Practical Introduction to Deep Learning Basics

```
import tensorflow as tf
from tensorflow import keras

import numpy as np
import matplotlib.pyplot as plt

print(tf.__version__)
```

```
2.18.0
```

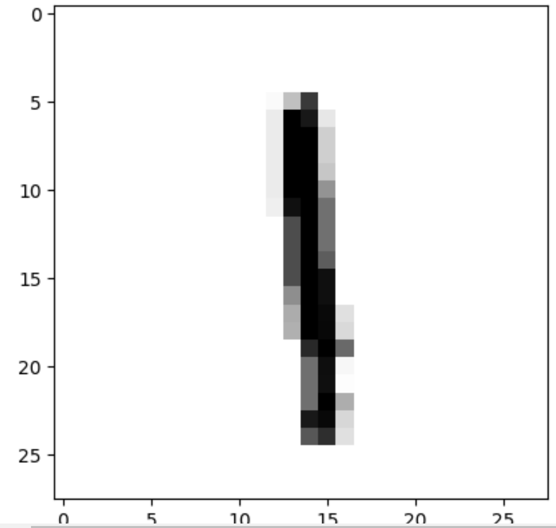
Load Data

```
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
#(x_train, y_train), (x_test, y_test) = mnist.load_data(path='/gpfs/projects/nct00/nct00002/basics-utils/mnist.npz')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
import matplotlib.pyplot as plt
plt.imshow(x_train[8], cmap=plt.cm.binary)
```

```
<matplotlib.image.AxesImage at 0x790825f76950>
```




```
print(y_train[8])
```

```
1
```

```
print(x_train.ndim)
```

```
3
```

```
print(x_train.shape)
```

```
(60000, 28, 28)
```

```
print(x_train.dtype)
```

```
uint8
```

Prepare data

```
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
```

```
x_train /= 255
x_test /= 255
```

```
x_train = x_train.reshape(60000, 784)
x_test = x_test.reshape(10000, 784)
```

```
print(x_train.shape)
print(x_test.shape)
```

```
(60000, 784)
(10000, 784)
```

```
from tensorflow.keras.utils import to_categorical
```

```
print(y_test[0])
```

```
7
```

```
print(y_train[0])
```

```
5
```

```
print(y_train.shape)
```

```
(60000,)
```

```
print(x_test.shape)
```

```
(10000, 784)
```

```
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)


print(y_test[0])
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

```
print(y_train[0])
```

```
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
print(y_train.shape)
```

```
(60000, 10)
```

```
print(y_test.shape)
```

```
(10000, 10)
```

Define Model

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense , Input

model = Sequential()
model.add(Input(shape=(784,)))  # Define the input shape here
model.add(Dense(128, activation='sigmoid')) # Antes tenia 10 neuronas
model.add(Dense(64, activation='relu'))  # Nueva capa con 64 neuronas
model.add(Dense(10, activation='softmax'))
```

```
model.summary()
```

```
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 128)	100,480
dense_2 (Dense)	(None, 64)	8,256
dense_3 (Dense)	(None, 10)	650

```
Total params: 109,386 (427.29 KB)
```

Compile model (configuration)

```
model.compile(loss="categorical_crossentropy",
              optimizer="sgd",
              metrics = ['accuracy'])
```

Training the model

```
model.fit(x_train, y_train, epochs=15)
```

```
Epoch 1/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.5297 - loss: 1.7614
Epoch 2/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8506 - loss: 0.5555
Epoch 3/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.8884 - loss: 0.3996
Epoch 4/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9010 - loss: 0.3472
Epoch 5/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9062 - loss: 0.3269
Epoch 6/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.9102 - loss: 0.3086
Epoch 7/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9168 - loss: 0.2938
Epoch 8/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9181 - loss: 0.2890
Epoch 9/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9203 - loss: 0.2738
Epoch 10/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9224 - loss: 0.2671
Epoch 11/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9262 - loss: 0.2609
Epoch 12/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 2ms/step - accuracy: 0.9270 - loss: 0.2537
Epoch 13/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.9294 - loss: 0.2405
Epoch 14/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9300 - loss: 0.2411
Epoch 15/15
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 2ms/step - accuracy: 0.9336 - loss: 0.2276
<keras.src.callbacks.history.History at 0x790821b438d0>
```

Evaluation the model

```
test_loss, test_acc = model.evaluate(x_test, y_test)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9249 - loss: 0.2555
```

