6.1

July 13, 2021

1 Assignment 6.1

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

[4]:
```
# instantiate a model
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'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64, (3,3), activation='relu'))

# add a classifier on top of Convnet
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation = 'softmax'))

# view summary
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
```

```
conv2d_6 (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
_________________________________________________________________
```

[5]:
```
from keras.datasets import mnist
from keras.utils import to_categorical
import numpy as np
```

[6]:
```
(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

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)
```

[7]:
```
# shuffle training set
for _ in range(5):
    indexes = np.random.permutation(len(train_images))

train_images = train_images[indexes]
train_labels = train_labels[indexes]

# put 10,000 aside for validation
val_images = train_images[:10000,:]
val_labels = train_labels[:10000,:]

# keep the rest in training set
train_images2 = train_images[10000:,:]
train_labels2 = train_labels[10000:,:]

# view their shape
train_images2.shape, val_images.shape
```

[7]:
```
((50000, 28, 28, 1), (10000, 28, 28, 1))
```

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

history = model.fit(train_images2, train_labels2, epochs=5, batch_size=64,
                    validation_data = (val_images, val_labels))
```

```
Epoch 1/5
782/782 [==============================] - 13s 16ms/step - loss: 0.4360 -
accuracy: 0.8591 - val_loss: 0.0528 - val_accuracy: 0.9830
Epoch 2/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0553 -
accuracy: 0.9840 - val_loss: 0.0478 - val_accuracy: 0.9837
Epoch 3/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0354 -
accuracy: 0.9893 - val_loss: 0.0385 - val_accuracy: 0.9868
Epoch 4/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0241 -
accuracy: 0.9920 - val_loss: 0.0296 - val_accuracy: 0.9903
Epoch 5/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0200 -
accuracy: 0.9941 - val_loss: 0.0275 - val_accuracy: 0.9927
```

[9]:
```
history.history.keys()
```

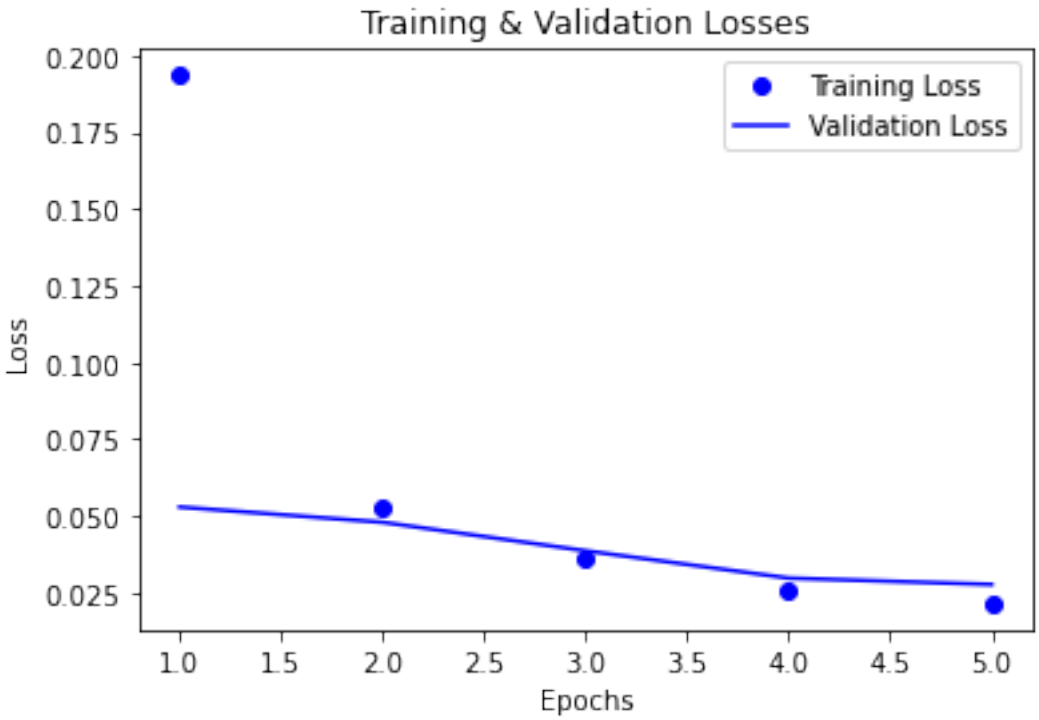
[9]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[10]:
```
import matplotlib.pyplot as plt
```

[13]:
```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(history.history['loss']) + 1)

