```
ccuracy: 0.8402
Epoch 10/10
391/391 [==============================] - 4s 10ms/step - loss: 0.3851 - accuracy: 0.8648 - val_loss: 0.4717 - val_a
ccuracy: 0.8483
[0.5125571489334106, 0.8346999883651733]
```

1.3 Tareas

Vamos a comenzar normalizando los datos de entrada según tres criterios: escalar los valores de entrada al rango 0-1, centrar a una media aproximada de 0 y transformar los datos de entrada aproximadamente a una distribución normal de media 0 y desviación unidad (N(0,1)).

A continuación, cambiaremos el criterio de parada del entrenamiento del número máximo de iteraciones (épocas) a terminar el entrenamiento cuando se cumplan unas ciertas condiciones en un subconjunto de los datos u opcionalmente en un conjunto de validación (independiente del entrenamiento).

1.3.1 Normalización 1: escalado de los valores al rango (0, 1) [0.5 pto]

A partir del código anterior, realizar las modificaciones necesarias para que los valores de las imágenes estén escalados al rango (0, 1).

In [3]:

```python
'''Función que nos permitirá hacer:
    - El split al dataset de train, valid y test.
    - La prepación de las imagenes al tamaño y tipo esperados por la DNN.
    - Devuelve el dataset de train, valid y test'''

def prepare_dataset (train_dataset,test_dataset):
    # Primeras 10000 imágenes, las utilizamos como validación
    X_valid = train_dataset[:10000]
    X_train = train_dataset[10000:]

    X_test = test_dataset

    X_train = X_train.reshape(X_train.shape[0], 28*28)
    X_valid = X_valid.reshape(X_valid.shape[0], 28*28)
    X_test = X_test.reshape(X_test.shape[0], 28*28)

    X_train = X_train.astype('float32')
    X_valid = X_valid.astype('float32')
    X_test = X_test.astype('float32')

    return X_train, X_valid, X_test

```

executed in 24ms, finished 18:19:51 2021-10-21

In [4]:
```
# TODO 1

'''Partimos de los dataset originales y les aplicamos la división entre 255,
ya que los valores de los pixeles contenidos en los dataset se mueven en el rango de 0 a 255
por lo que después de noramlilzarlos el rango estará entre (0,1)'''

train_scaled = train_images /255.0
test_scaled = test_images /255.0
print(f'Máximo del dataset de entreno {np.max(train_scaled):0.4f}')
print(f'Mínimo del dataset de entreno {np.min(train_scaled):0.4f}')
print(f'Máximo del dataset de test {np.max(test_scaled):0.4f}')
print(f'Mínimo del dataset de test {np.min(test_scaled):0.4f}')


X_train_scaled,X_valid_scaled,X_test_scaled = prepare_dataset(train_scaled, test_scaled)
```

executed in 430ms, finished 18:19:52 2021-10-21

```
Máximo del dataset de entreno 1.0000
Mínimo del dataset de entreno 0.0000
Máximo del dataset de test 1.0000
Mínimo del dataset de test 0.0000
```

1.3.2 Normalización 2: centrar a una media aproximada de 0 [0.5 pto]

AYUDA: Para centrar los valores a una media aproximada de 0, puedes calcular la media total y restarsela a todos los datos. Recuerda que la información de los datos de evaluación (test) no se puede utilizar, pero deben llevar el mismo procesamiento que los datos con los que se entrena la red.

In [5]:
```
# TODO 2
'''Partimos de los dataset originales y procedemos a
aplicarle la normalización de media aproximada a 0. Otra opción y tal vez más optima en cuanto a rango de valores
utilizar los dataset con la reducción de escala del apartado anterior y proceder a centrar entorno a la media.'''

train_mean_scaled = train_images - np.mean(train_images)
test_mean_scaled= test_images - np.mean(test_images)

print(f'Media del dataset de entreno {np.mean(train_mean_scaled):0.4f}')
print(f'Media del dataset de test {np.mean(test_mean_scaled):0.4f}')

X_train_mean_scaled,X_valid_mean_scaled,X_test_mean_scaled = prepare_dataset(train_mean_scaled, test_mean_scaled)
```

executed in 525ms, finished 18:19:52 2021-10-21

```
Media del dataset de entreno -0.0000
Media del dataset de test -0.0000
```

1.3.3 Normalización 3: distribución normal de media 0 y desviación stándard 1 (estandarización N(0,1)) [0.5 pto]

AYUDA: Para estandarizar los valores a una distribución aproximadamente normal N(0, 1), puedes calcular la media y la desviación total y aplicar la normalización: x_norm = (x - media)/desviacion.

Recuerda que la información de los datos de evaluación (test) no se puede utilizar, pero deben llevar el mismo procesamiento que los datos con los que se entrena la red.

In [6]:
```
############## Si al ejecutar el Kernel se bloquea,
############## utiliza estas líneas para permitir la
############## duplicación de librerías
import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
##############

```

executed in 12ms, finished 18:19:52 2021-10-21

In [7]:
```
'''Partimos de los dataset originales. Calculamos su media y su desviación tipica
y aplicamos la formula de normalización N(0,1)'''

train_standarized = (train_images - np.mean(train_images))/np.std(train_images)
test_standarized = (test_images - np.mean(test_images))/np.std(test_images)

print(f'Media del dataset de entreno {np.mean(train_standarized):0.4f}')
print(f'Desviación típica del dataset de entreno {np.std(train_standarized):0.4f}')
print(f'Media del dataset de test {np.mean(test_standarized):0.4f}')
print(f'Desviación típica del dataset de test {np.std(test_standarized):0.4f}')

X_train_standarized,X_valid_standarized,X_test_standarized = prepare_dataset(train_standarized, test_standarized)
```

executed in 1.97s, finished 18:19:54 2021-10-21

```
Media del dataset de entreno -0.0000
Desviación típica del dataset de entreno 1.0000
Media del dataset de test -0.0000
Desviación típica del dataset de test 1.0000
```

¿Por qué es recomendable hacer estas normalizaciones? ¿Ha mejorado el resultado? ¿Por qué? ¿Con cuál se obtiene el mejor resultado? [1 pto]

In [8]:

```
'''Función que nos permitirá crear, compilar, entrenar y validar marcadores con el dataset de test.
Se le pasa como parámetros los dataset de test, valid, y test así como las respetivas etiquetas.
También se le pueden pasar parámetros como el número de epochs, verbose,si queremos logs o no y callbacks.'''

def exec_DNN (x_train, y_train, x_valid, y_valid, x_test, y_test,my_epochs=10,my_verbose=0, my_callbacks=None):
    model = Sequential()
    model.add(Dense(512, activation='relu', input_shape=(28*28,)))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(10, activation='softmax'))

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

    model.fit(x_train, y_train,
              batch_size=128,
              epochs=my_epochs,
              verbose=my_verbose,
              validation_data=(x_valid, y_valid),
              callbacks=my_callbacks
             )

    model_scores = model.evaluate(x_test, y_test, verbose=my_verbose,)

    print(f'\nLa Evaluación del modelo con los datos de test es:')
    for indice in range(0,len(model.metrics_names),1):
        print(f'El Marcador {model.metrics_names[indice]} del modelo con datos de test es : {model_scores[indice]}')
    return None
```

executed in 25ms, finished 18:19:54 2021-10-21

In [9]:

```
# Ejecutamos la DNN con el dataset escalado con rango (0,1)
exec_DNN(X_train_scaled,Y_train,X_valid_scaled,Y_valid,X_test_scaled,Y_test,my_verbose=0)
```

executed in 40.5s, finished 18:20:35 2021-10-21

```
La Evaluación del modelo con los datos de test es:
El Marcador loss del modelo con datos de test es : 0.3320726454257965
El Marcador accuracy del modelo con datos de test es : 0.8859999775886536
```

In [10]:
```
# Ejecutamos la DNN con el dataset escalado con valores entorno a una media 0
exec_DNN(X_train_mean_scaled,Y_train,X_valid_mean_scaled,Y_valid,X_test_mean_scaled,Y_test)
```

executed in 40.8s, finished 18:21:15 2021-10-21

```
La Evaluación del modelo con los datos de test es:
El Marcador loss del modelo con datos de test es : 0.4595426619052887
El Marcador accuracy del modelo con datos de test es : 0.8537999987602234
```

In [11]:
```
# Ejecutamos la DNN con el dataset escalado con valores normalizados entorno a una N(0,1)
exec_DNN(X_train_standarized,Y_train,X_valid_standarized,Y_valid,X_test_standarized,Y_test)
```

executed in 39.8s, finished 18:21:55 2021-10-21

```
La Evaluación del modelo con los datos de test es:
El Marcador loss del modelo con datos de test es : 0.3723970353603363
El Marcador accuracy del modelo con datos de test es : 0.8851000070571899
```

- ***Respuesta: ¿Por qué es recomendable hacer estas normalizaciones?***:
  - Al igual que en muchos modelos de machine learning tanto supervisados como no supervisados, las redes neuronales son sensibles al rango de datos con el que trabajan, siendo más fácil para el modelo trabajar con valores pequeños y rangos de valores pequeños y acotados.

- ***Respuesta: ¿Ha mejorado el resultado?***:
  - Si ha mejorado bastante tanto en la función de coste, entorno a 0,20, como en el acurracy entorno a 0,05. Salvo en la normalización entorno a la media 0, que los ofrece algo más similares a los marcadores cuando no teniamos los datos normalizados.

- ***Respuesta: ¿Por qué?***:
  - Por que al estar todos los valores en la misma escala y con valores pequeños y rangos acotados, los calculos se facilitan y el coste de proceso de la red neuronal es mucho menos y más rapido.
  - Así también tenemos todos los valores bajo la misma escala y por lo tanto el calculo de optimización de la función con el learning rate se hace con la misma escala. Si no se tuviera así se podrían perder o no calcular correctamente los minimos en alguno de los valores.
  - Para el caso con media entorno a 0 más o menos ofrece los mismos valores que si no se hubiera realizado nada. Pese a reducir el rango de valores, sigue siendo un rango muy amplio. A esta normalización entorno a media 0, habría que haber realizado previamente, una reducción del rango como la del apartado 1. Así hubiera sido más efectiva y mejor.

- ***Respuesta: ¿Con cuál se obtiene el mejor resultado?***
  - En nuestro caso se obtienen mejores resultados en cuanto al accuracy o precisión con los valores **escalados en un rango (0,1)**, aunque las diferencias son mínimas con respecto a la normalización conforme a la normal N(0,1), por detrás se queda la normalización entorno a la media 0.

1.3.4 Ajuste de la tasa de aprendizaje para optimizar el rendimiento de la red

¿Qué sucede si elegimos una tasa de aprendizaje demasiado alta? ¿Y una demasiado baja? Explica brevemente qué es la tasa de aprendizaje o "learning rate" y cómo afecta a nuestro entrenamiento: [1 pto]

- **Respuesta1**: Si elegimos una tasa de aprendizaje demasiado alta ganaremos rapidez en el módelo, pero por el contrario será más impreciso ya que lo más probable es que no llegue bien a calcular el error mínimo.
- **Respuesta2**: Si por el contario elegimos una tasa de aprendizaje demasiado baja, vamos a penalizar el rendimento del modelo, ya que al calcular el valor optimo de error, dando "pasos" muy cortos, realizará muchas iteraciones hasta llegar al mínimo, por lo que el modelo requerira de un tiempo muy grande para converger al punto óptimo.
- **Respuesta3**:
  - A nivel de definición la tasa de aprendizaje o learning rate, dentro del algoritmo de descenso de gradiente, es el tamaño del "paso" que damos para volver a calcular el nuevo punto en el que se vuelve a calcular el gradiente. Todo esto con el fin de que en n pasos hayamos encontrado el valor mínimo que optimiza la función de coste. Más formalmente es el número que se le multiplica al gradiente de la recta en ese punto y se le resta al valor de ese punto, la formula se expresaria de esta manera, dado una variable aleatoria w cuyo valor inicial es aleatorio, la formula del descenso de gradiente es Wn+1 = Wn-ta.Gradiente_en_Wn:
    $$W_{n+1} = W_n - ta.\nabla(W_n) \quad (1)$$
    Siendo ta -> Tasa de Aprendizaje
  - La tasa de aprendizaje o learning rate en nuestro entrenamiento afecta desde el punto de vista que debemos encontrar el valor óptimo y normalmente es uno de los hiperparámetros que siempre se debe ajustar, para conseguir un número mínimo de iteracciones con un mayor acercamiento al punto mínimo de error. Valores como el 1 ya harían que el algoritmo no convergiera a ningún valor y estaría rebotando entre los mismos valores. Por experiencia normalmente se obtienen buenos valores y rendimiento con valores entre el 0,01 y el 0,1, pero esto siempre hay que probarlo para cada caso.

