Oneplus 2017 summer intern
Caffe2 x CaptcahCreck Report

Eric Chiang

環境建置

使用環境

CPU: i5-4430 3.00GHz
GPU: Nvidia GTX 950 2G
Memory: 8 + 4 + 4 G
OS: Ubuntu 16.04 LTS
Cudnn: 8.0

相依環境建置

基本上只要根據官網教學即可建置完成，GPU支援的部分則是可選。

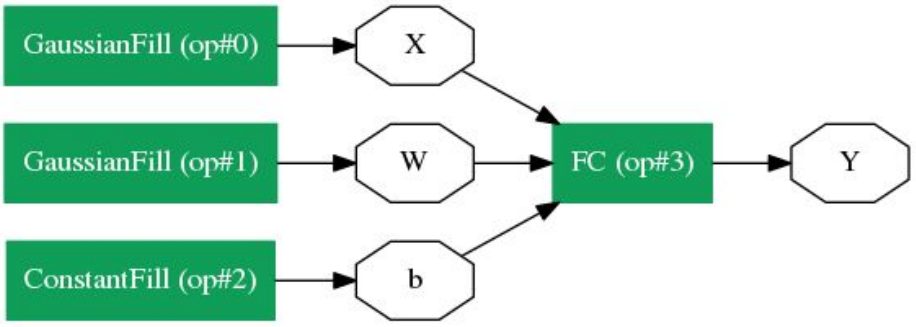
編譯測試

首先至caffe2 github clone一份程式碼直接編譯，一開始不建議使用docker，除非後來真的要進行產品化、進行包裝後才開始考慮。

使用Caffe2

Caffe2簡介

caffe2 模型基本架構就如下圖，每個運算子都必須有 input blob 與 output blob (至少選一)，預測時運算子會依序從 blob 拿取資料計算結果，計算梯度則是從輸出層一一計算梯度反推至輸入層，其中若有些不須梯度運算的運算子存在則必須手動呼叫 caffe API 來避免其計算梯度更新。

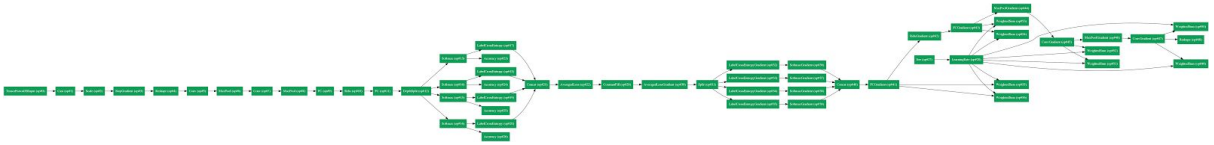


資料準備

caffe2 訓練時可以直接將資料放入輸入層的 input blob 或者預先產生 dataset檔案以產生 dbreader 。這次使用 genCaptchaCaffe.py 產生驗證碼訓練資料。

模型建立

本次模型架構圖如下圖：



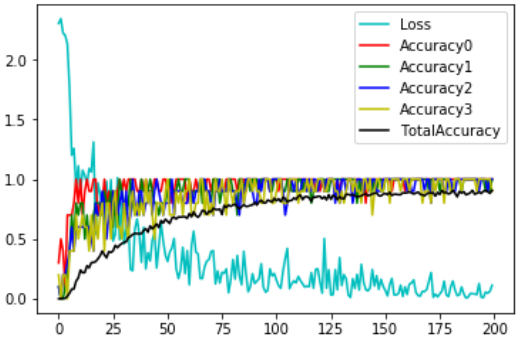
(因為模型實在太長，詳細請在資料夾內查看CaptchaReportSource/graphs/model_graph.png)

```
def AddNetModel(model, data):
    conv1 = model.Conv(data, 'conv1', dim_in=1, dim_out=16, kernel=5)
    pool1 = model.MaxPool(conv1, 'pool1', kernel=2, stride=2)
    conv2 = model.Conv(pool1, 'conv2', dim_in=16, dim_out=32, kernel=5)
    pool2 = model.MaxPool(conv2, 'pool2', kernel=2, stride=2)

    fc3 = model.FC(pool2, 'fc3', dim_in=32 * 12 * 37, dim_out=512)
    fc3 = model.Relu(fc3, fc3)
    fc4 = model.FC(fc3, 'fc4', 512, BIT_NUM * CLASS_NUM)
    fc4s = model.DepthSplit(fc4, BIT_NUM, split=CLASS_NUM, axis=1)

    softmax = []
    softmax.append(model.Softmax(fc4s[0], 'softmax0'))
    softmax.append(model.Softmax(fc4s[1], 'softmax1'))
    softmax.append(model.Softmax(fc4s[2], 'softmax2'))
    softmax.append(model.Softmax(fc4s[3], 'softmax3'))

    return softmax
```

上圖則是model的程式定義部分。

結果呈現

訓練



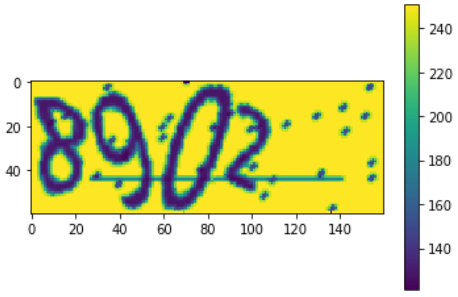
上圖為訓練過程中的各項參數變化，其中 Total Accuracy 為 TestAccuracy。訓練資料為 10000 種數字、每種 10 個樣本共十萬張圖片，訓練的 batch size 為 10 ， 20000 次迭代。

結果測試

上圖為對於最後的訓練結果進行 Test 之結果，其總體正確率約在 90% 左右，個個數據則有 95% 以上的正確率。

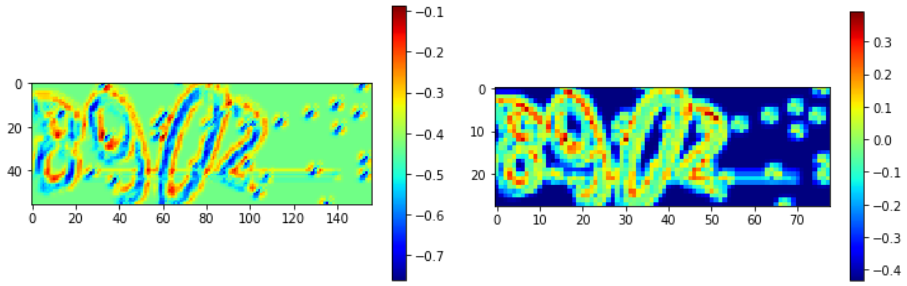
模型分析

Input:



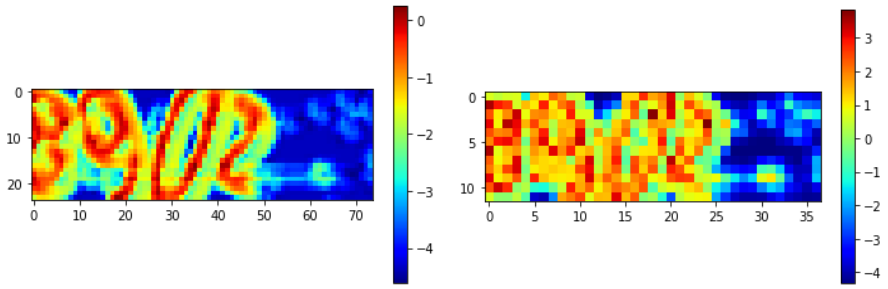
上圖為原始資料。

Conv1:



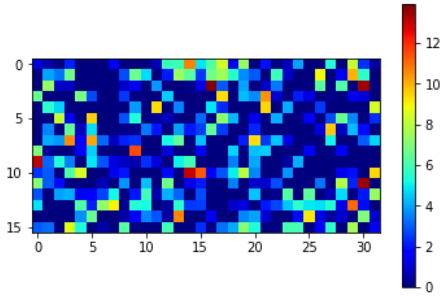
其中左圖為 conv1 的結果，右圖則是 Pooling 過後的結果。真正感興趣的部分訊號較為強烈，雖有些雜訊但其訊號並不夠強烈，可見本層除了邊緣偵測外，也兼任部分的雜音去除。

Conv2

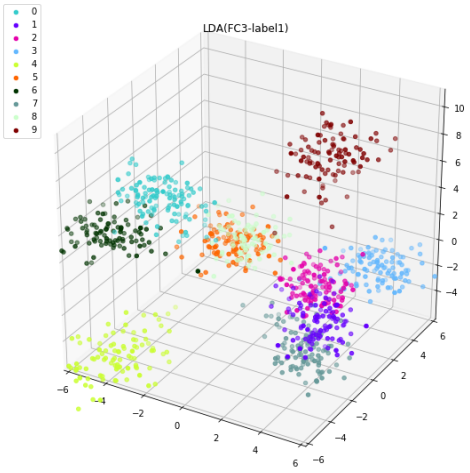


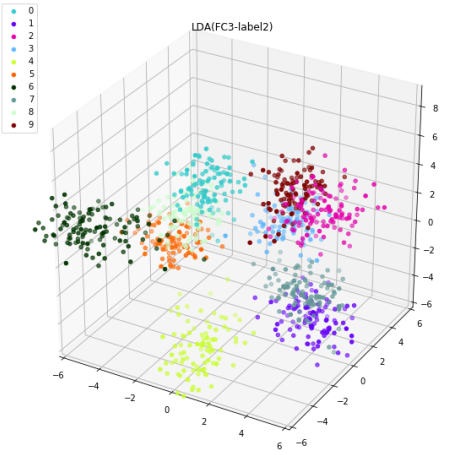
左圖同樣為 conv2 的輸出結果，右圖則是 pooling 後的結果。可以發現相較於上一層(conv1)，其數字部分的特徵被更加強調，雜訊部分則更加抑制，至此特徵萃取可說已完成。

Fc3

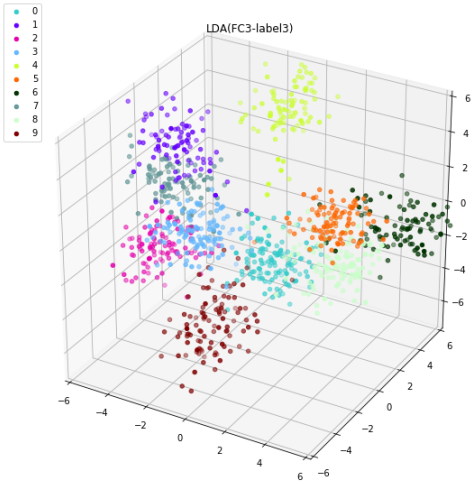


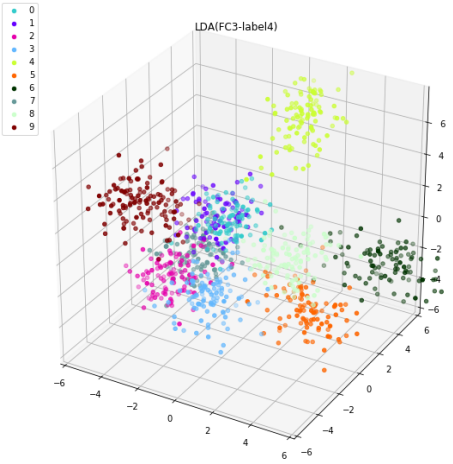
上圖為 fc3 的輸出，光看此圖實在看不出依些所以然。以下透過LDA降維，將資料置於三維空間觀察。(LDA: 與PCA同樣用於降維資料分析，但不同點在於LDA會盡量去除不相關的資訊進行降維分析，找出資料當中隱含的規則使得資料分布符合所提供的label。詳情見: LDA-線性判別分析)





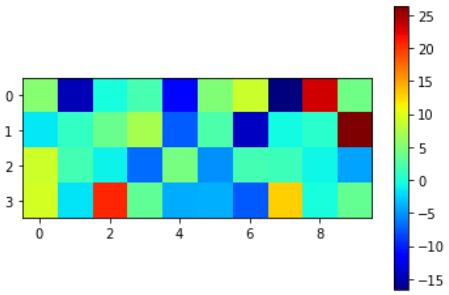
上圖兩者則是將 fc3 層的輸出 1000 個樣本，進行LDA的分析結果，可以發現在千位及百位的分類效果上兩者皆表現不錯。



