第7章 手勢辨識

本章節將基於 HT32F49395 開發板 與 Edge Impulse 平台 來進行手勢辨識。本系統結合 MPU6050 感測器，透過 I2C 介面 擷取三軸加速度，並使用 Edge Impulse 進行機器學習模型的訓練與部署。

首先，透過 Edge Impulse CLI 進行資料收集，標註不同的手勢類別（如 circle**、**fork），並使用時間序列分析與頻譜特徵提取方法來處理資料。隨後，透過 Keras分類模型 進行訓練與驗證，並調整參數以提高辨識準確度。當完成訓練後，模型將轉換為 CMSIS-Pack 格式，並部署至 HT32F49395。最後透過 Keil 開發環境，使用者可將模型匯入至裝置，撰寫推論程式碼，實現即時手勢辨識。

## MPU6050介紹

如圖7.1所示，為MPU6050示意圖。MPU 系列裝置提供全球首款整合式 6 軸動作處理器解決方案，有效消除傳統分離式設計中陀螺儀與加速度計在封裝層級的軸向錯位問題。這些裝置將 3 軸陀螺儀與 3 軸加速度計整合於同一晶片，並內建 數位動作處理器 (DMP™)，可利用成熟且專有的 MotionFusion™ 引擎處理複雜的 9 軸感測融合演算法。

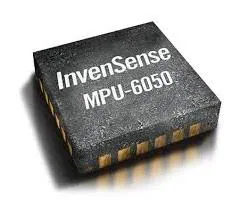


圖7.1、MPU6050示意圖

為了精確追蹤高速與低速運動，MPU-60X0 提供可程式設定的陀螺儀全量程範圍，包括 ±250、±500、±1000 及 ±2000°/秒 (dps)。此外，該元件的加速度計亦具備可程式設定的全量程範圍，涵蓋 ±2g、±4g、±8g 及 ±16g，以滿足不同應用需求。

## MPU6050暫存器位址 & I2C函式說明

* 為了讀取 MPU6050 的三軸加速度數據，HT32F49395 必須透過 I2C 介面獲取相關資料。如表7.1所列，為MPU6050暫存器位址說明表。

表7.1、MPU6050暫存器位址說明表

|  |  |  |
| --- | --- | --- |
| MPU6050 暫存器位址 | | |
| 暫存器名稱 | 位址 (Hex) | 說明 |
| WHO\_AM\_I | 0x75 | 讀取此暫存器可取得 MPU6050 的 I2C 位址 (預設 0x68 或 0x69) |
| PWR\_MGMT\_1 | 0x6B | 電源管理 1 (時脈選擇、睡眠模式等)，對此位址寫入0x00即可啟動裝置 |
| PWR\_MGMT\_2 | 0x6C | 電源管理 2 (加速度計與陀螺儀致能) |
| GYRO\_CONFIG | 0x1B | 陀螺儀靈敏度設定，寫入0x00(±250°/s)、0x08(±500°/s)、0x10(±1000°/s)、0x18(±2000°/s) |
| ACCEL\_CONFIG | 0x1C | 加速度計靈敏度設定，寫入0x00(±2g)、0x08(±4g)、0x10(±8g)、0x18(±16g) |

* 如表7.2所列，為I2C函式說明表。

表7.2、I2C函式說明表

|  |  |  |
| --- | --- | --- |
| I2C函式說明 | | |
| 1 | i2c\_status\_type i2c\_master\_transmit\_int(i2c\_handle\_type\* hi2c, uint16\_t address, uint8\_t\* pdata, uint16\_t size, uint32\_t timeout) | |
| 描述 | 使用 I2C 中斷模式進行數據傳輸的函式。這個函式會將指定的數據透過 I2C 介面傳送到目標設備，並且會基於中斷方式來處理傳輸過程。 |
| 參數 | \*hi2c：指向 I2C的指標，用於管理 I2C 設定與狀態  address：目標設備的 I2C 地址  \*pdata：需要寫入的資料  uint16\_t Size：這是要傳送的資料長度  uint32\_t timeout：超過時間沒有完成操作會返回錯誤狀態 |
| 返回值 | i2c\_status\_type |
| 2 | i2c\_status\_type i2c\_master\_receive\_int(i2c\_handle\_type\* hi2c, uint16\_t address, uint8\_t\* pdata, uint16\_t size, uint32\_t timeout) | |
| 描述 | 基於中斷模式的 I2C 接收數據函式 |
| 參數 | \*hi2c：指向 I2C的指標，用於管理 I2C 設定與狀態  address：目標設備的 I2C 地址  \*pdata：儲存接收到的資料  uint16\_t Size：要傳送的資料長度  uint32\_t timeout：超過時間沒有完成操作會返回錯誤狀態 |
| 返回值 | i2c\_status\_type |

