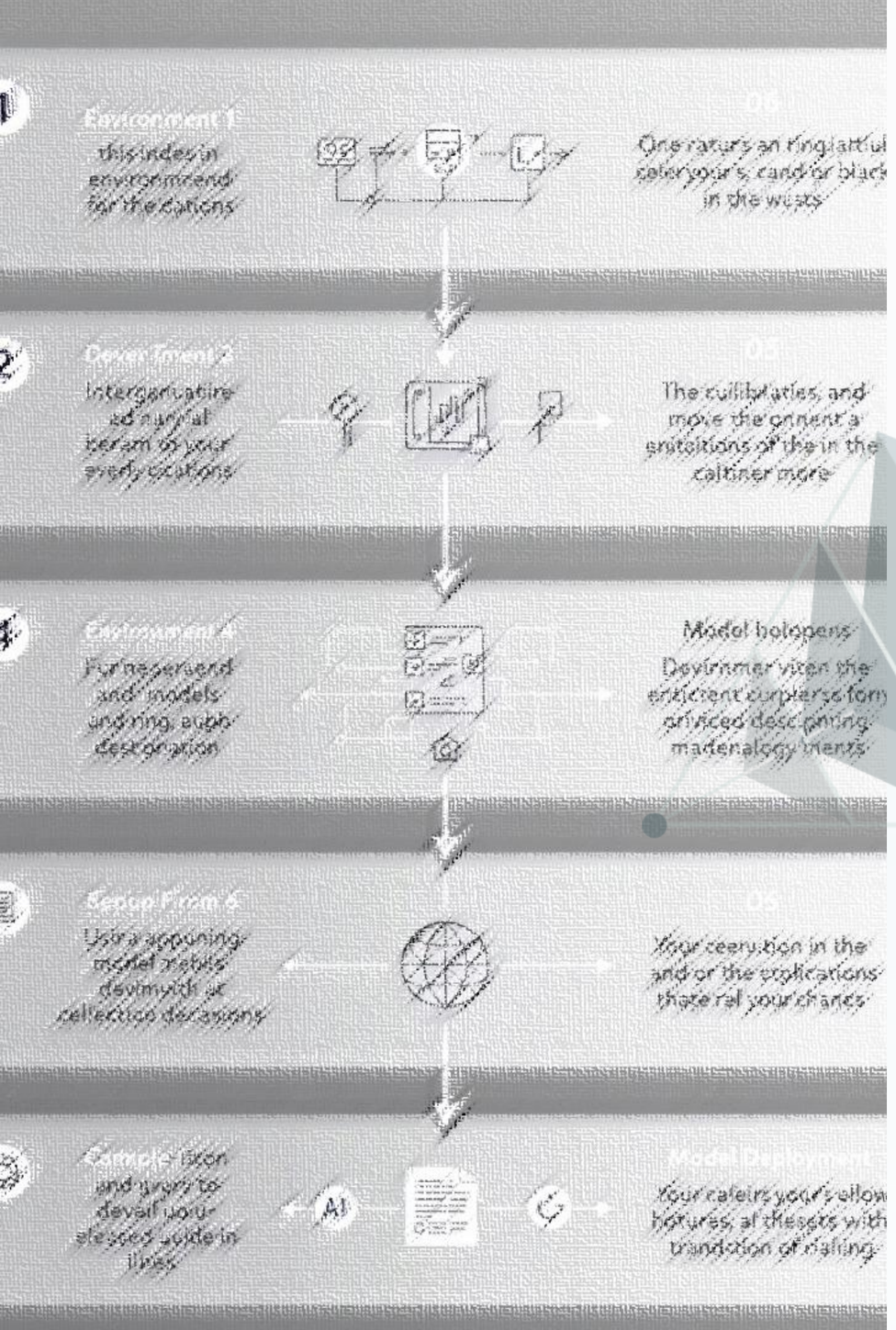
Hub 8735 Ultra AI 應用開發課程

本課程將帶領學員深入 Hub 8735 Ultra 開發板的 AI 應用開發，從環境建置、程式燒錄、硬體搭配，到深度學習模型的訓練與部署。

Hub 8735 Ultra 具備強大的 AI 推理能力，適用於邊緣運算，並支援 YOLO 物件偵測。本課程將指導學員如何透過 Robotflow 平台訓練自訂模型，轉換為 Hub 8735 Ultra 可運行的格式，並學習如何透過 SD 卡或 Flash 記憶體加載 AI 模型，最終打造專屬的 AI 應用系統。

簡報者：百兆鍶國際有限公司



課程大綱

1 環境建置與開發板應用

目標：掌握開發環境設定、Hub 8735 Ultra 程式開發流程，並成功燒錄與執行 AI 應用。

- Hub 8735 Ultra 環境建置
- AI 應用程式開發
- 實作與測試

2 YOLO 訓練與模型部署

目標：學會如何訓練自訂 YOLO 模型，轉換格式並在 Hub 8735 Ultra 上執行物件偵測應用。

- 自訂 YOLO 模型訓練
- 模型格式轉換與部署
- 整合與應用

課程收穫

熟悉 Hub 8735 Ultra

學員將能夠熟悉 Hub 8735 Ultra 的環境建置與開發流程，為後續的 AI 應用開發奠定基礎。

YOLO 物件偵測

瞭解 YOLO 物件偵測模型的訓練與應用，能夠自行訓練並部署自訂模型。

模型格式轉換

學會模型格式轉換與部署技術，能夠將訓練好的模型轉換為 Hub 8735 Ultra 可運行的格式。

SD/Flash 加載模型

使用 SD 卡或 Flash 記憶體加載模型，提升運行效率，實現更靈活的 AI 應用部署。

本課程適合具備基礎 Arduino 或 AI 應用經驗的開發者、學生或工程師，並將提供完整的範例程式與實作指引，幫助學員快速上手並應用於實際專案開發。

硬體設置

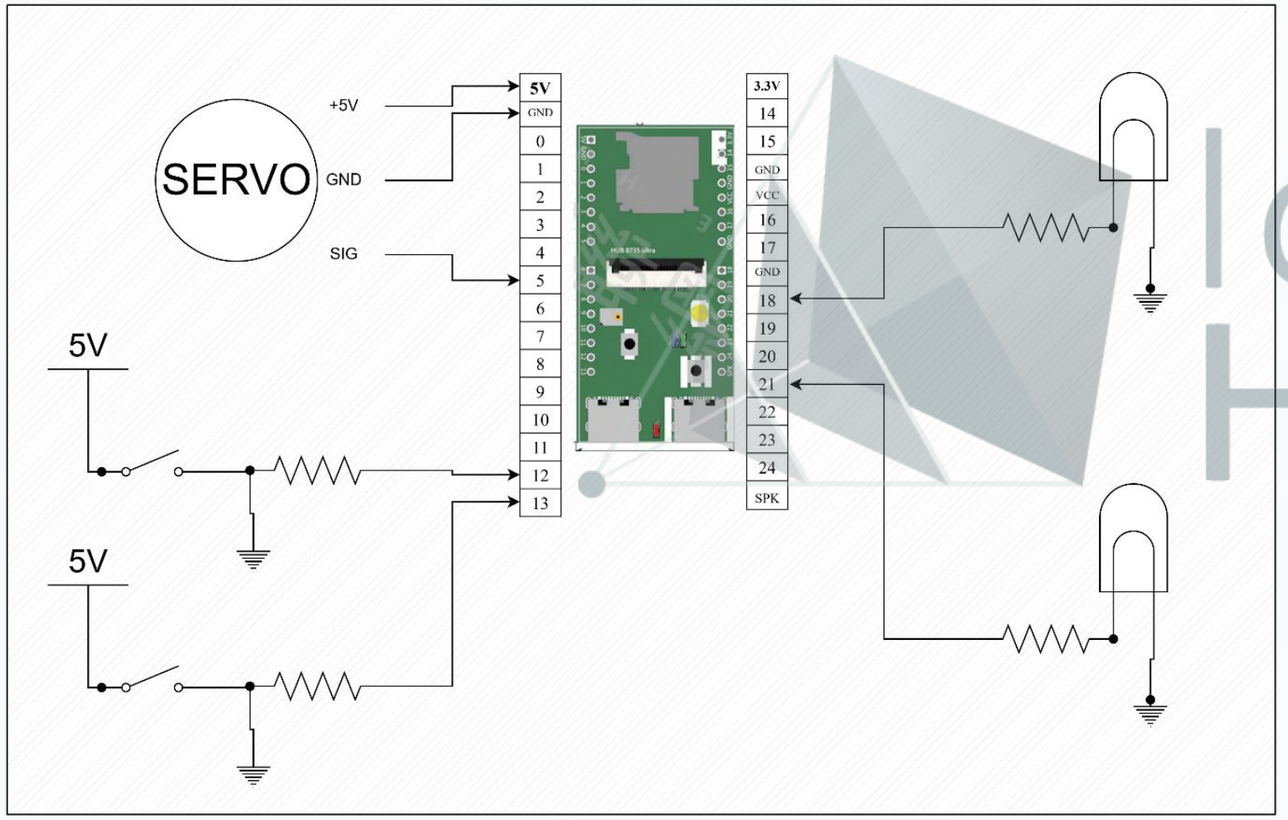
1 硬體需求

本專案所需的元件包括：

- Hub 8735 Ultra *1
- SD 卡 *1（用於儲存 AI 模型或設定）
- 按鈕 *2（作為用戶輸入控制）
- 綠色 LED *1（可選，指示成功狀態）
- 紅色 LED *1（可選，指示失敗或錯誤狀態）
- 伺服馬達 *1（例如 Tower Pro SG90，作為門鎖解鎖機構）
- 220 歐姆電阻 *2（限流 LED）
- 10K 歐姆電阻 *2（按鈕的下拉電阻）