Muchas veces, cuando la función de coste llega a una zona cercana al mínimo, la tasa de aprendizaje es muy grande para alcanzar el valor óptimo. Por eso, una de las técnicas utilizadas para evitar este problema consiste en reducir la tasa de aprendizaje cuando llegamos a un punto en que no vemos mejora del rendimiento en nuestro conjunto de validación.

Para ello, podemos utilizar uno de los Callbacks de Keras llamado: ReduceLROnPlateau. Puedes encontrar la información sobre él en el siguiente enlace: https://keras.io/callbacks/#reducelronplateau (https://keras.io/callbacks/#reducelronplateau)

Investiga (la documentación de keras es sencilla y muy muy útil, pero puedes tirar de google) cómo implementar un callback. Después, implementa dicho callback: puedes empezar con el código anterior, con una paciencia de 2 iteraciones y una reducción del 50% del valor de la tasa de aprendizaje.

Es posible que tengas que aumentar las iteraciones máximas para ver mejor su funcionamiento. [1 pto]

- **Respuesta :** Vamos a utilizar dos CallBack:
  - Un Callback a medida o Custom, **LossAndAccuracyPrintingCallback**, que hereda de la clase Callback y lo haremos para perosnalizar alguno de los mensajes
  - El propuesto, **ReduceLROnPlateau**, para ir bajando el learning rate o tasa de aprendizaje cada vez que la red detecte que ya no hay mejora en los marcadores. En este caso se nos propone un **patiente=2** y una disminución del learning rate del 50%, esto se hace configurando el parámetro **factor=0.5**, también configuraremos el parámetro verbose=1 para que nos vaya informando cuando se realiza algún cambio en el learning rate.

In [12]:
```
# Creamos un callback personalizado que hereda de la clase Callback, con el objetivo de visualizar
# los marcadores al final de cada epoch.
class LossAndAccuracyPrintingCallback(tf.keras.callbacks.Callback):

#   def on_train_batch_end(self, batch, logs=None):
#       print('Para el batch de entrenamiento {}, la perdida (loss) es {:7.2f}.'.format(batch, logs['loss']))

#   def on_test_batch_end(self, batch, logs=None):
#       print('Para el  batch de validación {}, la perdida (loss) es {:7.2f}.'.format(batch, logs['loss']))

  def on_epoch_end(self, epoch, logs=None):

    print(f"En la epoch {epoch+1} el marcador val_loss para validación es {logs['val_loss']:7.4f} y el val_accurac
```

executed in 11ms, finished 18:21:55 2021-10-21

In [13]:

```
from tensorflow.keras.callbacks import ReduceLROnPlateau

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2,verbose=1)
my_callbacks=[reduce_lr,LossAndAccuracyPrintingCallback()]
# my_callbacks=[reduce_lr]
```

executed in 26ms, finished 18:21:55 2021-10-21

In [14]:
```
my_epoch=20 # Para ver como actua el ReduceLROnPlateau subimos los epochs
my_verbose=0 # flag con el que activaremos o desctavaremos los logs.

exec_DNN(X_train_standarized,
         Y_train,
         X_valid_standarized,
         Y_valid,
         X_test_standarized,
         Y_test,my_epoch,
         my_verbose,
         my_callbacks)
```

executed in 1m 22.9s, finished 18:23:18 2021-10-21

```
En la epoch 1 el marcador val_loss para validación es  0.3601 y el val_accuracy es  0.8685
En la epoch 2 el marcador val_loss para validación es  0.3498 y el val_accuracy es  0.8719
En la epoch 3 el marcador val_loss para validación es  0.3057 y el val_accuracy es  0.8887
En la epoch 4 el marcador val_loss para validación es  0.3217 y el val_accuracy es  0.8860

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
En la epoch 5 el marcador val_loss para validación es  0.3207 y el val_accuracy es  0.8875
En la epoch 6 el marcador val_loss para validación es  0.2925 y el val_accuracy es  0.8959
En la epoch 7 el marcador val_loss para validación es  0.2911 y el val_accuracy es  0.8967
En la epoch 8 el marcador val_loss para validación es  0.2986 y el val_accuracy es  0.8972

Epoch 00009: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
En la epoch 9 el marcador val_loss para validación es  0.3197 y el val_accuracy es  0.8885
En la epoch 10 el marcador val_loss para validación es  0.2958 y el val_accuracy es  0.9036

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
En la epoch 11 el marcador val_loss para validación es  0.3149 y el val_accuracy es  0.8986
En la epoch 12 el marcador val_loss para validación es  0.3089 y el val_accuracy es  0.9029

Epoch 00013: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
En la epoch 13 el marcador val_loss para validación es  0.3175 y el val_accuracy es  0.9021
En la epoch 14 el marcador val_loss para validación es  0.3174 y el val_accuracy es  0.9047

Epoch 00015: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
En la epoch 15 el marcador val_loss para validación es  0.3173 y el val_accuracy es  0.9027
En la epoch 16 el marcador val_loss para validación es  0.3213 y el val_accuracy es  0.9037

Epoch 00017: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
```

```
En la epoch 17 el marcador val_loss para validación es  0.3237 y el val_accuracy es  0.9034
En la epoch 18 el marcador val_loss para validación es  0.3240 y el val_accuracy es  0.9039

Epoch 00019: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.
En la epoch 19 el marcador val_loss para validación es  0.3253 y el val_accuracy es  0.9037
En la epoch 20 el marcador val_loss para validación es  0.3257 y el val_accuracy es  0.9037

La Evaluación del modelo con los datos de test es:
El Marcador loss del modelo con datos de test es : 0.3541863262653351
El Marcador accuracy del modelo con datos de test es : 0.9007999897003174
```

In [15]:
```
exec_DNN(X_train_mean_scaled,
         Y_train,
         X_valid_mean_scaled,
         Y_valid,
         X_test_mean_scaled,
         Y_test,
         my_epoch,
         my_verbose,
         my_callbacks)
```

executed in 1m 22.3s, finished 18:24:41 2021-10-21

```
En la epoch 1 el marcador val_loss para validación es  0.8510 y el val_accuracy es  0.8011
En la epoch 2 el marcador val_loss para validación es  0.5744 y el val_accuracy es  0.8371
En la epoch 3 el marcador val_loss para validación es  0.5321 y el val_accuracy es  0.8481
En la epoch 4 el marcador val_loss para validación es  0.4892 y el val_accuracy es  0.8474
En la epoch 5 el marcador val_loss para validación es  0.5001 y el val_accuracy es  0.8488
En la epoch 6 el marcador val_loss para validación es  0.4712 y el val_accuracy es  0.8557
En la epoch 7 el marcador val_loss para validación es  0.4611 y el val_accuracy es  0.8538
En la epoch 8 el marcador val_loss para validación es  0.4639 y el val_accuracy es  0.8670
En la epoch 9 el marcador val_loss para validación es  0.4192 y el val_accuracy es  0.8717
En la epoch 10 el marcador val_loss para validación es  0.4313 y el val_accuracy es  0.8708

Epoch 00011: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
En la epoch 11 el marcador val_loss para validación es  0.4792 y el val_accuracy es  0.8509
En la epoch 12 el marcador val_loss para validación es  0.3749 y el val_accuracy es  0.8831
En la epoch 13 el marcador val_loss para validación es  0.3890 y el val_accuracy es  0.8809

Epoch 00014: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
En la epoch 14 el marcador val_loss para validación es  0.4108 y el val_accuracy es  0.8780
En la epoch 15 el marcador val_loss para validación es  0.3741 y el val_accuracy es  0.8885
En la epoch 16 el marcador val_loss para validación es  0.3884 y el val_accuracy es  0.8891

Epoch 00017: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
En la epoch 17 el marcador val_loss para validación es  0.3987 y el val_accuracy es  0.8856
En la epoch 18 el marcador val_loss para validación es  0.4050 y el val_accuracy es  0.8887

Epoch 00019: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
En la epoch 19 el marcador val_loss para validación es  0.4100 y el val_accuracy es  0.8892
En la epoch 20 el marcador val_loss para validación es  0.4081 y el val_accuracy es  0.8926

La Evaluación del modelo con los datos de test es:
```

```
El Marcador loss del modelo con datos de test es : 0.43250221014022827
El Marcador accuracy del modelo con datos de test es : 0.8827000260353088
```