## MPU6050連接Edge Impulse

為了要在Edge Impulse平台上取得MPU6050三軸加速度資料，需要在HT32F49395撰寫程式碼以及其他相關設定。

* 首先，撰寫main.c主程式碼。如圖7.2所示，為主程式的流程圖。

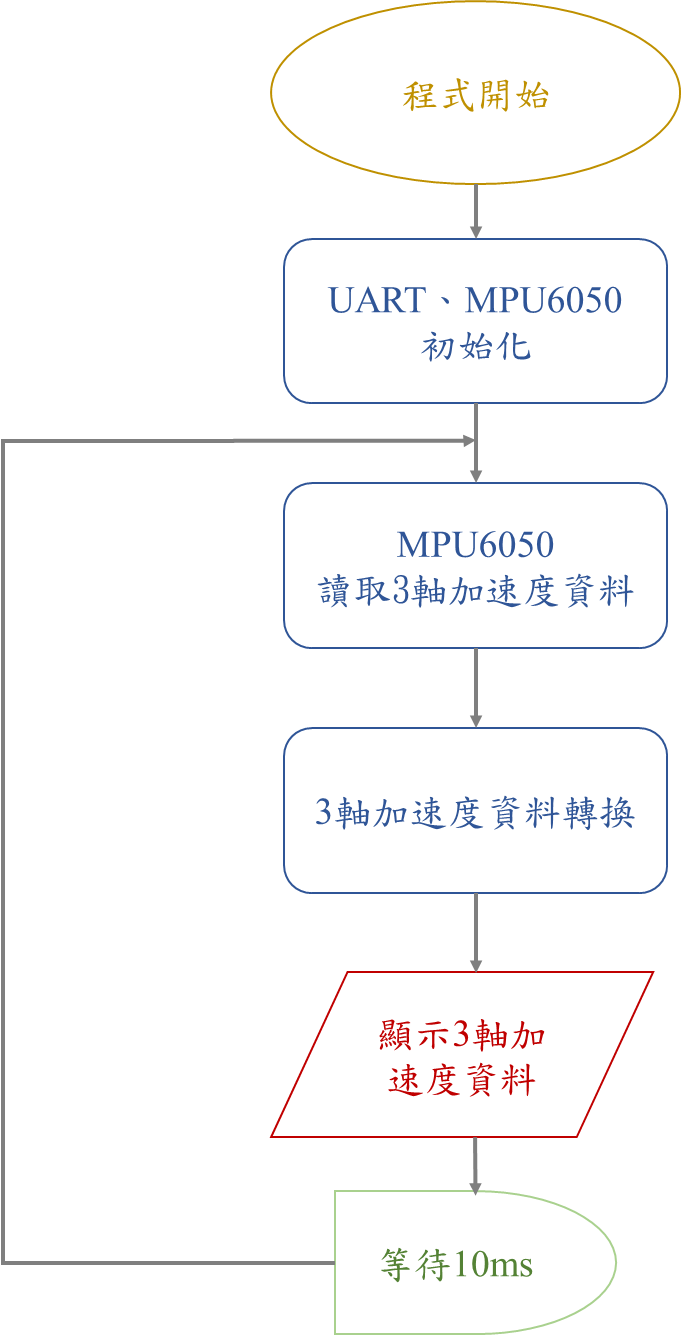


圖7.2、主程式流程圖

main.c主程式碼

|  |
| --- |
| #include "ht32f493x5\_board.h"  #include "ht32f493x5\_clock.h"  #include "i2c\_application.h"  #include "math.h"  #define I2C\_TIMEOUT 0xFFFFFFFF // I2C 超時定義  #define MPU6050\_ADDRESS 0xD0 // MPU6050 I2C 地址  #define I2Cx\_EVT\_IRQn I2C1\_EVT\_IRQn // I2C 事件中斷  #define I2Cx\_ERR\_IRQn I2C1\_ERR\_IRQn // I2C 錯誤中斷  i2c\_handle\_type hi2cx;  uint8\_t data[14]={0};  /\* MPU6050 初始化函數 \*/  void MPU6050\_Init(void)  {  //重置MPU6050  if (i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x6B, 0X80}, 2, I2C\_TIMEOUT) != I2C\_OK)  {  printf("Failed to Reset MPU6050!\n");  return;  }  delay\_ms(100); // 延遲 100ms    if (i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x6B, 0x00}, 2, I2C\_TIMEOUT) != I2C\_OK) // 唤醒 MPU6050  {  printf("Failed to wake up MPU6050!\n");  return;  }  delay\_ms(100); // 延遲 100ms  // 設置加速度計範圍 ±2g  if (i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x1C, 0x00}, 2, I2C\_TIMEOUT) != I2C\_OK)  {  printf("Failed to configure accelerometer range!\n");  return;  }  // 設置陀螺儀範圍 ±250°/s  if (i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x1B,0x00}, 2, I2C\_TIMEOUT) != I2C\_OK)  {  printf("Failed to configure gyroscope range!\n");  return;  }  printf("MPU6050 Initialization Complete.\n");  }  /\* 讀取 MPU6050 資料 \*/  void MPU6050\_ReadData(float\* accel)  {  i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x3B}, 1, I2C\_TIMEOUT);  i2c\_master\_receive\_int(&hi2cx, MPU6050\_ADDRESS, data, 14, I2C\_TIMEOUT);  accel[0]=(int16\_t)(data[0] << 8 | data[1]);  accel[1]=(int16\_t)(data[2] << 8 | data[3]);  accel[2]=(int16\_t)(data[4] << 8 | data[5]);  }  /\* 獲取數字的符號，返回 1 或 -1 \*/  float ei\_get\_sign(float number) {  return (number >= 0.0) ? 1.0 : -1.0;  }  /\* 初始化 I2C \*/  void i2c\_lowlevel\_init(i2c\_handle\_type\* hi2c)  {  gpio\_init\_type gpio\_initstructure;  if(hi2c->i2cx == I2C1)  {  crm\_periph\_clock\_enable(CRM\_I2C1\_PERIPH\_CLOCK, TRUE); //啟用I2C1  crm\_periph\_clock\_enable(CRM\_GPIOB\_PERIPH\_CLOCK, TRUE); //啟用GPIOB    // SCL 腳位設定  gpio\_initstructure.gpio\_out\_type = GPIO\_OUTPUT\_OPEN\_DRAIN;  gpio\_initstructure.gpio\_pull = GPIO\_PULL\_NONE;  gpio\_initstructure.gpio\_mode = GPIO\_MODE\_MUX;  gpio\_initstructure.gpio\_drive\_strength= GPIO\_DRIVE\_STRENGTH\_MODERATE;  gpio\_initstructure.gpio\_pins = GPIO\_PINS\_6;  gpio\_init(GPIOB, &gpio\_initstructure);  // SDA 腳位設定  gpio\_initstructure.gpio\_pins = GPIO\_PINS\_7;  gpio\_init(GPIOB, &gpio\_initstructure);    // 啟用 I2C 中斷  nvic\_irq\_enable(I2Cx\_EVT\_IRQn, 0, 0);  nvic\_irq\_enable(I2Cx\_ERR\_IRQn, 0, 0);  i2c\_init(hi2c->i2cx, I2C\_FSMODE\_DUTY\_2\_1, 100000); // 設置 I2C 工作模式和頻率  i2c\_own\_address1\_set(I2C1, I2C\_ADDRESS\_MODE\_7BIT, 0xD0); // 設置 I2C 地址  }  }  /\* 主程式 \*/  int main(void)  {  float accel[3], gyro[3];  int i;    // 初始化系統clock和板子  nvic\_priority\_group\_config(NVIC\_PRIORITY\_GROUP\_4);  system\_clock\_config();  ht32\_board\_init();  uart\_print\_init(115200); // 初始化 UART 輸出  hi2cx.i2cx = I2C1; // 設置 I2C 設備  i2c\_config(&hi2cx); // 配置 I2C  MPU6050\_Init(); // 初始化 MPU6050  printf("Accel ");  while (1)  {  MPU6050\_ReadData(accel);// 讀取加速度計數據  // 將加速度數據轉換為 g 單位  accel[0]=accel[0]\*4.0/32678.0;  accel[1]=accel[1]\*4.0/32678.0;  accel[2]=accel[2]\*4.0/32678.0;    // 限制加速度數據在 ±2g 範圍內  for ( i = 0; i < 3; i++) {  if (fabs(accel[i]) > 2.0 ){  accel[i] = ei\_get\_sign(accel[i]) \*2.0;  }  }  // 將加速度數據轉換為 m/s²  accel[0] \*= 9.80665;  accel[1] \*= 9.80665;  accel[2] \*= 9.80665;  //顯示讀取到的加速度資料  printf("%f",accel[0]);  printf("\t");  printf("%f",accel[1]);  printf("\t");  printf("%f\n",accel[2]);  delay\_ms(10);  }  } |

* 於ht32f493x5\_int.c程式中撰寫以下程式碼。最後將程式編譯後，燒錄至開發版上。

ht32f493x5\_int.c程式碼

|  |
| --- |
| #include "ht32f493x5\_int.h"  #include "ht32f493x5\_board.h"  #include "i2c\_application.h"  extern i2c\_handle\_type hi2cx; //指向 I2C的指標，用來操作 I2C 模組  #define I2Cx\_EVT\_IRQHandler I2C1\_EVT\_IRQHandler // I2C 事件中斷處理函式定義  #define I2Cx\_ERR\_IRQHandler I2C1\_ERR\_IRQHandler // I2C 錯誤中斷處理函式定義  \_\_IO uint32\_t uwTick = 0; // 用於記錄計數值  void NMI\_Handler(void)  {  }  /\*\*  \* @brief this function handles hard fault exception.  \* @param none  \* @retval none  \*/  void HardFault\_Handler(void)  {  /\* go to infinite loop when hard fault exception occurs \*/  while(1)  {  }  }  /\*\*  \* @brief this function handles memory manage exception.  \* @param none  \* @retval none  \*/  void MemManage\_Handler(void)  {  /\* go to infinite loop when memory manage exception occurs \*/  while(1)  {  }  }  /\*\*  \* @brief this function handles bus fault exception.  \* @param none  \* @retval none  \*/  void BusFault\_Handler(void)  {  /\* go to infinite loop when bus fault exception occurs \*/  while(1)  {  }  }  /\*\*  \* @brief this function handles usage fault exception.  \* @param none  \* @retval none  \*/  void UsageFault\_Handler(void)  {  /\* go to infinite loop when usage fault exception occurs \*/  while(1)  {  }  }  /\*\*  \* @brief this function handles svcall exception.  \* @param none  \* @retval none  \*/  void SVC\_Handler(void)  {  }  /\*\*  \* @brief this function handles debug monitor exception.  \* @param none  \* @retval none  \*/  void DebugMon\_Handler(void)  {  }  /\*\*  \* @brief this function handles pendsv\_handler exception.  \* @param none  \* @retval none  \*/  void PendSV\_Handler(void)  {  }  /\*\*  \* @brief this function handles systick handler.  \* @param none  \* @retval none  \*/  void SysTick\_Handler(void)  {  uwTick++; // 每當系統時鐘中斷時，計數器加 1  ht32\_led\_toggle(LED2); // 切換 LED2 的狀態 (用於顯示計數)    }  uint32\_t SysTick\_GetTick(void)  {  return uwTick; // 返回當前滴答計數值  }  /\*\*  \* @brief this function handles i2c event interrupt request.  \* @param none  \* @retval none  \*/  void I2Cx\_EVT\_IRQHandler(void)  {  i2c\_evt\_irq\_handler(&hi2cx); // 處理 I2C 事件中斷，並調用事件處理函式  }  /\*\*  \* @brief this function handles i2c error interrupt request.  \* @param none  \* @retval none  \*/  void I2Cx\_ERR\_IRQHandler(void)  {  i2c\_err\_irq\_handler(&hi2cx); // 處理 I2C 錯誤中斷，並調用錯誤處理函式  } |

