```
# coding: utf-8

import numpy as np
import matplotlib.pyplot as plt
from keras.utils import np_utils

from keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()

for i in range(0,5):                              ##    劃出數字
        plt.subplot(1, 5, i+1)
        img = x_train[i]
        plt.imshow(img, cmap="gray")

print(y_train[:5])    ## show  出前 5 筆  y

print(x_train.shape)
x_train_reshape = x_train.reshape(60000, 784).astype('float32')
print(x_train_reshape.shape)
x_train_normalized = x_train_reshape / 255
print(x_train_normalized[:5])

print(y_train[:5])
y_train_onehot = np_utils.to_categorical(y_train)    ##  轉成  10 個 0/1  碼
print(y_train_onehot[:5])    ## show  出前 5 筆  y
```

```
#########################################################
from keras.models import Sequential
from keras.layers import Dense


model = Sequential()
model.add(Dense(units=256, input_dim=784, kernel_initializer="normal", activation="relu"))#也可以有不同模型
model.add(Dense(units=10, kernel_initializer="normal", activation="softmax"))
print(model.summary())
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

train_history = model.fit(x=x_train_normalized, y=y_train_onehot, validation_split=0.2, epochs=10, batch_size=200, verbose=2)

get_ipython().magic('matplotlib inline')
## IPython 有一組預先定義好的所謂的魔法函數（Magic Functions），你可以通過命令列的語法形式來訪問它們。

import matplotlib.pyplot as plt
def show_train_history(train_history, train, validation):
    plt.plot(train_history.history[train])
    plt.plot(train_history.history[validation])
    plt.title("Train History")
    plt.ylabel(train)
    plt.xlabel('Epoch')
    plt.show()

show_train_history(train_history, "acc", "val_acc") ##  訓練正確率圖

show_train_history(train_history, "loss", "val_loss") ##  訓練誤差圖
```

```
print(len(y_test))

x_test_reshape = x_test.reshape(10000, 784).astype("float32")
x_test_normalized = x_test_reshape / 255
y_test_onehot = np_utils.to_categorical(y_test)


scores = model.evaluate(x_test_normalized, y_test_onehot)

print("Accuracy: {}%".format(scores[1]))


import itertools
def plot_confusion_matrix(cm, classes, normalize=False, title="Confusion Matrix", cmap=plt.cm.Blues):
    plt.figure()
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
    thresh = cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, cm[i, j], horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")
```

```
        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')

results = model.predict_classes(x_test_reshape)

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, results)
plot_confusion_matrix(cm, range(0, 9))    ##  彩色混屯矩陣

incorrect = np.where(y_test != results)[0]    ##  抓出錯誤的樣本資料
print(incorrect[:5])

for i in range(0, 9):                          ##    劃出數字
        plt.subplot(3, 3, i+1)
        idx = incorrect[i]
        img = x_test[idx]
        plt.imshow(img, cmap="gray")
        plt.title("{}/{}".format(y_test[idx], results[idx]))

correct = np.where(y_test == results)[0]
for i in range(0, 9):
        plt.subplot(3, 3, i+1)
        idx = correct[i]
        img = x_test[idx]
        plt.imshow(img, cmap="gray")
        plt.title("{}".format(y_test[idx]))
```