Assignment_6.1

January 14, 2021

0.1 File information

File: Assignment_6.1.ipynb

Name: Amie Davis

Date: 1/13/2021

Course: DSC650 - Big Data

Assignment Number: 6.1

Purpose: Create a ConvNet model that classifies images in the MNIST digit dataset.

1 Train the convnet on MNIST images

1.1 This file contains code from Deep Learning with Python

www.manning.com/books/deep-learning-with-python

1.2 Data Source: The MNIST dataset - comes packaged with Keras.

[2]:
```
# Import required packages
import keras

from keras import layers
from keras import models
from keras.datasets import mnist
from keras.utils import to_categorical

import os
from pathlib import Path
```

[3]:
```
# Set results directory for writing
import os

current_dir = Path(os.getcwd()).absolute()
results_dir = current_dir.joinpath('results')
results_dir.mkdir(parents=True, exist_ok=True)
```

```
output_path = results_dir.joinpath('6.1_output.txt')
model_path = results_dir.joinpath('6.1_model.h5')
```

[4]:
```
# Instantiate ConvNet
model = models.Sequential()

# Build ConvNet
# Stack of Conv2D and MaxPooling2D layers
# Input shape is height x width x channel
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

[5]:
```
# Show layer details
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 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

[6]:
```
# Last output is of shape (3, 3, 64)
# Flatten 3D output to 1D
model.add(layers.Flatten())

# Add densely-connected classifier network
# Final output has 10 classifications (each digit)
# Use softmax activation fxn since multi-classification problem
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
```

[7]:
```
# Show layer details
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 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
flatten (Flatten)            (None, 576)               0
_________________________________________________________________
dense (Dense)                (None, 64)                36928
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

[8]:
```
# Output model summary to file
with open(output_path, 'w') as f:
    f.write('Model Summary:')
    f.write('\n')

    # Pass the file handle in as a lambda function to make it callable
    model.summary(print_fn=lambda x: f.write(x + '\n'))
```

[9]:
```
# Load data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Prepare training data
# Reshape to sample of 60,000, height=28, width=28, channel=1 (greyscale)
train_images = train_images.reshape((60000, 28, 28, 1))
# Normalize to values btwn 0 and 1
train_images = train_images.astype('float32') / 255

# Prepare test data
# Reshape to sample of 60,000, height=28, width=28, channel=1 (greyscale)
```

```
test_images = test_images.reshape((10000, 28, 28, 1))
# Normalize to values btwn 0 and 1
test_images = test_images.astype('float32') / 255

# Prepare labels
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

[10]:
```
# Train ConvNet on the MNIST digits.
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 13s 14ms/step - loss: 0.1713 -
accuracy: 0.9466
Epoch 2/5
938/938 [==============================] - 13s 14ms/step - loss: 0.0477 -
accuracy: 0.9851
Epoch 3/5
938/938 [==============================] - 13s 13ms/step - loss: 0.0333 -
accuracy: 0.9896
Epoch 4/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0246 -
accuracy: 0.9927
Epoch 5/5
938/938 [==============================] - 12s 13ms/step - loss: 0.0192 -
accuracy: 0.9944
```

[10]:
```
<tensorflow.python.keras.callbacks.History at 0x7ffa69122970>
```

[11]:
```
# Save Model
model.save(model_path)
```

[12]:
```
# Evaluate model on test data
test_loss, test_acc = model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 4ms/step - loss: 0.0270 -
accuracy: 0.9913
```

[13]:
```
# Show test accuracy
test_acc
```

[13]:
```
0.9912999868392944
```

[14]:

```python
# Output model test accuracy to file
with open(output_path, 'a') as f:
    f.write('\n')
    f.write('Test accuracy:')
    f.write(str(test_acc))
```

Test accuracy is 99.1%.

[ ]:

```python

```