**Deep Learning HW2**

**1 – 1**

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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) ; padding = same**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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) ; padding = same**

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 ; padding = same**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **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。)

**2 – 1**

**batch size = 64 ; epoch = 10 ; stride size = (1,1) ; padding = same**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **filter size** | **train\_ACC** | **loss** | **val\_ACC** | **val\_loss** | **test\_ACC** |
| 2\*2 | 0.8226 | 0.5086 | 0.6100 | 1.3217 | 0.5985 |
| 3\*3 | 0.8821 | 0.3400 | 0.7102 | 0.9119 | 0.7026 |
| 4\*4 | 0.8995 | 0.2860 | 0.7056 | 1.0101 | 0.7034 |
| 5\*5 | 0.9190 | 0.2316 | 0.7276 | 1.0213 | 0.7182 |

**batch size = 64 ; epoch = 10 ; filter size = (5,5) ; padding = same**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **stride size** | **train\_ACC** | **loss** | **val\_ACC** | **val\_loss** | **test\_ACC** |
| (1,1) | 0.9190 | 0.2316 | 0.7276 | 1.0213 | 0.7182 |
| (2,2) | 0.6822 | 0.8930 | 0.5908 | 1.2046 | 0.5782 |
| (3,3) | 0.6239 | 1.0488 | 0.5470 | 1.2931 | 0.5362 |

**batch size = 64 ; epoch = 30 ; filter size = 5\*5 ; stride size = (1,1) ; padding = same**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **train\_ACC** | **loss** | **val\_ACC** | **val\_loss** | **test\_ACC** |
| 0.9825 | 0.0502 | 0.8078 | 1.0602 | 0.7905 |

**一張含有 文字, 圖表, 繪圖, 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, 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 圖表, 螢幕擷取畫面, 繪圖, 文字 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, 圖表, 繪圖, 螢幕擷取畫面 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, 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, 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**

**一張含有 螢幕擷取畫面, 圖表, 文字, 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, 圖表, 繪圖, 螢幕擷取畫面 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**

由於CIFAR-10的圖片比MNIST還要複雜，因此我嘗試實作簡易版的ResNet，其主要突破為residual block，架構如下:

一張含有 文字, 字型, 圖表, 行 的圖片

自動產生的描述

在經過residual block且使用shortcut (即把輸入加上輸出)時，因為最後輸出還會加上原本的輸入，不會因為網路太深而gradient vanishing，使其就算甚麼都沒學到，但至少還能保持原本輸入的樣貌，詳細實作架構為:

1. Input
2. Conv2D: 16 Filters
3. Residual\_Block: 16 Filters
4. Residual\_Block: 16 Filters
5. Residual\_Block: 32 Filters with shortcut
6. Residual\_Block: 32 Filters
7. Residual\_Block: 64 Filters with shortcut
8. Residual\_Block: 64 Filters
9. Global Average Pooling
10. Dense (output): Softmax

在一開始有先對input做normalization，層與層之間有做batch normalization。

**2 – 2**

**一張含有 螢幕擷取畫面, 文字 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 文字 的圖片

自動產生的描述**

在圖片分析上，我皆以預測狗的圖片當作範例，但說實在從圖片上來觀察實在很難判斷model是依據甚麼來判斷是否為狗，勉強猜測判斷依據可能為有腳且耳朵比較明顯，因此有腳且有明顯同一色塊在臉的兩側就有可能被判斷為狗，如:

一張含有 螢幕擷取畫面, 模糊的, 正方形, 藝術 的圖片

自動產生的描述

鹿角可能被判斷成耳朵再被判斷成狗，然而我實在看不太出來關聯性。

**2 – 3**

**一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述**

**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 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 字型, 行 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, 字型, 設計 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 綠色, Rectangle, 圖形 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 綠色, Rectangle, 圖形 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, Rectangle, 綠色, 圖形 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 圖形, Rectangle, 字型 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 圖形, 平面設計, 字型 的圖片

自動產生的描述**

**一張含有 螢幕擷取畫面, 樣式, 正方形, Rectangle 的圖片

自動產生的描述**

**一張含有 樣式, 正方形, 螢幕擷取畫面, Rectangle 的圖片

自動產生的描述**

**一張含有 樣式, 螢幕擷取畫面, 正方形, Rectangle 的圖片

自動產生的描述**

**一張含有 樣式, 螢幕擷取畫面, 正方形, Rectangle 的圖片

自動產生的描述**

**一張含有 樣式, 正方形, Rectangle, 螢幕擷取畫面 的圖片

自動產生的描述**

這些feature map的輸入為狗的圖片，原圖為:

一張含有 哺乳動物, 白色, 狗 的圖片

自動產生的描述

在前三層可看到feature map逐漸把狗的輪廓給抓出來，越後面的圖可能為更細部的特徵，但難以判斷確切為何。

**2 – 4**

**batch size = 64 ; epoch = 30 ; filter size = 5\*5 ; stride size = (1,1)**

**padding = same ; l2\_lambda = 0.001**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **train\_ACC** | **loss** | **val\_ACC** | **val\_loss** | **test\_ACC** |
| 0.9040 | 0.6095 | 0.7600 | 1.1350 | 0.7480 |

一張含有 文字, 圖表, 繪圖, 行 的圖片

自動產生的描述一張含有 文字, 圖表, 行, 繪圖 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, 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 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, 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, 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

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, 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, 行 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

如同第一部分，在加上L2 regularization後，各層的參數皆有往0靠攏，並試著減少overfitting，且由圖表可知，其實在18 epoch時，test accuracy已經相當接近0.8，但其表現隨著epoch成長而產生震盪，導致最後結果不盡理想，也許應該設立一個門檻，當accuracy超越門檻時就停下來(但這樣對門檻的設置又是一個大難題)，另外仍嘗試去觀察feature map，發現其實很多map全部都是0 ，可能導致某些特徵無法被判斷出來，也或許這是加了L2 regularization的副作用。

一張含有 正方形, 螢幕擷取畫面, 樣式, 鮮豔 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, 鮮豔 的圖片

自動產生的描述

**2 – 5**

我在一開始有先將每個pixel做normalization，使其比較好學習，且在層與層之間再加上batch normalization，使其不會因為某些特徵而帶偏整個模型。