


ML_CLASIFICACION_RN_06	Clasificación de dígitos escritos a mano	ML
<p>El problema a resolver es la clasificación de imágenes (en escala de grises, 28 x 28 píxeles) de dígitos escritos a mano, en las 10 categorías (0 a 9). Utilizamos para ello la base de datos MNIST (http://yann.lecun.com/exdb/mnist/), elaborada por el NIST (National Institute of Standards and Technology) y compuesta por un conjunto de 60.000 imágenes para entrenamiento, y 10.000 imágenes para prueba y validación.</p> <div data-bbox="576 654 1043 761" data-label="Image">  </div> <p>Ejemplos de imágenes (28x28) de dígitos escritos a mano</p> <p>La base de datos MNIST ya viene precargada dentro de la librería Keras. La instalación de Keras se puede realizar directamente desde el entorno Anaconda.</p>		
<h3><u>SOLUCIÓN</u></h3> <p>Definir las librerías a utilizar</p> <pre># Importar librerías a utilizar import numpy as np import matplotlib.pyplot as plt</pre> <p>Cargamos en Python la base de datos MNIST, y asignamos las imágenes y etiquetas de los conjuntos para entrenamiento y prueba.</p> <pre># Cargar la base de datos MNIST y asignar las imágenes y etiquetas de los conjuntos para entrenamiento y prueba from keras.datasets import mnist (X_train, y_train), (X_test, y_test) = mnist.load_data()</pre> <p>Las imágenes están codificadas en arrays (0 o 1), y las etiquetas son un array de números (0 a 9). Realizamos la consulta de algunas imágenes y etiquetas para entrenamiento.</p> <pre># Las imágenes están codificadas en arrays (0 o 1), y las etiquetas son un array números (0 a 9) print("X_train shape", X_train.shape) print("y_train shape", y_train.shape) print("X_test shape", X_test.shape) print("y_test shape", y_test.shape) X_train shape (60000, 28, 28) y_train shape (60000,) X_test shape (10000, 28, 28) y_test shape (10000,)</pre>		

Visualizar algunos ejemplos de la base de datos

```
# Consultar algunas imágenes y etiquetas para entrenamiento
# Visualizar algunos ejemplos
fig = plt.figure()
for i in range(9):
    plt.subplot(3,3,i+1)
    plt.tight_layout()
    plt.imshow(X_train[i], cmap='gray', interpolation='none')
    plt.title("Dígito: {}".format(y_train[i]))
    plt.xticks([])
    plt.yticks([])
fig
plt.show()
```

Dígito: 5



Dígito: 0



Dígito: 4



Dígito: 1



Dígito: 9



Dígito: 2



Dígito: 1



Dígito: 3

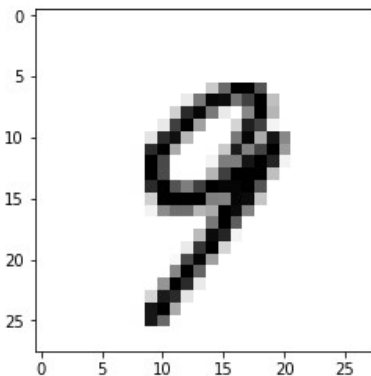


Dígito: 1



Visualizar un ejemplo concreto de la base de datos

```
# Visualizar un ejemplo de las 60.000 imágenes
digit = X_train[4345]
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```



Transformar y normalizar los datos a valores en el rango de 0 a 1. (Normalizar el histograma de la imagen)

```
# Antes de realizar el entrenamiento, preparar los datos transformando las imágenes
# iniciales con valores entre 0 y 255 (negro a blanco), a valores binarizados (0 a 1)
X_train = X_train.reshape((60000, 28 * 28))
X_train = X_train.astype('float32') / 255
X_test = X_test.reshape((10000, 28 * 28))
X_test = X_test.astype('float32') / 255
```

Transformar las etiquetas en categorías

```
# Preparar también las etiquetas en categorías:
from keras.utils import to_categorical
y_train_cat = to_categorical(y_train)
y_test_cat = to_categorical(y_test)
```

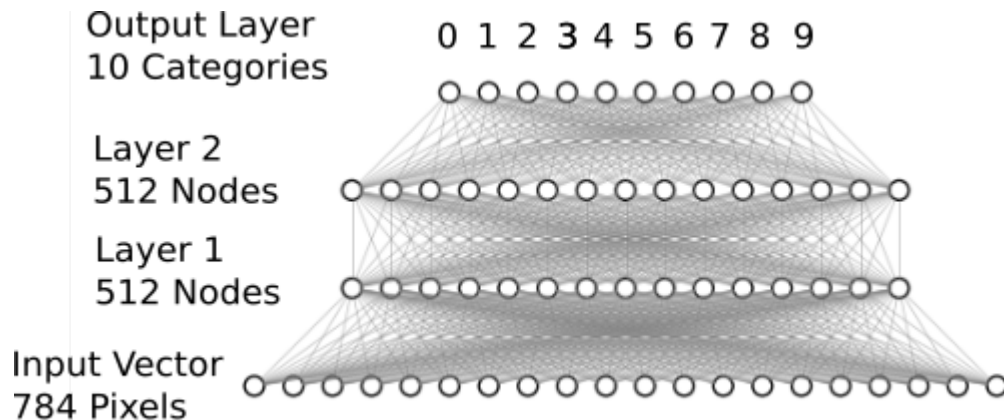
Crear la arquitectura de la red neuronal, formada por dos capas ocultas. La capa de salida representa las 10 categorías posibles (0 a 9).

```
# Crear la arquitectura de la red neuronal, formada por dos capas ocultas
# La capa de salida representa las 10 categorías posibles (0 a 9)
from keras import models
from keras import layers
network = models.Sequential()

# Capa de entrada y primera capa oculta
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))

# Capas ocultas
network.add(layers.Dense(512, activation='relu'))

# Capa de salida
network.add(layers.Dense(10, activation='softmax'))
```



Definir la función de pérdida, el optimizador y las métricas para monitorizar el entrenamiento y la prueba de validación.

```
# Definir la función de pérdida, el optimizador y las métricas para monitorizar el entrenamiento
y la prueba de validación
network.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])
# Más información en:
# Optimizer: https://keras.io/optimizers/
# Loss function: https://keras.io/losses/
# Metrics: https://keras.io/metrics/
```

Realizar el entrenamiento y posteriormente la predicción sobre el conjunto de prueba y se comprueba el ajuste o error del modelo. Guardar el resultado en una variable denominada 'history'

```
# Realizar el entrenamiento. Guardar el resultado en una variable denominada 'history'

history = network.fit(X_train, y_train_cat, epochs=5, batch_size=128, validation_data=(X_test,
y_test_cat))

Train on 60000 samples, validate on 10000 samples
Epoch 1/5
60000/60000 [=====] - 20116s 335ms/step - loss: 0.2235 - acc: 0.9297 -
val_loss: 0.0888 - val_acc: 0.9714
Epoch 2/5
60000/60000 [=====] - 20079s 335ms/step - loss: 0.0829 - acc: 0.9739 -
val_loss: 0.0897 - val_acc: 0.9738
Epoch 3/5
60000/60000 [=====] - 20020s 334ms/step - loss: 0.0554 - acc: 0.9829 -
val_loss: 0.0704 - val_acc: 0.9797
Epoch 4/5
60000/60000 [=====] - 20220s 337ms/step - loss: 0.0387 - acc: 0.9877 -
val_loss: 0.0767 - val_acc: 0.9804
Epoch 5/5
60000/60000 [=====] - 20211s 337ms/step - loss: 0.0307 - acc: 0.9907 -
val_loss: 0.0967 - val_acc: 0.9770
```

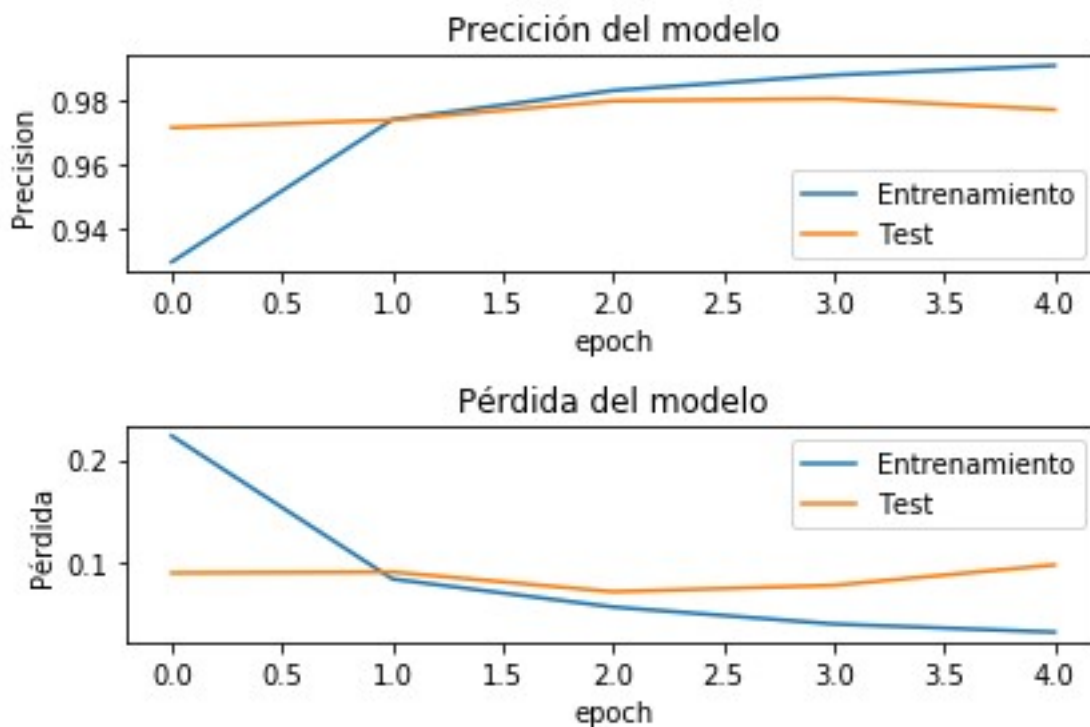
Visualizar las métricas

```
# Visualizar las métricas
fig = plt.figure()
plt.subplot(2,1,1)
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Precisión del modelo')
plt.ylabel('Precision')
plt.xlabel('epoch')
plt.legend(['Entrenamiento', 'Test'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Pérdida del modelo')
plt.ylabel('Pérdida')
plt.xlabel('epoch')
plt.legend(['Entrenamiento', 'Test'], loc='upper right')

plt.tight_layout()

plt.show()
```



Comprobar el error respecto del conjunto de prueba

```
# Comprobar el ajuste o error del modelo respecto del conjunto de prueba
test_loss, test_acc = network.evaluate(X_test, y_test_cat)
print('test_acc:', test_acc)
```

```
10000/10000 [=====] - 247s 25ms/step
test_acc: 0.977
```

Guardar el modelo (en formato JSON) y los pesos del modelo (en formato HDF5)

```
# Guardar el modelo en formato JSON
from keras.models import model_from_json
model_json = network.to_json()
with open("network.json", "w") as json_file:
    json_file.write(model_json)
```

```
# Guardar los pesos (weights) a formato HDF5
network.save_weights("network_weights.h5")
print("Guardado el modelo a disco")
```

```
Guardado el modelo a disco
Modelo cargado desde el disco
```

Cargar el modelo (en formato JSON) y los pesos del modelo (en formato HDF5)

```
# Leer JSON y crear el modelo
json_file = open("network.json", "r")
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
```

```
# Cargar los pesos (weights) en un nuevo modelo
loaded_model.load_weights("network_weights.h5")
print("Modelo cargado desde el disco")
```

Predicir sobre el conjunto de test

```
# Predicir sobre el conjunto de test
predicted_classes = loaded_model.predict_classes(X_test)
```

Comprobar predicciones correctas y falsas. Visualizar algunas correctas e incorrectas

```
# Comprobar que predicciones son correctas y cuales no
indices_correctos = np.nonzero(predicted_classes == y_test)[0]
indices_incorrectos = np.nonzero(predicted_classes != y_test)[0]
print()
print(len(indices_correctos), " clasificados correctamente")
print(len(indices_incorrectos), " clasificados incorrectamente")
```

```
9770 clasificados correctamente
230 clasificados incorrectamente
```

```
# Adaptar el tamaño de la figura para visualizar 18 subplots
plt.rcParams['figure.figsize'] = (7,14)
```

```
figure_evaluation = plt.figure()
```

```
# Visualizar 9 predicciones correctas
for i, correct in enumerate(indices_correctos[:9]):
```

```
plt.subplot(6,3,i+1)
plt.imshow(X_test[correct].reshape(28,28), cmap='gray', interpolation='none')
plt.title(
    "Pred: {}, Original: {}".format(predicted_classes[correct],
                                    y_test[correct]))

plt.xticks([])
plt.yticks([])

# Visualizar 9 predicciones incorrectas
for i, incorrect in enumerate(indices_incorrectos[:9]):
    plt.subplot(6,3,i+10)
    plt.imshow(X_test[incorrect].reshape(28,28), cmap='gray', interpolation='none')
    plt.title(
        "Pred: {}, Original: {}".format(predicted_classes[incorrect],
                                        y_test[incorrect]))

plt.xticks([])
plt.yticks([])

figure evaluation
```

