機器學習概論作業

範圍： Implementing CNN model using Keras

銘傳大學電腦與通訊工程系

班　　級	電通三乙
姓　　名	李柏賢
學　　號	07050862
作業成果	應繳作業共 1 題，每題 100 分 我共完成 1 題，應得 90 分
授課教師	陳慶逸

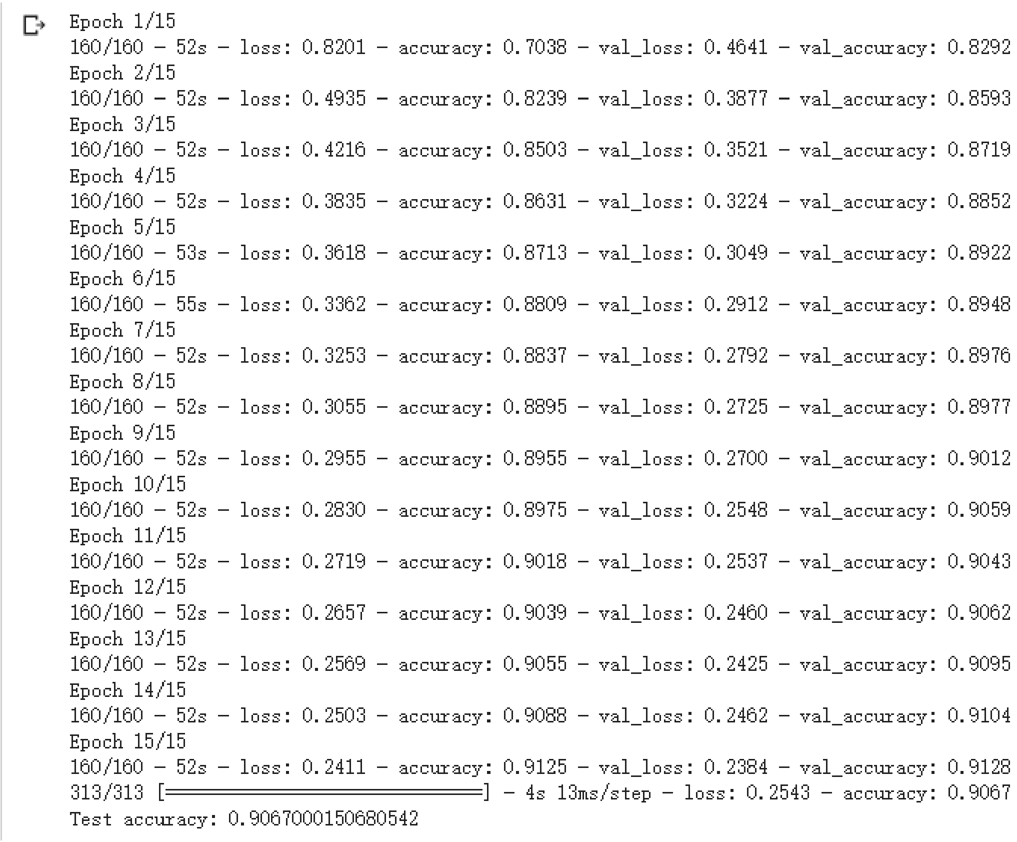
■ 請確實填寫自己寫完成題數，填寫不實者(如上傳與作業明顯無關的答案，或是計算題數有誤者)，本次作業先扣 50 分。

一、 試建立一個 CNN 模型來實現 Fashion-MNIST 的辨識:

程式碼:

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten,Conv2D,MaxPooling2D


fashion_mnist = keras.datasets.fashion_mnist
# Read fashion_mnist data
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()


# Translation of data
train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
test_images = test_images.reshape(test_images.shape[0], 28, 28, 1).astype('float32')
# Standardize feature data
train_images = train_images / 255.0
test_images = test_images / 255.0
# Label Onehot-encoding
y_TrainOneHot = np_utils.to_categorical(train_labels)
y_TestOneHot = np_utils.to_categorical(test_labels)




model = Sequential()
# Create CN layer 1
model.add(Conv2D(filters=16,
                 kernel_size=(5,5),
                 padding='same',
                 input_shape=(28,28,1),
                 activation='relu'))
# Create Max-Pool 1
model.add(MaxPooling2D(pool_size=(2,2)))

# Create CN layer 2
model.add(Conv2D(filters=36,
                 kernel_size=(5,5),
                 padding='same',
                 input_shape=(28,28,1),
                 activation='relu'))
# Create Max-Pool 2
model.add(MaxPooling2D(pool_size=(2,2)))

# Add Dropout layer
model.add(Dropout(0.25))

model.add(Flatten())
#建立 Hidden layer
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
#建立輸出層
model.add(Dense(10, activation='softmax'))

# 定義訓練方式
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=[
'accuracy'])

# 開始訓練
train_history = model.fit(x=train_images,
                          y=y_TrainOneHot, validation_split=0.2,
                          epochs=15, batch_size=300, verbose=2)




test_loss, test_acc = model.evaluate(test_images, y_TestOneHot)

print('Test accuracy:', test_acc)
```

輸出結果擷圖:



```
Epoch 1/15
160/160 - 52s - loss: 0.8201 - accuracy: 0.7038 - val_loss: 0.4641 - val_accuracy: 0.8292
Epoch 2/15
160/160 - 52s - loss: 0.4935 - accuracy: 0.8239 - val_loss: 0.3877 - val_accuracy: 0.8593
Epoch 3/15
160/160 - 52s - loss: 0.4216 - accuracy: 0.8503 - val_loss: 0.3521 - val_accuracy: 0.8719
Epoch 4/15
160/160 - 52s - loss: 0.3835 - accuracy: 0.8631 - val_loss: 0.3224 - val_accuracy: 0.8852
Epoch 5/15
160/160 - 53s - loss: 0.3618 - accuracy: 0.8713 - val_loss: 0.3049 - val_accuracy: 0.8922
Epoch 6/15
160/160 - 55s - loss: 0.3362 - accuracy: 0.8809 - val_loss: 0.2912 - val_accuracy: 0.8948
Epoch 7/15
160/160 - 52s - loss: 0.3253 - accuracy: 0.8837 - val_loss: 0.2792 - val_accuracy: 0.8976
Epoch 8/15
160/160 - 52s - loss: 0.3055 - accuracy: 0.8895 - val_loss: 0.2725 - val_accuracy: 0.8977
Epoch 9/15
160/160 - 52s - loss: 0.2955 - accuracy: 0.8955 - val_loss: 0.2700 - val_accuracy: 0.9012
Epoch 10/15
160/160 - 52s - loss: 0.2830 - accuracy: 0.8975 - val_loss: 0.2548 - val_accuracy: 0.9059
Epoch 11/15
160/160 - 52s - loss: 0.2719 - accuracy: 0.9018 - val_loss: 0.2537 - val_accuracy: 0.9043
Epoch 12/15
160/160 - 52s - loss: 0.2657 - accuracy: 0.9039 - val_loss: 0.2460 - val_accuracy: 0.9062
Epoch 13/15
160/160 - 52s - loss: 0.2569 - accuracy: 0.9055 - val_loss: 0.2425 - val_accuracy: 0.9095
Epoch 14/15
160/160 - 52s - loss: 0.2503 - accuracy: 0.9088 - val_loss: 0.2462 - val_accuracy: 0.9104
Epoch 15/15
160/160 - 52s - loss: 0.2411 - accuracy: 0.9125 - val_loss: 0.2384 - val_accuracy: 0.9128
313/313 [==============================] - 4s 13ms/step - loss: 0.2543 - accuracy: 0.9067
Test accuracy: 0.9067000150680542
```