CNN Image Classifier

June 25, 2025

1 CNN Image Classifier

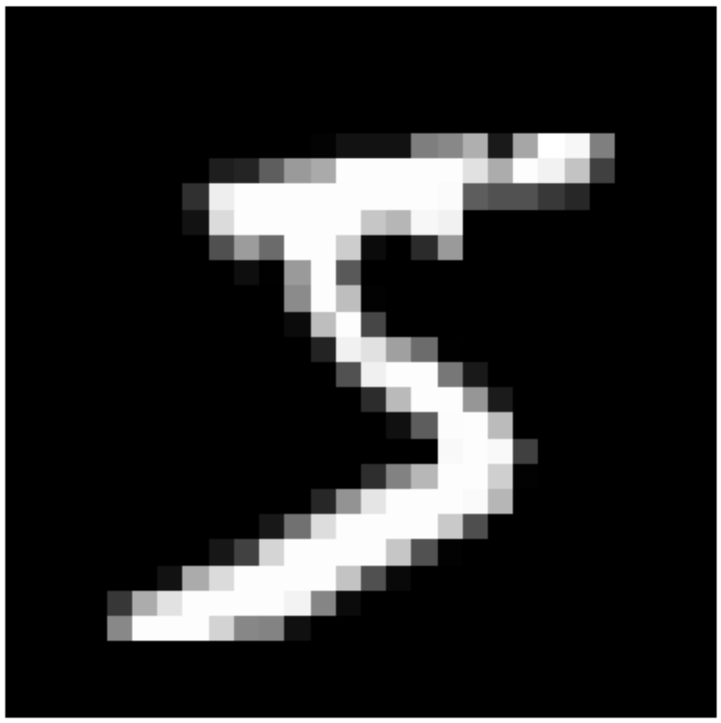
[1]:
```
from matplotlib import pyplot as plt
import tensorflow as tf
from keras.datasets import mnist
from sklearn.metrics import confusion_matrix
import numpy as np
from sklearn import metrics
```

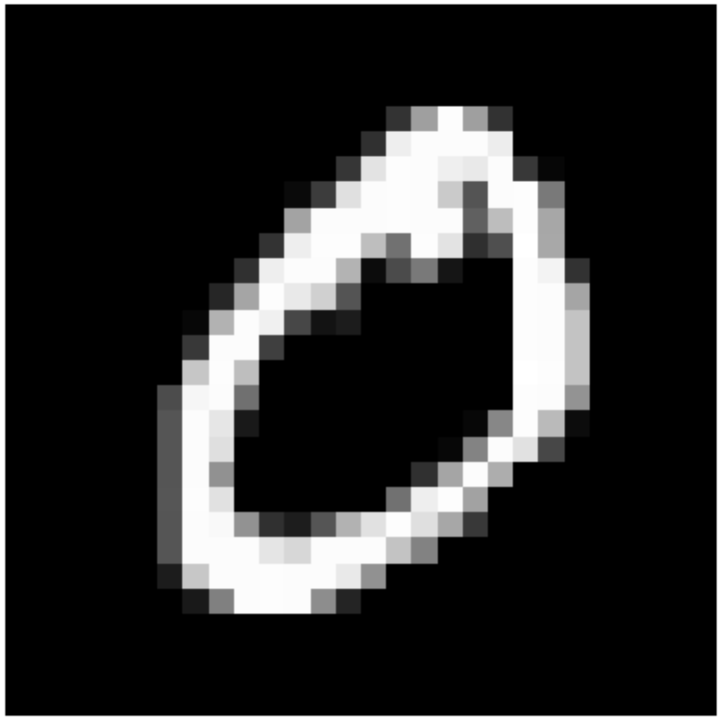
1.1 Load the MNIST data set.