```
print('Test accuracy:', test_acc)
```

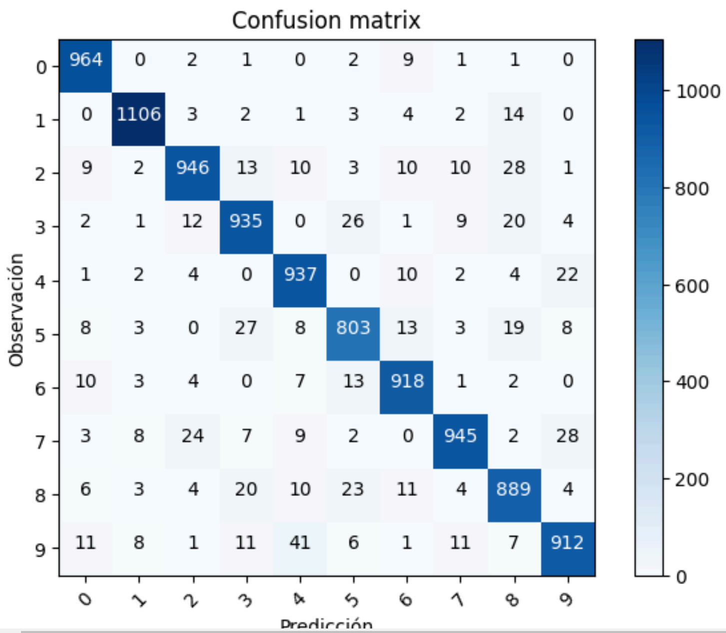
```
Test accuracy: 0.9355000257492065
```

```
# Look at confusion matrix
#Note, this code is taken straight from the SKLEARN website, an nice way of viewing confusion matrix.
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
```

```
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Observación')
    plt.xlabel('Predicción')
```

```
from collections import Counter
from sklearn.metrics import confusion_matrix
import itertools

# Predict the values from the validation dataset
Y_pred = model.predict(x_test)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred, axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(y_test, axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

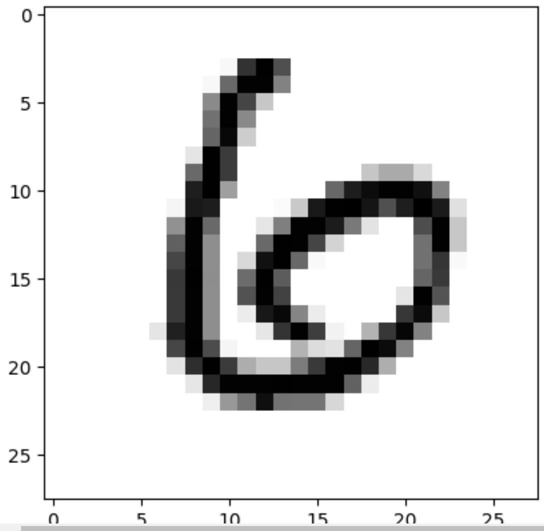
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



Use the model

```
x_test_old = x_test.reshape(10000, 28,28)
plt.imshow(x_test_old[11], cmap=plt.cm.binary)
```

<matplotlib.image.AxesImage at 0x79081052d9d0>



```
predictions = model.predict(x_test)
```

313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

```
np.argmax(predictions[11])
```

6

```
print(predictions[11])
```

```
[1.8964128e-03 5.7702418e-05 1.7207498e-02 4.6141120e-03 1.1903022e-03
 1.9227328e-03 9.1951627e-01 2.0993625e-06 5.3564698e-02 2.8224069e-05]
```

```
np.sum(predictions[11])
```

1.0

Convolutional Neural Network

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten

model = Sequential()

model.add(Conv2D(32, (5, 5), activation='relu', input_shape=(28, 28,
1)))
model.add(MaxPooling2D((2, 2)))
model.add(Conv2D(64, (5, 5), activation='relu'))
model.add(MaxPooling2D((2, 2)))

from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten

model.add(Flatten())
model.add(Dense(10, activation='softmax'))
model.summary()
from tensorflow.keras.utils import to_categorical

#mnist = tf.keras.datasets.mnist(train_images, train_labels),
(test_images, test_labels) = mnist.load_data()

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print (train_images.shape)
print (train_labels.shape)
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255
```

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model.compile(loss='categorical_crossentropy',
 optimizer='adam',
 metrics=['accuracy'])
model.fit(train_images, train_labels, batch_size=100, epochs=15,
verbose=1)
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 24, 24, 32)	832
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 32)	0
conv2d_3 (Conv2D)	(None, 8, 8, 64)	51,264
max_pooling2d_3 (MaxPooling2D)	(None, 4, 4, 64)	0
flatten_1 (Flatten)	(None, 1024)	0
dense_4 (Dense)	(None, 10)	10,250

