資料進行預處理

1.匯入所需模組

```
from  keras.datasets  import  cifar10       #從keras.datasets匯入cifar10資料集
import  numpy  as  np                       #匯入numpy模組，NumPy是Python語言的擴充程式庫。支援維度
np.random.seed(10)                           #設定seed可以讓每次需要隨機產生的資料，都有相同的輸出
```

2讀取cifar10資料

```
(X_img_train,Y_label_train),(X_img_test,Y_label_test)=cifar10.load_data()
```

```
      Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
      170500096/170498071 [==============================] - 2s 0us/step
      170508288/170498071 [==============================] - 2s 0us/step
```

3.顯示訓練與驗證資料的shape

```
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)
```

4.將features(照片影像特徵值)標準化

```
X_img_train_normalize  =  X_img_train.astype(’float32’)/255.0
X_img_test_normalize  =  X_img_test.astype(’float32’)/255.0
```

5.label(照片影像的真實的值)以Onehot encoding 轉換

```
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)
```

建立模型

1.匯入所需模組

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

2.建立keras的Sequential模型

```
model  =  Sequential()
```

3.建立卷積層1

```
model.add(Conv2D(filters=32,                 #設定隨機產生32個濾鏡filter  weight
                 kernel_size=(3,3),          #每一個濾鏡3*3大小
                 input_shape=(32,32,3),      #第1，2維度:代表輸入的影像形狀32*32大小，第3個維度:因為
                 activation='relu',          #設定ReLU激活函數
                 padding='same'))            #此設定讓卷積運算，產生的卷積影像大小不變
```

4.加入Dropout避免overfitting

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

5.建立池化層1建立池化層1

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

6.建立卷積層2

```
model.add(Conv2D(filters=64,                         #建立64鏡filter  weight
                 kernel_size=(3,3),                  #每一個濾鏡3*3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影像大小
```

7.加入Dropout避免overfitting

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

8.建立池化層2

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

9.建立平坦層

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

10.建立隱藏層

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

11.建立輸出層

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

12.查看模型的摘要

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

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896       
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 32)        0         
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 64)        18496     
_________________________________________________________________
dropout_1 (Dropout)          (None, 16, 16, 64)        0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 4096)              0         
_________________________________________________________________
dense (Dense)                (None, 1024)              4195328   
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                10250     
=================================================================
Total params: 4,224,970
Trainable params: 4,224,970
Non-trainable params: 0
_________________________________________________________________
None
```

進行訓練

1.定義訓練方式

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

2.開始訓練

```
train_history=model.fit(X_img_train_normalize,Y_label_train_OneHot,validation_split=0.2,epochs=10,b
Epoch 1/10
313/313 [==============================] - 47s 12ms/step - loss: 1.8507 - accuracy: 0.3353 -
Epoch 2/10
313/313 [==============================] - 3s 10ms/step - loss: 1.2275 - accuracy: 0.5639 - v
Epoch 3/10
313/313 [==============================] - 3s 10ms/step - loss: 1.0687 - accuracy: 0.6219 - v
Epoch 4/10
313/313 [==============================] - 3s 10ms/step - loss: 0.9390 - accuracy: 0.6665 - v
Epoch 5/10
313/313 [==============================] - 3s 10ms/step - loss: 0.8525 - accuracy: 0.6979 - v
Epoch 6/10
313/313 [==============================] - 3s 10ms/step - loss: 0.7594 - accuracy: 0.7308 - v
Epoch 7/10
313/313 [==============================] - 3s 10ms/step - loss: 0.6937 - accuracy: 0.7554 - v
Epoch 8/10
313/313 [==============================] - 3s 10ms/step - loss: 0.6182 - accuracy: 0.7853 - v
Epoch 9/10
313/313 [==============================] - 3s 10ms/step - loss: 0.5566 - accuracy: 0.8070 - v
Epoch 10/10
313/313 [==============================] - 3s 10ms/step - loss: 0.4937 - accuracy: 0.8260 - v
```

3.定義show_train_history函式

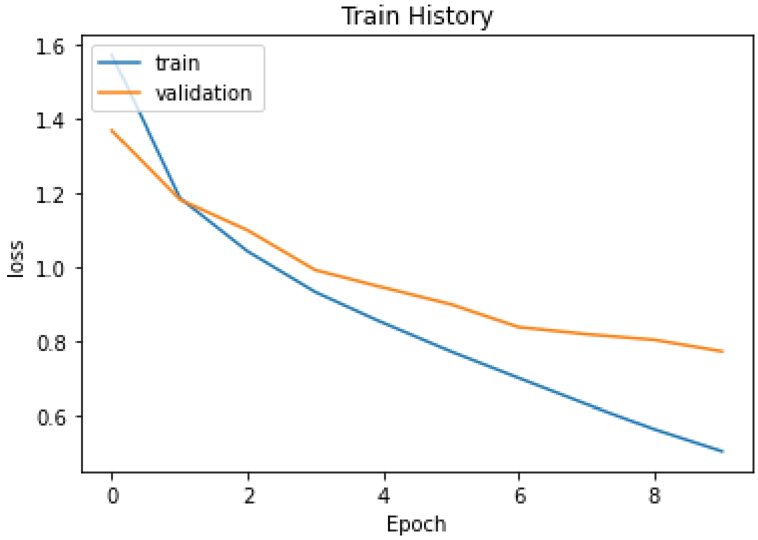
```
import  matplotlib.pyplot  as  plt                          #匯入matplotlib.pyplot模組，後續會使用
def  show_train_history(train_history,train,validation):     #定義show_train_history函數，輸入下列
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title('Train  History')                             #顯示圖的標題
    plt.ylabel(train)                                       #顯示y軸的標籤
    plt.xlabel('Epoch')                                     #設定x軸標籤是'Epoch'
    plt.legend(['train','validation'],loc='upper  left')     #設定圖例是顯示'train','validation',位
    plt.show()
```

4.畫出accuracy執行結果

```
show_train_history(train_history,'accuracy','val_accuracy')
```

5.畫出loss誤差執行結果

```
show_train_history(train_history,'loss','val_loss')
```



評估模型準確率

1.評估模型準確率

```
scores = model.evaluate(X_img_test_normalize,Y_label_test_OneHot,verbose=0)
scores[1]
```

```
    0.7325999736785889
```

進行預測

1.執行預測

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

```
    /usr/local/lib/python3.7/dist-packages/keras/engine/sequential.py:450: UserWarning: `model.pr
      warnings.warn('`model.predict_classes()` is deprecated and '
```

2.執行結果

```
prediction[:10]
```

```
    array([3, 8, 8, 0, 6, 6, 1, 6, 3, 1])
```

3.建立plot_images_labels_prediction函數

```
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()
```

4.定義label_dict字典

```
label_dict={0:″airplane″,1:″autombile″,2:″bird″,3:″cat″,4:″deer″,5:″dog″,6:″frog″,7:″horse″,8:″ship
```

5.預測結果

```
plot_images_labels_prediction(X_img_test,Y_label_test,prediction,0,10)
```

查看預測機率

1.使用測試資料進行預測

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

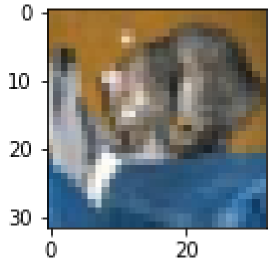
2.建立show_Predicted_Probability函數

```
def  show_Predicted_Probability(y,prediction,X_img,Predicted_Probability,i):
    print(’label:’,label_dict[y[i][0]],’predict’,label_dict[prediction[i]])          #顯示y(真實值)
    plt.figure(figsize=(2,2))                                                        #設定顯示影像
    plt.imshow(np.reshape(X_img_test[i],(32,32,3)))                                  #設定顯示影像
    plt.show()                                                                        #設定顯示影像
    for  j  in  range(10):                                                            #使用for迴圈,
        print(label_dict[j]+’Probability:%1.9f’%(Predicted_Probability[i][j]))
```

3.查看第0筆資料預測的機率

```
show_Predicted_Probability(Y_label_test,prediction,X_img_test,Predicted_Probability,0)
```

```
label: cat predict cat
```

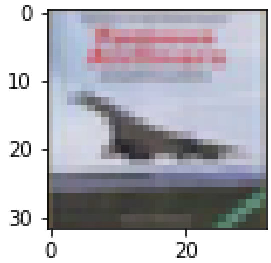


```
airplaneProbability:0.012357773
automobileProbability:0.000820371
birdProbability:0.005125463
catProbability:0.565821171
deerProbability:0.023896364
dogProbability:0.306629539
frogProbability:0.005355667
horseProbability:0.002768110
shipProbability:0.075323761
```

4.查看第3筆資料預測的機率

```
show_Predicted_Probability(Y_label_test,prediction,X_img_test,Predicted_Probability,3)
```

```
label: airplane predict airplane
```



```
airplaneProbability:0.573551536
automobileProbability:0.009738880
birdProbability:0.060519915
catProbability:0.012868146
deerProbability:0.023497602
dogProbability:0.000437201
frogProbability:0.001981785
horseProbability:0.002172791
shipProbability:0.309087425
truckProbability:0.006144671
```

顯示混淆矩陣(confusion matrix)

1查看prediction預測結果的形狀

```
prediction.shape
```

```
(10000,)
```

2.查看y_label_test真實值的shape形狀

```
Y_label_test.shape
```

```
(10000, 1)
```

3.將y_label_test真實值，轉換為1維陣列

```
Y_label_test.reshape(-1)
```

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

4.使用pandas crosstab建立混淆矩陣(confusion matrix)

```
import  pandas  as  pd                                                  #匯入pandas模組，後續會以
print(label_dict)                                                        #顯示label_dict字典，方便
pd.crosstab(                                                              #顯示pd.croostab建立混淆
        Y_label_test.reshape(-1),                                          #測試資料的真實值，使用
        prediction,                                                        #測試資料的預測結果
        rownames=['label'],                                                #設定行的名稱label
        colnames=['predict'])                                              #設定列的名稱predict
```

```
{0: 'airplane', 1: 'automobile', 2: 'bird', 3: 'cat', 4: 'deer', 5: 'dog', 6: 'frog',
```

predict	0	1	2	3	4	5	6	7	8	9
label										
0	807	10	33	24	17	2	11	12	51	33
1	19	837	12	14	6	4	12	6	12	78
2	57	4	615	59	108	42	77	22	10	6
3	27	4	70	556	92	134	65	41	5	6
4	16	2	50	55	755	12	39	60	9	2
5	15	2	56	202	73	575	26	44	3	4
6	6	3	27	56	51	16	827	7	6	1
7	15	0	41	35	72	38	8	784	0	7
8	81	41	20	20	12	10	5	5	776	30
9	30	73	12	32	6	5	10	18	20	794

建立3次的卷積運算神經網路

1.匯入所需模組

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

2.建立keras的Sequential模型

```
model  =  Sequential()
```

3.建立卷積層1與池化層1與池化層1

```
model.add(Conv2D(filters=32,
                 kernel_size=(3,  3),
                 input_shape=(32,  32,3),
                 padding='same',
                 activation='relu'))
model.add(Dropout(0.3))
model.add(Conv2D(filters=32,kernel_size=(3,3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,  2)))
```

4.建立卷積層2與池化層2建立卷積層2與池化層2

```
model.add(Conv2D(filters=64,
                 kernel_size=(3,  3),
                 padding='same',
                 activation='relu'))
model.add(Dropout(0.3))
model.add(Conv2D(filters=64,kernel_size=(3,  3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,  2)))
```

5.新增加卷積層3與池化層3與池化層3

```
model.add(Conv2D(filters=128,
                 kernel_size=(3,  3),
                 padding='same',
                 activation='relu'))
model.add(Dropout(0.3))
model.add(Conv2D(filters=128,kernel_size=(3,  3),activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2,  2)))
```

6.建立神經網路(平坦層、隱藏層1(2500個神經元)、隱藏層2(1500個神經元)、輸出層)

```
model.add(Flatten())
model.add(Dropout(0.3))
model.add(Dense(2500,activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1500,activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(10,activation='softmax'))
```

7.定義訓練方式

```
model.compile(loss='categorical_crossentropy',     #loss:設定損失函數(loss  function)
              optimizer='adam',                     #optimizer:設定訓練時的最優化方法，在深度學習使用
              metrics=['accuracy'])                 #設定評估模型的方式是accuracy準確率
```

8.訓練模型

```
train_history=model.fit(X_img_train_normalize,Y_label_train_OneHot,validation_split=0.2,epochs=50,b
```

```
Epoch 1/50
134/134 [==============================] - 8s 43ms/step - loss: 2.2042 - accuracy: 0.1607
Epoch 2/50
134/134 [==============================] - 5s 36ms/step - loss: 1.6467 - accuracy: 0.3876
Epoch 3/50
134/134 [==============================] - 5s 36ms/step - loss: 1.3796 - accuracy: 0.4983
Epoch 4/50
134/134 [==============================] - 5s 36ms/step - loss: 1.2368 - accuracy: 0.5531
Epoch 5/50
134/134 [==============================] - 5s 36ms/step - loss: 1.0776 - accuracy: 0.6112
Epoch 6/50
134/134 [==============================] - 5s 36ms/step - loss: 0.9814 - accuracy: 0.6494
Epoch 7/50
134/134 [==============================] - 5s 36ms/step - loss: 0.8982 - accuracy: 0.6775
Epoch 8/50
134/134 [==============================] - 5s 36ms/step - loss: 0.8186 - accuracy: 0.7085
Epoch 9/50
134/134 [==============================] - 5s 36ms/step - loss: 0.7781 - accuracy: 0.7237
Epoch 10/50
134/134 [==============================] - 5s 36ms/step - loss: 0.7113 - accuracy: 0.7459
Epoch 11/50
134/134 [==============================] - 5s 36ms/step - loss: 0.6657 - accuracy: 0.7607
Epoch 12/50
134/134 [==============================] - 5s 36ms/step - loss: 0.6013 - accuracy: 0.7832
Epoch 13/50
134/134 [==============================] - 5s 36ms/step - loss: 0.5658 - accuracy: 0.7985
Epoch 14/50
134/134 [==============================] - 5s 37ms/step - loss: 0.5286 - accuracy: 0.8111
Epoch 15/50
134/134 [==============================] - 5s 36ms/step - loss: 0.4902 - accuracy: 0.8262
Epoch 16/50
134/134 [==============================] - 5s 37ms/step - loss: 0.4798 - accuracy: 0.8262
Epoch 17/50
134/134 [==============================] - 5s 37ms/step - loss: 0.4267 - accuracy: 0.8461
Epoch 18/50
134/134 [==============================] - 5s 37ms/step - loss: 0.4070 - accuracy: 0.8540
Epoch 19/50
134/134 [==============================] - 5s 37ms/step - loss: 0.3825 - accuracy: 0.8639
Epoch 20/50
134/134 [==============================] - 5s 37ms/step - loss: 0.3569 - accuracy: 0.8725
Epoch 21/50
134/134 [==============================] - 5s 37ms/step - loss: 0.3468 - accuracy: 0.8763
Epoch 22/50
134/134 [==============================] - 5s 37ms/step - loss: 0.3223 - accuracy: 0.8846
Epoch 23/50
134/134 [==============================] - 5s 37ms/step - loss: 0.3028 - accuracy: 0.8937
Epoch 24/50
134/134 [==============================] - 5s 37ms/step - loss: 0.2817 - accuracy: 0.9003
```

```
Epoch 25/50
134/134 [==============================] - 5s 37ms/step - loss: 0.2741 - accuracy: 0.9028
Epoch 26/50
134/134 [==============================] - 5s 37ms/step - loss: 0.2565 - accuracy: 0.9097
Epoch 27/50
134/134 [==============================] - 5s 37ms/step - loss: 0.2477 - accuracy: 0.9121
Epoch 28/50
134/134 [==============================] - 5s 37ms/step - loss: 0.2332 - accuracy: 0.9185
Epoch 29/50
```

9.評估模型準確率

```
scores  =  model.evaluate(X_img_test_normalize,Y_label_test_OneHot,verbose=0)
scores[1]
```

```
0.7940000295639038
```

模型的儲存與載入

尚未成功!!!!尚未成功!!!!

```
train_history=model.fit(X_img_train_normalize,Y_label_train_OneHot,validation_split=0.2,epochs=5,ba
```

```
Epoch 1/5
313/313 [==============================] - 6s 19ms/step - loss: 0.2961 - accuracy: 0.8975 - v
Epoch 2/5
313/313 [==============================] - 6s 18ms/step - loss: 0.2887 - accuracy: 0.9004 - v
Epoch 3/5
313/313 [==============================] - 6s 19ms/step - loss: 0.2578 - accuracy: 0.9133 - v
Epoch 4/5
313/313 [==============================] - 6s 18ms/step - loss: 0.2505 - accuracy: 0.9146 - v
Epoch 5/5
313/313 [==============================] - 6s 18ms/step - loss: 0.2457 - accuracy: 0.9158 - v
```

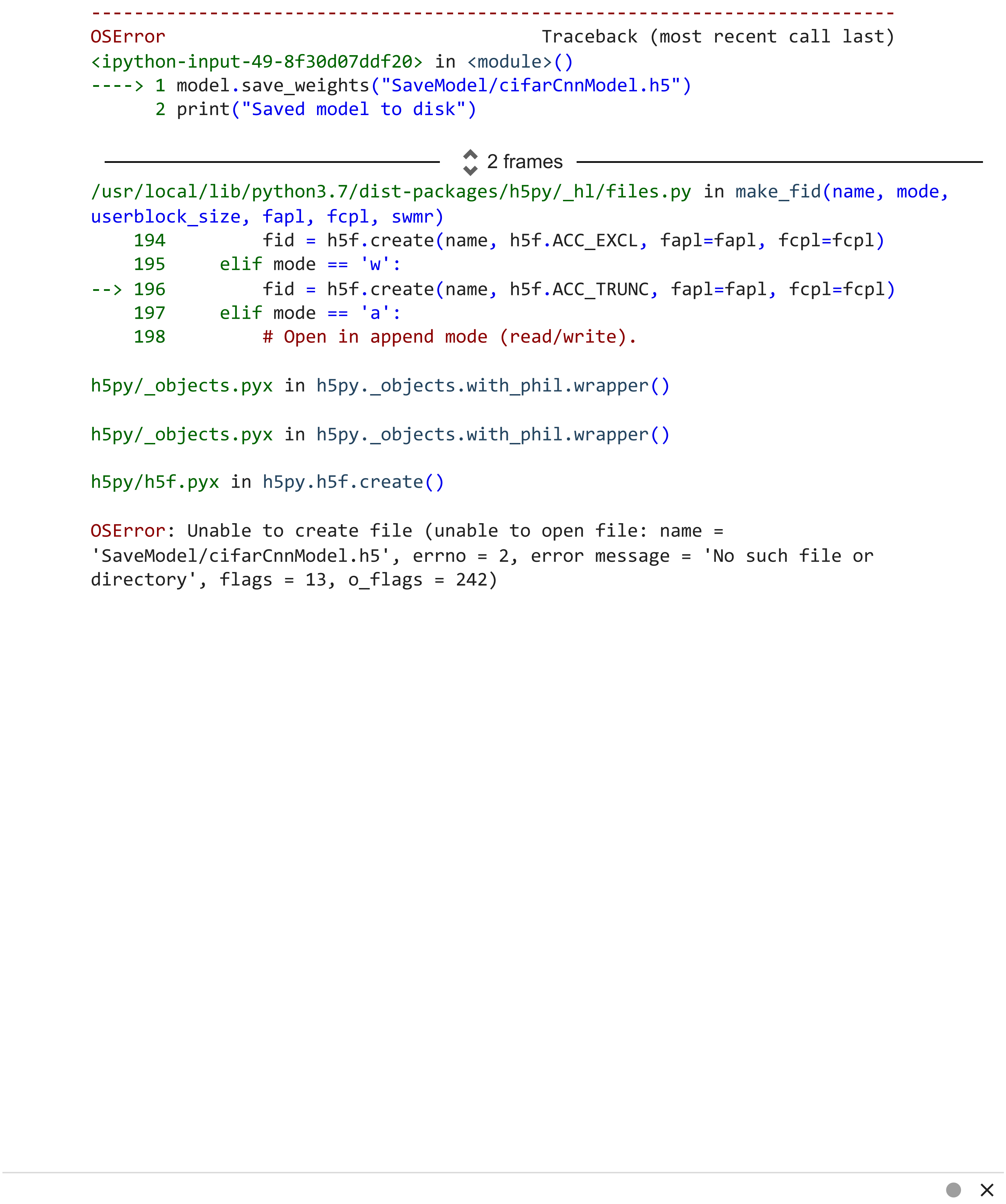
```
try:
    modle.load_weights("SaveModel/cifarCnnModel.h5")
    print("載入模型成功!繼續訓練模型")
except  :
    print("載入模型失敗!開始訓練一個新模型")
```

```
載入模型失敗!開始訓練一個新模型
```

```
pip  install  h5py
```

```
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (3.1.0)
Requirement already satisfied: numpy>=1.14.5; python_version == "3.7" in /usr/local/lib/pytho
Requirement already satisfied: cached-property; python_version < "3.8" in /usr/local/lib/pyth
```

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



```
---------------------------------------------------------------------------
OSError                                   Traceback (most recent call last)
<ipython-input-49-8f30d07ddf20> in <module>()
----> 1 model.save_weights("SaveModel/cifarCnnModel.h5")
      2 print("Saved model to disk")
```

```
/usr/local/lib/python3.7/dist-packages/h5py/_hl/files.py in make_fid(name, mode, userblock_size, fapl, fcpl, swmr)
    194         fid = h5f.create(name, h5f.ACC_EXCL, fapl=fapl, fcpl=fcpl)
    195     elif mode == 'w':
--> 196         fid = h5f.create(name, h5f.ACC_TRUNC, fapl=fapl, fcpl=fcpl)
    197     elif mode == 'a':
    198         # Open in append mode (read/write).

h5py/_objects.pyx in h5py._objects.with_phil.wrapper()

h5py/_objects.pyx in h5py._objects.with_phil.wrapper()

h5py/h5f.pyx in h5py.h5f.create()

OSError: Unable to create file (unable to open file: name = 'SaveModel/cifarCnnModel.h5', errno = 2, error message = 'No such file or directory', flags = 13, o_flags = 242)
```