

中華民國第 64 屆中小學科學展覽會

作品說明書

高級中等學校組 工程學(一)科

探究精神獎

052310

微控制器智慧之旅－8051 與深度學習

學校名稱： 國立彰化高級中學

作者： 高一 吳品諒	指導老師： 邱科文
-------------------	------------------

關鍵詞： 深度學習、神經網路、微處理機應用

摘要

本研究旨在將 AI 深度學習應用於 8051 開發板的數字辨識，希望在低功耗的情況下進行 AI 實作來達到環境保育的目標，探討在硬體受限的情況下如何操縱硬體並使用 Python 進行參數訓練。通過對不同層數和學習速率的實驗，我們發現兩層層數和 0.01 的學習速率能取得較好效果。在模型訓練中，我們運用了 Cross-entropy 計算 Loss 值，並探討了不同 Learning rate 對 Loss 值的影響，以及不同 Layers 對 Loss 值的影響。此外，我們還討論了深度學習與機器學習的差異，並介紹了 8051 開發板的基本原理和應用。未來，我們將繼續改進模型效能和準確性，並優化硬體設計和性能，以應用深度學習技術於更廣泛的領域。

壹、研究動機

AI 深度學習在現代是一個熱門研究領域，但是隨著 AI 的功能越來越強大，計算量與耗電量也成為備受關注的議題。並不是每一個 AI 都需要非常大的電力或者能源才能進行高效率的運算，因此本次實作想藉由控制 8051 微處理器來達到環保與節省能源的目標，並達成聯合國永續發展目標(Sustainable Development Goals, SDGs)其中幾項目標。由於 AI 可以被應用到其他領域而有革命性的應用，而大多數都需要高階硬體與高速運算來進行。我高中開始對資訊有相當濃厚的興趣，參加國家高速網路與計算中心與大學合辦的 AI 營隊後，便想針對 AI 深度學習進行學習及研究。因此，我藉由自學的 Python 來進行參數訓練存至 VScode 裡的檔案夾中，並結合主程式(main)燒錄到 8051 微控制器的微控制器，讓此具 AI 深度學習的 8051 微控制器微控制器在低耗能的單機作業下進行數字輸入的辨識，並探討不同 Learning rate 對 Loss 值的影響以及不同 Layers、訓練次數對 Loss 值的影響，希望在 8 位元的 8051 微控制器上達到最佳及最快速的數字辨識。

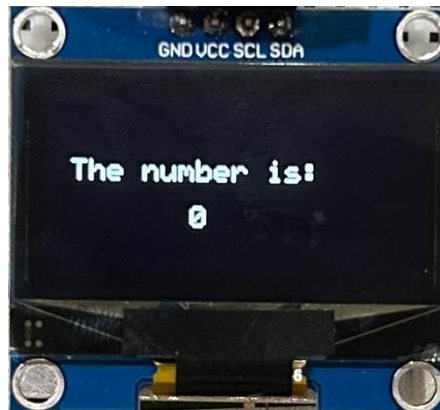
貳、研究目的

- 一、運用 Python 訓練深度學習模型。
- 二、探討不同超參數（Learning rate, Layers, Epoch）對模型的影響。
- 三、測試手寫辨識在 8051 微控制器上的辨識結果。
- 四、量測 8051 為處理器在運算時的耗電量。

參、研究器材與軟體

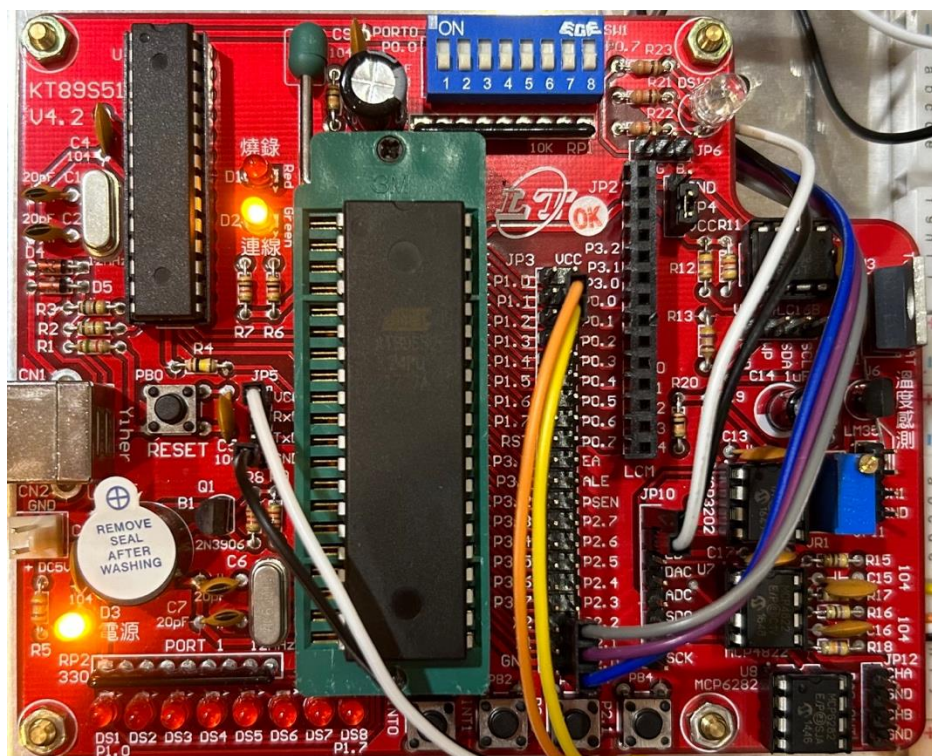
一、 硬體:

1. OLED 板



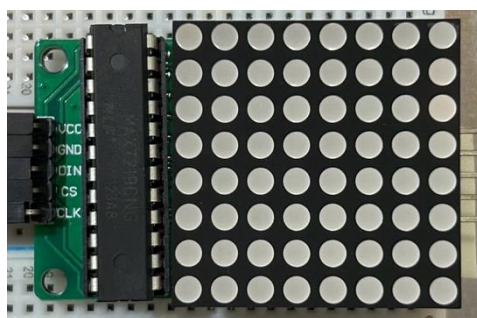
(自行拍攝)

2. 8051 微控制器



(自行拍攝)

3. 矩陣顯示器



(自行拍攝)

4. 筆記型電腦

MacBook Air(M2 晶片),Microsoft laptop4

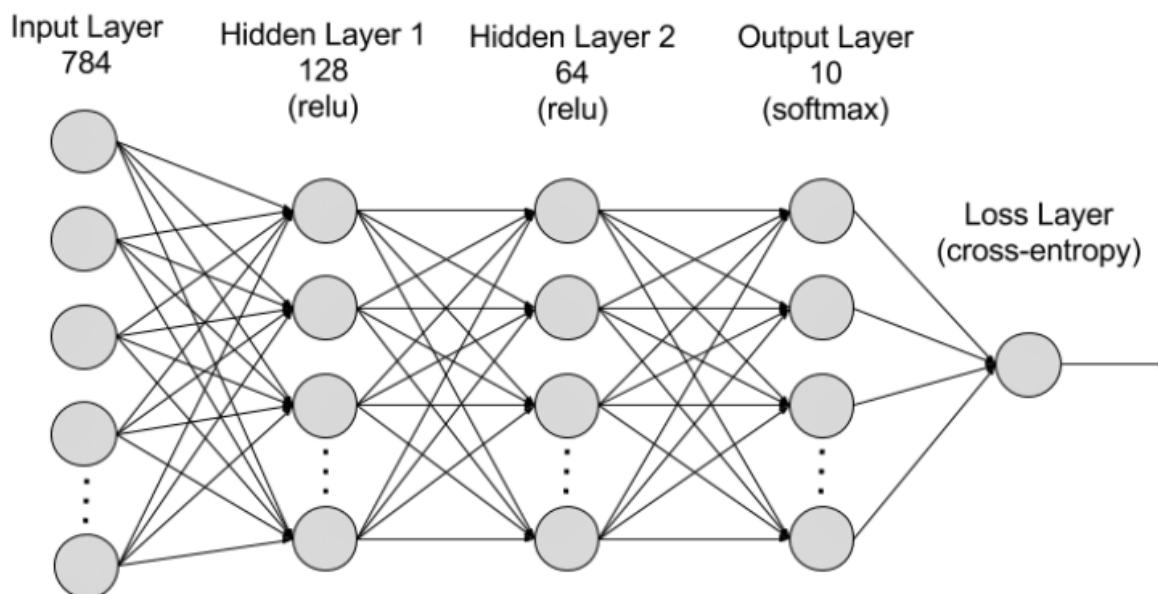
二、 軟體：

1. Visual Studio Code
2. Python 3.10.2 版本、Google Colab、Python torch.nn 以及其他函式庫

肆、 原理介紹

一、 深度學習介紹：

深度學習是一種機器學習的方法，它模仿人腦神經元之間的連接方式，通過多層次的神經網絡來學習和理解數據。這種學習方式使得機器能夠自動地從數據中學習到複雜的特徵和模式，並做出對應的預測或決策。深度學習的核心是神經網絡，它由多層次（Layer）的神經元組成，每一層都將上一層的輸出作為輸入，如圖(一)。每個神經元與上一層的所有神經元都有連接，並且帶有權重(weight)，這些權重可以通過訓練過程來調整，以最大程度地擬合訓練數據。深度學習通常需要大量的數據來訓練模型，以及強大的計算資源來進行模型訓練和推理。它在許多領域都取得了巨大的成功，包括圖像識別、語音識別、自然語言處理等。深度學習的一些常見模型包括卷積神經網絡（CNN）、循環神經網絡（RNN）和遞迴神經網絡（Recursive Neural Networks，RNNs）等，每種模型都有其特定的應用場景和優缺點。深度學習作為一種強大的機器學習方法，為解決複雜的問題提供了有效的工具，並在人工智慧領域中發揮著重要作用。



圖(一)神經網路示意圖

(參考來源：來自 amazon 官網)

二、 8051 微控制器介紹

8051 微控制器是一種基於 8051 單片機的開發平台，用於學習、開發和測試嵌入式系統。8051 單片機是一款經典的 8 位單片機，廣泛應用於嵌入式系統中，具有低功耗、高穩定性和豐富的外設接口等特點。

