逢　甲　大　學
資　訊　工　程　學　系

專題題目

用 CNN 神經網路做驗證碼辨識

指導教授：張哲誠

學　　生：D0886214劉冠杰

D0886261黃柏儒

D0843692陳威辰

D0813011楊子瑨

D0845367賴彥宇

D0813114蘇筱媛

中　華　民　國　一　百　一　十　年　五　月

摘要

驗證碼(CAPTCHA)這個詞最早於 2002 年被提出，直到現在，驗證碼的難度不斷提高，有些圖片連人類都無法輕易辨識，難度提高的原因不外乎就是機器學習技術的進步，本次專題就選擇一些圖像分割難度較低的驗證碼，來進行專題的實作。

實作範例中，我們使用 TensorFlow 二代結合 Keras，針對逢甲大學選課系統的驗證碼，將驗證碼圖片轉為矩陣，建立卷積神經網路(CNN)的模型，之後進行神經網路的訓練，最終將正確資料與預測資料做比對，得出正確率及各數字的誤判率。

目錄

圖目錄

第一章、模型訓練

1.1 抓取圖片

開始模型的訓練前，需要先獲取訓練資料，資料來源選擇逢甲大學選課系統的驗證碼，原因為驗證碼內容較簡易，若內容包含英文字母，或是含有底線、文字扭曲，甚至是辨識車輛、斑馬線...等，所需的訓練時間將大幅增加。

圖1.1 驗證碼

確定資料來源後，利用 python 的 os 與 requests 套件將圖片抓取下來，並將圖片檔命名為答案，方便此後進行圖片比對時使用。

```python
# -*- coding: utf-8 -*-
import requests
import os

image_links = "https://course.fcu.edu.tw/validateCode.aspx"
image_num = 20

for i in range(image_num):

    DIR = "tests"

    if not os.path.exists(DIR):
        os.mkdir(DIR)  # 建立資料夾

    img = requests.get(image_links)  # 下載圖片

    with open(DIR + '/' + str(i) + ".jpg", "wb") as file:  # 開啟資料夾及命名圖片檔
        file.write(img.content)  # 寫入圖片的二進位碼
```

圖 1.2 爬蟲程式碼

1.2 開始訓練

1.2.1 圖片分割

此段程式碼第 25 至 30 行的作用為定義圖片的分割方法，將圖片(統一為四位數)分割成單一數字，並以 one-hot 形式儲存，one-hot 形式僅允許一個位元為 1。

```
1   import numpy as np
2   import os, yaml
3   from munch import Munch
4   from sklearn.model_selection import train_test_split
5   from tensorflow import keras
6   from tensorflow.keras import layers
7   from tensorflow.keras import models
8   from tensorflow.keras.preprocessing.image import img_to_array
9   from tensorflow.keras.preprocessing.image import load_img
10
11  with open("config.yaml", mode="r", encoding="utf8") as f:
12    config = Munch(yaml.safe_load(f))
13
14  epochs = 10                         #訓練的次數
15  img_rows = None                     #驗證碼影像檔的高
16  img_cols = None                     #驗證碼影像檔的寬
17  digits_in_img = config.codeDigit    #驗證碼影像檔中有幾位數
18  x_list = list()                     #存所有驗證碼數字影像檔的array
19  y_list = list()                     #存所有的驗證碼數字影像檔array代表的正確數字
20  x_train = list()                    #存訓練用驗證碼數字影像檔的array
21  y_train = list()                    #存訓練用驗證碼數字影像檔array代表的正確數字
22  x_test = list()                     #存測試用驗證碼數字影像檔的array
23  y_test = list()                     #存測試用驗證碼數字影像檔array代表的正確數字
24
25  #將圖片各碼切割並存至list
26  def split_digits_in_img(img_array, x_list, y_list):
27      for i in range(digits_in_img):
28          step = 10
29          x_list.append(img_array[:, 5 + i * step:(i + 1) * step] / 255)
30          y_list.append(img_filename[i])  #將正確數字轉為one-hot格式
```

圖 1.3 驗證碼圖片切割

1.2.2 模型建立與訓練

定義完分割方式後，並設定好網路模型參數後，即可建立完模型，其中第 54 行程式碼的 padding 策略選擇 same 而非 valid，兩者差別在於 same 可以保證矩陣大小始終一致，此模型的訓練集為 192 張圖片，訓練次數 10。

```
32  img_filenames = os.listdir(config.trainImage)
33
34  for img_filename in img_filenames:
35      if config.imageType not in img_filename:
36          continue
37      img = load_img(config.trainImage + '/{0}'.format(img_filename), color_mode='grayscale')  #以灰階載入圖片
38      img_array = img_to_array(img)   #將圖片轉為矩陣
39      img_rows, img_cols, _ = img_array.shape
40      split_digits_in_img(img_array, x_list, y_list)  #切割圖片
41
42  #將資料拆解為訓練及測試用資料
43  y_list = keras.utils.to_categorical(y_list, num_classes=10)
44  x_train, x_test, y_train, y_test = train_test_split(x_list, y_list)
45
46
47  if os.path.isfile(config.modelName):  #確認是否已有模型
48      model = models.load_model(config.modelName)
49      print('Model loaded from file.')
50  else:  #若無，建立模型
51      model = models.Sequential()
52      model.add(layers.Conv2D(32, kernel_size=(3, 3), activation='relu'))
53      model.add(layers.Conv2D(64, (3, 3), activation='relu'))
54      model.add(layers.MaxPooling2D(pool_size=(2, 2), padding='same'))
55      model.add(layers.Dropout(rate=0.25))
56      model.add(layers.Flatten())
57      model.add(layers.Dense(128, activation='relu'))
58      model.add(layers.Dropout(rate=0.5))
59      model.add(layers.Dense(10, activation='softmax'))
60      print('New model created.')
```

圖 1.4 建立模型

```
65  #開始訓練
66  model.fit(np.array(x_train), np.array(y_train), batch_size=digits_in_img,
67            epochs=epochs, verbose=1, validation_data=(np.array(x_test), np.array(y_test)))
68
69  #顯示準確率
70  loss, accuracy = model.evaluate(np.array(x_test), np.array(y_test), verbose=0)
71  print('Test loss:', loss)
72  print('Test accuracy:', accuracy)
73
74  model.save(config.modelName)
```

圖 1.5 訓練模型

```
Epoch 1/10
144/144 [==============================] - 1s 2ms/step - loss: 2.2734 - accuracy: 0.1374 - val_loss: 1.8355 - val_accuracy: 0.3089
Epoch 2/10
144/144 [==============================] - 0s 2ms/step - loss: 1.6068 - accuracy: 0.4640 - val_loss: 0.8819 - val_accuracy: 0.7120
Epoch 3/10
144/144 [==============================] - 0s 1ms/step - loss: 0.7399 - accuracy: 0.7838 - val_loss: 0.3133 - val_accuracy: 0.9686
Epoch 4/10
144/144 [==============================] - 0s 1ms/step - loss: 0.3668 - accuracy: 0.9141 - val_loss: 0.1815 - val_accuracy: 0.9738
Epoch 5/10
144/144 [==============================] - 0s 1ms/step - loss: 0.2811 - accuracy: 0.9302 - val_loss: 0.1368 - val_accuracy: 0.9791
Epoch 6/10
144/144 [==============================] - 0s 1ms/step - loss: 0.2054 - accuracy: 0.9335 - val_loss: 0.1064 - val_accuracy: 0.9738
Epoch 7/10
144/144 [==============================] - 0s 1ms/step - loss: 0.1484 - accuracy: 0.9549 - val_loss: 0.0917 - val_accuracy: 0.9791
Epoch 8/10
144/144 [==============================] - 0s 1ms/step - loss: 0.1145 - accuracy: 0.9796 - val_loss: 0.0550 - val_accuracy: 0.9895
Epoch 9/10
144/144 [==============================] - 0s 1ms/step - loss: 0.1392 - accuracy: 0.9692 - val_loss: 0.0772 - val_accuracy: 0.9895
Epoch 10/10
144/144 [==============================] - 0s 1ms/step - loss: 0.1480 - accuracy: 0.9613 - val_loss: 0.0698 - val_accuracy: 0.9738
Test loss: 0.06977220624685287
Test accuracy: 0.9738219976425171
```

圖 1.6 訓練過程

第二章、數字預測

有了訓練過的模型，接下來就能實際來分辨數字，辨識前同樣需要對圖片進行分割，保險起見也要確認是否已有模型存在。

```python
# -*- coding: utf-8 -*-
import numpy as np
import os, yaml
from munch import Munch
from tensorflow.keras import models
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.preprocessing.image import load_img

with open("config.yaml", mode="r", encoding="utf8") as f:
  config = Munch(yaml.safe_load(f))

img_rows = None
img_cols = None
digits_in_img = config.codeDigit
model = None
np.set_printoptions(suppress=True, linewidth=150, precision=9, formatter={'float': '{: 0.9f}'.format})
actual_num_count = np.zeros([10, 1])  #各數字實際出現次數統計
predit_error_count = np.zeros([10, 1])  #各數字預測錯誤次數

# 將圖片的4碼切割
def split_digits_in_img(img_array):
    x_list = list()
    for i in range(digits_in_img):
        step = 10  #數字寬度
        x_list.append(img_array[:, 5 + i * step:(i + 1) * step] / 255)
    return x_list

#預測
if os.path.isfile(config.modelName):  #確認是否已有模型
    model = models.load_model(config.modelName)
else:
    print('No trained model found.')
```

圖 2.1 預測程式碼

圖 2.2 中程式碼前半部為模型預測一個數字可能的機率，從中挑出機率最高的可能性當作預測數字；最後 51 到 61 行的程式碼則為輸出預判結果，並將正確數字與預測數字做比對，且統計誤判率，圖 2.4 為 100 張測試圖片的測試結果，最終得到的正確率高達 98%。

```
34  img_filenames = os.listdir(config.testImage)
35  match_count = 0  #計算正確次數
36  for img_filename in img_filenames:
37    for i in img_filename[:4]:  actual_num_count[int(i)] += 1
38    img = load_img(config.testImage + '/' + img_filename, color_mode='grayscale')  #以灰階載入圖片
39    img_array = img_to_array(img)  #轉為矩陣
40    img_rows, img_cols, _ = img_array.shape
41    x_list = split_digits_in_img(img_array)  #切割數字
42
43    varification_code = list()  #預測的數字
44    for i in range(digits_in_img):
45        confidences = model.predict(np.array([x_list[i]]), verbose=0)
46        result_class = model.predict_classes(np.array([x_list[i]]), verbose=0)
47        varification_code.append(str(result_class[0]))
48        if result_class[0] != int(img_filename[i]):  predit_error_count[int(img_filename[i])] += 1
49        #顯示各位數所有可能的機率及結果
50
51    predit_code = ''.join(varification_code)
52    actual_code = img_filename[:4]
53    match = (predit_code==actual_code)
54    print('Predicted:', predit_code, 'actual:', img_filename[:4], 'match:', match)
55    if match:  match_count += 1
56
57  print('number of test image:', len(img_filenames), 'match number:', match_count)
58  print('match percentage:', match_count/len(img_filenames)*100, '%')
59  print("各數字誤判機率：")
60  for i in range(10):
61    print(i, "：", "{:.3f}".format(predit_error_count[i][0]/actual_num_count[i][0]*100), '%')
```

圖 2.2 預測與計算誤判率

```
Predicted: 0995 actual: 0995 match: True
Predicted: 1070 actual: 1070 match: True
Predicted: 1085 actual: 1085 match: True
Predicted: 1166 actual: 1166 match: True
Predicted: 1490 actual: 1490 match: True
Predicted: 2000 actual: 2000 match: True
Predicted: 2200 actual: 2200 match: True
Predicted: 2305 actual: 2305 match: True
Predicted: 2766 actual: 2766 match: True
Predicted: 2851 actual: 2851 match: True
Predicted: 3258 actual: 3258 match: True
Predicted: 3483 actual: 3480 match: False
Predicted: 3576 actual: 3576 match: True
Predicted: 3724 actual: 3724 match: True
Predicted: 3905 actual: 3905 match: True
Predicted: 4076 actual: 4076 match: True
Predicted: 4161 actual: 4161 match: True
Predicted: 4170 actual: 4170 match: True
Predicted: 4401 actual: 4401 match: True
Predicted: 4597 actual: 4597 match: True
Predicted: 5001 actual: 5001 match: True
Predicted: 5007 actual: 5007 match: True
Predicted: 5192 actual: 5192 match: True
Predicted: 5672 actual: 5672 match: True
Predicted: 5768 actual: 5768 match: True
Predicted: 5949 actual: 5949 match: True
Predicted: 6378 actual: 6278 match: False
Predicted: 6363 actual: 6363 match: True
Predicted: 6434 actual: 6434 match: True
Predicted: 6502 actual: 6502 match: True
Predicted: 6698 actual: 6698 match: True
Predicted: 7044 actual: 7044 match: True
Predicted: 7788 actual: 7788 match: True
Predicted: 7873 actual: 7873 match: True
Predicted: 8373 actual: 8373 match: True
Predicted: 8469 actual: 8469 match: True
Predicted: 8640 actual: 8640 match: True
Predicted: 9370 actual: 9370 match: True
Predicted: 9427 actual: 9427 match: True
Predicted: 9609 actual: 9609 match: True
Predicted: 9794 actual: 9794 match: True
```

圖 2.3 部分預測結果

```
number of test image: 100 match number: 98
match percentage: 98.0 %
各數字誤判機率：
0 : 0.000 %
1 : 0.000 %
2 : 0.000 %
3 : 0.000 %
4 : 0.000 %
5 : 0.000 %
6 : 4.878 %
7 : 0.000 %
8 : 0.000 %
9 : 0.000 %
```

圖 2.4 正確率與數字誤判率

參考資料

[1] https://tf.wiki/zh_hant/basic/models.html

[2] https://medium.com/chiukevin0321/tensorflow%E8%88%87keras%E5%9F%BA%E6%9C%AC%E4%BB%8B%E7%B4%B9-621352fc7150

[3] https://course.fcu.edu.tw/

專題 PPT 講解

https://youtu.be/r4ZjlCsQI60