Exercise

1. Change the number of neurons in the hidden layer.

Number of neurons (Hidden layer):

1. 64 neurons

```
model = keras . Sequential ([
keras . layers . Dense (64 , activation ="relu", input_shape =(784 ,) ) ,
keras . layers . Dense (10 , activation ="softmax")
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/core/dense.py:93: UserWarning: Do not pass an `input_shape`/`input_
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
[13] model . compile ( optimizer ="adam",
     loss ="sparse_categorical_crossentropy",
     metrics =["accuracy"])
```

```
model . fit ( x_train , y_train , epochs =5 , batch_size =32)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 2ms/step - accuracy: 0.8589 - loss: 0.4988
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9572 - loss: 0.1451
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9697 - loss: 0.1031
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 4s 2ms/step - accuracy: 0.9746 - loss: 0.0830
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9808 - loss: 0.0622
<keras.src.callbacks.history.History at 0x7df50c1295b0>
```

```
[15] test_loss , test_acc = model . evaluate ( x_test , y_test )
     print ("Test accuracy :", test_acc )
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9680 - loss: 0.1095
Test accuracy : 0.9725000262260437
```

2. 128 neurons

```
model = keras . Sequential ([
keras . layers . Dense (128 , activation ="relu", input_shape =(784 ,) ) ,
keras . layers . Dense (10 , activation ="softmax")
])
```

```
[19] model . compile ( optimizer ="adam",
     loss ="sparse_categorical_crossentropy",
     metrics =["accuracy"])
```

```
[20] model . fit ( x_train , y_train , epochs =5 , batch_size =32)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8778 - loss: 0.4297
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9661 - loss: 0.1144
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9769 - loss: 0.0784
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9840 - loss: 0.0527
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9889 - loss: 0.0373
<keras.src.callbacks.history.History at 0x7df508f9d520>
```

```
[21] test_loss , test_acc = model . evaluate ( x_test , y_test )
     print ("Test accuracy :", test_acc )
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9713 - loss: 0.0949
Test accuracy : 0.9743000268936157
```

3. 256 neurons

```
model = keras . Sequential ([
keras . layers . Dense (256 , activation ="relu", input_shape =(784 ,) ) ,
keras . layers . Dense (10 , activation ="softmax")
])
```

```
[25] model . compile ( optimizer ="adam",
     loss ="sparse_categorical_crossentropy",
     metrics =["accuracy"])
```

```
[26] model . fit ( x_train , y_train , epochs =5 , batch_size =32)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.8872 - loss: 0.3855
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 5ms/step - accuracy: 0.9727 - loss: 0.0954
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.9814 - loss: 0.0587
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 4ms/step - accuracy: 0.9887 - loss: 0.0401
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 5ms/step - accuracy: 0.9911 - loss: 0.0294
<keras.src.callbacks.history.History at 0x7df508ebd520>
```

```
[27] test_loss , test_acc = model . evaluate ( x_test , y_test )
     print ("Test accuracy :", test_acc )
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9759 - loss: 0.0799
Test accuracy : 0.9799000024795532
```

When we increase the number of neurons in the hidden layer, the neural network takes longer to train because each neuron must process all inputs and perform more calculations. However, the accuracy usually improves because more neurons allow the network to recognise more subtle patterns in the images. This trade-off indicates that while a larger network can learn more effectively, it also requires more computational resources and training time. Beyond a certain point, adding too many neurons may not significantly improve accuracy and could lead to overfitting.

2. Change the activation function from ReLU to sigmoid

1. Using ReLU:

```
model = keras . Sequential ([
keras . layers . Dense (128 , activation ="relu", input_shape =(784 ,) ) ,
keras . layers . Dense (10 , activation ="softmax")
])
```

```
[19] model . compile ( optimizer ="adam",
     loss ="sparse_categorical_crossentropy",
     metrics =["accuracy"])
```

```
[20] model . fit ( x_train , y_train , epochs =5 , batch_size =32)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8778 - loss: 0.4297
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9661 - loss: 0.1144
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9769 - loss: 0.0784
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9840 - loss: 0.0527
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9889 - loss: 0.0373
<keras.src.callbacks.history.History at 0x7df508f9d520>
```

```
[21] test_loss , test_acc = model . evaluate ( x_test , y_test )
     print ("Test accuracy :", test_acc )
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9713 - loss: 0.0949
Test accuracy : 0.9743000268936157
```

2. Using Sigmod:

```
model = keras . Sequential ([
keras . layers . Dense (128 , activation ="sigmoid", input_shape =(784 ,) ) ,
keras . layers . Dense (10 , activation ="softmax")
])
```

```
[31] model . compile ( optimizer ="adam",
     loss ="sparse_categorical_crossentropy",
     metrics =["accuracy"])
```

```
[32] model . fit ( x_train , y_train , epochs =5 , batch_size =32)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8410 - loss: 0.6461
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9404 - loss: 0.2060
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9585 - loss: 0.1446
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9674 - loss: 0.1117
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9749 - loss: 0.0888
<keras.src.callbacks.history.History at 0x7df508ceb5c0>
```

```
[33] test_loss , test_acc = model . evaluate ( x_test , y_test )
     print ("Test accuracy :", test_acc )
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9670 - loss: 0.1084
Test accuracy : 0.9711999893188477
```

Changing the activation function from ReLU to sigmoid makes the neurons output values between 0 and 1 instead of passing positive numbers directly. This usually slows down training and can slightly reduce accuracy because the network learns more cautiously. For small networks like this MNIST example, the difference isn't huge, but ReLU is generally faster and more effective for hidden layers.

3. Train for 1, 5, and 10 epochs. Compare results.

1. 1 epochs

```
model . fit ( x_train , y_train , epochs =1 , batch_size =32)
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.8371 - loss: 0.6660
<keras.src.callbacks.history.History at 0x7df5089ff590>
```

```
[39] test_loss , test_acc = model . evaluate ( x_test , y_test )
     print ("Test accuracy :", test_acc )
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9220 - loss: 0.2650
Test accuracy : 0.9322999715805054
```

2. 5 epochs

```
model . fit ( x_train , y_train , epochs =5 , batch_size =32)
```

```
Epoch 1/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.8334 - loss: 0.6680
Epoch 2/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9409 - loss: 0.2054
Epoch 3/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 9s 3ms/step - accuracy: 0.9575 - loss: 0.1486
Epoch 4/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9672 - loss: 0.1137
Epoch 5/5
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9740 - loss: 0.0890
<keras.src.callbacks.history.History at 0x7df5089df2f0>
```

```
[45] test_loss , test_acc = model . evaluate ( x_test , y_test )
     print ("Test accuracy :", test_acc )
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9663 - loss: 0.1158
Test accuracy : 0.970300018787384
```

3. 10 epochs

```
model . fit ( x_train , y_train , epochs =10 , batch_size =32)
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 8s 4ms/step - accuracy: 0.8389 - loss: 0.6545
Epoch 2/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9416 - loss: 0.2063
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9562 - loss: 0.1506
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9678 - loss: 0.1110
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9758 - loss: 0.0874
Epoch 6/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9792 - loss: 0.0716
Epoch 7/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 6s 3ms/step - accuracy: 0.9836 - loss: 0.0598
Epoch 8/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9867 - loss: 0.0502
Epoch 9/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.9883 - loss: 0.0413
Epoch 10/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 10s 4ms/step - accuracy: 0.9910 - loss: 0.0368
<keras.src.callbacks.history.History at 0x7df4ff7d9a90>
```

```
[51] test_loss , test_acc = model . evaluate ( x_test , y_test )
     print ("Test accuracy :", test_acc )
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9705 - loss: 0.0898
Test accuracy : 0.9753000140190125
```

Training for more epochs allows the neural network to see the images multiple times, which helps it learn better. With 1 epoch, accuracy is low because the network has only seen the data once. With 5 epochs, accuracy improves significantly as the network has learned most patterns. By 10 epochs, accuracy improves slightly more, but the network takes longer to train, and further increases may not help much.

Reflection

1. Why do smaller models train faster but may have lower accuracy?

Smaller models have fewer neurons and connections, so they do fewer calculations and can learn patterns quickly. However, they have less capacity to understand complex patterns, so their guesses may be less accurate. It's like a small brain: fast, but not as smart.

2. Why does accuracy sometimes stop improving after many epochs?

After a certain point, the network has already learned most of the patterns in the training data. Seeing the data more times doesn't teach it much more. If you train too long, it can even start memorizing the training images (overfitting), which may not help with new, unseen images.

3. How do these trade-offs matter for microcontrollers?

- Microcontrollers have limited memory and processing power.
- A big network may give higher accuracy but could be too slow or use too much memory.
- A small network trains and runs fast but may not be accurate enough.
- You have to balance size, speed, and accuracy to make it work efficiently on a microcontroller.