tensorflow2教程-CNN變體網路

In [1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
print(tf.__version__)
```

```
2.3.1
```

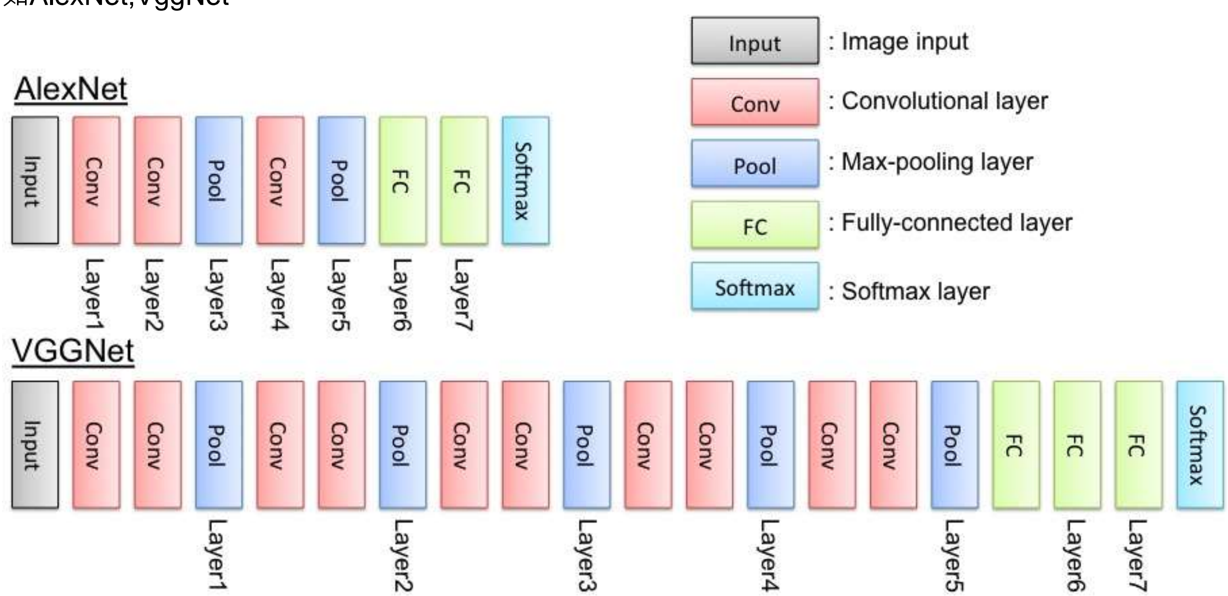
1.載入數據

In [2]:
```
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
x_train = x_train.reshape((-1,28,28,1))
x_test = x_test.reshape((-1,28,28,1))
print(x_train.shape, ' ', y_train.shape)
print(x_test.shape, ' ', y_test.shape)
```

```
(60000, 28, 28, 1)   (60000,)
(10000, 28, 28, 1)   (10000,)
```

2.簡單的深度網路

如AlexNet,VggNet



In [3]:
```
x_shape  = x_train.shape
deep_model = keras.Sequential(
[
    layers.Conv2D(input_shape=((x_shape[1], x_shape[2], x_shape[3])),
                 filters=32, kernel_size=(3,3), strides=(1,1), padding='same', ac
    layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), padding='same', a
    layers.MaxPool2D(pool_size=(2,2)),
    layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), padding='same', a
    layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), padding='same', a
    layers.MaxPool2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(32, activation='relu'),
    layers.Dense(10, activation='softmax')

])
```

In [4]:
```
deep_model.compile(optimizer=keras.optimizers.Adam(),
             loss=keras.losses.SparseCategoricalCrossentropy(),
            metrics=['accuracy'])
deep_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320       
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 28, 28, 32)        9248      
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 32)        9248      
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 14, 14, 32)        9248      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 32)          0         
_________________________________________________________________
flatten (Flatten)            (None, 1568)              0         
_________________________________________________________________
dense (Dense)                (None, 32)                50208     
_________________________________________________________________
dense_1 (Dense)              (None, 10)                330       
=================================================================
Total params: 78,602
Trainable params: 78,602
Non-trainable params: 0
_________________________________________________________________
```

In [5]:
```
history = deep_model.fit(x_train, y_train, batch_size=64, epochs=5, validation_sp
```

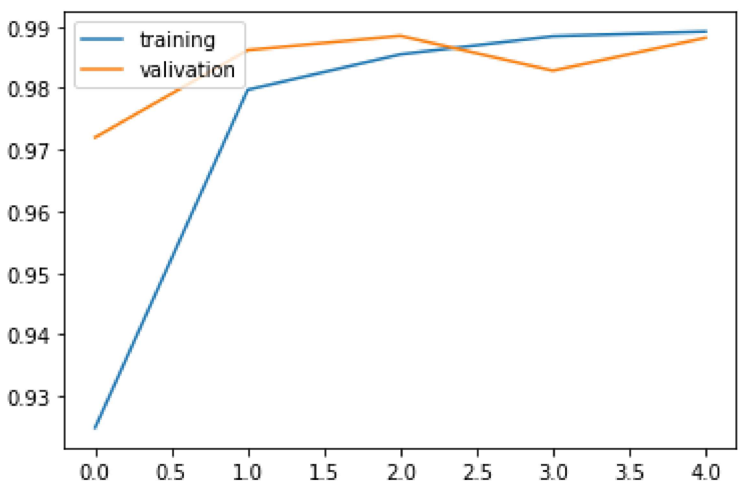
```
Epoch 1/5
844/844 [==============================] - 65s 76ms/step - loss: 0.2987 - accur
acy: 0.9249 - val_loss: 0.1042 - val_accuracy: 0.9720
Epoch 2/5
844/844 [==============================] - 70s 83ms/step - loss: 0.0657 - accur
acy: 0.9797 - val_loss: 0.0457 - val_accuracy: 0.9862
Epoch 3/5
844/844 [==============================] - 75s 88ms/step - loss: 0.0463 - accur
acy: 0.9855 - val_loss: 0.0388 - val_accuracy: 0.9885
Epoch 4/5
844/844 [==============================] - 86s 102ms/step - loss: 0.0371 - accu
racy: 0.9884 - val_loss: 0.0564 - val_accuracy: 0.9828
Epoch 5/5
844/844 [==============================] - 92s 109ms/step - loss: 0.0335 - accu
racy: 0.9892 - val_loss: 0.0381 - val_accuracy: 0.9882
```

In [6]:
```
deep_model.evaluate(x_test, y_test)
```

```
313/313 [==============================] - 4s 14ms/step - loss: 0.0416 - accura
cy: 0.9882
```

Out[6]:
```
[0.04156742990016937, 0.9882000088691711]
```

In [7]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'valivation'], loc='upper left')
plt.show()
```



In [10]:
```
result = deep_model.evaluate(x_test, y_test)
```

```
10000/10000 [==============================] - 2s 219us/sample - loss: 0.0445 -
accuracy: 0.9863
```

3.添加了其它功能層的深度卷積

In [8]:
```
x_shape  = x_train.shape
deep_model = keras.Sequential(
[
    layers.Conv2D(input_shape=((x_shape[1], x_shape[2], x_shape[3])),
                 filters=32, kernel_size=(3,3), strides=(1,1), padding='same', ac
    layers.BatchNormalization(),
    layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), padding='same', a
    layers.BatchNormalization(),
    layers.MaxPool2D(pool_size=(2,2)),
    layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), padding='same', a
    layers.BatchNormalization(),
    layers.BatchNormalization(),
    layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), padding='same', a
    layers.MaxPool2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')

])
```

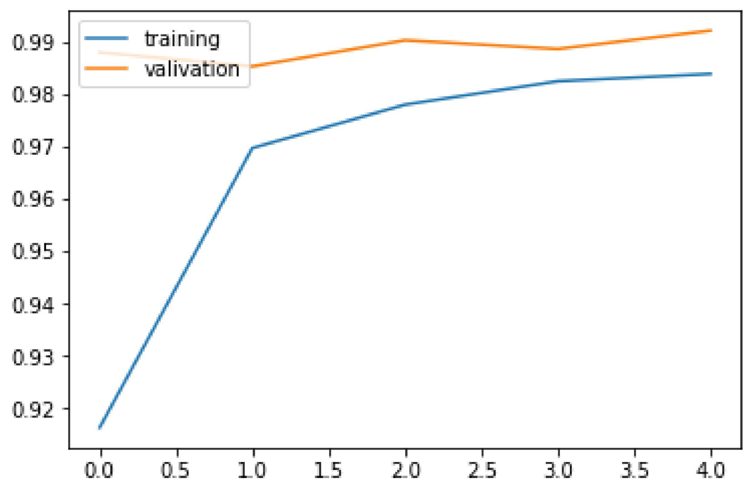
In [10]:
```
deep_model.compile(optimizer=keras.optimizers.Adam(),
             loss=keras.losses.SparseCategoricalCrossentropy(),
            metrics=['accuracy'])
deep_model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 32)        128
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 28, 28, 32)        9248
_________________________________________________________________
batch_normalization_1 (Batch (None, 28, 28, 32)        128
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
batch_normalization_2 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
batch_normalization_3 (Batch (None, 14, 14, 32)        128
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 14, 14, 32)        9248
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 32)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1568)              0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                50208
_________________________________________________________________
dropout (Dropout)            (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                330
=================================================================
Total params: 79,114
Trainable params: 78,858
Non-trainable params: 256
_________________________________________________________________
```