* 使用Edge Impulse CLI-Data forwarder，在CMD輸入以下指令

edge-impulse-data-forwarder

* 此時，Data forwarder會詢問使用者要連接的伺服器，提示使用者登錄，然後設定設備。如圖7.3所示，為Data forwarder輸出範例圖。



圖7.3、Data forwarder輸出範例圖

* 若要清除配置，請執行：edge-impulse-data-forwarder --clean

若要覆蓋頻率，請使用：edge-impulse-data-forwarder --frequency 100

若要設定不同的波特率(460800可以自行變更欲要設定的值)，請使用：

edge-impulse-data-forwarder --baud-rate 460800

## 7.4 在Edge Impulse建置手勢辨識模型

以下，為讀者說明在Edge Impulse建置手勢辨識模型的主要步驟。如圖7.4所示，為Edge Impulse建置手勢辨識模型流程圖。

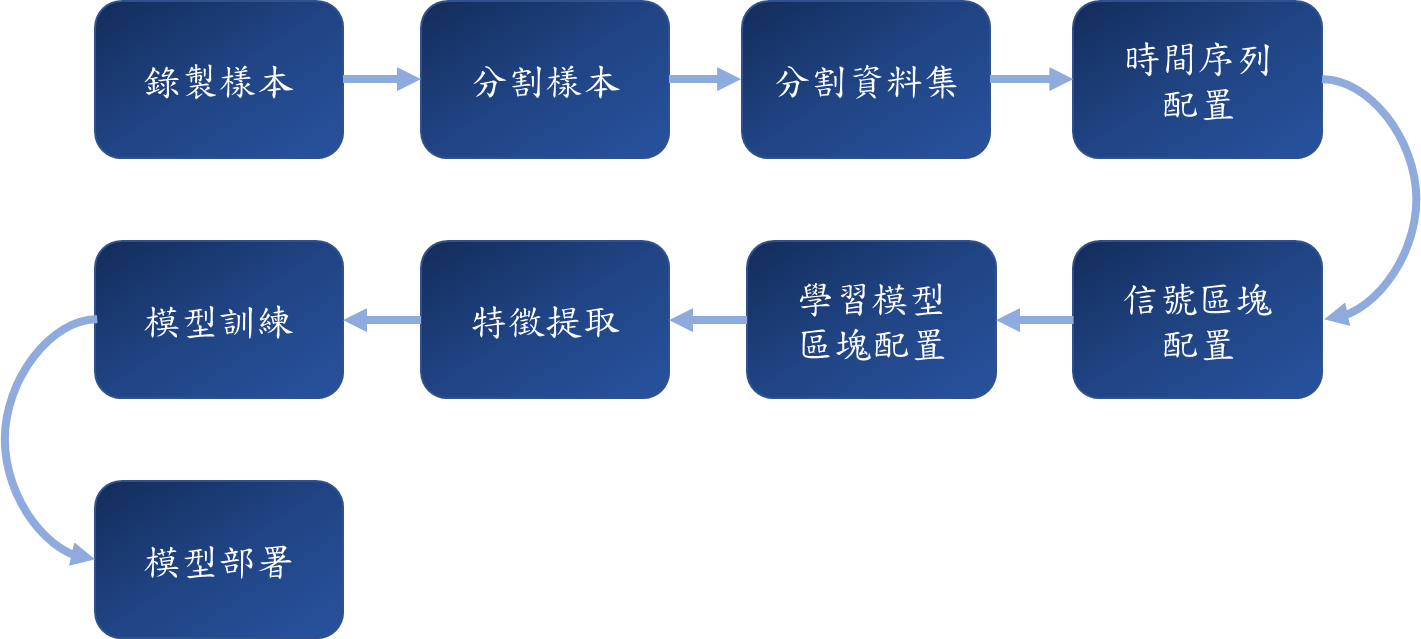


圖7.4、Edge Impulse建置手勢辨識模型流程圖

* 如圖7.5所示為，收集及建置資料集圖。其中，資料擷取-使用者自行製作資料集，點選Data acquisition，分別在Label建置circle(畫圓圈)、fork(畫叉) ，Sensor設定為三軸，以及取樣頻率100Hz。在每個Label分別建立所需的樣本集，本範例是每個Label約有40筆樣本，且每筆樣本時長為2秒。但需注意到，需盡量將每筆樣本時長一致，避免在測試時出現誤差。最後，點選Start sample，開始錄製樣本。

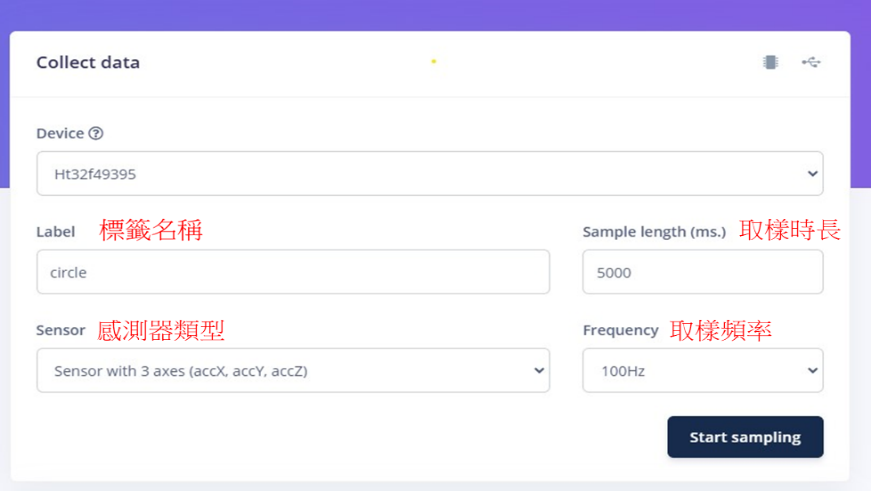


圖7.5、收集及建置資料集圖

* 如圖7.6所示，為分割樣本示意圖。其中，待錄製樣本時間結束，先點選箭頭處，在點擊Split sample分割樣本

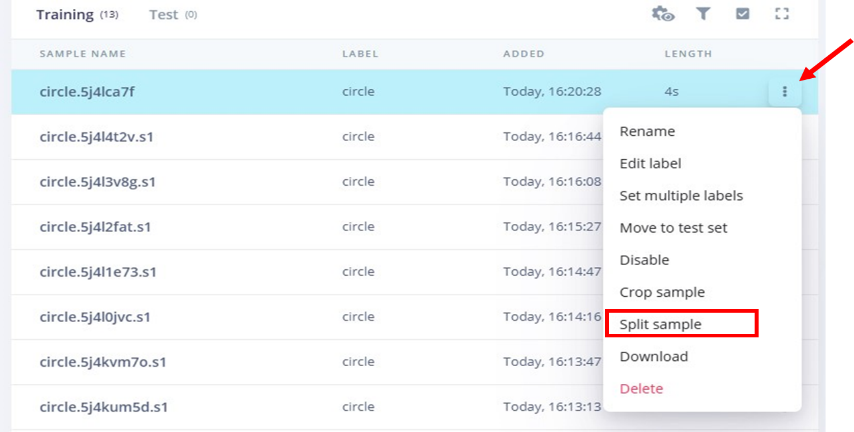


圖7.6、分割樣本示意圖

* 如圖7.7所示，為分割樣本功能介紹示意圖。其中，分割樣本，按照所需進行分割，最後再按下”Split”按鈕。



圖7.7、分割樣本功能介紹示意圖

* 如圖7.8所示，為已分割好樣本圖。其中，顯示了本範例分割好的樣本。

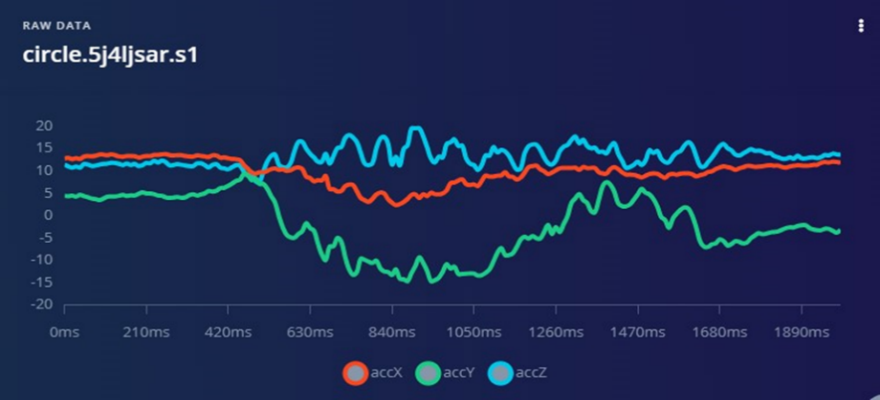


圖7.8、已分割好樣本圖

* 如圖7.9所示，為分割資料集說明圖。當收集完足夠的資料後，從Dashboard下的Danger zone中，可以找到Perform train/test split選項。點擊即可拆分訓練和測試資料集。在此，我們設定訓練集80%測試集20%。

