

深度學習期末報告

主題:

CAPTCHA Recognition

組別: Group 9

組員:

309704033 周德鎧

309704028 張育誠

OUTLINE

- 一、 研究動機和背景描述
- 二、 Datasets & Data Collection
- 三、 Part 1.
- 四、 Part 2.
- 五、 結論
- 六、 參考資料

研究動機和背景描述

在深度學習的課程中，作業 1-2 讓我們學習到如何使用 CNN 來對圖片進行辨識，因此有了啟發，我們希望能夠透過 CNN based 的方式來進行 image recognition 的 task，這是最初的動機來源。

接著就開始進行相關資料的收尋，我們希望把 task 放在「生活化、實用性」上，也就是訓練的模型是真的能幫助我們解決生活上的問題，最後經過和組員的討論之後，我們決定 task 為『CAPTCHA

Recognition』。

現在不論什麼網頁幾乎都需要驗證碼來防止濫用系統，交大的選課系統也不例外，如右圖。



因此我們希望可以利用訓練一個 model 來成功辨識驗證碼，這樣就不用每一次登入時都要重複的輸入煩人的驗證碼，雖然自從學校變成了陽明交大之後選課系統就不再需要驗證碼了，但是數位教學平台的 E3 如果直接從網頁登入的話，還是需要輸入驗證碼的，如右圖。

因此我們選擇這個主題的目標就是

「訓練一個 model 來自動化的辨識這些驗證碼，也就是 CAPTCHA Recognition」。



Datasets & Data Collection

接下來將要介紹如何我們的數據是從哪裡來的以及怎麼獲得這些數據。

首先是先說明 Datasets。總共有兩個不同的 Datasets。

(最後有三個，後面會說明原因)

1. NCTU 選課系統 (easy dataset) (Sample size = 2600)

這個 Dataset 將 NCTU 選課系統的登入畫面的驗證碼去收集圖片，我們利用爬蟲來抓取圖片。因為此 dataset 的噪音相對比較小，因此我們會簡稱此 dataset 為 easy dataset。

2. 監理服務網-車輛通訊地址查詢 (hard dataset)

這個 dataset 是從監理站的車輛通訊地址查詢上面獲得的驗證碼圖片，我們也是利用爬蟲來抓取圖片。反之，這個 dataset 的噪音明顯比較複雜，因此簡稱為 hard dataset。

以下兩張圖分別是示意圖:

Part1:
Easy Dataset



Part2:
Hard Dataset (Digits + Letters)



Dataset 說明:

我們選擇兩個不同複雜度的 dataset 有兩個目的:

1. 比較 preprocessing

透過兩種不同複雜度的 dataset 可以觀察出不同的 preprocessing 在其中發揮的程度有甚麼差異。

2. 比較 model

因為不同的複雜度需要不同的前處理，因此當前處理完的 data 餵給 model 的樣子也會有差異，所以該怎麼選擇合適的 model 可以透過兩個不同的 dataset 來觀察差異。

因此在進行訓練 model 的過程中，我們根據 dataset 分成 **part 1** 和 **part 2** 兩個部分來進行，因為我們必須根據噪音的複雜度去客製化個別 dataset 的降噪程度。

每個 Part 的重點有三個部分:

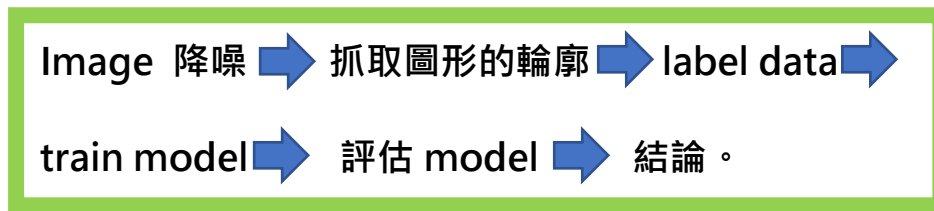
1. Data Preprocessing
2. Models
3. Evaluation

最後結合兩個 Part 的實驗結果得到一些結論。

Part 1.

在 part 1. 中，我們使用的是 NCTU 選課系統收集來的驗證碼，是屬於比較簡單處理的 dataset。

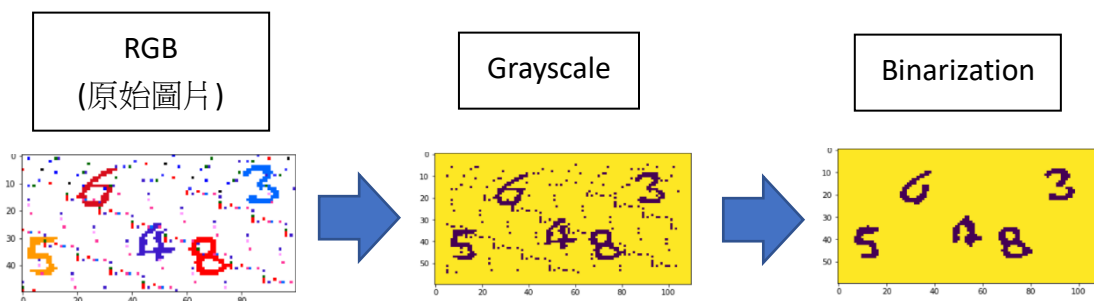
大致上 Part 1. 和 Part 2. 的處理流程為



1. Preprocessing

Part 的噪音都是一些不同顏色點點的背景，因此我們的第一步是將圖片轉成單一色調，也就是 **RGB to Grayscale**，接下來為了更凸顯特徵(e.g.，數字的形狀)，因此我們把 image 轉成**二元圖片**，這樣帶來的好處是可以同時**降低計算量**並且同時**增強對照**的強度。


整個圖片前處理的流程大致是以下的操作：



有了比較清楚的 image 之後就可以透過抓取圖片輪廓的方式來偵測 image 是哪一些數字。抓取輪廓的方法是使用 Opencv 的 cv2 來達成。在 easy dataset 中，因為我們能夠透過降噪的方式把 images 轉成 clean images，因此在抓取輪廓上比較容易，只需要設定一些條件就能完整的抓取數字，在抓取數字後，驗證碼就會從原本一張 image 變成五張數字的 images，最後我們再依照這些數字應該屬於哪些類別去分類，接下來根據建立好的資料庫取出來並且轉乘灰階，以下是關於抓數字輪廓和 label 的示意圖：



在上一個 step 我們得到了乾淨的數字圖片，因此可以來建立 model or classifier 來對這些圖分類。但是在餵給 model 之前還要再做一些 preprocessing。

- **Resize image:** (600, 400)  (50, 33)

設定這個 shape 是為了不希望太大的 pixel 要訓練很久。

- **Training & testing 比例:** training=0.7, testing=0.3

(共有 13000 張 image, 即 2600 張驗證碼)

- **Standardization:** 對圖片標準化

- **One-hot Encoding for label:** label 轉成 one-hot vector

準備好所有 dataset 的 preprocessing 之後就可以準備開始 train model 了。

2. Model

因為 Part 1. 在降噪的處裡效果非常好，因此相當於原本 CAPTCHA

Recognition 的任務就變成了類似 MNIST 手寫辨識的任務。

因此不需要用到太複雜的 model 就可以有很高的 accuracy。

我們在 Part 1. 總共使用了三種 model 來做 evaluation，

A. **MLP**

B. **CNN**

C. **SVM**

以下分別簡述一下 model 的架構

MLP

- Activatoin: ReKU
- Hidden layer: (30, 30, 30)
- Iteration: 10000 次

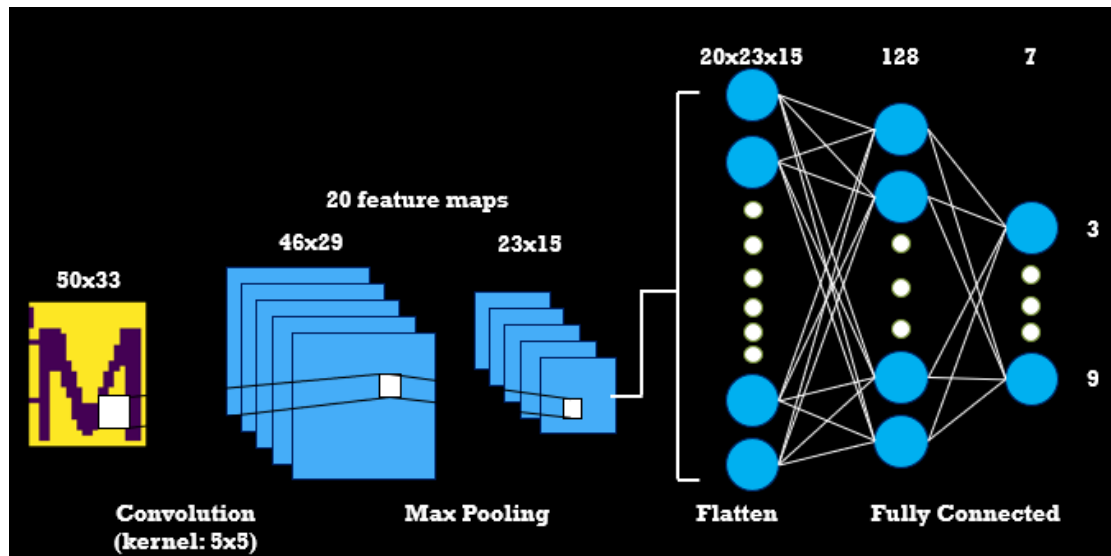
```
In [44]: from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes = (30,30,30), activation='relu', max_iter = 10000)
mlp.fit(X_train,y_train)

Out[44]: MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
beta_2=0.999, early_stopping=False, epsilon=1e-08,
hidden_layer_sizes=(30, 30, 30), learning_rate='constant',
learning_rate_init=0.001, max_fun=15000, max_iter=10000,
momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
power_t=0.5, random_state=None, shuffle=True, solver='adam',
tol=0.0001, validation_fraction=0.1, verbose=False,
warm_start=False)

In [45]: #data_predict = mlp.predict(X_test)
mlp.score(X_test, y_test)

Out[45]: 0.9980334316617503
```

CNN



- Convolutional layer: 一層
- Output channel: 20

```
CNN = Sequential()

# Create CN Layer 1
CNN.add(Conv2D(filters=20, kernel_size=(5,5), padding='same', input_shape = (50,33,1), activation = 'relu'))
|
# Create Max-Pool 1
CNN.add(MaxPooling2D(pool_size=(2,2)))

# Add Dropout Layer
CNN.add(Dropout(0.1))

#Add Flatten Layer
CNN.add(Flatten())

#Add NN Layer
CNN.add(Dense(128, activation='relu'))

# Add Dropout Layer
CNN.add(Dropout(0.3))

#Add Output Layer
CNN.add(Dense(10, activation='softmax'))

CNN.summary()
```

```
scores = CNN.evaluate(X_test_cnn, y_test_onehot)
print('acc:', scores[1])
```

```
4068/4068 [=====] - 1s 277us/step
acc 0.997787594795227
```

SVM

由於 SVM 有 kernel 和 hyper-parameter C 要選擇，因此我們使用 Grid Search 針對這兩個超參數來測試，最後選出最好的結果為

- C: 1
- Kernel: RBF

```
{'C': 1, 'kernel': 'rbf'}  
SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,  
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',  
    max_iter=-1, probability=False, random_state=None, shrinking=True,  
    tol=0.001, verbose=False)  
svc acc: 0.9995083579154376  
<Figure size 1440x1440 with 0 Axes>
```

3. Evaluation

- MLP 的準確率為 99.8 %
- CNN 的準確度為 99.77 %
- SVM 的準確度為 99.995 %

接下來換成 Part 2. Dataset 來 model。

Part 2.

在 part 2. 中的驗證碼與之前有兩個不同之處:

- i. 不再只是單純的數字組合，同時包含大寫英文字母。
- ii. 背景的噪音更大，有些噪音會穿過需要辨識出來的字。

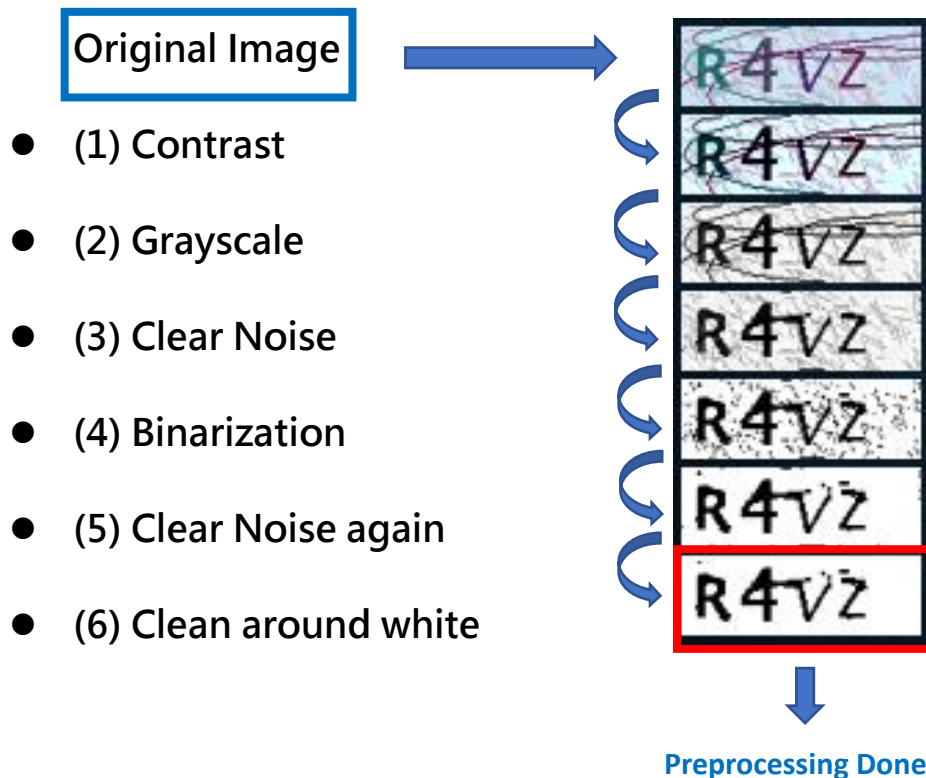
由於上面兩個因素，使得我們再前處理的部分會遇到困難，

不過一開始著手處理 Part 2. 的資料也希望用一樣的手法來做看看，但途中卻碰到問題，導致我們必須想其他辦法來解決，這些在等一下 Preprocessing 的部分都會詳細說明。

1. Preprocessing

在 Part 2. 的 Preprocessing 將使用更多的前處理手法。

我們將原始的驗證碼經過以下六個降噪的 Step:

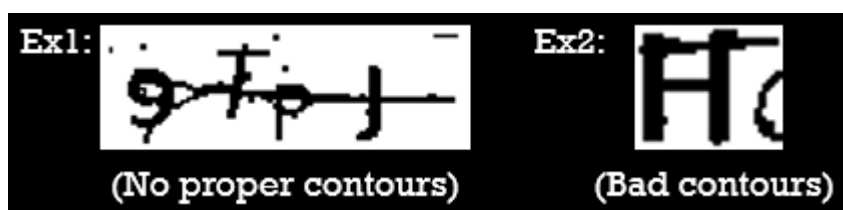


Problem

在處理完 Image 的前處理之後，我們發現 Part 2. 的 dataset 即便做了很多前處理，也不能讓噪音完全消失，因此在接下來抓取輪廓的階段產生兩個嚴重的問題。

1. Hard Over-lapping

即便使用很多的降噪技術，最後的圖片有時還是有著許多噪音，我們沒辦法完美地分開它們，導致抓取物件輪廓時，經常發生抓錯的問題，下面舉兩個例子：



從右邊的例子可以發現本來要抓 H 的輪廓，因為噪音還存在的關係，連同臨近字的部分也抓進去了，如果直接把這樣的圖片去 train 是會有問題的。

2. Hard Balance

有時候希望把噪音降到最小，但是同時也會傷害資訊，也就是有時候會把字砍掉一部分，以下是一個例子：

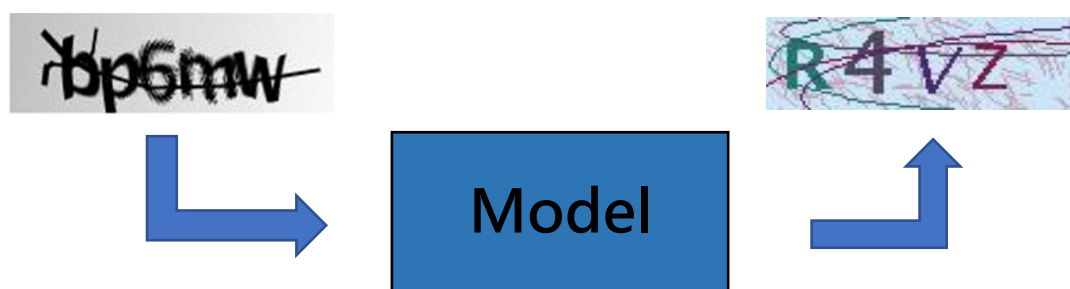


當使用很大的降噪時，K 變成了兩截，而 9 甚至直接被抹去了一整條，使得抓輪廓時完全抓不到 9。

綜合上面這些問題，導致我們沒辦法順利的像 Part 1.，先透過抓取輪廓得到每一個驗證碼數字+字母的 Image 再 label，這個情況變成我們必須一張一張去 label 才行，沒有其他辦法，但是一張一張 label 的方法效率比起 part 1. Label 的方式效率差異很大，在經過 Part 1. 的 label 之後，由於時間有限，我們沒有辦法在 part 2. 用更多的時間在 label 上。

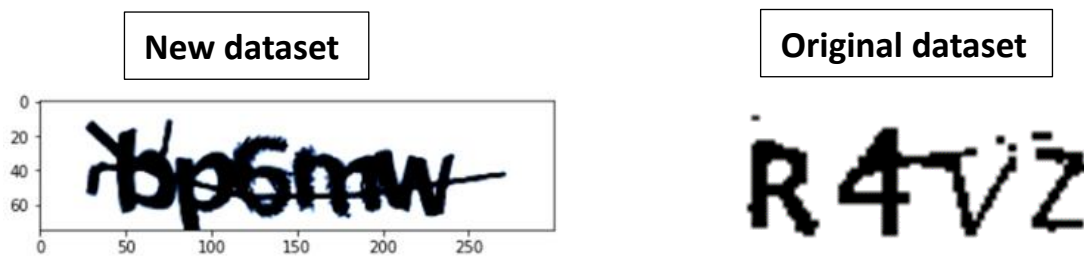
這個情況使得我們在 part 2. 面臨沒有 “label” 的問題。

因此我們尋找新的 Dataset，也就是開頭提到的第三個 Dataset，整體的想法為「因為時間不足和 Image 更複雜的關係，我們在 part 2. 沒有辦法完成大量 label，因此轉而尋找新的 dataset，但是同時又希望能夠解決原本設定的問題，所以會先用新的 dataset 做 training，接著把 train 好的 model 餵給我們本來想要做的 dataset 上，看看能不能成功」。



但是新的 dataset 明顯跟原本的 dataset 不一樣，如果這樣直接拿來 train 會有問題，因此我們打算先把兩個不同的 dataset 透過降噪的方式，使兩個 dataset 越接近越好。

下面兩張圖分別是降噪之後的 dataset。

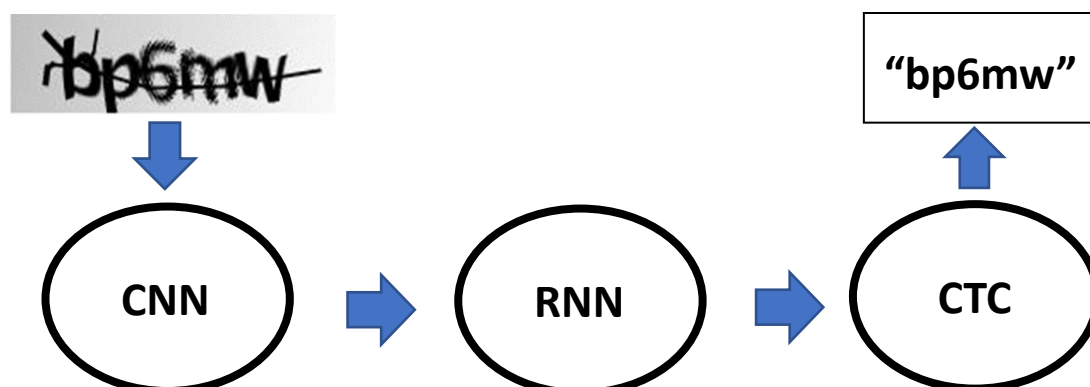


接下來我們就會使用前處理後的 dataset 訓練 model。

2. Model

我們將使用的 model 為 **CNN + RNN + CTC Loss**。

下面是這個 model 簡單的示意圖



model 說明

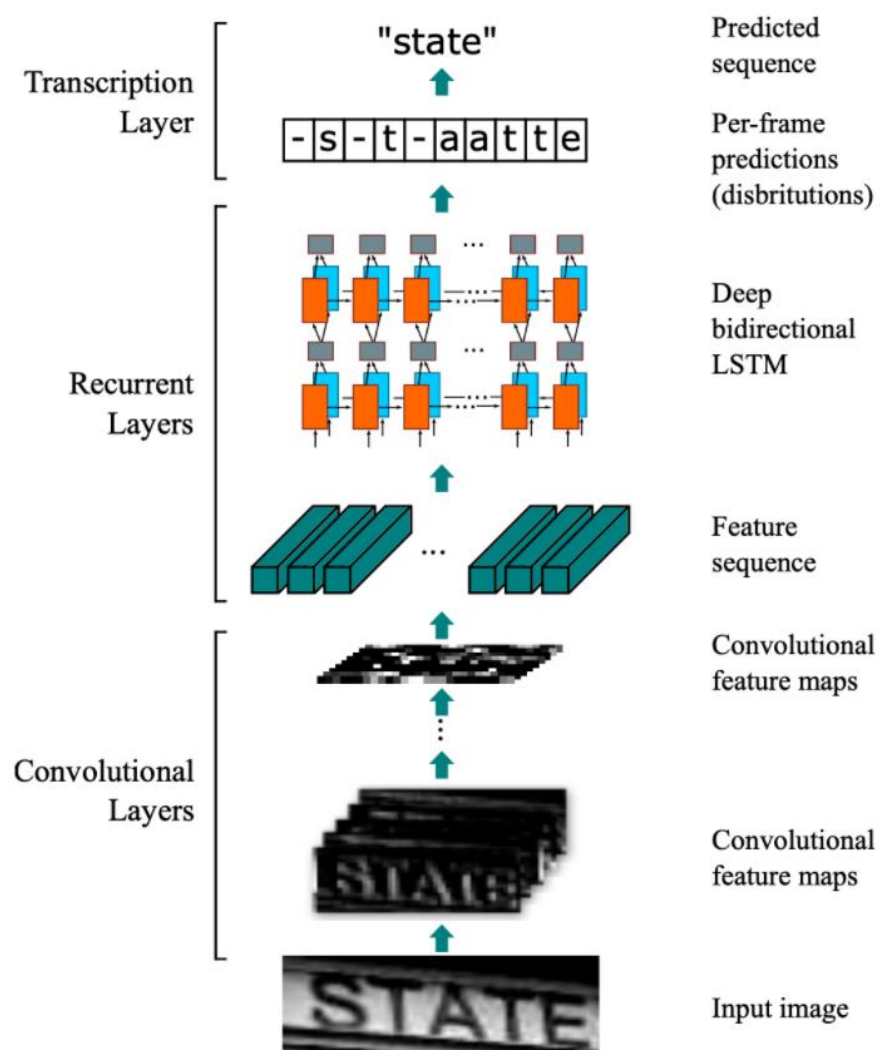
這裡說明一下 model 為什麼要這樣子設計。

這個 model 簡稱 CRNN+CTC，起源於下面這篇 paper

[An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition \(2015\), Baoguang Shi et al.](#)

在我們的 task 中，CAPTCHA Recognition 基本上就是一種圖片的辨識，因此在 Image 辨識中常用的手法就是使用 CNN 先進行**提取特徵**操作，接下來再透過 RNN 計算序列的 Information，RNN 在這裡扮演的腳色相當於每個時間點 output 出對應的字的分數，也就是在某個時間點 machine 認為最有可能 output 出哪一個字，最後再透過 CTC 的 loss function 去計算綜觀每個時間點的 loss。

整個 model 的結構可以參考原始 paper 上畫的結構圖：



其中我們把原本 LSTM 的部份換成 GRU。

Model 實作的部份我們使用 Pytorch 來完成。

因為 Pytorch 本身就有寫好的 CTC Loss 因此我們只需要建立 CNN 和 RNN 的部份就可以。

```
class CaptchaModel(nn.Module):
    def __init__(self, num_chars):
        super(CaptchaModel, self).__init__()
        self.conv_1 = nn.Conv2d(3, 128, kernel_size=(3, 3), padding=(1, 1))
        self.pool_1 = nn.MaxPool2d(kernel_size=(2, 2))
        self.conv_2 = nn.Conv2d(128, 64, kernel_size=(3, 3), padding=(1, 1))
        self.pool_2 = nn.MaxPool2d(kernel_size=(2, 2))
        self.linear_1 = nn.Linear(1152, 64)
        self.drop_1 = nn.Dropout(0.2)
        self.lstm = nn.GRU(64, 32, bidirectional=True, num_layers=2, dropout=0.25,
        self.output = nn.Linear(64, num_chars + 1)
```

使用 nn.CTCLoss

```
loss = nn.CTCLoss(blank=0) (
    log_probs, targets, input_lengths, target_lengths
)
return x, loss
```

接下來就開始 train model。

3. Evaluation

我們總共跑 200 個 epoch

以下是訓練的情況:

Epoch = 23	Epoch = 99
<pre>[('cd6p4', 'd4'), ('y5dpp', 'dp'), ('yx2d4', 'd4'), ('n4xx5', '4'), ('n7ebx', 'b'), ('2b827', ''), ('px2xp', 'p'), ('x6pdb', 'db'), ('4fp5g', '4pg'), ('en4n4', '4')]</pre>	<pre>[('cd6p4', 'cd6p4'), ('y5dpp', 'y5dp'), ('yx2d4', 'yx2d4'), ('n4xx5', 'n4x5'), ('n7ebx', 'n7ebx'), ('2b827', '2b827'), ('px2xp', 'px2xp'), ('x6pdb', 'x6pdb'), ('4fp5g', '4fp5g'), ('en4n4', 'en4n4')]</pre>
Accuracy = 0.0	Accuracy = 0.99

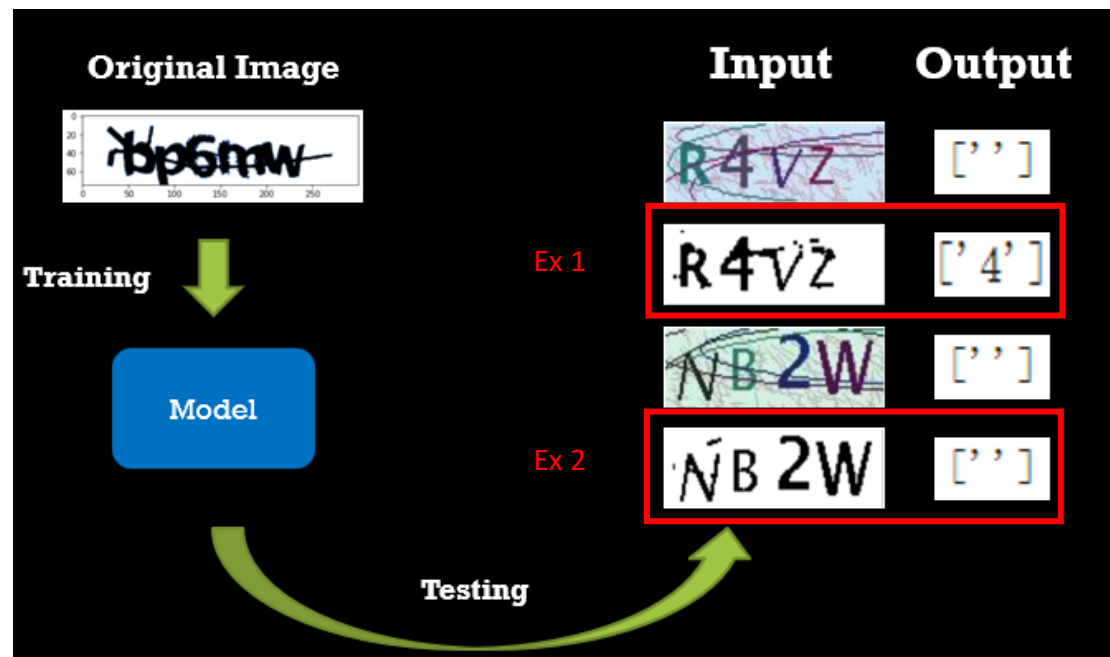
當 model 還沒訓練起來之前，基本上 output 都會是我們設定的特別 token，表示說 machine 認為 output 都不在 label 之中，接下來訓練到 99 epochs，machine 差不多能夠辨識字體了。但是還是有一點小問題，如果有疊字出現，model 會把它們刪掉只保留一個，因此這是可以改進的地方。但是這邊可以發現

「model 經過我們使用的 model 後，只要是同一個 Dataset 的資料，testing Accuracy 達到 0.99，可以說 model 是非常有用的」

根據前面訓練好的 model 接下來我們要把原本 Part 2.

Dataset 餵給 model 看看表現如何。

下面是示意圖：



我們發現因為 dataset 本身的內容物不太一樣，

比如說 training dataset 的英文字是小寫，但是我們想要

Inference 的 dataset 卻是大寫英文，並且加上有些字甚至在 training data 中沒有出現過，因此我們把評估表現的標準放在有重和的地方。

我們可以看到在 EX 1 即便是完全不同的 dataset，因為數字 4 曾經出現在 training dataset 之中，因此 model 還是能夠正常 predict 出來；但是也有失敗的例子，EX 2 雖然有很完美的 2，但是 model 卻無法判斷出來，我們認為這是因為 train 可能過

於 over-fitting，因此把 training dataset 的噪音也當作學習目標學進去了，導致清楚的數字 model 看不懂。

補充說明

這裡要說明一下，雖然最後在 Part 2. 的評估上基本上算是不可行的，但這是因為我們沒有時間 label Part 2. 的 data 所使用的其他辦法，如果單純評估 training dataset 在自己的 dataset，那麼表現其實非常良好，因此如果未來有時間可以把每一張圖片都做 label，那麼利用這個 model 就可以解決 CAPTCHA 辨識的問題。

結論

Part 1. 的結果可以發現，如果 dataset 比較簡單，那麼後續的 model 也很簡單就能處理；

反之 Part 2. Dataset 比較複雜，我們必須用其他方法來處理，比如我們使用 CRNN+CTC 這個 model 來處理。

在 Part 2. 因為沒有時間 label，所以我們選擇把 model train 在新的 dataset 上，並且該 dataset 有 label，希望最後 model 能夠學習如何辨識文字。雖然最後我們把訓練完的 model 帶回本來要測試的 Part 2. Dataset 上，結果並不理想，但是本質上就不一樣的 dataset，model 學到的東西就不一樣，因此這個結果也還算合理，因為這是 Part 2. 沒有 label 的問題。

根據這些結果下面是總結

1. Preprocessing 很重要

CAPTCHA Recognition 雖然可以當作一般的圖形辨識問題，但是處理噪音是這個 task 最核心的問題所在，因此怎麼操作前處理是非常中要的，我們在實作中都是不斷的調前處理的參數，調到效果不錯之後才餵給 model，這樣 model 才能有很好的預測能力。

2. 降噪和保留資訊之間必須做 trade-off

我們在把資料餵給 model 前都會做前處理，如果前處理做太多，會把原本的資訊給抹去，也就是連問字的部份也去掉了，如果前處理太少，則噪音沒有去掉，Model 會連背景噪音都學進去，導致 generalization 能力變得很差，因此怎麼找到這兩者之間的平衡，是我們在整個實作中花費時間除了 label 最多的地方。

3. Model 學習必須是 dataset by dataset 的

雖然我們把比較複雜的 model 訓練得很好，但這不代表把 model 用來預測簡單的 dataset 也能如此，原因可能是 model 怎麼學習的我們並不清楚，很有可能 model 連噪音的部份也 learn 進去，以至於簡單的 dataset 變得很難預測，因此如果要做 CAPTCHA 辨識最好的辦法可能就是根據不同 dataset 去訓練 model，雖然這樣花時間，但是對於 model 有看過的資料辨識率其實很高。

參考資料

1. [An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition](#)
2. [OCR model for reading Captchas](#)