CNN

2021 年 4 月 25 日

[2]:
```
import keras
from keras import layers
from keras import models
from keras.datasets import mnist
from keras.utils import to_categorical
```

0.0.1 数据的加载并将其划分为训练数据与测试数据以及归一化

[3]:
```
(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)
```

0.0.2 构建卷积神经网络模型

[4]:
```
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'))
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
_________________________________________________________________
```

[5]:
```
model.compile(optimizer='SGD',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

0.0.3 利用训练数据进行模型的训练

[6]:
```
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

```
Epoch 1/5
938/938 [==============================] - 36s 39ms/step - loss: 0.8801 -
accuracy: 0.7405
Epoch 2/5
938/938 [==============================] - 36s 39ms/step - loss: 0.2198 -
accuracy: 0.9337
Epoch 3/5
938/938 [==============================] - 37s 40ms/step - loss: 0.1428 -
accuracy: 0.9567
Epoch 4/5
938/938 [==============================] - 38s 41ms/step - loss: 0.1092 -
accuracy: 0.9672
Epoch 5/5
938/938 [==============================] - 36s 38ms/step - loss: 0.0923 -
accuracy: 0.9715
```

[6]:
```
<tensorflow.python.keras.callbacks.History at 0x168019d3b80>
```

0.0.4 利用测试数据进行模型的测试

[7]:
```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('测试数据准确率',test_acc)
```

```
313/313 [==============================] - 2s 6ms/step - loss: 0.0712 -
accuracy: 0.9782
测试数据准确率 0.9782000184059143
```