8051 開發板通常包含了一個 8051 單片機芯片，以及與之相應的外設，如 LED 顯示器、按鈕、蜂鳴器、數字顯示器、通訊接口等，可以方便地進行各種實驗和應用開發。微控制器上通常還提供了擴展接口，可以連接其他外設或模塊，擴展其功能。

使用 8051 微控制器可以進行各種嵌入式系統的開發，如智能家居控制系統、溫度監測系統、車輛控制系統等。開發過程中，可以使用 C 語言或組合語言等編程語言進行開發，並通過燒錄器將程序燒錄到 8051 單片機中運行。

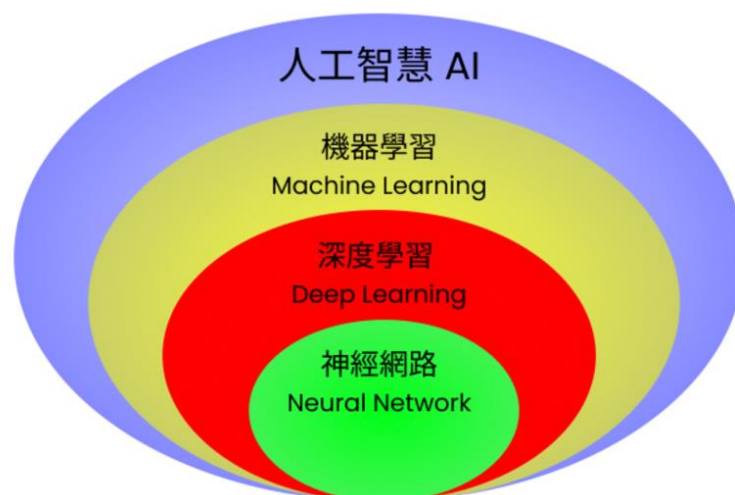
而我之所以選用 8051 微控制器是由於它的成本相較低廉，且對我來說操作也相較其他微控制器要來得簡單，因此選擇利用 8051 微控制器進行實作。

三、 深度學習(Deep learning) 與 機器學習(machine learning)的差別

Deep learning 與 machine learning 是兩個相關但又不完全相同的概念。機器學習是一個更廣泛的領域，它涵蓋了多種方法和技術，用於使計算機系統通過學習從數據中提取模式和知識，從而完成特定任務，而不需要明確地編程。機器學習可以分為監督學習、非監督學習和強化學習等不同的類型，包括了多種算法和技術，如支持向量機（Support Vector Machines）、決策樹（Decision Trees）、K-最近鄰（K-Nearest Neighbors）等。

而深度學習是機器學習的一個分支,如圖(二)，它主要基於人工神經網絡（Artificial Neural Networks）來實現。深度學習模型通常包含多個隱藏層，並且每一層都包含大量的神經元，這些模型能夠從大量數據中學習到更高層次的抽象特徵。深度學習在處理大量數據和複雜任務時表現出色，尤其在圖像識別、語音識別、自然語言處理等方面取得了巨大的成功。

總而言之，深度學習是機器學習的一個重要分支，它通過多層次的神經網絡模型來實現對複雜問題的建模和學習，是機器學習中的一個強大而有效的技術手段。



圖（二）顯示 AI 與深度學習關係

（參考來源：來自 glints 官網）

四、 Loss 值介紹：

在深度學習中，「Loss 值」是一個重要的概念，用於衡量模型預測與真實值之間的差異。Loss 函數通常是一個數值，表示模型在訓練過程中的性能表現。Loss 值越小，代表模型的預測結果與真實值越接近。

伍、研究過程以及方法

由於 8051 微控制器的記憶體大小僅 128bytes,但一個 int(整數) 就需要記憶體空間 4bytes,最多也只能存 32 個 int,因此我的解決方案就是把 8 個數字存到一個 char(字元)內部,如圖(三),一共有 8 個字元陣列,因為一個 char(字元)只佔記憶體位置 1 個 byte,然而我們的目的是要將所以值是[1]的權重(weight)加起來,所以要讀取時需要圖(四)程式碼的輔助。程式碼如(附錄 1.)

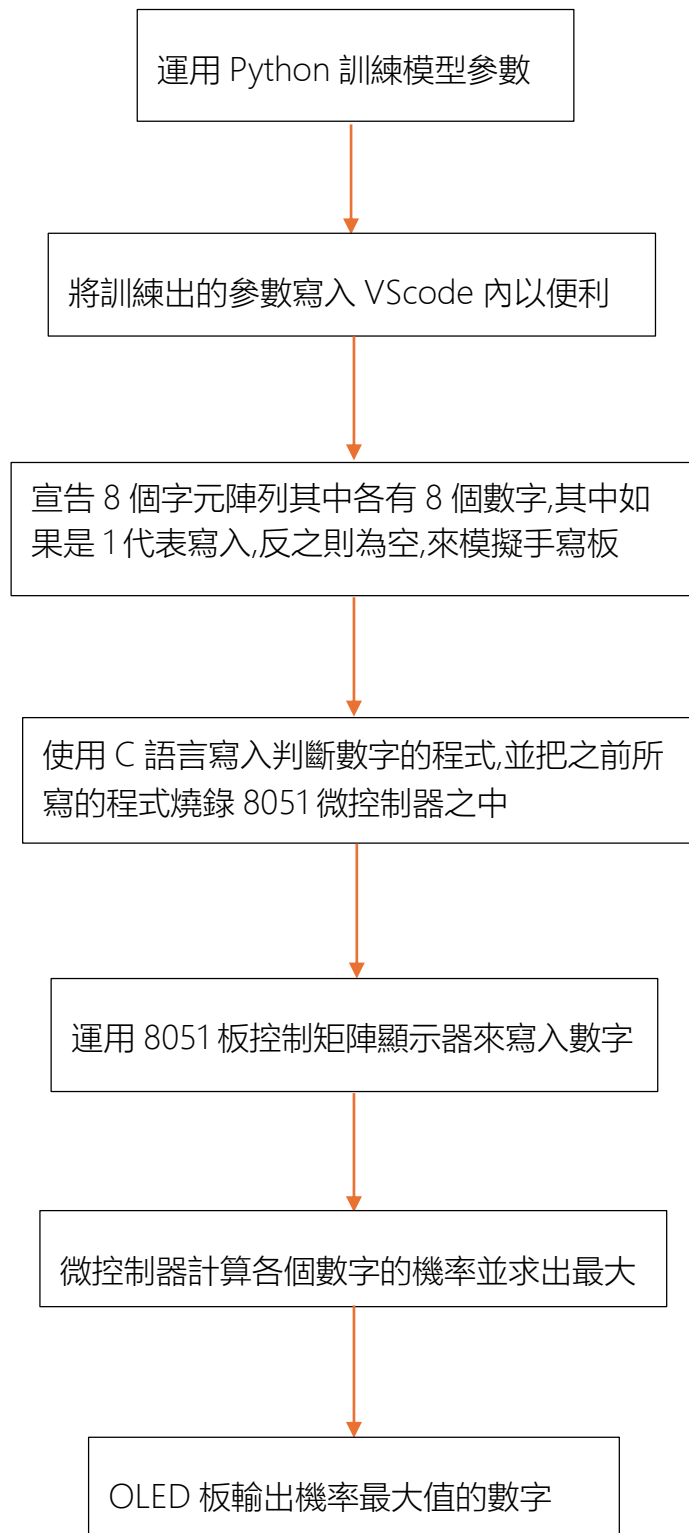
```
char word[8] = {
    0b00000001,//1
    0b00100000,//32
    0b00010000,//16
    0b10000000,//128
    0b00001001,//9
    0b00100001,//33
    0b10010000,//144
    0b00100110//38
};
```

圖(三)運用 8 個字元陣列存入 8 個數字模擬手寫板 (參考來源：自行拍攝)

```
},
float sum = 0;
char c = 0b00000001;
for(int i=0; i<8;i++){
    for(int j = 0;j<8;j++){
        if((word[i] & c) == 0){
            continue;
        }
        else{
            sum += HIDDEN_LAYER_weight[8 * i + j];
            c = c << 1;
        }
    }
}
```

圖(四)運用 C 語言讀取寫入的數字 (自行拍攝)

一、研究流程



(自行繪製)

二、如何訓練深度學習的模型(model)

1. Loss 值運算

藉由圖（五）,圖（六）,其中 $\hat{y}$ 是理想中應該要輸出的值， y 是指經由神經網路(Network)所得出的結果，而 y' 則是 y 通過 SoftMax 函數所得的值（如圖（八）所示），SoftMax 函數運算過程示意圖如圖（七），圖（九）所示。

認為 Loss 值的運算選用 Cross-entropy 會比 Mean Square Error 有效率(因為較多人使用，導致許多重要函式庫都與 Cross-entropy 綁在一起)，因此接下來的操作都會運用 Cross-entropy。

$$\text{Cross-entropy} \quad e = - \sum_i \hat{y}_i \ln y'_i$$

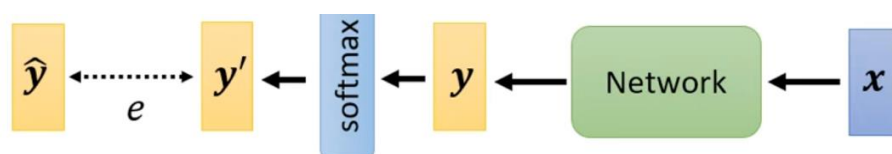
圖（五）介紹 Cross-entropy

（參考資料：台灣大學李弘毅教授上課投影片）

$$\text{Mean Square Error (MSE)} \quad e = \sum_i (\hat{y}_i - y'_i)^2$$

圖（六）介紹 Mean Square Error

（參考資料：台灣大學李鴻毅教授上課投影片）



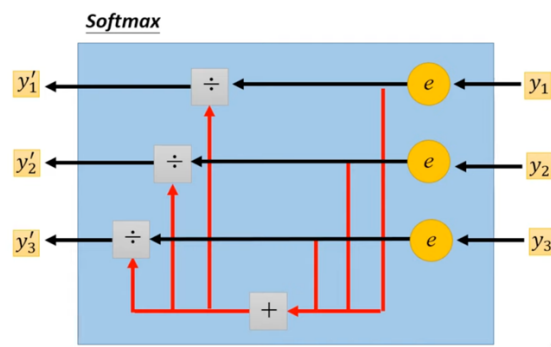
圖（七）產生 Loss 值的流程圖

（參考資料：台灣大學李弘毅教授上課投影片）

$$y'_i = \frac{\exp(y_i)}{\sum_j \exp(y_j)}$$

圖（八）SoftMax 函數方程式

（參考資料：台灣大學李弘毅教授上課投影片）



圖（九）介紹 SoftMax 運作的方式

（參考資料：台灣大學李弘毅教授上課投影片）

2. 訓練(train)深度學習的模型(model)

