```
# === Cell 1: Imports ===
import numpy as np                       # import numpy for numerical array operations
import tensorflow as tf                  # import TensorFlow (includes Keras API) for deep learning
from tensorflow.keras.datasets import mnist   # import MNIST dataset loader (handwritten digits)
from tensorflow.keras.models import Sequential # import Sequential model class (stack layers linearly)
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, Dropout
# import layer classes used: Dense (fully-connected), Conv2D (conv layer), MaxPooling2D (pooling), Flatten, Dropout
import matplotlib.pyplot as plt          # import matplotlib for plotting images and graphs
```

```
# === Cell 2: Load dataset ===
print("[INFO] accessing MNIST...")        # print a small info message to indicate dataset loading
(x_train, y_train), (x_test, y_test) = mnist.load_data()  # load MNIST into training and test tuples
# after this: x_train shape = (60000, 28, 28), y_train shape = (60000,)
#                x_test shape  = (10000, 28, 28), y_test shape  = (10000,)
```

```
[INFO] accessing MNIST...
```

```
# === Cell 3: Preprocess input data ===
# reshape training images to (num_samples, height, width, channels) for Keras Conv2D
x_train = x_train.reshape((x_train.shape[0], 28, 28, 1))  # add channel dimension (1 for grayscale)
x_train = x_train.astype('float32')                       # convert integer pixels to float32 for NN computations
x_train = x_train / 255.0                                 # normalize pixel values from [0,255] to [0.0,1.0]

# reshape and normalize test images the same way
x_test = x_test.reshape((x_test.shape[0], 28, 28, 1))     # add channel dim to test images
x_test = x_test.astype('float32')                         # cast to float32
x_test = x_test / 255.0                                   # scale to [0.0,1.0]
```

```
# === Cell 4: Build the model ===
model = Sequential()                                      # create an empty Sequential model container

# add a 2D convolutional layer with 28 filters of size 3x3
# input_shape=(28,28,1) tells the model the shape of the input images (height,width,channels)
model.add(Conv2D(28, kernel_size=(3, 3), input_shape=(28, 28, 1)))
# add a max pooling layer to downsample the feature maps by a factor of 2 in each spatial dim
model.add(MaxPooling2D(pool_size=(2, 2)))
# flatten the 3D feature maps into a 1D vector so it can be fed into Dense layers
model.add(Flatten())

# add a fully connected layer with 200 neurons and ReLU activation for non-linearity
model.add(Dense(200, activation="relu"))
# add dropout to randomly turn off 30% of neurons during training to help prevent overfitting
model.add(Dropout(0.3))
# final output layer with 10 neurons (one per digit class) and softmax to produce probabilities
model.add(Dense(10, activation="softmax"))

# print the model architecture summary (layers, output shapes, and number of parameters)
model.summary()
```

```
Model: "sequential_2"
┏━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━┓
┃ Layer (type)                    ┃ Output Shape           ┃       Param # ┃
┡━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━┩
│ conv2d_1 (Conv2D)               │ (None, 26, 26, 28)     │           280 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 13, 13, 28)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten_1 (Flatten)             │ (None, 4732)           │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_5 (Dense)                 │ (None, 200)            │       946,600 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dropout_1 (Dropout)             │ (None, 200)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_6 (Dense)                 │ (None, 10)             │         2,010 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 948,890 (3.62 MB)
 Trainable params: 948,890 (3.62 MB)
 Non-trainable params: 0 (0.00 B)
```

```
# === Cell 5: Compile the model ===
# compile prepares the model for training by setting the optimizer, loss function and metrics
# optimizer='adam' --> adaptive optimizer well-suited for many problems
# loss='sparse_categorical_crossentropy' --> use when labels are integer-coded (0..9)
# metrics=['accuracy'] --> track accuracy during training and evaluation
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
# === Cell 6: Train the model ===
# fit() trains the model on the training data
# x_train, y_train --> training images and integer labels
# epochs=2 --> the model will see the entire training set twice
# (default batch_size is 32 unless you specify another value)
model.fit(x_train, y_train, epochs=2)
```

```
Epoch 1/2
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 47s 24ms/step - accuracy: 0.8949 - loss: 0.3440
Epoch 2/2
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 82s 25ms/step - accuracy: 0.9747 - loss: 0.0810
<keras.src.callbacks.history.History at 0x7ef319d08dd0>
```

```
# === Cell 7: Evaluate on test set ===
# evaluate returns [loss, metric_values...] according to what was specified in compile()
test_loss, test_accuracy = model.evaluate(x_test, y_test)  # compute loss and accuracy on unseen test data
# print the test accuracy as a percentage with two decimal places
print(f'Test accuracy: {test_accuracy*100:.2f}%')
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9741 - loss: 0.0708
Test accuracy: 97.99%
```

```
# === Cell 8: Visualize a test image ===
image = x_test[9]                       # select the 10th test image (index 9)
# imshow expects shape (height, width) or (height, width, channels); squeeze channel to show grayscale
plt.imshow(image.reshape(28, 28), cmap='Greys')  # show the image in grayscale color map 'Greys'
plt.title(f'Label: {y_test[9]}')        # (optional) title showing the true label of the image
plt.axis('off')                         # hide axis ticks for a cleaner display
plt.show()                              # render the plot to the output
```

Start coding or generate with AI.