MNIST

June 18, 2020

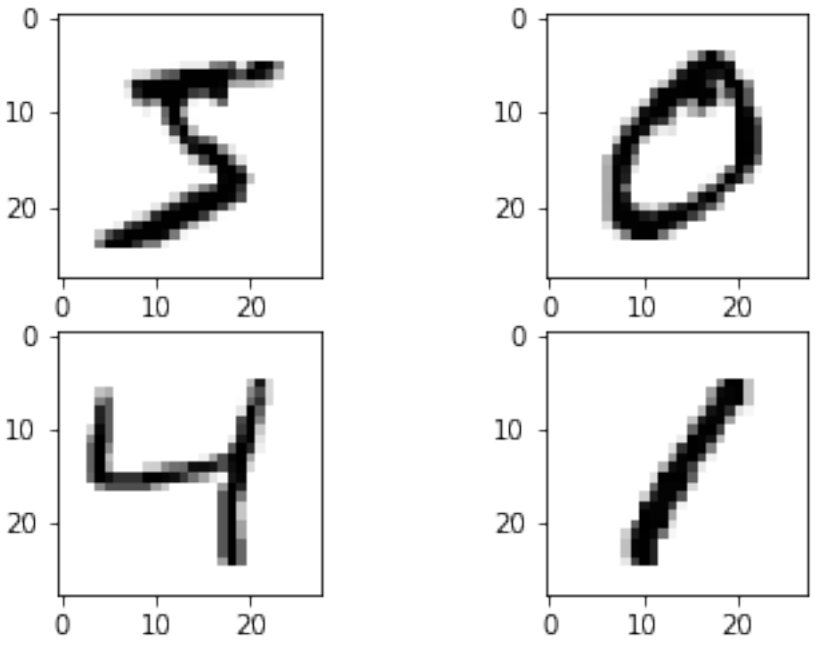
When starting out in the field of machine learning, one way or another, it is invariable that one will encounter the MNIST dataset. For those uninitiated, the MNIST dataset is a collection of 70,000 images of handwritten digits. This project will attempt to correctly classify what those digits are.

[1]:
```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784', version = 1)
#relevant dict keys = data, target, feature_names
```

Here's what a few of them look like:

[2]:
```
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
digits = mnist['data'][0:4]
pic_list = list()
for digit in digits:
    pic_list.append(digit.reshape(28, 28))

ax = plt.subplot(2, 2, 1)
ax1 = plt.subplot(2, 2, 2)
ax2 = plt.subplot(2, 2, 3)
ax3 = plt.subplot(2, 2, 4)
ax.imshow(pic_list[0], cmap = 'binary')
ax1.imshow(pic_list[1], cmap = 'binary')
ax2.imshow(pic_list[2], cmap = 'binary')
ax3.imshow(pic_list[3], cmap = 'binary')
plt.show()
```



To prepare the data for the neural network, it must be normalized (with dividing by 255, the max RGB value), the targets one-hot encoded, and split. It should be noted that the dataset is pre-split at the index 60000.

[3]:
```
from tensorflow import keras
from tensorflow.keras.utils import to_categorical
x = mnist['data'] / 255
x = x.reshape(len(x), 28, 28, 1)
y = to_categorical(mnist['target'])

trainx, trainy, testx, testy = x[:60000], y[:60000], x[60000:], y[60000:]
testy = testy.reshape(10000, 10)
trainy = trainy.reshape(60000, 10)
print(trainx.shape, trainy.shape, testx.shape, testy.shape)
```

```
(60000, 28, 28, 1) (60000, 10) (10000, 28, 28, 1) (10000, 10)
```

With the data properly prepared, it is time to construct the model. Since this dataset is so well researched, it is known that a neural network is the proper approach, specifically, a convolutional neural network. As far as images go, a 28 x 28 picture is not too large of an image, so only one pair of convolutional and pooling layers will be used. After the image features are learned, the data will go through a flatten layer and a dense layer to determine how the image should be classified.

[4]:
```
from tensorflow.keras.layers import Dense, Conv2D, Dropout, Flatten,␣
↪MaxPooling2D

model = keras.models.Sequential([
```

```
Conv2D(32, 3, activation = 'relu', input_shape = [28, 28, 1],␣
 ↪kernel_initializer='he_uniform'),
MaxPooling2D(2),
Flatten(),
Dense(100, activation = 'relu', kernel_initializer='he_uniform'),
Dense(10, activation = 'softmax')
])
```

[5]:
```
from tensorflow.keras.optimizers import SGD
opt = SGD(lr=0.01, momentum=0.9)
model.compile(optimizer = opt, loss = 'categorical_crossentropy', metrics =␣
 ↪['accuracy'])
fitted = model.fit(x = trainx, y = trainy, epochs = 3)
```

```
Epoch 1/3
1875/1875 [==============================] - 52s 28ms/step - loss: 0.1533 -
accuracy: 0.9534
Epoch 2/3
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0560 -
accuracy: 0.9833
Epoch 3/3
1875/1875 [==============================] - 51s 27ms/step - loss: 0.0375 -
accuracy: 0.9885
```

It is important to be aware of how many times the model is trained (epochs). Too little, and an inaccurate model is produced, but if it is trained too much, overfitting occurs. At three epochs, the gains on accuracy and loss diminish to the point where it is probably time to evaluate the model. A quick peek at the above output will show that the model should perform quite well if it were to be evaluated, provided the model has not been overfit.

[6]:
```
loss, acc = model.evaluate(testx, testy)
print('The model predicts handwritten numbers correctly at an accuracy rate of␣
 ↪%s percent with a loss of %s'
      % (round(acc * 100, 2),  round(loss, 3)))
```

```
313/313 [==============================] - 3s 9ms/step - loss: 0.0484 -
accuracy: 0.9830
The model predicts handwritten numbers correctly at an accuracy rate of 98.3
percent with a loss of 0.048
```

This model is about as good as it is going to get. It shows no signs of under/overfitting, and is ready for use.

[ ]:
```

```