如圖（十）與圖（十一），參考加州大學 Dwiti Patel 的 Verilog Implementation of a Digit Classifier Using Deep Learning Primitives 論文。在下載完 mnist 手寫之後，如圖（十），便開始一系列的訓練。先宣告重要的超參數 Learning rate 等之後進入迴圈：對每個 batch 的輸入圖像進行 resize，將大小從 28x28（mnist）調整為 8x8，然後將圖像扁平化為一維 tensor。然後將輸入數據傳遞給模型進行前向傳播，計算 Loss 值（方法如上（1.Loss 值運算）所示所示），並進行反向傳播和權重（weight）更新。

對驗證數據進行迴圈：對每個 batch 的驗證數據進行相同的 resize 和扁平化操作，然後將輸入數據傳遞給模型進行前向傳播，計算 Loss 值，細節請參下列 Python 程式碼。

```
from torchvision.datasets import MNIST # Load MNIST Dataset and apply transformation
mnist_dataset = MNIST("data", download=True, train=True, transform= transform_funcs)
```

圖（十）引用 MNIST 數據集來訓練模型（自行拍攝）

```

import matplotlib.pyplot as plt
import cv2
import numpy as np
NUM_EPOCHS = 200
resized_image = []

# 宣告超參數
learning_rate = 0.01
weight_decay = 1e-6

from torch import optim
cross_entropy_loss_function = torch.nn.CrossEntropyLoss()
model_optimizer = optim.Adam(digit_classifier_model.parameters(),
                              lr=learning_rate, weight_decay=weight_decay)

for epoch in range(1, NUM_EPOCHS):
    training_loss = []
    validation_loss = [] ## 訓練迴圈
    for input_data, target_prediction in training_loader:
        rimages = []
        for i in range(input_data.shape[0]):# 將圖像從 28x28 調整為 8x8
            image = input_data[i][0]
            rimage = cv2.resize(image.numpy(), dsize=(8, 8), interpolation=cv2.INTER_CUBIC)
            rimage = rimage.reshape(1, rimage.shape[0], rimage.shape[1])
            rimages.append(rimage)

        resized_image_batch = np.stack(rimages, axis=0)
        resize_tensor = torch.tensor(resized_image_batch)

        input_data = resize_tensor.view(-1, 64)
        model_optimizer.zero_grad()# 前向傳播
        model_output = digit_classifier_model(input_data)# 損失計算
        model_loss = cross_entropy_loss_function(model_output, target_prediction)# 反向傳播
        model_loss.backward()# 權重優化
        model_optimizer.step()
        training_loss.append(model_loss.item())## 準備模型進行評估
    digit_classifier_model.eval()

    # 評估迴圈
    for input_data, target_prediction in validation_loader:
        rimages = []
        for i in range(input_data.shape[0]):
            image = input_data[i][0]
            rimage = cv2.resize(image.numpy(), dsize=(8, 8), interpolation=cv2.INTER_CUBIC)
            rimage = rimage.reshape(1, rimage.shape[0], rimage.shape[1])
            rimages.append(rimage)

        resized_image_batch = np.stack(rimages, axis=0)
        resize_tensor = torch.tensor(resized_image_batch)

        input_data = resize_tensor.view(-1, 64)
        model_output = digit_classifier_model(input_data)
        model_loss = cross_entropy_loss_function(model_output, target_prediction)
        validation_loss.append(model_loss.item())

    print(np.mean(training_loss))

```

圖（十一）運用 Python 訓練模型（自行拍攝）

3. 訓練完成的模型參數應用

將 Python 訓練出來的實驗參數,藉由圖(十二)的程式碼把訓練好的 hidden_layer, hidden_bias, output_weight, output_bias 儲存至電腦檔案夾裡。所示、所得的結果如圖(十三)呈現。

```
import torch
import onnx
x = torch.randn(1, 64)
torch_out = digit_classifier_model(x)
torch.onnx.export(digit_classifier_model, x, "digit_classifier.onnx", export_params=True, opset_version=10, input_names = ['input'], output_names = ['output'])

onnx_model = onnx.load("digit_classifier.onnx")
onnx.checker.check_model(onnx_model)

import onnxruntime
ort_session = onnxruntime.InferenceSession("digit_classifier.onnx")

def to_numpy(tensor):
    return tensor.detach().cpu().numpy() if tensor.requires_grad else tensor.cpu().numpy()

ort_inputs = {ort_session.get_inputs()[0].name: to_numpy(x)}
ort_outs = ort_session.run(None, ort_inputs)

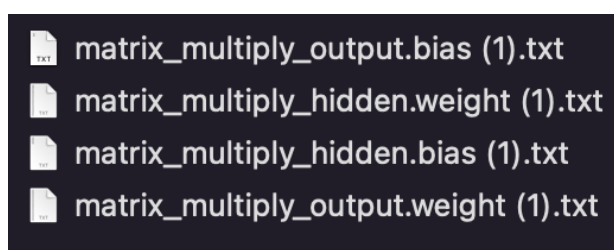
np.testing.assert_allclose(to_numpy(torch_out), ort_outs[0], rtol=1e-03, atol=1e-05)

import numpy as np
for param_tensor in digit_classifier_model.state_dict():
    print(param_tensor, "\t", digit_classifier_model.state_dict()[param_tensor].size())
    values = list(digit_classifier_model.state_dict()[param_tensor].numpy().flatten())
    filename = param_tensor + '.txt'
    txt_file = open(filename, 'w')
    for v in values:
        txt_file.write([str(v)+'\n'])
    txt_file.close()

matrix_multiply_hidden1.weight torch.Size([18, 64])
matrix_multiply_hidden1.bias torch.Size([18])
matrix_multiply_hidden2.weight torch.Size([18, 18])
matrix_multiply_hidden2.bias torch.Size([18])
matrix_multiply_output.weight torch.Size([10, 18])
matrix_multiply_output.bias torch.Size([10])

from google.colab import files
files.download('matrix_multiply_hidden.weight.txt')
files.download('matrix_multiply_hidden.bias.txt')
files.download('matrix_multiply_output.weight.txt')
files.download('matrix_multiply_output.bias.txt')
```

圖(十二) 將訓練好的參數(模型)下載到電腦的程式碼(自行拍攝)



圖(十三) 下載到電腦的文字檔(.txt)(自行拍攝)

最後再將所得的參數存成圖（十四）的形式（陣列）。若要在主程式程式裡面用的話需要圖（十四）與圖（十五）紅色圓圈的程式碼。

```
#ifndef OUTPUT_LAYER_H
#define OUTPUT_LAYER_H

short int output_layer_bias[] = {
-20,24,-26,7,13,-6,4,-32,4,7,-5,-4,-16,34,-1,-7,17,9
};

short int output_layer_weight[] = {
-87,-24,-73,-139,101,-1,-125,-99,38,0,-3,0,-54,-21,-29,90,
150,134,7,202,-114,-99,-140,104,100,123,-25,-65,-252,39,-103,
130,-93,-11,24,129,9,-136,51,-80,94,-90,42,0,-121,-106,17,62,
135,115,0,75,17,117,124,-106,111,-20,122,78,-30,-55,-68,-39,
-2,69,60,5,-54,-22,-78,-141,-99,71,91,-71,-184,-10,109,4,105,
-70,16,113,100,9,-52,-178,-26,-16,122,-39,7,-97,116,74,-138,
-43,58,175,44,-143,-81,-36,97,103,-102,-84,13,-84,-77,-5,-254,
-78,-106,48,55,154,22,-96,-116,44,163,55,40,155,-158,116,67,81,
16,11,47,-189,134,19,4,25,-26,-47,57,42,26,-36,34,-74,-40,49,91,
17,-17,65,-44,-13,33,7,48,-136,17,-50,54,-136,7,51,38,35,144,39,
88,-12,-38,-71,108,16,4,-61,-69,-89,-69
};

#endif
```

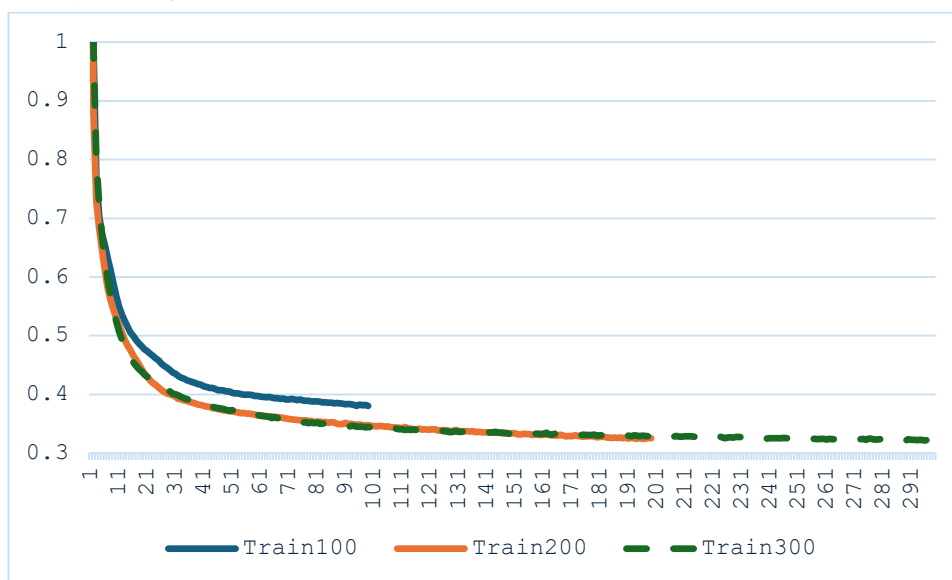
圖（十四）呈現儲存的情形（自行拍攝）

```
#include "hidden_layer.h"
#include "output_layer.h"
#include <stdio.h>
short int ReLu(short int number);
int main(){
    unsigned char word[]={
0b00000000,
0b00111110,
0b0010010,
0b0000100,
0b0000100,
0b0000100,
0b0000100,
0b0000000,
0b0000000
    };
    short int hidden_node[18]= { }; // declaration of the node of
    int output_node[10]= { }; //declaration of the node of the output
    for(int i=0; i<18; i++){ // calculating 18 node separately
        for(int k=0; k<8; k++){
            int temp=0b10000000; // this temp number is used for filtering
            for(int j=0; j<8; j++){
                if(temp & word[k]){ // if word[k][j] is one -> then
                    hidden_node[i]+=hidden_layer_weight[64*i+8*k+j];
                }
                temp=temp>>1; // right shift to filter next number
            }
        }
    }
}
```

圖（十五）展示部分主程式（main）（自行拍攝）

三、訓練次數多寡的差異

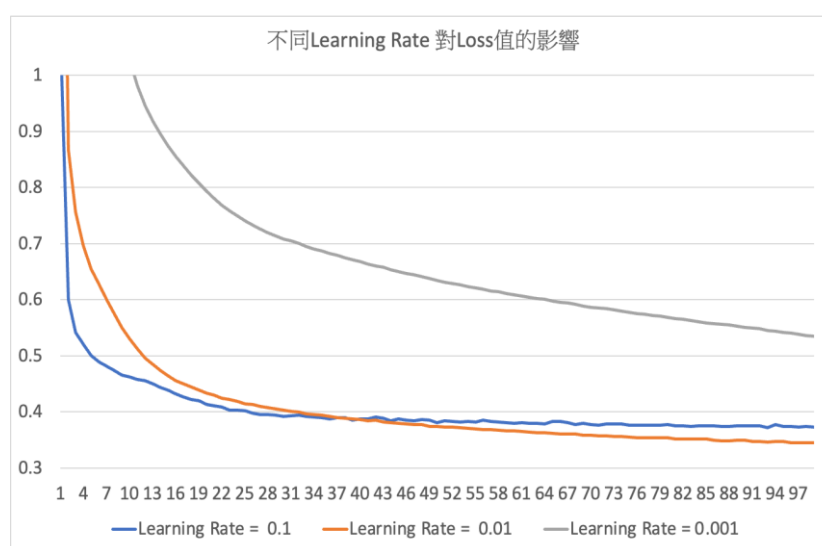
藉由上述測試 Loss 值的方法量測訓練 100 次,200 次,300 次個別的 Loss 值，如圖（十三），發現訓練 100 次和訓練 200 次與訓練 300 次的 Loss 值有較大的差距，然而訓練 200 次與訓練 300 次的結果幾乎沒有任何差別，因此選擇使用訓練 200 次的模型（參數）。



圖（十六）3 種不同訓練(Training)次數所對應的 Loss 值（自行拍攝）

四、不同 learning rate(學習速率) 對於 lose 值的影響

1. 測試 learning rate = 0.01,訓練 199 次與 learning rate = 0.1,訓練 199 次 Loss 值的關係，如表一。



表一 呈現在 Epoch 固定之下兩種學習速率對應到 Lose 值的差異關係（自行拍攝）

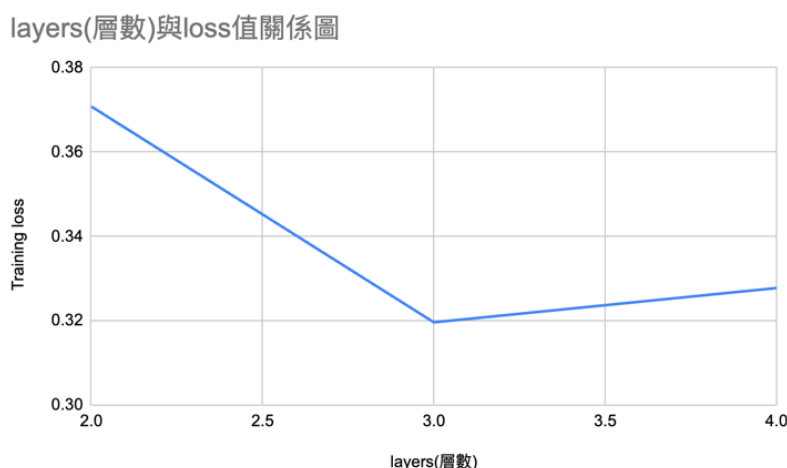
在這份深度學習實作中，學習率（learning rate）是一個至關重要的參數，它控制著模型在每次更新時調整參數的幅度。而在深度學習中的 **Epoch** 指的是**完整遍歷一次訓練數據集的過程**。在一個 Epoch 中，模型會看到整個訓練數據集一次並進行參數更新。一個 Epoch 的完成意味著模型已經學習了整個訓練數據集，並且我們可以開始下一個 Epoch 或者停止訓練。通常情況下，我們會多次重複執行 Epoch，以便模型能夠更好地學習訓練數據的特徵並提高準確性。訓練深度學習模型時，Epoch 是一個很重要的超參數，適當的 Epoch 數量可以幫助模型達到更好的性能。儘管調整學習率可以影響模型的性能，但較大的學習率並不一定意味著模型性能會更好，如表一所示。

我們可以將學習率比喻為學習騎自行車時調整踩踏速度的控制器。如果學習率設置得太小，學習過程可能會變得緩慢，就像學習騎自行車需要花費更長的時間一樣。但是，若學習率設置得太大，可能會難以控制，容易導致失控並無法有效學習。

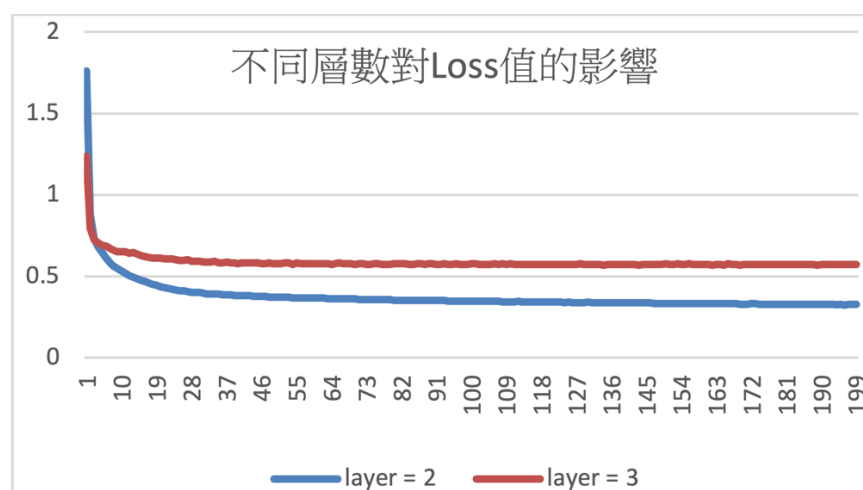
在深度學習中，若學習率設置得太小，模型更新速度將變慢，訓練時間可能會拉長。但若學習率設置得太大，則可能導致模型無法收斂，甚至發生梯度爆炸問題，使模型難以達到最佳解。因此，我們需要透過實驗和調參來找到一個適合的學習率，根據具體問題和數據進行調整。通常，我們會從較小的學習率開始，逐步增加，直到找到在訓練過程中能夠快速收斂且不失控的值。

五、探討不同 layers(層數)的節點對於 Loss 值的影響

1. 當層數(layer)為 2 且訓練 199 次時,Epoch: 199 Training Loss: 0.370831
2. 當層數(layer)為 3 且訓練 199 次時,Epoch: 199 Training Loss: 0.319628
3. 當層數(layer)為 4 且訓練 199 次時,Epoch: 199 Training Loss: 0.327762



圖（十七）不同 Layers 對於 Loss 值的影響（自行拍攝）



圖（十七-2）不同 Layers 對於 Loss 值的影響（自行拍攝）

由圖（十七）、圖（十七-2）可以知道運用 Python 訓練模型時，使用 3 層的效果最好，但是由於 8051 微控制器的記憶體空間有限，因此本實驗都以 2 層節點進行，原因如下：

深度學習模型中的層數，**更多的節點不一定意味著模型會更好。**

首先，增加層數會**增加模型的複雜性**，使得**模型更難訓練**。深度學習模型需要在訓練過程中學習到很多不同層次的特徵表示，如果模型太深或者層數太多，可能會導致梯度消失或梯度爆炸的問題，使得模型無法收斂或者收斂速度很慢。

此外，增加層數也會增加模型的計算成本和存儲需求。深度學習模型通常需要大量的計算資源來訓練和推理，如果模型過於龐大，可能會需要更多的計算資源，這在實際應用中可能會造成困難。

因此，我們在設計深度學習模型時，需要考慮到層數的適當性。**有時候，一個較淺但更有效的模型可能比一個較深但更複雜的模型效果更好。**在實際應用中，通常會通過實驗和調參來找到最適合的模型結構，以平衡模型的性能和計算成本。