```
 Total params: 62,346 (243.54 KB)
 Trainable params: 62,346 (243.54 KB)
 Non-trainable params: 0 (0.00 B)
(60000, 28, 28)
(60000,)
Epoch 1/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.8524 - loss: 0.4963
Epoch 2/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9817 - loss: 0.0634
Epoch 3/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9878 - loss: 0.0398
Epoch 4/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9913 - loss: 0.0294
Epoch 5/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9923 - loss: 0.0252
Epoch 6/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9942 - loss: 0.0198
Epoch 7/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9947 - loss: 0.0163
Epoch 8/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9955 - loss: 0.0139
Epoch 9/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9970 - loss: 0.0100
Epoch 10/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9969 - loss: 0.0097
Epoch 11/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9970 - loss: 0.0102
Epoch 12/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9980 - loss: 0.0073
Epoch 13/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9986 - loss: 0.0051
Epoch 14/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9984 - loss: 0.0050
Epoch 15/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9985 - loss: 0.0045
```

```
from tensorflow.keras.utils import to_categorical

#mnist = tf.keras.datasets.mnist(train_images, train_labels), (test_images, test_labels) = mnist.load_data(path='/gpfs/projects/nct00/nc

mnist = tf.keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

print (train_images.shape)
print (train_labels.shape)
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255

test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

print (train_images.shape)
print (train_labels.shape)
```

```
(60000, 28, 28)
(60000,)
(60000, 28, 28, 1)
(60000, 10)
```

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

```
model.fit(train_images, train_labels, batch_size=100, epochs=15, verbose=1)
```

```
Epoch 1/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9982 - loss: 0.0047
Epoch 2/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9992 - loss: 0.0031
Epoch 3/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9985 - loss: 0.0041
Epoch 4/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9990 - loss: 0.0025
Epoch 5/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9992 - loss: 0.0025
Epoch 6/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9988 - loss: 0.0035
Epoch 7/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9988 - loss: 0.0030
Epoch 8/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9992 - loss: 0.0021
Epoch 9/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9991 - loss: 0.0025
Epoch 10/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9992 - loss: 0.0021
Epoch 11/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9987 - loss: 0.0035
Epoch 12/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9991 - loss: 0.0024
Epoch 13/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9995 - loss: 0.0015
Epoch 14/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9994 - loss: 0.0022
Epoch 15/15
600/600 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9995 - loss: 0.0018
<keras.src.callbacks.history.History at 0x79081023c7d0>
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)

print('Test accuracy:', test_acc)
```

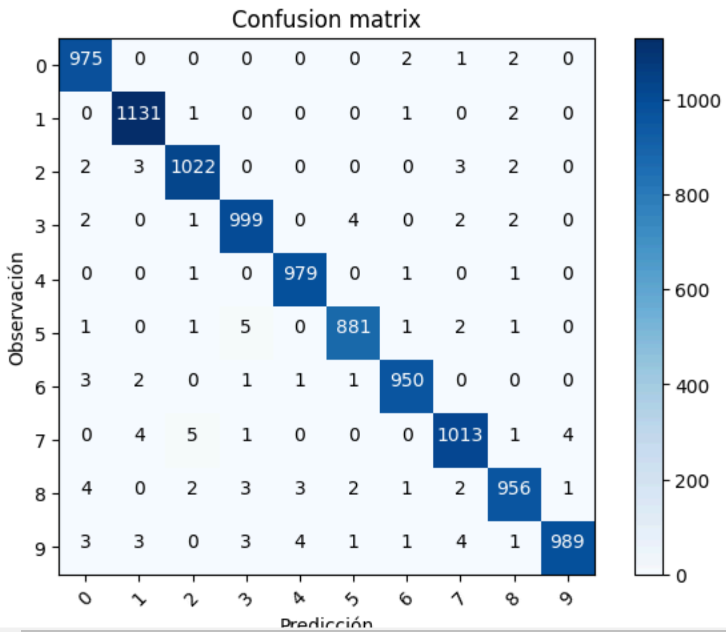
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9882 - loss: 0.0538
Test accuracy: 0.9907000064849854
```

Mostrar la matriz de confusión del modelo mejorado con ayuda de Gemini AI

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import itertools

# Predice las clases para los datos de prueba
Y_pred = model.predict(test_images)
Y_pred_classes = np.argmax(Y_pred, axis = 1)
Y_true = np.argmax(test_labels, axis = 1)
```

```python
# Calcula la matriz de confusión
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)

# Define la función para graficar la matriz de confusión
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Observación')
    plt.xlabel('Predicción')

# Grafica la matriz de confusión
plot_confusion_matrix(confusion_mtx, classes = range(10))
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
```



Nuevo Modelo Mejorado

Generar | randomly select 5 items from a list | Cerrar

```python
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.utils import to_categorical

# Definir el modelo CNN mejorado
model = keras.Sequential([
    # Primera capa convolucional
    layers.Conv2D(32, (3,3), activation='relu', input_shape=(28,28,1)),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2,2)),
```