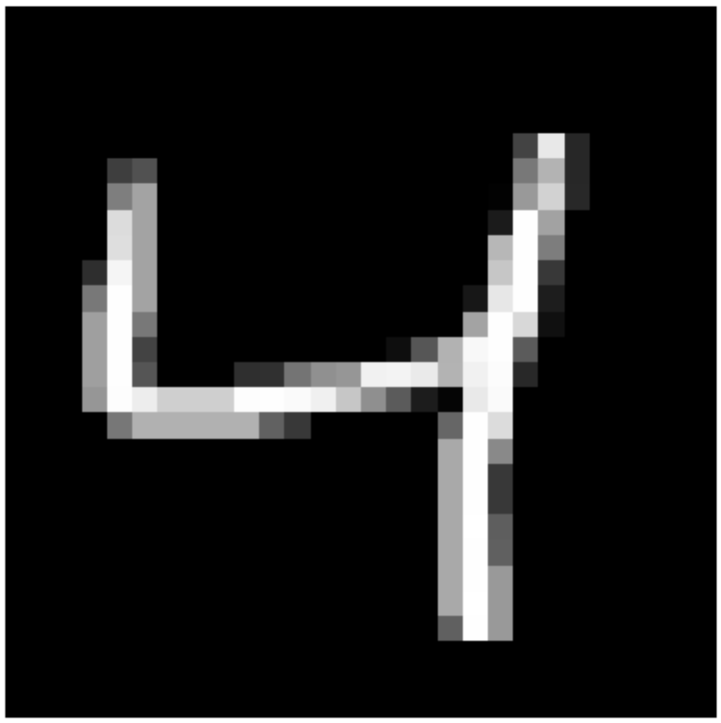
[2]:
```
# Set a seed for the split of the data.
np.random.seed(1)

# Load in the data while splitting it into testing and training sets.
(train_X, train_y), (test_X, test_y) = mnist.load_data()
```

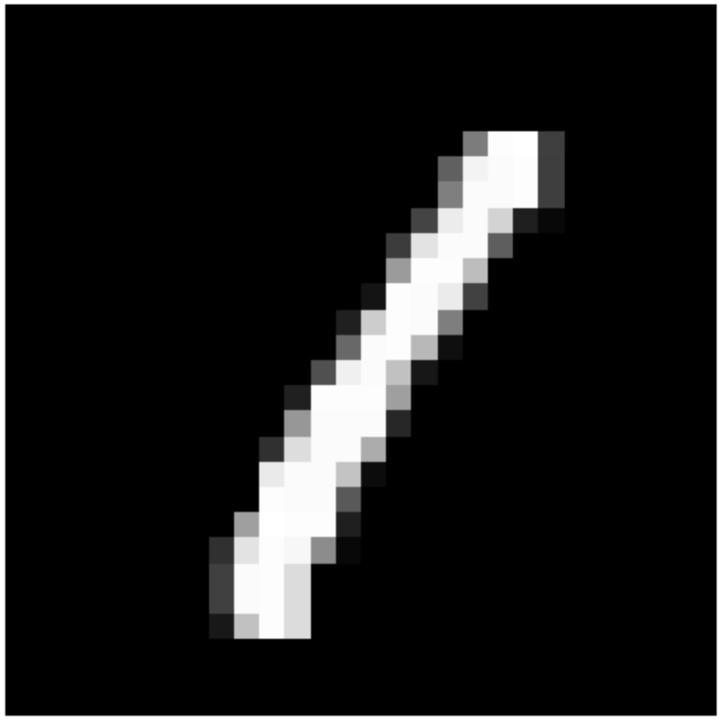
1.2 Display the first five images in the training data set. Compare these to the first five training labels.