六、運用 C 語言模擬手寫板辨識

圖（十八）為 讀取 8 x 8 的矩陣的程式碼，所畫的數字即是把所有值為 1 的元所連線所得的數字，並把其中將值是 1 的權重相加，最後再將最高者的作為輸出（如下圖（十九））

```
#include "hidden_layer.h"
#include "output_layer.h"
#include <stdio.h>

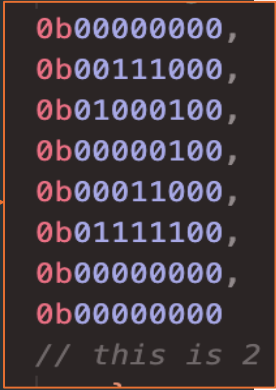
short int ReLu(short int number);

int main() {
    unsigned char word[] = {
        0b00000000,
        0b0011000,
        0b01000100,
        0b0100100,
        0b0100100,
        0b0100100,
        0b0100100,
        0b0011000,
        0b0000000,
        0b0000000
    };

    short int hidden_node[18] = { }; // 隱藏層節點的宣告
    int output_node[10] = { }; // 輸出層節點的宣告

    for (int i = 0; i < 18; i++) { // 分別計算 18 個節點
        for (int k = 0; k < 8; k++) {
            int temp = 0b10000000; // 此 temp 數字用於篩選 word[k] 的第 j 個數
            for (int j = 0; j < 8; j++) {
                if (temp & word[k]) { // 如果 word[k][j] 是 1 -> 則加上對應的權重
                    hidden_node[i] += hidden_layer_weight[64 * i + 8 * k + j]; // 將權重加到神經網路的隱藏節點
                }
                temp = temp >> 1; // 右移以篩選下一個數字
            }
        }
    }

    for (int i = 0; i < 18; i++) {
        hidden_node[i] = ReLu(hidden_node[i] + hidden_layer_bias[i]); // 將模型的偏差加到每個節點並應用激活函數 (ReLu 函數)
        printf("hidden node %d %d \n", i + 1, hidden_node[i]);
    }
}
```



The diagram on the right shows a vertical list of 10 binary strings, each representing a weight vector. An orange arrow points from the 7th binary string in the code (0b0100100) to the 7th binary string in the diagram (0b00011000). The diagram also shows the 8th binary string (0b01111100) and the 9th binary string (0b00000000). Below these, it says "// this is 2".

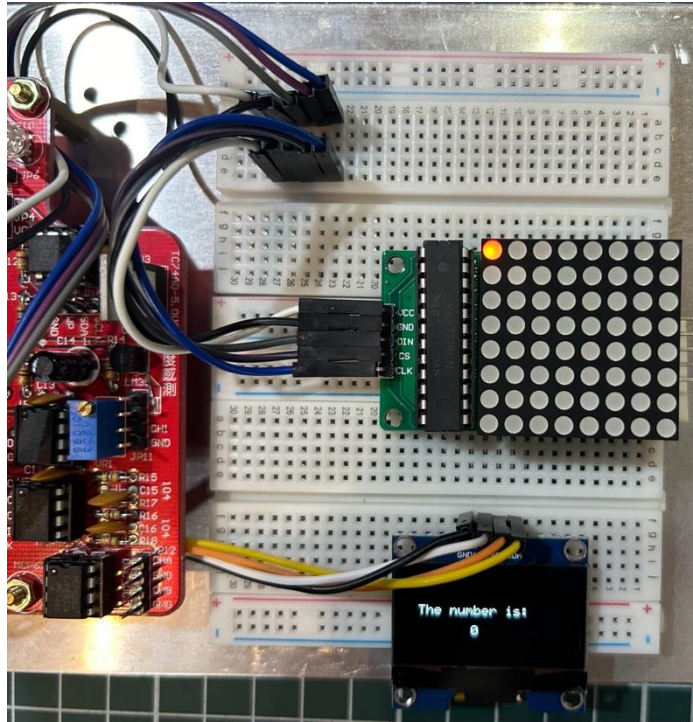
圖（十八）將手寫板上值=1 的位置加上其所對應的 weight 得到 Hidden_node,在近一步取得最終所求的 output_node（自行拍攝）

```
.....
.###..
.#...#..
.....#..
...##...
.####..
.....
.....
number0, possibility:0
number1, possibility:14818
number2, possibility:17925
number3, possibility:0
number4, possibility:0
number5, possibility:0
number6, possibility:0
number7, possibility:13293
number8, possibility:12672
number9, possibility:0
the number you draw is " 2 "
```

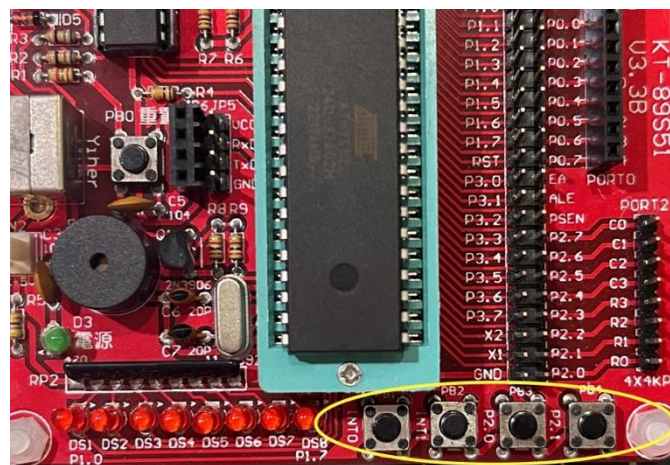
圖（十九）辨識成功（自行拍攝）

七、 在微處理機(8051 微控制器)上的實作

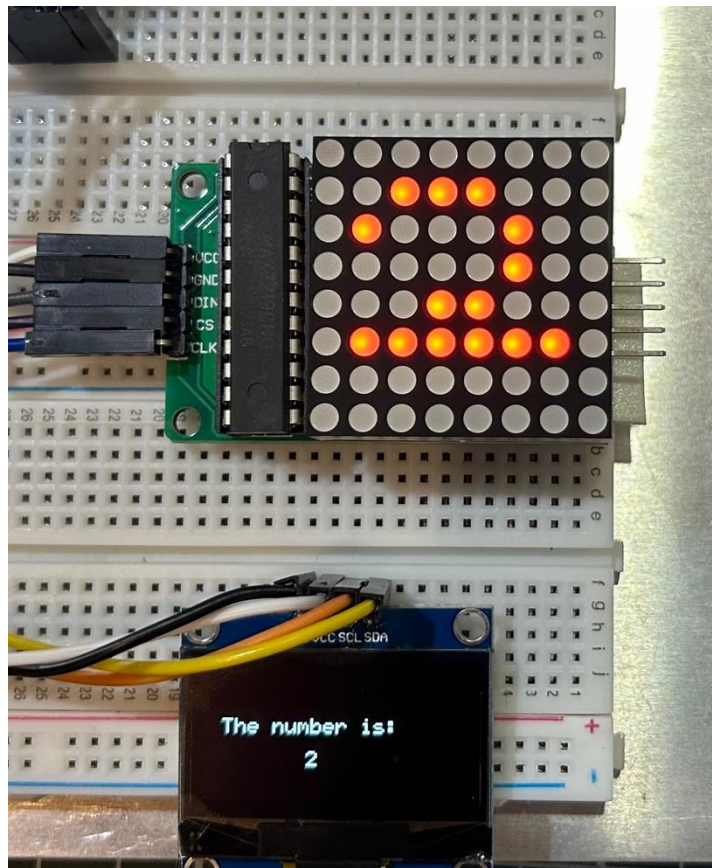
如圖（二十），能夠藉由手指指向的四個按鈕控制 LED 燈的位置。如圖（二十一），為 8051 板右下角的 4 顆按鈕，由左至右分別能控制 LED 燈向左、向右、寫入、完成操作，並藉由 OLED 板輸出可能性(possibility)最大的數字。以輸入 2 為例，如圖（二十二），輸入 6，如圖（二十三）。



