VRANJAN_Week6.1

April 26, 2021

[1]:
```
# Course DSC 650 - Data Mining
# Name - Vikas Ranjan
# Assignment - Assignment 6.1 ConvNet model that classifies images in the MNIST␣
 ↪digit dataset
```

[2]:
```
# Import packages
from keras import layers
from keras import models
from keras.datasets import mnist
from keras.utils import to_categorical
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[3]:
```
#initiate a small convnet

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 clasifier on top of the convnet
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

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
_________________________________________________________________
```

[4]:
```
(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)

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

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

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

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

train_images2.shape, val_images.shape
```

[4]:
```
((50000, 28, 28, 1), (10000, 28, 28, 1))
```

[5]:
```
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: 0.4126 -
accuracy: 0.8688 - val_loss: 0.0659 - val_accuracy: 0.9792
Epoch 2/5
782/782 [==============================] - 11s 15ms/step - loss: 0.0583 -
accuracy: 0.9811 - val_loss: 0.0486 - val_accuracy: 0.9852
Epoch 3/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0365 -
accuracy: 0.9889 - val_loss: 0.0467 - val_accuracy: 0.9852
Epoch 4/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0262 -
accuracy: 0.9924 - val_loss: 0.0344 - val_accuracy: 0.9887
Epoch 5/5
782/782 [==============================] - 11s 14ms/step - loss: 0.0190 -
accuracy: 0.9943 - val_loss: 0.0323 - val_accuracy: 0.9902
```

[6]:
```
history.history.keys()
```

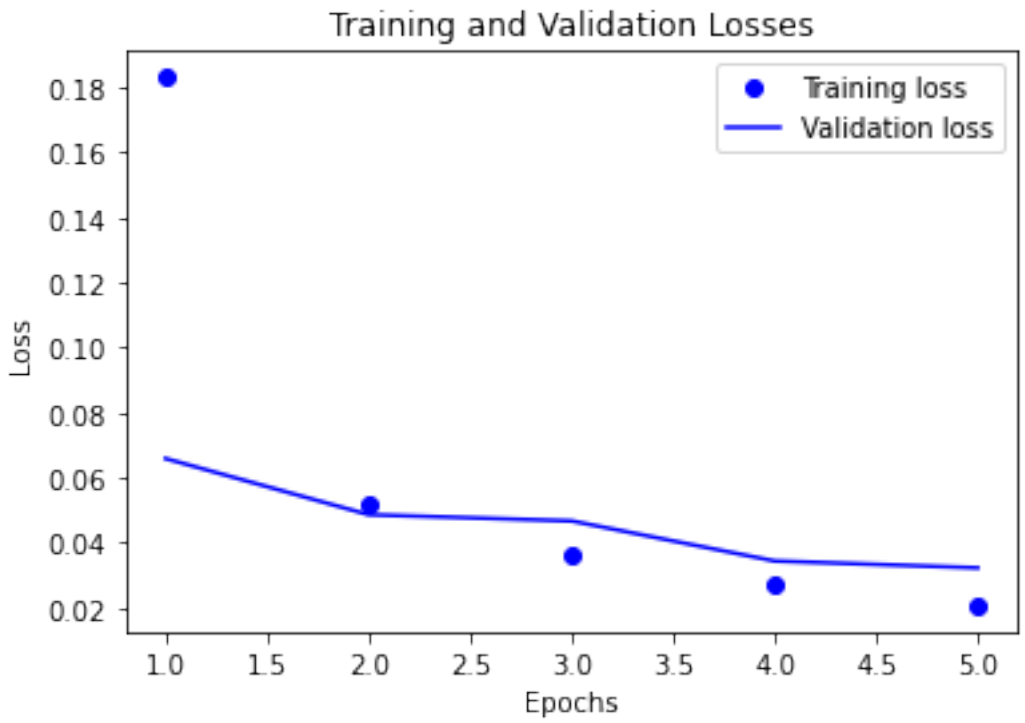
[6]:
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

[7]:
```
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 and Validation Losses')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
plt.savefig('results/6_1_lossplot.png')
```



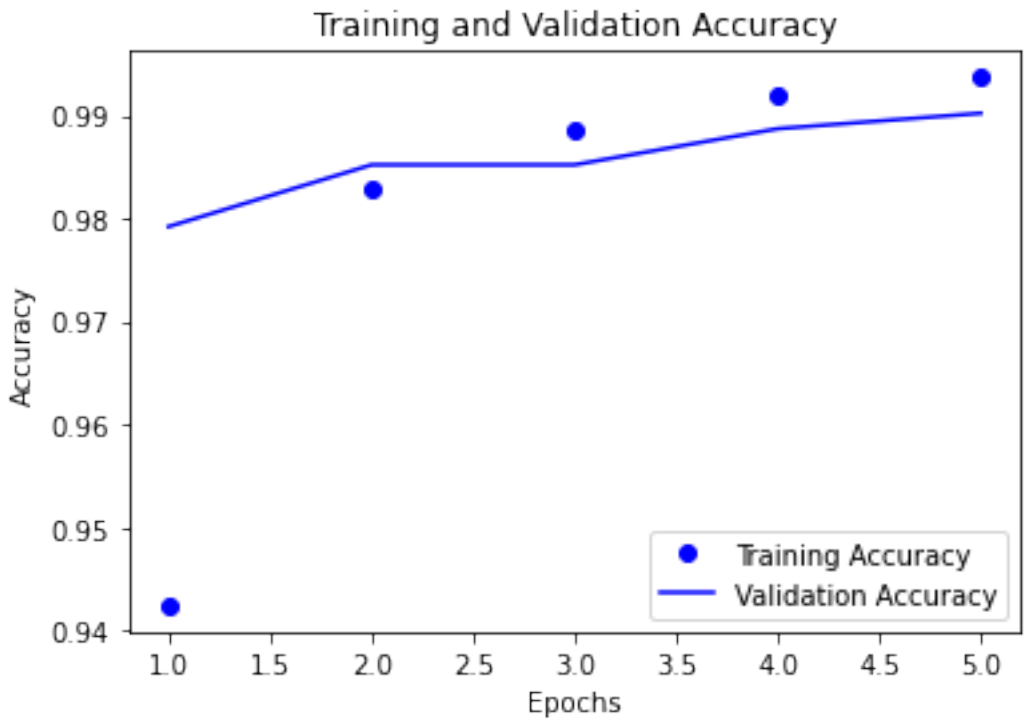

```
<Figure size 432x288 with 0 Axes>
```

[8]:
```
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

epochs = 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 and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.show()
plt.savefig('results/6_1_accplot.png')
```




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

[9]:
```
#retrain and 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 [==============================] - 13s 13ms/step - loss: 0.0207 -
accuracy: 0.9936
Epoch 2/3
938/938 [==============================] - 12s 13ms/step - loss: 0.0157 -
accuracy: 0.9957
Epoch 3/3
938/938 [==============================] - 12s 13ms/step - loss: 0.0111 -
accuracy: 0.9961
313/313 [==============================] - 1s 4ms/step - loss: 0.0388 -
accuracy: 0.9916
```

[10]:
```
results
```

[10]:
```
[0.0387532040476799, 0.991599977016449]
```

[11]:
```
history.history
```

[11]:
```
{'loss': [0.020284589380025864, 0.016059156507253647, 0.012322544120252132],
 'accuracy': [0.9939000010490417, 0.9951833486557007, 0.9959333539009094]}
```

[12]:
```
model.save('results/6_1_model.h5')
```

[13]:
```
prediction_results = model.predict(test_images)
```

[14]:
```
#write metrics to file
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]))
```

[15]:
```
predictions = pd.DataFrame(prediction_results,␣
↪columns=['0','1','2','3','4','5','6','7','8','9'])
predictions.to_csv('results/6_1_predictions.csv', index=False)
```

[ ]:
```

```