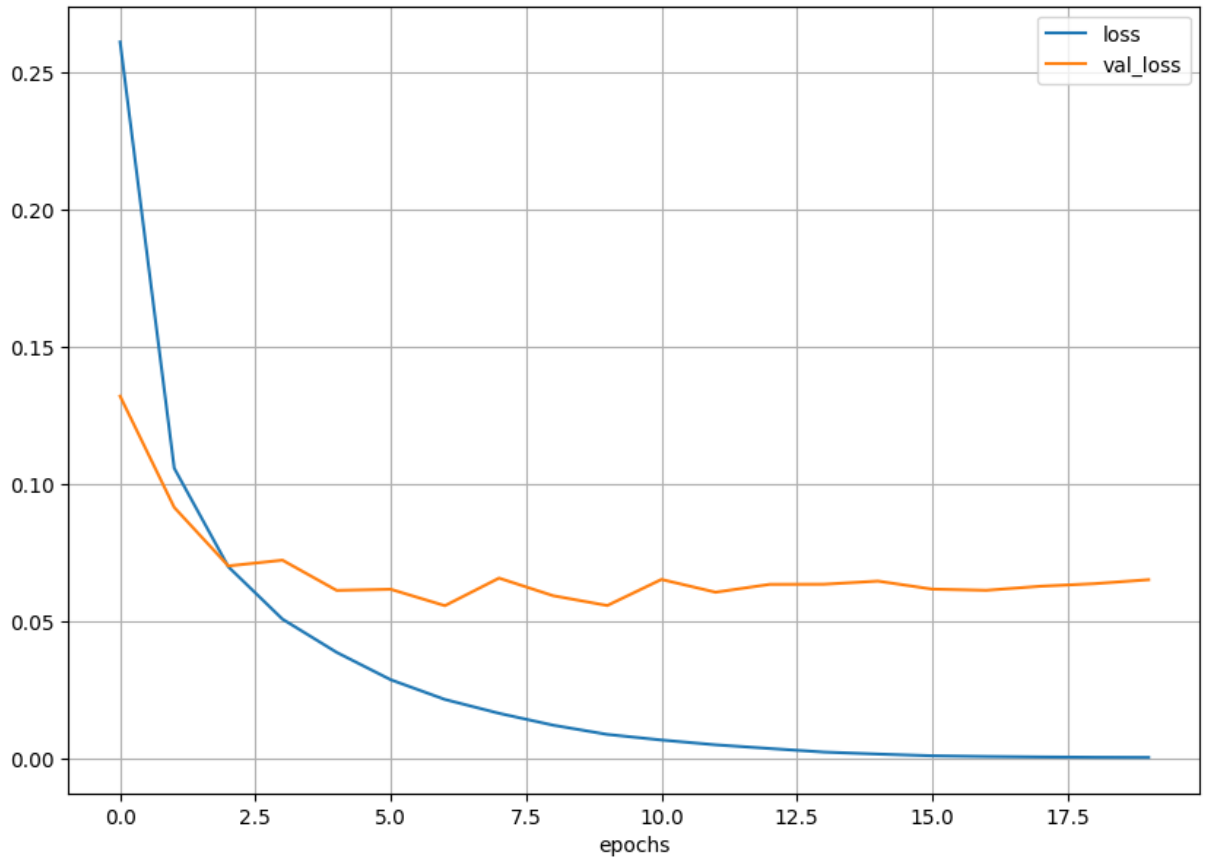
A continuación, se inicia el proceso de entrenamiento de la red neuronal utilizando los conjuntos de datos de entrenamiento x_train (imágenes pre procesadas) e y_train (etiquetas). A su vez, se especifica el número de **épocas** que durará el proceso de entrenamiento, cabe mencionar que una época es una pasada completa a través de todo el conjunto de entrenamiento. Por último, se especifica el tamaño del lote (batch size), que es el número de muestras que se utilizarán en cada iteración del proceso de entrenamiento. En este caso, se están utilizando lotes de 128 muestras. Durante el entrenamiento, **el modelo ajusta sus parámetros (pesos y sesgos) utilizando el optimizador (RMSprop en este caso) y la función de pérdida (categorical crossentropy) que fueron configurados previamente.** El objetivo es minimizar la función de pérdida y mejorar la precisión del modelo en el conjunto de entrenamiento.