```
    # Segunda capa convolucional
    layers.Conv2D(64, (3,3), activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D((2,2)),

    # Tercera capa convolucional (extra)
    layers.Conv2D(128, (3,3), activation='relu'),
    layers.BatchNormalization(),

    # Flatten para conectar con la capa densa
    layers.Flatten(),

    # Capa densa con más neuronas
    layers.Dense(256, activation='relu'),
    layers.Dropout(0.4),  # Reduce el sobreajuste
    layers.Dense(10, activation='softmax')  # Capa de salida
])

# Compilar el modelo con Adam
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Load MNIST dataset again to ensure we have the correct data shape for the CNN
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Reshape and normalize data for CNN input
x_train = x_train.reshape((60000, 28, 28, 1)).astype('float32') / 255
x_test = x_test.reshape((10000, 28, 28, 1)).astype('float32') / 255


# Convert labels to categorical
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

# Entrenar el modelo
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_test, y_test))
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 14s 5ms/step - accuracy: 0.9321 - loss: 0.2265 - val_accuracy: 0.9818 - val_loss: 0.0520
Epoch 2/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 16s 4ms/step - accuracy: 0.9835 - loss: 0.0553 - val_accuracy: 0.9829 - val_loss: 0.0594
Epoch 3/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 4ms/step - accuracy: 0.9875 - loss: 0.0442 - val_accuracy: 0.9881 - val_loss: 0.0414
Epoch 4/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9903 - loss: 0.0330 - val_accuracy: 0.9911 - val_loss: 0.0320
Epoch 5/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9913 - loss: 0.0306 - val_accuracy: 0.9906 - val_loss: 0.0340
Epoch 6/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9922 - loss: 0.0282 - val_accuracy: 0.9860 - val_loss: 0.0597
Epoch 7/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9932 - loss: 0.0223 - val_accuracy: 0.9919 - val_loss: 0.0345
Epoch 8/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9935 - loss: 0.0221 - val_accuracy: 0.9892 - val_loss: 0.0446
Epoch 9/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9951 - loss: 0.0159 - val_accuracy: 0.9923 - val_loss: 0.0397
Epoch 10/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 11s 5ms/step - accuracy: 0.9941 - loss: 0.0214 - val_accuracy: 0.9912 - val_loss: 0.0442
Epoch 11/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9960 - loss: 0.0142 - val_accuracy: 0.9910 - val_loss: 0.0377
Epoch 12/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9955 - loss: 0.0165 - val_accuracy: 0.9917 - val_loss: 0.0396
Epoch 13/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9956 - loss: 0.0160 - val_accuracy: 0.9915 - val_loss: 0.0425
Epoch 14/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9962 - loss: 0.0120 - val_accuracy: 0.9934 - val_loss: 0.0344
Epoch 15/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9974 - loss: 0.0094 - val_accuracy: 0.9914 - val_loss: 0.0398
Epoch 16/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9970 - loss: 0.0104 - val_accuracy: 0.9924 - val_loss: 0.0530
Epoch 17/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9972 - loss: 0.0089 - val_accuracy: 0.9915 - val_loss: 0.0455
Epoch 18/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9969 - loss: 0.0124 - val_accuracy: 0.9926 - val_loss: 0.0357
Epoch 19/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9971 - loss: 0.0101 - val_accuracy: 0.9925 - val_loss: 0.0387
Epoch 20/20
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9982 - loss: 0.0066 - val_accuracy: 0.9919 - val_loss: 0.0614
```

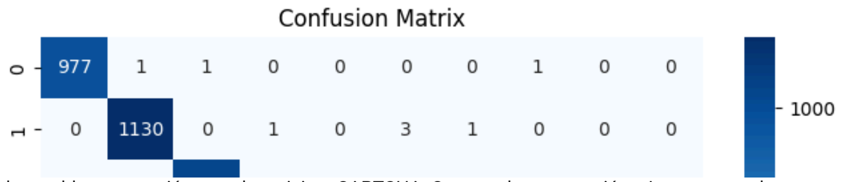
```
test_loss, test_acc = model.evaluate(x_test, y_test)
print(f"Test Accuracy: {test_acc:.4f}")
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9887 - loss: 0.0797
```

Mostrar matriz de confusión

```
import seaborn as sns
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

# Obtener predicciones
y_pred = model.predict(x_test)
y_pred_classes = y_pred.argmax(axis=1)

# Convert y_test to class labels
y_true = np.argmax(y_test, axis=1)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_true, y_pred_classes)

# Graficar la matriz de confusión
plt.figure(figsize=(8,6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```



No se ha podido establecer conexión con el servicio reCAPTCHA. Comprueba tu conexión a Internet y vuelve a cargar la página para ver otro reCAPTCHA.