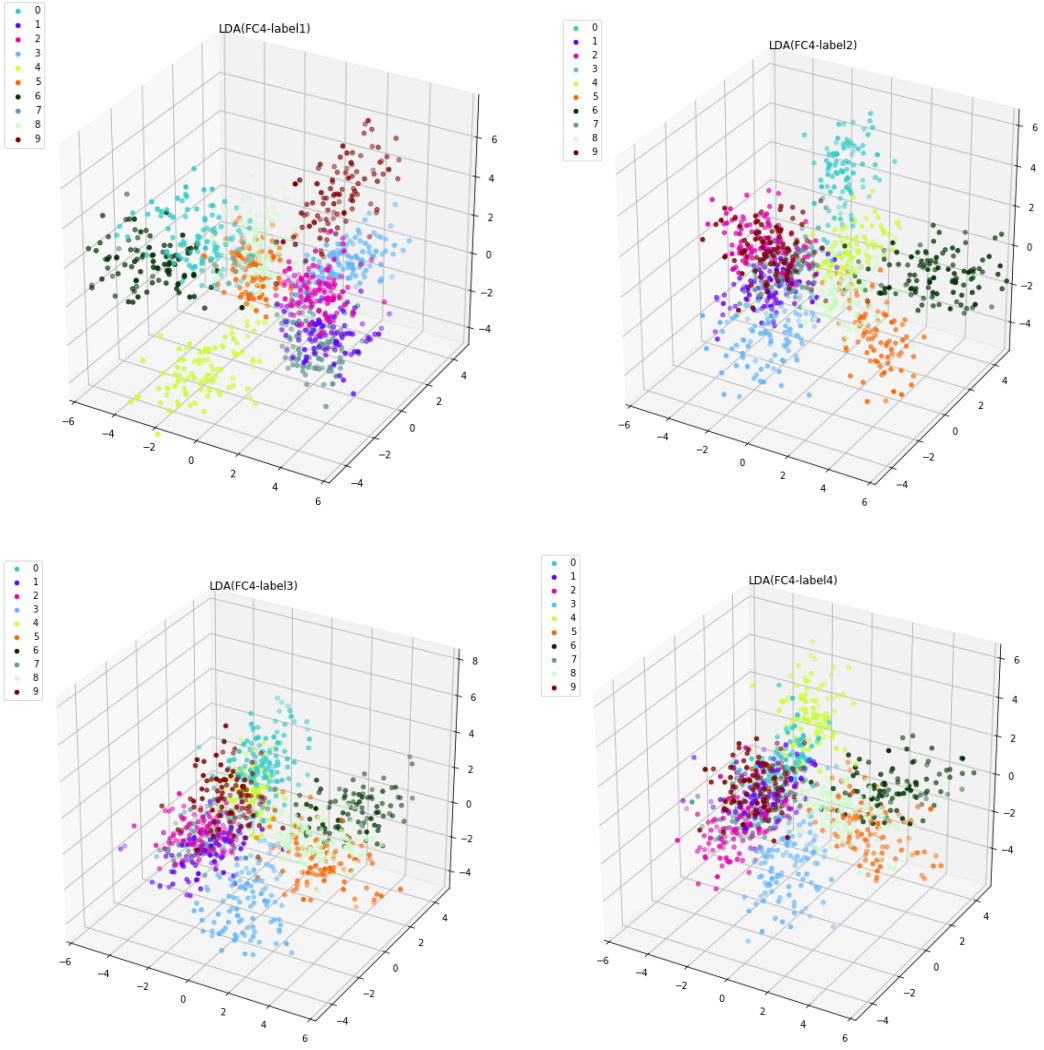


上圖為十位數與個位數的分類效果，其分類效果依位數越小而愈加難以區分，不過大致都有區分開。

Fc4



上圖為 fc4 的輸出，此層已是輸出層，可看出訊號最強的由上至下依序為 < 8 9 0 2 > 。以下透過LDA降維，將資料置於三維空間觀察。



上面由上至下由左至右依序是千百十個的分類效果，位數越小同樣有效果越差的趨勢。雖其各個群集之距離比fc3來的小，但考慮到 fc3(14208 * 512) 對於 fc4(512 * 40) 大小差距懸殊，研判因此導致可表現空間差異。雖然表現空間縮小，但其實際分類效率並未降低。

結論與改善方向

caffe2 效能的確很好，在這個本場景之下經過實驗後， caffe2 對於 Tensorflow 1.2 在GPU上有十倍左右的加速比，CPU上應該更可觀，但其文件缺乏，實現效率低，並且對於模型分析支援也較Tensorflow 少(沒有Tensorbroad)，依目前看來 caffe2 較適合實際應用時加速服務反應速度。維護性方面由於其最終將採用C++編譯，導致錯誤訊息難以理解以及較難在 compile time 得知，因此略低於 Tensorflow。彈性方面 caffe2 亦提供 python / C++ 兩種主流接口，差異不大。

本次訓練模型對於 conv1、conv2、fc3、fc4 這幾層當中共用所有權重，若要繼續提升正確率除了取得更多資料外也許可以嘗試：

1. 將 fc4 權重切割成 4 位，根據訓練難度決定神經元數量。
2. 調整 learning rate，若同個數字以相同答案答錯太多次則調升learning rate 而不是僅有調降(如 fc3、4 中 數字 4 與 5 混雜度高)。