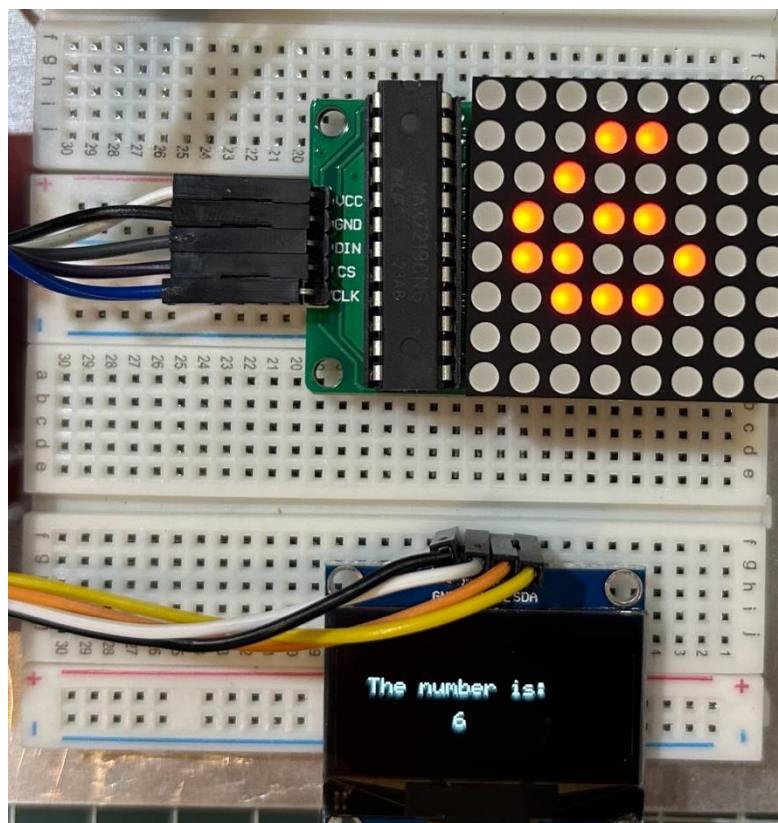
圖（二十）尚未寫入任何數字的矩陣顯示器（自行拍攝）



圖（二十一）為處理器右下角的四個按鈕（自行拍攝）



圖（二十二）以輸入數字 2 為例（自行拍攝）

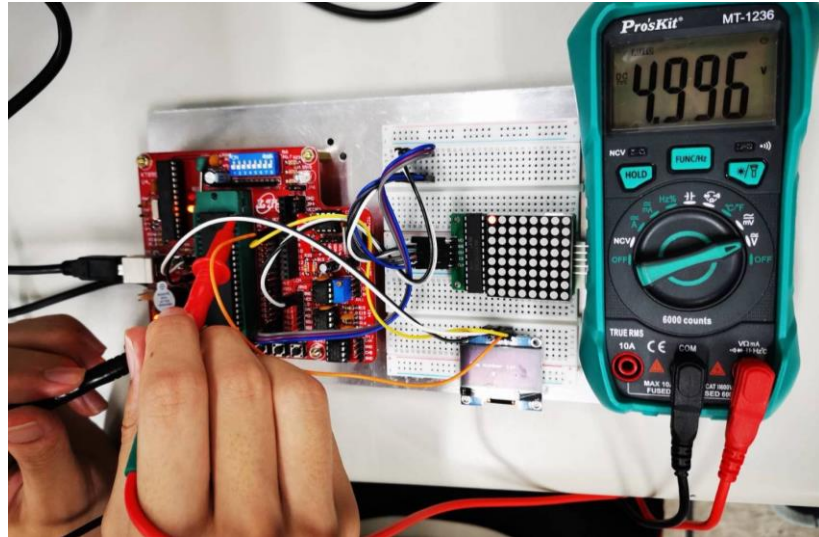


圖（二十三）以輸入數字 6 為例（自行拍攝）

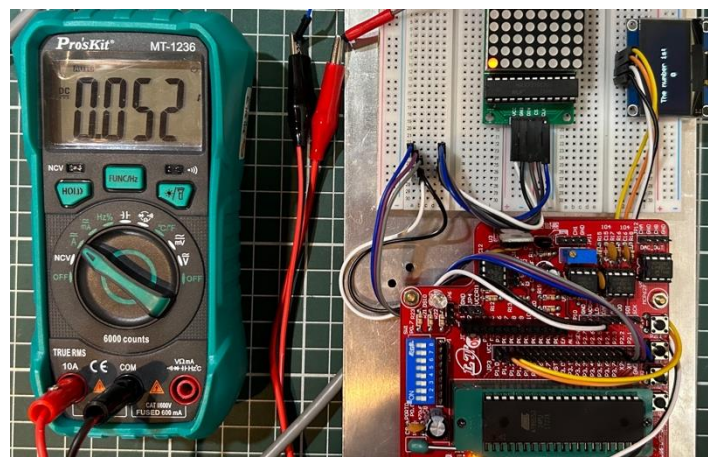
八、量測 8051 為處理器在運算時的耗電量

1. 量測電壓與電流

為了證實本次實作能夠比其他 AI 的低功耗，決定運用三用電表來測量運算時所需的電壓(伏特)如下圖（二十四），與電壓(安培)如圖（二十五），並分別得到 4.996(V) 與 0.052(A)的結果。



圖（二十四）運用三用電表測量 8051 運算時的電壓（自行拍攝）



圖（二十五）運用三用電表測量 8051 運算時的電流（自行拍攝）

2. 計算消耗功率

在得知 8051 在計算時的電流與電壓之後，由公式 $W(\text{瓦特}) = I(\text{安培}) \times V(\text{伏特})$ 推得運算時的功率為 0.25(W)。

陸、研究結果與未來展望

- 一、在 8051 微控制器上應用深度學習進行數字辨識的研究中，我完成了利用 Python 訓練參數的目標、C 語言操縱硬體，並在只有 8 位元的微處理器上成功的進行數字辨識。
- 二、通過對不同訓練次數(Epoch)對模型效能的影響進行實驗如圖（十六），發現在本深度學習研究中，軟體裡運用 Python 訓練參數(Epoch)200 次、訓練參數(Epoch)300 次差幾乎沒有差異，因此為了節省儲存空間以及時間，選用訓練次數(Epoch)200 的參數。
- 三、通過對不同層數(layers)對模型效能的影響進行實驗如圖（十七），發現在使用三層層數雖然能夠取得較好的效果。但是因為硬體儲存空間只有 128bytes，本次的實驗皆採取 2 層節點來進行數字辨識，仍也有相當高的準確度(loss 為 0.325367537)。
- 四、透過表（一）的數據顯示在訓練次數相同的情況下 Learning Rate = 0.01 所得的 Loss 值較小，因此選擇採用 Learning Rate = 0.01 的模型。
- 五、本研究成果達到聯合國永續發展目標(Sustainable Development Goals, SDGs)，如（圖二十五）的
 1. SDG 7 - Affordable and Clean Energy: 透過控制 8051 微處理器來達到節省能源的目標，符合可負擔且乾淨的能源使用。
 2. SDG 9 - Industry, Innovation, and Infrastructure: 探討了 Python 訓練深度學習模型、不同超參數對模型的影響並搭配 8051 微控制器進行影像的識別，涉及到創新和基礎設施的發展。



圖（二十五）SDGS 示意圖（資料來源：聯合國官網）

在只有 8 位元的微處理器上成功的進行數字辨識，未來，我將使用更高階的軟體改進模型的效能和準確性，計劃探索更複雜的模型結構，如卷積神經網絡（CNN），以提高模型的擬合能力和泛化能力，並普及這種低功耗的 AI，將繼續優化學習參數和訓練過程，以提高模型的收斂速度和準確率並應用到影像及聲音的辨識。

柒、參考文獻與資料

1. 台灣大學李宏毅教授對於深度學習的教學影片
<https://www.youtube.com/watch?v=O2Vkp8dJ5FE> 檢索日期 2024 年 2 月 28 日
2. 李宏毅教授對於 Network 解說
<https://www.youtube.com/watch?v=OP5HcXJg2Aw> 檢索日期 2024 年 2 月 28 日
3. GitHub 上寫好的數字模組
<https://github.com/Hanksome1/Digitclassifier-in-C/blob/main/Simulation/numbers.txt>
4. Deep Learning: 用 Python 進行深度學習的基礎理論實作 作者：斎藤康毅
譯者：吳嘉芳 出版社：歐萊禮
5. Knock Knock! Deep Learning: 新手入門深度學習的敲門磚 作者：廖珮妤
繪者：張巧心 出版社：博碩文化股份有限公司
6. 運用 Google Colab 上的 Python 訓練模型參數
<https://colab.research.google.com/drive/1V92Uz5TTwsCj3RCfPXfl--b4tgDTVJsJ?authuser=1#scrollTo=7bQ-YxjOFpxq>
7. Amazon 對於神經網路的解釋
<https://aws.amazon.com/tw/what-is/neural-network/>
8. Glints 對於機器學習與深度學習差異的比較
<https://glints.com/tw/blog/what-does-a-machine-learning-engineer-need/>
9. 聯合國永續發展目標(Sustainable Development Goals, SDGs)
<https://sdgs.un.org/>

捌、附錄

1. 控制 8051 判斷數字的 C 語言

```
#include "hidden_layer.h"
#include "output_layer.h"
#include <stdio.h>
short int ReLu(short int number);
int main() {
    unsigned char word[]={
0b00000000,
0b00011000,
0b00011000,
0b00011000,
0b00011000,
0b00011000,
0b00000000,
0b00000000,
0b00000000
    };
    short int hidden_node[18]= { } ;
    int output_node[10]= { };
    for(int i=0; i<18; i++){
        for(int k=0; k<8; k++){
            int temp=0b10000000;
            for(int j=0; j<8;j++){
                if(temp & word[k]){
                    hidden_node[i]+=hidden_layer_weight[64*i+8*k+j];
                }
                temp=temp>>1;
            }
        }
    }
    for(int i=0; i<18; i++){
        hidden_node[i] = ReLu(hidden_node[i]+hidden_layer_bias[i]);
        printf("hidden node %d %d \n", i+1, hidden_node[i]);
    }
    for(int i=0; i<10; i++){
        for(int j=0; j<18; j++){
```

```

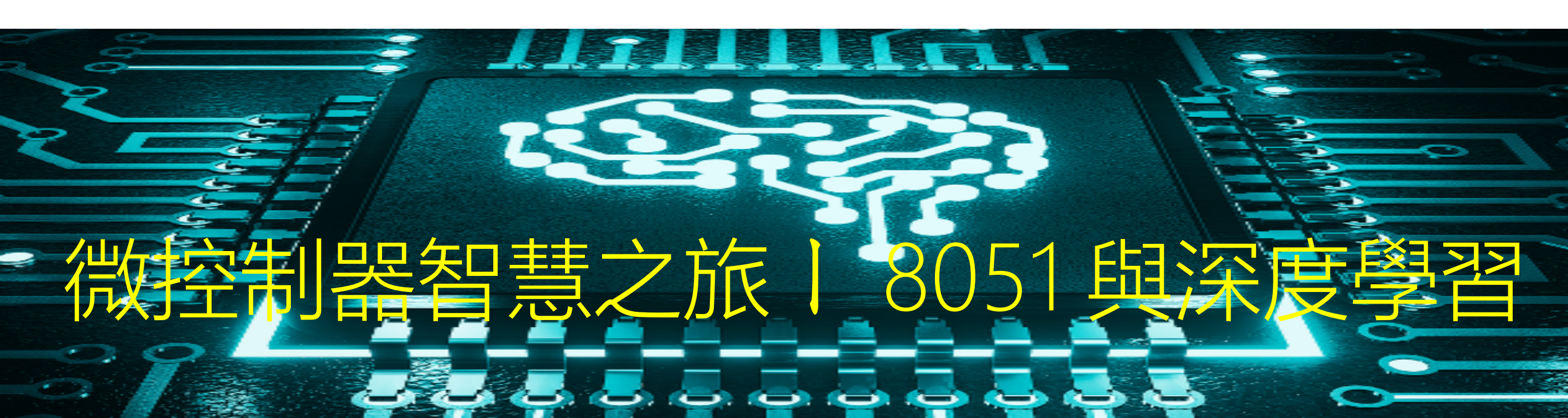
output_node[i]+=hidden_node[j]*output_layer_weight[18*i+j];
    }
}
for(int i=0; i<10; i++){
    output_node[i] = ReLu(output_node[i]+output_layer_bias[i]);
//
}
for(int k=0; k<8; k++){
    int temp=0b100000000;
    for(int j=0; j<8;j++){
        if(temp & word[k]){
            printf("#");
        }
        else printf(".");
        temp=temp>>1;
    }
    printf("\n");
}
for(int i=0;i<10; i++){
    printf("number%d, possibility:%d\n", i, output_node[i]);
}
int highest=0;
int highest_number=output_node[0];
for(int i=0;i<10;i++){
    if(output_node[i]>highest_number){
        highest=i;
        highest_number=output_node[i];
    }
}
printf("the number you draw is \" %d \", highest);
}
short int ReLu(short int number){
    if(number>0) return number;
    else return 0;
}