plt.plot(epochs, train_loss, 'bo', label = 'Training Loss')
plt.plot(epochs, val_loss, 'b', label = 'Validation Loss')
plt.title('Training & Validation Losses')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.savefig('results/6_1_Loss.png')
```

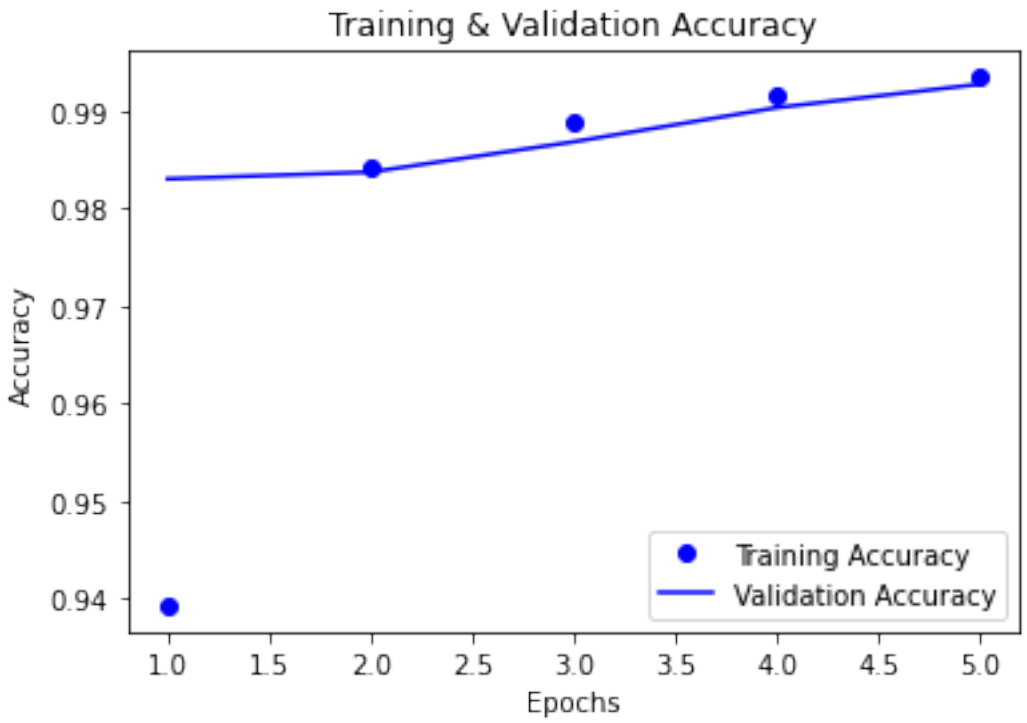



```
<Figure size 432x288 with 0 Axes>
```

[14]:
```
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epcohs = range(1, len(history.history['accuracy']) + 1)

plt.plot(epochs, train_acc, 'bo', label = 'Training Accuracy')
plt.plot(epochs, val_acc, 'b', label = 'Validation Accuracy')
plt.title('Training & Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
plt.savefig('results/6_1_Accuracy')
```



<Figure size 432x288 with 0 Axes>

[15]:
```
# retrain model & evaluate for 3 epochs
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs = 3, batch_size = 64)
results = model.evaluate(test_images, test_labels)
```

```
Epoch 1/3
938/938 [==============================] - 12s 12ms/step - loss: 0.0208 -
accuracy: 0.9935
Epoch 2/3
938/938 [==============================] - 12s 12ms/step - loss: 0.0136 -
accuracy: 0.9951
Epoch 3/3
938/938 [==============================] - 12s 12ms/step - loss: 0.0113 -
accuracy: 0.9969
313/313 [==============================] - 1s 4ms/step - loss: 0.0334 -
accuracy: 0.9918
```

[16]:
```
results
```

[16]:
```
[0.03339030593633652, 0.9918000102043152]
```

[17]:
```
history.history
```

[17]:
```
{'loss': [0.01979673281311989, 0.014603352174162865, 0.011986973695456982],
 'accuracy': [0.9939000010490417, 0.995199978351593, 0.9965000152587891]}
```

[18]:
```
model.save('results/6_1_model.h5')
```

[19]:
```
prediction_results = model.predict(test_images)
```

[20]:
```
prediction_results
```

[20]:
```
array([[4.66899869e-12, 1.82152557e-11, 1.81191676e-11, …,
        1.00000000e+00, 1.84051441e-10, 4.50120691e-10],
       [5.26576027e-09, 2.70119031e-11, 1.00000000e+00, …,
        1.02967866e-16, 4.61887574e-15, 8.79112500e-18],
       [1.56529154e-08, 9.99997973e-01, 5.92940319e-09, …,
        1.39287908e-06, 1.69522110e-07, 2.49986787e-09],
       …,
       [5.61513774e-18, 1.49294720e-13, 1.95674754e-15, …,
        6.97360301e-13, 3.07600334e-10, 9.17092601e-14],
       [1.66979196e-13, 1.25102478e-16, 2.11825569e-17, …,
        1.00040495e-12, 2.75879920e-05, 1.25346497e-10],
       [1.07952047e-09, 1.48777448e-13, 1.89103977e-09, …,
        3.36854203e-15, 1.39291123e-09, 1.56850813e-12]], dtype=float32)
```

[24]:
```
with open('results/6_1_metrics.txt', 'w') as f:
    f.write('Training Loss: {}'.format(str(history.history['loss'])))
    f.write('\nTraining Accuracy: {}'.format(str(history.history['accuracy'])))
    f.write('\nTest Loss: {}'.format(results[0]))
    f.write('\nTest Accuracy: {}'.format(results[1]))
```

[25]:
```
import pandas as pd
```

[26]:
```
preds = pd.DataFrame(prediction_results,
                     columns = ['0','1','2','3','4','5','6','7','8','9'])

preds.to_csv('results/6_1_predicitons.csv', index = False)
```

[ ]:
```

```