0.0.5 进行数据的可视化处理，加大 **epoch** 观察模型在训练与测试数据集上的情况

[8]:
```
import tensorflow as tf
import os
import numpy as np
from matplotlib import pyplot as plt
```

```
checkpoint_save_path = "./checkpoint_2/mnist.ckpt"
if os.path.exists(checkpoint_save_path + '.index'):
    print('-------------load the model-----------------')
    model.load_weights(checkpoint_save_path)

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_save_path,
                                                 save_weights_only=True,
                                                 save_best_only=True)

history = model.fit(train_images,
                    train_labels,
                    batch_size=64,
                    epochs=55,
                    validation_data=(test_images, test_labels),
                    callbacks=[cp_callback])
model.summary()

# print(model.trainable_variables)
file = open('./weights_2.txt', 'w')
for v in model.trainable_variables:
    file.write(str(v.name) + '\n')
    file.write(str(v.shape) + '\n')
    file.write(str(v.numpy()) + '\n')
file.close()
```

```
Epoch 1/55
938/938 [==============================] - 37s 40ms/step - loss: 0.0800 -
accuracy: 0.9753 - val_loss: 0.0688 - val_accuracy: 0.9797
Epoch 2/55
938/938 [==============================] - 37s 39ms/step - loss: 0.0710 -
accuracy: 0.9779 - val_loss: 0.0601 - val_accuracy: 0.9797
Epoch 3/55
938/938 [==============================] - 38s 41ms/step - loss: 0.0645 -
accuracy: 0.9798 - val_loss: 0.0580 - val_accuracy: 0.9807
Epoch 4/55
```

```
938/938 [==============================] - 40s 43ms/step - loss: 0.0588 -
accuracy: 0.9818 - val_loss: 0.0560 - val_accuracy: 0.9837
Epoch 5/55
938/938 [==============================] - 33s 36ms/step - loss: 0.0550 -
accuracy: 0.9825 - val_loss: 0.0783 - val_accuracy: 0.9747
Epoch 6/55
938/938 [==============================] - 35s 37ms/step - loss: 0.0505 -
accuracy: 0.9843 - val_loss: 0.0455 - val_accuracy: 0.9865
Epoch 7/55
938/938 [==============================] - 33s 36ms/step - loss: 0.0475 -
accuracy: 0.9850 - val_loss: 0.0496 - val_accuracy: 0.9849
Epoch 8/55
938/938 [==============================] - 35s 37ms/step - loss: 0.0447 -
accuracy: 0.9862 - val_loss: 0.0462 - val_accuracy: 0.9864
Epoch 9/55
938/938 [==============================] - 33s 35ms/step - loss: 0.0424 -
accuracy: 0.9868 - val_loss: 0.0393 - val_accuracy: 0.9874
Epoch 10/55
938/938 [==============================] - 35s 38ms/step - loss: 0.0397 -
accuracy: 0.9876 - val_loss: 0.0425 - val_accuracy: 0.9869
Epoch 11/55
938/938 [==============================] - 34s 37ms/step - loss: 0.0384 -
accuracy: 0.9883 - val_loss: 0.0366 - val_accuracy: 0.9884
Epoch 12/55
938/938 [==============================] - 34s 37ms/step - loss: 0.0359 -
accuracy: 0.9890 - val_loss: 0.0360 - val_accuracy: 0.9886
Epoch 13/55
938/938 [==============================] - 35s 37ms/step - loss: 0.0337 -
accuracy: 0.9897 - val_loss: 0.0519 - val_accuracy: 0.9832
Epoch 14/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0329 -
accuracy: 0.9900 - val_loss: 0.0391 - val_accuracy: 0.9877
Epoch 15/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0309 -
accuracy: 0.9904 - val_loss: 0.0368 - val_accuracy: 0.9885
Epoch 16/55
938/938 [==============================] - 35s 38ms/step - loss: 0.0294 -
```

```
accuracy: 0.9909 - val_loss: 0.0347 - val_accuracy: 0.9888
Epoch 17/55
938/938 [==============================] - 33s 36ms/step - loss: 0.0283 -
accuracy: 0.9916 - val_loss: 0.0330 - val_accuracy: 0.9885
Epoch 18/55
938/938 [==============================] - 34s 37ms/step - loss: 0.0271 -
accuracy: 0.9916 - val_loss: 0.0383 - val_accuracy: 0.9876
Epoch 19/55
938/938 [==============================] - 35s 37ms/step - loss: 0.0259 -
accuracy: 0.9920 - val_loss: 0.0330 - val_accuracy: 0.9889
Epoch 20/55
938/938 [==============================] - 34s 36ms/step - loss: 0.0247 -
accuracy: 0.9924 - val_loss: 0.0322 - val_accuracy: 0.9903
Epoch 21/55
938/938 [==============================] - 34s 37ms/step - loss: 0.0241 -
accuracy: 0.9925 - val_loss: 0.0477 - val_accuracy: 0.9851
Epoch 22/55
938/938 [==============================] - 32s 34ms/step - loss: 0.0228 -
accuracy: 0.9929 - val_loss: 0.0320 - val_accuracy: 0.9908
Epoch 23/55
938/938 [==============================] - 32s 34ms/step - loss: 0.0215 -