In [11]:
```
history = deep_model.fit(x_train, y_train, batch_size=64, epochs=5, validation_sp
```

```
Epoch 1/5
844/844 [==============================] - 161s 191ms/step - loss: 0.3188 - acc
uracy: 0.8983 - val_loss: 0.0598 - val_accuracy: 0.9822
Epoch 2/5
844/844 [==============================] - 165s 195ms/step - loss: 0.1280 - acc
uracy: 0.9599 - val_loss: 0.0625 - val_accuracy: 0.9837
Epoch 3/5
844/844 [==============================] - 182s 215ms/step - loss: 0.0981 - acc
uracy: 0.9693 - val_loss: 0.0368 - val_accuracy: 0.9897
Epoch 4/5
844/844 [==============================] - 190s 225ms/step - loss: 0.0803 - acc
uracy: 0.9738 - val_loss: 0.0424 - val_accuracy: 0.9893
Epoch 5/5
844/844 [==============================] - 181s 215ms/step - loss: 0.0662 - acc
uracy: 0.9789 - val_loss: 0.0412 - val_accuracy: 0.9890
```

In [14]:
```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'valivation'], loc='upper left')
plt.show()
```

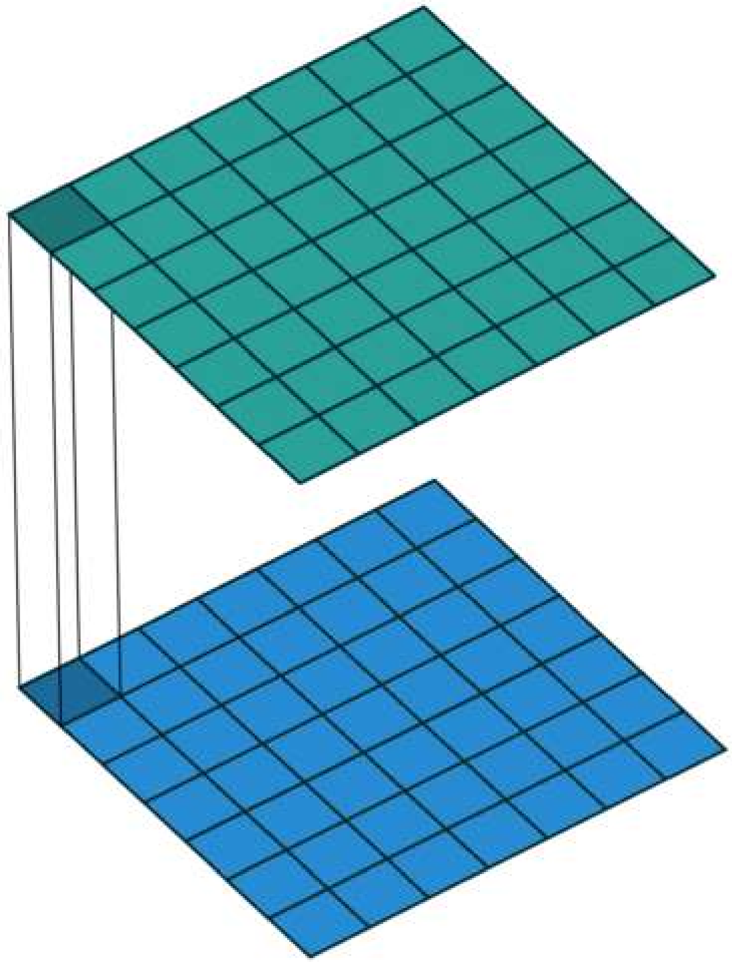


In [15]:
```
result = deep_model.evaluate(x_test, y_test)
```

```
10000/10000 [==============================] - 4s 365us/sample - loss: 0.0288 -
accuracy: 0.9909
```

4.NIN網路

Min等人在 2013年（[https://arxiv.org/abs/1312.4400）提出了減少模型中參數數量的方法之一](https://arxiv.org/abs/1312.4400%EF%BC%89%E6%8F%90%E5%87%BA%E4%BA%86%E6%B8%9B%E5%B0%91%E6%A8%A1%E5%9E%8B%E4%B8%AD%E5%8F%83%E6%95%B8%E6%95%B8%E9%87%8F%E7%9A%84%E6%96%B9%E6%B3%95%E4%B9%8B%E4%B8%80) 即“網路中的網路（NIN）”或“1X1卷積” 方法很簡單 - 在其他卷積層之後添加卷積層 具有降低圖像空間的維度（深度）的效果，有效地減少了參數的數量