```

【評語】 052310

1. 本研究探討 AI 深度學習應用於 8051 開發板的數字辨識，擬在低功耗的情況下進行 AI 實作來達到環境保育，值得鼓勵！
2. 作品可達成在硬體受限的情況下進行 AI 實作，達成學生軟硬體實作能力。
3. 建議探討不同運算對耗電之影響與原因。高速運算是否亦有省電建議說明耗電之計算方式及基礎計算方式及基礎。
4. 作品可強化與作品相關技術之過去研究文獻的了解。

作品簡報



微控制器智慧之旅 | 8051 與深度學習

摘要

本研究旨在將 AI 深度學習應用於 8051 微處理器的數字辨識，希望在低功耗的情況下進行 AI 實作來達到環境保育的目標，探討在硬體受限的情況下如何操縱硬體並使用 Python 進行參數訓練。通過對不同層數和學習速率的實驗，我們發現兩層層數和 0.01 的學習速率能取得較好效果。在模型訓練中，我們探討了不同 Learning rate 對 Loss 值的影響以及不同 Layers 對 Loss 值的影響。此外我們還量測 8051 的工作電壓、電流，並介紹了 8051 微控制器的基本原理和應用。未來，也將繼續改進模型效能和準確性，並優化硬體設計和性能，應用深度學習技術於更廣泛的領域。

壹、研究動機

當前 AI 深度學習在現代是一個熱門的研究領域，伴隨 AI 功能日益強大，計算量與耗電量也成為備受關注的議題。並非所有 AI 應用都需大量電力才能運行，因此本研究想藉由控制 8051 微處理器結合 AI 深度學習，以實現數字辨識，並達成聯合國永續發展目標(Sustainable Development Goals, SDGs)中若干目標。

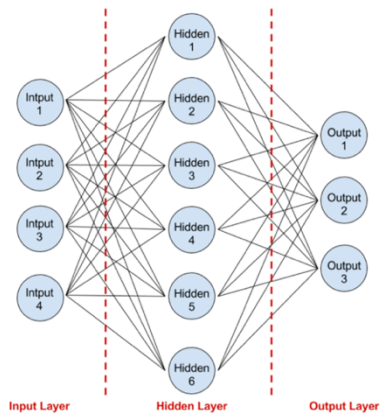
貳、研究目的

- 一、運用 Python 訓練深度學習模型。
- 二、探討不同超參數(Learning rate, Layers, Epoch)對模型的影響。
- 三、藉由 C 語言模擬手寫板辨識並將程式碼燒入至 8051 微控制器上。
- 四、測試手寫辨識在 8051 微控制器上的辨識結果。

參、原理介紹

一、深度學習介紹：

深度學習是一種機器學習的方法，它模仿人腦神經元之間的連接方式，通過多層次的神經網絡來學習和理解數據。這種學習方式使得機器能夠自動地從數據中學習到複雜的特徵和模式。深度學習的核心是神經網絡，它由多層次 (Layer) 的神經元組成，每一層都將上一層的輸出作為輸入,如圖(一)。



圖（一）神經網路示意圖
(參考來源：amazon 官網)

二、Loss 值介紹：

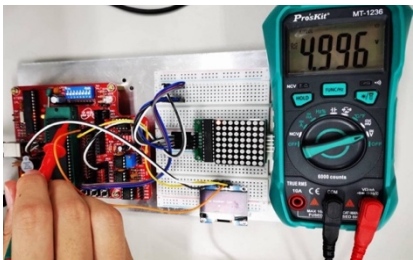
在深度學習中，「Loss 值」是一個重要的概念，用於衡量模型預測與真實值之間的差異。Loss 函數通常是一個數值，表示模型在訓練過程中的性能表現。Loss 值越小，代表模型的預測結果與真實值越接近，本次實作採用 Cross-entropy（交叉熵）的方法計算 Loss 值。

量測 8051 微處理器在運算時的耗電量

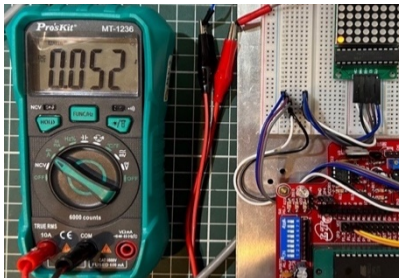
使用三用電表測量 8051 微處理器在運算時的電壓和電流。根據圖(二)所示，測得電壓為 4.996 伏特；根據圖(三)所示，測得電流為 0.052 安培。由此推算，8051 微處理器在運算時的功率約為 0.26 瓦，計算如下：功率(P)=電壓(V)×電流(I)

$0.26W \approx 4.996V \times 0.052A$,

測量結果表明，8051 微處理器在運算時的功耗相對較低。

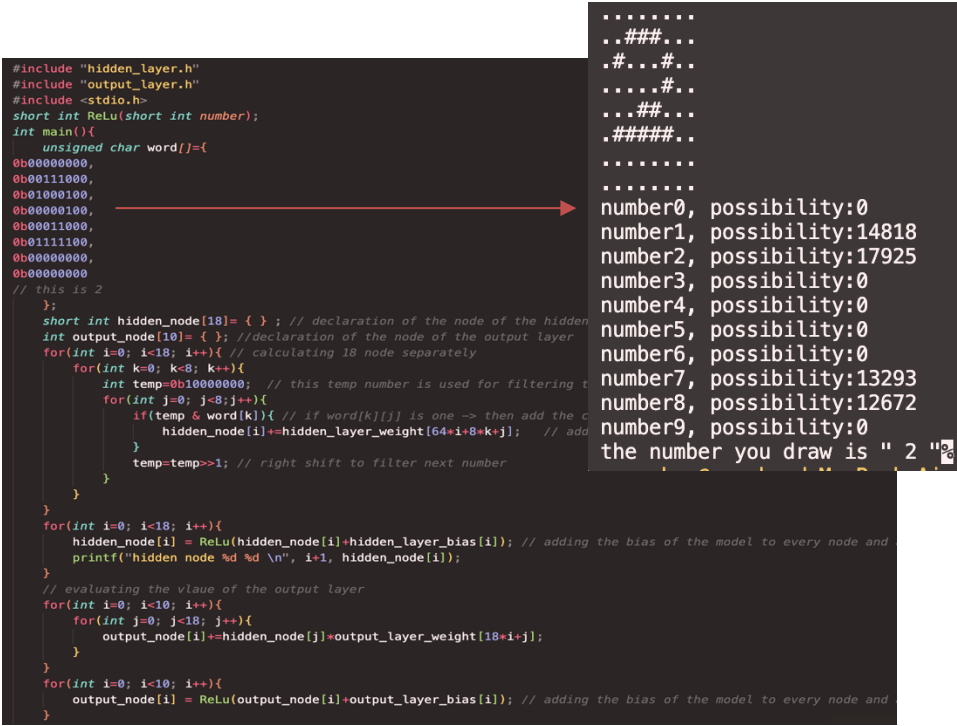
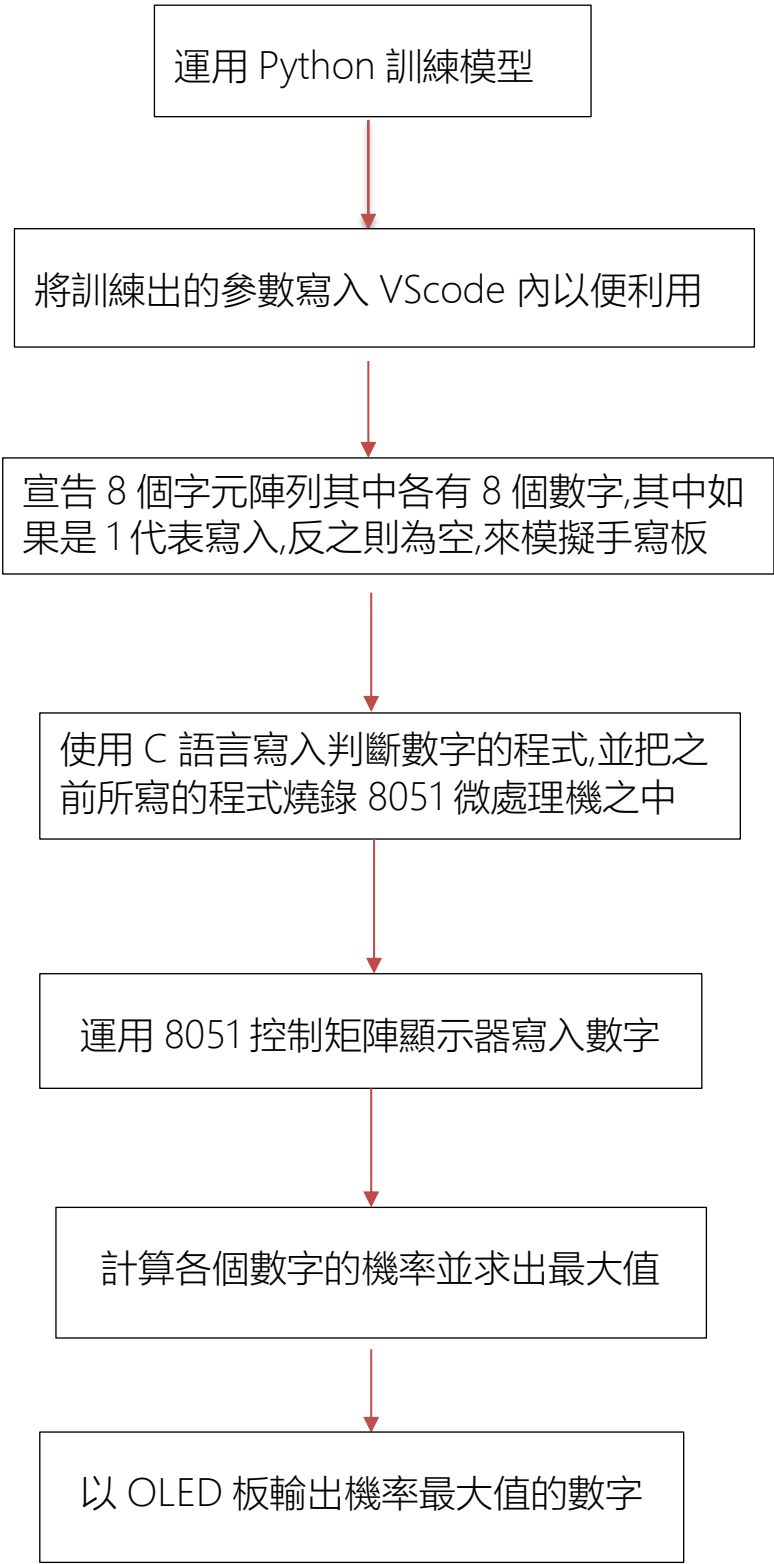