In [25]:
```
history = model.fit(x_train, y_train, epochs=20,callbacks=[tensorboardDenso],
batch_size=128, validation_data=(x_test, y_test))
```

```
Epoch 1/20
469/469 [==============================] - 13s 22ms/step - loss: 0.2623 - accu
racy: 0.9252 - precision: 0.9524 - val_loss: 0.1232 - val_accuracy: 0.9637 - v
al_precision: 0.9715
Epoch 2/20
469/469 [==============================] - 8s 18ms/step - loss: 0.1061 - accur
acy: 0.9689 - precision: 0.9748 - val_loss: 0.0840 - val_accuracy: 0.9732 - va
l_precision: 0.9778
Epoch 3/20
469/469 [==============================] - 8s 16ms/step - loss: 0.0705 - accur
acy: 0.9783 - precision: 0.9821 - val_loss: 0.0859 - val_accuracy: 0.9708 - va
l_precision: 0.9756
Epoch 4/20
469/469 [==============================] - 8s 16ms/step - loss: 0.0508 - accur
acy: 0.9849 - precision: 0.9869 - val_loss: 0.0755 - val_accuracy: 0.9763 - va
l_precision: 0.9799
Epoch 5/20
469/469 [==============================] - 11s 23ms/step - loss: 0.0387 - accu
racy: 0.9883 - precision: 0.9901 - val_loss: 0.0620 - val_accuracy: 0.9809 - v
al_precision: 0.9825
Epoch 6/20
469/469 [==============================] - 8s 18ms/step - loss: 0.0288 - accur
acy: 0.9917 - precision: 0.9926 - val_loss: 0.0646 - val_accuracy: 0.9786 - va
l_precision: 0.9810
Epoch 7/20
469/469 [==============================] - 8s 16ms/step - loss: 0.0224 - accur
acy: 0.9934 - precision: 0.9942 - val_loss: 0.0555 - val_accuracy: 0.9833 - va
l_precision: 0.9845
Epoch 8/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0170 - accu
racy: 0.9954 - precision: 0.9960 - val_loss: 0.0652 - val_accuracy: 0.9796 - v
al_precision: 0.9814
Epoch 9/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0132 - accur
acy: 0.9964 - precision: 0.9968 - val_loss: 0.0594 - val_accuracy: 0.9808 - va
l_precision: 0.9823
Epoch 10/20
469/469 [==============================] - 9s 19ms/step - loss: 0.0094 - accur
acy: 0.9979 - precision: 0.9981 - val_loss: 0.0601 - val_accuracy: 0.9823 - va
l_precision: 0.9833
Epoch 11/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0072 - accur
acy: 0.9984 - precision: 0.9985 - val_loss: 0.0617 - val_accuracy: 0.9816 - va
l_precision: 0.9828
Epoch 12/20
469/469 [==============================] - 9s 20ms/step - loss: 0.0058 - accur
acy: 0.9988 - precision: 0.9988 - val_loss: 0.0628 - val_accuracy: 0.9820 - va
l_precision: 0.9833
Epoch 13/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0035 - accu
racy: 0.9994 - precision: 0.9995 - val_loss: 0.0628 - val_accuracy: 0.9830 - v
al_precision: 0.9846
Epoch 14/20
469/469 [==============================] - 8s 18ms/step - loss: 0.0025 - accur
acy: 0.9996 - precision: 0.9997 - val_loss: 0.0722 - val_accuracy: 0.9808 - va
```

```
l_precision: 0.9817
Epoch 15/20
469/469 [==============================] - 9s 18ms/step - loss: 0.0018 - accur
acy: 0.9998 - precision: 0.9998 - val_loss: 0.0657 - val_accuracy: 0.9825 - va
l_precision: 0.9832
Epoch 16/20
469/469 [==============================] - 10s 21ms/step - loss: 0.0012 - accu
racy: 0.9999 - precision: 0.9999 - val_loss: 0.0649 - val_accuracy: 0.9835 - v
al_precision: 0.9841
Epoch 17/20
375/469 [======================>.......] - ETA: 1s - loss: 8.0896e-04 - accura
cy: 1.0000 - precision: 1.0000
```

In [ ]:
```
pd.DataFrame({'loss': history.history['loss'],
              'val_loss': history.history['val_loss']}).plot(figsize=(10, 7))
plt.grid(True)
plt.xlabel("epochs")
plt.show()
```



A continuación se evaluará la red neuronal con el conjunto de datos de prueba. **El método devuelve una lista que contiene la métrica de pérdida (el valor de la función de pérdida evaluada en el conjunto de prueba) y las métrica de exactitud (accuracy) y precisión (precision) en el conjunto de prueba, que se especificaron en la compilación del modelo.** Por ejemplo, si la salida es [0.045, 0.985, 0.985], significa que la pérdida en el conjunto de prueba es 0.045 y la exactitud y precisión son del 98.5%. Esta evaluación es importante para entender cómo el modelo generaliza a datos que no ha visto durante el entrenamiento y proporciona una indicación de su rendimiento en situaciones reales.

In [ ]:
```
model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0651 - accura
cy: 0.9849 - precision: 0.9856
[0.06512981653213501, 0.9848999977111816, 0.9855899214744568]
```

Out[ ]:

Finalmente, se usará el entorno de tensorboard para poder visualizar de forma gráfica el desempeño del entrenamiento

In [ ]:
```
%load_ext tensorboard
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
```

In [ ]:
```
 %reload_ext tensorboard
```

In [ ]:
```
%tensorboard --logdir logs
```

In [ ]:
```
!kill 8369
```