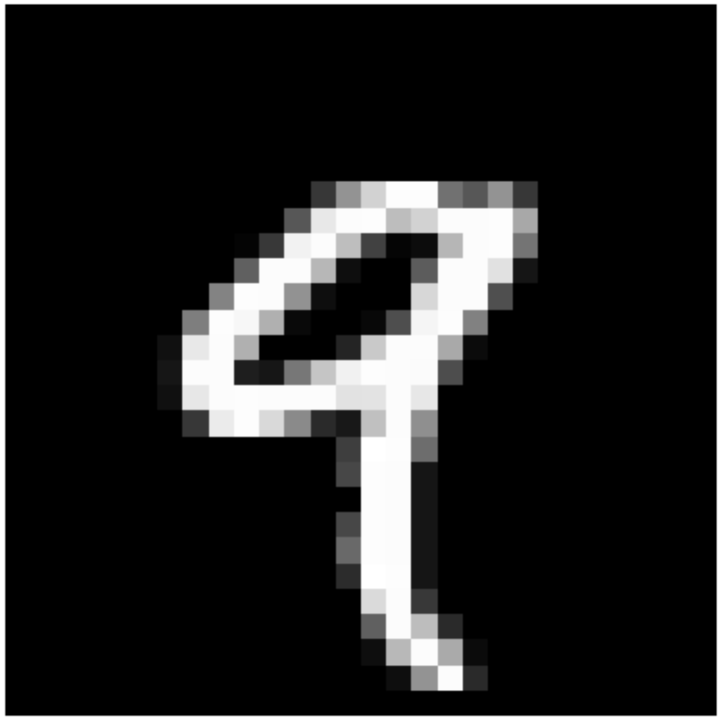
[3]:
```
# Use a for loop to plot the first 5 images.
for i in range(5):
    plt.imshow(train_X[i], cmap='gray'), plt.axis('off')
    plt.show()
```











[4]:
```
# Print the first 5 training labels.
print(train_y[0:5])
```

```
[5 0 4 1 9]
```

The images match the value assigned to them in the training labels. However, I thought the image for the 1 was a forward slash.

[5]:
```
# Normalize the training and testing image sets.

train_X = tf.keras.utils.normalize(train_X, axis=1)
test_X = tf.keras.utils.normalize(test_X, axis=1)
```

[6]:
```
# Set up the network with the Sequential function.
network = tf.keras.models.Sequential()

# Flatten the image input layer.
network.add(tf.keras.layers.Flatten())

# Creat a dense layer with 126 units and relu as the activation function.
network.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))

# Make a second dense layer.
network.add(tf.keras.layers.Dense(128, activation=tf.nn.relu))
```

```
# Make and output layer with the expected number of out puts and soft max as a probability ditrsibution.
network.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))
```

[7]:
```
# Complile the network with an optimizer, loss metric and an accuracy metric.
network.compile(optimizer = 'adam', loss = 'sparse_categorical_crossentropy',metrics = ['accuracy'])
```

[8]:
```
# Run the network with the training data through 3 instances.
network.fit(train_X, train_y, epochs = 3)
```

```
Epoch 1/3
1875/1875             2s 855us/step -
accuracy: 0.8667 - loss: 0.4731
Epoch 2/3
1875/1875             2s 873us/step -
accuracy: 0.9655 - loss: 0.1124
Epoch 3/3
1875/1875             2s 850us/step -
accuracy: 0.9769 - loss: 0.0751
```

[8]:
```
<keras.src.callbacks.history.History at 0x322374650>
```

[9]:
```
# Obtain the accuracy of the model
loss, accuracy = network.evaluate(test_X, test_y)
print(f'The accuracy of the model is {accuracy * 100}%')
```

```
313/313             0s 279us/step -
accuracy: 0.9670 - loss: 0.1070
The accuracy of the model is 97.00000286102295%
```

[10]:
```
# Obtain the predictions from the model on the test data.
prediction = network.predict(test_X)

# Create a confusion matrix from the predictions and the test labels.
confusion = confusion_matrix(test_y, np.argmax(prediction,axis=1))
```

```
313/313             0s 298us/step
```

[11]:
```
print(confusion)
```

```
[[ 969    0    0    2    1    3    2    1    2    0]
 [   0 1111    3    1    0    0    3    0   17    0]
 [   5    2  992    6    2    0    1   12   12    0]
 [   0    0    3  977    0    8    1   11    8    2]
 [   0    0    4    0  953    0    5    6    4   10]
 [   2    0    1   15    0  860    6    1    6    1]
 [   6    2    1    1    3    5  936    0    4    0]
```

```
[   3    2   10    0    1    0    0 1005    5    2]
[   1    0    1    4    5    6    1    2  951    3]
[   7    3    0    5   25    5    0    4   14  946]]
```

1.3 Summarize your results.

The accuracy of the model/ network was 97% after the third itteration. The model seemed to perform well based on the confusion matrix. All the large values on the diagonal are correct classifications of an instance in that class. There were a few instances that were classified incorreclty but they appear to be minor. Overall, the model will perform well on identifying unseen data based on its accuracy.

[ ]: