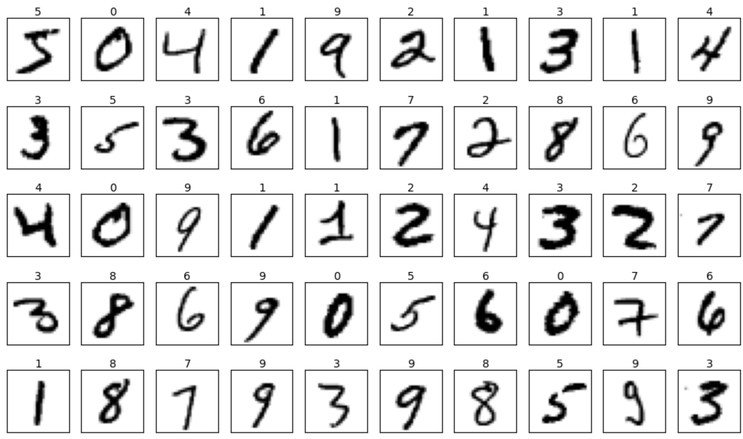
Digit recognition using convolutional neural networks

The MNIST dataset, officially known as "The MNIST Database of Handwritten Digits," is a famous dataset used to train machine-learning models (and neural networks) to recognize handwritten digits. Each digit in the dataset consists of a 28x28 array of numbers representing pixel values from 0 to 255. In this example, we will use Keras to build and train a convolutional neural network (CNN) on the MNIST dataset. The dataset is included in Keras as a sample dataset, so we'll begin by loading it and examining its content and structure.

In [2]:
```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
print('train_images: ' + str(train_images.shape))
print('train_labels: ' + str(train_labels.shape))
print('test_images: ' + str(test_images.shape))
print('test_labels: ' + str(test_labels.shape))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 8s 1us/step
train_images: (60000, 28, 28)
train_labels: (60000,)
test_images: (10000, 28, 28)
test_labels: (10000,)
```

In [3]:
```
%matplotlib inline
import matplotlib.pyplot as plt

fig, axes = plt.subplots(5, 10, figsize=(12, 7), subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(train_images[i], cmap=plt.cm.gray_r)
    ax.text(0.45, 1.05, str(train_labels[i]), transform=ax.transAxes)
```



In [4]:
```
from tensorflow.keras.utils import to_categorical

x_train = train_images.reshape(60000, 28, 28, 1) / 255
x_test = test_images.reshape(10000, 28, 28, 1) / 255

y_train = to_categorical(train_labels)
y_test = to_categorical(test_labels)
```

In [5]:
```
#Create a convolutional neural network with a softmax output layer for classification.
```

In [6]:
```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Dense, Flatten

model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(2, 2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(2, 2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 128)               204928

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 225,034
Trainable params: 225,034
Non-trainable params: 0
_________________________________________________________________
```

In [7]:
```
hist = model.fit(x_train, y_train, validation_data=(x_test, y_test), epochs=10, batch_size=50)
```

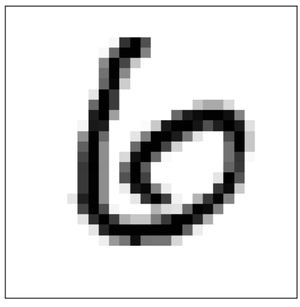
```
Epoch 1/10
1200/1200 [==============================] - 76s 61ms/step - loss: 0.1501 - accuracy: 0.9554 - val_loss: 0.0519 - val_accuracy: 0.9831
Epoch 2/10
1200/1200 [==============================] - 70s 59ms/step - loss: 0.0484 - accuracy: 0.9850 - val_loss: 0.0409 - val_accuracy: 0.9874
Epoch 3/10
1200/1200 [==============================] - 71s 59ms/step - loss: 0.0331 - accuracy: 0.9898 - val_loss: 0.0303 - val_accuracy: 0.9901
Epoch 4/10
1200/1200 [==============================] - 74s 62ms/step - loss: 0.0236 - accuracy: 0.9924 - val_loss: 0.0335 - val_accuracy: 0.9894
Epoch 5/10
1200/1200 [==============================] - 65s 54ms/step - loss: 0.0183 - accuracy: 0.9940 - val_loss: 0.0285 - val_accuracy: 0.9906
Epoch 6/10
1200/1200 [==============================] - 65s 54ms/step - loss: 0.0141 - accuracy: 0.9955 - val_loss: 0.0276 - val_accuracy: 0.9914
Epoch 7/10
1200/1200 [==============================] - 68s 56ms/step - loss: 0.0098 - accuracy: 0.9966 - val_loss: 0.0369 - val_accuracy: 0.9894
Epoch 8/10
1200/1200 [==============================] - 72s 60ms/step - loss: 0.0082 - accuracy: 0.9972 - val_loss: 0.0357 - val_accuracy: 0.9913
Epoch 9/10
1200/1200 [==============================] - 73s 61ms/step - loss: 0.0079 - accuracy: 0.9973 - val_loss: 0.0396 - val_accuracy: 0.9903
Epoch 10/10
1200/1200 [==============================] - 76s 64ms/step - loss: 0.0060 - accuracy: 0.9978 - val_loss: 0.0497 - val_accuracy: 0.9888
```

In [8]:
```
scores = model.evaluate(x_test, y_test, verbose=0)
print(f'Accuracy: {scores[1]:.1%}')
```

```
Accuracy: 98.9%
```

In [9]:
```
test_image = test_images[11]
plt.tick_params(axis='both', which='both', bottom=False, top=False, left=False, right=False, labelbottom=False, labelleft=False)
plt.imshow(test_image, cmap=plt.cm.gray_r)
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x1b601eabbe0>
```



In [10]:
```
x = test_image.reshape(1, 28, 28, 1) / 255
model.predict(x)
```

```
1/1 [==============================] - 0s 318ms/step
```

Out[10]:
```
array([[3.3424284e-11, 3.5541718e-18, 5.6632106e-20, 2.0246494e-19,
        4.1997564e-18, 4.9361795e-12, 1.0000000e+00, 5.3914107e-22,
        1.9858419e-11, 6.1234781e-22]], dtype=float32)
```

In [11]:
```
import numpy as np

predicted_class = np.argmax(model.predict(x), axis=-1)[0]
print('Looks like a ' + str(predicted_class) + '!')
```

```
1/1 [==============================] - 0s 43ms/step
Looks like a 6!
```

In [ ]:

In [ ]: