HW2 Design a neural network for handwriting recognition

N26130697 尤時雨 2025.4

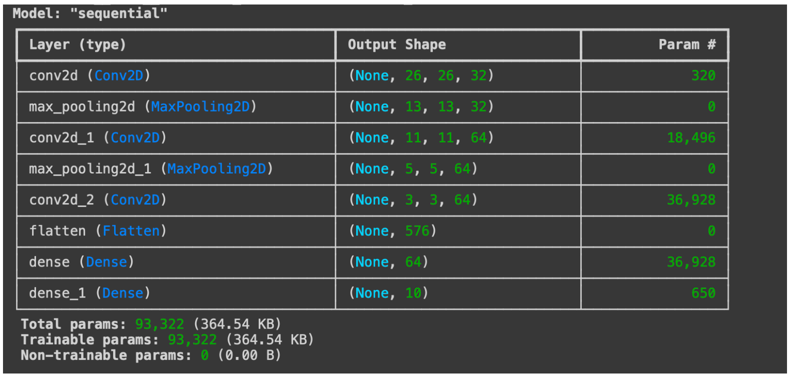
1.Model Description

(1)Which parameters (learning_rate, training_steps, batch_size) you change? How it affect your result?

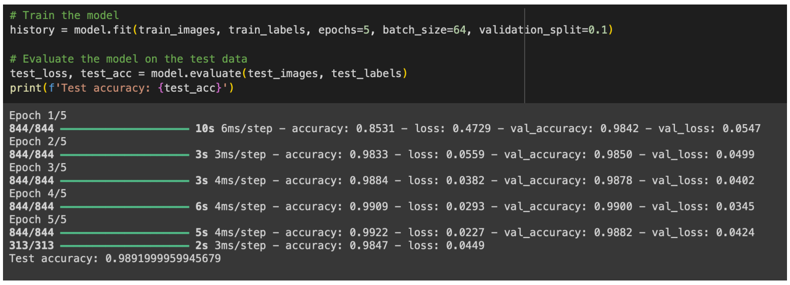
A:

[0-1]basic model (learning_rate=0.0001, epochs=5, batch_size=64)

-> Accuracy = 0.9892



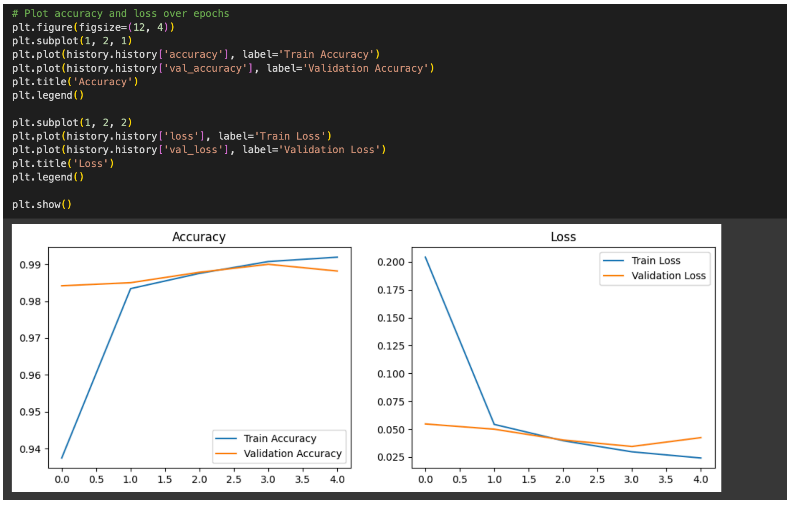
```
Model: "sequential"
┌─────────────────────────────────┬────────────────────────┬───────────────┐
│ Layer (type)                    │ Output Shape           │       Param # │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d (Conv2D)                 │ (None, 26, 26, 32)     │           320 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d (MaxPooling2D)    │ (None, 13, 13, 32)     │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_1 (Conv2D)               │ (None, 11, 11, 64)     │        18,496 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ max_pooling2d_1 (MaxPooling2D)  │ (None, 5, 5, 64)       │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ conv2d_2 (Conv2D)               │ (None, 3, 3, 64)       │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ flatten (Flatten)               │ (None, 576)            │             0 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense (Dense)                   │ (None, 64)             │        36,928 │
├─────────────────────────────────┼────────────────────────┼───────────────┤
│ dense_1 (Dense)                 │ (None, 10)             │           650 │
└─────────────────────────────────┴────────────────────────┴───────────────┘
 Total params: 93,322 (364.54 KB)
 Trainable params: 93,322 (364.54 KB)
 Non-trainable params: 0 (0.00 B)
```



```
# Train the model
history = model.fit(train_images, train_labels, epochs=5, batch_size=64, validation_split=0.1)

# Evaluate the model on the test data
test_loss, test_acc = model.evaluate(test_images, test_labels)
print(f'Test accuracy: {test_acc}')
```

```
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 6ms/step - accuracy: 0.8531 - loss: 0.4729 - val_accuracy: 0.9842 - val_loss: 0.0547
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9833 - loss: 0.0559 - val_accuracy: 0.9850 - val_loss: 0.0499
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9884 - loss: 0.0382 - val_accuracy: 0.9878 - val_loss: 0.0402
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9909 - loss: 0.0293 - val_accuracy: 0.9900 - val_loss: 0.0345
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9922 - loss: 0.0227 - val_accuracy: 0.9882 - val_loss: 0.0424
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9847 - loss: 0.0449
Test accuracy: 0.9891999959945679
```



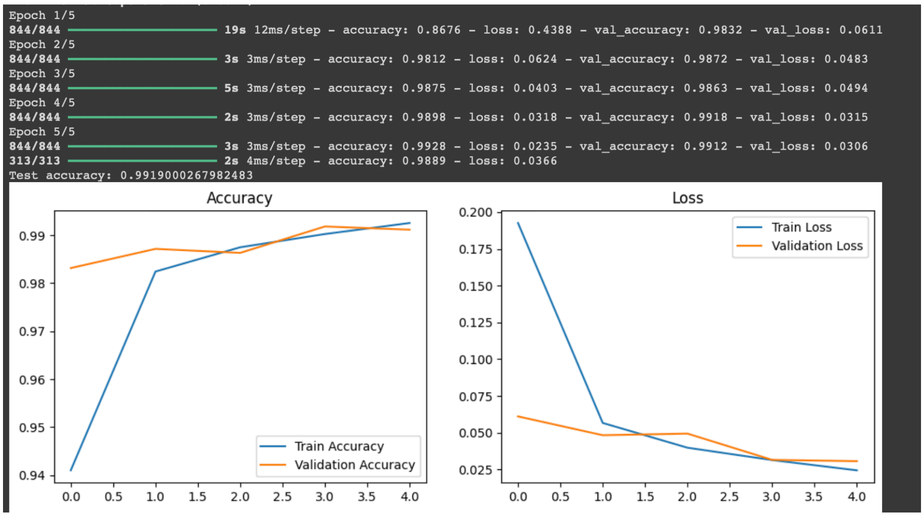
```
# Plot accuracy and loss over epochs
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.legend()

plt.show()
```