2 硬體連接方式

詳細的接線步驟包括伺服馬達、LED 指示燈和按鈕開關的連接。
每個元件都有特定的 GPIO 接腳和連接方式。



以下是詳細的接線步驟，請對應 電路圖 進行組裝。

◇ (1) 伺服馬達 (Servo) 接線

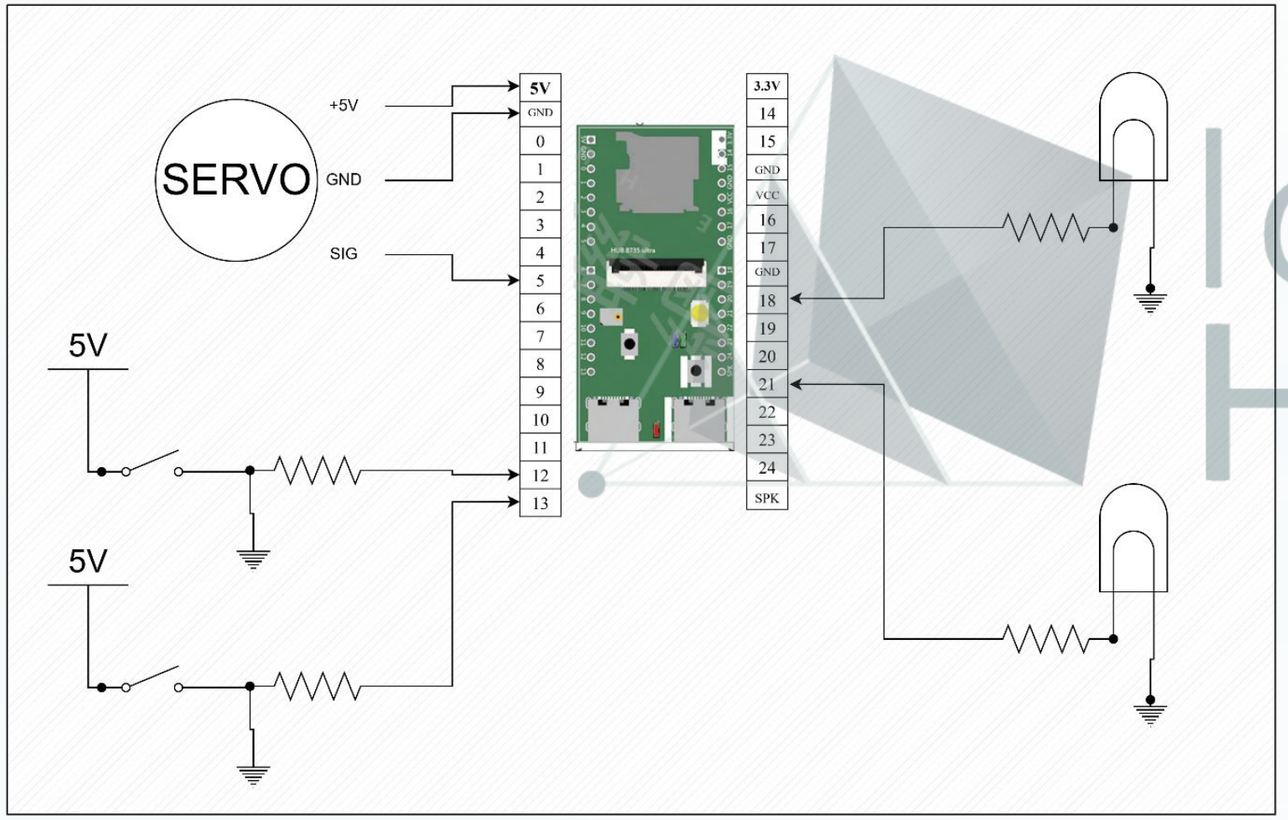
伺服馬達用於模擬門鎖機構的開啟與關閉，通常包含三條線：

- 紅色 (VCC) → 5V (Hub 8735 Ultra)
- 黑色 (GND) → GND
- 黃色 (SIG) → GPIO 5

☑ GPIO 5 負責 PWM 信號輸出，控制伺服馬達角度。

2 硬體連接方式

詳細的接線步驟包括伺服馬達、LED 指示燈和按鈕開關的連接。
每個元件都有特定的 GPIO 接腳和連接方式。



以下是詳細的接線步驟，請對應 電路圖 進行組裝。

◇ (2) LED 指示燈接線

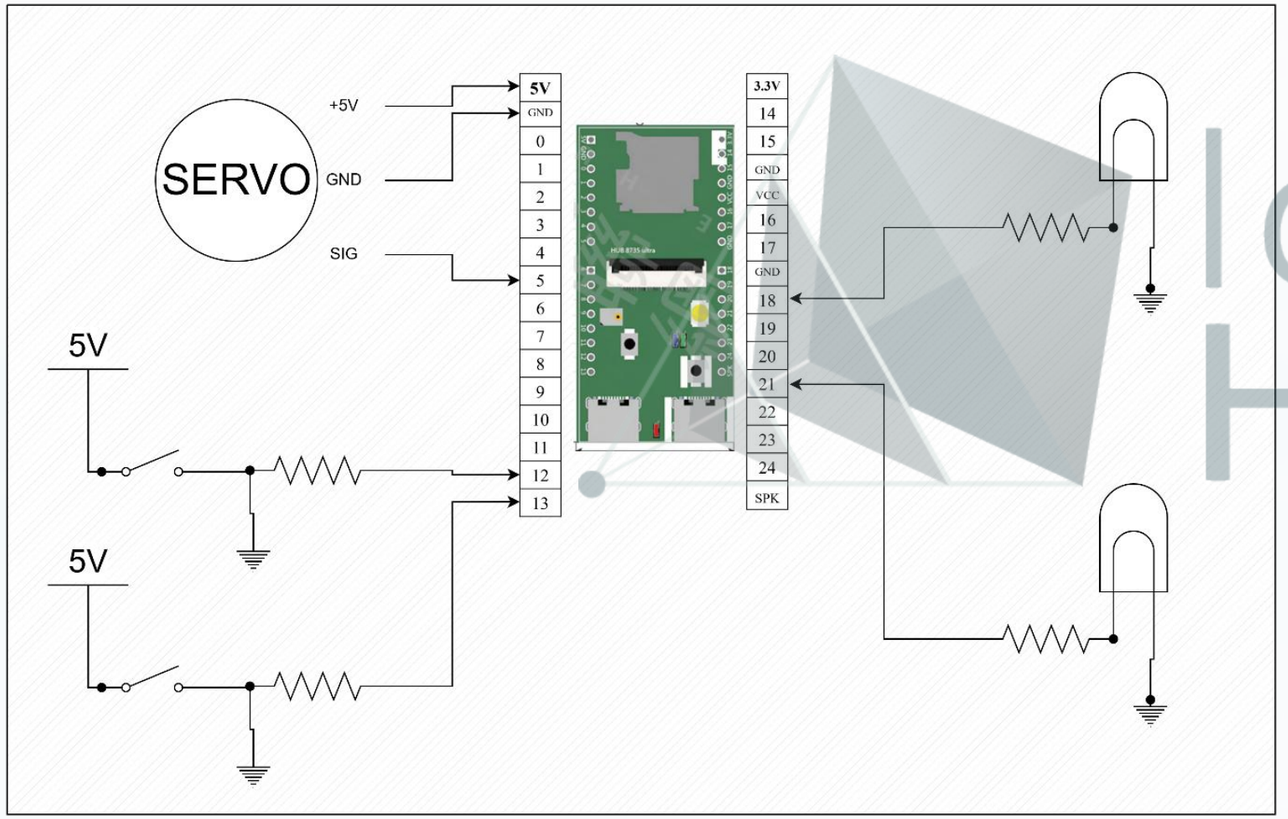
LED 指示燈用來顯示執行狀態，例如：

- **綠色 LED (成功狀態)**
  - 正極 (長腳) → 透過 220Ω 電阻連接到 GPIO 18
  - 負極 (短腳) → GND
- **紅色 LED (失敗/錯誤狀態)**
  - 正極 (長腳) → 透過 220Ω 電阻連接到 GPIO 19
  - 負極 (短腳) → GND

☑ **GPIO 18 & 19 分別控制綠色與紅色 LED，根據 AI 偵測結果進行狀態顯示。**

2 硬體連接方式

詳細的接線步驟包括伺服馬達、LED 指示燈和按鈕開關的連接。
每個元件都有特定的 GPIO 接腳和連接方式。



以下是詳細的接線步驟，請對應 電路圖 進行組裝。

◇ (3) 按鈕開關接線

按鈕用於控制 AI 訓練或手動觸發門鎖開啟：

- **按鈕 1 (註冊新臉部)**
  - 一端接 GPIO 12
  - 另一端接 5V
  - 透過 10KΩ 電阻下拉至 GND
- **按鈕 2 (手動開鎖)**
  - 一端接 GPIO 13
  - 另一端接 5V
  - 透過 10KΩ 電阻下拉至 GND

☑ **當按鈕被按下時，GPIO 會讀取高電位 (HIGH)，觸發對應的動作。**

3 硬體安裝與測試

完成接線後，需要進行基本的硬體測試，包括按鈕功能、LED 顯示和伺服馬達動作測試。

1. **確保所有元件已正確連接**，特別是：
    - **按鈕是否接入 5V 與下拉電阻**
    - **LED 的正負極連接是否正確**
    - **伺服馬達的 PWM 訊號線是否接到 GPIO 5**
2. **將 SD 卡插入 Hub 8735 Ultra，確保可用**
3. **接上 5V 電源**
4. **開發板上電，進行測試**
    - 按下 **按鈕 1**：應該觸發註冊動作
    - 按下 **按鈕 2**：應該觸發門鎖解鎖
    - 伺服馬達應根據狀態轉動
    - 綠色/紅色 LED 根據 AI 識別結果變化

硬體測試與除錯

✔ 按鈕按下無反應 ？

- 檢查按鈕的 **5V 與 GPIO 是否正確連接**
- **10KΩ 下拉電阻** 是否正確連接至 GND

✔ 伺服馬達無法動作 ？

- 檢查 **PWM 訊號是否輸出**
- 確保 **GND 連接良好**
- 可能需要較大電流，嘗試外部 5V 供電

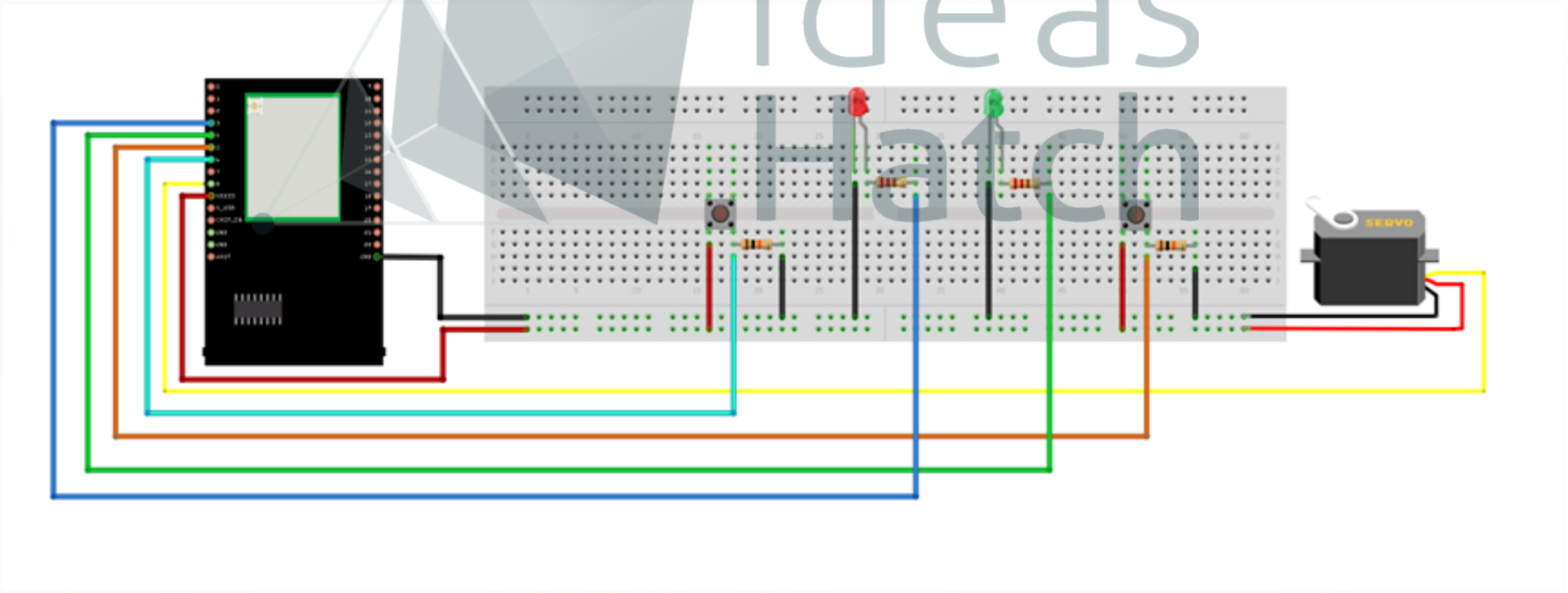
✔ LED 不亮 ？

- 確保 **220Ω 限流電阻** 存在
- 確保 **GPIO 設為輸出模式**

總結

本章節完成了 Hub 8735 Ultra 與按鈕、LED 以及伺服馬達的完整接線，並進行基礎測試。

接下來的章節將進入 軟體程式設計與 AI 應用開發，讓硬體能夠與 AI 模型進行互動。

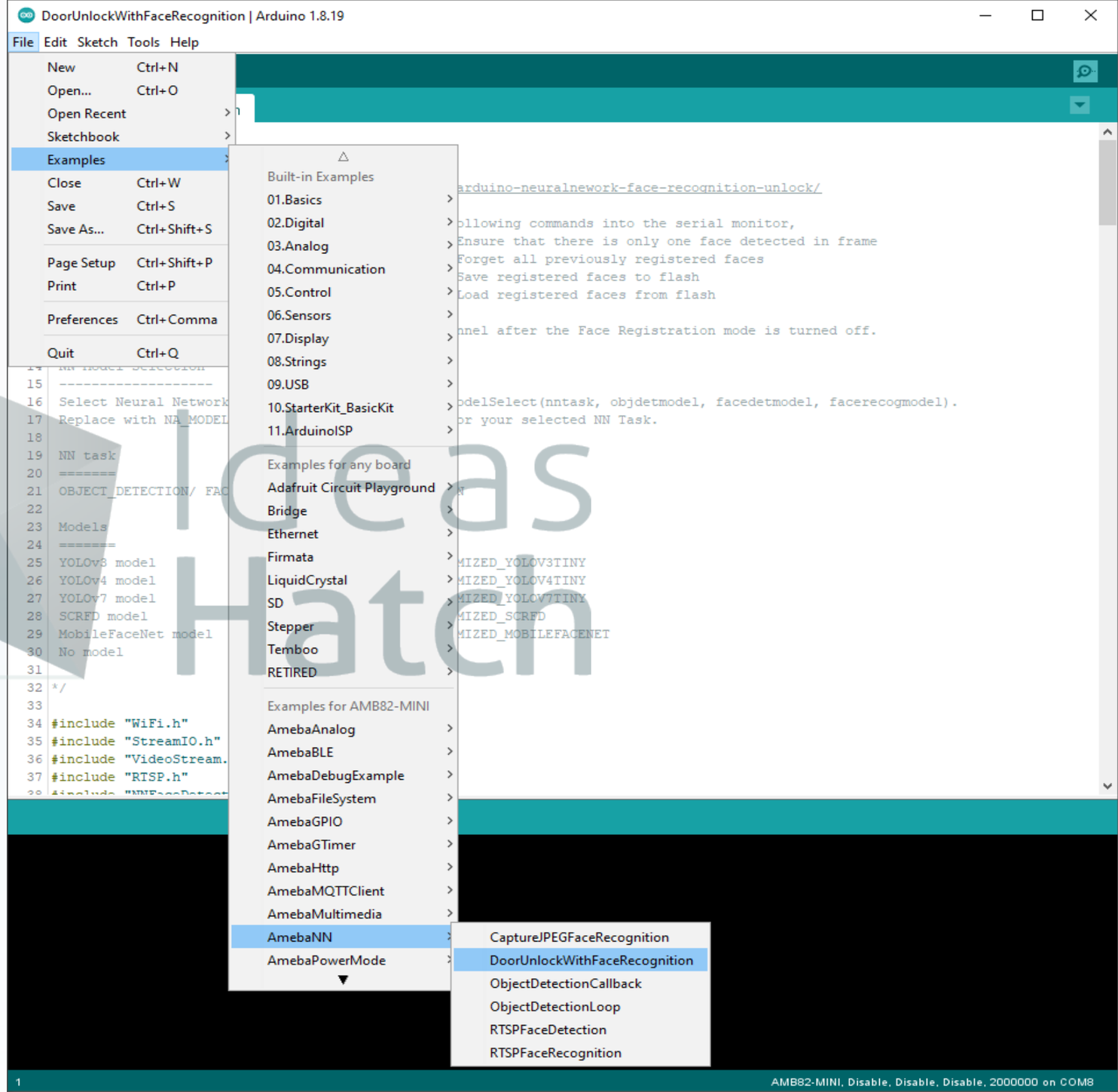


環境設置

本章節將指導學生如何在 Arduino IDE 上設定 Hub 8735 Ultra 的開發環境，包含 開發板管理員的設定、必要函式庫安裝，以及 測試範例程式 的執行，確保學員能夠順利開始開發。

安裝 Arduino IDE

從 Arduino 官方網站下載並安裝 Arduino IDE，建議使用 Arduino IDE 1.8.19 或 2.x 版本，本課程以 Arduino 1.8.19 進行示範。



新增 Hub 8735 Ultra 開發板

在 Arduino IDE 中手動新增 Hub 8735 Ultra 的 Board Manager URL，然後安裝相應的開發板套件。

◇ (1) 設定開發板管理員 URL

打開 Arduino IDE，點選上方選單的**「檔案 (File)」 → 「偏好設定 (Preferences)」**

在 **「Additional Boards Manager URLs」** 欄位中，填入以下網址：

https://github.com/ideashatch/HUB-8735/raw/main/amebapro2_arduino/Arduino_package/ideasHatch.json

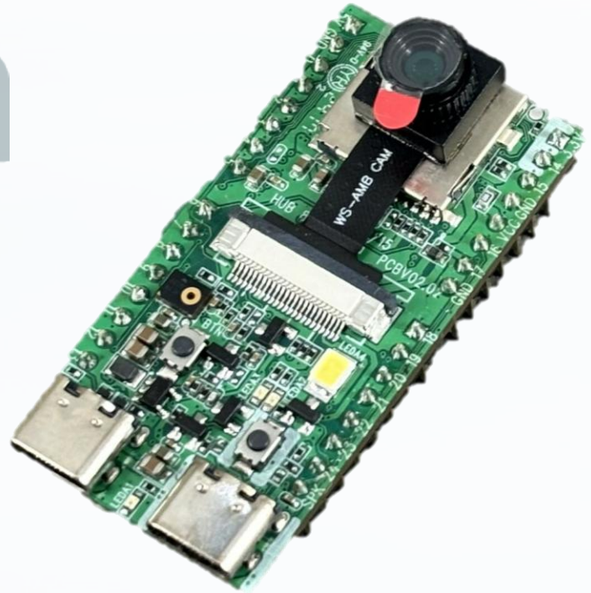
按下「確定 (OK)」以儲存設定。

◇ (2) 安裝 Hub 8735 Ultra 開發板

點選 **「工具 (Tools)」 → 「開發板 (Board)」 → 「開發板管理員 (Boards Manager)」**

在搜尋欄輸入 **Hub 8735**，找到相應的開發板套件。

點擊 **「安裝 (Install)」** 按鈕，等待下載與安裝完成。



驗證安裝與選擇開發板

選擇開發板：

- 點選 **「工具 (Tools)」→「開發板 (Board)」**
- 選擇 **「Hub 8735 Ultra」**

選擇序列埠 (COM Port)：

- 連接開發板到電腦後，前往 **「工具 (Tools)」→「Port」**
- 選擇 **對應 Hub 8735 Ultra 的序列埠 (通常為 COMx 或 /dev/ttyUSBx)**

安裝必要函式庫

打開「程式庫管理員 (Library Manager)」：

- 點選 **「工具 (Tools)」→「程式庫管理員 (Library Manager)」**

搜尋並安裝以下函式庫：

- **WiFi.h**（WiFi 連線）
- **Stream.h**（串流處理）
- **RTSP.h**（影像串流）
- **AmebaNN**（Hub 8735 AI 推理函式庫）

測試範例程式

◇ **(1) 開啟範例程式**

點選 **「檔案 (File)」** → **「範例 (Examples)」**

選擇 **「AmebaNN」** → **「DoorUnlockWithFaceRecognition」**

這是一個 **臉部識別門鎖解鎖的範例**，適合測試 AI 功能。

◇ **(2) 燒錄程式**

確保 **開發板與序列埠** 設定正確。

點選 **「上傳 (Upload)」** 按鈕，將程式燒錄到 **Hub 8735 Ultra**。

開啟序列監視器 (Serial Monitor)，確認輸出是否正常。

環境設置總結

- ☑ **完成 Arduino IDE 設定與開發板安裝**
- ☑ **成功安裝必要函式庫**
- ☑ **選擇正確的開發板與序列埠**
- ☑ **成功燒錄測試範例**

這代表 **Hub 8735 Ultra** 的開發環境已設置完成，接下來可以進行 **硬體測試與 AI 應用開發！** 🚀

程式碼燒錄

選擇開發板與 COM Port

進入 Flash Download 模式

上傳程式

燒錄除錯

- **(1) 選擇 Hub 8735 Ultra**
  1. **打開 Arduino IDE**
  2. 點擊 **「工具 (Tools)」→「開發板 (Board)」**
  3. 在搜尋框輸入 **hub 8735**，選擇 **「HUB-8735 Ultra」**
- **(2) 選擇正確的 COM Port**
  1. **開啟「工具 (Tools)」→「Port (埠)」**
  2. 選擇 **「HUB-8735 Ultra」對應的 COM Port**
     - 在 Windows 通常顯示為 **COMx** (如 COM11、COM12)
     - 在 macOS/Linux 通常顯示為 **/dev/ttyUSBx**
  3. 若不確定 COM Port，可先拔除開發板，觀察消失的 Port，再重新插上開發板來確認。

程式碼燒錄

選擇開發板與 COM Port

進入 Flash Download 模式

上傳程式

燒錄除錯

- **(1) 透過按鍵進入 Download 模式**

  1.按住「功能鍵 (Function Button)」不放

  2.短按一下「RESET 鍵」

  3.放開「功能鍵」

  4.確認開發板已進入 Download 模式

  ->若進入成功，Arduino IDE 應能夠偵測到正確的 COM Port。

- **(2) 確保短路設定**

  如果需要，開發板上還有一個 BOOT_V3P3 & BOOT_MODE 短路點，某些情況下可能需要透過短接方式進入燒錄模式。

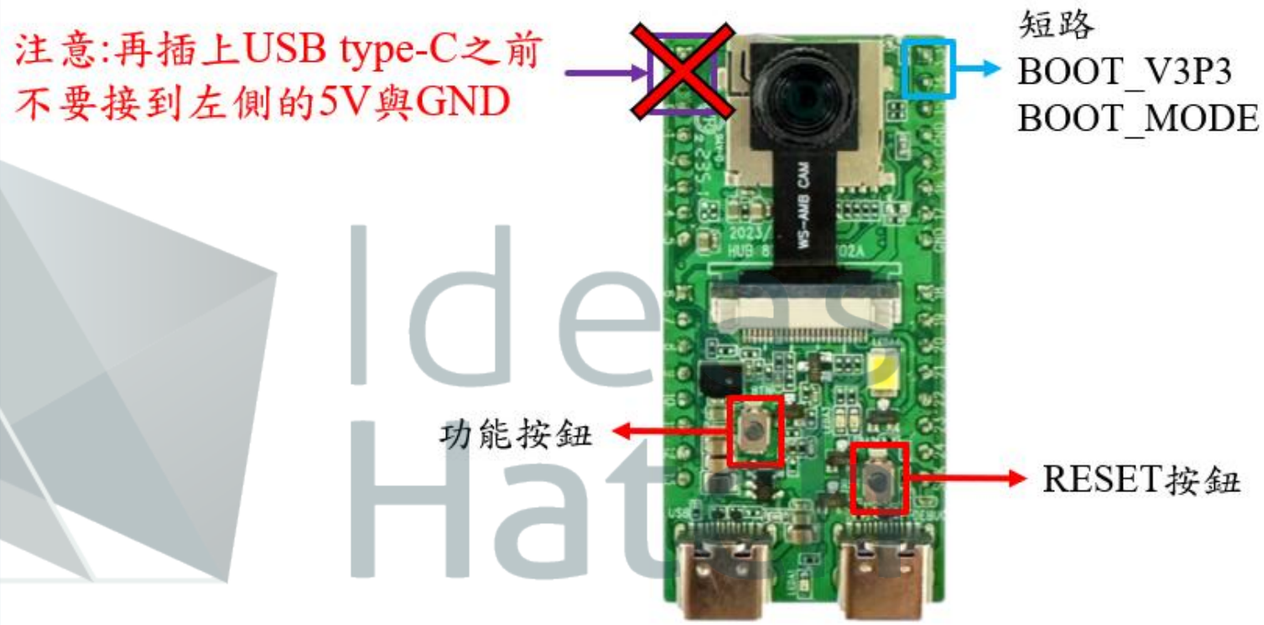
程式碼燒錄

選擇開發板與 COM Port

進入 Flash Download 模式

上傳程式

燒錄除錯



- 可直接透過按住功能鍵與短按Reset鍵進入Download mode。
- (仍保留短路3V3及Pin14的方式)

