

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 <u>1</u> 題，每題 <u>100</u> 分 我共完成 <u>1</u> 題，應得 <u>94</u> 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 VGG 模型來實現 Fashion-MNIST 的辨識:

程式碼:

```
from keras.optimizers import SGD
import numpy as np
import keras
from keras.datasets import mnist
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Dense, Activation, Conv2D, MaxPooling2D, Flatten, Dropout
from keras.optimizers import Adam

# load train and test dataset

# load dataset
(trainX, trainY), (testX, testY) = fashion_mnist.load_data()
# reshape dataset to have a single channel
trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))
testX = testX.reshape((testX.shape[0], 28, 28, 1))
# one hot encode target values
trainY = to_categorical(trainY)
testY = to_categorical(testY)

# scale pixels
# convert from integers to floats
train_norm = trainX.astype('float32')
test_norm = testX.astype('float32')
# normalize to range 0-1
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0

model = Sequential()
model.add(Conv2D(32, (3, 3), strides=(1, 1), input_shape=(28, 28, 1), padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2)))
```

```

model.add(Conv2D(64, (3, 2), strides=(1, 1), padding='same', activation='relu'))
model.add(Conv2D(64, (3, 3), strides=(1, 1), padding='same', activation='relu'))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))
# compile model
opt = SGD(lr=0.01, momentum=0.9)
model.compile(optimizer=Adam(lr=0.001, beta_1=0.7, beta_2=0.999, epsilon=1e-
8), loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(train_norm, trainY, epochs=10, batch_size=32, validation_data=(test_norm, testY), verbose=1)
test_loss, test_acc = model.evaluate(test_norm, testY)
print('Test accuracy:', test_acc)

```

輸出結果擷圖：

```

Epoch 1/10
1875/1875 [=====] - 136s 72ms/step - loss: 0.4899 - accuracy: 0.8261 - val_loss: 0.3168 - val_accuracy: 0.8811
Epoch 2/10
1875/1875 [=====] - 135s 72ms/step - loss: 0.3206 - accuracy: 0.8879 - val_loss: 0.2720 - val_accuracy: 0.8971
Epoch 3/10
1875/1875 [=====] - 135s 72ms/step - loss: 0.2714 - accuracy: 0.9033 - val_loss: 0.2539 - val_accuracy: 0.9052
Epoch 4/10
1875/1875 [=====] - 135s 72ms/step - loss: 0.2413 - accuracy: 0.9140 - val_loss: 0.2437 - val_accuracy: 0.9105
Epoch 5/10
1875/1875 [=====] - 135s 72ms/step - loss: 0.2172 - accuracy: 0.9210 - val_loss: 0.2365 - val_accuracy: 0.9132
Epoch 6/10
1875/1875 [=====] - 134s 72ms/step - loss: 0.1985 - accuracy: 0.9277 - val_loss: 0.2326 - val_accuracy: 0.9168
Epoch 7/10
1875/1875 [=====] - 136s 72ms/step - loss: 0.1808 - accuracy: 0.9342 - val_loss: 0.2329 - val_accuracy: 0.9196
Epoch 8/10
1875/1875 [=====] - 136s 72ms/step - loss: 0.1663 - accuracy: 0.9388 - val_loss: 0.2275 - val_accuracy: 0.9206
Epoch 9/10
1875/1875 [=====] - 137s 73ms/step - loss: 0.1522 - accuracy: 0.9431 - val_loss: 0.2315 - val_accuracy: 0.9222
Epoch 10/10
1724/1875 [=====>...] - ETA: 10s - loss: 0.1429 - accuracy: 0.9472

```