圖(二)量測工作電壓(參考來源：自行拍攝)

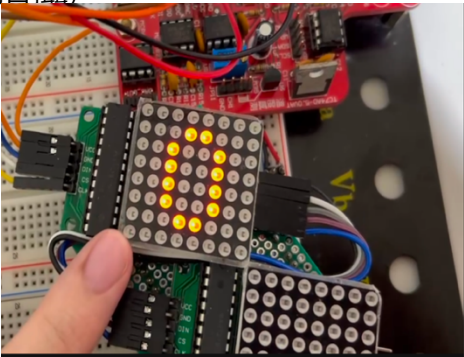


圖(三)量測工作電流(參考來源：自行拍攝)

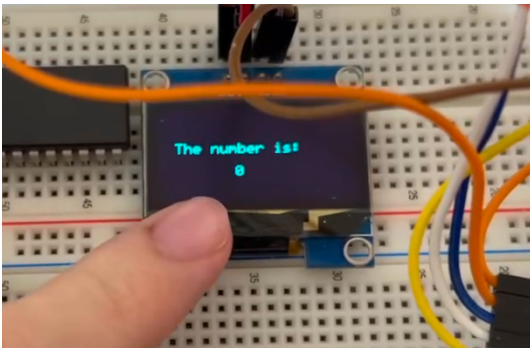
一、實驗流程



自撰 C 語言模擬在 8051 上的情況，在每個數字都測試成功後便把程式與模型燒錄 8051(參考來源：自行拍攝)

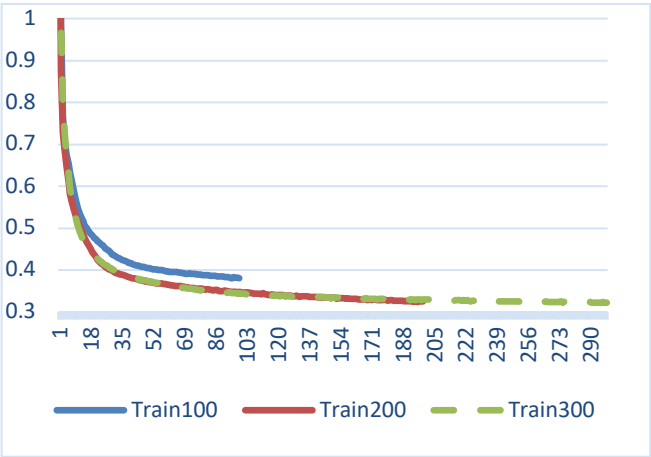


8051 輸入數字(參考來源：自行拍攝)

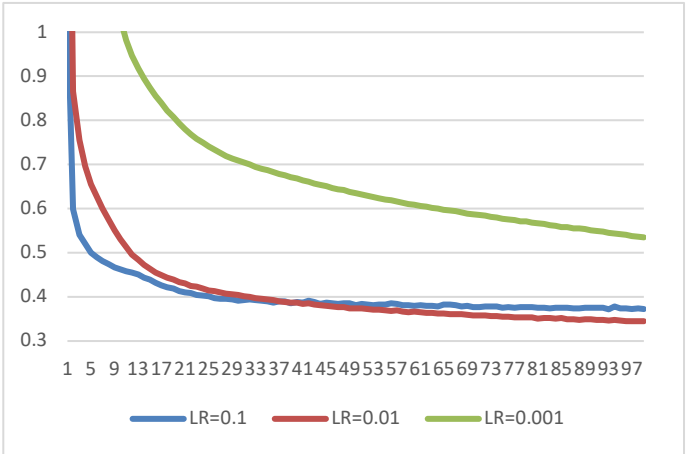


OLED 板輸出結果(參考來源：自行拍攝)

二、不同訓練次數、學習速率以及層數探討 Loss 值的關係



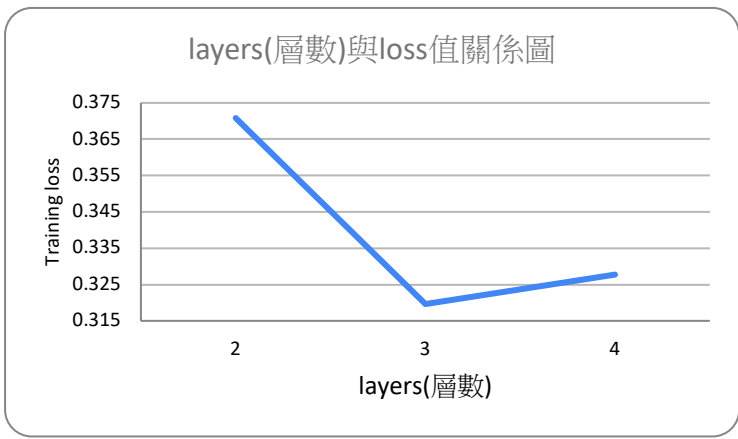
不同訓練次數與 Loss 值的關係圖參考來源：自行繪製



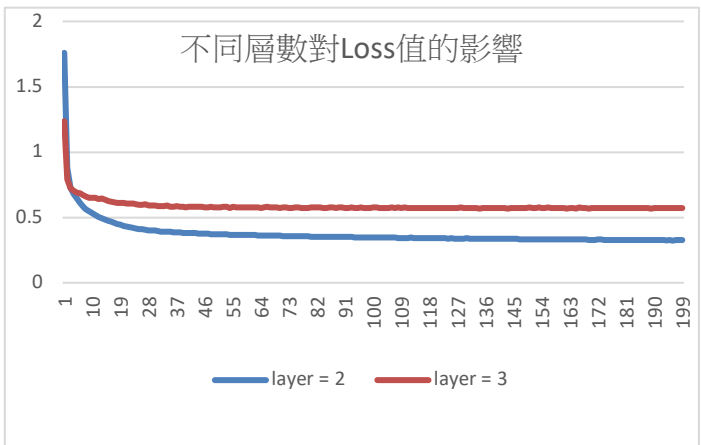
不同 Learning Rate 與 Loss 值關係圖參考來源：自行繪製

量測訓練 200 次與訓練 300 次的 Loss 值幾乎沒有任何差別，因此選擇使用訓練 200 次的參數

學習率設置得太小，模型更新速度將變慢，訓練時間可能會拉長。但若學習率設置得太大，則可能導致模型無法收斂，甚至發生梯度爆炸問題，使模型難以達到最佳解。



在硬體資源一般的情況之下層數與 Loss 值關係圖參考來源：自行繪製



結點數調降之下層數與 Loss 值關係圖參考來源：自行繪製

由於 8051 的內部 RAM 只能存 256Bytes(2056Bits)因此在存滿三層的情況下將會沒有辦法燒錄，而在兩層存滿情況下的辨識率與未限制硬體時的三層模型的辨識率相比其實並沒有相差太多，反而是有限制硬體情況下，兩層的辨識率就開始跟三層拉開差距了。但是其實導致這個結果的另一個原因也有可能是因為增加層數會增加模型的複雜性，使得模型更難訓練。

伍、 研究結果與討論

- 一、在 8051 微控制器上應用深度學習進行數字辨識的研究中，我完成了利用 Python 訓練參數的目標、C 語言操縱硬體，並在只有 8 位元的微處理器上成功的進行數字辨識。
- 二、通過對不同訓練次數(Epoch)對模型效能的影響進行實驗，發現在本深度學習研究中，軟體裡運用 Python 訓練參數(Epoch)200 次、訓練參數(Epoch)300 次差幾乎沒有差異，因此為了節省儲存空間以及時間，選用訓練次數(Epoch)200 的參數。
- 三、通過對不同層數(layers)對模型效能的影響進行實驗，發現在使用三層層數雖然能夠取得較好的效果。但是因為硬體儲存空間只有 128bytes，本次的實驗皆採取 2 層節點來進行數字辨識，仍也有相當高的準確度。
- 四、透過表（一）的數據顯示在訓練次數相同的情況下 Learning Rate = 0.01 所得的 Loss 值較小，因此選擇採用 Learning Rate = 0.01 的模型。
- 五、本研究成果達到聯合國永續發展目標(Sustainable Development Goals, SDGs)
 1. **SDG 7 - Affordable and Clean Energy**. 透過控制 8051 微處理器來達到節省能源的目標，符合可負擔且乾淨的能源使用。
 2. **SDG 9 - Industry, Innovation, and Infrastructure**. 將深度學習應用於 8051 微處理器上，是一種創新的技術應用，其低功耗的情況下進行 AI 實作，將有助於環境保護，亦符合 SDG 9 中促進創新和技術進步的目標。

在只有 8 位元的微處理器上成功的進行數字辨識，未來，我將使用更高階的軟硬體改進模型的效能和準確性，計劃探索更複雜的模型結構，如卷積神經網絡 (CNN)，以提高模型的擬合能力和泛化能力。將繼續優化學習參數和訓練過程，以提高模型的收斂速度和準確率並應用到影像及聲音的辨識。

陸、 參考文獻

1. Deep Learning: 用 Python 進行深度學習的基礎理論實作 作者：齋藤康毅
譯者：吳嘉芳 出版社：歐萊禮
2. Knock Knock! Deep Learning: 新手入門深度學習的敲門磚 作者：廖珮妤
繪者：張巧心 出版社：博碩文化股份有限公司
3. Cross-Entropy 介紹以及教學 <https://reurl.cc/kO6dOx>