GoogleNet 中就用到了NIN結構

In [16]:
```
x_shape  = x_train.shape
deep_model = keras.Sequential(
[
    layers.Conv2D(input_shape=((x_shape[1], x_shape[2], x_shape[3])),
                  filters=32, kernel_size=(3,3), strides=(1,1), padding='same', ac
    layers.BatchNormalization(),
    layers.Conv2D(filters=16, kernel_size=(1,1), strides=(1,1), padding='valid',
    layers.BatchNormalization(),
    layers.MaxPool2D(pool_size=(2,2)),
    layers.Conv2D(filters=32, kernel_size=(3,3), strides=(1,1), padding='same', a
    layers.BatchNormalization(),
    layers.Conv2D(filters=16, kernel_size=(1,1), strides=(1,1), padding='valid',
    layers.BatchNormalization(),
    layers.MaxPool2D(pool_size=(2,2)),
    layers.Flatten(),
    layers.Dense(32, activation='relu'),
    layers.Dropout(0.2),
    layers.Dense(10, activation='softmax')

])
```

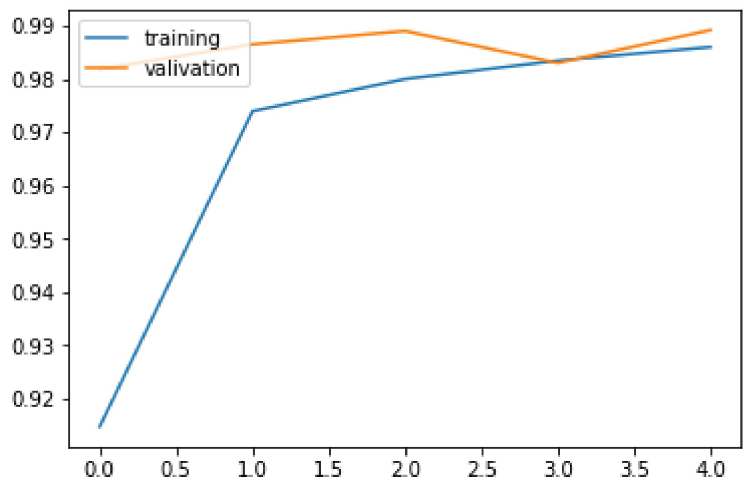
In [17]:
```
deep_model.compile(optimizer=keras.optimizers.Adam(),
             loss=keras.losses.SparseCategoricalCrossentropy(),
            metrics=['accuracy'])
deep_model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_8 (Conv2D)            (None, 28, 28, 32)        320       
_________________________________________________________________
batch_normalization_v2_4 (Ba (None, 28, 28, 32)        128       
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 16)        528       
_________________________________________________________________
batch_normalization_v2_5 (Ba (None, 28, 28, 16)        64        
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 14, 14, 16)        0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 14, 14, 32)        4640      
_________________________________________________________________
batch_normalization_v2_6 (Ba (None, 14, 14, 32)        128       
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 14, 14, 16)        528       
_________________________________________________________________
batch_normalization_v2_7 (Ba (None, 14, 14, 16)        64        
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 7, 7, 16)          0         
_________________________________________________________________
flatten_2 (Flatten)          (None, 784)               0         
_________________________________________________________________
dense_4 (Dense)              (None, 32)                25120     
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_5 (Dense)              (None, 10)                330       
=================================================================
Total params: 31,850
Trainable params: 31,658
Non-trainable params: 192
_________________________________________________________________
```

In [18]:
```
history = deep_model.fit(x_train, y_train, batch_size=64, epochs=5, validation_sp
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/5
54000/54000 [==============================] - 62s 1ms/sample - loss: 0.2729 -
accuracy: 0.9147 - val_loss: 0.0657 - val_accuracy: 0.9818
Epoch 2/5
54000/54000 [==============================] - 63s 1ms/sample - loss: 0.0872 -
accuracy: 0.9739 - val_loss: 0.0437 - val_accuracy: 0.9865
Epoch 3/5
54000/54000 [==============================] - 59s 1ms/sample - loss: 0.0657 -
accuracy: 0.9800 - val_loss: 0.0404 - val_accuracy: 0.9890
Epoch 4/5
54000/54000 [==============================] - 49s 913us/sample - loss: 0.0535
- accuracy: 0.9834 - val_loss: 0.0622 - val_accuracy: 0.9830
Epoch 5/5
54000/54000 [==============================] - 49s 913us/sample - loss: 0.0441
- accuracy: 0.9860 - val_loss: 0.0435 - val_accuracy: 0.9892
```

In [19]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.legend(['training', 'valivation'], loc='upper left')
plt.show()
```



In [20]:
```
result = deep_model.evaluate(x_test, y_test)
```

```
10000/10000 [==============================] - 2s 196us/sample - loss: 0.0335 -
accuracy: 0.9887
```