In [16]:
```
exec_DNN(X_train_standarized,
         Y_train,
         X_valid_standarized,
         Y_valid,X_test_standarized,
         Y_test,
         my_epoch,
         my_verbose,
         my_callbacks)
```

executed in 1m 19.3s, finished 18:26:00 2021-10-21

```
En la epoch 1 el marcador val_loss para validación es  0.3826 y el val_accuracy es  0.8636
En la epoch 2 el marcador val_loss para validación es  0.3379 y el val_accuracy es  0.8721
En la epoch 3 el marcador val_loss para validación es  0.3243 y el val_accuracy es  0.8796
En la epoch 4 el marcador val_loss para validación es  0.3381 y el val_accuracy es  0.8794

Epoch 00005: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
En la epoch 5 el marcador val_loss para validación es  0.3328 y el val_accuracy es  0.8777
En la epoch 6 el marcador val_loss para validación es  0.2870 y el val_accuracy es  0.8993
En la epoch 7 el marcador val_loss para validación es  0.2967 y el val_accuracy es  0.8978

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
En la epoch 8 el marcador val_loss para validación es  0.2963 y el val_accuracy es  0.8972
En la epoch 9 el marcador val_loss para validación es  0.2949 y el val_accuracy es  0.9024

Epoch 00010: ReduceLROnPlateau reducing learning rate to 0.0001250000059371814.
En la epoch 10 el marcador val_loss para validación es  0.3036 y el val_accuracy es  0.9007
En la epoch 11 el marcador val_loss para validación es  0.2966 y el val_accuracy es  0.9017

Epoch 00012: ReduceLROnPlateau reducing learning rate to 6.25000029685907e-05.
En la epoch 12 el marcador val_loss para validación es  0.3034 y el val_accuracy es  0.9027
En la epoch 13 el marcador val_loss para validación es  0.2994 y el val_accuracy es  0.9047

Epoch 00014: ReduceLROnPlateau reducing learning rate to 3.125000148429535e-05.
En la epoch 14 el marcador val_loss para validación es  0.3054 y el val_accuracy es  0.9021
En la epoch 15 el marcador val_loss para validación es  0.3083 y el val_accuracy es  0.9024

Epoch 00016: ReduceLROnPlateau reducing learning rate to 1.5625000742147677e-05.
En la epoch 16 el marcador val_loss para validación es  0.3099 y el val_accuracy es  0.9050
En la epoch 17 el marcador val_loss para validación es  0.3110 y el val_accuracy es  0.9034

Epoch 00018: ReduceLROnPlateau reducing learning rate to 7.812500371073838e-06.
```

```
En la epoch 18 el marcador val_loss para validación es  0.3119 y el val_accuracy es  0.9026
En la epoch 19 el marcador val_loss para validación es  0.3115 y el val_accuracy es  0.9039

Epoch 00020: ReduceLROnPlateau reducing learning rate to 3.906250185536919e-06.
En la epoch 20 el marcador val_loss para validación es  0.3124 y el val_accuracy es  0.9035

La Evaluación del modelo con los datos de test es:
El Marcador loss del modelo con datos de test es : 0.34904801845550537
El Marcador accuracy del modelo con datos de test es : 0.8989999890327454
```

Analiza los resultados: ¿Qué hace el callback? ¿Mejora ahora el resultado? ¿Por qué? [0.5]

- ***Respuesta ¿Qué hace el callback? :***
  - Según lo que podemos observar, y se hace más evidente cuando subimos a 20 epochs, en cuanto hay poca mejora en el loss o perdida, aplica una reducción del learnning rate o tasa de aprendizaje de lo paramétrizado en el parámetro factor, que en este caso es **factor=0.5** que equivale a una reducción cada vez que lo reduce del 50%. Empezando con una reducción de **lr** desde **0.0005000000237487257.**, por lo que lr con el que arranco era de 0,001, hasta el **lr 3.906250185536919e-06** que nos ha aparecido en alguna de las ejecuciones de la red.

- ***Respuesta ¿Mejora ahora el resultado? :***
  - Si los resultados se ven mejorados sobre todo el Accuracy/Precisión que para el test ya llega 90% de acierto en el dataset con normalización (0,1). Sin embargo para la función de coste o loss los valores se mantienen entorno al 0.33-0.34 cuando se validan contra test.

- ***Respuesta ¿Por qué? :***
  - Por que al ir reduciendo el **lr**, los pasos que va dando hasta encontrar el valor mínimo u optimo, son cada vez más pequeños y por lo tanto ese valor mínimo es más seguro encontrarlo. Por otro lado la contra está en que contra mayor es el lr, mayor es el coste computacional por que hay que hacer más iteracciones hasta llegar a el.

In [ ]:
```
1
```