accuracy: 0.9937 - val_loss: 0.0308 - val_accuracy: 0.9900
Epoch 24/55
938/938 [==============================] - 32s 34ms/step - loss: 0.0208 -
accuracy: 0.9935 - val_loss: 0.0296 - val_accuracy: 0.9911
Epoch 25/55
938/938 [==============================] - 32s 34ms/step - loss: 0.0199 -
accuracy: 0.9938 - val_loss: 0.0333 - val_accuracy: 0.9896
Epoch 26/55
938/938 [==============================] - 32s 34ms/step - loss: 0.0197 -
accuracy: 0.9938 - val_loss: 0.0438 - val_accuracy: 0.9865
Epoch 27/55
938/938 [==============================] - 34s 36ms/step - loss: 0.0187 -
accuracy: 0.9945 - val_loss: 0.0292 - val_accuracy: 0.9907
Epoch 28/55
938/938 [==============================] - 34s 36ms/step - loss: 0.0177 -
accuracy: 0.9948 - val_loss: 0.0299 - val_accuracy: 0.9894
```

```
Epoch 29/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0175 -
accuracy: 0.9948 - val_loss: 0.0306 - val_accuracy: 0.9898
Epoch 30/55
938/938 [==============================] - 33s 36ms/step - loss: 0.0166 -
accuracy: 0.9951 - val_loss: 0.0311 - val_accuracy: 0.9896
Epoch 31/55
938/938 [==============================] - 34s 36ms/step - loss: 0.0154 -
accuracy: 0.9955 - val_loss: 0.0319 - val_accuracy: 0.9900
Epoch 32/55
938/938 [==============================] - 34s 36ms/step - loss: 0.0152 -
accuracy: 0.9953 - val_loss: 0.0319 - val_accuracy: 0.9902
Epoch 33/55
938/938 [==============================] - 33s 35ms/step - loss: 0.0146 -
accuracy: 0.9958 - val_loss: 0.0312 - val_accuracy: 0.9903
Epoch 34/55
938/938 [==============================] - 33s 35ms/step - loss: 0.0137 -
accuracy: 0.9962 - val_loss: 0.0378 - val_accuracy: 0.9879
Epoch 35/55
938/938 [==============================] - 34s 36ms/step - loss: 0.0136 -
accuracy: 0.9959 - val_loss: 0.0323 - val_accuracy: 0.9903
Epoch 36/55
938/938 [==============================] - 33s 36ms/step - loss: 0.0130 -
accuracy: 0.9961 - val_loss: 0.0319 - val_accuracy: 0.9906
Epoch 37/55
938/938 [==============================] - 34s 36ms/step - loss: 0.0123 -
accuracy: 0.9964 - val_loss: 0.0301 - val_accuracy: 0.9909
Epoch 38/55
938/938 [==============================] - 33s 36ms/step - loss: 0.0120 -
accuracy: 0.9964 - val_loss: 0.0310 - val_accuracy: 0.9903
Epoch 39/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0112 -
accuracy: 0.9967 - val_loss: 0.0314 - val_accuracy: 0.9904
Epoch 40/55
938/938 [==============================] - 37s 39ms/step - loss: 0.0108 -
accuracy: 0.9969 - val_loss: 0.0311 - val_accuracy: 0.9907
Epoch 41/55
```

```
938/938 [==============================] - 37s 39ms/step - loss: 0.0107 -
accuracy: 0.9968 - val_loss: 0.0303 - val_accuracy: 0.9904
Epoch 42/55
938/938 [==============================] - 37s 40ms/step - loss: 0.0096 -
accuracy: 0.9976 - val_loss: 0.0309 - val_accuracy: 0.9901
Epoch 43/55
938/938 [==============================] - 37s 39ms/step - loss: 0.0100 -
accuracy: 0.9971 - val_loss: 0.0317 - val_accuracy: 0.9904
Epoch 44/55
938/938 [==============================] - 42s 45ms/step - loss: 0.0096 -
accuracy: 0.9975 - val_loss: 0.0304 - val_accuracy: 0.9900
Epoch 45/55
938/938 [==============================] - 39s 41ms/step - loss: 0.0089 -
accuracy: 0.9974 - val_loss: 0.0306 - val_accuracy: 0.9905
Epoch 46/55
938/938 [==============================] - 35s 38ms/step - loss: 0.0085 -
accuracy: 0.9980 - val_loss: 0.0346 - val_accuracy: 0.9906
Epoch 47/55
938/938 [==============================] - 35s 38ms/step - loss: 0.0083 -
accuracy: 0.9977 - val_loss: 0.0335 - val_accuracy: 0.9904
Epoch 48/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0078 -
accuracy: 0.9978 - val_loss: 0.0368 - val_accuracy: 0.9896
Epoch 49/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0079 -
accuracy: 0.9978 - val_loss: 0.0323 - val_accuracy: 0.9902
Epoch 50/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0073 -
accuracy: 0.9982 - val_loss: 0.0329 - val_accuracy: 0.9904
Epoch 51/55
938/938 [==============================] - 37s 40ms/step - loss: 0.0067 -
accuracy: 0.9984 - val_loss: 0.0313 - val_accuracy: 0.9906
Epoch 52/55
938/938 [==============================] - 38s 40ms/step - loss: 0.0066 -
accuracy: 0.9983 - val_loss: 0.0326 - val_accuracy: 0.9899
Epoch 53/55
938/938 [==============================] - 37s 40ms/step - loss: 0.0062 -
```

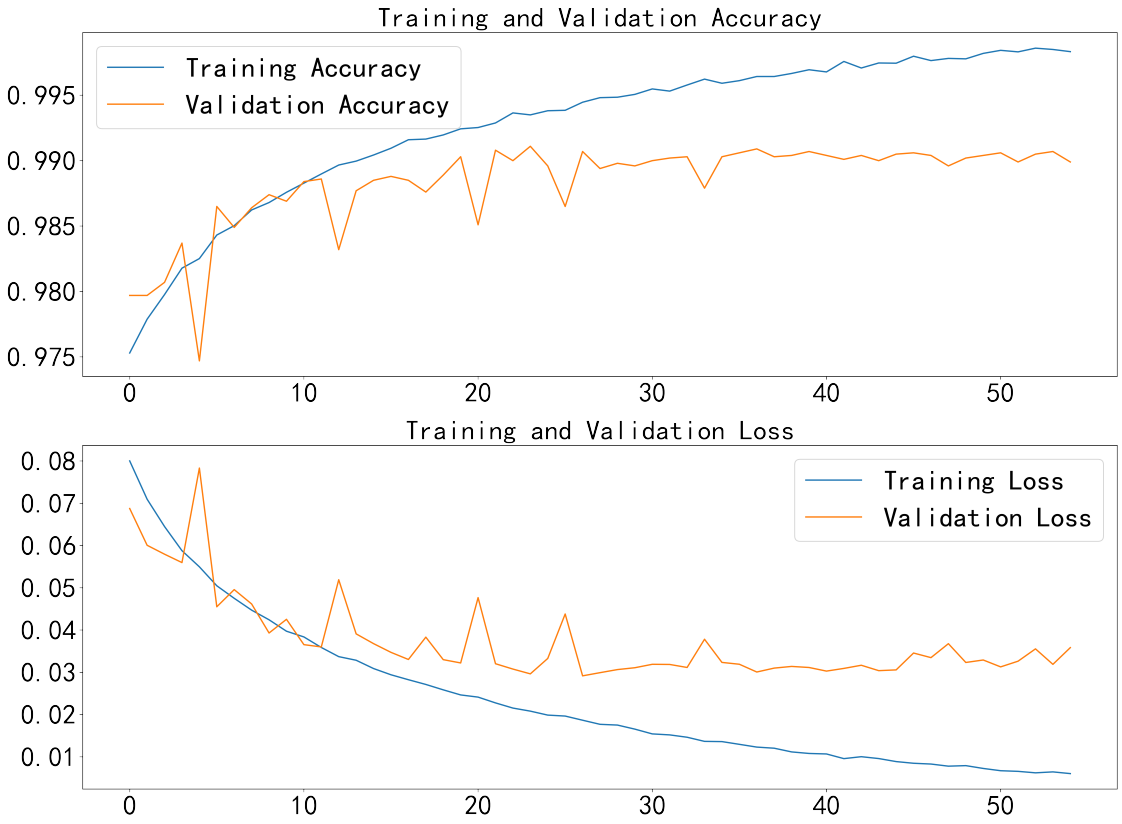
```
accuracy: 0.9986 - val_loss: 0.0355 - val_accuracy: 0.9905
Epoch 54/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0064 -
accuracy: 0.9985 - val_loss: 0.0319 - val_accuracy: 0.9907
Epoch 55/55
938/938 [==============================] - 36s 38ms/step - loss: 0.0060 -
accuracy: 0.9983 - val_loss: 0.0358 - val_accuracy: 0.9899
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
_________________________________________________________________
```

