**Deep Learning HW2**

**1 – 1**

**batch size = 64 ; epoch = 50 ; stride size = (1,1)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **filter size** | **train\_ACC** | **loss** | **val\_ACC** | **val\_loss** | **test\_ACC** |
| 2\*2 | 0.9999 | 0.0002 | 0.9913 | 0.0758 | 0.9897 |
| 3\*3 | 0.9996 | 0.0011 | 0.9922 | 0.0770 | 0.9909 |
| 4\*4 | 1.0000 | ~= 0 | 0.9937 | 0.0668 | 0.9938 |
| 5\*5 | 0.9986 | 0.0060 | 0.9940 | 0.0577 | 0.9909 |

**batch size = 64 ; epoch = 50 ; filter size = (4,4)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **stride size** | **train\_ACC** | **loss** | **val\_ACC** | **val\_loss** | **test\_ACC** |
| (1,1) | 1.0000 | ~= 0 | 0.9937 | 0.0668 | 0.9938 |
| (2,2) | 0.9972 | 0.0076 | 0.9850 | 0.0925 | 0.9820 |
| (3,3) | 0.9805 | 0.0551 | 0.9647 | 0.1249 | 0.9645 |

首先利用stride size不變與filter size改變來測出4\*4是test loss最好的，接著再改變stride size發現(1,1)是最好的，並利用這些參數畫出下列圖表。

**batch size = 64 ; epoch = 50 ; filter size = 4\*4 ; stride size = (1,1)**

一張含有 文字, 圖表, 螢幕擷取畫面, 行 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 圖表, 陳列 的圖片

自動產生的描述

一張含有 文字, 螢幕擷取畫面, 繪圖, 圖表 的圖片

自動產生的描述 一張含有 螢幕擷取畫面, 文字, 圖表, 繪圖 的圖片

自動產生的描述

一張含有 文字, 圖表, 螢幕擷取畫面, 繪圖 的圖片

自動產生的描述 一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述 一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述

在1-1中我有先將圖片每個pixel除255，使其可以藉於0 ~ 1之間，方便模型訓練，接著利用類似LeNet-5的架構訓練，架構為:

1. Input 7. Flattern
2. Conv2D: 6 Filters 8. Dense: 84 neurons
3. MaxPooling: 採樣區域2\*2 9. Dense (output): 10 neurons
4. Conv2D: 16 Filters
5. MaxPooling: 採樣區域2\*2
6. Conv2D: 120 Filters

1 – 2

一張含有 文字, 字型, 螢幕擷取畫面, 白色 的圖片

自動產生的描述

一張含有 文字, 字型, 螢幕擷取畫面, 白色 的圖片

自動產生的描述

上面兩類圖皆利用預測為5來當作範例，可發現model的判斷依據為最上面有一橫且下面有個向右突出的圓弧，而上列所有圖都符合這些特徵，因此被分類成5。(但我在網路上看到有人分析似乎這些特徵抓取只是我們人類的想像，機器其實並沒有這些概念，有點因為結果如此所以我們事後諸葛的感覺。)

1 - 3

**一張含有 文字, 螢幕擷取畫面, 字型, 圖形 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 行, 字型, 圖形 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 正方形, 樣式, Rectangle 的圖片

自動產生的描述**

這三張圖是以數字5當作input而得到的結果，在第一層的時候有描繪出5的形狀，第二層則是開始找出更細節的部分，像是5的上面一橫與下面一個圓弧，如:

一張含有 鮮豔, 螢幕擷取畫面, Rectangle, 正方形 的圖片

自動產生的描述

但也有很多看不出來關聯性的，如:

一張含有 螢幕擷取畫面, 鮮豔, 正方形, Rectangle 的圖片

自動產生的描述

而到第三層則開始找更微小、碎片的特徵，也更加難以分析，但正如1-2所述，也許這些分析只是我們的想像而已。

**1 – 4**

**batch size = 64 ; epoch = 50 ; filter size = 4\*4 ;**

**stride size = (1,1) ; l2\_lambda = 0.001**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **train\_ACC** | **loss** | **val\_ACC** | **val\_loss** | **test\_ACC** |
| 0.9903 | 0.0887 | 0.9912 | 0.0874 | 0.9902 |

**一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 圖表, 行 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 圖表, 繪圖 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 圖表, 行 的圖片

自動產生的描述一張含有 文字, 螢幕擷取畫面, 行, 圖表 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 行, 圖表 的圖片

自動產生的描述**

加了L2 regularization最初的用意是為了使model不要overfitting，導致model預測變差，但應用在MNIST dataset時，推測由於training set與testing set的分布應該都長得差不多，所以沒有overfitting反而使準確度略為下降，且因為L2 regularization的做法為限制參數的大小，所以可從參數分布圖中發現參數大多集中為0，另外，由於皆為0可能代表其實該層沒什麼作用，因此我嘗試將conv 3與dense 1給去掉並再訓練一次模型，可發現其實最後效果差不多:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **train\_ACC** | **loss** | **val\_ACC** | **val\_loss** | **test\_ACC** |
| 0.9898 | 0.0848 | 0.9902 | 0.0854 | 0.9891 |

因此加上L2 regularization可能的副作用為減少模型的參數，加速訓練模型。(但這種情況為case by case，不見得永遠適用，只是一種可能性。)