[1-1]Adjust learning_rate (learning_rate=0.001, epochs=5, batch_size=64)

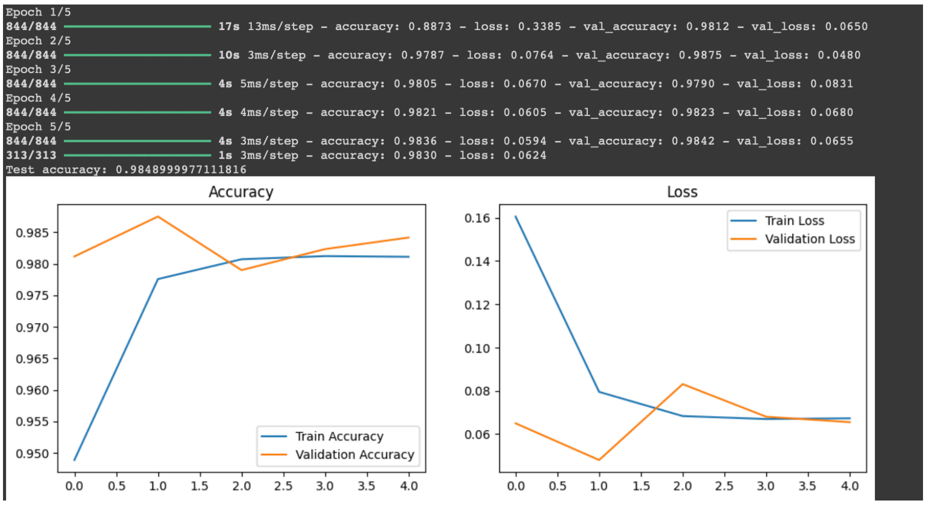
-> Accuracy = 0.9919



```
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 19s 12ms/step - accuracy: 0.8676 - loss: 0.4388 - val_accuracy: 0.9832 - val_loss: 0.0611
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9812 - loss: 0.0624 - val_accuracy: 0.9872 - val_loss: 0.0483
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9875 - loss: 0.0403 - val_accuracy: 0.9863 - val_loss: 0.0494
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9898 - loss: 0.0318 - val_accuracy: 0.9918 - val_loss: 0.0315
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9928 - loss: 0.0235 - val_accuracy: 0.9912 - val_loss: 0.0306
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9889 - loss: 0.0366
Test accuracy: 0.9919000267982483
```

[1-2]Adjust learning_rate (learning_rate=0.01, epochs=5, batch_size=64)

-> Accuracy = 0.9849



```
Epoch 1/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 17s 13ms/step - accuracy: 0.8873 - loss: 0.3385 - val_accuracy: 0.9812 - val_loss: 0.0650
Epoch 2/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 10s 3ms/step - accuracy: 0.9787 - loss: 0.0764 - val_accuracy: 0.9875 - val_loss: 0.0480
Epoch 3/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9805 - loss: 0.0670 - val_accuracy: 0.9790 - val_loss: 0.0831
Epoch 4/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9821 - loss: 0.0605 - val_accuracy: 0.9823 - val_loss: 0.0680
Epoch 5/5
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9836 - loss: 0.0594 - val_accuracy: 0.9842 - val_loss: 0.0655
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9830 - loss: 0.0624
Test accuracy: 0.9848999977111816
```

—>Conclusion:

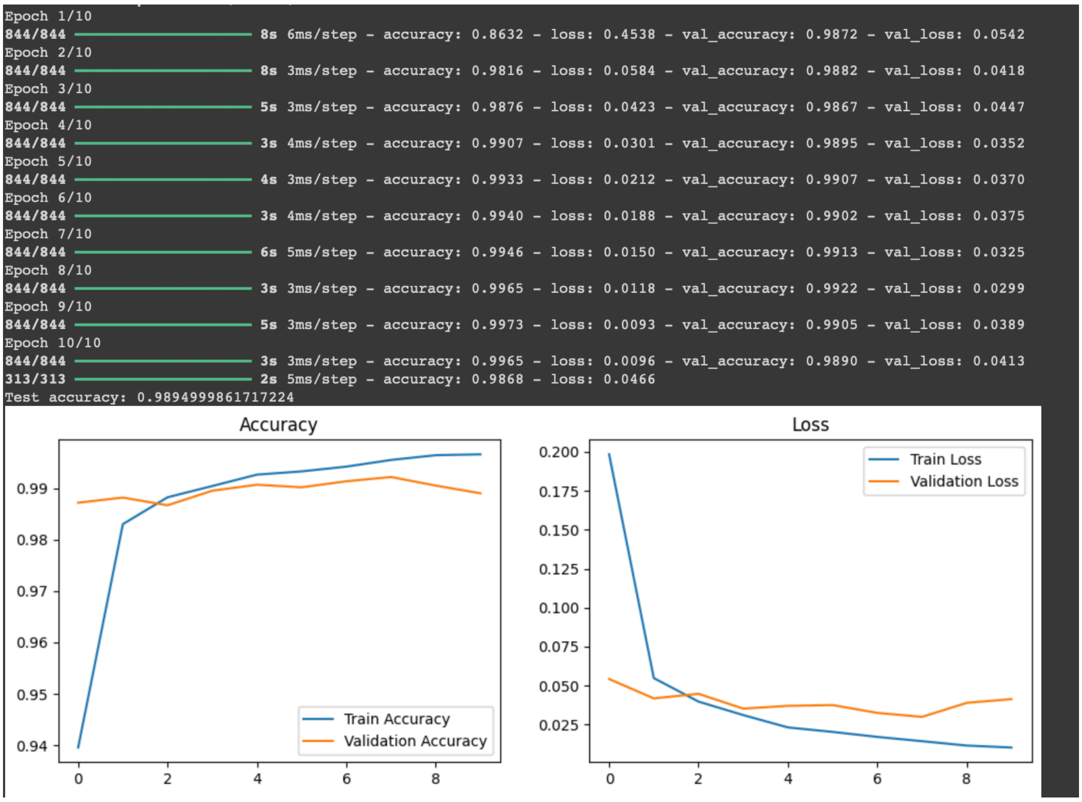
The learning rate of 0.001 provides the best accuracy (0.9919).

At 0.0001, the model converges more slowly, causing the accuracy to drop to 0.9892.

At 0.01, the learning rate is too high, causing the accuracy to drop to 0.9849.

[2-1]Adjust epochs to 10 (learning_rate=0.001, epochs=10, batch_size=64)

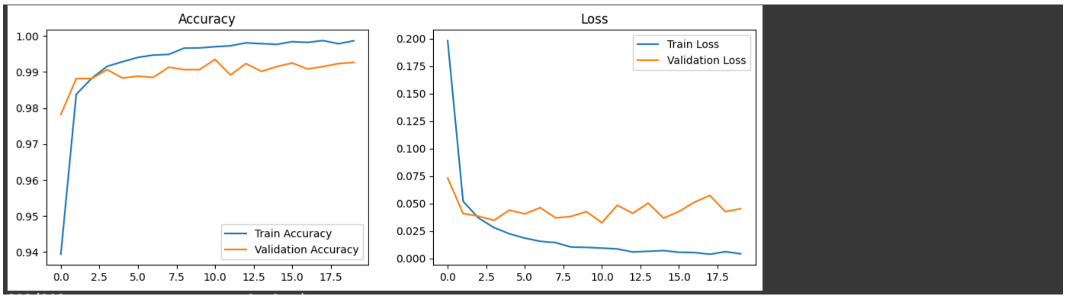
-> Accuracy = 0.9895

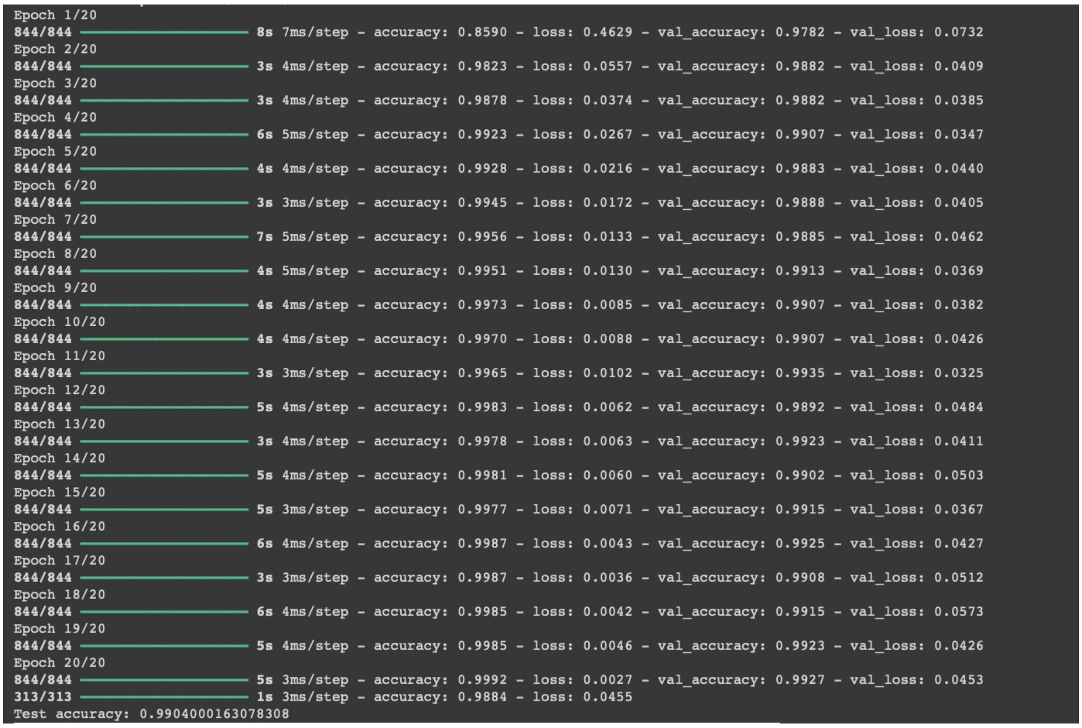


```
Epoch 1/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 8s 6ms/step - accuracy: 0.8632 - loss: 0.4538 - val_accuracy: 0.9872 - val_loss: 0.0542
Epoch 2/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 8s 3ms/step - accuracy: 0.9816 - loss: 0.0584 - val_accuracy: 0.9882 - val_loss: 0.0418
Epoch 3/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9876 - loss: 0.0423 - val_accuracy: 0.9867 - val_loss: 0.0447
Epoch 4/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9907 - loss: 0.0301 - val_accuracy: 0.9895 - val_loss: 0.0352
Epoch 5/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9933 - loss: 0.0212 - val_accuracy: 0.9907 - val_loss: 0.0370
Epoch 6/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9940 - loss: 0.0188 - val_accuracy: 0.9902 - val_loss: 0.0375
Epoch 7/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9946 - loss: 0.0150 - val_accuracy: 0.9913 - val_loss: 0.0325
Epoch 8/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9965 - loss: 0.0118 - val_accuracy: 0.9922 - val_loss: 0.0299
Epoch 9/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9973 - loss: 0.0093 - val_accuracy: 0.9905 - val_loss: 0.0389
Epoch 10/10
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9965 - loss: 0.0096 - val_accuracy: 0.9890 - val_loss: 0.0413
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9868 - loss: 0.0466
Test accuracy: 0.9894999861717224
```

[2-2]Adjust epochs to 20 (learning_rate=0.001, epochs=20, batch_size=64)

-> Accuracy = 0.9904





```
Epoch 1/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 8s 7ms/step - accuracy: 0.8590 - loss: 0.4629 - val_accuracy: 0.9782 - val_loss: 0.0732
Epoch 2/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9823 - loss: 0.0557 - val_accuracy: 0.9882 - val_loss: 0.0409
Epoch 3/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9878 - loss: 0.0374 - val_accuracy: 0.9882 - val_loss: 0.0385
Epoch 4/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 5ms/step - accuracy: 0.9923 - loss: 0.0267 - val_accuracy: 0.9907 - val_loss: 0.0347
Epoch 5/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9928 - loss: 0.0216 - val_accuracy: 0.9883 - val_loss: 0.0440
Epoch 6/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9945 - loss: 0.0172 - val_accuracy: 0.9888 - val_loss: 0.0405
Epoch 7/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 7s 5ms/step - accuracy: 0.9956 - loss: 0.0133 - val_accuracy: 0.9885 - val_loss: 0.0462
Epoch 8/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 5ms/step - accuracy: 0.9951 - loss: 0.0130 - val_accuracy: 0.9913 - val_loss: 0.0369
Epoch 9/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9973 - loss: 0.0085 - val_accuracy: 0.9907 - val_loss: 0.0382
Epoch 10/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9970 - loss: 0.0088 - val_accuracy: 0.9907 - val_loss: 0.0426
Epoch 11/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9965 - loss: 0.0102 - val_accuracy: 0.9935 - val_loss: 0.0325
Epoch 12/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9983 - loss: 0.0062 - val_accuracy: 0.9892 - val_loss: 0.0484
Epoch 13/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9978 - loss: 0.0063 - val_accuracy: 0.9923 - val_loss: 0.0411
Epoch 14/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9981 - loss: 0.0060 - val_accuracy: 0.9902 - val_loss: 0.0503
Epoch 15/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9977 - loss: 0.0071 - val_accuracy: 0.9915 - val_loss: 0.0367
Epoch 16/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9987 - loss: 0.0043 - val_accuracy: 0.9925 - val_loss: 0.0427
Epoch 17/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9987 - loss: 0.0036 - val_accuracy: 0.9908 - val_loss: 0.0512
Epoch 18/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9985 - loss: 0.0042 - val_accuracy: 0.9915 - val_loss: 0.0573
Epoch 19/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9985 - loss: 0.0046 - val_accuracy: 0.9923 - val_loss: 0.0426
Epoch 20/20
844/844 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9992 - loss: 0.0027 - val_accuracy: 0.9927 - val_loss: 0.0453
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9884 - loss: 0.0455
Test accuracy: 0.9904000163078308
```

—>Conclusion:

Epochs = 5 -> accuracy = 0.9892

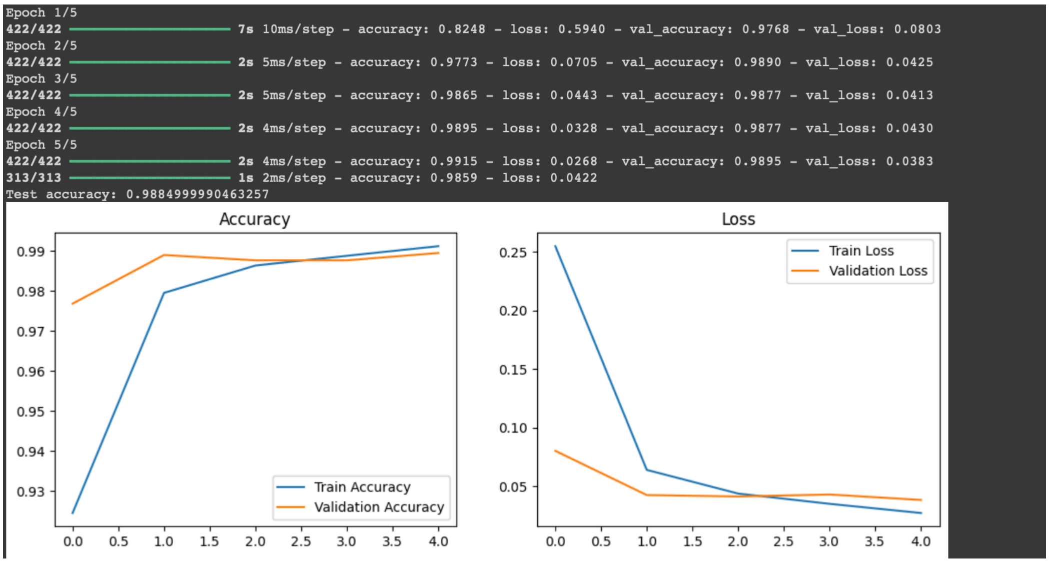
Epochs = 10 -> accuracy = 0.9895

Epochs = 20 -> accuracy = 0.9904

With the increment of the epochs, the accuracy became higher.

[3-1]Adjust batch_size to 128(learning_rate=0.001, epochs=5, batch_size=128)

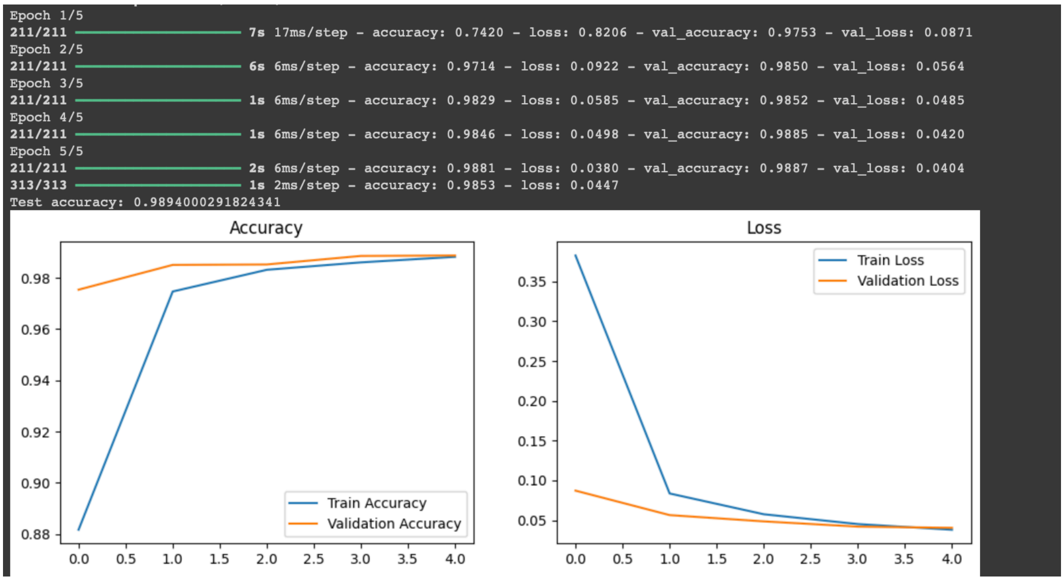
-> Accuracy = 0.9884



```
Epoch 1/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 7s 10ms/step - accuracy: 0.8248 - loss: 0.5940 - val_accuracy: 0.9768 - val_loss: 0.0803
Epoch 2/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9773 - loss: 0.0705 - val_accuracy: 0.9890 - val_loss: 0.0425
Epoch 3/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9865 - loss: 0.0443 - val_accuracy: 0.9877 - val_loss: 0.0413
Epoch 4/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9895 - loss: 0.0328 - val_accuracy: 0.9877 - val_loss: 0.0430
Epoch 5/5
422/422 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - accuracy: 0.9915 - loss: 0.0268 - val_accuracy: 0.9895 - val_loss: 0.0383
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9859 - loss: 0.0422
Test accuracy: 0.9884999990463257
```

[3-2]Adjust batch_size to 256(learning_rate=0.001, epochs=5, batch_size=256)

-> Accuracy = 0.9894



```
Epoch 1/5
211/211 ━━━━━━━━━━━━━━━━━━━━ 7s 17ms/step - accuracy: 0.7420 - loss: 0.8206 - val_accuracy: 0.9753 - val_loss: 0.0871
Epoch 2/5
211/211 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.9714 - loss: 0.0922 - val_accuracy: 0.9850 - val_loss: 0.0564
Epoch 3/5
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9829 - loss: 0.0585 - val_accuracy: 0.9852 - val_loss: 0.0485
Epoch 4/5
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9846 - loss: 0.0498 - val_accuracy: 0.9885 - val_loss: 0.0420
Epoch 5/5
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9881 - loss: 0.0380 - val_accuracy: 0.9887 - val_loss: 0.0404
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9853 - loss: 0.0447
Test accuracy: 0.9894000291824341
```

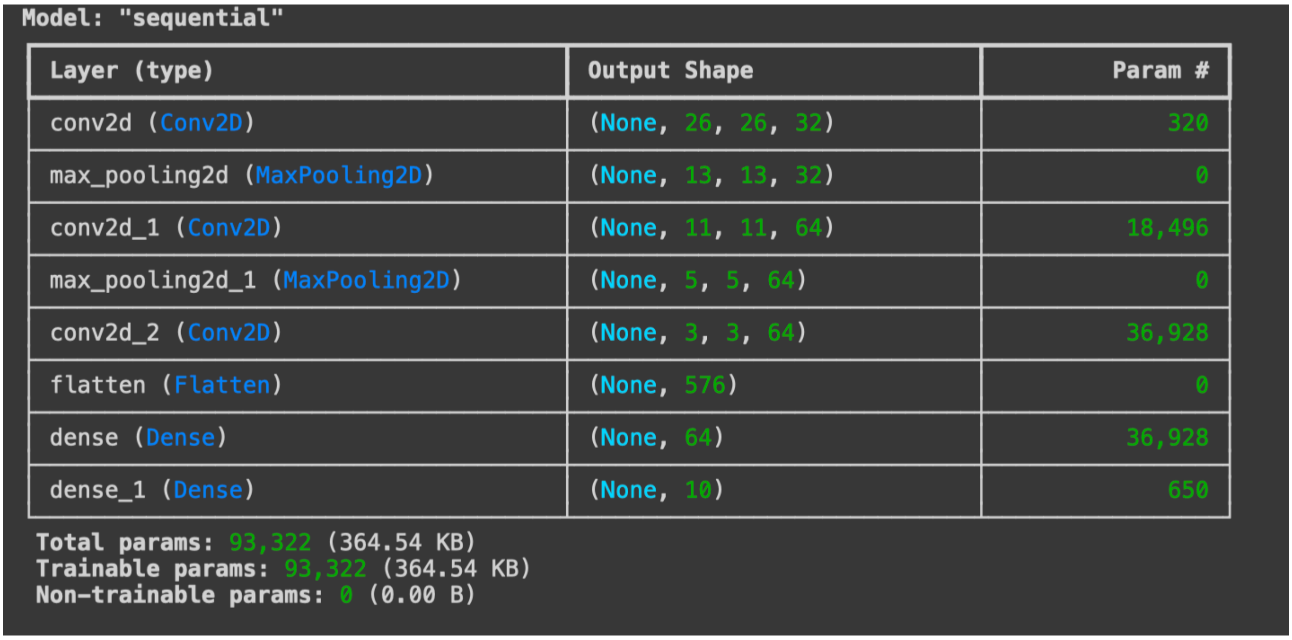
—>Conclusion:

Batch_size = 64 -> accuracy = 0.9892

Batch_size = 128 -> accuracy = 0.9884

Batch_size = 256 -> accuracy = 0.9894

Batch_size does not influence accuracy significantly.



```
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)       │      18,496 │
├──────────────────────────────┼──────────────────────────┼─────────────┤
│ max_pooling2d_1 (MaxPooling2D) │ (None, 5, 5, 64)       │           0 │
├──────────────────────────────┼──────────────────────────┼─────────────┤
│ conv2d_2 (Conv2D)            │ (None, 3, 3, 64)         │      36,928 │
├──────────────────────────────┼──────────────────────────┼─────────────┤
│ flatten (Flatten)            │ (None, 576)              │           0 │
├──────────────────────────────┼──────────────────────────┼─────────────┤
│ dense (Dense)                │ (None, 64)               │      36,928 │
├──────────────────────────────┼──────────────────────────┼─────────────┤
│ dense_1 (Dense)              │ (None, 10)               │         650 │
└──────────────────────────────┴──────────────────────────┴─────────────┘
 Total params: 93,322 (364.54 KB)
 Trainable params: 93,322 (364.54 KB)
 Non-trainable params: 0 (0.00 B)
```

Nonetheless, from the experiments, Batch_size = 256 gives the highest accuracy and allows faster computation since fewer weight updates per epoch.

(2)How many layers? How many neurons for each layer? How it affect your result?

A:

There are 8 layers:

Layer1:

First Convolutional Layer (Conv2D), 32 filters, Activation: ReLU

Effect: Extracts 32 feature maps (filters), each highlighting different patterns from the input image.

Layer2:

First MaxPooling Layer (MaxPooling2D)

Effect: Reduces the spatial dimensions of the feature maps by a factor of 2, helping to reduce computation and prevent overfitting while retaining important features.

Layer3:

Second Convolutional Layer (Conv2D), 64 filters, Activation: ReLU

Effect: Increases the number of filters to 64, enabling the model to learn more complex and detailed features from the downsampled image.

Layer4:

Second MaxPooling Layer (MaxPooling2D)

Effect: Reduces the spatial dimensions of the feature maps, further reducing the computational load while focusing on the most prominent features.

Layer5:

Third Convolutional Layer (Conv2D), 64 filters, Activation: ReLU

Effect: Further refines the learned features with another set of 64 filters.

Layer6:

Flatten Layer

Effect: Flattens the 3D output of the last convolutional layer into a 1D vector, preparing it for the fully connected layers.

Layer7:

Fully Connected Layer, 64 Neurons, Activation: ReLU

Effect: Processes the flattened feature vector, allowing the model to combine the learned features from the previous layers.

Layer8:

Fully Connected Layer, 10 Neurons, Activation: ReLU

Effect: The soft-max activation normalizes the output into a probability distribution over the 10 possible classes (digits 0-9).

—>Impact of Layers and Neurons on Results:

Convolutional Layers and Filters: Increasing the number of filters (from 32 to 64) allows the model to capture more complex features.

Pooling Layers: Help reduce the dimensionality and computational complexity.

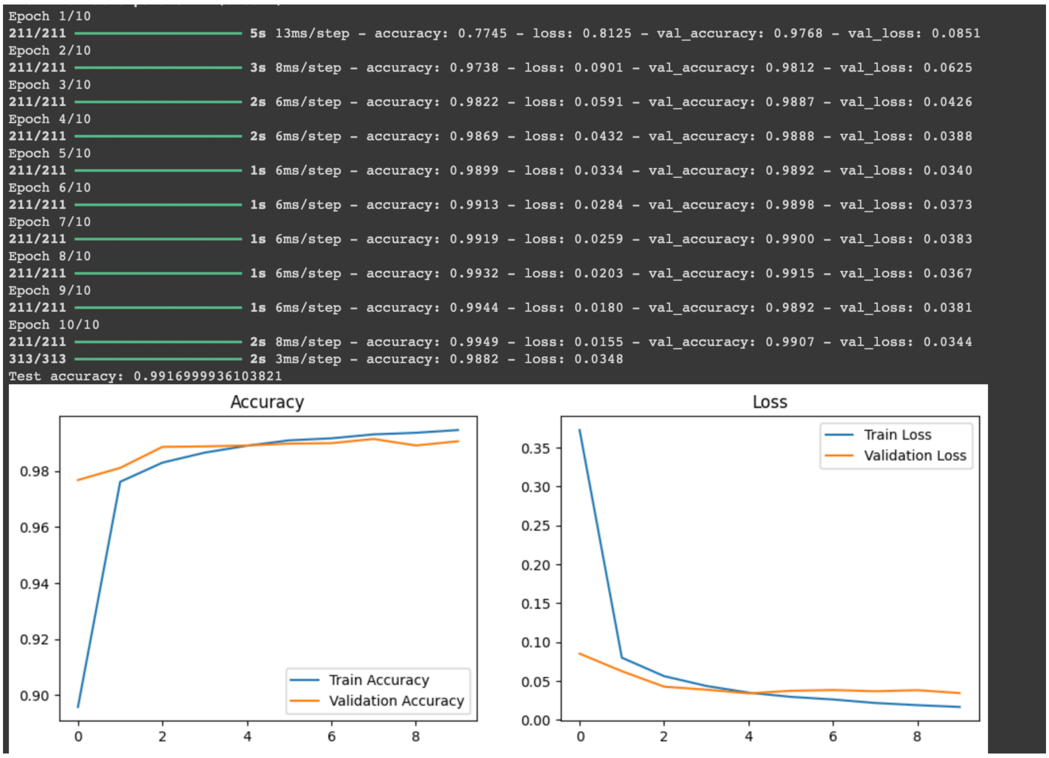
Dense Layer Neurons: The 64 neurons in the fully connected layer allow the model to combine the extracted features before making a prediction.

(3)Which function have you tried? Which one is better? Why? Please compare to each other.

A:

[4-0]ReLU (learning_rate=0.001, epochs=10 batch_size=256)

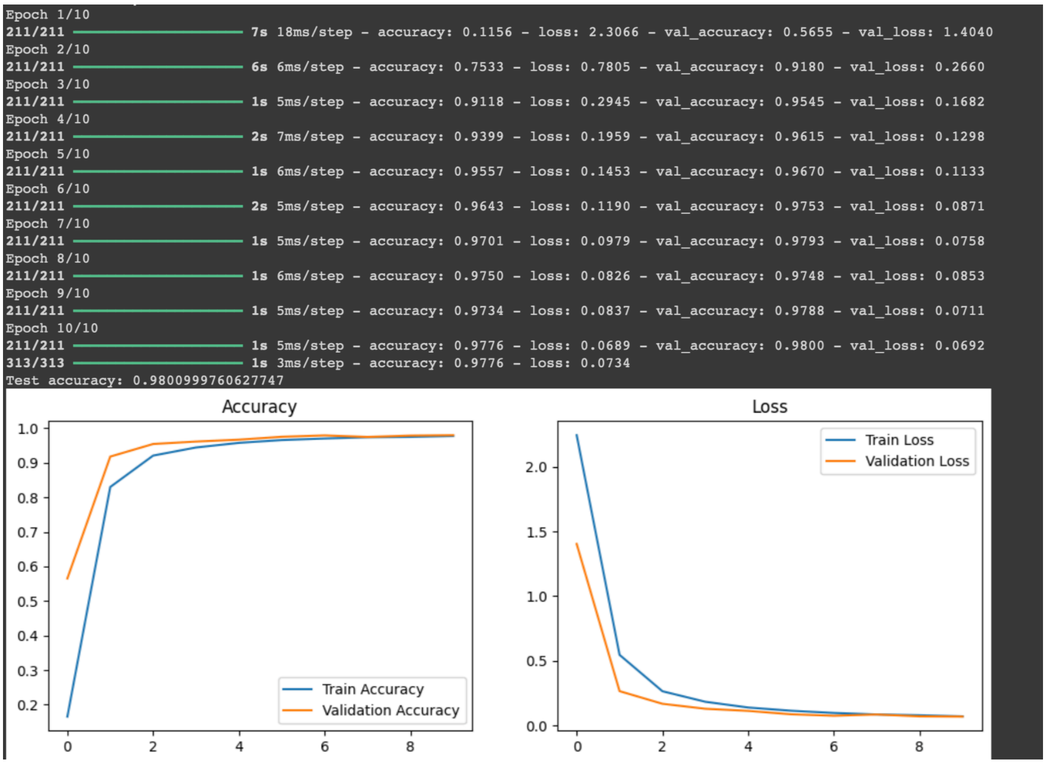
->Accuracy = 0.9916



```
Epoch 1/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 5s 13ms/step - accuracy: 0.7745 - loss: 0.8125 - val_accuracy: 0.9768 - val_loss: 0.0851
Epoch 2/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 3s 8ms/step - accuracy: 0.9738 - loss: 0.0901 - val_accuracy: 0.9812 - val_loss: 0.0625
Epoch 3/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9822 - loss: 0.0591 - val_accuracy: 0.9887 - val_loss: 0.0426
Epoch 4/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - accuracy: 0.9869 - loss: 0.0432 - val_accuracy: 0.9888 - val_loss: 0.0388
Epoch 5/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9899 - loss: 0.0334 - val_accuracy: 0.9892 - val_loss: 0.0340
Epoch 6/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9913 - loss: 0.0284 - val_accuracy: 0.9898 - val_loss: 0.0373
Epoch 7/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9919 - loss: 0.0259 - val_accuracy: 0.9900 - val_loss: 0.0383
Epoch 8/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9932 - loss: 0.0203 - val_accuracy: 0.9915 - val_loss: 0.0367
Epoch 9/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9944 - loss: 0.0180 - val_accuracy: 0.9892 - val_loss: 0.0381
Epoch 10/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 8ms/step - accuracy: 0.9949 - loss: 0.0155 - val_accuracy: 0.9907 - val_loss: 0.0344
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9882 - loss: 0.0348
Test accuracy: 0.9916999936103821
```

[4-1]Sigmoid (learning_rate=0.001, epochs=10 batch_size=256)

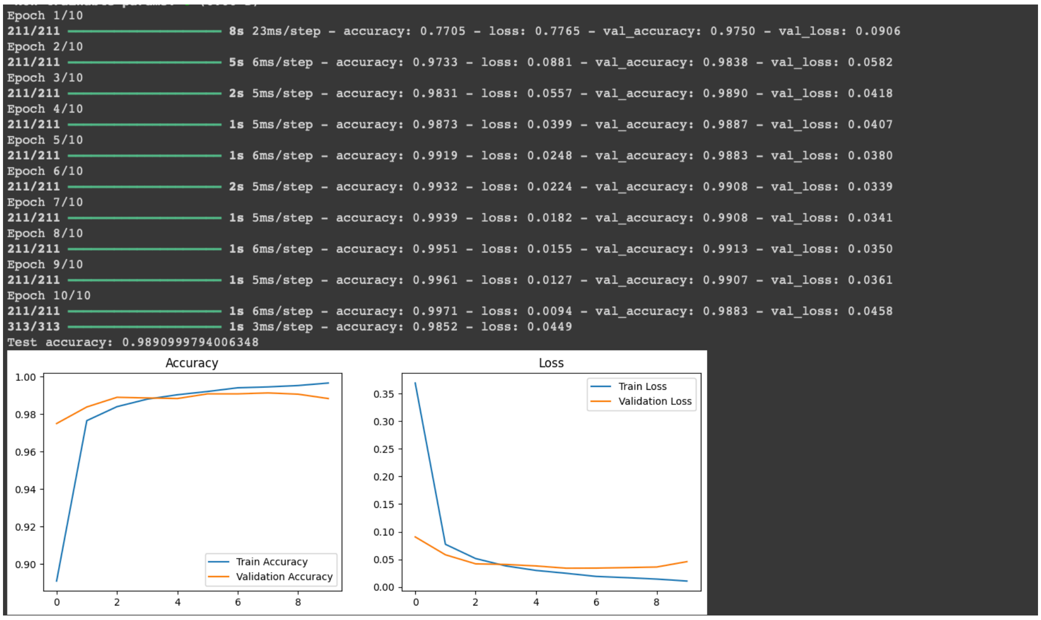
->Accuracy = 0.9801



```
Epoch 1/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 7s 18ms/step - accuracy: 0.1156 - loss: 2.3066 - val_accuracy: 0.5655 - val_loss: 1.4040
Epoch 2/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 6s 6ms/step - accuracy: 0.7533 - loss: 0.7805 - val_accuracy: 0.9180 - val_loss: 0.2660
Epoch 3/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9118 - loss: 0.2945 - val_accuracy: 0.9545 - val_loss: 0.1682
Epoch 4/10
        ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.9399 - loss: 0.1959 - val_accuracy: 0.9615 - val_loss: 0.1298
Epoch 5/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9557 - loss: 0.1453 - val_accuracy: 0.9670 - val_loss: 0.1133
Epoch 6/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9643 - loss: 0.1190 - val_accuracy: 0.9753 - val_loss: 0.0871
Epoch 7/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9701 - loss: 0.0979 - val_accuracy: 0.9793 - val_loss: 0.0758
Epoch 8/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9750 - loss: 0.0826 - val_accuracy: 0.9748 - val_loss: 0.0853
Epoch 9/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9734 - loss: 0.0837 - val_accuracy: 0.9788 - val_loss: 0.0711
Epoch 10/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9776 - loss: 0.0689 - val_accuracy: 0.9800 - val_loss: 0.0692
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9776 - loss: 0.0734
Test accuracy: 0.9800999760627747
```

[4-2] tanh (learning_rate=0.001, epochs=10 batch_size=256)

->Accuracy = 0.9890



```
Epoch 1/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 8s 23ms/step - accuracy: 0.7705 - loss: 0.7765 - val_accuracy: 0.9750 - val_loss: 0.0906
Epoch 2/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 5s 6ms/step - accuracy: 0.9733 - loss: 0.0881 - val_accuracy: 0.9838 - val_loss: 0.0582
Epoch 3/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9831 - loss: 0.0557 - val_accuracy: 0.9890 - val_loss: 0.0418
Epoch 4/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9873 - loss: 0.0399 - val_accuracy: 0.9887 - val_loss: 0.0407
Epoch 5/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9919 - loss: 0.0248 - val_accuracy: 0.9883 - val_loss: 0.0380
Epoch 6/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9932 - loss: 0.0224 - val_accuracy: 0.9908 - val_loss: 0.0339
Epoch 7/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9939 - loss: 0.0182 - val_accuracy: 0.9908 - val_loss: 0.0341
Epoch 8/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9951 - loss: 0.0155 - val_accuracy: 0.9913 - val_loss: 0.0350
Epoch 9/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9961 - loss: 0.0127 - val_accuracy: 0.9907 - val_loss: 0.0361
Epoch 10/10
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9971 - loss: 0.0094 - val_accuracy: 0.9883 - val_loss: 0.0458
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.9852 - loss: 0.0449
Test accuracy: 0.9890999794006348
```

—>Conclusion:

ReLU -> Accuracy = 0.9916

Sigmoid -> Accuracy = 0.9801

tanh-> Accuracy = 0.9894

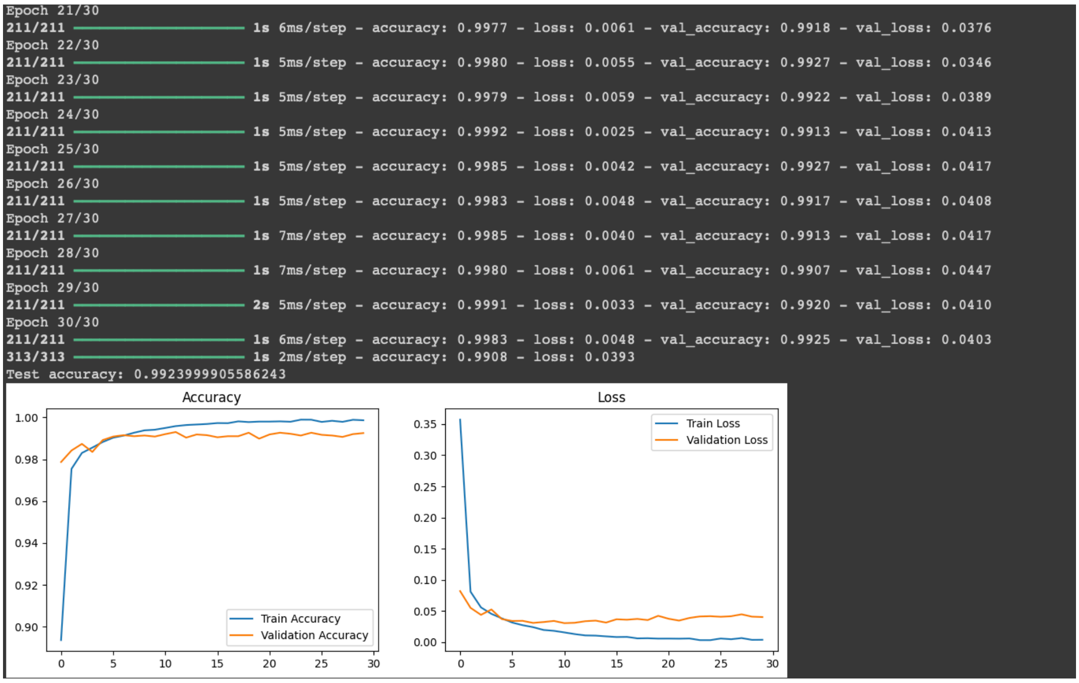
From the experiments, using ReLU has the best accuracy.

2.Loss

(1)Your final loss of testing data is?

A:

[5-0]Final Accuracy and Loss (learning_rate=0.001, epochs=30, batch_size=256, ReLU)



```
Epoch 21/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9977 - loss: 0.0061 - val_accuracy: 0.9918 - val_loss: 0.0376
Epoch 22/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9980 - loss: 0.0055 - val_accuracy: 0.9927 - val_loss: 0.0346
Epoch 23/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9979 - loss: 0.0059 - val_accuracy: 0.9922 - val_loss: 0.0389
Epoch 24/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9992 - loss: 0.0025 - val_accuracy: 0.9913 - val_loss: 0.0413
Epoch 25/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9985 - loss: 0.0042 - val_accuracy: 0.9927 - val_loss: 0.0417
Epoch 26/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9983 - loss: 0.0048 - val_accuracy: 0.9917 - val_loss: 0.0408
Epoch 27/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9985 - loss: 0.0040 - val_accuracy: 0.9913 - val_loss: 0.0417
Epoch 28/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9980 - loss: 0.0061 - val_accuracy: 0.9907 - val_loss: 0.0447
Epoch 29/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9991 - loss: 0.0033 - val_accuracy: 0.9920 - val_loss: 0.0410
Epoch 30/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9983 - loss: 0.0048 - val_accuracy: 0.9925 - val_loss: 0.0403
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9908 - loss: 0.0393
Test accuracy: 0.9923999905586243
```

After previous experiments, the final loss = 0.0393

(2)Is there any way you can improve it? Please describe.

A:

The following are the possible methods:

Increase the model complexity such as adding more layers or neurons

Increase training epochs

Fine-Tune the learning rate

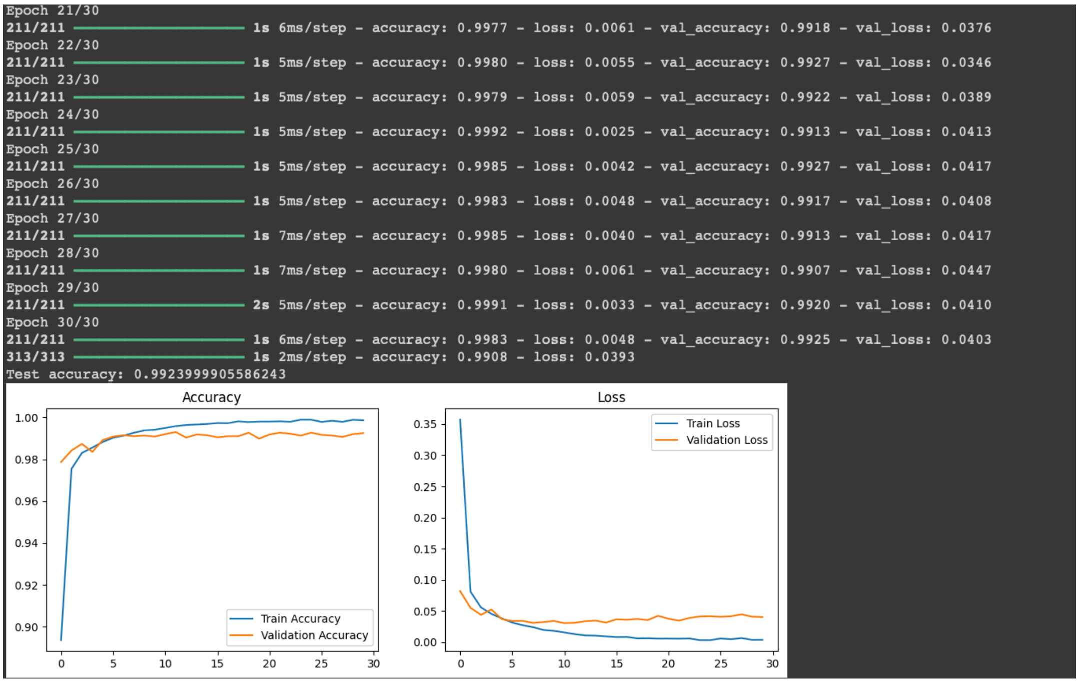
Using data augmentation

Fine-Tune the batch size

3.Accuracy

(1)Your final accuracy is?

A:



```
Epoch 21/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9977 - loss: 0.0061 - val_accuracy: 0.9918 - val_loss: 0.0376
Epoch 22/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9980 - loss: 0.0055 - val_accuracy: 0.9927 - val_loss: 0.0346
Epoch 23/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9979 - loss: 0.0059 - val_accuracy: 0.9922 - val_loss: 0.0389
Epoch 24/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9992 - loss: 0.0025 - val_accuracy: 0.9913 - val_loss: 0.0413
Epoch 25/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9985 - loss: 0.0042 - val_accuracy: 0.9927 - val_loss: 0.0417
Epoch 26/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 5ms/step - accuracy: 0.9983 - loss: 0.0048 - val_accuracy: 0.9917 - val_loss: 0.0408
Epoch 27/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9985 - loss: 0.0040 - val_accuracy: 0.9913 - val_loss: 0.0417
Epoch 28/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.9980 - loss: 0.0061 - val_accuracy: 0.9907 - val_loss: 0.0447
Epoch 29/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - accuracy: 0.9991 - loss: 0.0033 - val_accuracy: 0.9920 - val_loss: 0.0410
Epoch 30/30
211/211 ━━━━━━━━━━━━━━━━━━━━ 1s 6ms/step - accuracy: 0.9983 - loss: 0.0048 - val_accuracy: 0.9925 - val_loss: 0.0403
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9908 - loss: 0.0393
Test accuracy: 0.9923999905586243
```

After previous experiments, the final accuracy = 0.9924

(2)Is there any way you can improve it? Please describe.

A:

The following are the possible methods:

Increase the model complexity such as adding more layers or neurons

Increase training epochs

Fine-Tune the learning rate

Using data augmentation

Fine-Tune the batch size