2. Number Recognition with Keras

May 4, 2024

> "You can think of solving MNIST as the"Hello world" of deep learning." - François Chollet

The MNIST dataset consists of a collection of 70,000 images of handwritten numerical digits. Each image consists of a 28x28 pixel grayscale image.

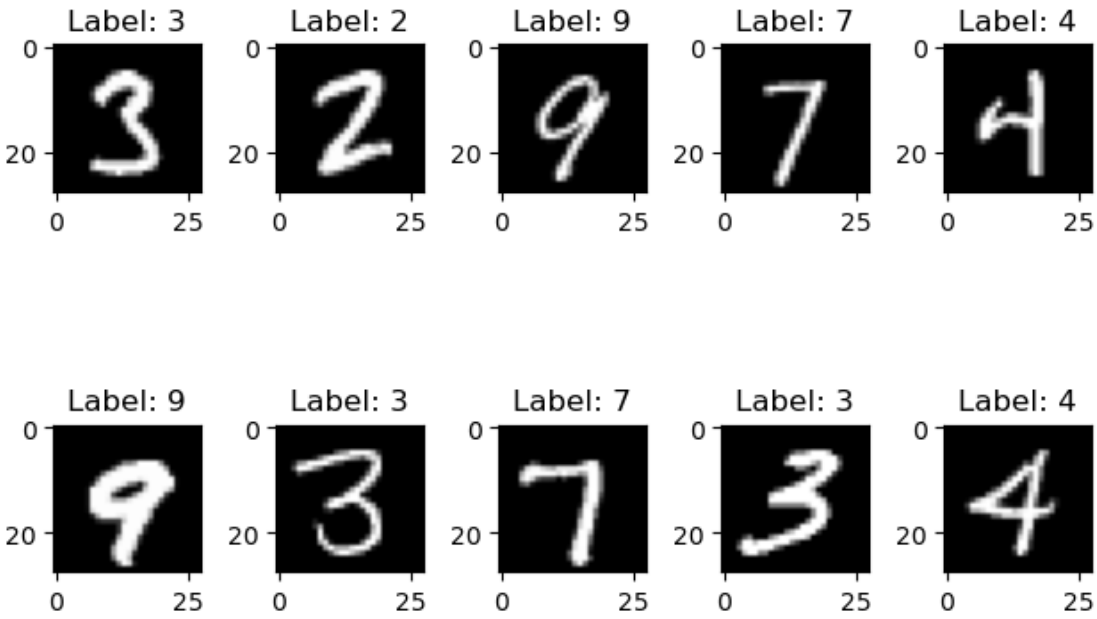
THe cell below displays 10 randomly selected images out of the training dataset.

[2]:
```
import matplotlib.pyplot as plt
import numpy as np

# Load the MNIST dataset
from tensorflow.keras.datasets import mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Select a random sample of images
num_samples = 10
random_indices = np.random.choice(len(train_images), num_samples)

# Plot the random samples
for i, idx in enumerate(random_indices):
  plt.subplot(2, 5, i+1)
  plt.imshow(train_images[idx], cmap='gray')
  plt.title(f"Label: {train_labels[idx]}")
plt.tight_layout()
plt.show()
```



We can verify as follows that each image is simply a 2d numy array, with grayscale intensity in the range 0 to 255.

[56]:
```
example_image = train_images[random_indices[0]]
print(
    type(example_image),
    example_image,
    f"Number of Train images: {len(train_images)}",
    f"Number of Test images: {len(test_images)}",
    sep='\n'
)
```

```
<class 'numpy.ndarray'>
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  18  43 123 201 227 148 131  29   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  84 216 253 252 252 252 252 253 231  95   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  73 246 252 253 252 252 247 249 245 252 242 130
    0   0   0   0   0   0   0   0   0   0]
```

```
 [  0   0   0   0   0   0 127 252 252 190 189 136  63  70  49 211 252 231
   28   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0  21  86  42   0   0   0   0   0   0 146 252 252
  129   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 103 253 253
  147   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0 190 252 252
   68   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 101 247 252 194
    4   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0  32 134 169 253 252 252 231
   81   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  36 223 252 252 253 252 252 252
  252  45   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0 129 253 253 253 247 211 211 239
  253 237  55   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0  21 159 203 168  53   0   0  69
  239 253 196   7   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   9   0   0   0   0   0
  106 253 252  74   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0 253 252 126   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0 253 252 126   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  11  87  80   0   0   0   0   0   0   0  38
  192 255 253 109   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0 143 252 211   0   0   0   0   0   0  87 206
  252 253 189  14   0   0   0   0   0   0]
 [  0   0   0   0   0   0   9 204 252 250 118  22  22  22 128 206 251 252
  252 170  32   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0  83 242 253 252 252 252 252 253 252 252 210
  145   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0  35 165 252 252 252 252 253 252 146  14
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0   0   0]]
Number of Train images: 60000
Number of Test images: 10000
```

1 Defining the Network Architecture

[57]:
```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential([
  layers.Dense(512, activation="relu"),
  layers.Dense(10, activation="softmax")
])
```

1.1 The Compilation Step

[58]:
```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])
```

1.2 Preparing the Training Data

The reshape we are doing can be represented as: 60000, 28, 28 - > 60000, 28*28

[59]:
```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype("float32") / 255
```

1.3 Fitting the Model

[60]:
```
model.fit(train_images, train_labels, epochs=5, batch_size=128)
```

```
Epoch 1/5
469/469 [==============================] - 6s 12ms/step - loss: 0.2645 -
accuracy: 0.9235
Epoch 2/5
469/469 [==============================] - 7s 14ms/step - loss: 0.1053 -
accuracy: 0.9685
Epoch 3/5
469/469 [==============================] - 7s 14ms/step - loss: 0.0701 -
accuracy: 0.9791
Epoch 4/5
469/469 [==============================] - 7s 14ms/step - loss: 0.0503 -
accuracy: 0.9849
Epoch 5/5
469/469 [==============================] - 7s 15ms/step - loss: 0.0376 -
accuracy: 0.9890
```

[60]:
```
<keras.callbacks.History at 0x19bde2710>
```

[61]:
```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_6 (Dense)             (None, 512)               401920

 dense_7 (Dense)             (None, 10)                5130

=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
_________________________________________________________________
```

1.4 Making Predictions on the Test Set

Each number of index `i` in that array corresponds to the probability that digit image `test_digits[0]` belongs to class `i`.

[62]:
```
test_digits = test_images[0:10]
predictions = model.predict(test_digits)
predictions[0]
```

```
1/1 [==============================] - 0s 103ms/step
```

[62]:
```
array([1.9355136e-07, 7.8654594e-09, 6.5878298e-06, 1.2391029e-04,
       4.3995693e-12, 8.3856868e-08, 1.7252054e-11, 9.9986339e-01,
       4.2469955e-07, 5.3094614e-06], dtype=float32)
```

1.5 Evaluating the Model on the Test Set

[63]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print (f"test_acc: {test_acc}")
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.0581 -
accuracy: 0.9808
test_acc: 0.9807999730110168
```

The test-set accuracy turns out to be 98.0%—that's quite a bit lower than the training-set accuracy (98.9%). This gap between training accuracy and test accuracy is an example of overfitting: the fact that machine learning models tend to perform worse on new data than on their training data. Overfitting is a central topic in chapter 3.

Let's plot some images from the test set along with the predictions by the model:

[65]:
```
# Select a random sample of images
num_samples = 10
```

```
random_indices = np.random.choice(len(test_images), num_samples)

# Plot the random samples with their predicted labels
for i, idx in enumerate(random_indices):
    test_prediction = model.predict(test_images[idx:idx+1])
    predicted_label = np.argmax(test_prediction, axis=1)
    plt.subplot(2, 5, i+1)
    plt.imshow(test_images[idx].reshape(28, 28), cmap='gray')
    plt.title(f"True: {test_labels[idx]}\nPred: {predicted_label[0]}")
plt.tight_layout()
plt.show()
```

```
1/1 [==============================] - 0s 57ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 91ms/step
1/1 [==============================] - 0s 37ms/step
1/1 [==============================] - 0s 35ms/step
1/1 [==============================] - 0s 28ms/step
```