⚠ 注意：

請確保 USB Type-C 接上之前，不要接錯 5V 與 GND 位置，避免燒毀開發板！

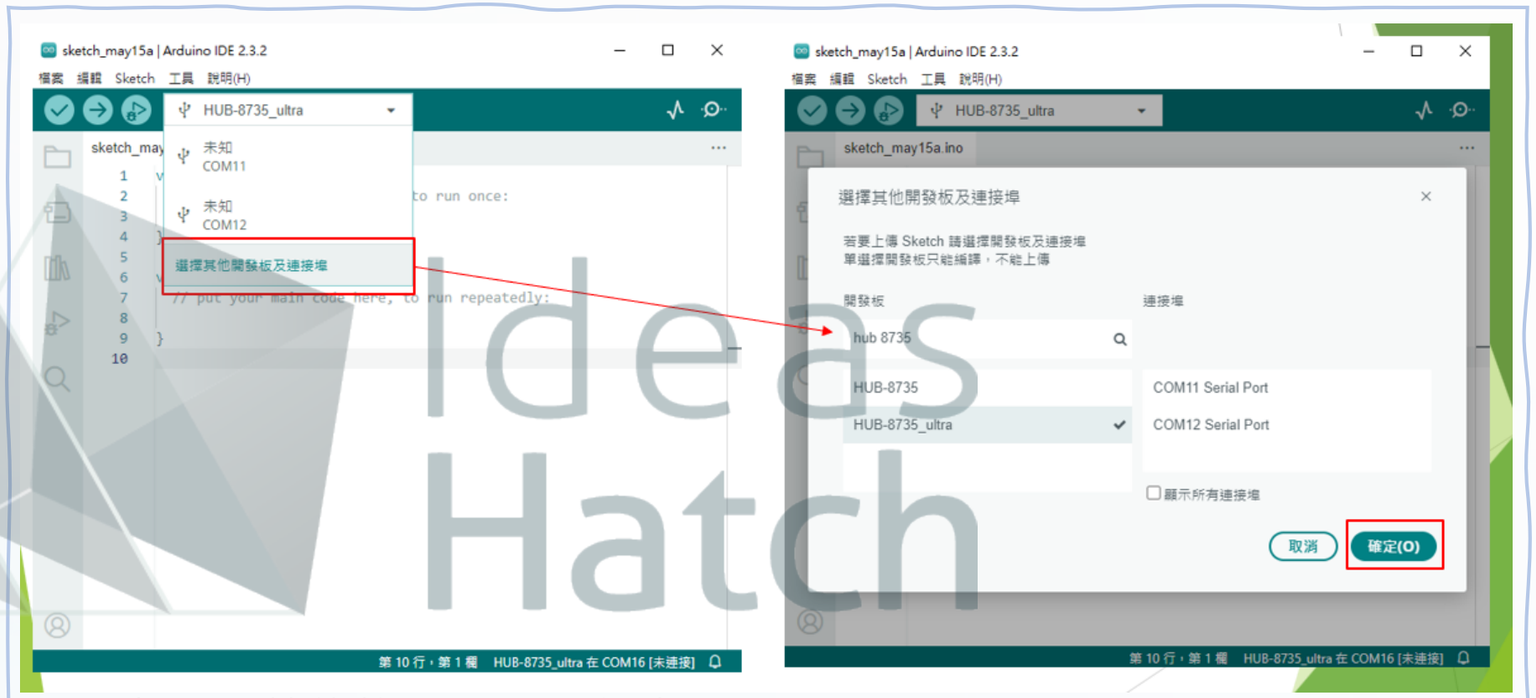
程式碼燒錄

選擇開發板與 COM Port

進入 Flash Download 模式

上傳程式

燒錄除錯



1. 打開要燒錄的 Arduino 程式碼
2. 點擊「**上傳 (Upload)**」按鈕
3. IDE 會開始編譯與上傳，請等待「**Done Uploading**」訊息出現
4. 如果成功，開發板會重新啟動並執行新的程式碼

程式碼燒錄

選擇開發板與 COM Port

進入 Flash Download 模式

上傳程式

燒錄除錯

如果程式燒錄失敗，請依照以下步驟排查問題：

✔ **COM Port 未顯示？**

- 檢查 USB Type-C 連線是否正常
- 在「工具」→「開發板管理員」重新安裝 **Hub 8735 Ultra** 驅動

✔ **無法進入 Download 模式？**

- 嘗試 **重新按住功能鍵 + 按一下 Reset**，確保按鍵順序正確
- 確保 BOOT_MODE 設定正確 (若適用)

✔ **燒錄失敗或 Upload Card Error？**

- 檢查是否選擇了 **正確的開發板型號**
- 嘗試降低 **燒錄 Baud Rate** (可在開發板選項調整)

變數與物件初始化

設定 GPIO、WiFi、RTSP、AI 模型等

此區塊負責：

- 設定 **GPIO 腳位**（LED、按鈕、伺服馬達）
- 設定 **RTSP 串流**
- 設定 **臉部辨識 AI 模型**
- 初始化 **WiFi 連線**
- 設定 **影像處理** 與 **串流物件**

變數與物件初始化

變數與函式庫

```
#include "WiFi.h"
#include "StreamIO.h"
#include "VideoStream.h"
#include "RTSP.h"
#include "NNFaceDetectionRecognition.h"
#include "VideoStreamOverlay.h"
#include <AmebaServo.h>
#include "AmebaFatFS.h"
```

- **WiFi.h**：用於連接無線網路
- **StreamIO.h / VideoStream.h**：處理影像串流
- **RTSP.h**：設定 RTSP（即時影像串流）
- **NNFaceDetectionRecognition.h**：處理 **臉部辨識**
- **AmebaServo.h**：用於控制伺服馬達
- **AmebaFatFS.h**：處理 **SD 卡** 或 **Flash 記憶體** 儲存

設定影像通道與影像解析度

```
#define CHANNELVID  0     // RTSP 影像串流通道
#define CHANNELJPEG 1     // 擷取快照用的通道
#define CHANNELNN   3     // AI 模型專用的影像通道

#define NNWIDTH  576
#define NNHEIGHT 320
```

- **CHANNELVID**：用於即時 RTSP 影像串流
- **CHANNELJPEG**：用於拍照擷取快照
- **CHANNELNN**：提供 AI 訓練影像，格式為 RGB
- **NNWIDTH / NNHEIGHT**：設定 **AI 影像解析度**（576x320）

變數與物件初始化

📌定義 GPIO 腳位

```
#define RED_LED                21
#define GREEN_LED              22
#define BACKUP_FACE_BUTTON_PIN 19
#define EN_REGMODE_BUTTON_PIN  18
#define SERVO_PIN              8
```

- **紅色 LED (GPIO 21)**：表示錯誤或無法開門
- **綠色 LED (GPIO 22)**：表示成功開門
- **按鈕 1 (GPIO 19)**：備份已註冊的臉部
- **按鈕 2 (GPIO 18)**：進入臉部註冊模式
- **伺服馬達 (GPIO 8)**：控制開關門（180° 鎖住，0° 解鎖）

📌設定影像串流與 AI 物件

```
VideoSetting configVID(VIDEO_FHD, 30, VIDEO_H264, 0);
VideoSetting configJPEG(VIDEO_FHD, CAM_FPS, VIDEO_JPEG, 1);
VideoSetting configNN(NNWIDTH, NNHEIGHT, 10, VIDEO_RGB, 0);

NNFaceDetectionRecognition facerecog;
RTSP rtsp;
StreamIO videoStreamer(1, 1);
StreamIO videoStreamerFDFR(1, 1);
StreamIO videoStreamerRGBFD(1, 1);
AmebaServo myservo;
```

- **configVID**：設定 RTSP 串流的影像格式（Full HD、30 FPS、H.264 編碼）
- **configJPEG**：設定拍照模式（JPEG 格式）
- **configNN**：設定 AI 模型的影像輸入解析度（RGB 格式）
- **facerecog**：建立臉部辨識物件
- **rtsp**：設定 RTSP 影像串流
- **videoStreamer**：影像串流處理物件
- **myservo**：伺服馬達物件

變數與物件初始化

📌WiFi 連線資訊

```
char ssid[] = "你的SSID";     // WiFi SSID
char pass[] = "你的密碼";         // WiFi 密碼
int status = WL_IDLE_STATUS;
```

- 這裡設定了 **WiFi 名稱 (SSID)** 與 **密碼**
- status 用來儲存 WiFi 連線狀態

📌變數宣告

```
bool doorOpen = false;
bool backupButtonState = false;
bool RegModeButtonState = false;
bool regMode = false;
uint32_t img_addr = 0;
uint32_t img_len = 0;
String fileName;
long counter = 0;

// File Initialization
AmebaFatFS fs;
```

- **doorOpen**：表示門是否開啟
- **backupButtonState / RegModeButtonState**：儲存按鈕狀態
- **regMode**：是否進入「臉部註冊模式」
- **img_addr / img_len**：儲存影像快照的記憶體地址與長度
- **fileName**：儲存辨識到的人臉名稱
- **counter**：用於計數快照存檔編號
- **fs (AmebaFatFS)**：負責 SD 卡或 Flash 記憶體的檔案管理

setup() 函式

硬體初始化、WiFi 連線、攝影機與 RTSP 設定

第二個區塊：setup() 初始化函式

setup() 是 Arduino 內建的初始化函式，在開機或重置後只會執行 **一次**。這段程式碼負責 **設定 GPIO、WiFi 連線、影像串流、臉部辨識模型** 以及 **伺服馬達** 的初始狀態。

第二個區塊：setup() 初始化函式

setup() 是 Arduino 內建的初始化函式，在開機或重置後只會執行 **一次**。這段程式碼負責 **設定 GPIO、WiFi 連線、影像串流、臉部辨識模型** 以及 **伺服馬達** 的初始狀態。

```
void setup()
{
    // GPIO Initialisation
    pinMode(RED_LED, OUTPUT);
    pinMode(GREEN_LED, OUTPUT);
    pinMode(BACKUP_FACE_BUTTON_PIN, INPUT);
    pinMode(EN_REGMODE_BUTTON_PIN, INPUT);
    myservo.attach(SERVO_PIN);

    Serial.begin(115200);
```


- ◇ **設定 GPIO 腳位模式**
- pinMode(RED_LED, OUTPUT); → 設定紅色 LED 為輸出模式
- pinMode(GREEN_LED, OUTPUT); → 設定綠色 LED 為輸出模式
- pinMode(BACKUP_FACE_BUTTON_PIN, INPUT); → 設定 **備份臉部按鈕** 為輸入模式
- pinMode(EN_REGMODE_BUTTON_PIN, INPUT); → 設定 **臉部註冊模式按鈕** 為輸入模式
- myservo.attach(SERVO_PIN); → **將伺服馬達綁定到 GPIO 8**
- ◇ **初始化序列埠 (Serial Monitor)**
- Serial.begin(115200); → 設定 **序列監視器波特率為 115200**，用於 **偵錯與除錯輸出**。

setup() 初始化函式

📌 WiFi 連線

```
// Attempt to connect to Wifi network:
    while (status != WL_CONNECTED) {
        Serial.print("Attempting to connect to WPA SSID: ");
        Serial.println(ssid);
        status = WiFi.begin(ssid, pass);

        // wait 2 seconds for connection:
        delay(2000);
    }
```

這段程式碼會 **嘗試連接 WiFi**，如果失敗，會不斷重試：

1. 顯示正在連線的 WiFi **SSID**
2. 呼叫 WiFi.begin(ssid, pass); 開始連線
3. 每次失敗後等待 **2 秒**，然後再嘗試
4. **直到 WiFi 連線成功** 才會繼續執行

☑ **如果成功連線，WiFi 會變成 WL_CONNECTED，並繼續執行後續程式**

📌設定攝影機與影像串流

```
// Configure camera video channels with video format information
  Camera.configVideoChannel(CHANNELVID, configVID);
  Camera.configVideoChannel(CHANNELJPEG, configJPEG);
  Camera.configVideoChannel(CHANNELNN, configNN);
  Camera.videoInit();
```

這裡設定 **攝影機影像串流**：

- CHANNELVID（通道 0）：**RTSP 影像串流**
- CHANNELJPEG（通道 1）：**拍攝快照用**
- CHANNELNN（通道 3）：**提供 AI 辨識影像**

Camera.videoInit(); → 啟動攝影機影像串流

setup() 初始化函式

📌啟動 RTSP 串流

```
// Configure RTSP with corresponding video format information
    rtsp.configVideo(configVID);
    rtsp.begin();
```

這段程式碼設定 RTSP 影像串流：

1. rtsp.configVideo(configVID); → 設定 RTSP 影像格式
2. rtsp.begin(); → 啟動 **RTSP 影像串流**，可以透過 RTSP 串流軟體（如 VLC）觀看攝影機畫面

📌啟動臉部辨識模型

```
// Configure Face Recognition model
facerecog.configVideo(configNN);
facerecog.modelSelect(FACE_RECOGNITION, NA_MODEL, DEFAULT_SCRFD, DEFAULT_MOBILEFACENET);
facerecog.begin();
facerecog.setResultCallback(FRPostProcess);
```

這裡設定 **臉部辨識模型**：

1. facerecog.configVideo(configNN); → 指定 **使用 AI 影像通道**
2. facerecog.modelSelect(FACE_RECOGNITION, NA_MODEL, DEFAULT_SCRFD, DEFAULT_MOBILEFACENET);
   - DEFAULT_SCRFD：負責 **人臉偵測**
   - DEFAULT_MOBILEFACENET：負責 **臉部辨識**
3. facerecog.begin(); → 啟動臉部辨識模型
4. facerecog.setResultCallback(FRPostProcess); → 設定 **辨識結果的回調函式**

📌 **當攝影機偵測到臉部後，會自動呼叫 FRPostProcess() 來處理辨識結果！**

setup() 初始化函式

📌啟動影像串流

```
// Configure StreamIO object to stream data from video channel to RTSP
videoStreamer.registerInput(Camera.getStream(CHANNELVID));
videoStreamer.registerOutput(rtsp);
if (videoStreamer.begin() != 0) {
    Serial.println("StreamIO link start failed");
}
```

1. **取得 RTSP 影像串流來源 →** videoStreamer.registerInput(Camera.getStream(CHANNELVID));
2. **設定 RTSP 輸出 →** videoStreamer.registerOutput(rtsp);
3. videoStreamer.begin(); 啟動影像串流（若失敗，會輸出錯誤）

📌啟動 AI 影像處理

```
// Configure StreamIO object to stream data from RGB video channel to face detection
videoStreamerRGBFD.registerInput(Camera.getStream(CHANNELNN));
videoStreamerRGBFD.setStackSize();
videoStreamerRGBFD.setTaskPriority();
videoStreamerRGBFD.registerOutput(facerecog);
if (videoStreamerRGBFD.begin() != 0) {
    Serial.println("StreamIO link start failed");
}
```

- 影像通道 **CHANNELNN** 提供 AI 影像
- 將影像輸入到 facerecog 進行辨識
- 啟動影像處理流程

setup() 初始化函式

📌開啟 OSD 疊加資訊

```
// Start OSD drawing on RTSP video channel
OSD.configVideo(CHANNELVID, configVID);
OSD.begin();
```

這裡啟動 **OSD (On-Screen Display)** 疊加：

- OSD.configVideo(CHANNELVID, configVID); 設定 RTSP 影像的 **文字疊加**
- OSD.begin(); 啟動 OSD，讓影像上可顯示偵測到的資訊（如人名）

📌從 Flash 恢復已註冊的臉

```
facerecog.restoreRegisteredFace();
```

開機時會自動恢復之前註冊的臉部資訊，避免每次重新註冊！

📌設定門鎖初始狀態

```
// Servo moves to position the angle 180 degree (LOCK CLOSE)
myservo.write(180);
```

上電後，馬達預設為 180°（門鎖關閉狀態）。

setup() 初始化函式

📌開啟 OSD 疊加資訊

```
// Start OSD drawing on RTSP video channel
OSD.configVideo(CHANNELVID, configVID);
OSD.begin();
```

這裡啟動 **OSD (On-Screen Display)** 疊加：

- OSD.configVideo(CHANNELVID, configVID); 設定 RTSP 影像的 **文字疊加**
- OSD.begin(); 啟動 OSD，讓影像上可顯示偵測到的資訊（如人名）

📌從 Flash 恢復已註冊的臉

```
facerecog.restoreRegisteredFace();
```

開機時會自動恢復之前註冊的臉部資訊，避免每次重新註冊！

📌設定門鎖初始狀態

```
// Servo moves to position the angle 180 degree (LOCK CLOSE)
myservo.write(180);
```

上電後，馬達預設為 180°（門鎖關閉狀態）。

第三個區塊：loop() 主迴圈邏輯

- 偵測按鈕狀態
- 處理臉部辨識模式
- 控制門鎖（開/關）
- 管理序列監視器 (Serial Monitor) 輸入
- 更新影像處理資訊

此函式會 **不斷重複執行**，確保系統即時回應外部輸入。

loop() 函式

處理按鈕輸入、註冊與備份臉部、開關門控制

📌 3.1 讀取按鈕輸入

```
void loop()
{
    backupButtonState = digitalRead(BACKUP_FACE_BUTTON_PIN);
    RegModeButtonState = digitalRead(EN_REGMODE_BUTTON_PIN);
```

從 GPIO 讀取按鈕狀態

- backupButtonState：**是否按下「備份臉部」按鈕**
- RegModeButtonState：**是否按下「進入註冊模式」按鈕**

📝 **當按鈕被按下時，變數值會變為 HIGH (1)**

📌 3.2 處理臉部備份按鈕

```
if ((backupButtonState == HIGH) && (regMode == true)) {    // Button is pressed when registration mode is
on
        for (int count = 0; count < 3; count++) {
            digitalWrite(RED_LED, HIGH);
            digitalWrite(GREEN_LED, HIGH);
            delay(500);
            digitalWrite(RED_LED, LOW);
            digitalWrite(GREEN_LED, LOW);
            delay(500);
        }
        facerecog.backupRegisteredFace();    // 備份已註冊的臉部到 Flash
        regMode = false;                     // 關閉註冊模式
    }
```

◇ **如果註冊模式 (regMode) 為開啟狀態，且按下備份按鈕**

1. **閃爍 LED**（紅+綠）**3 次**，提醒使用者
2. 呼叫 facerecog.backupRegisteredFace(); **將註冊的臉部資料儲存到 Flash**
3. 設定 regMode = false; **關閉註冊模式**

📝 **這樣即使重啟開發板，已註冊的臉部仍然可以被辨識**

📌 3.3 處理序列監視器輸入

```
if (Serial.available() > 0) {
    String input = Serial.readString();
    input.trim();
```

◇ **如果有從電腦序列埠 (Serial Monitor) 送入指令**

1. 讀取字串 input
2. input.trim(); **移除空白字符**

📌 3.4 註冊、刪除、重設臉部

```
if (regMode == true) {
    if (input.startsWith(String("REG="))) {
        String name = input.substring(4);
        facerecog.registerFace(name);
    } else if (input.startsWith(String("DEL="))) {
        String name = input.substring(4);
        facerecog.removeFace(name);
    } else if (input.startsWith(String("RESET"))) {
        facerecog.resetRegisteredFace();
    } else if (input.startsWith(String("BACKUP"))) {
        facerecog.backupRegisteredFace();
    } else if (input.startsWith(String("RESTORE"))) {
        facerecog.restoreRegisteredFace();
    }
}
```

◇ **如果處於「註冊模式」，允許使用序列埠指令**

- **REG=名字** → 註冊新臉部
- **DEL=名字** → 刪除特定臉部
- **RESET** → 清除所有註冊臉部
- **BACKUP** → 將註冊臉部儲存到 Flash
- **RESTORE** → 從 Flash 恢復已儲存臉部

📝 **允許使用者用電腦來手動管理人臉辨識資料**

📌 3.5 切換註冊模式

```
if (regMode == false) {
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, LOW);
    if ((RegModeButtonState == HIGH)) {
        regMode = true;
        digitalWrite(RED_LED, HIGH);
        digitalWrite(GREEN_LED, HIGH);
    }
} else {
    digitalWrite(RED_LED, HIGH);
    digitalWrite(GREEN_LED, HIGH);
}
```

◇ **管理註冊模式**

- 如果 regMode **關閉**：
  - 按下註冊按鈕 (RegModeButtonState == HIGH) 時，**啟動註冊模式**
  - 開啟紅燈、綠燈，表示「註冊模式 ON」
- 如果 regMode **開啟**：
  - 持續亮起紅燈、綠燈，提醒用戶處於註冊模式

📌 3.6 控制門鎖開關

```
if ((doorOpen == true) && (regMode == false)) {
    delay(1000);
    Camera.getImage(CHANNELJPEG, &img_addr, &img_len);
    myservo.write(0);
    Serial.println("Opening Door!");

    delay(10000);
    myservo.write(180);
    digitalWrite(RED_LED, LOW);
    digitalWrite(GREEN_LED, HIGH);
    doorOpen = false;
}
```

◇ **當 doorOpen == true 且 regMode == false 時**

1. 延遲 1 秒
2. 拍攝快照 (Camera.getImage())
3. 伺服馬達轉動到 0°（開門）
4. 等待 10 秒
5. 門鎖回到 180°（鎖門）
6. 更新 LED 狀態
7. doorOpen = false;，表示門已關閉

📝 **確保門不會一直開啟，而是 10 秒後自動關閉**

📌 3.7 更新 OSD 疊加

```
    delay(2000);
    OSD.createBitmap(CHANNELVID);
    OSD.update(CHANNELVID);
}
```

◇ **每 2 秒更新一次 OSD**

- OSD.createBitmap(CHANNELVID); **建立 RTSP 影像的 OSD 疊加**
- OSD.update(CHANNELVID); **更新 OSD 顯示**

📝 **這樣，螢幕上的影像會即時更新辨識資訊**

第四個區塊：FRPostProcess() 臉部辨識後處理函式

FRPostProcess() 是一個 回調函式 (Callback Function)，當攝影機成功辨識到人臉後，這個函式會被自動呼叫來處理 開門判斷與 OSD 標記。這段程式碼的主要功能是：

1. 計算偵測到的臉部數量
2. 根據人臉辨識結果決定是否開門
3. 在 RTSP 影像上標記臉部 (OSD 疊加)
4. 控制 LED 指示燈與門鎖狀態

FRPostProcess()

處理臉部辨識結果，更新 OSD 顯示

📌 4.1 讀取辨識結果

```
void FRPostProcess(std::vector<FaceRecognitionResult> results)
{
    uint16_t im_h = configVID.height();
    uint16_t im_w = configVID.width();

    printf("Total number of faces detected = %d\r\n", facerecog.getResultCount());
    OSD.createBitmap(CHANNELVID);
```

◇ 這裡先取得 **攝影機 RTSP 影像的寬度 (im_w) 與高度 (im_h)**，用於計算人臉座標。

◇ facerecog.getResultCount(); 取得 **目前偵測到的臉部數量**，並印出結果。

📝 **這是辨識的第一步，確認攝影機是否真的偵測到人臉！**

📌 4.2 判斷是否開門

```
if (facerecog.getResultCount() > 0) {
    if (regMode == false) {
        if (facerecog.getResultCount() > 1) {     // 多於 1 張臉時不開門
            doorOpen = false;
            digitalWrite(RED_LED, HIGH);
            digitalWrite(GREEN_LED, LOW);
        } else {
            FaceRecognitionResult face = results[0];
            if (String(face.name()) == String("unknown")) {     // 偵測到陌生人時不開門
                doorOpen = false;
                digitalWrite(RED_LED, HIGH);
                digitalWrite(GREEN_LED, LOW);
            } else {     // 偵測到註冊過的臉部，開門
                doorOpen = true;
                digitalWrite(RED_LED, LOW);
                digitalWrite(GREEN_LED, HIGH);
                fileName = String(face.name()); // 記錄辨識到的名稱
            }
        }
    }
}
```

◇ 辨識條件：

1. 如果「**偵測到超過 1 張臉**」→ 不開門
2. 如果「**臉部未註冊 (unknown)**」→ 不開門
3. 如果「**辨識到註冊過的臉部**」→ 開門，並記錄名稱

📝這個邏輯可以防止陌生人闖入，也避免多張臉同時偵測時造成誤開門！

📌 4.3 在 RTSP 影像標記人臉

```
for (int i = 0; i < facerecog.getResultCount(); i++) {
    FaceRecognitionResult item = results[i];

    // 轉換人臉座標（原本是 0~1 的浮點數，轉換成實際像素值）
    int xmin = (int)(item.xMin() * im_w);
    int xmax = (int)(item.xMax() * im_w);
    int ymin = (int)(item.yMin() * im_h);
    int ymax = (int)(item.yMax() * im_h);

    uint32_t osd_color;

    // 設定 OSD 顏色（紅色代表陌生人，綠色代表已註冊的臉）
    if (String(item.name()) == String("unknown")) {
        osd_color = OSD_COLOR_RED;
    } else {
        osd_color = OSD_COLOR_GREEN;
    }
```

◇ 將 AI 模型輸出的「**相對座標**」**(0~1)** 轉換成「**實際影像像素座標**」，確保標記位置正確。

◇ **決定框線顏色**：

- **綠色 (OSD_COLOR_GREEN)** → 代表已註冊的臉部
- **紅色 (OSD_COLOR_RED)** → 代表陌生人 (unknown)

📝**這樣可以讓使用者在 RTSP 影像中即時看到人臉辨識結果！**

4.4 在影像上繪製人臉標記

```
1       printf("Face %d name %s:\t%d %d %d %d\n\r", i, item.name(), xmin, xmax, ymin, ymax);
        OSD.drawRect(CHANNELVID, xmin, ymin, xmax, ymax, 3, osd_color);

        // 在人臉框的上方顯示名稱
2       char text_str[40];
        snprintf(text_str, sizeof(text_str), "Face:%s", item.name());
        OSD.drawText(CHANNELVID, xmin, ymin - OSD.getTextHeight(CHANNELVID), text_str, osd_color);
    }
3   OSD.update(CHANNELVID);
}
```

1 標記人臉框

- 在 CHANNLEVID (RTSP 影像通道) 上繪製人臉框
- 框線顏色由 osd_color 決定 (綠色: 註冊過的臉，紅色: 陌生人)
- 框線寬度設定為 3

2 顯示臉部名稱

- 在框線 上方顯示人臉名稱
- 如果是「unknown」則顯示「Face: unknown」
- 如果是註冊過的名稱則顯示「Face: [名字]」

3 更新 OSD 顯示

- 這行確保 OSD 疊加即時更新，讓影像顯示最新的人臉資訊！

FRPostProcess() 區塊總結

- ☑ 計算偵測到的臉部數量
- ☑ 根據辨識結果決定是否開門
- ☑ 防止陌生人或多人同時出現在畫面時開門
- ☑ 在 RTSP 影像上標記人臉並顯示名稱
- ☑ 更新 OSD 疊加資訊，確保影像即時更新

YOLO 訓練與模型部署

本章節將指導學生如何使用Roboflow建立資料集與標籤，
並透過Colab darknet框架訓練YOLO模型。

訓練YOLO模型-準備步驟 1

1.收集樣本照片

- **多樣性**：確保收集的照片涵蓋不同的背景、光照條件、角度和距離。這樣可以讓模型更具泛化能力。
- **數量**：每種水果至少需要數百張照片。更多的樣本可以提高模型的準確性。
- **標籤**：每張照片中的蘋果和草莓都需要進行標註，標註的格式應該符合YOLO的要求（例如，YOLO格式的標註文件）。

2.標註樣本

- 使用標註工具（如Roboflow、LabelImg等）來標註每張照片中的蘋果和草莓。
- 標註格式應該符合YOLO的要求，通常是每個目標的邊界框（bounding box）和類別標籤。

訓練YOLO模型-準備步驟 2

1.訓練數據集

- 將標註好的圖片和標註文件整理到指定的文件夾結構中。
- 分割數據集為訓練集和驗證集，通常比例為80%訓練集和20%驗證集。

2.訓練YOLO模型

- 使用YOLO框架（如YOLOv4、YOLOv7等）進行訓練。
- 調整超參數（如學習率、批次大小等）以獲得最佳性能。
- 定期檢查模型在驗證集上的表現，避免過擬合。

3.評估和調整

- 使用混淆矩陣、精確度、召回率等指標評估模型性能。
- 根據評估結果調整模型或數據集，進行迭代訓練。

訓練YOLO模型

1.使用 YOLO（例如 YOLOv4-tiny）進行訓練，訓練完成後會產生的檔案

- **權重檔案**：這些檔案包含了訓練後的模型權重，通常會保存在你指定的目錄中。常見的權重檔案包括：
  1) **yolov4-tiny-obj_last.weights**：這是訓練過程中最後一次保存的權重檔案。
  2) **yolov4-tiny-obj_final.weights**：這是訓練完成後的最終權重檔案（如果有設定保存最終權重）。
  3) **yolov4-tiny-obj_xxxx.weights**：這些是訓練過程中每隔一定次數迭代保存的權重檔案，例如每 1000 次迭代保存一次。

- **日誌檔案**：訓練過程中的日誌檔案記錄了訓練過程中的各種訊息，包括損失值、精度等。這些日誌檔案可以幫助你分析和調試訓練過程。常見的日誌檔案包括：
  1) **train_log.txt**：這是訓練過程中的日誌檔案，記錄了每次迭代的損失值、精度等訊息。

- **配置檔案**：這些檔案通常在訓練開始前就已經存在，但在訓練過程中可能會被更新或修改：
  1) **obj.data**：這是數據配置檔案，包含了類別數量、訓練和驗證數據的路徑等訊息。
  2) **yolov4-tiny-obj.cfg**：這是模型配置檔案，定義了模型的結構和超參數。

- **結果檔案**：這些檔案記錄了訓練過程中的結果，例如每次迭代的損失值、精度等：
  1) **chart.png**：這是一個圖表檔案，顯示了訓練過程中的損失值變化趨勢。

2.評估YOLO模型，訓練完成的模型是否是一個良好的模型，可以從以下幾個方面進行評估：

- 準確率、精確率、召回率、平均精度均值、損失函數、可視化結果、推理速度

模型格式轉換與部署

1.轉換AI模型，先下載 acuity toolkit 並安裝在 Ubuntu 底下
(以下步驟皆是在 Ubuntu 底下 acuity toolkit 目錄下進行)

- **導入模型**：
  1) 編輯 0_import_model.sh 如圖1
  2) 使用 pegasus import darknet 指令
  3) 將 --model 跟 --weights 指定到訓練好的 .cfg 和 .weights 檔
  4) 修改 --channel-mean-value 修改成 0 0 0 0.00392156
  5) 執行 bash 0_import_model.sh

- **量化模型**：
  1) 編輯 1_quantize_model.sh 如圖3
  2) 執行 bash 1_quantize_model.sh

- **生成Binary**：
  1) 編輯 2_export_case_code.sh 如圖5
  2) 將 --optimize 改成 VIPNANOQI_PID0XAD確認 acuity toolkit/bin 底下有此檔案，若沒有請自行新增如圖6
  3) 執行 bash 2_export_case_code.sh

- 如上述步驟執行都沒錯誤，之後可以直接執行 bash 0_import_model.sh && bash 1_quantize_model.sh && **bash** 2_export_case_code.sh 最後可以在xxx_nbg_viplite底下找到轉換完成的xxx.nb檔。

2.模型套用：
https://www.amebaiot.com/zh/amebapro2-apply-ai-model-docs/

```
#!/bin/bash


NAME=yolov4
ACUITY_PATH=../bin/

pegasus=${ACUITY_PATH}pegasus
if [ ! -e "$pegasus" ]; then
    pegasus=${ACUITY_PATH}pegasus.py
fi

$pegasus import darknet\
    --model ./model/yolov4-tiny-custom.cfg \
    --weights ./model/yolov4-tiny-custom_last.weights \
    --output-model ${NAME}.json \
    --output-data ${NAME}.data

#generate inpumeta  --source-file dataset.txt
$pegasus generate inputmeta \
        --model ${NAME}.json \
        --input-meta-output ${NAME}_inputmeta.yml \
        --channel-mean-value "0 0 0 0.00392156"  \
        --source-file dataset.txt
```

圖 1、0_import_model 文件內容

```
#!/bin/bash

#NAME=mobilenet_tf
NAME=yolov4
ACUITY_PATH=../bin/

pegasus=$ACUITY_PATH/pegasus
if [ ! -e "$pegasus" ]; then
    pegasus=$ACUITY_PATH/pegasus.py
fi

$pegasus export ovxlib\
    --model ${NAME}.json \
    --model-data ${NAME}.data \
    --model-quantize ${NAME}.quantize \
    --with-input-meta ${NAME}_inputmeta.yml \
    --dtype quantized \
    --optimize VIPNANOQI_PID0XAD  \
    --viv-sdk ${ACUITY_PATH}vcmdtools \
    --pack-nbg-viplite

rm -rf ${NAME}_nbg_viplite

mv ../*_nbg_viplite ${NAME}_nbg_viplite

cd ${NAME}_nbg_viplite

mv network_binary.nb ${NAME}.nb

cd ..

#save normal case demo export.data
mkdir -p ${NAME}_normal_case_demo
mv  *.h *.c .project .cproject *.vcxproj BUILD *.linux *.export.data ${NAME}_normal_case_demo

# delete normal case demo source
#rm  *.h *.c .project .cproject *.vcxproj  BUILD *.linux *.export.data

rm *.data *.quantize *.json *_inputmeta.yml
```

圖 5、2_export_case_code.sh 文件

```
#!/bin/bash

#NAME=mobilenet_tf
NAME=yolov4
ACUITY_PATH=../bin/

pegasus=${ACUITY_PATH}pegasus
if [ ! -e "$pegasus" ]; then
    pegasus=${ACUITY_PATH}pegasus.py
fi

#--quantizer asymmetric_affine --qtype  uint8
#--quantizer dynamic_fixed_point  --qtype int8(int16,note s905d3 not support int16 quantize)
# --quantizer perchannel_symmetric_affine --qtype int8(int16, note only T3(0xBE) can support perchannel quantize)
$pegasus  quantize \
        --quantizer asymmetric_affine \
        --qtype uint8 \
        --rebuild \
        --with-input-meta  ${NAME}_inputmeta.yml \
        --model  ${NAME}.json \
        --model-data  ${NAME}.data
```

圖 3、1_quantize_model.sh 文件內容

```
[device]
target = VIP8000NANONI_PID0XAD
pid = 0XAD
optimization = True
core = 8
mad_per_core = 64
input_buffer_depth = 12
float16_support = False
dynamic_fixed_point-8_support = True
dynamic_fixed_point-16_support = True
asymmetric_quantized-u8_support = True
evis = 2
```

圖 6、VIPNANOQI_PID0XAD 文件