Report of Handwritten digit classification using ANN / CNN

Xuan ZHOU
Institut Polytechnique de Parise
xuan.zhou@telecom-sudparis.eu

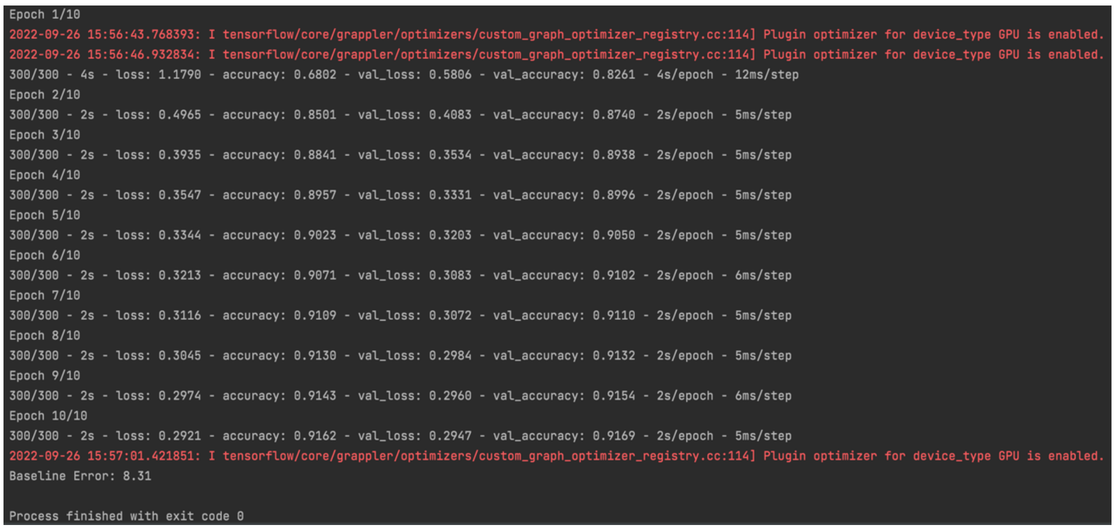
Part I. Artificial Neural Networks

This laboratory report is focused on the handwritten digit classification problem using artificial neural networks (ANN) or convolutional neural networks (CNN). On top of implementing a numerical recognition framework and determining the performance of the system in terms of loss and accuracy metrics.

ANN made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other.

The purpose of this experiment is to determine the best solution for classifying handwritten digits in the MNIST dataset. Each grayscale image in this dataset is a 28x28 pixel square. A standard segmentation of the dataset was used to evaluate and compare models, with 60,000 images used for training and 10,000 for testing.

First, the MNIST dataset is loaded in Keras. Second, for recognition purposes, we use a baseline model with a multilayer perceptron. A simple neural network model with a single hidden layer is trained, and the training and test sets and their associated content labels are used as input. Third, the MNINT dataset is reshaped. The training set is loaded into a 3-dimensional array structure and the multilayer perceptron model goes to reduce the image into a pixel vector. That is, a 28x28 size image is reduced to 784 input vectors. Fourth, the input values are normalized and then all the labels are transformed into a binary matrix. Fifth, the model structure is created, and the model is evaluated with a test dataset. The results are displayed in Figure 1.



```
Epoch 1/10
2022-09-26 15:56:43.768393: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.
2022-09-26 15:56:46.932834: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.
300/300 - 4s - loss: 1.1790 - accuracy: 0.6802 - val_loss: 0.5806 - val_accuracy: 0.8261 - 4s/epoch - 12ms/step
Epoch 2/10
300/300 - 2s - loss: 0.4965 - accuracy: 0.8501 - val_loss: 0.4083 - val_accuracy: 0.8740 - 2s/epoch - 5ms/step
Epoch 3/10
300/300 - 2s - loss: 0.3935 - accuracy: 0.8841 - val_loss: 0.3534 - val_accuracy: 0.8938 - 2s/epoch - 5ms/step
Epoch 4/10
300/300 - 2s - loss: 0.3547 - accuracy: 0.8957 - val_loss: 0.3331 - val_accuracy: 0.8996 - 2s/epoch - 5ms/step
Epoch 5/10
300/300 - 2s - loss: 0.3344 - accuracy: 0.9023 - val_loss: 0.3203 - val_accuracy: 0.9050 - 2s/epoch - 5ms/step
Epoch 6/10
300/300 - 2s - loss: 0.3213 - accuracy: 0.9071 - val_loss: 0.3083 - val_accuracy: 0.9102 - 2s/epoch - 6ms/step
Epoch 7/10
300/300 - 2s - loss: 0.3116 - accuracy: 0.9109 - val_loss: 0.3072 - val_accuracy: 0.9110 - 2s/epoch - 5ms/step
Epoch 8/10
300/300 - 2s - loss: 0.3045 - accuracy: 0.9130 - val_loss: 0.2984 - val_accuracy: 0.9132 - 2s/epoch - 5ms/step
Epoch 9/10
300/300 - 2s - loss: 0.2974 - accuracy: 0.9143 - val_loss: 0.2960 - val_accuracy: 0.9154 - 2s/epoch - 6ms/step
Epoch 10/10
300/300 - 2s - loss: 0.2921 - accuracy: 0.9162 - val_loss: 0.2947 - val_accuracy: 0.9169 - 2s/epoch - 5ms/step
2022-09-26 15:57:01.421851: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_type GPU is enabled.
Baseline Error: 8.31

Process finished with exit code 0
```

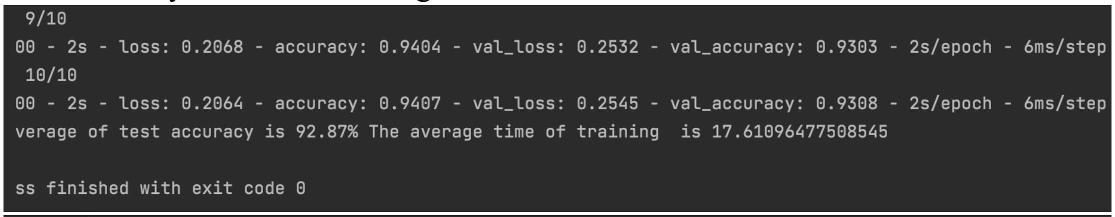
Figure 1. The result of the model

Considering the randomness of the algorithm or the evaluation procedure or the difference in numerical accuracy, the results are compared by modifying the number of neurons on the hidden layer. The results are shown in Table 1.

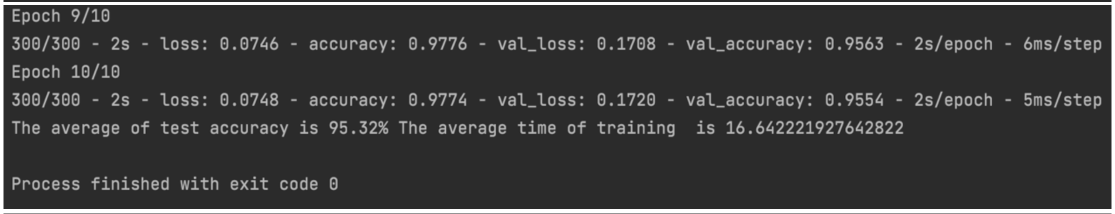
No of neurons	8	16	32	64	128
System accuracy	92.97%	95.32%	96.68%	97.46%	97.92%
Average time	17.61 s	16.64 s	17.14 s	17.21 s	17.75 s

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

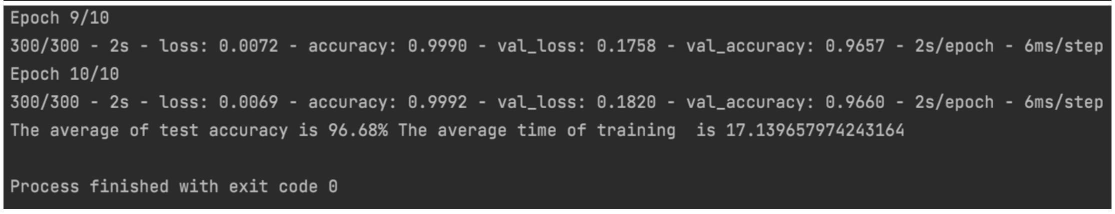
The results of the system performance evaluation for different numbers of neurons on the hidden layer are shown in Figure 2.



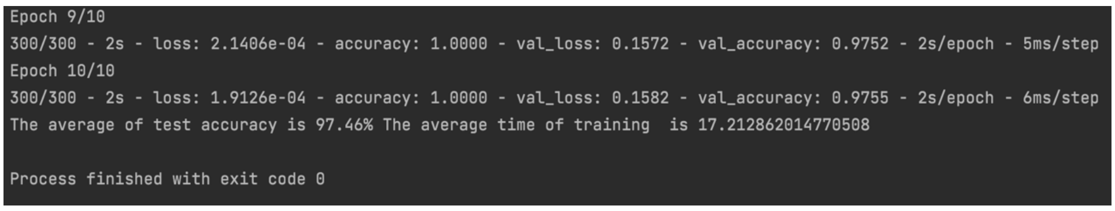
```
 9/10
00 - 2s - loss: 0.2068 - accuracy: 0.9404 - val_loss: 0.2532 - val_accuracy: 0.9303 - 2s/epoch - 6ms/step
 10/10
00 - 2s - loss: 0.2064 - accuracy: 0.9407 - val_loss: 0.2545 - val_accuracy: 0.9308 - 2s/epoch - 6ms/step
verage of test accuracy is 92.87% The average time of training  is 17.61096477508545

ss finished with exit code 0
```



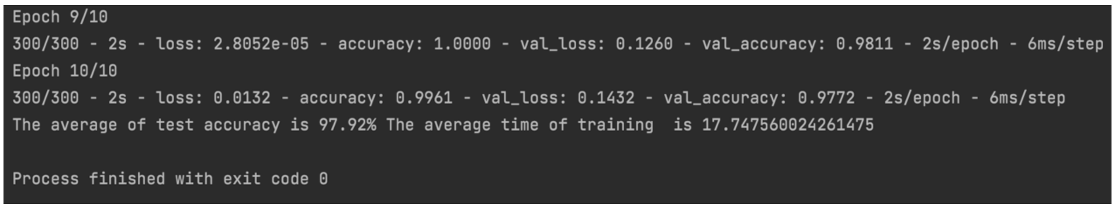
```
Epoch 9/10
300/300 - 2s - loss: 0.0746 - accuracy: 0.9776 - val_loss: 0.1708 - val_accuracy: 0.9563 - 2s/epoch - 6ms/step
Epoch 10/10
300/300 - 2s - loss: 0.0748 - accuracy: 0.9774 - val_loss: 0.1720 - val_accuracy: 0.9554 - 2s/epoch - 5ms/step
The average of test accuracy is 95.32% The average time of training  is 16.642221927642822

Process finished with exit code 0
```



```
Epoch 9/10
300/300 - 2s - loss: 0.0072 - accuracy: 0.9990 - val_loss: 0.1758 - val_accuracy: 0.9657 - 2s/epoch - 6ms/step
Epoch 10/10
300/300 - 2s - loss: 0.0069 - accuracy: 0.9992 - val_loss: 0.1820 - val_accuracy: 0.9660 - 2s/epoch - 6ms/step
The average of test accuracy is 96.68% The average time of training  is 17.139657974243164

Process finished with exit code 0
```



```
Epoch 9/10
300/300 - 2s - loss: 2.1406e-04 - accuracy: 1.0000 - val_loss: 0.1572 - val_accuracy: 0.9752 - 2s/epoch - 5ms/step
Epoch 10/10
300/300 - 2s - loss: 1.9126e-04 - accuracy: 1.0000 - val_loss: 0.1582 - val_accuracy: 0.9755 - 2s/epoch - 6ms/step
The average of test accuracy is 97.46% The average time of training  is 17.212862014770508

Process finished with exit code 0
```



```
Epoch 9/10
300/300 - 2s - loss: 2.8052e-05 - accuracy: 1.0000 - val_loss: 0.1260 - val_accuracy: 0.9811 - 2s/epoch - 6ms/step
Epoch 10/10
300/300 - 2s - loss: 0.0132 - accuracy: 0.9961 - val_loss: 0.1432 - val_accuracy: 0.9772 - 2s/epoch - 6ms/step
The average of test accuracy is 97.92% The average time of training  is 17.747560024261475

Process finished with exit code 0
```

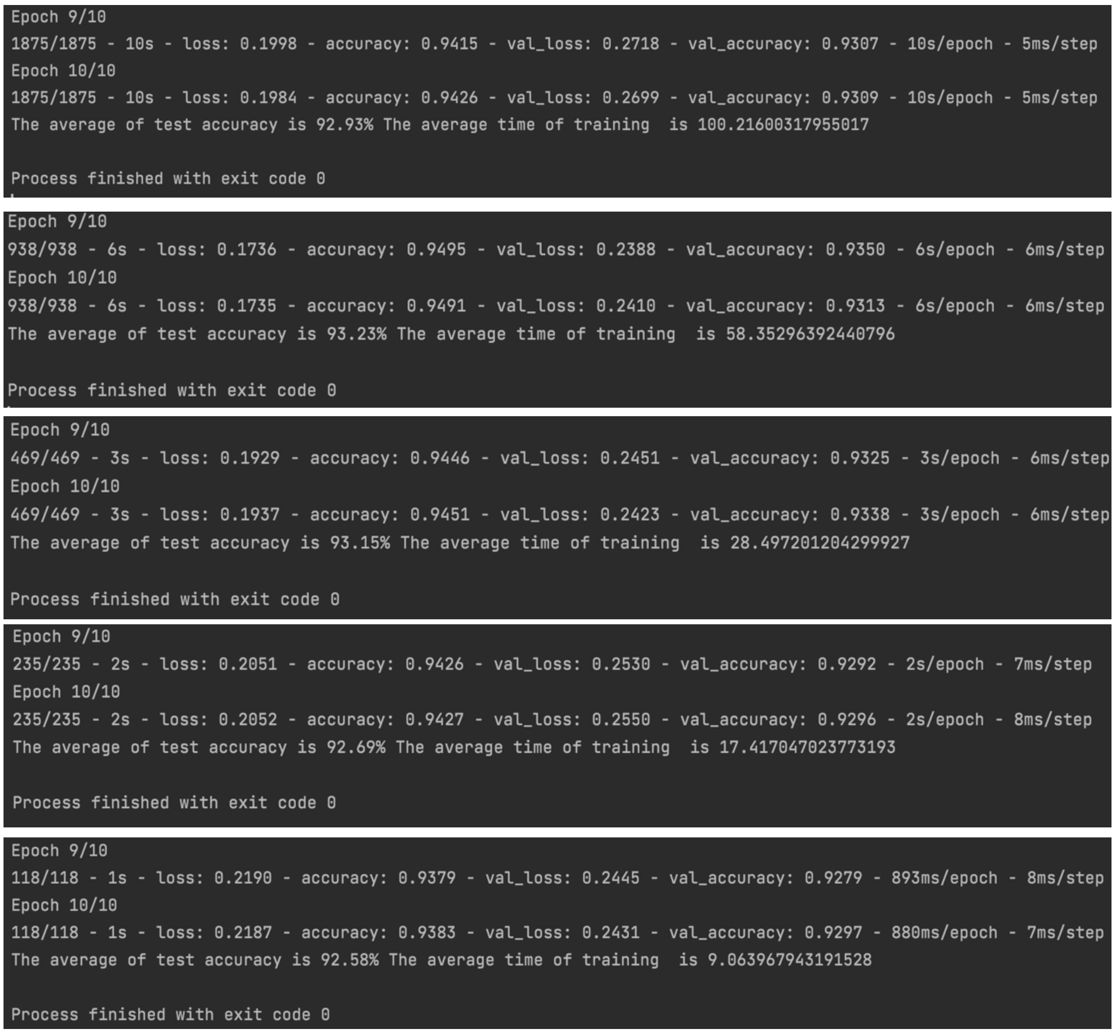
Figure 2. Results of modifying the number of neurons on the hidden layer

Next, by modifying the batch size used to train the model and selecting a value of 8 neurons for the hidden layer, the accuracy results of the observed system are shown in Table 2.

Batch size	32	64	128	256	512
System accuracy	92.93%	93.23%	93.15%	92.69%	92.58%
Average time	100.22 s	58.35 s	28.50 s	17.42 s	9.06 s

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

Figure 3 shows the results of the system performance evaluation for different values of the batch size.



```
Epoch 9/10
1875/1875 - 10s - loss: 0.1998 - accuracy: 0.9415 - val_loss: 0.2718 - val_accuracy: 0.9307 - 10s/epoch - 5ms/step
Epoch 10/10
1875/1875 - 10s - loss: 0.1984 - accuracy: 0.9426 - val_loss: 0.2699 - val_accuracy: 0.9309 - 10s/epoch - 5ms/step
The average of test accuracy is 92.93% The average time of training  is 100.21600317955017

Process finished with exit code 0
```

```
Epoch 9/10
938/938 - 6s - loss: 0.1736 - accuracy: 0.9495 - val_loss: 0.2388 - val_accuracy: 0.9350 - 6s/epoch - 6ms/step
Epoch 10/10
938/938 - 6s - loss: 0.1735 - accuracy: 0.9491 - val_loss: 0.2410 - val_accuracy: 0.9313 - 6s/epoch - 6ms/step
The average of test accuracy is 93.23% The average time of training  is 58.35296392440796

Process finished with exit code 0
```

```
Epoch 9/10
469/469 - 3s - loss: 0.1929 - accuracy: 0.9446 - val_loss: 0.2451 - val_accuracy: 0.9325 - 3s/epoch - 6ms/step
Epoch 10/10
469/469 - 3s - loss: 0.1937 - accuracy: 0.9451 - val_loss: 0.2423 - val_accuracy: 0.9338 - 3s/epoch - 6ms/step
The average of test accuracy is 93.15% The average time of training  is 28.497201204299927

Process finished with exit code 0
```

```
Epoch 9/10
235/235 - 2s - loss: 0.2051 - accuracy: 0.9426 - val_loss: 0.2530 - val_accuracy: 0.9292 - 2s/epoch - 7ms/step
Epoch 10/10
235/235 - 2s - loss: 0.2052 - accuracy: 0.9427 - val_loss: 0.2550 - val_accuracy: 0.9296 - 2s/epoch - 8ms/step
The average of test accuracy is 92.69% The average time of training  is 17.417047023773193

Process finished with exit code 0
```

```
Epoch 9/10
118/118 - 1s - loss: 0.2190 - accuracy: 0.9379 - val_loss: 0.2445 - val_accuracy: 0.9279 - 893ms/epoch - 8ms/step
Epoch 10/10
118/118 - 1s - loss: 0.2187 - accuracy: 0.9383 - val_loss: 0.2431 - val_accuracy: 0.9297 - 880ms/epoch - 7ms/step
The average of test accuracy is 92.58% The average time of training  is 9.063967943191528

Process finished with exit code 0
```

Figure 3. System performance evaluation results for different values of modified batch size

To determine the effect of the parameters on the performance of the system, next I will determine the effect of MSE on the training process by changing the precision to mean square error. First I will compare the results based on varying the number of neurons in the hidden layer as shown in Table 3. Then I will compare the impact of MSE evaluation on the system by varying the batch size with different values, and the results are shown in Table 4.

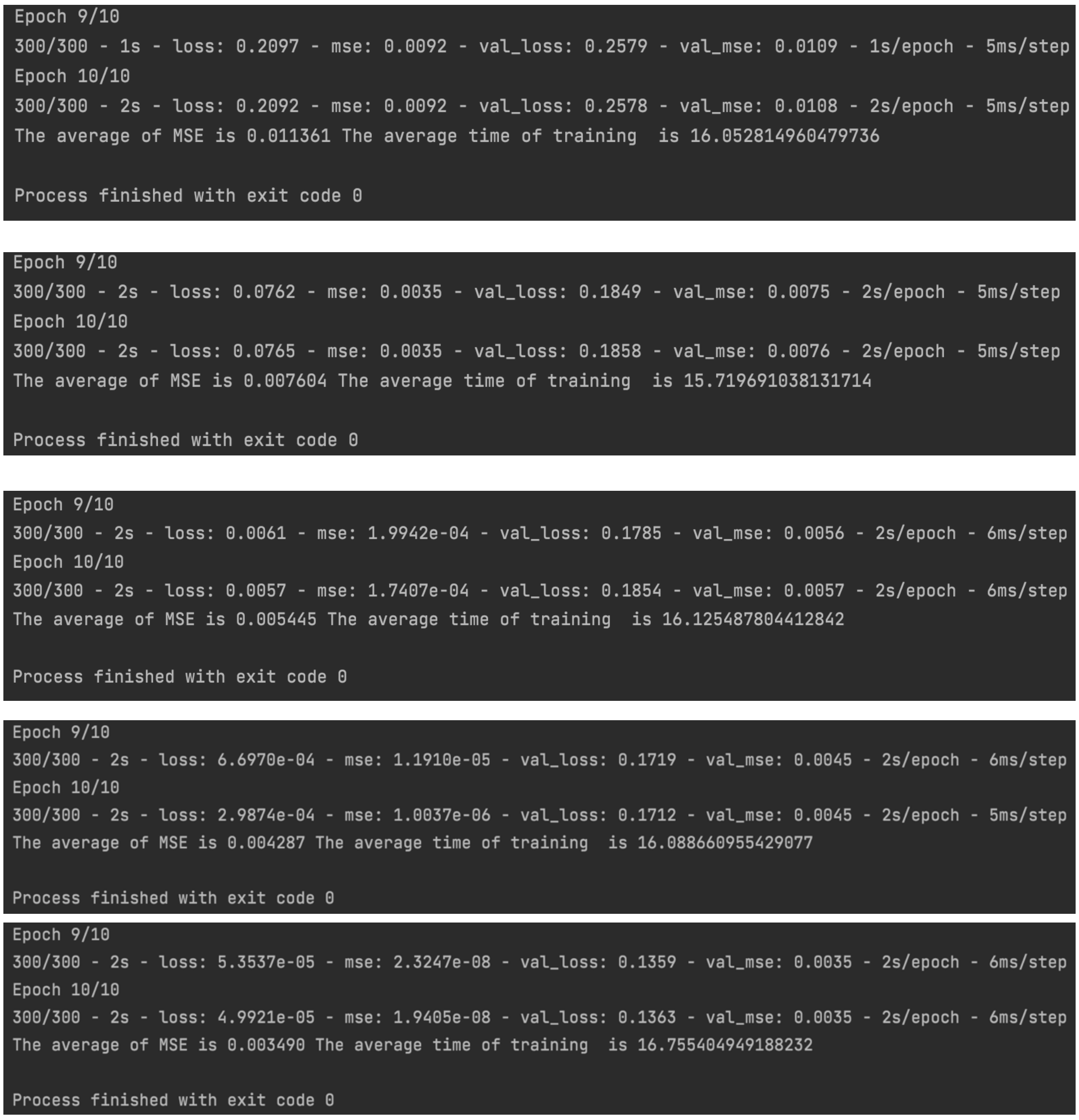
No of neurons	8	16	32	64	128
MSE	0.011361	0.007604	0.005445	0.004287	0.003490
Average time	16.05 s	15.72 s	16.13 s	16.10 s	16.76 s

Table 3. Results of modifying different numbers of hidden layers

Batch size	32	64	128	256	512
MSE	0.010859	0.010899	0.011590	0.011015	0.011772
Average time	88.47 s	52.67 s	27.60 s	14.04 s	7.97 s

Table 4. The result of modifying the value of the batch size

The impact of MSE on the training process based on comparing different numbers of neurons by varying the hidden layer is shown in Figure 4.



```
Epoch 9/10
300/300 - 1s - loss: 0.2097 - mse: 0.0092 - val_loss: 0.2579 - val_mse: 0.0109 - 1s/epoch - 5ms/step
Epoch 10/10
300/300 - 2s - loss: 0.2092 - mse: 0.0092 - val_loss: 0.2578 - val_mse: 0.0108 - 2s/epoch - 5ms/step
The average of MSE is 0.011361 The average time of training  is 16.052814960479736

Process finished with exit code 0
```

```
Epoch 9/10
300/300 - 2s - loss: 0.0762 - mse: 0.0035 - val_loss: 0.1849 - val_mse: 0.0075 - 2s/epoch - 5ms/step
Epoch 10/10
300/300 - 2s - loss: 0.0765 - mse: 0.0035 - val_loss: 0.1858 - val_mse: 0.0076 - 2s/epoch - 5ms/step
The average of MSE is 0.007604 The average time of training  is 15.719691038131714

Process finished with exit code 0
```

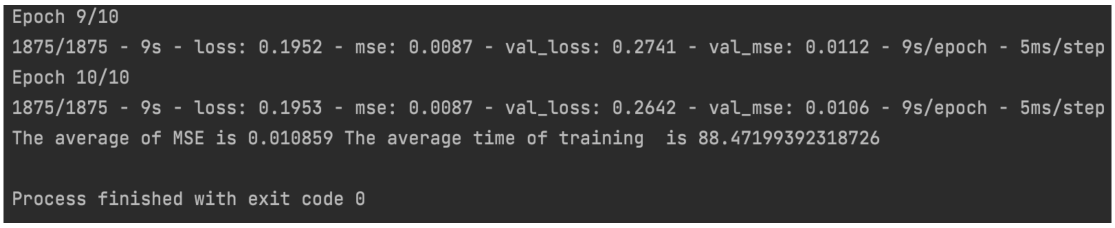
```
Epoch 9/10
300/300 - 2s - loss: 0.0061 - mse: 1.9942e-04 - val_loss: 0.1785 - val_mse: 0.0056 - 2s/epoch - 6ms/step
Epoch 10/10
300/300 - 2s - loss: 0.0057 - mse: 1.7407e-04 - val_loss: 0.1854 - val_mse: 0.0057 - 2s/epoch - 6ms/step
The average of MSE is 0.005445 The average time of training  is 16.125487804412842

Process finished with exit code 0
```

```
Epoch 9/10
300/300 - 2s - loss: 6.6970e-04 - mse: 1.1910e-05 - val_loss: 0.1719 - val_mse: 0.0045 - 2s/epoch - 6ms/step
Epoch 10/10
300/300 - 2s - loss: 2.9874e-04 - mse: 1.0037e-06 - val_loss: 0.1712 - val_mse: 0.0045 - 2s/epoch - 5ms/step
The average of MSE is 0.004287 The average time of training  is 16.088660955429077

Process finished with exit code 0
```

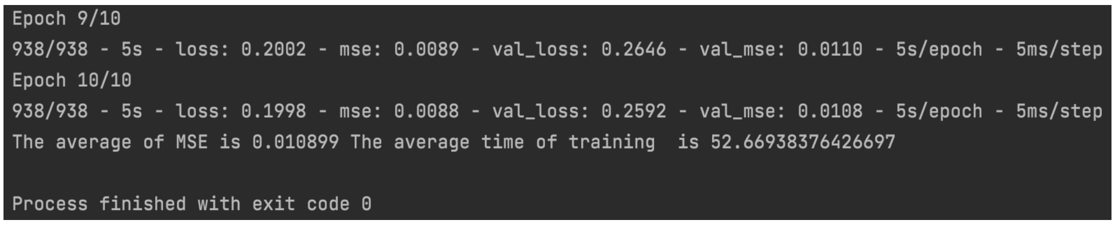
```
Epoch 9/10
300/300 - 2s - loss: 5.3537e-05 - mse: 2.3247e-08 - val_loss: 0.1359 - val_mse: 0.0035 - 2s/epoch - 6ms/step
Epoch 10/10
300/300 - 2s - loss: 4.9921e-05 - mse: 1.9405e-08 - val_loss: 0.1363 - val_mse: 0.0035 - 2s/epoch - 6ms/step
The average of MSE is 0.003490 The average time of training  is 16.755404949188232

Process finished with exit code 0
```

Figure 4. Results of varying different numbers of neurons in the hidden layer

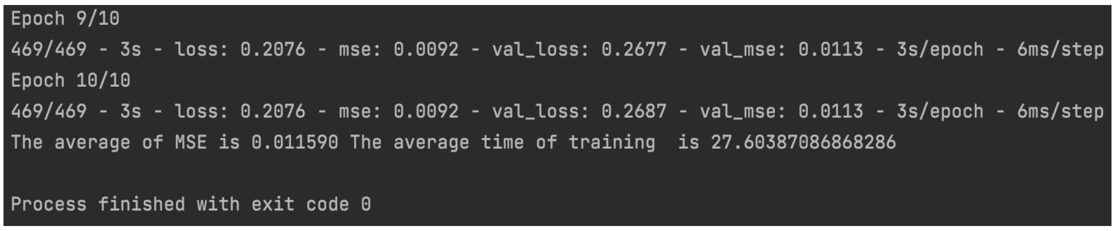
The impact of comparing MSE evaluation on the system based on varying different values of batch size is shown in Figure 5.



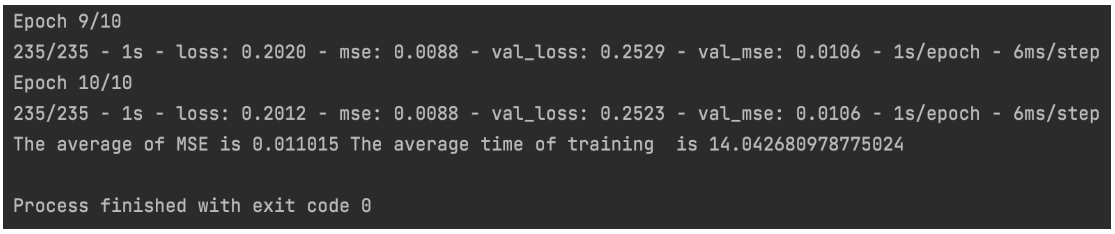
```
Epoch 9/10
1875/1875 - 9s - loss: 0.1952 - mse: 0.0087 - val_loss: 0.2741 - val_mse: 0.0112 - 9s/epoch - 5ms/step
Epoch 10/10
1875/1875 - 9s - loss: 0.1953 - mse: 0.0087 - val_loss: 0.2642 - val_mse: 0.0106 - 9s/epoch - 5ms/step
The average of MSE is 0.010859 The average time of training  is 88.47199392318726

Process finished with exit code 0
```



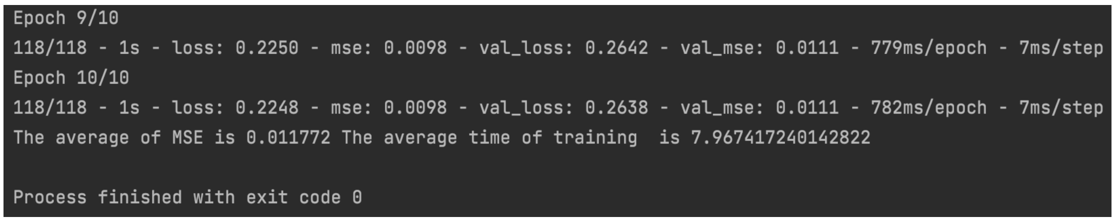
```
Epoch 9/10
938/938 - 5s - loss: 0.2002 - mse: 0.0089 - val_loss: 0.2646 - val_mse: 0.0110 - 5s/epoch - 5ms/step
Epoch 10/10
938/938 - 5s - loss: 0.1998 - mse: 0.0088 - val_loss: 0.2592 - val_mse: 0.0108 - 5s/epoch - 5ms/step
The average of MSE is 0.010899 The average time of training  is 52.66938376426697

Process finished with exit code 0
```



```
Epoch 9/10
469/469 - 3s - loss: 0.2076 - mse: 0.0092 - val_loss: 0.2677 - val_mse: 0.0113 - 3s/epoch - 6ms/step
Epoch 10/10
469/469 - 3s - loss: 0.2076 - mse: 0.0092 - val_loss: 0.2687 - val_mse: 0.0113 - 3s/epoch - 6ms/step
The average of MSE is 0.011590 The average time of training  is 27.60387086868286

Process finished with exit code 0
```



```
Epoch 9/10
235/235 - 1s - loss: 0.2020 - mse: 0.0088 - val_loss: 0.2529 - val_mse: 0.0106 - 1s/epoch - 6ms/step
Epoch 10/10
235/235 - 1s - loss: 0.2012 - mse: 0.0088 - val_loss: 0.2523 - val_mse: 0.0106 - 1s/epoch - 6ms/step
The average of MSE is 0.011015 The average time of training  is 14.042680978775024

Process finished with exit code 0
```



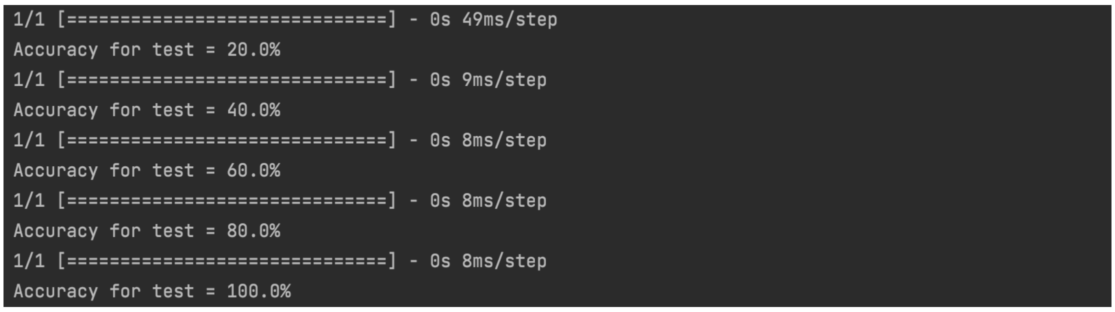
```
Epoch 9/10
118/118 - 1s - loss: 0.2250 - mse: 0.0098 - val_loss: 0.2642 - val_mse: 0.0111 - 779ms/epoch - 7ms/step
Epoch 10/10
118/118 - 1s - loss: 0.2248 - mse: 0.0098 - val_loss: 0.2638 - val_mse: 0.0111 - 782ms/epoch - 7ms/step
The average of MSE is 0.011772 The average time of training  is 7.967417240142822

Process finished with exit code 0
```

Figure 5. Results of varying different values of batch size

The impact on the system is evaluated by changing the precision to mean square error while comparing the MSE based on varying the number of neurons by varying the hidden layers and comparing the MSE based on varying different values of the batch size. As we modify the number of hidden layers, we can see that the EMS value decreases gradually, as shown in Table3. When we go back to Table1, we can see that the accuracy is gradually increasing as the number of hidden layers is modified. After modifying the value of batch size, the MSE value is gradually increasing, as shown in Table 5. When we answer Table 2, we find that the accuracy decreases after modifying the batch size.

This shows that the MSE value and accuracy are consistent in evaluating the system performance both when modifying the number of hidden layers and modifying the batch size, so modifying the MSE parameter has no effect on the training process.

Next, save the weights associated with the model and write a new python script to load the saved weights file. Striving to do the same as we did in application 1, we need to specify the artificial neural network and then load the set of weights before predicting the results on the input as shown in Figure 6.

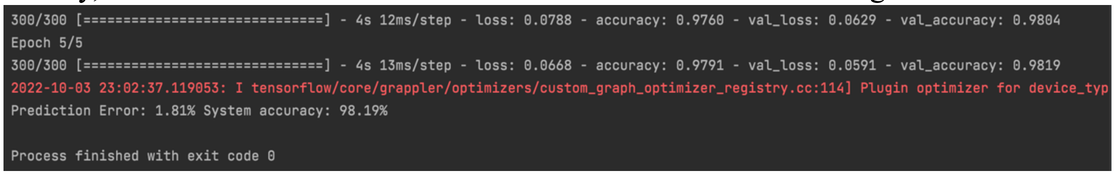


```
1/1 [==============================] - 0s 49ms/step
Accuracy for test = 20.0%
1/1 [==============================] - 0s 9ms/step
Accuracy for test = 40.0%
1/1 [==============================] - 0s 8ms/step
Accuracy for test = 60.0%
1/1 [==============================] - 0s 8ms/step
Accuracy for test = 80.0%
1/1 [==============================] - 0s 8ms/step
Accuracy for test = 100.0%
```

Figure 6. The results of predicted

Part II. Convolutional Neural Networks

A Convolutional Neural Network is a deep learning algorithm that can take in an input image, assign importance to various aspects/objects in the image and is able differentiate between different images. Convolutional Neural Networks take advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. A ConvNet can successfully capture the spatial dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and the reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. In particular, the layers of a ConvNet have neurons arranged in 3 dimensions: width, height and depth.
Next , we recognize numbers from the image as input by using CNN architecture. Firstly, the MINST dataset is reshaped. Secondly, normalize the dataset. Third, transform the category labels into binary matrices. Fourth, the CNN model architecture is built. Finally, the CNN model is evaluated and the results are shown in Figure 7.



```
300/300 [==============================] - 4s 12ms/step - loss: 0.0788 - accuracy: 0.9760 - val_loss: 0.0629 - val_accuracy: 0.9804
Epoch 5/5
300/300 [==============================] - 4s 13ms/step - loss: 0.0668 - accuracy: 0.9791 - val_loss: 0.0591 - val_accuracy: 0.9819
2022-10-03 23:02:37.119053: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_typ
Prediction Error: 1.81% System accuracy: 98.19%

Process finished with exit code 0
```

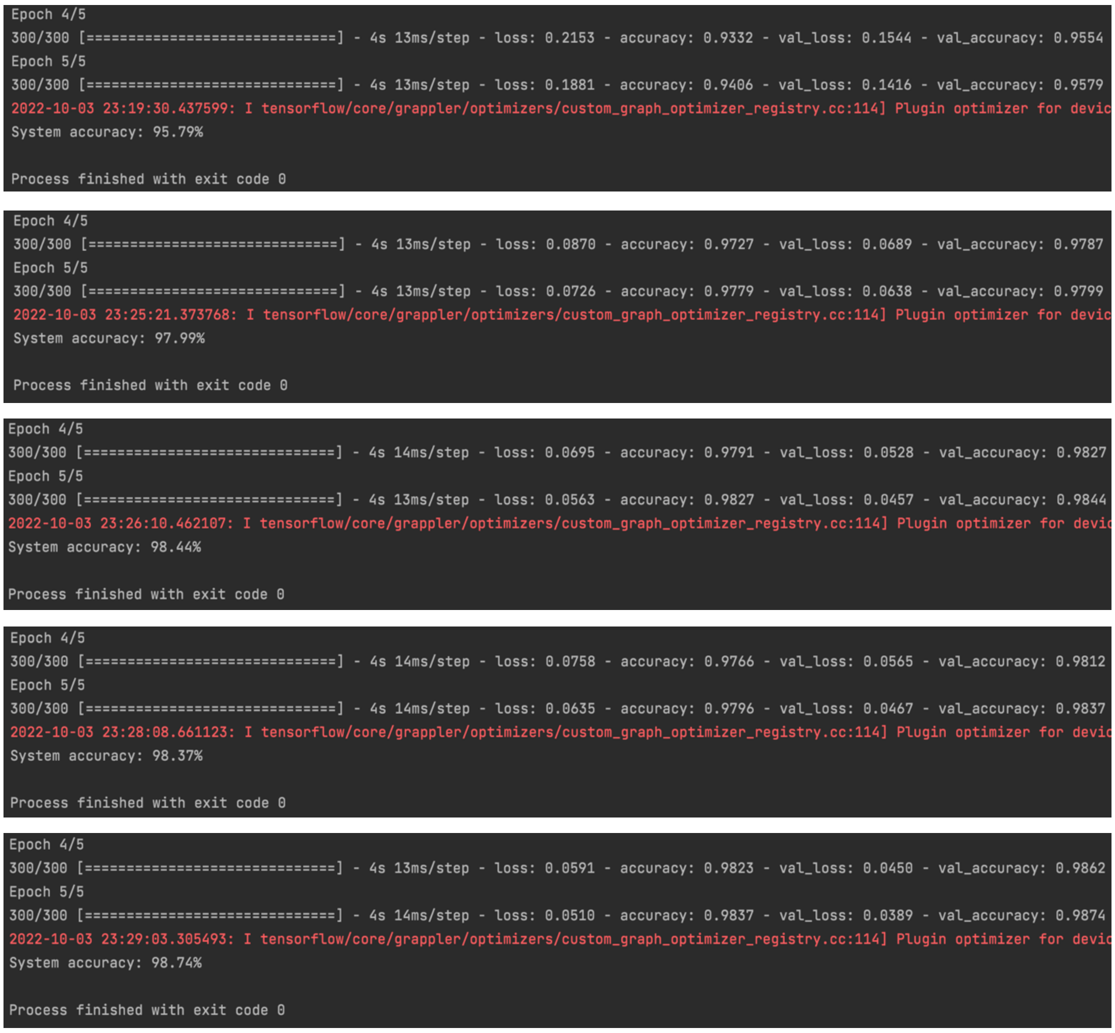
Figure 7. The result of the CNN model

Next, the effect of the parameters on the system was evaluated by modifying the feature map size within the convolutional layer. The sizes of the eigenmaps were 1x1,3x3,5x5,7x7 and 9x9.The results are shown in Table 5.

Size of the feature map	1 x 1	3 x 3	5 x 5	7 x 7	9 x 9
System accuracy	95.79%	97.99%	98.44%	98.37%	98.74

Table 5. The result of modifying the size of the features map within the volume base

The results of modifying the feature map size de within the convolutional layer are shown in Figure 8.



```
Epoch 4/5
300/300 [==============================] - 4s 13ms/step - loss: 0.2153 - accuracy: 0.9332 - val_loss: 0.1544 - val_accuracy: 0.9554
Epoch 5/5
300/300 [==============================] - 4s 13ms/step - loss: 0.1881 - accuracy: 0.9406 - val_loss: 0.1416 - val_accuracy: 0.9579
2022-10-03 23:19:30.437599: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for devic
System accuracy: 95.79%

Process finished with exit code 0
```

```
Epoch 4/5
300/300 [==============================] - 4s 13ms/step - loss: 0.0870 - accuracy: 0.9727 - val_loss: 0.0689 - val_accuracy: 0.9787
Epoch 5/5
300/300 [==============================] - 4s 13ms/step - loss: 0.0726 - accuracy: 0.9779 - val_loss: 0.0638 - val_accuracy: 0.9799
2022-10-03 23:25:21.373768: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for devic
System accuracy: 97.99%

Process finished with exit code 0
```

```
Epoch 4/5
300/300 [==============================] - 4s 14ms/step - loss: 0.0695 - accuracy: 0.9791 - val_loss: 0.0528 - val_accuracy: 0.9827
Epoch 5/5
300/300 [==============================] - 4s 13ms/step - loss: 0.0563 - accuracy: 0.9827 - val_loss: 0.0457 - val_accuracy: 0.9844
2022-10-03 23:26:10.462107: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for devic
System accuracy: 98.44%

Process finished with exit code 0
```

```
Epoch 4/5
300/300 [==============================] - 4s 14ms/step - loss: 0.0758 - accuracy: 0.9766 - val_loss: 0.0565 - val_accuracy: 0.9812
Epoch 5/5
300/300 [==============================] - 4s 14ms/step - loss: 0.0635 - accuracy: 0.9796 - val_loss: 0.0467 - val_accuracy: 0.9837
2022-10-03 23:28:08.661123: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for devic
System accuracy: 98.37%

Process finished with exit code 0
```

```
Epoch 4/5
300/300 [==============================] - 4s 13ms/step - loss: 0.0591 - accuracy: 0.9823 - val_loss: 0.0450 - val_accuracy: 0.9862
Epoch 5/5
300/300 [==============================] - 4s 14ms/step - loss: 0.0510 - accuracy: 0.9837 - val_loss: 0.0389 - val_accuracy: 0.9874
2022-10-03 23:29:03.305493: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for devic
System accuracy: 98.74%

Process finished with exit code 0
```

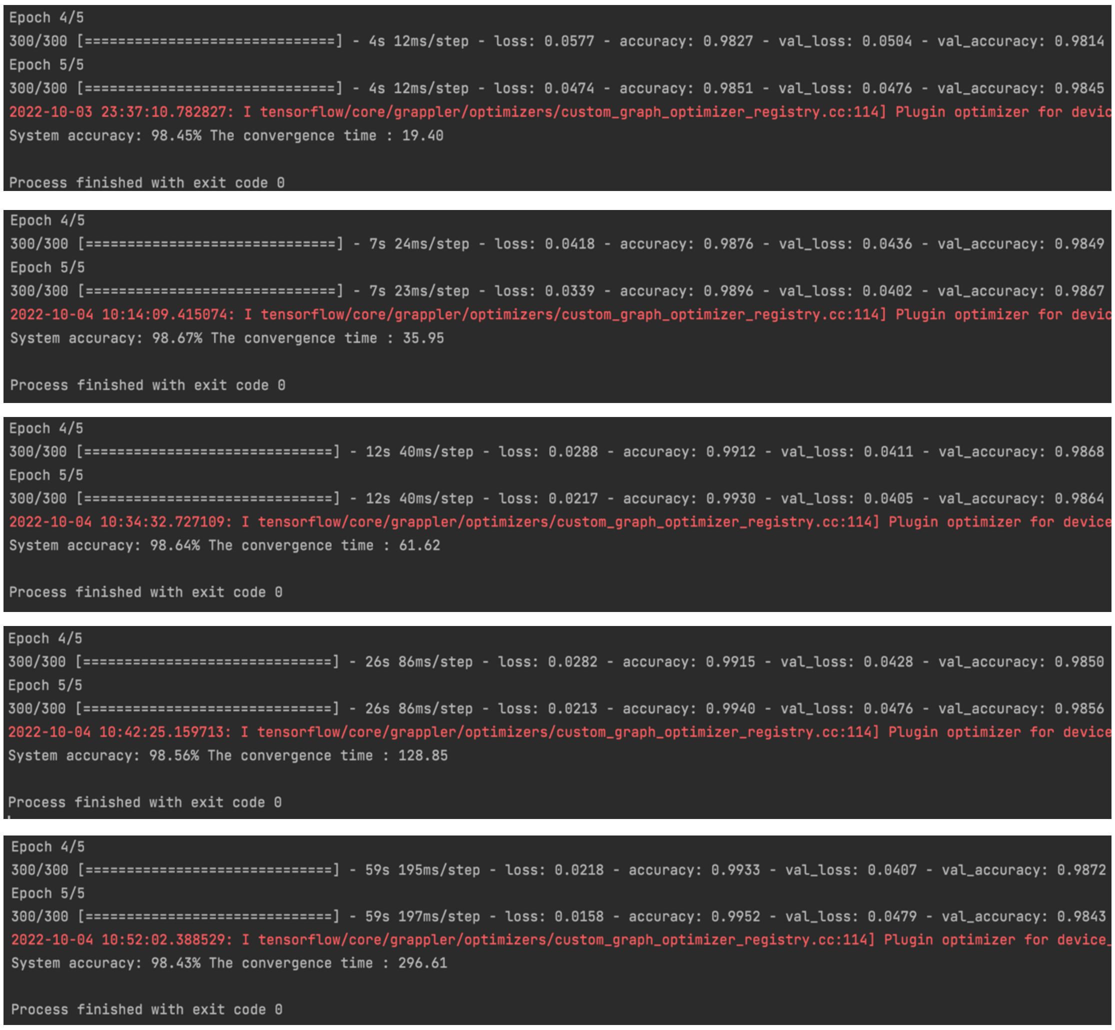
Figure 8. The results of modifying the feature map size

Next, the number of neurons on the hidden layer is modified to observe the change in accuracy and the time required to train a model. And 3x3 neurons are selected for the convolution kernel. The number of hidden layers is 16, 64, 128, 256 and 512. The results are shown in Table 6.

No of neurons	16	64	128	256	512
System accuracy	98.45%	98.67%	98.64%	98.56%	98.43%
convergence time	19.40 s	35.95 s	61.62 s	128.85 s	296.61

Table 6. The result of Modify the number of neurons on the dense hidden layer

The results of modifying the number of neurons on the dense hidden layer as shown in Figure 9.



```
Epoch 4/5
300/300 [==============================] - 4s 12ms/step - loss: 0.0577 - accuracy: 0.9827 - val_loss: 0.0504 - val_accuracy: 0.9814
Epoch 5/5
300/300 [==============================] - 4s 12ms/step - loss: 0.0474 - accuracy: 0.9851 - val_loss: 0.0476 - val_accuracy: 0.9845
2022-10-03 23:37:10.782827: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for devic
System accuracy: 98.45% The convergence time : 19.40

Process finished with exit code 0
```

```
Epoch 4/5
300/300 [==============================] - 7s 24ms/step - loss: 0.0418 - accuracy: 0.9876 - val_loss: 0.0436 - val_accuracy: 0.9849
Epoch 5/5
300/300 [==============================] - 7s 23ms/step - loss: 0.0339 - accuracy: 0.9896 - val_loss: 0.0402 - val_accuracy: 0.9867
2022-10-04 10:14:09.415074: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for devic
System accuracy: 98.67% The convergence time : 35.95

Process finished with exit code 0
```

```
Epoch 4/5
300/300 [==============================] - 12s 40ms/step - loss: 0.0288 - accuracy: 0.9912 - val_loss: 0.0411 - val_accuracy: 0.9868
Epoch 5/5
300/300 [==============================] - 12s 40ms/step - loss: 0.0217 - accuracy: 0.9930 - val_loss: 0.0405 - val_accuracy: 0.9864
2022-10-04 10:34:32.727109: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device
System accuracy: 98.64% The convergence time : 61.62

Process finished with exit code 0
```

```
Epoch 4/5
300/300 [==============================] - 26s 86ms/step - loss: 0.0282 - accuracy: 0.9915 - val_loss: 0.0428 - val_accuracy: 0.9850
Epoch 5/5
300/300 [==============================] - 26s 86ms/step - loss: 0.0213 - accuracy: 0.9940 - val_loss: 0.0476 - val_accuracy: 0.9856
2022-10-04 10:42:25.159713: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device
System accuracy: 98.56% The convergence time : 128.85

Process finished with exit code 0
```

```
Epoch 4/5
300/300 [==============================] - 59s 195ms/step - loss: 0.0218 - accuracy: 0.9933 - val_loss: 0.0407 - val_accuracy: 0.9872
Epoch 5/5
300/300 [==============================] - 59s 197ms/step - loss: 0.0158 - accuracy: 0.9952 - val_loss: 0.0479 - val_accuracy: 0.9843
2022-10-04 10:52:02.388529: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device_
System accuracy: 98.43% The convergence time : 296.61

Process finished with exit code 0
```

Figure 9. The results of modifying the number of neurons on the dense hidden layer

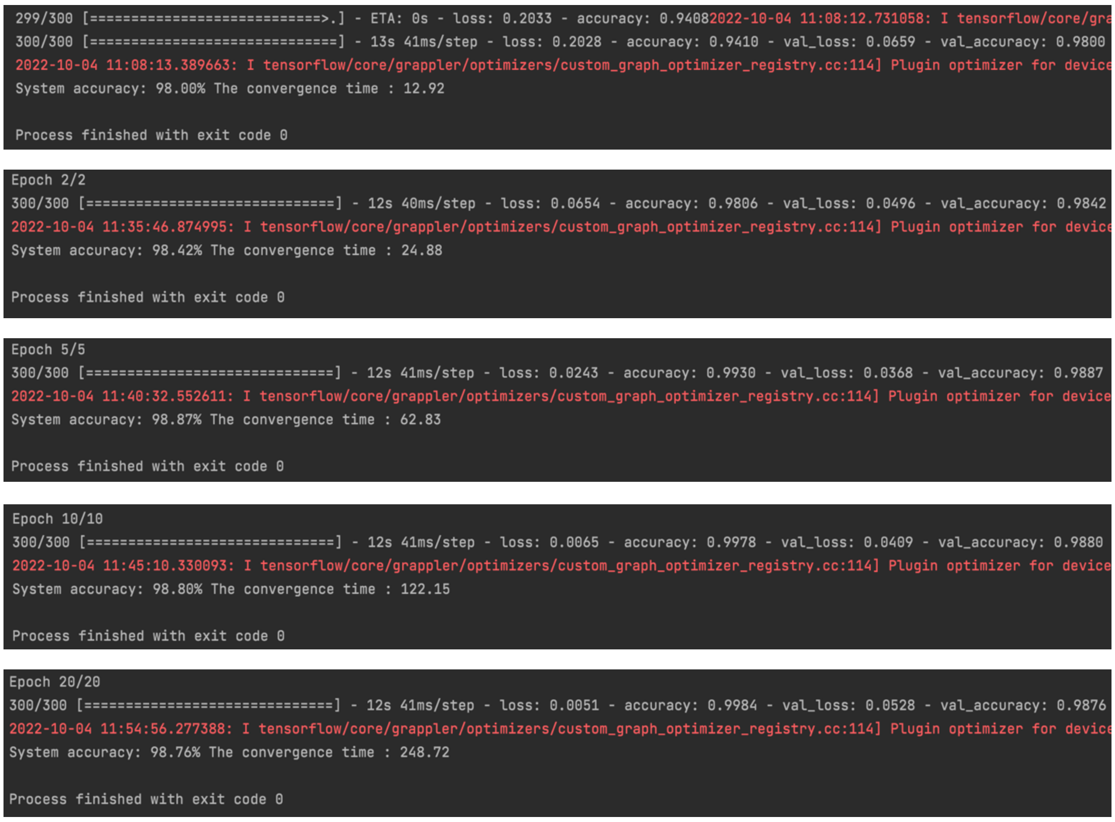
As shown by the results of modifying the number of neurons on the dense hidden layer, we can find that by modifying the number of hidden layers of neurons, the accuracy of the system does not change much, but the convergence time increases with the increase of the size of the number of hidden layers on the modified dense layer.

Next, the number of epochs used to train the model is modified as specified in Table 6. The value of 128 neurons in the dense hidden layer was chosen. In this way, the change in the accuracy and convergence time of the system is observed, and the results are shown in Table 7.

No of epochs	1	2	5	10	20
System accuracy	98.00%	98.42%	98.87%	98.88%	98.76%
Convergence time	12.92 s	24.88 s	62.83 s	122.15 s	248.72 s

Table 7. The result of modifying the number of epochs of the training model

The results by modifying the number of epochs for the training model are shown in Figure 10.



```
299/300 [============================>.] - ETA: 0s - loss: 0.2033 - accuracy: 0.94082022-10-04 11:08:12.731058: I tensorflow/core/gra
300/300 [==============================] - 13s 41ms/step - loss: 0.2028 - accuracy: 0.9410 - val_loss: 0.0659 - val_accuracy: 0.9800
2022-10-04 11:08:13.389663: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device
System accuracy: 98.00% The convergence time : 12.92

Process finished with exit code 0
```

```
Epoch 2/2
300/300 [==============================] - 12s 40ms/step - loss: 0.0654 - accuracy: 0.9806 - val_loss: 0.0496 - val_accuracy: 0.9842
2022-10-04 11:35:46.874995: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device
System accuracy: 98.42% The convergence time : 24.88

Process finished with exit code 0
```

```
Epoch 5/5
300/300 [==============================] - 12s 41ms/step - loss: 0.0243 - accuracy: 0.9930 - val_loss: 0.0368 - val_accuracy: 0.9887
2022-10-04 11:40:32.552611: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device
System accuracy: 98.87% The convergence time : 62.83

Process finished with exit code 0
```

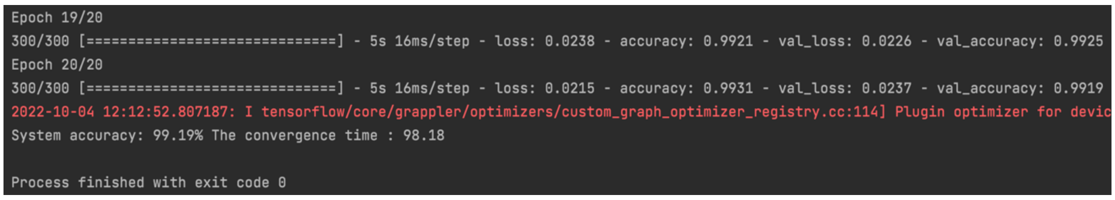
```
Epoch 10/10
300/300 [==============================] - 12s 41ms/step - loss: 0.0065 - accuracy: 0.9978 - val_loss: 0.0409 - val_accuracy: 0.9880
2022-10-04 11:45:10.330093: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device
System accuracy: 98.80% The convergence time : 122.15

Process finished with exit code 0
```

```
Epoch 20/20
300/300 [==============================] - 12s 41ms/step - loss: 0.0051 - accuracy: 0.9984 - val_loss: 0.0528 - val_accuracy: 0.9876
2022-10-04 11:54:56.277388: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for device
System accuracy: 98.76% The convergence time : 248.72

Process finished with exit code 0
```

Figure 10. The result of modifying the number of epochs of the training model

As we can see from Table 7 and Figure 9 when we go to modify the number of epochs, the accuracy of the system increases very slowly with the increase of epochs, but the change is not very large, but at the same time the convergence time increases substantially.

Next, the CNN model is modified by adding a convolutional layer, a maximum set layer and a fully connected layer to evaluate the accuracy of the CNN model, and the results are shown in Figure 11.



```
Epoch 19/20
300/300 [==============================] - 5s 16ms/step - loss: 0.0238 - accuracy: 0.9921 - val_loss: 0.0226 - val_accuracy: 0.9925
Epoch 20/20
300/300 [==============================] - 5s 16ms/step - loss: 0.0215 - accuracy: 0.9931 - val_loss: 0.0237 - val_accuracy: 0.9919
2022-10-04 12:12:52.807187: I tensorflow/core/grappler/optimizers/custom_graph_optimizer_registry.cc:114] Plugin optimizer for devic
System accuracy: 99.19% The convergence time : 98.18

Process finished with exit code 0
```

Figure 11. The result of the CNN model

After modifying the CNN model by adding the convolutional layer, the maximum set layer and the fully connected layer, we can obviously find that the accuracy of the system increases to 99.19% and the convergence time is 98.18. When we are classifying handwritten digits, we can consider modifying the CNN model and adjusting the convolutional layer to improve the accuracy of the classification. However, we also need to pay attention to the problem of overfitting.

Appendix

Code for Application 1

```
import keras
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras import layers
import time

def baseline_model(num_pixels, num_classes):
    model = keras.models.Sequential()
    model.add(layers.Dense(8, input_dim=num_pixels, kernel_initializer='normal',
activation='relu'))
    model.add(layers.Dense(num_classes,kernel_initializer='normal',activation='soft
max'))
    model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics =
['accuracy'])
    return model

def trainAndPredictMLP(X_train, Y_train, X_test, Y_test):
    num_pixels = X_train.shape[1] * X_train.shape[2]
    X_train = X_train.reshape((X_train.shape[0], num_pixels)).astype('float32')
    X_test = X_test.reshape((X_test.shape[0],num_pixels)).astype('float32')
    X_train = X_train / 255
    X_test = X_test / 255
    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]
    model = baseline_model(num_pixels, num_classes)
    test_array = []
    time_1 = []
    for i in range(10):
        time_begin = time.time()
        model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10,
                batch_size=200, verbose=2)
        time_end = time.time()
        scores = model.evaluate(X_test, Y_test, verbose=0)
        test_array.append(scores[1])
        time_1 = time_end - time_begin
        #print("Time costing = {}s".format(time_1))
    average_scores = np.average(test_array)*100
```

```
        average_time = np.average(time_1)
        print("The average of test accuracy is {:.2f}%".format(average_scores),"The average time of training   is {}".format(average_time))
#TODO - Application 1 - Step 8 - System evaluation - compute and display the prediction error
        #model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10, batch_size=200, verbose=2)
        #print("Baseline Error: {:.2f}".format(100 - scores[1] * 100))
        return

def main():
        #TODO - Application 1 - Step 1 - Load the MNIST dataset in Keras
        (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
        print(X_train.shape)
       #TODO - Application 1 - Step 2 - Train and predict on a MLP - Call the trainAndPredictMLP function
       trainAndPredictMLP(X_train, Y_train, X_test, Y_test)

       return


if __name__ == '__main__':
        main()
```

Comment: Load the MNINT dataset in Keras and use a baseline model with a multi-layer perceptron. The images are reduced to vectors by reshaping the MINT dataset, and the input values are then normalized. After converting the category labels into binary matrices, the model architecture is built.

Code for Exercise 1

```
def baseline_model(num_pixels, num_classes):
        model = keras.models.Sequential()
        model.add(layers.Dense(8, input_dim=num_pixels, kernel_initializer='normal', activation='relu'))   # exercise 1
       model.add(layers.Dense(num_classes,kernel_initializer='normal',activation='softmax'))
       model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy'])
        return model
```

Comment: The performance of the system is evaluated by modifying different numbers of neurons on the hidden layers. The number of hidden layer modifications are 8, 16, 32, 64 and 128.

Code for Exercise 2

```
test_array = []
    time_1 = []
    for i in range(10):
        time_begin = time.time()
        model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10,
              batch_size=128, verbose=2)     # exercise 2
        time_end = time.time()
        scores = model.evaluate(X_test, Y_test, verbose=0)
        test_array.append(scores[1])
        time_1 = time_end - time_begin
        #print("Time costing = {}s".format(time_1))
    average_scores = np.average(test_array)*100
    average_time = np.average(time_1)
    print("The average of test accuracy is {:.2f}%".format(average_scores),"The
average time of training    is {}".format(average_time))
```

Comment: The performance of the system is evaluated by modifying different values of the batch size. The number of hidden layer modifications are 32, 64, 128, 256 and 512.

Code for Exercise 3

```
#exercise3- - - Modify different numbers of neurons on the hidden layer
def baseline_model(num_pixels, num_classes):
    model = keras.models.Sequential()
    model.add(layers.Dense(16, input_dim=num_pixels,
                               kernel_initializer='normal', activation='relu'))
    model.add(layers.Dense(num_classes,kernel_initializer='normal',activation='soft
max'))
    model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics =
['mse'])     # exercise 3
    return model


# exercise 3 - - - Modify different values of batch size
test_array = []
    time_1 = []
    for i in range(10):
        time_begin = time.time()
        model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=10,
              batch_size=32, verbose=2)    # exercise 3
        time_end = time.time()
        scores = model.evaluate(X_test, Y_test, verbose=0)
        test_array.append(scores[1])
        time_1 = time_end - time_begin
        #print("Time costing = {}s".format(time_1))
    average_scores = np.average(test_array)
    average_time = np.average(time_1)
    print("The average of MSE is {:.6f}".format(average_scores),"The average time
of training   is {}".format(average_time))
```

Comment: Compare the results based on changing the number of neurons in the hidden layer and modifying the batch size of different values to compare the impact of MSE evaluation on the system.

Code for Exercise 5

```
import keras
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras import layers

def baseline_model(num_pixels, num_classes):
    model = keras.models.Sequential()
        model.add(layers.Dense(8,
    input_dim=num_pixels,kernel_initializer='normal',activation='relu'))
    model.add(layers.Dense(num_classes,kernel_initializer='normal',activation='soft
    max'))
    model.compile(loss='categorical_crossentropy', optimizer = 'adam', metrics =
['accuracy'])

    return model

def trainAndPredictMLP(X_train, Y_train, X_test, Y_test):
    num_pixels = X_train.shape[1] * X_train.shape[2]
    X_train = X_train.reshape((X_train.shape[0], num_pixels)).astype('float32')
    X_test = X_test.reshape((X_test.shape[0],num_pixels)).astype('float32')
    X_train = X_train / 255
    X_test = X_test / 255
    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]
    model = baseline_model(num_pixels, num_classes)
    # Exercise 5
    model.load_weights('./weights')
    CorrectPredicted = 0
    for idx, i in enumerate(X_test[:5]):
        p = i.reshape(1, 784)
        PredictLabel = np.argmax(model.predict(p))
        if PredictLabel == np.argmax(Y_test[idx]):
            CorrectPredicted = CorrectPredicted +1
        accuracy = CorrectPredicted / 5 *100
        print("Accuracy for test = {}%".format(accuracy))
    return

def main():
    (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
    print(X_train.shape)
    trainAndPredictMLP(X_train, Y_train, X_test, Y_test)
    return

if __name__ == '__main__':
    main()
```

Comment: The first five images of the testing set were predicted by writing a new script to load the weights file in exercise 4. In particular, it is important to note that the reshaping of the predicted images is very important.

Code for Application 2

```
import keras
import numpy as np
from keras.datasets import mnist
from keras.utils import np_utils
from keras import layers
import time
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Activation, Dropout

def CNN_model(input_shape, num_classes):
    model = keras.models.Sequential()
    model.add(Conv2D(8,    (3,    3),activation='relu',    input_shape=(input_shape,
input_shape, 1)))
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam',                loss='categorical_crossentropy',
metrics=['accuracy'])
    return model

def trainAndPredictCNN(X_train, Y_train, X_test, Y_test):
    X_train = X_train.reshape(60000, 28, 28)
    X_text = X_test.reshape(10000, 28, 28)
    X_train = X_train / 255
    X_test = X_test / 255
    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]
    print(num_classes)
    input_shape = 28
    model = CNN_model(input_shape, num_classes)
    model.fit(X_train,   Y_train,   validation_data=(X_test,   Y_test),   epochs=5,
batch_size=200)
    scores = model.evaluate(X_test, Y_test, verbose=0)
    print("Prediction Error: {:.2f}%".format(100 - scores[1] * 100),"System accuracy:
{:.2f}%".format(scores[1] * 100))
    return

def main():
    (X_train, Y_train), (X_test, Y_test) = mnist.load_data()
    print(X_train.shape)
    trainAndPredictCNN(X_train, Y_train, X_test, Y_test)
    return

if __name__ == '__main__':
    main()
```

Comment: Build the CNN model: first reshape the MINST dataset, then normalize the dataset and transform the category labels into a binary matrix. Finally, the CNN model architecture is built.

Code for exercise 6 & 7

```
def CNN_model(input_shape, num_classes):
    model = keras.models.Sequential()
    model.add(Conv2D(8, (3, 3),activation='relu', input_shape=(input_shape,
input_shape, 1)))  # Exercise 6 & 7
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    return model
```

Comment: The size of the feature map within the convolutional layer and the number of neurons on the hidden layer is modified to observe changes in accuracy and changes in the time required to train the model.

Code for exercise 8

```
def trainAndPredictCNN(X_train, Y_train, X_test, Y_test):
    X_train = X_train.reshape(60000, 28, 28)
    X_text = X_test.reshape(10000, 28, 28)
    X_train = X_train / 255
    X_test = X_test / 255
    Y_train = np_utils.to_categorical(Y_train)
    Y_test = np_utils.to_categorical(Y_test)
    num_classes = Y_test.shape[1]
    print(num_classes)
    input_shape = 28
    model = CNN_model(input_shape, num_classes)
    model.fit(X_train, Y_train, validation_data=(X_test, Y_test), epochs=5,
batch_size=200)  # Exercise 8
    scores = model.evaluate(X_test, Y_test, verbose=0)
    print("Prediction Error: {:.2f}%".format(100 - scores[1] * 100),"System accuracy:
{:.2f}%".format(scores[1] * 100))
    return
```

Comment: The change in accuracy and convergence time of the system is observed by modifying the number of epochs used to train the model.

Code for exercise 9

```
def CNN_model(input_shape, num_classes):
    model = keras.models.Sequential()
    # Exercise 9
    model.add(Conv2D(30, (5, 5),activation='relu', input_shape=(input_shape, input_shape, 1)))
    model.add(MaxPooling2D(2, 2))
    model.add(Conv2D(15, (3, 3), activation='relu'))
    model.add(MaxPooling2D(2, 2))
    model.add(Dropout(0.2))
    model.add(Flatten())
    model.add(Dense(50, activation='relu'))
    model.add(Dense(10, activation='softmax'))
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

Comment: The accuracy of the CNN model was evaluated by modifying the CNN model by adding a convolutional layer, a maximum set layer, and a fully connected layer.