[9]:
```
##loss 是训练集 loss，val_loss 是测试集 loss
#sparse_categorical_accuracy 是训练集 sparse_categorical_accuracy，
val_sparse_categorical_accuracy 是测试集 sparse_categorical_accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
```

```
loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(dpi=500, figsize=[20, 15])
plt.rcParams["font.sans-serif"] = ["SimHei"]
#subplot 函数将图像分为两行一列，这段代码画出第一行
plt.subplot(2, 1, 1)
#plot 描述曲线
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
#title 函数设置图标题
plt.title('Training and Validation Accuracy',fontsize=30)
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)
# 画出图例
plt.legend(fontsize=30)

# 这段代码画出第二行
plt.subplot(2, 1, 2)
plt.plot(loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss',fontsize=30)
plt.xticks(fontsize=30)
plt.yticks(fontsize=30)
plt.legend(fontsize=30)
plt.show()
```

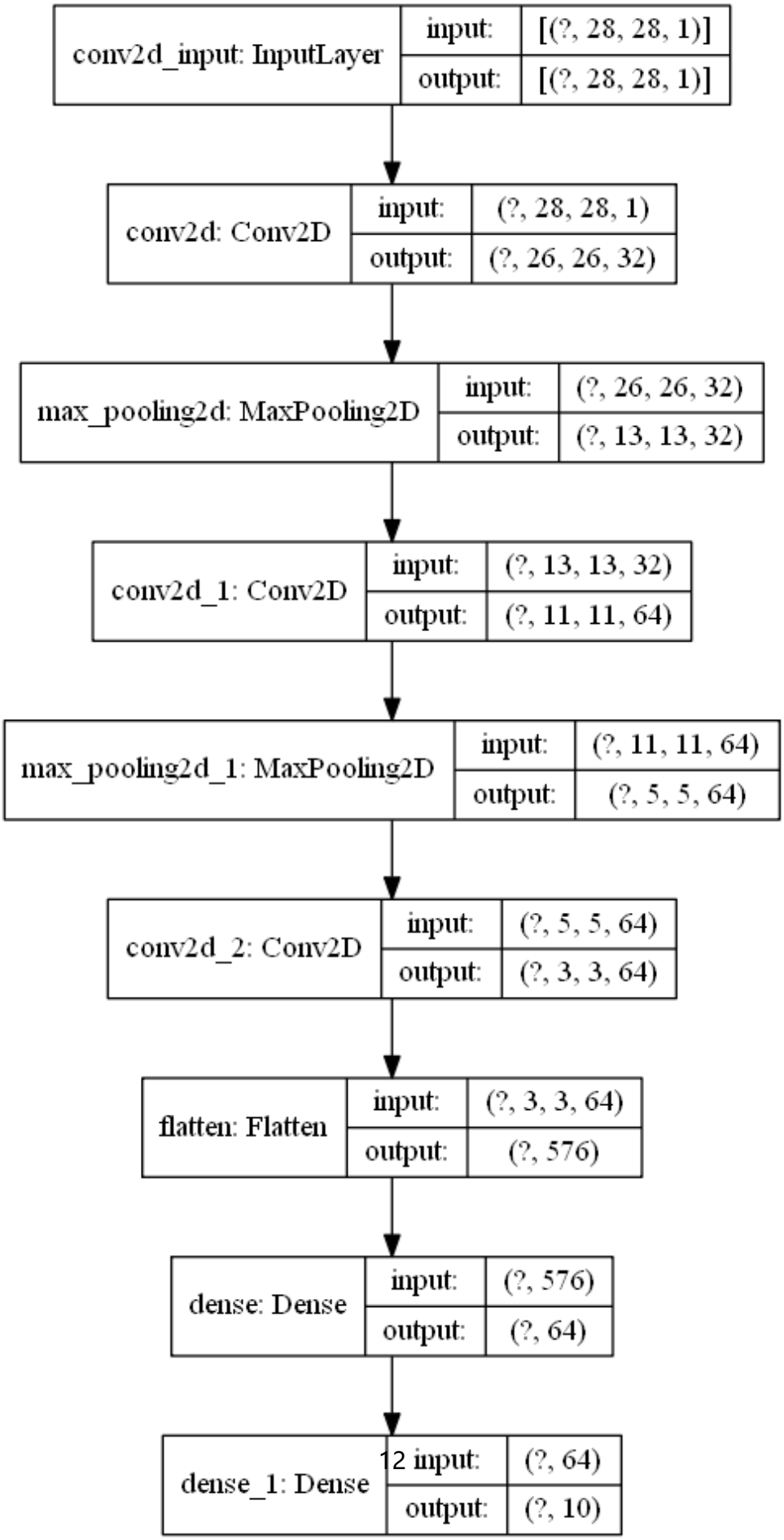


0.0.6 可视化绘图

[10]:
```
from keras.models import load_model
from keras.utils import plot_model


# 输出模型，将结果保存到项目文件夹中
plot_model(model, to_file='model_2.png', show_shapes='True')
```

[10]:


conv2d_input: InputLayer
input: [(?, 28, 28, 1)]
output: [(?, 28, 28, 1)]
conv2d: Conv2D
input: (?, 28, 28, 1)
output: (?, 26, 26, 32)
max_pooling2d: MaxPooling2D
input: (?, 26, 26, 32)
output: (?, 13, 13, 32)
conv2d_1: Conv2D
input: (?, 13, 13, 32)
output: (?, 11, 11, 64)
max_pooling2d_1: MaxPooling2D
input: (?, 11, 11, 64)
output: (?, 5, 5, 64)
conv2d_2: Conv2D
input: (?, 5, 5, 64)
output: (?, 3, 3, 64)
flatten: Flatten
input: (?, 3, 3, 64)
output: (?, 576)
dense: Dense
input: (?, 576)
output: (?, 64)
dense_1: Dense
input: (?, 64)
output: (?, 10)

[ ]: