Report Handwritten Digit Recognition

Zemin XU

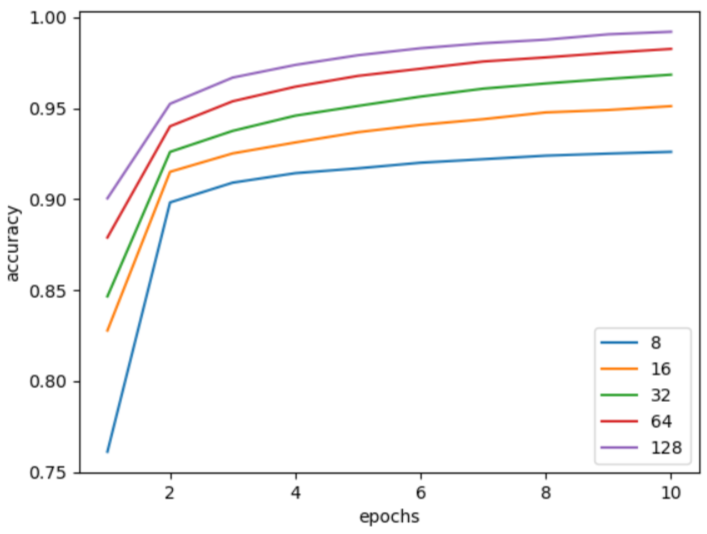
Exercise 1: Modify the number of neurons on the hidden layer as specified in Table 1. What can be observed regarding the system accuracy? How about the convergence time necessary for the system to train a model?

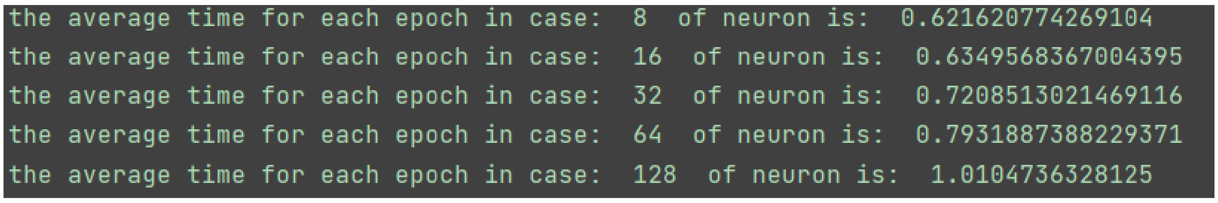
Answer:

Table 1.System performance evaluation for various numbers of neurons on the hidden layer

No of neurons	8	16	32	64	128
System accuracy	0.9233	0.9469	0.9659	0.9825	0.9913

The graph below shows the change of accuracy with increase of epoch for each case, we can see that after the 3rd epoch, the lines of all the cases become flat, which signs convergence. As the console output below, the convergence time is multiplied by 3. The case of 128 uses 3.03s, which is 60% more than the case of 8 and 16.





```
the average time for each epoch in case:  8  of neuron is:  0.621620774269104
the average time for each epoch in case:  16  of neuron is:  0.6349568367004395
the average time for each epoch in case:  32  of neuron is:  0.7208513021469116
the average time for each epoch in case:  64  of neuron is:  0.7931887388229371
the average time for each epoch in case:  128  of neuron is:  1.0104736328125
```

Exercise 2: Modify the batch size used to train the model as specified in Table 2. Select a value of 8 neurons for the hidden layer. What can be observed regarding the system accuracy?

Answer:

Table 2. System performance evaluation for different values of the batch size

Batch size	32	64	128	256	512
System accuracy	0.9275	0.9292	0.9249	0.9230	0.9121

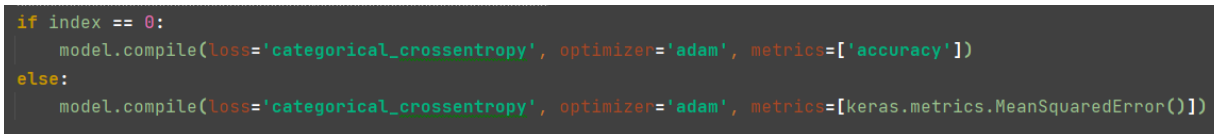
From the table above we can see that the accuracies are very close to 0.92. An observation is that the best system accuracy is obtained with a batch size of 64, with which we can draw a conclusion that a greater value in batch size will not always result in a better accuracy.

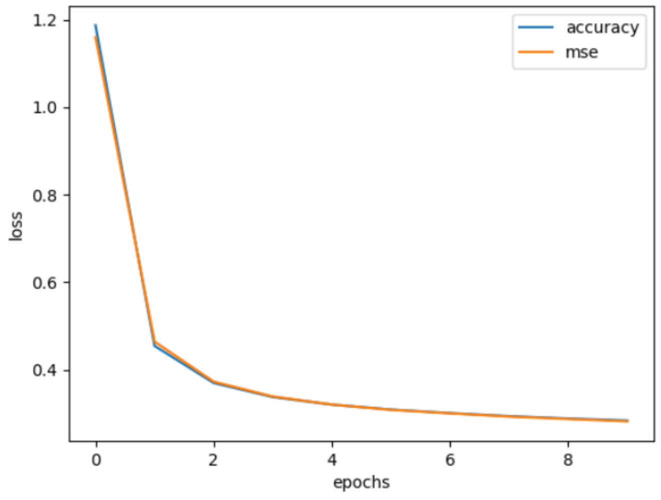
Exercise 3: Modify the metric used to determine the system performance from accuracy to mean square error (mse). Has this parameter any influence over the training process?

Answer:

This parameter will no influence over the training process, because metric is a way to interpret the loss and the fitting result. As the documentation of Keras explains: "metric functions are similar to loss functions, except that the results from evaluating a metric are not used when training the model.[1]"

As the graph below, using different metric parameters will not influence loss value at each epoch.

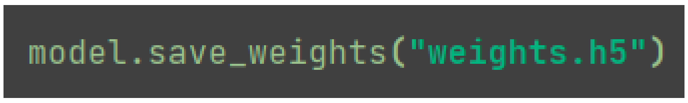

```
if index == 0:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
else:
    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=[keras.metrics.MeanSquaredError()])
```



Exercise 4: Using the default parameters specified at Application 1 (8 neurons in hidden layer, 10 epochs and a batch size of 200 images), train the model and save the associated weights in a file(model.save_weights), so it can be load anytime to perform further tests.

Answer:

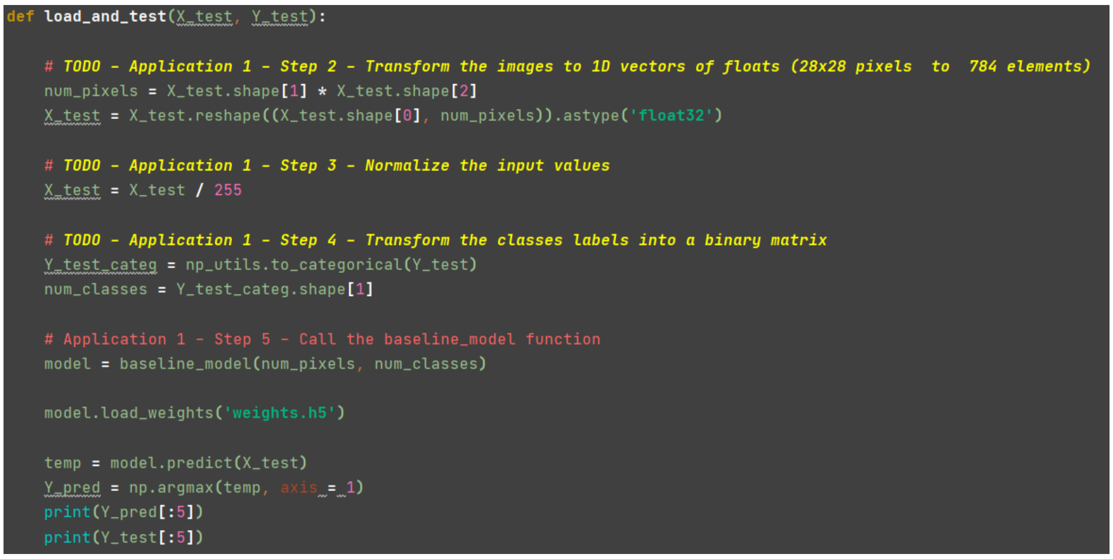
As the script exercise2_4.py shows, I use model.save_weights(weights.h5) to save these weights into a file "weights.h5", so that I can use it later.

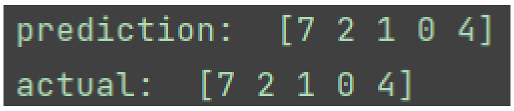

```
model.save_weights("weights.h5")
```

Exercise 5:Write a novel Pyhton script that loads the saved weights (model.load_weights) at Exercise 5 and make a prediction on the first 5 images of the testing dataset (mnist.test_images()).

Answer:

The script is implemented in “exercise2_5.py”, in which the “load and test()” function do the loading weights and prediction tasks. The result is the same as the images below.


```
def load_and_test(X_test, Y_test):

    # TODO - Application 1 - Step 2 - Transform the images to 1D vectors of floats (28x28 pixels  to  784 elements)
    num_pixels = X_test.shape[1] * X_test.shape[2]
    X_test = X_test.reshape((X_test.shape[0], num_pixels)).astype('float32')

    # TODO - Application 1 - Step 3 - Normalize the input values
    X_test = X_test / 255

    # TODO - Application 1 - Step 4 - Transform the classes labels into a binary matrix
    Y_test_categ = np_utils.to_categorical(Y_test)
    num_classes = Y_test_categ.shape[1]

    # Application 1 - Step 5 - Call the baseline_model function
    model = baseline_model(num_pixels, num_classes)

    model.load_weights('weights.h5')

    temp = model.predict(X_test)
    Y_pred = np.argmax(temp, axis = 1)
    print(Y_pred[:5])
    print(Y_test[:5])
```


```
prediction:  [7 2 1 0 4]
actual:  [7 2 1 0 4]
```

Exercise 6: Modify the size of the feature map within the convolutional layer as specified in Table 3. How is the system accuracy influenced by this parameter?

Answer:

Table 3. System performance evaluation for various sizes of the convolutional filters

Size of the feature map	1 x 1	3 x 3	5 x 5	7 x 7	9 x 9
System accuracy	0.9492	0.9781	0.9824	0.9846	0.9831

From the table above, we can see that as size of the feature map increases from 1*1 to 7*7, the accuracy increases as well, and the accuracy for 9*9 will be a bit worse than that of 7*7. Another observation is that the rate of improvement of accuracy is going down, as the size increases.

Exercise 7: Modify the number of neurons on the dense hidden layer as specified in Table 4. Select a value of 3x3 neurons for convolutional kernel. What can be observed regarding the system accuracy? How about the convergence time necessary for the system to train a model?

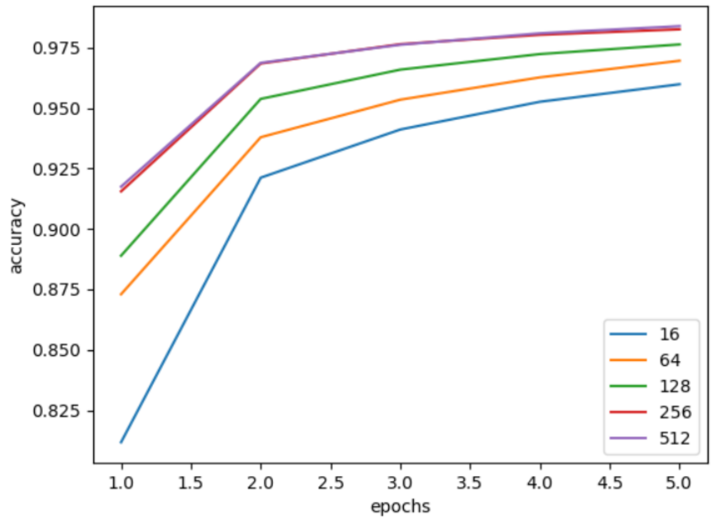
Answer:

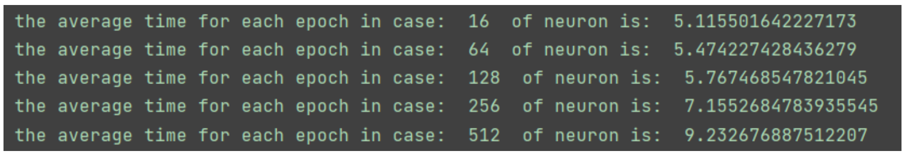
Table 4. System performance evaluation for various numbers of neurons on the dense hidden layer

No of neurons	16	64	128	256	512
System accuracy	0.9359	0.9734	0.9790	0.9822	0.9851

The graph below shows the change of accuracy with increase of epoch for each case, we can see that after the 3rd epoch, the lines of all the cases become flat, which signs convergence. It is worth noting that the result of case 512 is the same as that of 256, meaning that number greater than 256 as neurons will not improve accuracy.

As the console output below, the convergence time is multiplied by 3. The case of 512 uses 27.69s, which is 81% more than the case of 16.





```
the average time for each epoch in case:  16  of neuron is:  5.115501642227173
the average time for each epoch in case:  64  of neuron is:  5.474227428436279
the average time for each epoch in case:  128  of neuron is:  5.767468547821045
the average time for each epoch in case:  256  of neuron is:  7.1552684783935545
the average time for each epoch in case:  512  of neuron is:  9.232676887512207
```

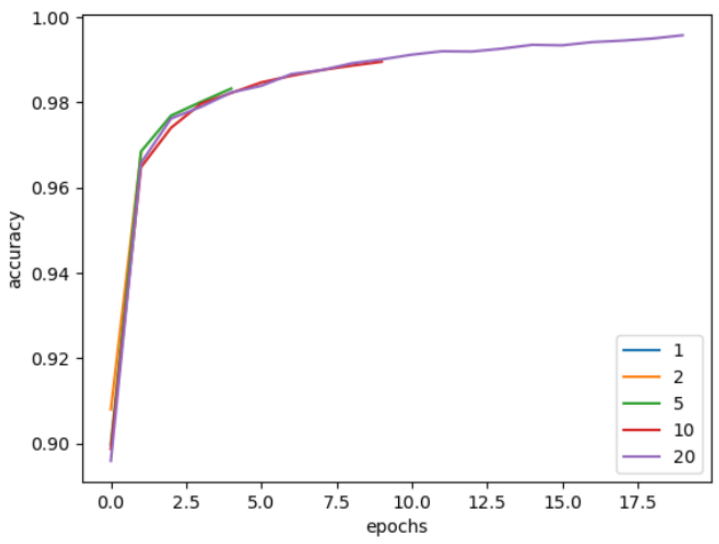
Exercise 8: Modify the number of epochs used to train the model as specified in Table 5. Select a value of 128 neurons in the dense hidden layer. What can be observed regarding the system accuracy? How about the convergence time?

Answer:

Table 5. System performance evaluation for different values of the number of epochs

No of epochs	1	2	5	10	20
System accuracy	0.8981	0.9680	0.9832	0.9896	0.9958

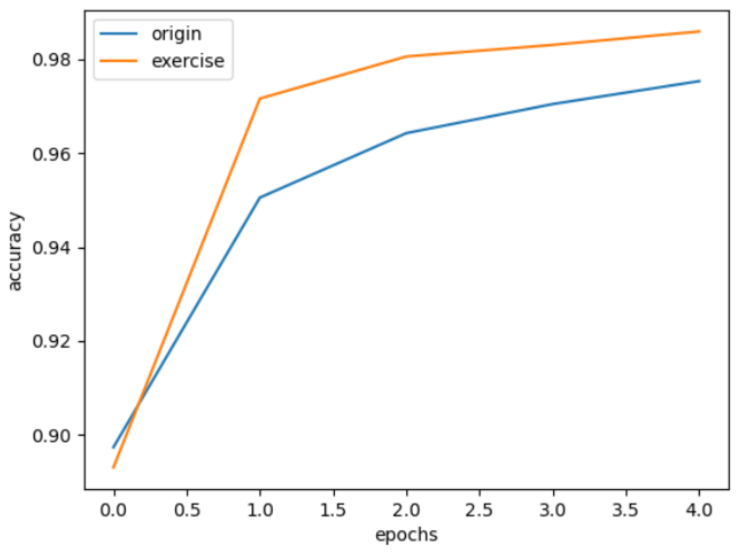
As the graph below and table above show, the accuracy keeps increasing as the number of epochs increases, but the changing rate keep decreasing. The model is converged after the 3rd epoch. Since the other variables remain the same, the time taken for each epoch is equal. Therefore, the convergence time is 5.77 * 3 = 17.31s, with the time for each epoch taken from exercise above.



Exercise 9: Modify the CNN architecture with additional convolutional, max pooling layers and fully connected layers. Determine the system accuracy using the novel configuration of the CNN architecture.

Answer:

The modified script can be found as "exercise2_9.py", with function name "CNN_model_second". The accuracy result is 0.9860(exercise) and 0.9754(origin), the former slightly better than the latter, which are very close to each other. In the graph below we can find that, with additional layers, the model converges after the 2nd epoch, however, the origin convers after the 3rd epoch.



Reference:

[1]. Metrics (keras.io)