Assignment 6.1

October 12, 2020

1 Assignment 6.1

Brandon Sams

05Oct2020

Using section 5.1 in Deep Learning with Python as a guide (listing 5.3 in particular), create a ConvNet model that classifies images in the MNIST digit dataset. Save the model, predictions, metrics, and validation plots in the dsc650/assignments/assignment06/results directory. If you are using JupyterHub, you can include those plots in your Jupyter notebook.

1.1 Instantiating a small convnet

[1]:
```
from keras import layers
from keras import models
```

[2]:
```
model = models.Sequential()
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',input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3),activation='relu',input_shape=(28,28,1)))
```

[3]:
```
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
_________________________________________________________________
```

1.2 Adding a classifier on top of the convnet

[4]:
```python
model.add(layers.Flatten())
model.add(layers.Dense(64,activation='relu'))
model.add(layers.Dense(10,activation='softmax'))
```

[5]:
```python
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
_________________________________________________________________
```

1.3 Training the convnet on MNIST images

[6]:
```python
from keras.datasets import mnist
from keras.utils import to_categorical
```

[7]:
```python
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype('float32') / 255
```

```python
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

[8]:
```python
model.compile(optimizer='rmsprop',
             loss='categorical_crossentropy',
             metrics=['accuracy'])

model.fit(train_images,
         train_labels,
         epochs=50,
         batch_size=64)
```

```
Epoch 1/50
938/938 [==============================] - 15s 16ms/step - loss: 0.1699 -
accuracy: 0.9464
Epoch 2/50
938/938 [==============================] - 15s 16ms/step - loss: 0.0480 -
accuracy: 0.9853
Epoch 3/50
938/938 [==============================] - 15s 15ms/step - loss: 0.0329 -
accuracy: 0.9900
Epoch 4/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0246 -
accuracy: 0.9923
Epoch 5/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0193 -
accuracy: 0.9943
Epoch 6/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0154 -
accuracy: 0.9952
Epoch 7/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0118 -
accuracy: 0.9964
Epoch 8/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0107 -
accuracy: 0.9967
Epoch 9/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0089 -
accuracy: 0.9972
Epoch 10/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0071 -
accuracy: 0.9978
Epoch 11/50
```

```
938/938 [==============================] - 14s 15ms/step - loss: 0.0063 -
accuracy: 0.9981
Epoch 12/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0052 -
accuracy: 0.9986
Epoch 13/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0052 -
accuracy: 0.9985
Epoch 14/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0046 -
accuracy: 0.9987
Epoch 15/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0052 -
accuracy: 0.9986
Epoch 16/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0036 -
accuracy: 0.9990
Epoch 17/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0033 -
accuracy: 0.9991
Epoch 18/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0030 -
accuracy: 0.9991
Epoch 19/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0033 -
accuracy: 0.9993
Epoch 20/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0028 -
accuracy: 0.9992
Epoch 21/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0034 -
accuracy: 0.9991
Epoch 22/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0041 -
accuracy: 0.9991
Epoch 23/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0026 -
accuracy: 0.9993
Epoch 24/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0028 -
accuracy: 0.9993
Epoch 25/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0032 -
accuracy: 0.9993
Epoch 26/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0034 -
accuracy: 0.9991
Epoch 27/50
```

```
938/938 [==============================] - 14s 15ms/step - loss: 0.0021 -
accuracy: 0.9995
Epoch 28/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0012 -
accuracy: 0.9997
Epoch 29/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0019 -
accuracy: 0.9995
Epoch 30/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0027 -
accuracy: 0.9993
Epoch 31/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0026 -
accuracy: 0.9993
Epoch 32/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0018 -
accuracy: 0.9995
Epoch 33/50
938/938 [==============================] - 15s 16ms/step - loss: 0.0019 -
accuracy: 0.9995
Epoch 34/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0025 -
accuracy: 0.9995
Epoch 35/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0011 -
accuracy: 0.9997
Epoch 36/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0015 -
accuracy: 0.9997
Epoch 37/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0014 -
accuracy: 0.9997
Epoch 38/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0013 -
accuracy: 0.9997
Epoch 39/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0016 -
accuracy: 0.9997
Epoch 40/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0019 -
accuracy: 0.9997
Epoch 41/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0019 -
accuracy: 0.9995
Epoch 42/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0010 -
accuracy: 0.9998
Epoch 43/50
```

```
938/938 [==============================] - 14s 15ms/step - loss: 0.0031 -
accuracy: 0.9995
Epoch 44/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0016 -
accuracy: 0.9997
Epoch 45/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0025 -
accuracy: 0.9996
Epoch 46/50
938/938 [==============================] - 14s 15ms/step - loss: 8.1034e-04 -
accuracy: 0.9998
Epoch 47/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0023 -
accuracy: 0.9995
Epoch 48/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0025 -
accuracy: 0.9996
Epoch 49/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0024 -
accuracy: 0.9996
Epoch 50/50
938/938 [==============================] - 14s 15ms/step - loss: 0.0033 -
accuracy: 0.9994
```

[8]:
```
<tensorflow.python.keras.callbacks.History at 0x7f456ed90760>
```

[9]:
```
# Let's evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_images,test_labels)
test_acc
```

```
313/313 [==============================] - 1s 5ms/step - loss: 0.1693 -
accuracy: 0.9907
```

[9]:
```
0.9907000064849854
```

1.4 Save model

[10]:
```
model.save('./results/model_6_1.h5')
```

[11]:
```
import pandas as pd
import numpy as np

num_testing = len(test_images)
pred = np.argmax(model.predict(test_images), axis=-1)
sub_df = pd.DataFrame()
sub_df["ImageId"] = list(range(1, num_testing + 1))
sub_df["Label"] = pred
```

[12]:
```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

sub_df["True_Label"] = test_labels
```

[13]:
```
sub_df
```

[13]:

	ImageId	Label	True_Label
0	1	7	7
1	2	2	2
2	3	1	1
3	4	0	0
4	5	4	4
…	…	…	…
9995	9996	2	2
9996	9997	3	3
9997	9998	4	4
9998	9999	5	5
9999	10000	6	6

```
[10000 rows x 3 columns]
```

[14]:
```
sub_df[sub_df.Label != sub_df.True_Label]
```

[14]:

	ImageId	Label	True_Label
62	63	5	9
104	105	5	9
321	322	7	2
340	341	3	5
582	583	2	8
…	…	…	…
9664	9665	7	2
9698	9699	1	6
9700	9701	4	2
9729	9730	6	5
9839	9840	7	2

```
[93 rows x 3 columns]
```

[ ]:
```
```