DSC550-T301 11.2 Exercise by Stephanie Benavidez

Building a CNN Image Classifier

Step 1. Load the MNIST data set.

Step 2. Display the first five images in the training data set (see section 8.1 in the Machine Learning with Python Cookbook). Compare these to the first five training labels.

Step 3. Build and train a Keras CNN classifier on the MNIST training set.

Step 4. Report the test accuracy of your model.

Step 5. Display a confusion matrix on the test set classifications.

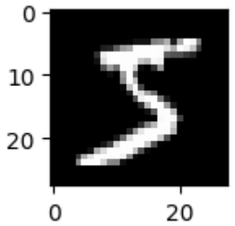
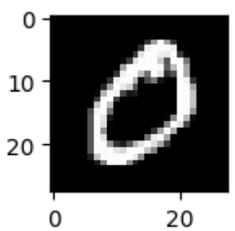
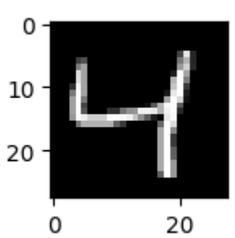
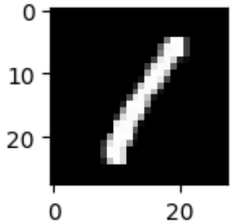
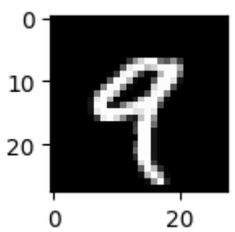
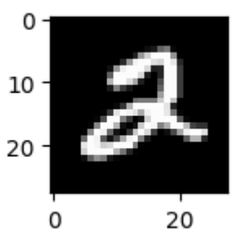
Step 6. Summarize your results.

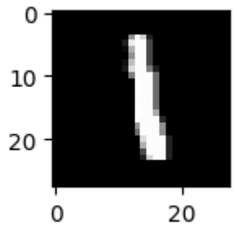
In [1]:
```
# Import libraries
import numpy as np
from numpy import mean
from numpy import std
from matplotlib import pyplot as plt
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPool2D, Flatten
from tensorflow.keras.utils import to_categorical
```

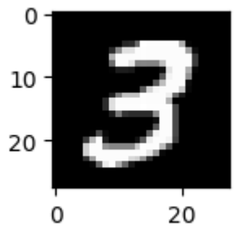
In [2]:
```
# Load dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

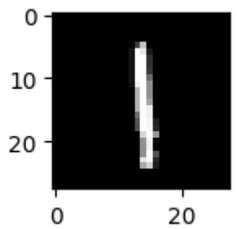
In [3]:
```
# Reshape and normalize data from dataset
x_train = x_train.reshape((-1, 28, 28, 1))
x_test = x_test.reshape((-1, 28, 28, 1))
x_train = x_train.astype('float32')/255
x_test = x_test.astype('float32')/255
```

In [4]:
```
# Plot first five images
for i in range(9):
# define subplot
    plt.subplot(330 + 1 + i)
# plot raw pixel data
    plt.imshow(x_train[i], cmap=plt.get_cmap('gray'))
# show the figure
    plt.show()
```







In [5]:
```
# One-hot encode target
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)
```

In [6]:
```
#Build model and compile data accuracy
model = Sequential()
model.add(Conv2D(6, kernel_size=(5, 5), activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPool2D((2, 2)))
model.add(Flatten())
model.add(Dense(84, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

In [7]:
```
# Train model and fit
model.fit(x_train, y_train, epochs=10, batch_size=128)
```

```
Epoch 1/10
469/469 [==============================] - 5s 11ms/step - loss: 0.3218 - accuracy: 0.9100
Epoch 2/10
469/469 [==============================] - 5s 11ms/step - loss: 0.1057 - accuracy: 0.9691
Epoch 3/10
469/469 [==============================] - 5s 12ms/step - loss: 0.0740 - accuracy: 0.9781
Epoch 4/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0582 - accuracy: 0.9827
Epoch 5/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0491 - accuracy: 0.9850
Epoch 6/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0411 - accuracy: 0.9876
Epoch 7/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0355 - accuracy: 0.9892
Epoch 8/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0303 - accuracy: 0.9904
Epoch 9/10
469/469 [==============================] - 6s 12ms/step - loss: 0.0262 - accuracy: 0.9917
Epoch 10/10
469/469 [==============================] - 5s 11ms/step - loss: 0.0234 - accuracy: 0.9929
```

Out[7]: <keras.callbacks.History at 0x1fa34527790>

In [8]:
```
# Evaluate model
score = model.evaluate(x_test, y_test)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0399 - accuracy: 0.9868
Test loss: 0.039881251752376556
Test accuracy: 0.9868000149726868
```

In [9]:
```
# summarize performance
def summarize_performance(scores):
    # Print Summary
    print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))
    # box and whisker plots of results
    plt.boxplot(scores)
    plt.show()
```

In [ ]: