

第十章

Keras捲積神經網路(CNN)辨識
Cifar-10影像



前言

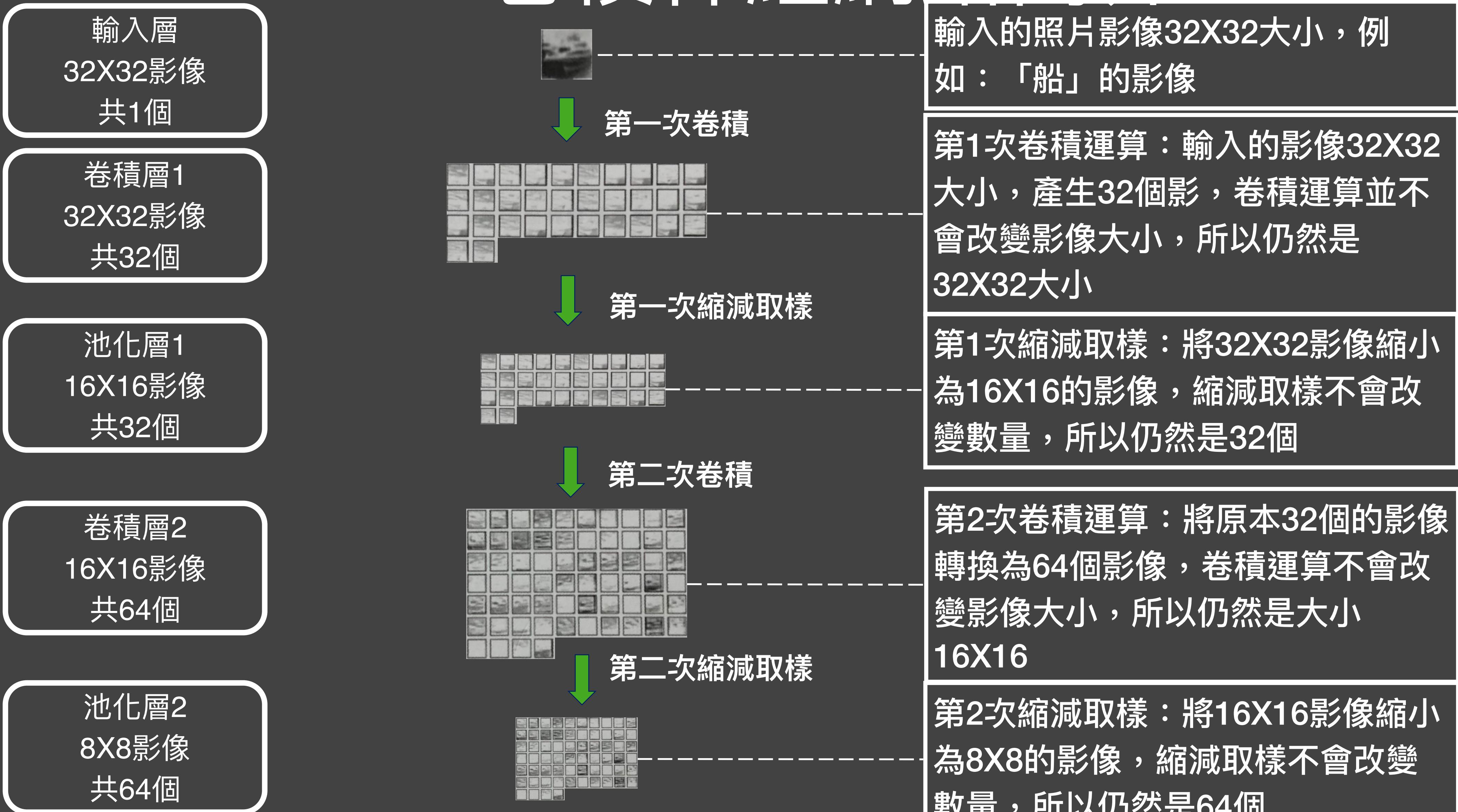
本章我們將介紹，使用Keras建立卷積神經網路CNN(convolutional neural network)模型，並且訓練型、評估模型準確率，然後使用訓練完成的模型，辨識Cifar-10影像資料集。

由於CIFAR-10影像辨識的難度，比起MNIST資料集高很多。所以我們嘗試更多次的卷積與池化運算，來提高辨識的準確率。

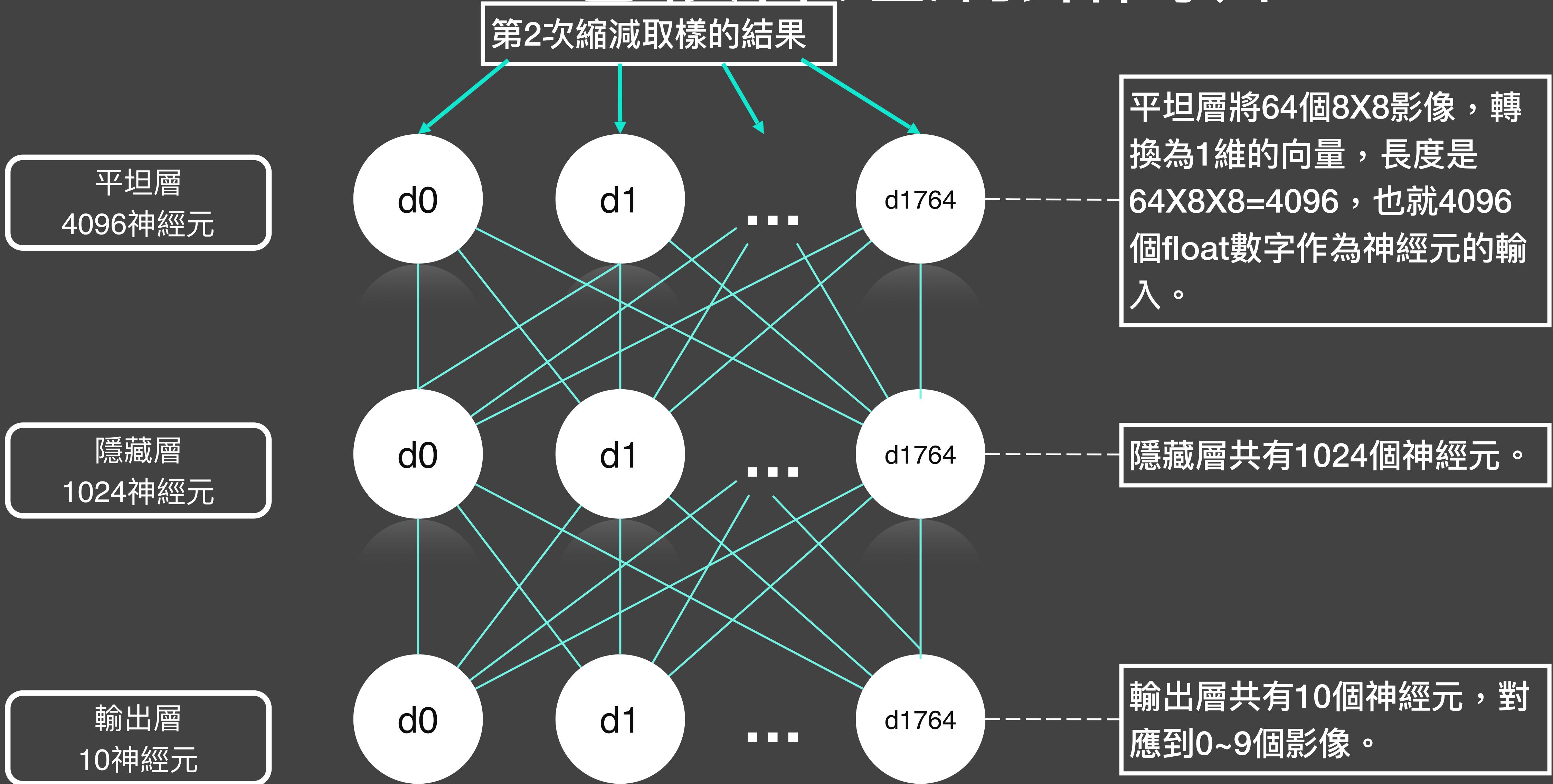
10.1

CNN捲積神經網路簡介

10.1 CNN卷積神經網路簡介



10.1 CNN卷積神經網路簡介



10.1 CNN卷積神經網路簡介

從前面的投影片我們可以看到CNN卷積神經網路，可分為2大部分：

- 影像的特徵提取：透過卷積層1、池化層1、卷積層2、池化層2的處理，提取影像的特徵。
- 完全連結神經網路fully connected layer：包含平坦層、隱藏層、輸出層，所組成的類神經網路。

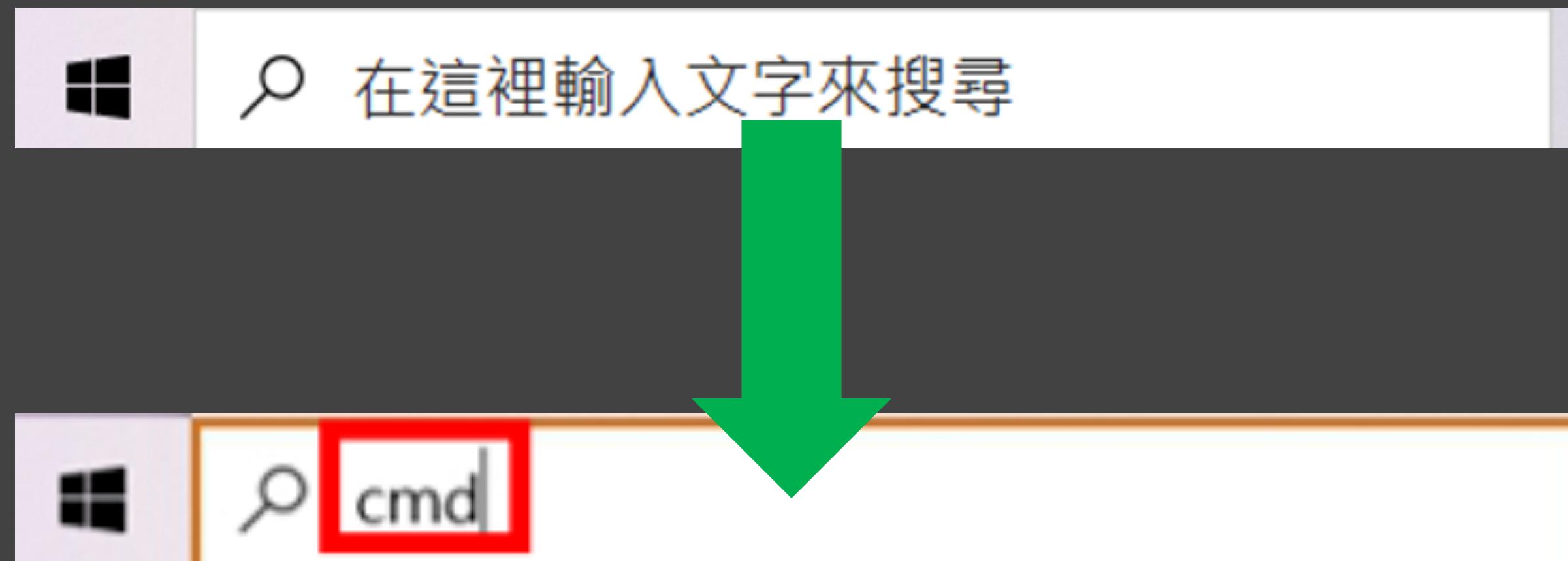
10.2

進行資料預處理(Preprocess)

10.2 進行資料預處理(Preprocess)

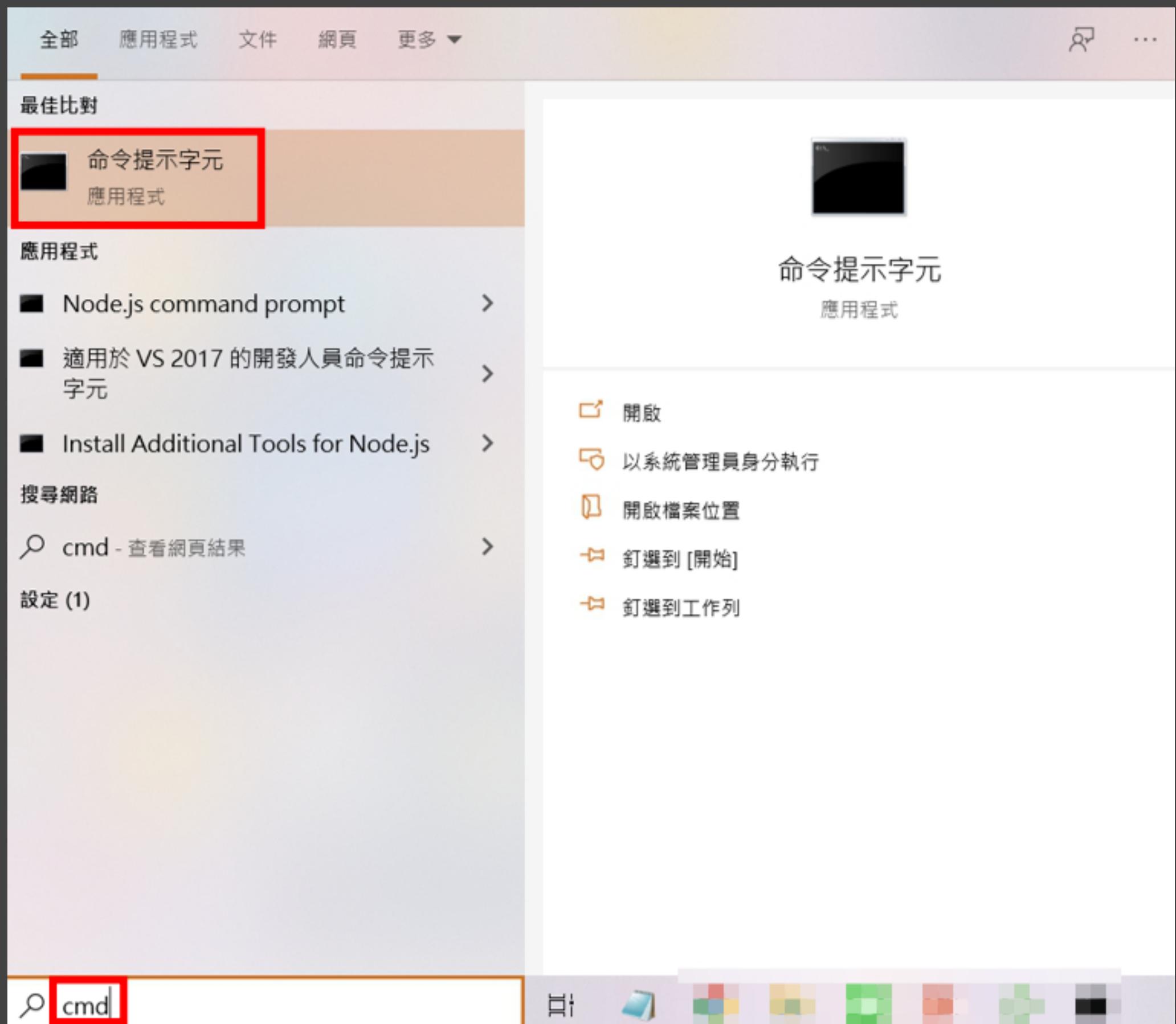
- 開啟命令提示字元

1. 左鍵單擊電腦工具列的搜尋欄位，並在當中輸入cmd。



10.2 進行資料預處理(Preprocess)

2. 以左鍵單擊的方式開啟搜尋結果中的命令提示字元。



10.2 進行資料預處理(Preprocess)

- 開啟Jupyter NoteBook

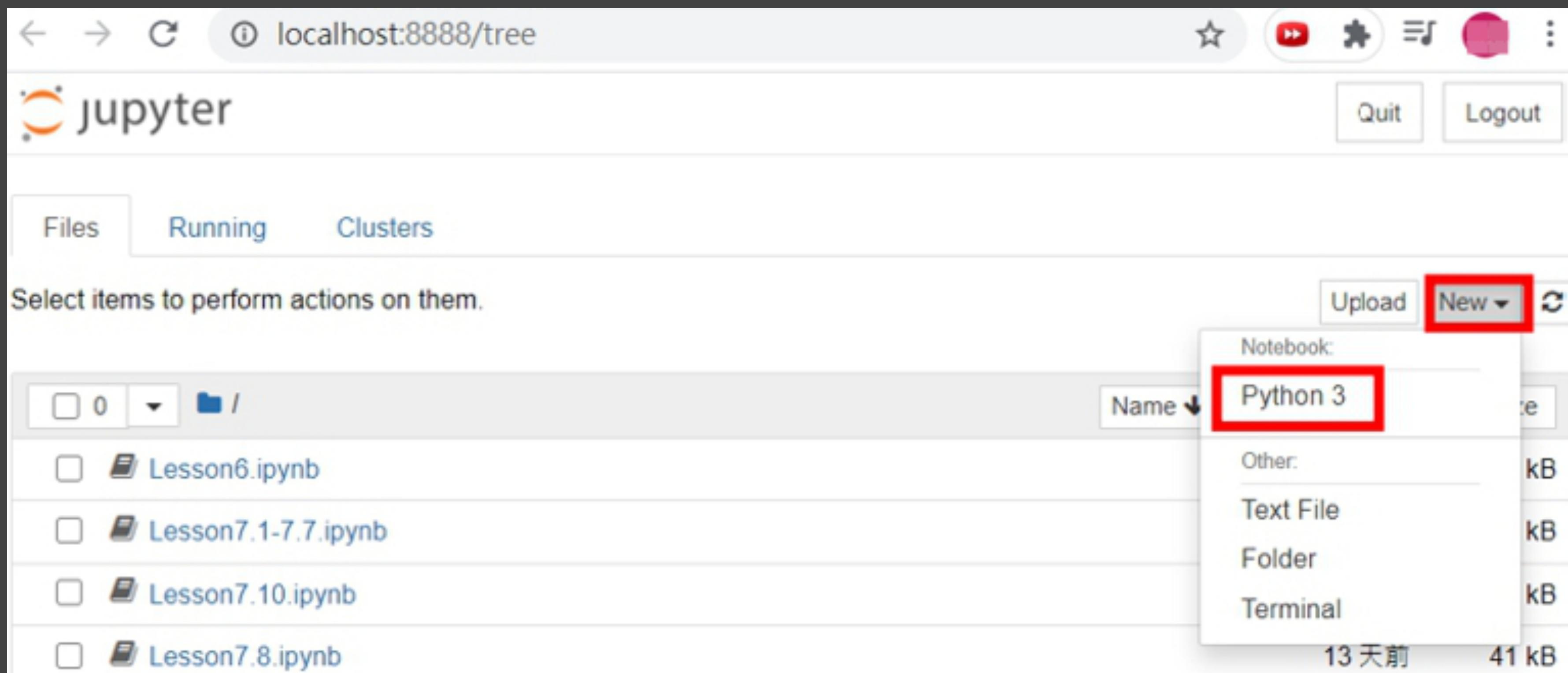
3. 在命令列輸入「jupyter notebook」接著按下Enter鍵。

```
C:\pythonwork:jupyter notebook
[I 18:43:38.680 NotebookApp] JupyterLab extension loaded from C:\Users\nan\anaconda3\lib\site-packages\jupyterlab
[I 18:43:38.686 NotebookApp] JupyterLab application directory is C:\Users\nan\anaconda3\share\jupyter\lab
[I 18:43:38.696 NotebookApp] Serving notebooks from local directory: C:\pythonwork
[I 18:43:38.697 NotebookApp] Jupyter Notebook 6.1.4 is running at:
[I 18:43:38.699 NotebookApp] http://localhost:8888/?token=cbc96ad8661f78ed3dc3073404088a3affce32ed4c84242e
[I 18:43:38.702 NotebookApp] or http://127.0.0.1:8888/?token=cbc96ad8661f78ed3dc3073404088a3affce32ed4c84242e
[I 18:43:38.702 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 18:43:38.911 NotebookApp]
```

10.2 進行資料預處理(Preprocess)

- 創建新的Python 3檔案

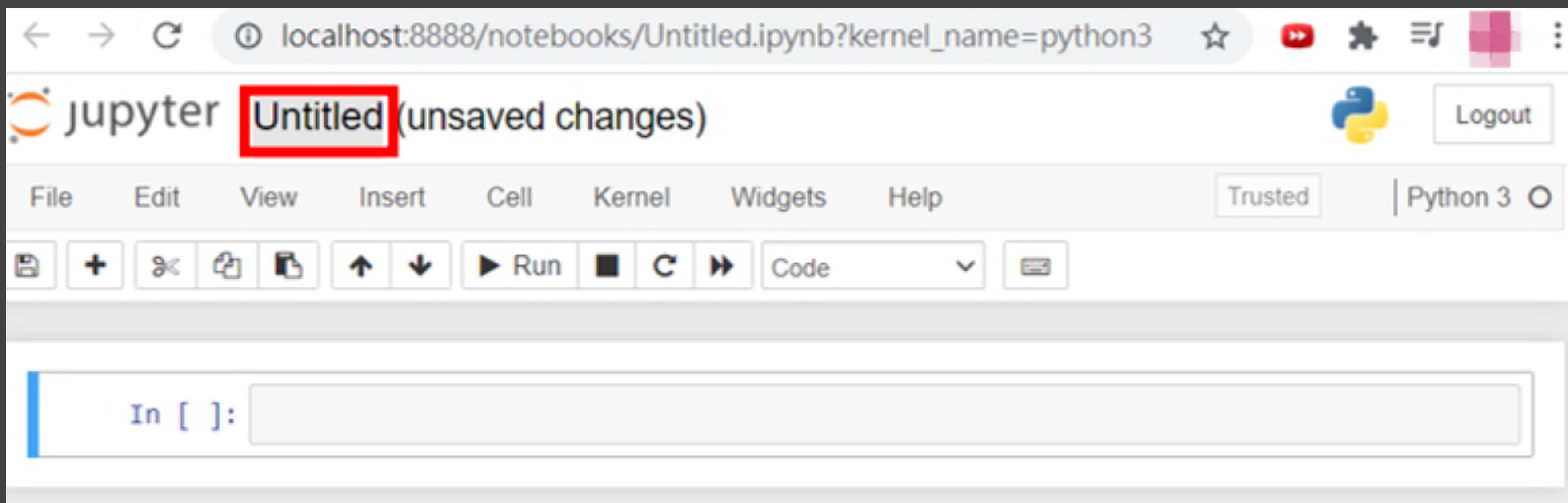
4. 左鍵單擊畫面右上方的New按鈕，並在出現的選單中左鍵單擊Python3。



10.2 進行資料預處理(Preprocess)

- 更改新檔案的名稱

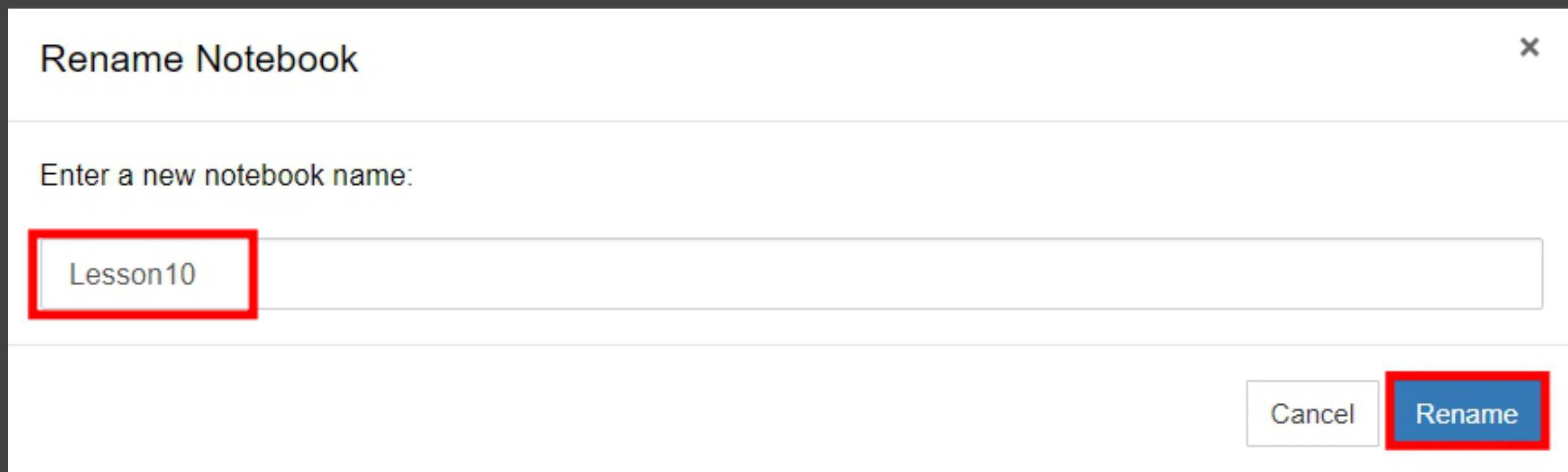
5. 左鍵單擊Untitled(圖中紅框處)。



10.2 進行資料預處理(Preprocess)

- 開啟Jupyter NoteBook

6. 在輸入框輸入Lesson 10，並且左鍵單擊Rename按鈕。



10.2 進行資料預處理(Preprocess)

- 決入所需模組

7. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
from keras.datasets import cifar10
import numpy as np
np.random.seed(10)
```

從keras.datasets匯入
cifar10資料集

匯入numpy模組，NumPy是
Python語言的擴充程式庫。支
援維度陣列與矩陣運算。

設定seed可以讓每次需要隨機
產生的資料，都有相同的輸出。

10.2 進行資料預處理(Preprocess)

- 讀取cifar10資料

8. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
(x_img_train,y_label_train),  
(x_img_test,y_label_test)=cifar10.load_data()
```

10.2 進行資料預處理(Preprocess)

- 顯示訓練與驗證資料的shape

9. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
print("train data:",'images:',x_img_train.shape,
      " labels:",y_label_train.shape)
print("test  data:",'images:',x_img_test.shape ,
      " labels:",y_label_test.shape)
```

```
train data: images: (50000, 32, 32, 3)  labels: (50000, 1)
test  data: images: (10000, 32, 32, 3)  labels: (10000, 1)
```

10.2 進行資料預處理(Preprocess)

- 將features(照片影像特徵值)標準化：可提高模型預測的準確度，並且更快收斂。

10. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
x_img_train_normalize = x_img_train.astype('float32') / 255.0  
x_img_test_normalize = x_img_test.astype('float32') / 255.0
```

10.2 進行資料預處理(Preprocess)

- label(照片影像的真實的值)以Onehot encoding轉換

11. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
from keras.utils import np_utils
y_label_train_OneHot = np_utils.to_categorical(y_label_train)
y_label_test_OneHot = np_utils.to_categorical(y_label_test)
```

10.3

建立模型

10.3 建立模型

- 汇入所需模組

1. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
from keras.models import Sequential  
from keras.layers import Dense, Dropout, Activation, Flatten  
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D
```

匯入keras的
Sequential模組

匯入keras的
layers模組

匯入keras的
layers模組

10.3 建立模型

- 建立keras的Sequential模型：建立一個Sequential 線性堆疊模型，後續只需要將各神經網路層加入模型即可。

2. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model = Sequential()
```

10.3 建立模型

- 建立卷積層1

3. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(Conv2D(filters=32,kernel_size=(3,3),  
                 input_shape=(32, 32,3),  
                 activation='relu',  
                 padding='same'))
```

程式碼	說明
<code>filters=32</code>	設定隨機產生3個濾鏡filter weight
<code>kernel_size=(3,3)</code>	每一個濾鏡3X3大小
<code>input_shape=(32, 32,3)</code>	此設定讓卷積運算，產生的卷積影像大小不變
<code>activation='relu'</code>	設定ReLU激活函數
<code>padding='same'</code>	第1,2維度代表輸入的影像形狀32X32大小,第3個維度：因為是彩色代表RGB三原色是3

10.3 建立模型

- 加入Dropout避免overfitting

4. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(Dropout(rate=0.25))
```

加入Dropout層至模型中。
Dropout(0.25)的功能是，每次
訓練迭代時，會隨機地在神經網
路中放棄25%的神經元，以避免
overfitting。

10.3 建立模型

- 建立池化層1

5. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

建立池化層1，輸入參數
pool_size=(2, 2)，執行第1次縮減取樣，將32X32影像，縮小為16X16的影像，縮減取樣不會改變數量，所以仍然是32個。

10.3 建立模型

- 建立卷積層2

6. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(Conv2D(filters=64, kernel_size=(3, 3),  
activation='relu', padding='same'))
```

程式碼	說明
filters=64	建立64個濾鏡filter weight
kernel_size=(3,3)	每一個濾鏡3X3大小
activation='relu'	設定ReLU激活函數
padding='same'	此設定讓卷積運算並不會改變大小

執行第2次卷積運算：將原本32個的影像轉換為64個影像，卷積運算不會改變影像大小，所以仍然是大小16X16。

10.3 建立模型

- 加入Dropout避免overfitting

7. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(Dropout(0.25))
```

Dropout(0.25)的功能是，每次訓練迭代時，會隨機地在神經網路中放棄25%的神經元，以避免overfitting。

10.3 建立模型

- 建立池化層2

8. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(MaxPooling2D(pool_size=(2, 2)))
```

建立池化層2，輸入參數
pool_size=(2, 2)，執行第2次縮減取樣，將16X16影像，縮小為8X8的影像，縮減取樣不會改變數量，所以仍然是64個。

10.3 建立模型

- 建立平坦層

9. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(Flatten())
model.add(Dropout(rate=0.25))
```

此段程式碼建立平坦層，將之前的步驟建立的「池化層2」的64個8X8影像轉換為1維的向量，長度是 $64 \times 8 \times 8 = 4096$ ，也就是4096個float數字，正好對應到4096神經元。並且加入 Dropout(0.25)，每次訓練迭代時，會隨機地在神經網路放棄25%的神經元，以避免overfitting。

10.3 建立模型

- 建立隱藏層

10. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(Dense(1024, activation='relu'))  
model.add(Dropout(rate=0.25))
```

此段程式碼建立隱藏層，共有1024個神經元。並且加入Dropout(0.25)，每次訓練迭代時，會隨機地在神經網路放棄25%的神經元，以避免overfitting。

10.3 建立模型

- 建立輸出層

11. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
model.add(Dense(10, activation='softmax'))
```

共有10個神經元，對應到0~9共10個影像類別。並且使用softmax激活函數進行轉換，softmax可以將神經元的輸出，轉換為預測每一個影像類別的機率。

10.3 建立模型

- 查看模型摘要

12. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
print(model.summary())
```

10.3 建立模型

- 輸出結果說明

In [17]: `print(model.summary())`

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 32, 32, 32)	896
dropout_1 (Dropout)	(None, 32, 32, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 32)	0
conv2d_2 (Conv2D)	(None, 16, 16, 64)	18496
dropout_2 (Dropout)	(None, 16, 16, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 8, 8, 64)	0
flatten_1 (Flatten)	(None, 4096)	0
dropout_3 (Dropout)	(None, 4096)	0
dense_1 (Dense)	(None, 1024)	4195328
dropout_4 (Dropout)	(None, 1024)	0
dense_2 (Dense)	(None, 10)	10250

Total params: 4,224,970
Trainable params: 4,224,970
Non-trainable params: 0

None

捲積層1 & 池化層1

捲積層2 & 池化層2

神經網路層
(平坦、隱藏、輸出層)

10.4

進行訓練

10.4 進行訓練

- 定義訓練方式：使用**compile**方法，對模型進行訓練

1. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

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

compile方法須輸入下列參數：

- loss**：設定損失函數(**loss function**)，在深度學習通常使用**cross_entropy**交叉熵，訓練的效果比較好。
- optimizer**：設定訓練時的最優化方法，在深學習使用**adam**最優化方法，可以讓運練更快收斂，並提高準確率。
- metrics**:設定評估模型的方式是**accuracy**準確率。

10.4 進行訓練

- # • 開始訓練

2. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

10.4 進行訓練

• 程式碼說明

```
train_history=model.fit(x_img_train_normalize, y_label_train_OneHot,  
validation_split=0.2,  
epochs=10, batch_size=128, verbose=1)
```

使用model.fit訓練，訓練過程會儲存在train_history變數，並且輸入下列參數：

➤ 輸入訓練資料參數

- x=x_img_train_normalize(feature數字影像的特徵值，經過標準化處理)
- y=y_label_train_OneHot(label相片影像真實的值，經過OneHot encoding轉換)

➤ 設定訓練與驗證資料比例

- 設定參數validation_split=0.2：訓練之前Keras會自動將資料80%分為訓練資料、20%分為驗證資料

➤ 設定epoch(訓練週期)次數與每一批次筆數

- epochs=10：執行10次訓練週期
- batch_size=128：每一批次128筆資料

➤ 設定顯示訓練過程

- verbose=2：顯示訓練過程。

10.4 進行訓練

• 輸出結果說明

```
train_history=model.fit(x_img_train_normalize, y_label_train_OneHot,
                       validation_split=0.2,
                       epochs=10, batch_size=128, verbose=1)

Epoch 1/10
313/313 [=====] - 146s 462ms/step - loss: 1.8148 - accuracy: 0.3517 - val_loss: 1.3040 - val_accuracy: 0.5690
Epoch 2/10
313/313 [=====] - 133s 426ms/step - loss: 1.1912 - accuracy: 0.5789 - val_loss: 1.1258 - val_accuracy: 0.6432
Epoch 3/10
313/313 [=====] - 143s 456ms/step - loss: 1.0126 - accuracy: 0.6410 - val_loss: 1.0177 - val_accuracy: 0.6757
Epoch 4/10
313/313 [=====] - 142s 454ms/step - loss: 0.9011 - accuracy: 0.6806 - val_loss: 0.9249 - val_accuracy: 0.7054
Epoch 5/10
313/313 [=====] - 144s 461ms/step - loss: 0.7936 - accuracy: 0.7218 - val_loss: 0.8692 - val_accuracy: 0.7183
Epoch 6/10
313/313 [=====] - 140s 448ms/step - loss: 0.7087 - accuracy: 0.7499 - val_loss: 0.8353 - val_accuracy: 0.7247
Epoch 7/10
313/313 [=====] - 148s 473ms/step - loss: 0.6290 - accuracy: 0.7835 - val_loss: 0.7733 - val_accuracy: 0.7464
Epoch 8/10
313/313 [=====] - 143s 457ms/step - loss: 0.5488 - accuracy: 0.8090 - val_loss: 0.7915 - val_accuracy: 0.7438
Epoch 9/10
313/313 [=====] - 143s 457ms/step - loss: 0.4859 - accuracy: 0.8290 - val_loss: 0.7430 - val_accuracy: 0.7521
Epoch 10/10
313/313 [=====] - 139s 444ms/step - loss: 0.4311 - accuracy: 0.8494 - val_loss: 0.7637 - val_accuracy: 0.7383
```

以上程式碼共執行10次 epoch，且每一次都執行：

- ✓ 使用40000筆訓練資料進行訓練，分為每一批次128筆，所以大約分成160批次(40000/128=313)進行訓練
- ✓ Epoch(訓練週期)訓練完成後會計算此次訓練週期的accuracy(準確率)與loss(誤差)，並且新增一筆資料在 train_history 中

10.4 進行訓練

- 使用之前建立的show_train_history函數

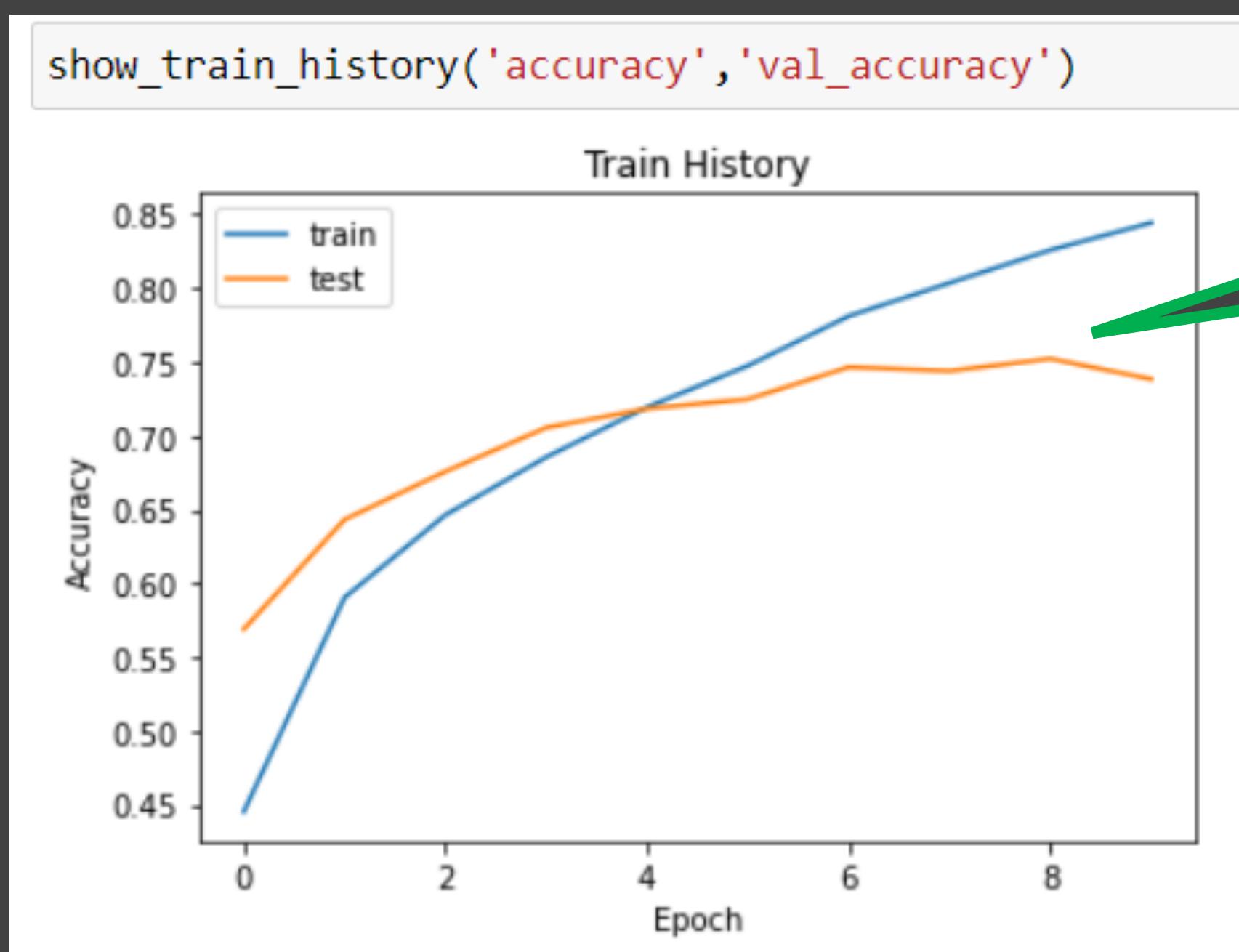
3. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
import matplotlib.pyplot as plt
def show_train_history(train_acc,test_acc):
    plt.plot(train_history.history[train_acc])
    plt.plot(train_history.history[test_acc])
    plt.title('Train History')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```

10.4 進行訓練

- 使用show_train_history函數畫出accuracy執行結果

4. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。



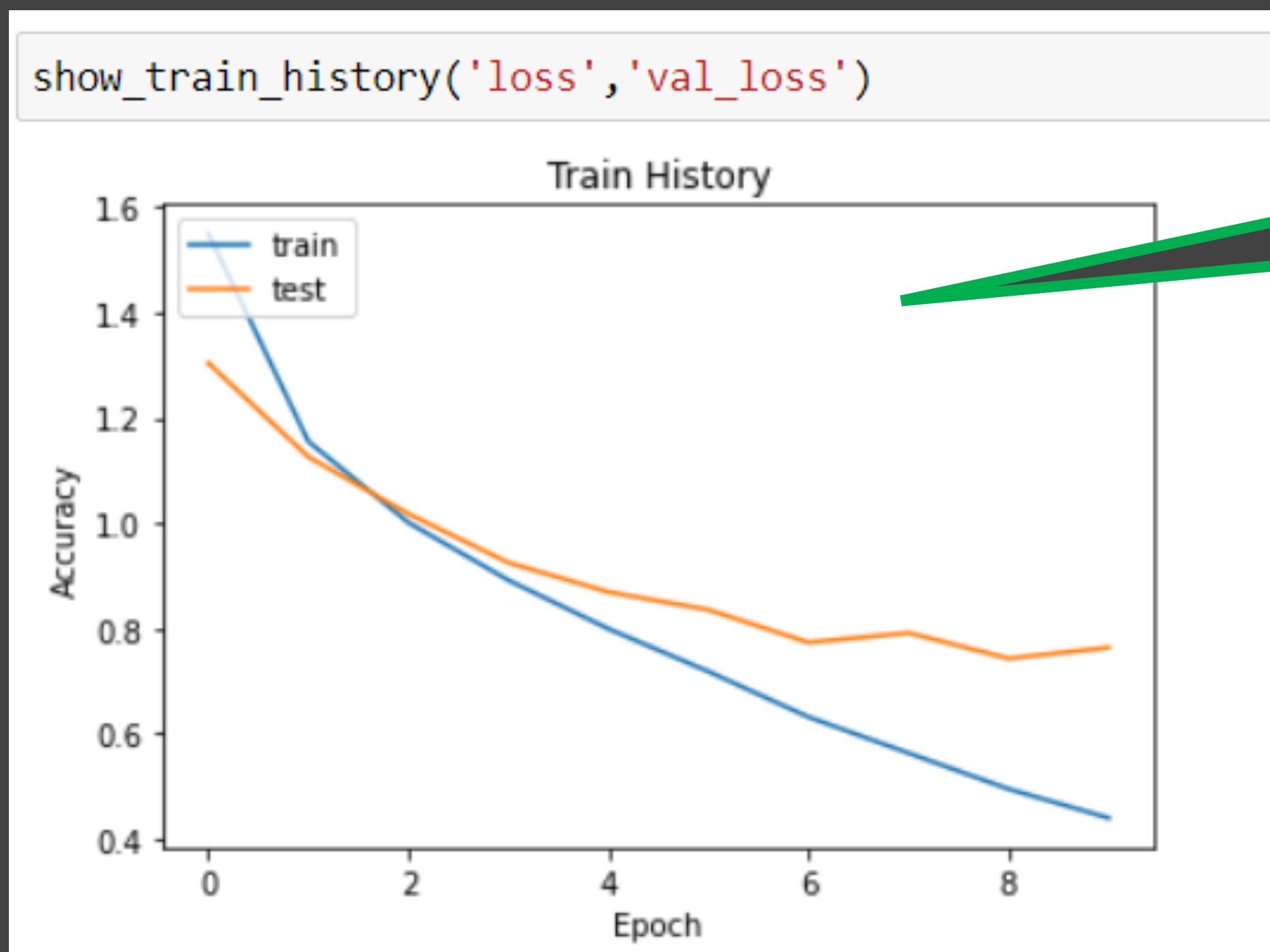
「accuracy訓練的準確率」(藍色)
&
「val_accuracy訓練的準確率」(橘色)

由結果可以發現：
✓ 不論訓練與驗證，準確率都越來越高
✓ 在Epoch訓練後期「accuracy訓練的準確率」比「val_accuracy訓練的準確率」高

10.4 進行訓練

- 使用show_train_history函數畫出loss誤差執行結果

5. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。



「loss訓練的準確率」(藍色)
&
「val_loss訓練的準確率」(橘色)

由結果可以發現：
✓ 不論訓練與驗證，驗證的誤差都越來越低
✓ 在Epoch訓練後期 「loss訓練的誤差」 比
「 val_loss驗證的誤差」 小

10.5

評估模型準確率

10.5 評估模型準確率

- 評估模型準確率

1. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
scores = model.evaluate(x_img_test_normalize,  
                      y_label_test_OneHot, verbose=0)  
scores[1]  
  
0.7329999804496765
```

程式碼	說明
<code>scores = model.evaluate(</code>	使用 <code>model.evaluate</code> 進行評估模型準確率，評估後的準確率，會儲存在 <code>scores</code>
<code>x_img_test_normalize,</code>	測試資料的 <code>features</code> (照片影像的特徵值，經過標準化處理)
<code>y_label_test_OneHot</code>	測試資料的 <code>label</code> (照片影像真實的值，經過One-hot encoding轉換)

10.6

進行預測

10.6 進行預測

- 執行預測

1. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
prediction=model.predict_classes(x_img_test_normalize)
```

使用**model.predict_classes**，輸入參數
x_Test(測試資料的照片影像)，進行預測。

10.6 進行預測

- 預測結果：查看預測結果的前10筆資料

2. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
prediction[:10]  
array([3, 8, 8, 0, 6, 6, 1, 6, 3, 1], dtype=int64)
```

10.6 進行預測

- 使用之前建立的plot_images_labels_prediction函數

3. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
import matplotlib.pyplot as plt
def plot_images_labels_prediction(images,labels,prediction,
                                   idx,num=10):
    fig = plt.gcf()
    fig.set_size_inches(12, 14)
    if num>25: num=25
    for i in range(0, num):
        ax=plt.subplot(5,5, 1+i)
        ax.imshow(images[idx],cmap='binary')

        title=str(i)+','+label_dict[labels[i][0]]
        if len(prediction)>0:
            title+='=>'+label_dict[prediction[i]]

        ax.set_title(title,fontsize=10)
        ax.set_xticks([]);ax.set_yticks([])
        idx+=1
    plt.show()
```

10.6 進行預測

- 顯示前10筆預測結果

4. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
plot_images_labels_prediction(x_img_test,y_label_test,  
                             prediction,0,10)
```

The code above is a function call to plot 10 images from the test set. Each image is labeled with its index (0-9), the true label, and the predicted label. The images show various objects: a cat, a ship, an airplane, a frog, another frog, an automobile, and another cat. The images are somewhat blurry or low-quality.

Index	True Label	Predicted Label
0	cat	cat
1	ship	ship
2	ship	ship
3	airplane	airplane
4	frog	frog
5	frog	frog
6	automobile	automobile
7	frog	frog
8	cat	cat
9	automobile	automobile

10.7

查看預測機率

10.7 查看預測機率

- 使用model.predict輸入預測資料，以預測機率

1. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
Predicted_Probability=model.predict(x_img_test_normalize)
```

10.7 查看預測機率

- 建立show_Predicted_Probability函數

2. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
def show_Predicted_Probability(y,prediction,
                                x_img,Predicted_Probability,i):
    print('label:',label_dict[y[i][0]],
          'predict:',label_dict[prediction[i]])
    plt.figure(figsize=(2,2))
    plt.imshow(np.reshape(x_img_test[i],(32, 32,3)))
    plt.show()
    for j in range(10):
        print(label_dict[j]+
              ' Probability:%1.9f'%(Predicted_Probability[i][j]))
```

10.7 查看預測機率

- 程式碼說明

定義**show_Predicted_Probability**函數。傳入下列參數：
y(真實值)、**prediction(預測結果)**、**x_img(預測的影像)**、
Predicted_Probability(預測機率)、**i(開始顯示的資料
index)**。

```
def show_Predicted_Probability(y,prediction,  
                                x_img,Predicted_Probability,i):  
{  
    print('label:',label_dict[y[i][0]],  
         'predict:',label_dict[prediction[i]])  
    plt.figure(figsize=(2,2))  
    plt.imshow(np.reshape(x_img_test[i],(32, 32,3)))  
    plt.show()  
    for j in range(10):  
        print(label_dict[j]+  
              ' Probability:%1.9f '%(Predicted_Probability[i][j]))
```

設定顯示影像
的大小，並且
畫出照片影像

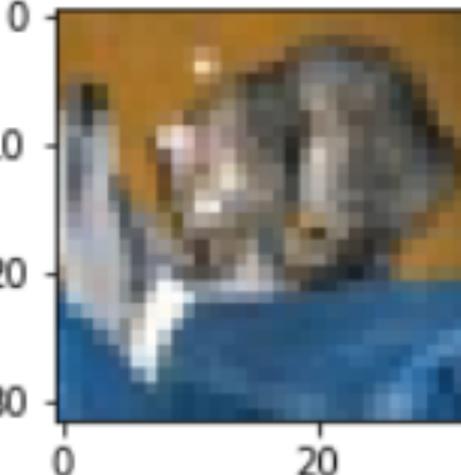
顯示**y(真實值)**與
**prediction(預測
結果)**

使用**for**迴圈，讀取
Predicted_Probability顯
示預測機率

10.7 查看預測機率

- 查看第0筆資料預測的機率

3. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
show_Predicted_Probability(y_label_test,prediction,  
                           x_img_test,Predicted_Probability,0)  
  
label: cat predict: cat  
  
  
  
airplane Probability:0.009049814  
automobile Probability:0.004270773  
bird Probability:0.009257487  
cat Probability:0.667897582  
deer Probability:0.009027884  
dog Probability:0.159547865  
frog Probability:0.003485858  
horse Probability:0.004825624  
ship Probability:0.125612140  
truck Probability:0.007025036
```

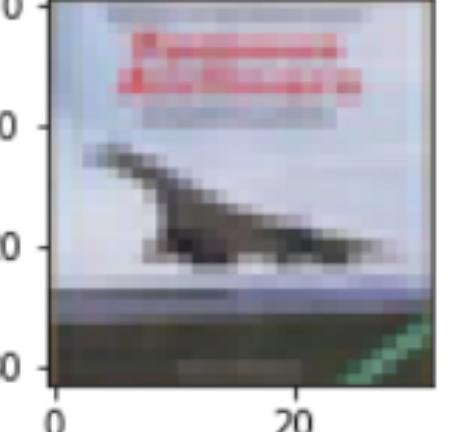
預測為「貓」的機率最高

預測為「狗」的機率次高

10.7 查看預測機率

- 查看第3筆資料預測的機率

4. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
show_Predicted_Probability(y_label_test,prediction,  
                           x_img_test,Predicted_Probability,3)  
  
label: airplane predict: airplane  
  
  
  
airplane Probability:0.560479105  
automobile Probability:0.016172931  
bird Probability:0.029778995  
cat Probability:0.003971634  
deer Probability:0.008766360  
dog Probability:0.000196363  
frog Probability:0.003189082  
horse Probability:0.000726878  
ship Probability:0.368772686  
truck Probability:0.007945805
```

預測為「飛機」的機率次高

預測為「船」的機率最高

10.8

顯示混淆矩陣 (Confusion Matrix)

10.8 顯示混淆矩陣

- 查看prediction預測結果的shape形狀

1. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
prediction.shape  
(10000,)
```

prediction是1維
陣列

我們將使用pd.crosstab建立混淆矩陣
(confusion matrix)但是pd.crosstab的輸入都必
須是1維陣列，所以必須先確認prediction(預測
結果)與y_label_test(真實值)是1維陣列。如果不
是1維陣列，必須先轉換為1維陣列。

10.8 顯示混淆矩陣

- 查看y_label_test真實值的shape形狀

2. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
y_label_test.shape  
(10000, 1)
```

y_label_test真實值
的shape形狀是2維
陣列，後續必須轉
換為1維陣列

10.8 顯示混淆矩陣

- 將y_label_test真實值轉換為1維陣列

3. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
y_label_test.reshape(-1)
```

```
array([3, 8, 8, ..., 5, 1, 7], dtype=uint8)
```

使用reshape(-1)轉
換為1維陣列

由結果可以看到已經
轉換為1維陣列。

10.8 顯示混淆矩陣

- 執行預測

4. 在命令列輸入下列程式碼，並且按下Shift+Enter執行。

```
import pandas as pd
print(label_dict)
pd.crosstab(y_label_test.reshape(-1), prediction,
             rownames=['label'], colnames=['predict'])

{0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'}
```

predict	0	1	2	3	4	5	6	7	8	9
label										
0	760	6	67	23	29	11	13	9	63	19
1	20	795	12	17	5	5	20	2	48	76
2	47	1	637	66	110	43	60	21	12	3
3	8	5	64	585	91	133	80	21	10	3
4	12	3	63	54	751	16	57	35	8	1
5	9	0	51	219	64	579	40	30	5	3
6	2	1	40	48	38	13	852	1	4	1
7	8	1	39	53	76	51	8	760	3	1
8	51	16	18	15	17	13	9	3	850	8
9	39	60	15	37	11	8	10	22	37	761

10.8 顯示混淆矩陣

- 程式碼說明

```
import pandas as pd  
print(label_dict)  
  
pd.crosstab(y_label_test.reshape(-1), prediction,  
            rownames=['label'], colnames=['predict'])
```

匯入pandas模組，
後續會以pd引用。

顯示label_dict字典，方
便後續對照查看。

使用pd.crosstab建立混淆矩陣，輸入下列參
數：

- 測試資料的真實值，使用.reshape(-1)轉換為1
維陣列
- 測試資料的預測結果
- 設定行的名稱是label
- 設定列的名稱是predict

10.8 顯示混淆矩陣

- 執行結果說明

```
{0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'}
```

predict	0	1	2	3	4	5	6	7	8	9
label	760	6	67	23	29	11	13	9	63	19
0	20	795	12	17	5	5	20	2	48	76
1	47	1	637	66	110	43	60	21	12	3
2	8	5	64	585	91	133	80	21	10	3
3	12	3	63	54	751	16	57	35	8	1
4	9	0	51	219	64	579	40	30	5	3
5	2	1	40	48	38	13	852	1	4	1
6	8	1	39	53	76	51	8	760	3	1
7	51	16	18	15	17	13	9	3	850	8
8	39	60	15	37	11	8	10	22	37	761
9										

真實值是3，但預測是5

正確的預測

實際上是狗，但被辨識成貓

10.8 顯示混淆矩陣

- 執行結果說明

```
{0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog', 7: 'horse', 8: 'ship', 9: 'truck'}
```

predict	0	1	2	3	4	5	6	7	8	9
label										
0	760	6	67	23	29	11	13	9	63	19
1	20	795	12	17	5	5	20	2	48	76
2	47	1	637	66	110	43	60	21	12	3
3	8	5	64	585	91	133	80	21	10	3
4	12	3	63	54	751	16	57	35	8	1
5	9	0	51	219	64	579	40	30	5	3
6	2	1	40	48	38	13	852	1	4	1
7	8	1	39	53	76	51	8	760	3	1
8	51	16	18	15	17	13	9	3	850	8
9	39	60	15	37	11	8	10	22	37	761

2(鳥),3(貓),4(鹿),5(狗),6(蛙)
,7(馬)都是動物不易混淆為
1(汽車)

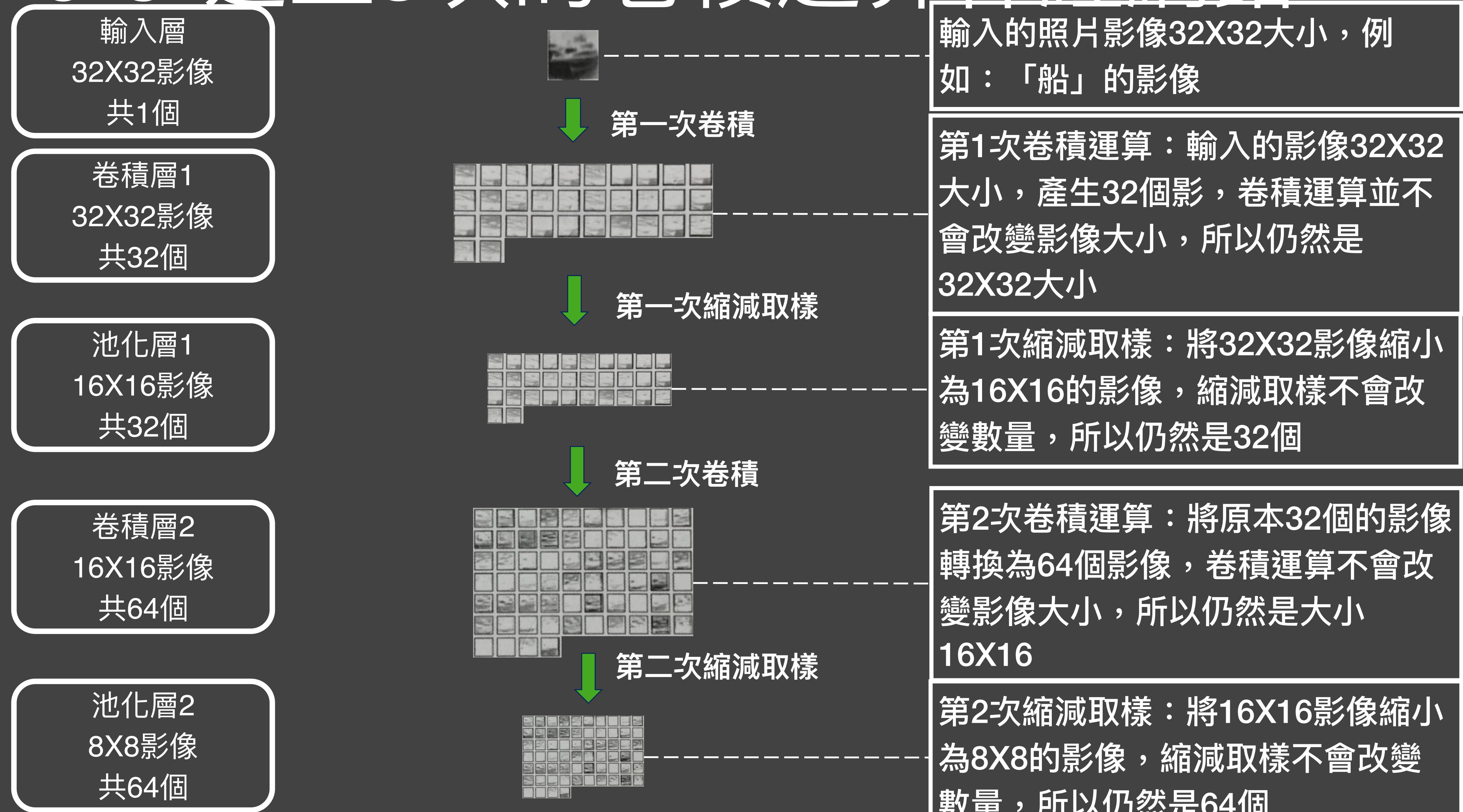
8(船),9(卡車)屬於交通工
具，容易混淆為1(汽車)

10.9

建立3次的卷積運算神經網路

為提高準確率，本小節嘗試更多次的卷積運算

10.9 建立3次的卷積運算神經網路



10.9 建立3次的卷積運算神經網路

第2次縮減取樣的結果

第三次卷積

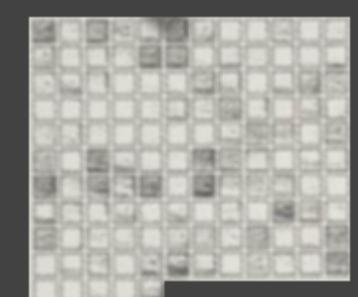


第3次卷積運算：將原本64個的影像轉換為128個影像，卷積運算不會改變影像大小，所以仍然是大小 8×8

卷積層3
8X8影像
共128個

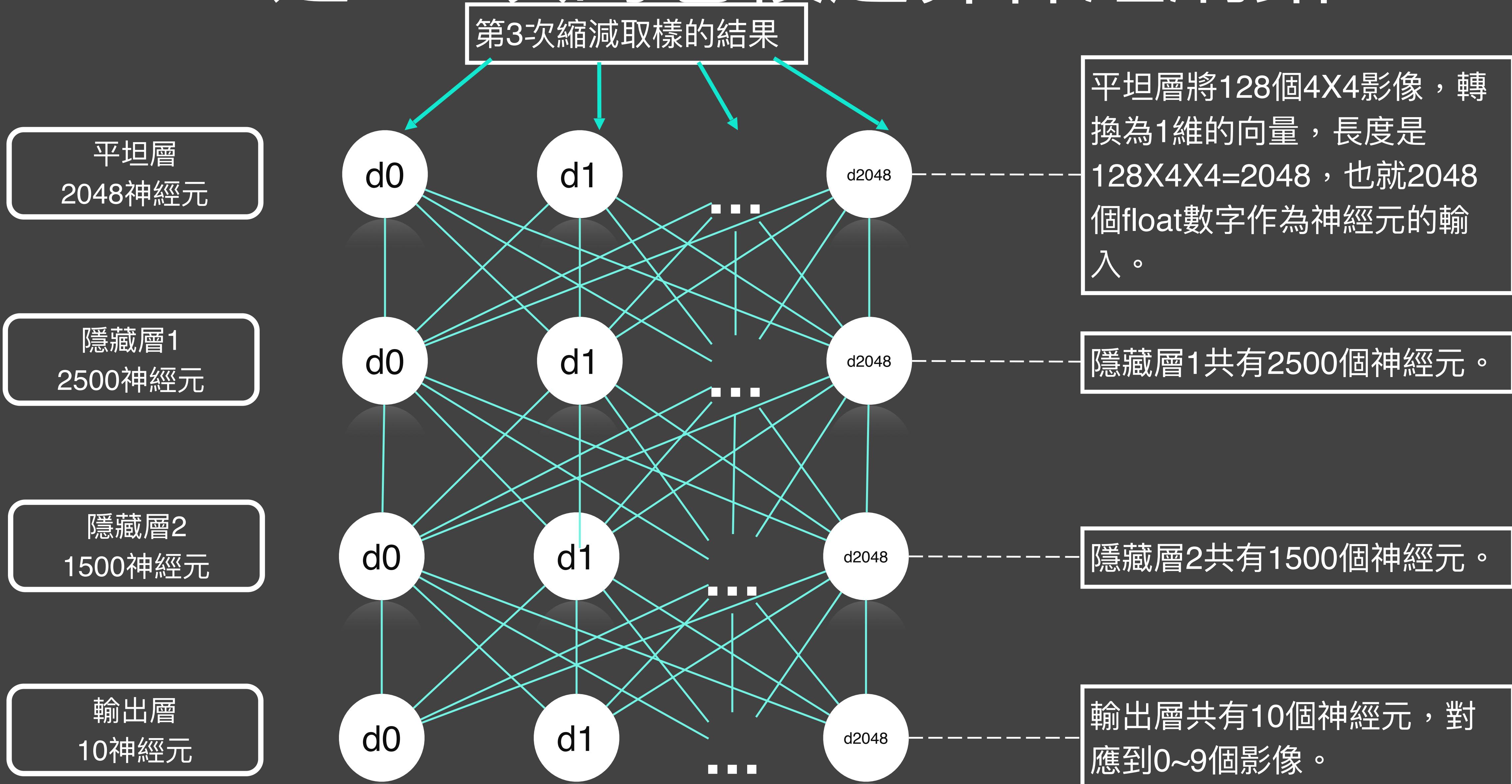
池化層3
4X4影像
共128個

第三次縮減取樣



第3次縮減取樣：將 8×8 影像縮小為 4×4 的影像，縮減取樣不會改變數量，所以仍然是128個

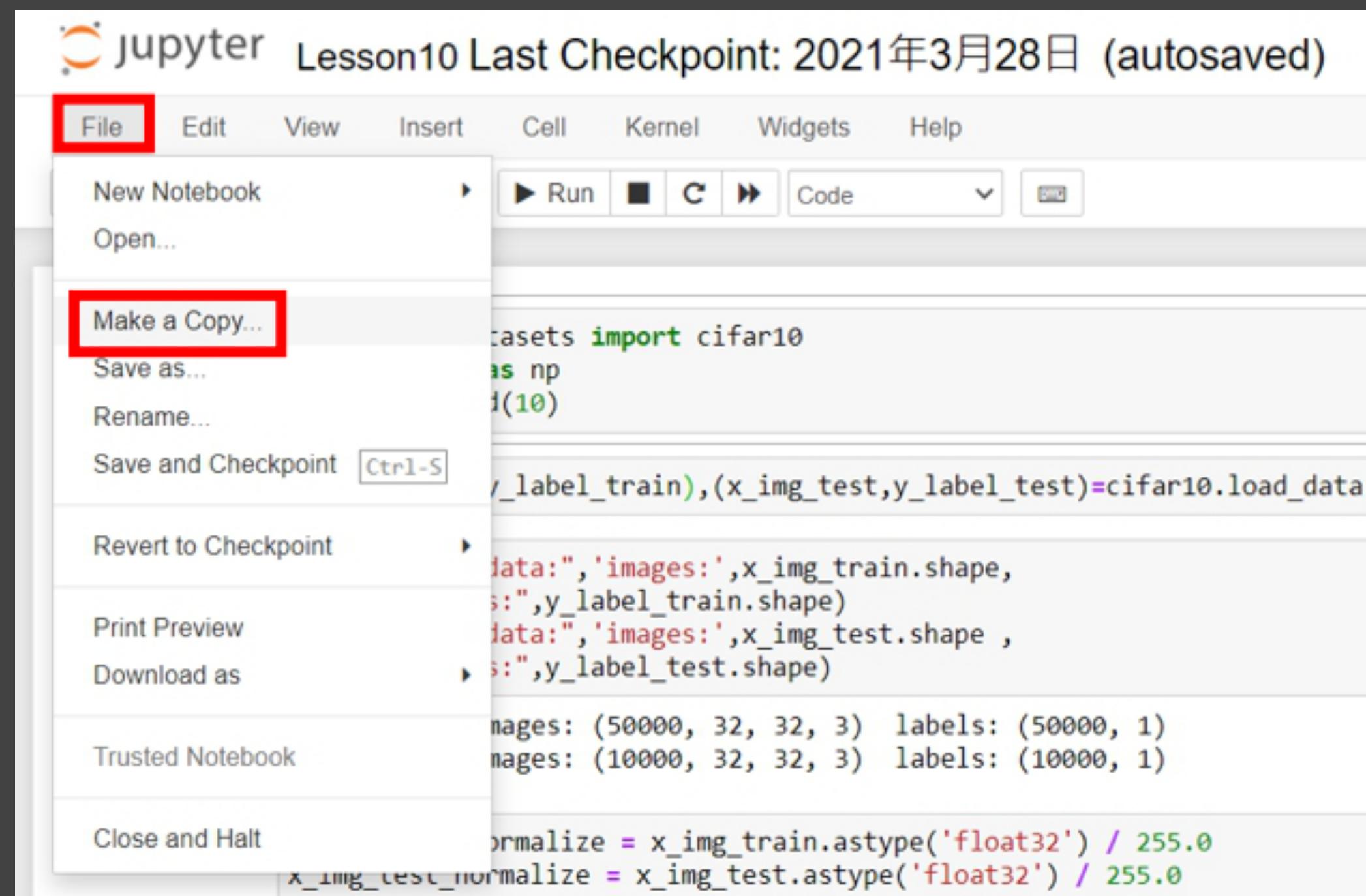
10.9 建立3次的卷積運算神經網路



10.9 建立3次的卷積運算神經網路

- 製作副本。

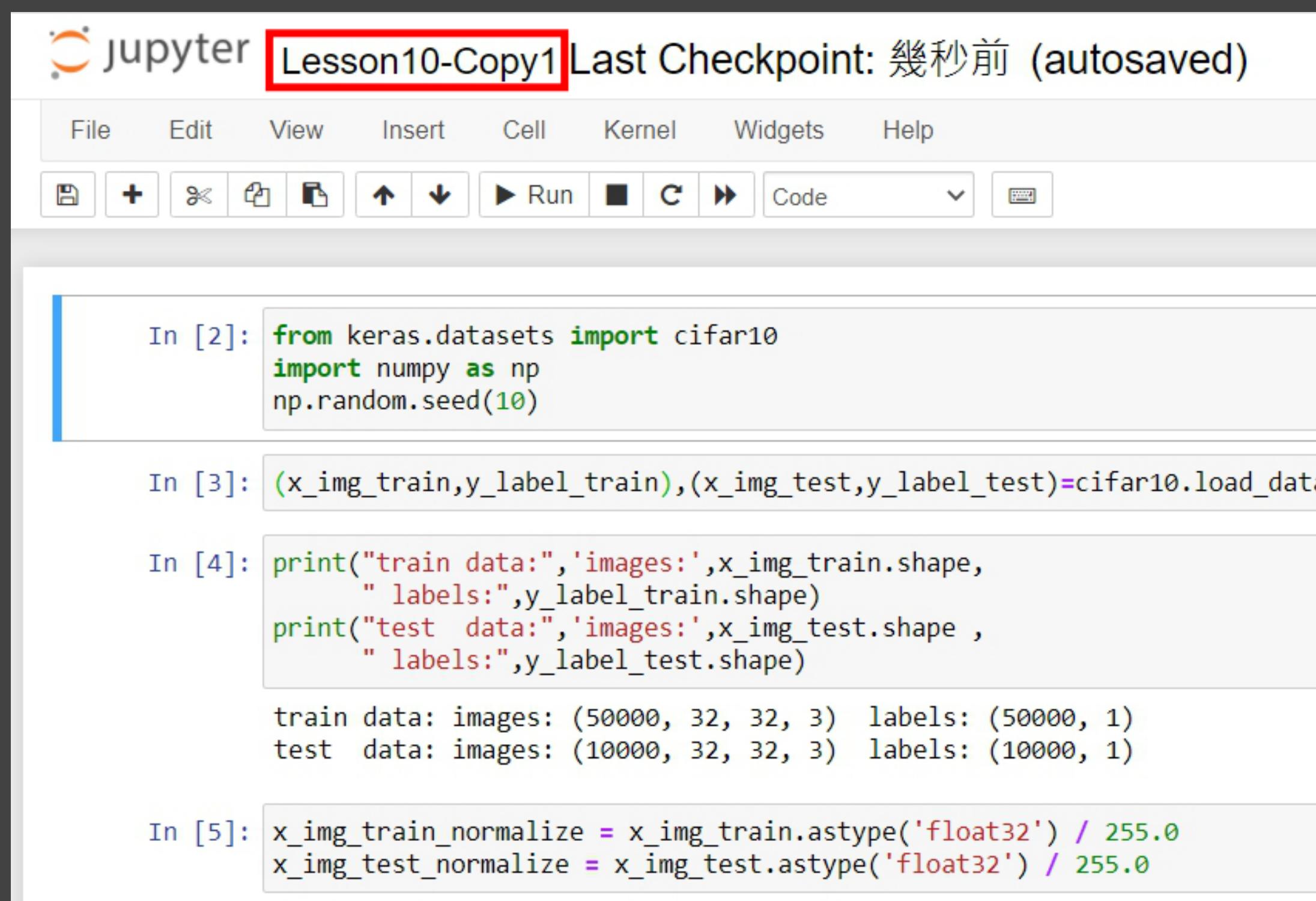
1. 左鍵單擊工具列中的File選項，並在出現的選單中左鍵單擊Make a Copy。



10.9 建立3次的卷積運算神經網路

- 為副本重新命名

2. 左鍵單擊副本名稱(圖中紅框處)重新命名。



The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** jupyter Lesson10-Copy1 Last Checkpoint: 幾秒前 (autosaved)
- Toolbar:** File, Edit, View, Insert, Cell, Kernel, Widgets, Help.
- Cell Buttons:** Save, New, Cell, Run, Stop, Cell Type: Code.
- In [2]:**

```
from keras.datasets import cifar10
import numpy as np
np.random.seed(10)
```
- In [3]:**

```
(x_img_train,y_label_train),(x_img_test,y_label_test)=cifar10.load_data()
```
- In [4]:**

```
print("train data:",'images:',x_img_train.shape,
      "labels:",y_label_train.shape)
print("test data:",'images:',x_img_test.shape ,
      "labels:",y_label_test.shape)
```

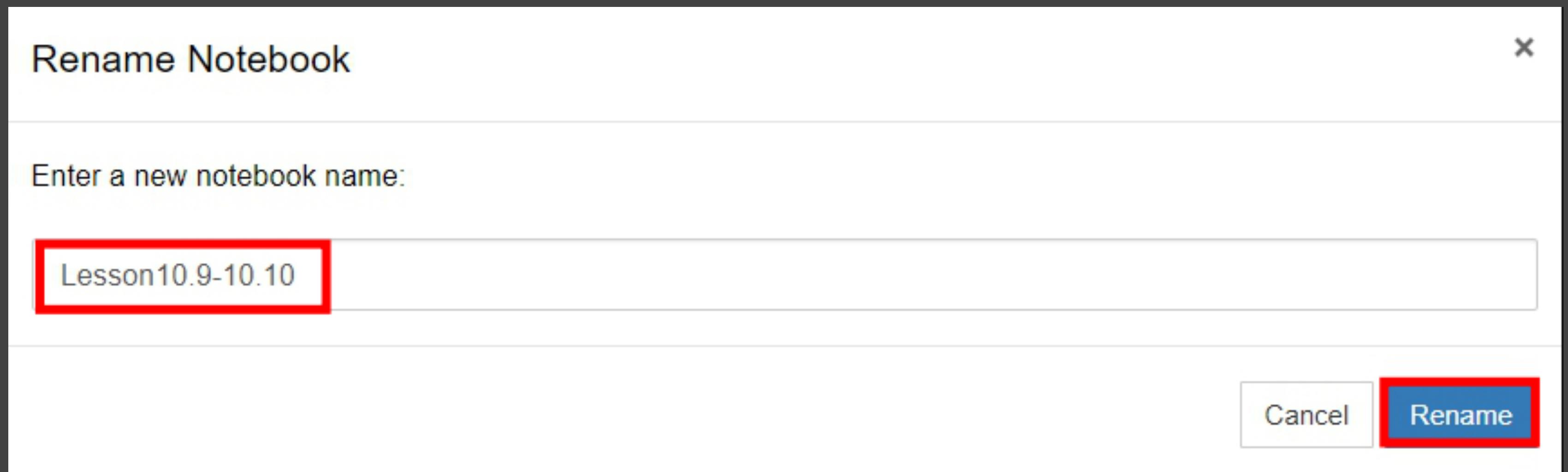
Output:
train data: images: (50000, 32, 32, 3) labels: (50000, 1)
test data: images: (10000, 32, 32, 3) labels: (10000, 1)
- In [5]:**

```
x_img_train_normalize = x_img_train.astype('float32') / 255.0
x_img_test_normalize = x_img_test.astype('float32') / 255.0
```

10.9 建立3次的卷積運算神經網路

- 執行預測

3. 在彈出的對話框內的輸入框中輸入Lesson10.9-10.10後，左鍵單擊Rename按鈕完成重新命名。



10.9 建立3次的卷積運算神經網路

- 更改Dropout的值

4. 找到卷積層1和卷積層2的Dropout，將原本的0.25更改为0.3。

```
In [8]: model = Sequential()

In [9]: model.add(Conv2D(filters=32,kernel_size=(3,3),
                      input_shape=(32, 32,3),
                      activation='relu',
                      padding='same'))

In [10]: model.add(Dropout(rate=0.3))

In [11]: model.add(MaxPooling2D(pool_size=(2, 2)))

In [12]: model.add(Conv2D(filters=64, kernel_size=(3, 3),
                      activation='relu', padding='same'))

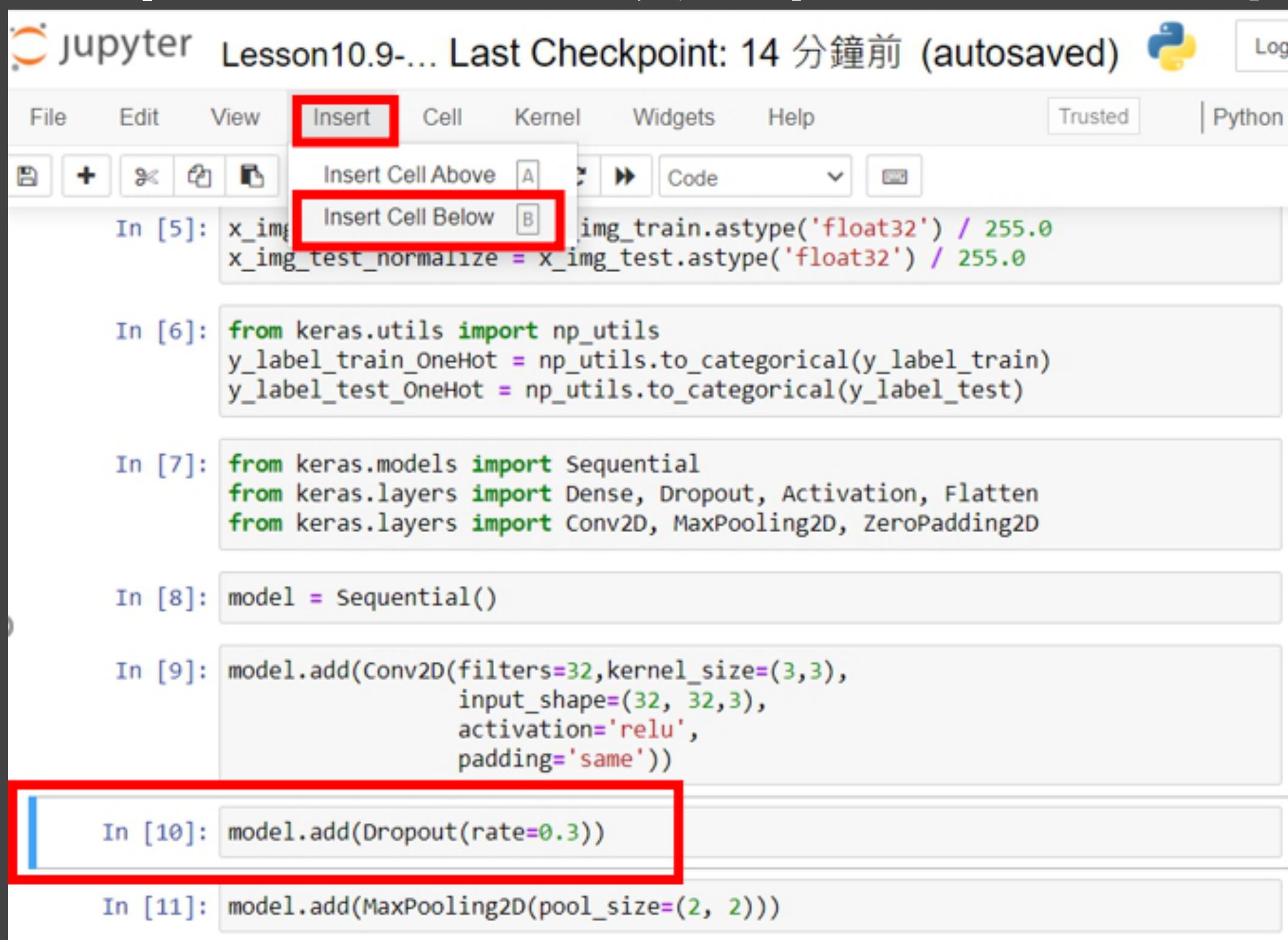
In [13]: model.add(Dropout(0.3))

In [14]: model.add(MaxPooling2D(pool_size=(2, 2)))
```

10.9 建立3次的卷積運算神經網路

- 插入Cell

5. 以左鍵單擊的方式選取卷積層1的Dropout的Cell後，左鍵單擊工具列中的Insert選項，在出現的選單中左鍵單擊Insert Cell Below選項。



```
jupyter Lesson10.9... Last Checkpoint: 14 分鐘前 (autosaved) Logout  
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3  
In [5]: x_img_train.astype('float32') / 255.0  
x_img_test_normalize = x_img_test.astype('float32') / 255.0  
In [6]: from keras.utils import np_utils  
y_label_train_OneHot = np_utils.to_categorical(y_label_train)  
y_label_test_OneHot = np_utils.to_categorical(y_label_test)  
In [7]: from keras.models import Sequential  
from keras.layers import Dense, Dropout, Activation, Flatten  
from keras.layers import Conv2D, MaxPooling2D, ZeroPadding2D  
In [8]: model = Sequential()  
In [9]: model.add(Conv2D(filters=32,kernel_size=(3,3),  
input_shape=(32, 32, 3),  
activation='relu',  
padding='same'))  
In [10]: model.add(Dropout(rate=0.3))  
In [11]: model.add(MaxPooling2D(pool_size=(2, 2)))
```

10.9 建立3次的卷積運算神經網路

- 增加Conv2D層

6. 在新增的Cell中輸入下列程式碼。

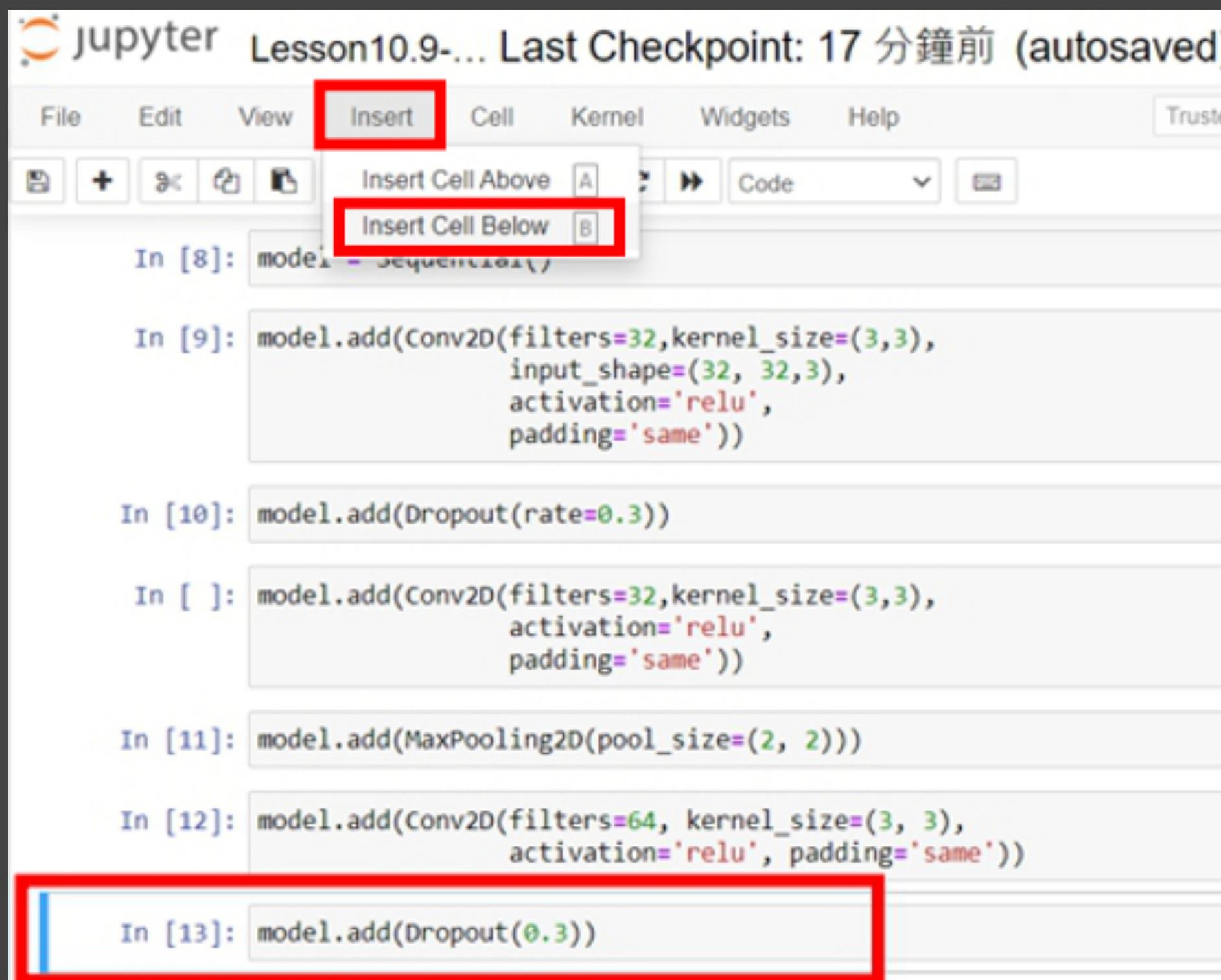
```
In [10]: model.add(Dropout(rate=0.3))
```

```
In [ ]: model.add(Conv2D(filters=32,kernel_size=(3,3),  
activation='relu',  
padding='same'))
```

10.9 建立3次的卷積運算神經網路

- 插入Cell

7.以左鍵單擊的方式選取卷積層2的Dropout的Cell後，左鍵單擊工具列中的Insert選項，在出現的選單中左鍵單擊Insert Cell Below選項。



The screenshot shows a Jupyter Notebook window titled "jupyter Lesson10.9-... Last Checkpoint: 17 分鐘前 (autosaved)". The menu bar is visible with "Insert" highlighted by a red box. A dropdown menu is open under "Insert" with "Insert Cell Below" also highlighted by a red box. The code cells are numbered In [8] through In [13]. In [8] contains "model.add(Dropout(rate=0.3))". In [9] contains "model.add(Conv2D(filters=32,kernel_size=(3,3), input_shape=(32, 32,3), activation='relu', padding='same'))". In [10] contains "model.add(Dropout(rate=0.3))". In [11] contains "model.add(MaxPooling2D(pool_size=(2, 2)))". In [12] contains "model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'))". In [13] contains "model.add(Dropout(0.3))". The cell In [13] is currently active, indicated by a red border around its input field.

10.9 建立3次的卷積運算神經網路

- 增加Conv2D層

8. 在新增的Cell中輸入下列程式碼。

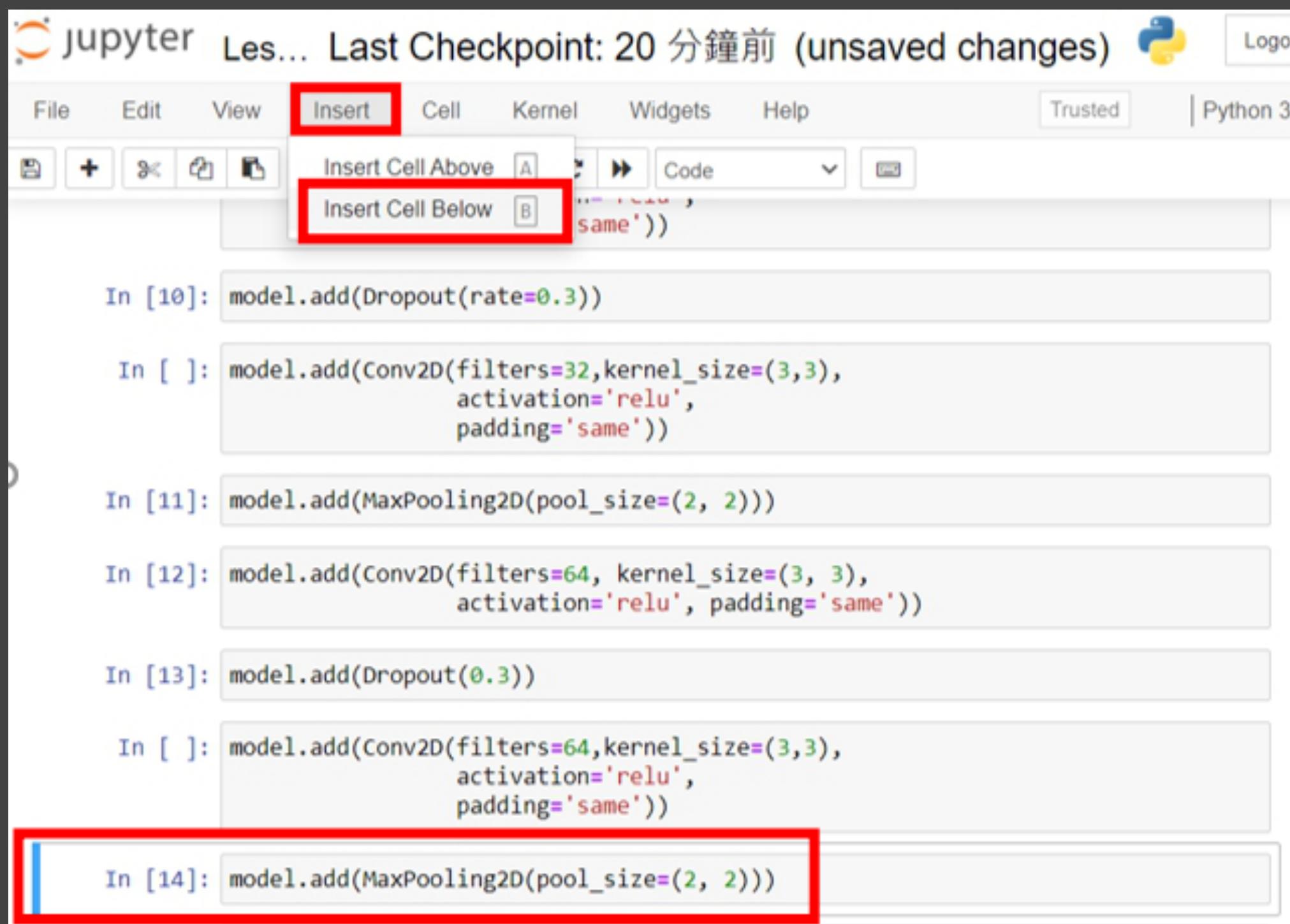
```
In [13]: model.add(Dropout(0.3))
```

```
In [ ]: model.add(Conv2D(filters=64,kernel_size=(3,3),  
activation='relu',  
padding='same'))
```

10.9 建立3次的卷積運算神經網路

- 插入Cell

9. 以左鍵單擊的方式選取池化層2的Cell後，左鍵單擊工具列中的Insert選項，在出現的選單中左鍵單擊Insert Cell Below選項。



10.9 建立3次的卷積運算神經網路

- 新增卷積層3與池化層3

10. 在新增的Cell中輸入下列程式碼。

```
In [14]: model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
In [ ]: model.add(Conv2D(filters=64, kernel_size=(3, 3),  
activation='relu', padding='same'))  
model.add(Dropout(0.3))  
model.add(Conv2D(filters=64, kernel_size=(3, 3),  
activation='relu',  
padding='same'))
```

10.9 建立3次的卷積運算神經網路

- 變更平坦層、隱藏層1的參數，並且新增隱藏層2

11.

- a) 將原本的Dropout的值從0.25變更為0.3。
- b) 將隱藏層1的神經元個數從1024變更為2500。
- c) 新增隱藏層2以及Dropout，神經元個數為1500。

```
In [15]: model.add(Flatten())
model.add(Dropout(rate=0.25))
```

```
In [16]: model.add(Dense(1024, activation='relu'))
model.add(Dropout(rate=0.25))
```

```
In [17]: model.add(Dense(10, activation='softmax'))
```

```
In [15]: model.add(Flatten())
model.add(Dropout(0.3))
```

```
In [16]: model.add(Dense(2500, activation='relu'))
model.add(Dropout(rate=0.3))
model.add(Dense(1500, activation='relu'))
model.add(Dropout(rate=0.3))
```

```
In [17]: model.add(Dense(10, activation='softmax'))
```

Before

After

• 10.9 建立3次的卷積運算神經網路

12. 將原本epoch值變更為50，但由於本次執行會花上不少時間，所以在此先不要執行。

我們將介紹模型的儲存與載入來切割工作量，使每次程式執行完一批訓練後，可以將模型權重儲存，以便下次程式執行訓練前，先載入模型權重再繼續訓練。避免突發狀況導致之前的訓練前功盡棄。

10.9 建立3次的卷積運算神經網路

- 變更epoch(訓練週期)

12. 將原本epoch值變更為50，但由於本次執行會花上不少時間，所以在此**先不要執行**。

```
train_history=model.fit(x_img_train_normalize, y_label_train_OneHot,  
                        validation_split=0.2,  
                        epochs=50, batch_size=128, verbose=1)
```

我們將介紹模型的儲存與載入來切割工作量，使每次程式執行完一批訓練後，可以將模型權重儲存。

- 下次程式執行訓練前，可以載入先前模型權重繼續訓練。
- 避免突發狀況導致之前的訓練前功盡棄。

10.10

模型的儲存與載入

10.10 模型的儲存與載入

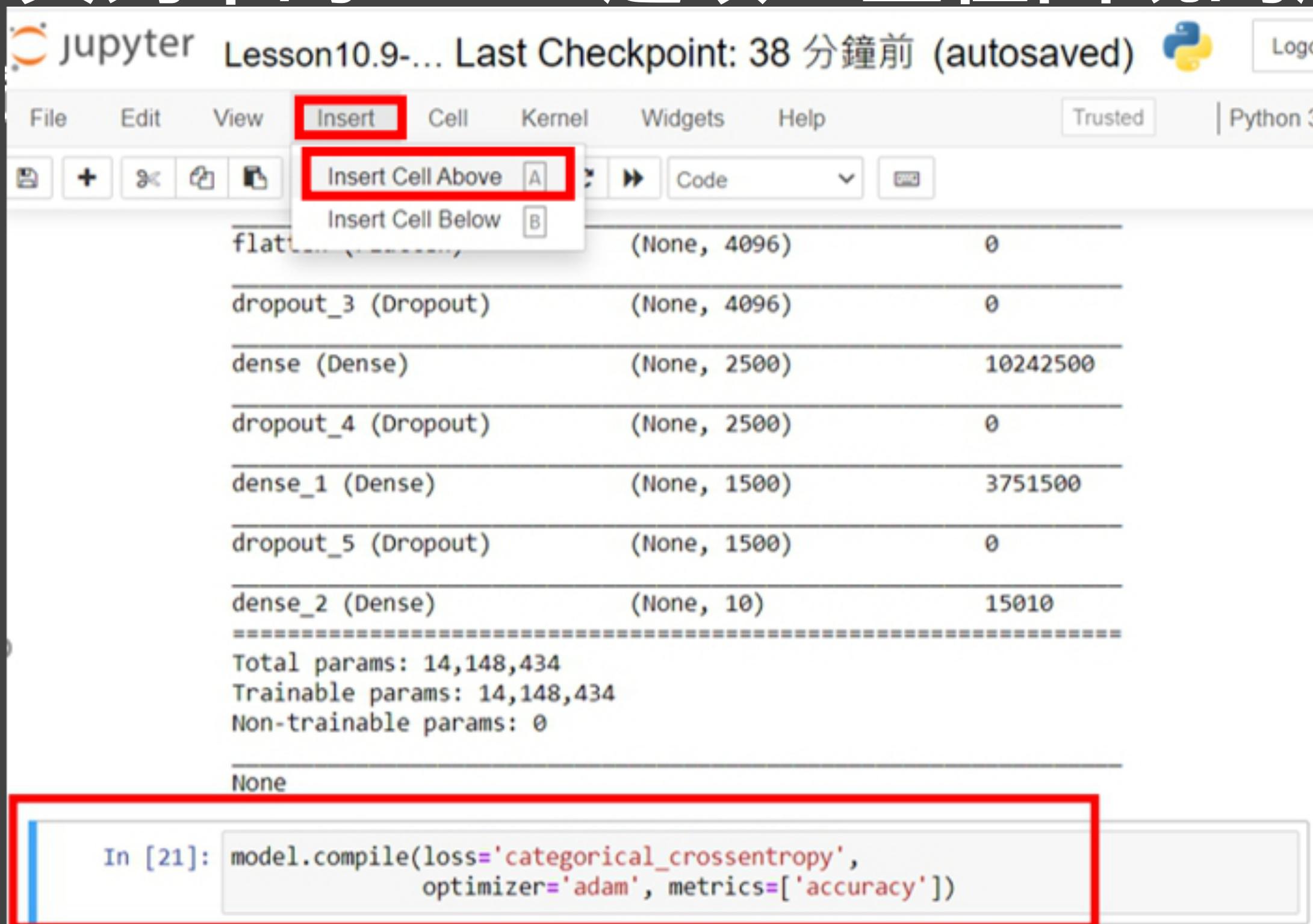
- 變更epoch(訓練週期)

1. 將原本epoch值設定為較小的值，在此我們設定為1。

10.10 模型的儲存與載入

- 新增Cell

2. 以左鍵單擊的方式選取原有的model.compile的Cell，接著左鍵單擊工具列中的Insert選項，並在出現的選單中左鍵單擊Insert Cell Above。



The screenshot shows a Jupyter Notebook interface. The top navigation bar has 'File', 'Edit', 'View', 'Insert' (which is highlighted with a red box), 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu bar is a toolbar with icons for file operations like new, open, save, and run cell. The main area displays a neural network model summary and its configuration code.

```
In [21]: model.compile(loss='categorical_crossentropy',  
                      optimizer='adam', metrics=['accuracy'])
```

The model summary output is:

```
flat1 (Flatten) (None, 4096) 0  
dropout_3 (Dropout) (None, 4096) 0  
dense (Dense) (None, 2500) 10242500  
dropout_4 (Dropout) (None, 2500) 0  
dense_1 (Dense) (None, 1500) 3751500  
dropout_5 (Dropout) (None, 1500) 0  
dense_2 (Dense) (None, 10) 15010  
=====  
Total params: 14,148,434  
Trainable params: 14,148,434  
Non-trainable params: 0
```

10.10 模型的儲存與載入

- 使用model.load_weights載入模型權重

3. 在新增的Cell中輸入下列程式碼。

```
In [ ]: try:  
    model.load_weights("SaveModel/cifarCnnModel.h5")  
    print("載入模型成功! 繼續訓練模型")  
except :  
    print("載入模型失敗! 開始訓練一個新模型")
```

```
In [21]: model.compile(loss='categorical_crossentropy',  
                      optimizer='adam', metrics=['accuracy'])
```

10.10 模型的儲存與載入

- 使用model.save_weights載入模型權重

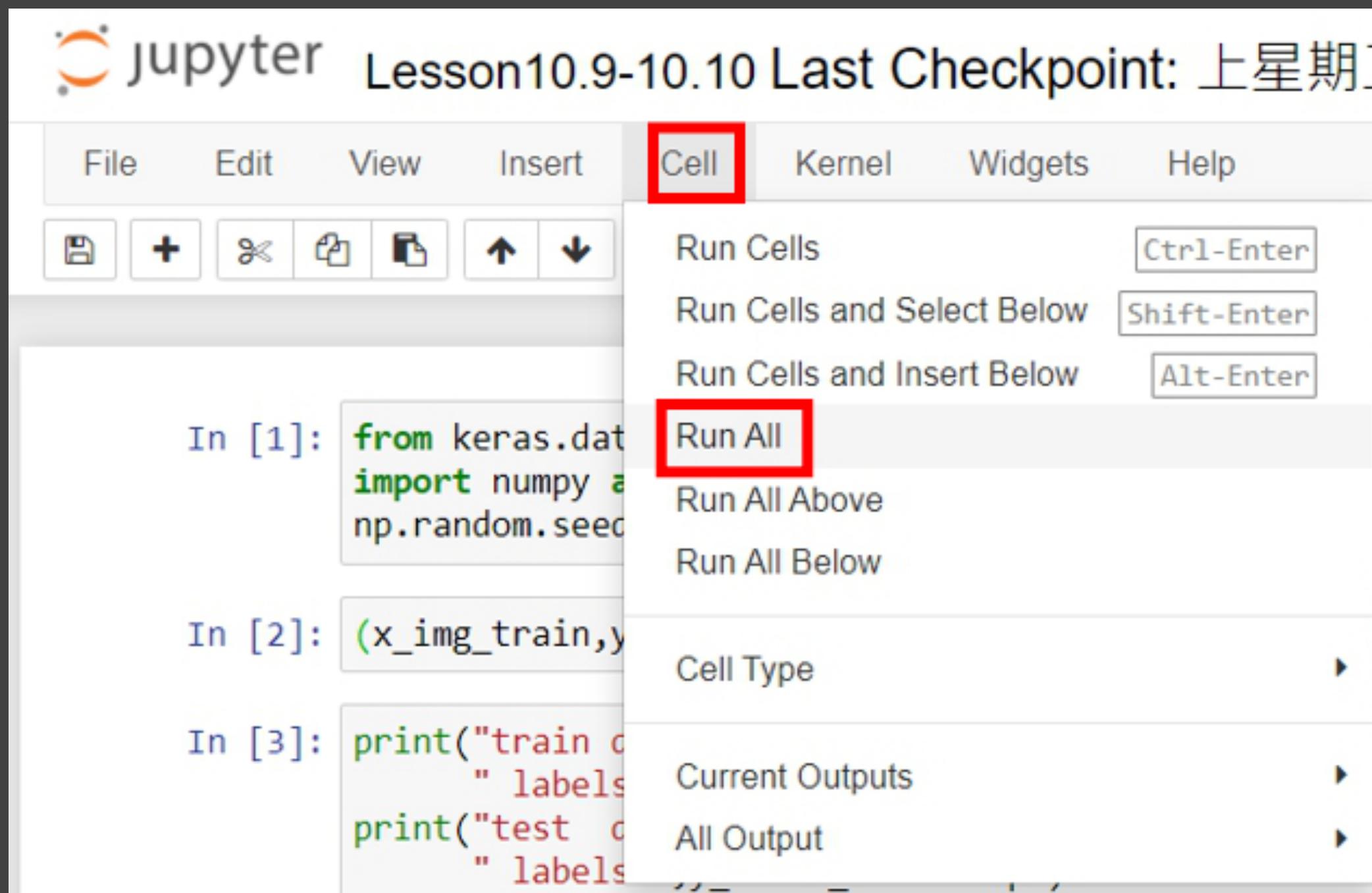
4.在本檔案中的最後一個命令列輸入下列程式碼。

```
model.save_weights("SaveModel/cifarCnnModel.h5")
print("Saved model to disk")
```

10.10 模型的儲存與載入

- 執行程式

5. 左鍵單擊工具列中的Cell選項，並在出現的選單中左鍵單擊Run All選項。



10.10 模型的儲存與載入

- `model.load_weights`執行結果
- 因為在此是第一次執行，所以沒有先前的`model`可以載入，所以印出「載入模型失敗!開始訓練一個新模型」字樣。

```
try:  
    model.load_weights("SaveModel/cifarCnnModel.h5")  
    print("載入模型成功!繼續訓練模型")  
except :  
    print("載入模型失敗!開始訓練一個新模型")
```

載入模型失敗!開始訓練一個新模型

10.10 模型的儲存與載入

- `model.save_weights`執行結果
- 模型權重儲存成功，出現「Saved model to disk」字樣。

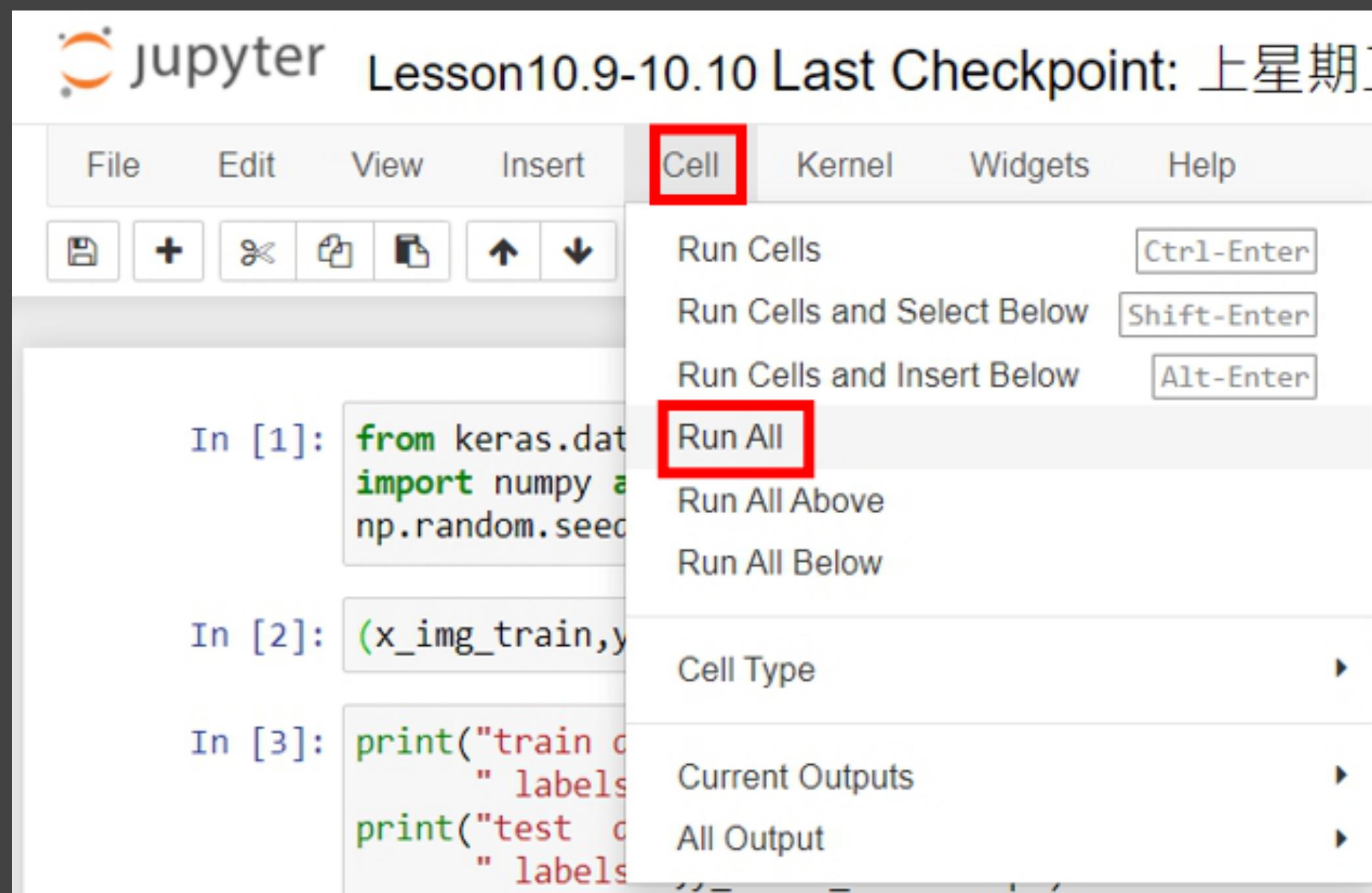
```
model.save_weights("SaveModel/cifarCnnModel.h5")
print("Saved model to disk")
```

```
Saved model to disk
```

10.10 模型的儲存與載入

- 第2次執行程式

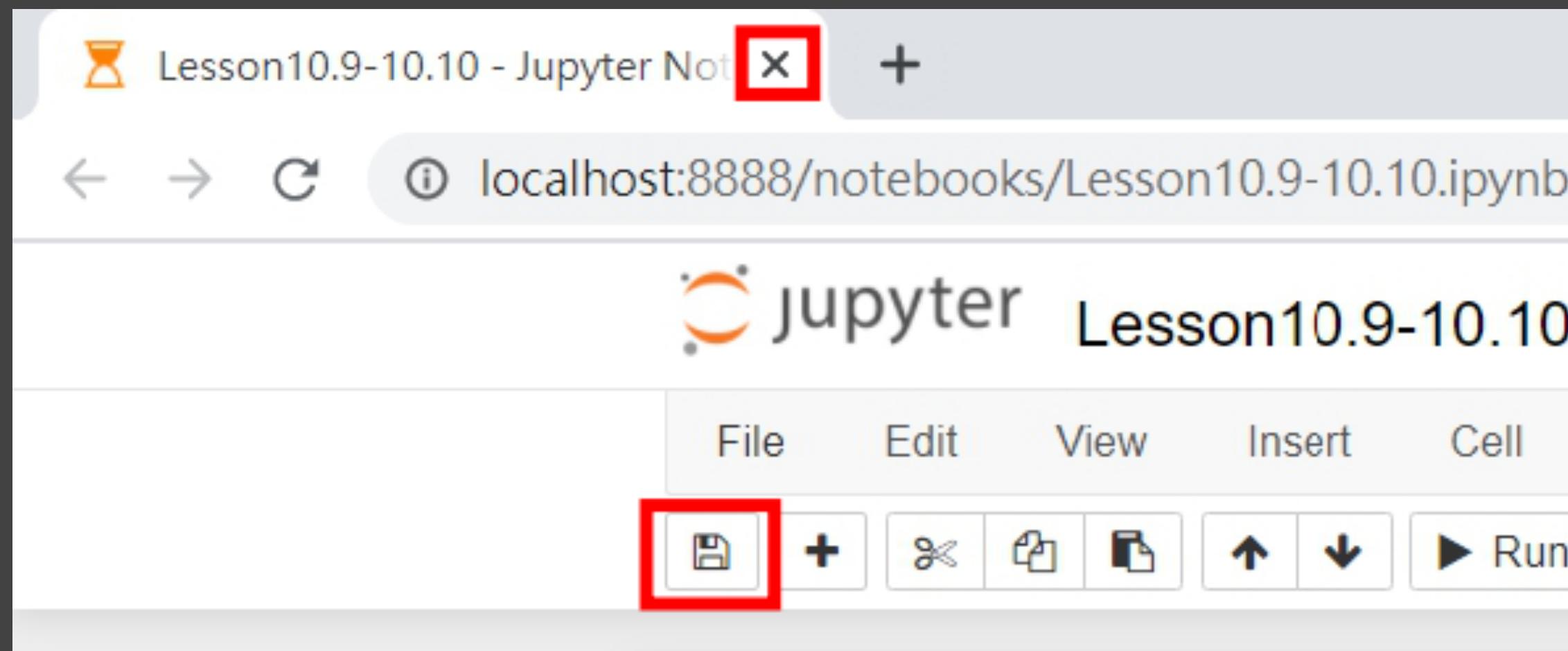
6. 左鍵單擊工具列中的Cell選項，並在出現的選單中左鍵單擊Run All選項。



10.10 模型的儲存與載入

- 儲存檔案

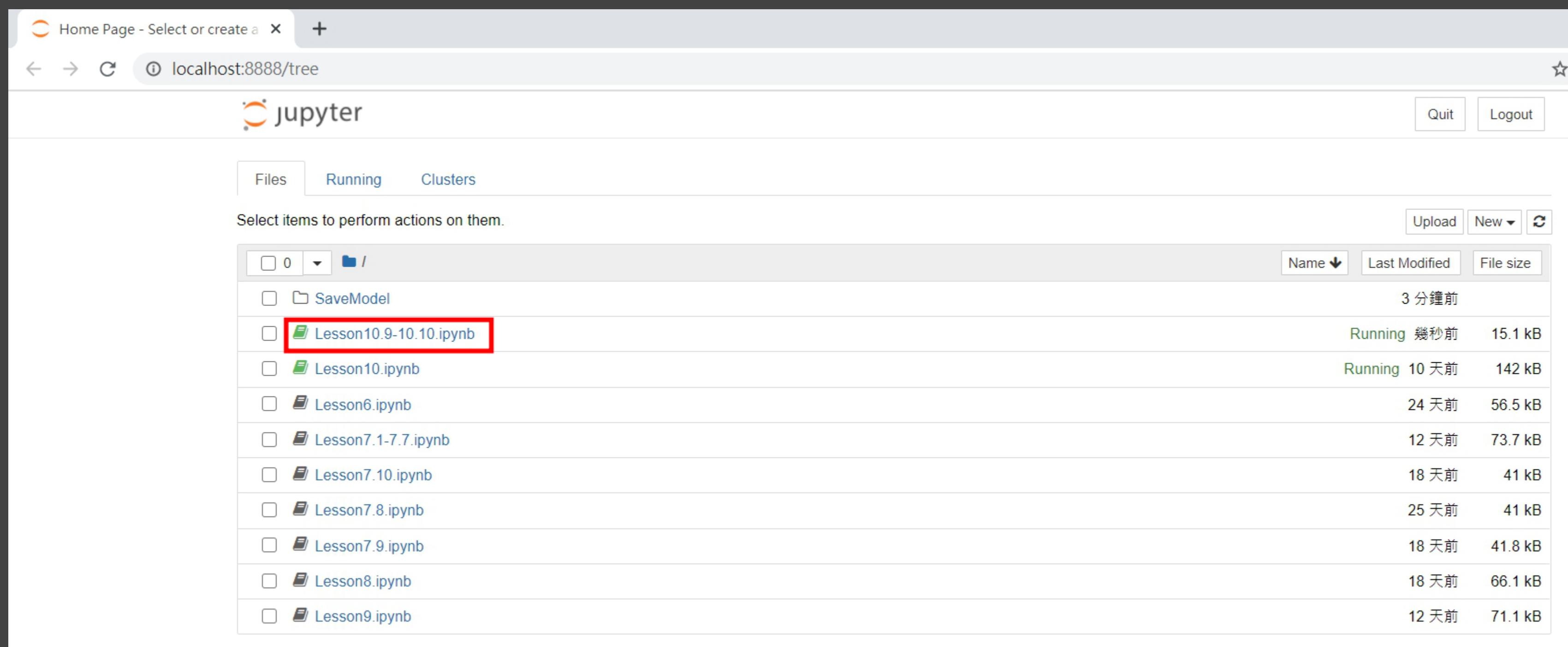
7. 左鍵單擊儲存圖示，接著關閉此檔案。



10.10 模型的儲存與載入

- 開啟檔案

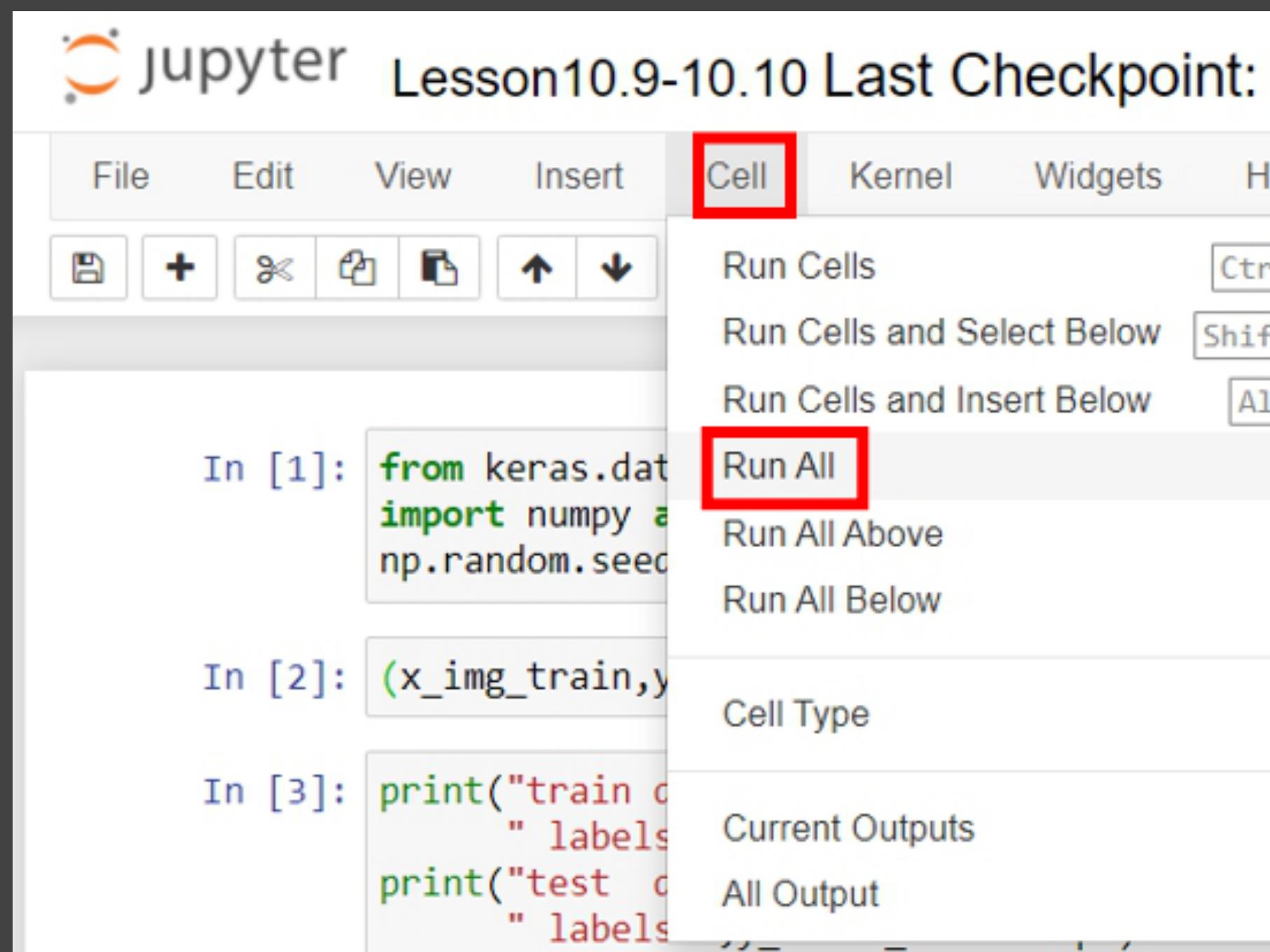
8. 左鍵單擊檔案列表中的Lesson10.9-10.10.ipynb檔案。



10.10 模型的儲存與載入

- 第2次執行程式

9. 左鍵單擊工具列中的Cell選項，並在出現的選單中左鍵單擊Run All選項。



10.10 模型的儲存與載入

- `model.load_weights`執行結果
- 因為是第二次執行，已有先前的`model`可以載入，所以印出「載入模型成功!繼續訓練模型」字樣。

```
try:  
    model.load_weights("SaveModel/cifarCnnModel.h5")  
    print("載入模型成功!繼續訓練模型")  
except :  
    print("載入模型失敗!開始訓練一個新模型")
```

載入模型成功!繼續訓練模型

10.11

結論

10.11 結論

- 本章，使用Keras建立卷積神經網路CNN辨識Cifar10影像資料。
- 下一章，將介紹以多層感知器模型預測鐵達尼號乘客的生存機率。