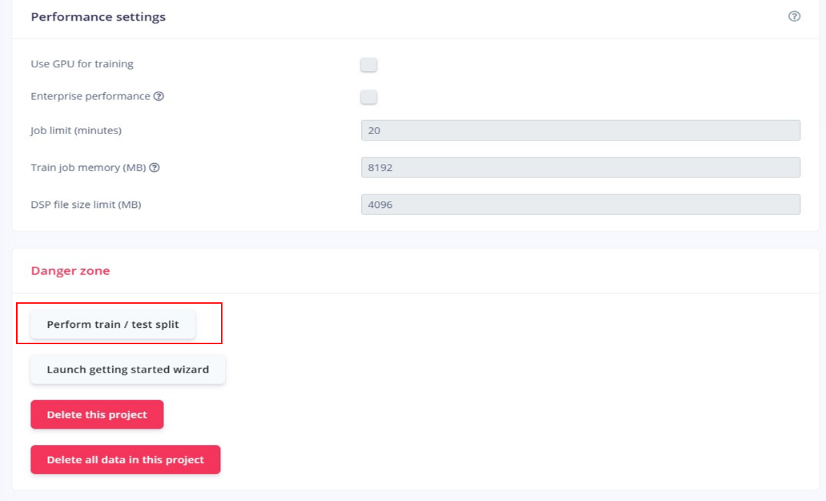


圖7.9、分割資料集說明圖

* 如圖7.10所示，為設定設備示意圖。其中，待所有Label的樣本集蒐集完畢後，點擊左側選單中的”Create impulse”，建立impulse。在該頁面會出現如下圖所示(用預設即可)

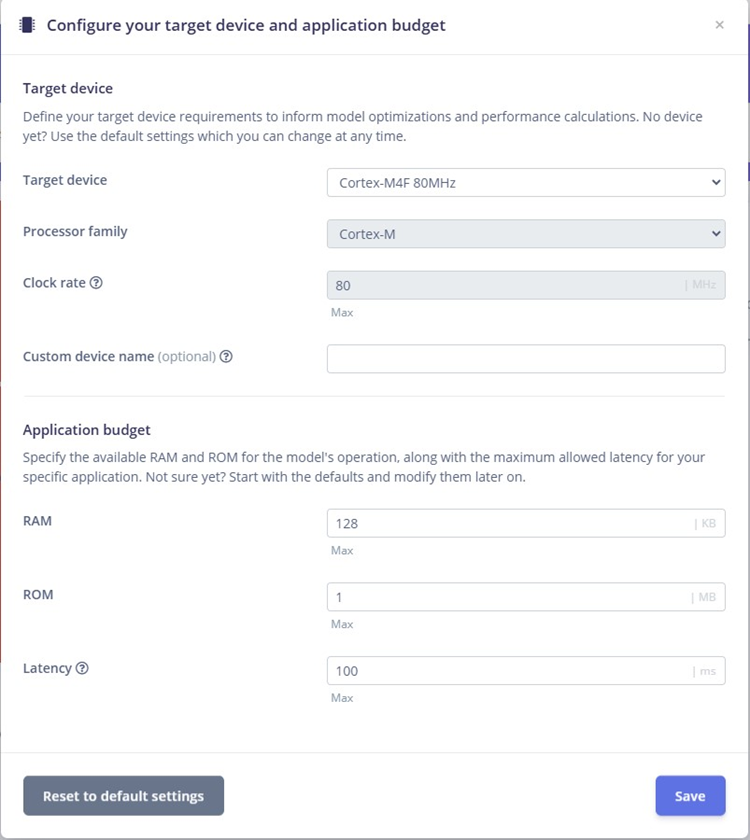


圖7.10、設定設備示意圖

* 如圖7.11所示，為時間序列配置示意圖。在這個區塊主要設定時間序列資料處理的基本參數，包括視窗大小、視窗移動遞增，取樣頻率及資料不足視窗大小時是否補零（Zero-pad data,預設勾選，會自動補零）。在此範例中，Window size為2000ms。

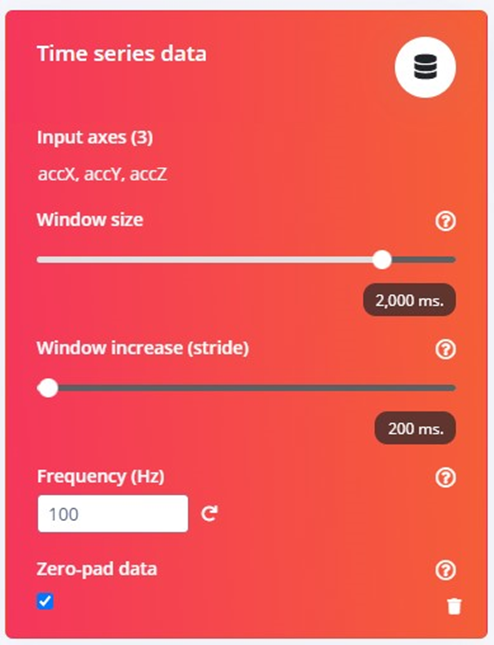


圖7.11、時間序列配置示意圖

* 如圖7.12所示，為信號區塊配置示意圖。使用者點擊閃電符號後，會新增信號處理區塊(Add a processing block)，即一小段信號前處理程式。此時，新增一個「處理區塊(Processing Block)」，並選擇系統推薦的「頻譜分析(Spectral Analysis)」即可。

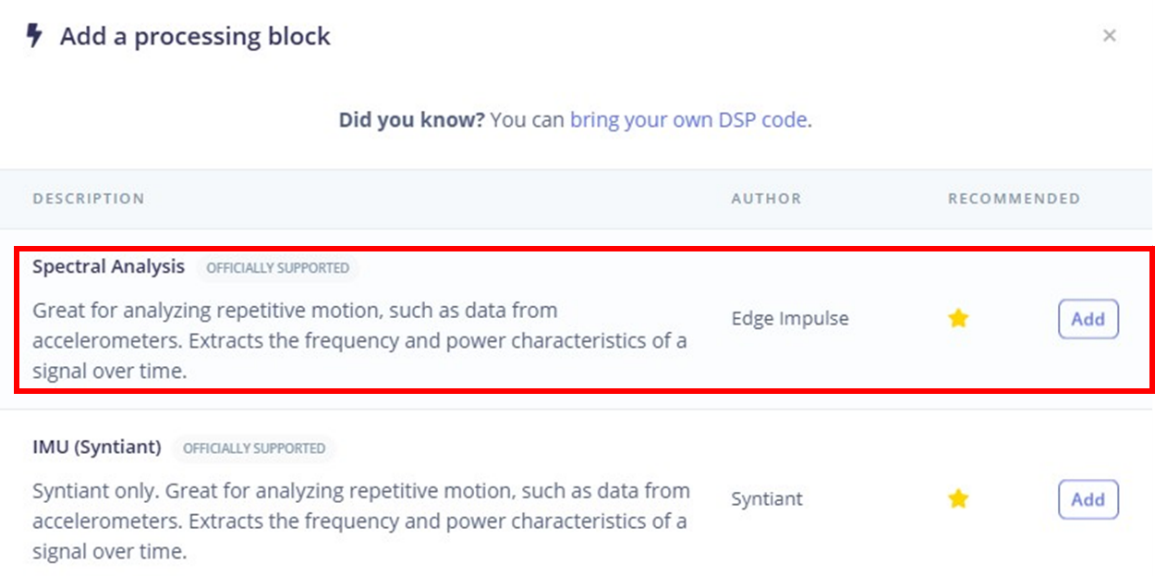


圖7.12、信號區塊配置示意圖

* 如圖7.13所示，為Spectral Analysis配置示意圖。其中，選擇accX，accY以及 accZ選項。

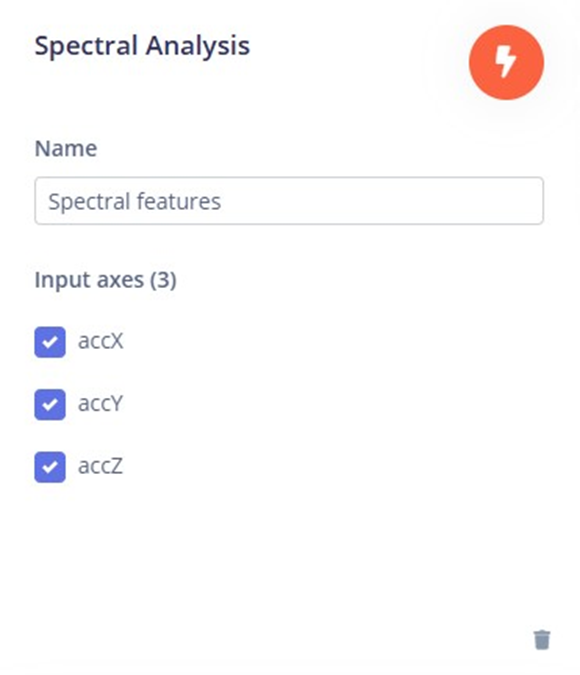


圖7.13、Spectral Analysis配置示意圖

* 如圖7.14所示，為學習模型區塊配置示意圖。此時，使用者新增一個「學習區塊(Learning Block)」，選擇系統推薦的「分類Classification(Keras)」即可。

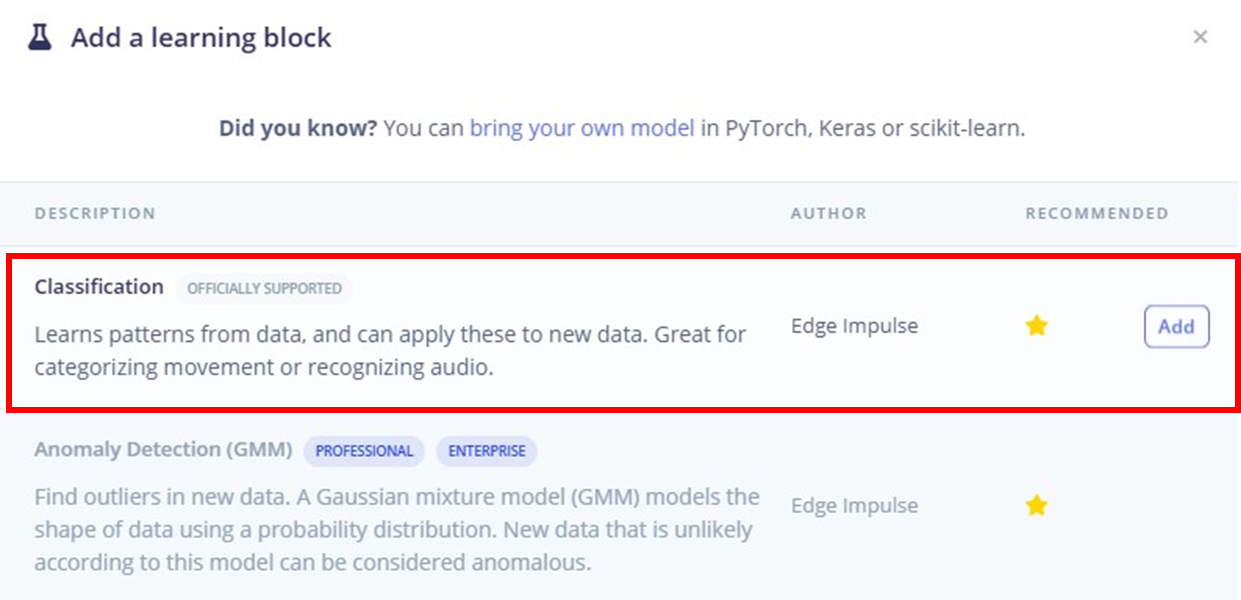


圖7.14、學習模型區塊配置示意圖

* 如圖7.15所示，為儲存feature示意圖。此時，按下”Save Impulse”按鈕即可

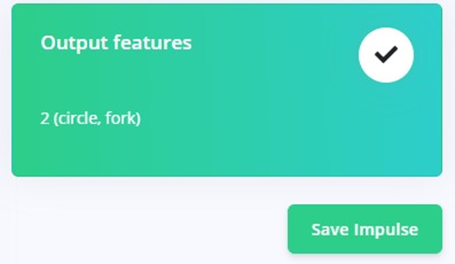


圖7.15、儲存feature示意圖

* 如圖7.16所示，為頻譜特徵(Spectral Features)說明示意圖。此時，使用者切換到「頻譜特徵(Spectral Features)」頁面，第一子頁為「參數（Parameters)」。並可以在右上角下拉式選單中切換想觀看的資料。頁面就會顯示對應的原始三軸信號波形。若沒有特殊需求，可先略過調整頻譜相關參數。



圖7.16、頻譜特徵(Spectral Features)說明示意圖

* 如圖7.17所示，為特徵提取示意圖。此時，使用者切換到第二子頁「產生特徵(Generate Features)」按下”Generate Features”按鈕，開始計算所有訓練樣本的特徵值，並以立體空間可視化結果來呈現資料分佈狀態。如果各分類自己本身（相同顏色點）越群聚，且不同分類越分開，表示資料很容易分割，可以得到較好的分類效果。反之，則要重新處理資料集收集及建置問題。

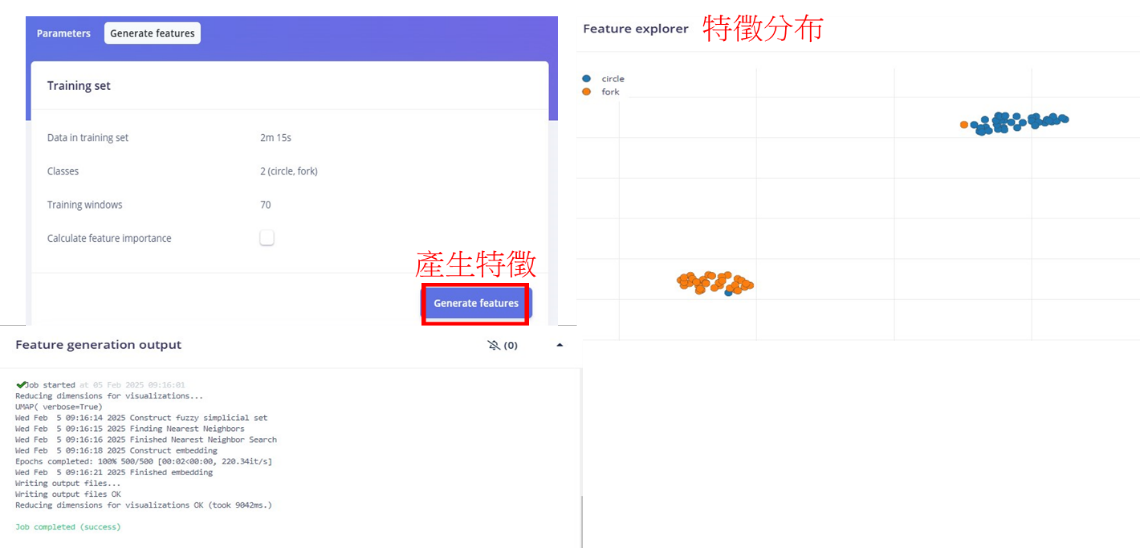


圖7.17、特徵提取示意圖

* 如圖7.18所示，為模型訓練與神經網路設定示意圖。此時，使用者點選”Classifier”，是用來設計模型結構及進行訓練用的。如果訓練後精確度不高，混淆矩陣表現不佳，那就可以增加次數或改變模型結構來改善。使用者按下”Start &Train”按鈕後就能得到精確度、損失值、特徵分佈圖及佈署到裝置後的資源耗損。

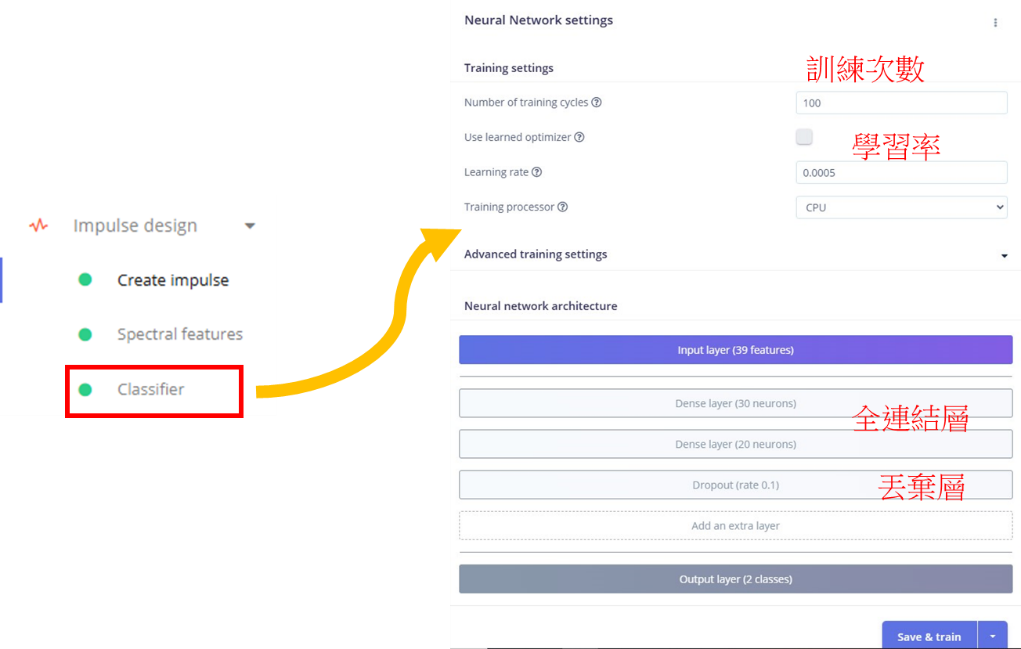


圖7.18、模型訓練與神經網路設定示意圖

* 如圖7.19所示，為模型訓練結果示意圖。由於本範例使用較少的資料樣本，所以訓練後得到100%的精準度是很正常的（因為已經過擬合了）。如果你的訓練集很大，但卻得到很高的精確度和很完美的混淆矩陣，那可能已產生過擬合現象了。



圖7.19、模型訓練結果示意圖

* 如圖7.20所示，則為模型部署步驟一示意圖，此時，點選”Deployment”選項。

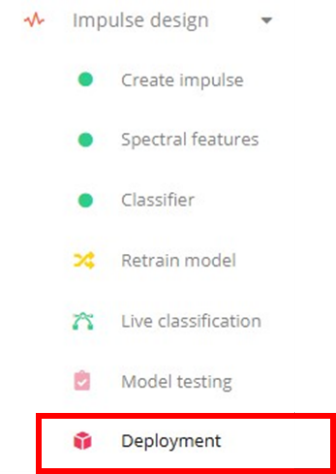


圖7.20、模型部署步驟一示意圖

* 如圖7.21所示，為模型部署步驟二示意圖。並此時選”Open CMSIS pack”選項。

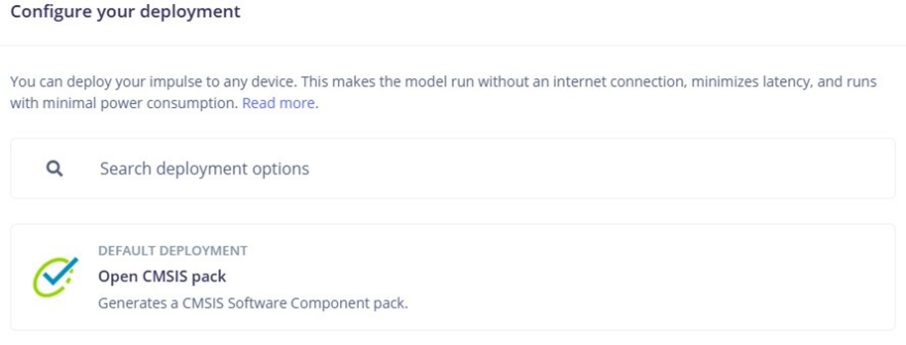


圖7.21、模型部署步驟二示意圖

* 如圖7.22所示，為模型部署步驟三示意圖。此時按下”Build”按鈕，就會產生模型壓縮檔。

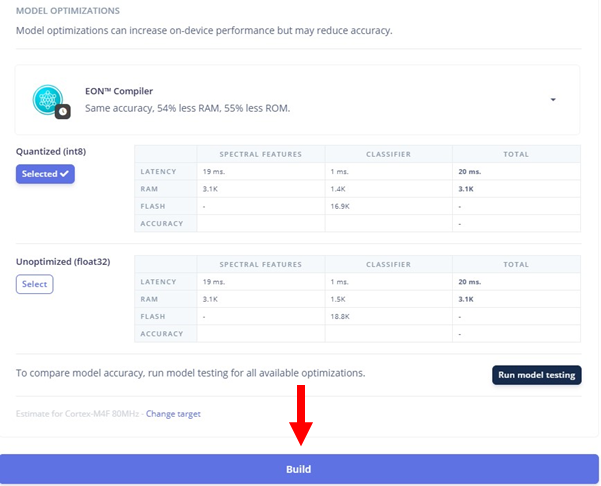


圖7.22、模型部署步驟三示意圖

## 將模型佈署到HT32F49395微控制器中

有了前章所產生的模型壓縮檔，緊接著我們將此模型壓縮檔佈署到HT32F49395。

* 如圖7.23所示，為匯入模型檔案的前置設定示意圖。其中，先將載好的壓縮檔解壓縮，接著至Keil C點選”Pack installer”。然後，點擊File，再按下”Import”選項。

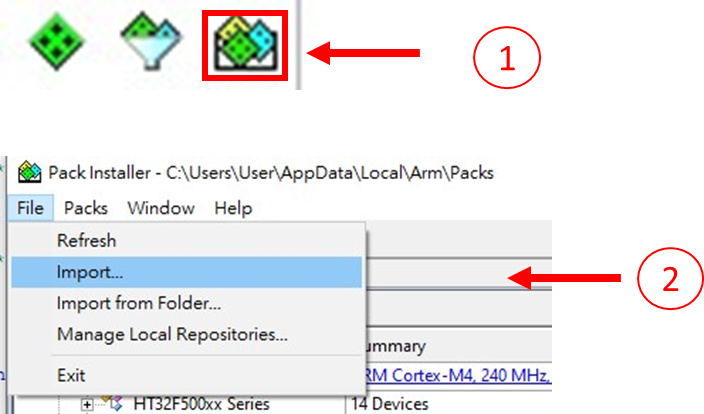


圖7.23、前置設定示意圖

* 如圖7.24所示，為匯入AI模型檔案示意圖。此時，將解完壓縮的模型檔案會匯入進去。

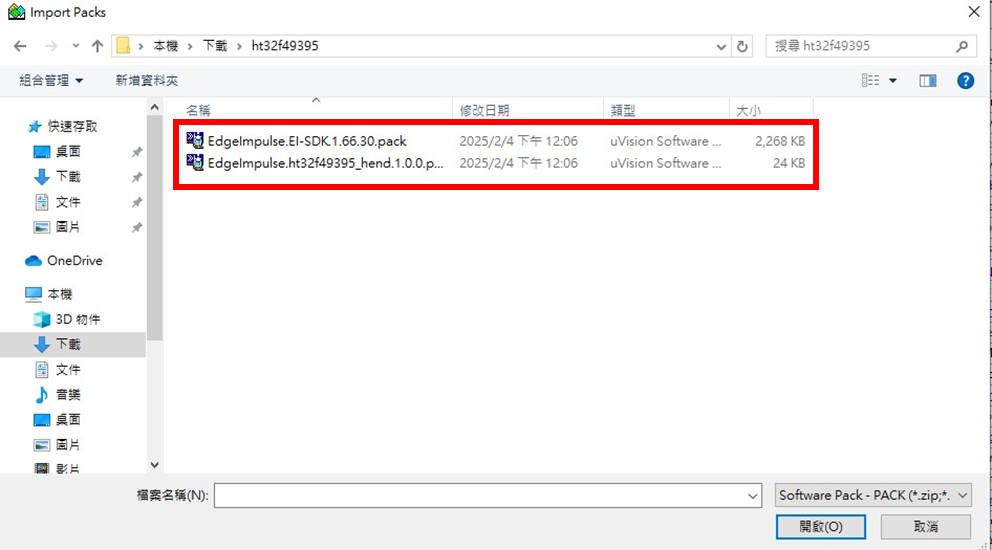


圖7.24、匯入AI模型檔案示意圖

* 緊接著，讀者需記好EdgeImpulse.EI-SDK檔案的版本。如圖7.25所示，點擊”Select Software Packs”選項，並在EdgeImpulse::EI-SDK勾選該版本號。最後，至EdgeImpulse::你的專案名稱設定為latest，並按下”OK”按鈕。

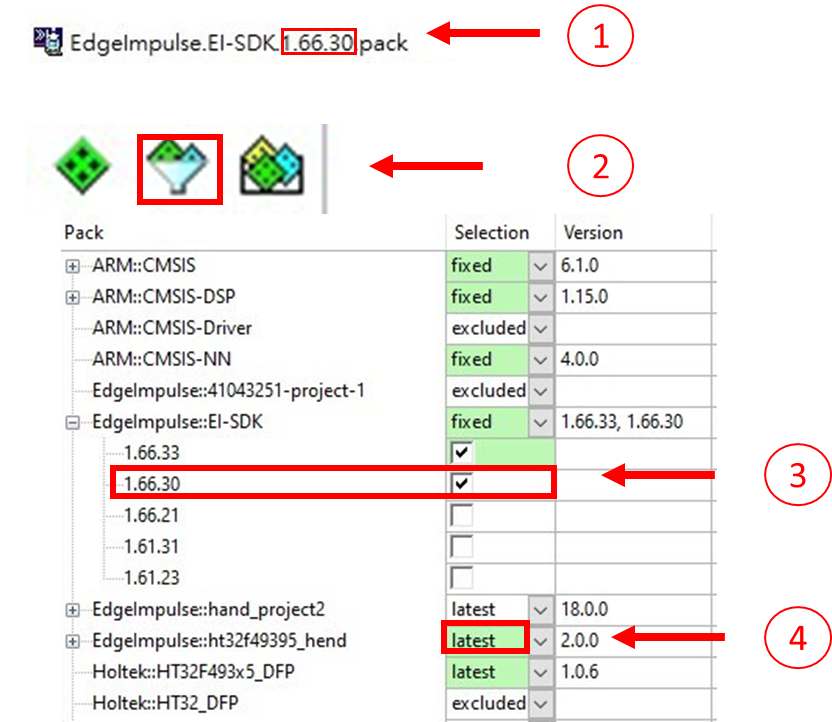


圖7.25、更改EI-SDK版本示意圖

* 最後，如圖7.26所示，為選擇模型示意圖。其中，點選Manage Run-time Environment頁面，並接著點擊EdgeImpulse -> model，並勾選你的專案名稱，再按下”OK”按鈕。

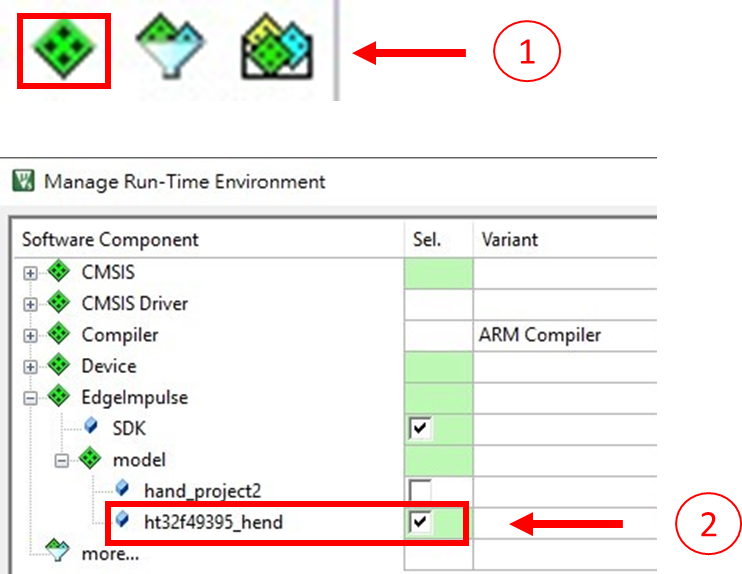


圖7.26、選擇模型示意圖

## 撰寫AI推論程式碼

有了上述將模型佈署到HT32F49395微控制器上後，接著就是在Keil撰寫AI推論程式碼。

* + 1. main.c撰寫

如圖7.27所示，為AI推論程式流程圖。

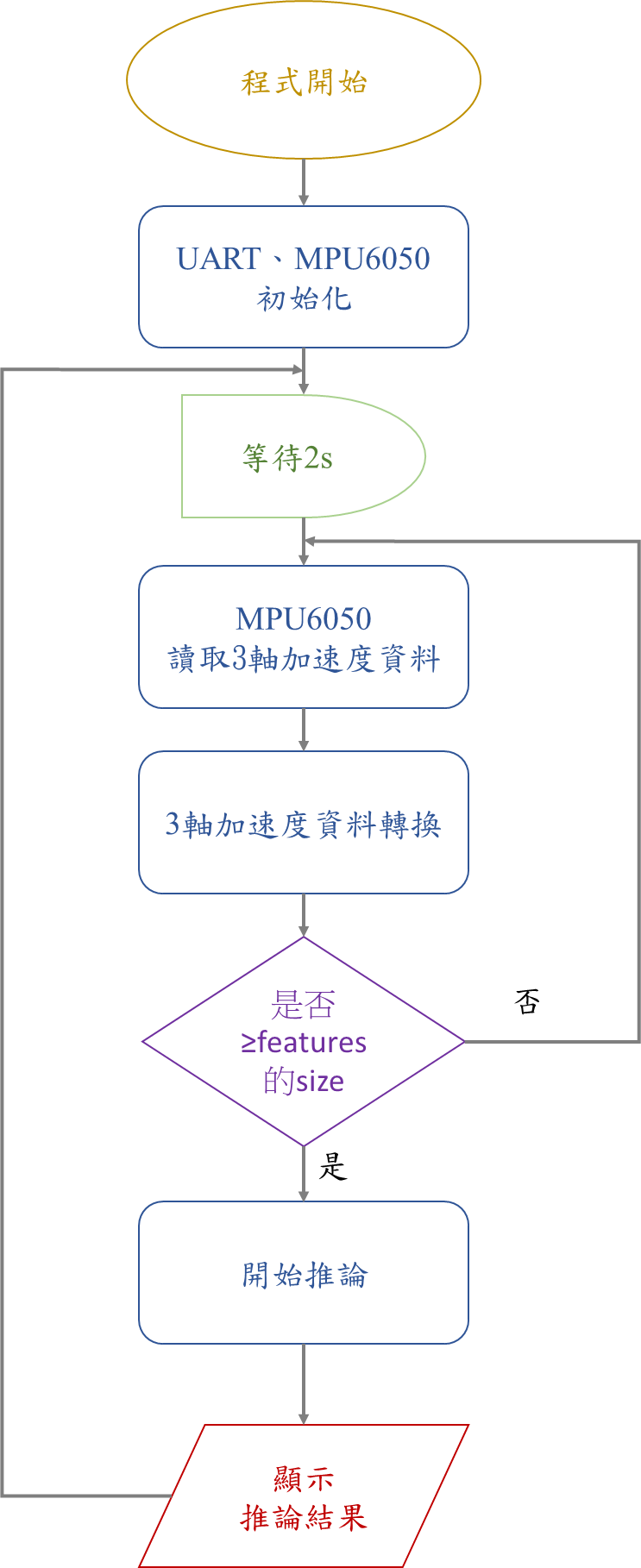


圖7.27、AI推論程式流程圖

配置（包含檔案和定義）

|  |
| --- |
| #include "ht32f493x5\_board.h"  #include "ht32f493x5\_clock.h"  #include "i2c\_application.h"  #define I2C\_TIMEOUT 0xFFFFFFFF // I2C超時設定  #define MPU6050\_ADDRESS 0xD0 // MPU6050 I2C 地址  #define I2Cx\_EVT\_IRQn I2C1\_EVT\_IRQn // I2C事件中斷  #define I2Cx\_ERR\_IRQn I2C1\_ERR\_IRQn // I2C錯誤中斷  i2c\_handle\_type hi2cx; // 用來處理I2C通訊  uint8\_t data[14]={0};  void i2c\_lowlevel\_init(i2c\_handle\_type\* hi2c);  extern int ei\_main(void); //告訴編譯器 ei\_main 函式會在其他檔案中定義 |

初始化MPU6050感測器

|  |
| --- |
| /\* MPU6050 初始化函數 \*/  void MPU6050\_Init(void)  {  uint8\_t data;  //重置MPU6050  if (i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x6B, 0X80}, 2, I2C\_TIMEOUT) != I2C\_OK)  {  printf("Failed to Reset MPU6050!\n");  return;  }  delay\_ms(100); // 延遲 100ms    data = 0x00; // 清除 SLEEP 位元  if (i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x6B, data}, 2, I2C\_TIMEOUT) != I2C\_OK) // 唤醒 MPU6050  {  printf("Failed to wake up MPU6050!\n"); // 如果寫入失敗，顯示錯誤訊息  return;  }  delay\_ms(100); // 延遲 100ms  // 設置加速度計範圍  data = 0x00; // ±2g  if (i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x1C, data}, 2, I2C\_TIMEOUT) != I2C\_OK)  {  printf("Failed to configure accelerometer range!\n"); // 設置加速度計失敗  return;  }  // 設置陀螺儀範圍  data = 0x00; // ±250°/s  if (i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x1B, data}, 2, I2C\_TIMEOUT) != I2C\_OK)  {  printf("Failed to configure gyroscope range!\n"); // 設置陀螺儀失敗  return;  }  printf("MPU6050 Initialization Complete.\n"); // 完成初始化  } |

初始化I2C

|  |
| --- |
| /\* 初始化 I2C \*/  void i2c\_lowlevel\_init(i2c\_handle\_type\* hi2c)  {  gpio\_init\_type gpio\_initstructure;  if(hi2c->i2cx == I2C1)  {  crm\_periph\_clock\_enable(CRM\_I2C1\_PERIPH\_CLOCK, TRUE); // 啟用I2C1時鐘  crm\_periph\_clock\_enable(CRM\_GPIOB\_PERIPH\_CLOCK, TRUE); // 啟用GPIOB時鐘    // SCL 腳位設定  gpio\_initstructure.gpio\_out\_type = GPIO\_OUTPUT\_OPEN\_DRAIN;  gpio\_initstructure.gpio\_pull = GPIO\_PULL\_NONE;  gpio\_initstructure.gpio\_mode = GPIO\_MODE\_MUX;  gpio\_initstructure.gpio\_drive\_strength= GPIO\_DRIVE\_STRENGTH\_MODERATE;  gpio\_initstructure.gpio\_pins = GPIO\_PINS\_6;  gpio\_init(GPIOB, &gpio\_initstructure);  // SDA 腳位設定  gpio\_initstructure.gpio\_pins = GPIO\_PINS\_7;  gpio\_init(GPIOB, &gpio\_initstructure);    nvic\_irq\_enable(I2Cx\_EVT\_IRQn, 0, 0); // 啟用I2C事件中斷  nvic\_irq\_enable(I2Cx\_ERR\_IRQn, 0, 0); // 啟用I2C錯誤中斷  i2c\_init(hi2c->i2cx, I2C\_FSMODE\_DUTY\_2\_1, 100000); // 設置 I2C 工作模式和頻率  i2c\_own\_address1\_set(I2C1, I2C\_ADDRESS\_MODE\_7BIT, 0x68); // 設定I2C從設備地址  }  } |

主程式：main

|  |
| --- |
| int main(void)  {  nvic\_priority\_group\_config(NVIC\_PRIORITY\_GROUP\_4); //設定中斷優先順序  system\_clock\_config(); // 配置系統時鐘  ht32\_board\_init();  uart\_print\_init(115200); // 初始化UART，設定波特率為115200  hi2cx.i2cx = I2C1; // 設定I2C1  i2c\_config(&hi2cx); // 配置I2C  MPU6050\_Init(); // 初始化MPU6050  ei\_main(); // 執行外部庫中的主函式  while(1)  {  }  } |

7.6.2 ht32f493x5\_int.c程式碼撰寫

以下是針對ht32f493x5\_int.c有修改與新增程式碼部份

|  |
| --- |
| #include "ht32f493x5\_int.h"  #include "ht32f493x5\_board.h"  #include "i2c\_application.h"  extern i2c\_handle\_type hi2cx; //指向 I2C的指標  #define I2Cx\_EVT\_IRQHandler I2C1\_EVT\_IRQHandler // I2C 事件中斷處理函式定義  #define I2Cx\_ERR\_IRQHandler I2C1\_ERR\_IRQHandler // I2C 錯誤中斷處理函式定義  \_\_IO uint32\_t uwTick = 0; // 用於記錄計數值 |

修改SysTick\_Handler與SysTick\_GetTick函式

|  |
| --- |
| void SysTick\_Handler(void)  {  uwTick++; // 每當系統時鐘中斷時，計數器加 1  ht32\_led\_toggle(LED2); // 切換 LED2 的狀態 (用於顯示計數)    }  uint32\_t SysTick\_GetTick(void)  {  return uwTick; // 返回當前滴答計數值  } |

新增以下函式

|  |
| --- |
| /\*\*  \* @brief this function handles i2c event interrupt request.  \* @param none  \* @retval none  \*/  void I2Cx\_EVT\_IRQHandler(void)  {  i2c\_evt\_irq\_handler(&hi2cx); // 處理 I2C 事件中斷，並調用事件處理函式  }  /\*\*  \* @brief this function handles i2c error interrupt request.  \* @param none  \* @retval none  \*/  void I2Cx\_ERR\_IRQHandler(void)  {  i2c\_err\_irq\_handler(&hi2cx); // 處理 I2C 錯誤中斷，並調用錯誤處理函式  } |

7.6.3建立ei\_main.h檔案

以下為ei\_main.h程式碼

|  |
| --- |
| #ifndef \_EI\_MAIN\_H\_  #define \_EI\_MAIN\_H\_  extern "C" int ei\_main(void); //以 C 方式編譯，避免名稱不匹配的錯誤  #endif |

7.6.4建立ei\_main.cpp檔案

以下為ei\_main.cpp程式碼

|  |
| --- |
| #include "ei\_main.h" // 引入 Edge Impulse 的主要頭檔  #include "ht32f493x5\_board.h"  #include "ht32f493x5\_clock.h"  #include "ht32f493x5\_conf.h"  #include <string.h>  #include "ei\_run\_classifier.h" // 引入 Edge Impulse 的classifier程式  #include "edge-impulse-sdk/porting/ei\_classifier\_porting.h" // 引入 Edge Impulse 的 SDK 端口檔  #include "i2c\_application.h"  #include "math.h"  #define I2C\_TIMEOUT 0xFFFFFFFF // 定義 I2C 時間超時  #define MPU6050\_ADDRESS 0xD0 // MPU6050 I2C 地址  extern uint8\_t data[14]; // 用於儲存從 MPU6050 接收到的數據  extern i2c\_handle\_type hi2cx;  float features[EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE]{0}; // 定義特徵陣列，存從 MPU6050 獲得的數據  /\* 讀取 MPU6050 資料 \*/  void MPU6050\_ReadData(float \*features, int i)  {  // 向 MPU6050 發送請求讀取數據  i2c\_master\_transmit\_int(&hi2cx, MPU6050\_ADDRESS, (uint8\_t[]){0x3B}, 1, I2C\_TIMEOUT);    // 從 MPU6050 讀取 14 字節數據  i2c\_master\_receive\_int(&hi2cx, MPU6050\_ADDRESS, data, 14, I2C\_TIMEOUT);  // 將數據存儲到 features 中  features[i]=(int16\_t)(data[0] << 8 | data[1]);  features[i+1]=(int16\_t)(data[2] << 8 | data[3]);  features[i+2]=(int16\_t)(data[4] << 8 | data[5]);  }  /\* 函數：根據數字的符號返回 1 或 -1 \*/  float ei\_get\_sign(float number) {  return (number >= 0.0) ? 1.0 : -1.0;  }  /\* 加速度數據處理 \*/  void accel(){    // 每次讀取三個加速度數據  for(int i=0;i<EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE;i+=3){  MPU6050\_ReadData(features,i); // 從 MPU6050 讀取數據    // 將原始數據轉換為加速度（單位：g）  features[i]=features[i]\*4.0/32678.0;  features[i+1]=features[i+1]\*4.0/32678.0;  features[i+2]=features[i+2]\*4.0/32678.0;    // 若加速度值大於 2g，則進行限制  for (int j = 0; j < 3; j++) {  if (fabs(features[i+j]) > 2.0 ){  features[i+j] = ei\_get\_sign(features[i+j]) \*2.0; // 限制最大值為 2g  }  }  // 將加速度轉換為米每秒平方（m/s^2）  features[i+0] \*= 9.80665;  features[i+1] \*= 9.80665;  features[i+2] \*= 9.80665;  delay\_ms(10);  }  }  /\* 從 features 中獲取數據 \*/  int raw\_feature\_get\_data(size\_t offset, size\_t length, float \*out\_ptr)  {  memcpy(out\_ptr, features + offset, length \* sizeof(float)); // 複製特徵數據  return 0;  }  /\* 主程式 \*/  int ei\_main(void)  {  ei\_impulse\_result\_t result = {nullptr}; // 儲存分類結果  while (true) {  ei\_printf("Edge Impulse standalone inferencing (ht32f493x5)\n");  // 檢查 features 的大小是否與預期的一致  if (sizeof(features) / sizeof(float) != EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE) {  ei\_printf("The size of your 'features' array is not correct. Expected %d items, but had %u\n",EI\_CLASSIFIER\_DSP\_INPUT\_FRAME\_SIZE, sizeof(features) / sizeof(float));  return 1;  }    ei\_printf("Entering the loop!\n");  while (1) {  // 等待 2 秒開始推理  ei\_printf("\nStarting inferencing in 2 seconds...\n");  delay\_ms(2000);  ei\_printf("Sampling...\n");    accel(); // 讀取並處理加速度數據  printf("success");  signal\_t features\_signal;  features\_signal.total\_length = sizeof(features) / sizeof(features[0]);  features\_signal.get\_data = &raw\_feature\_get\_data;    // invoke the impulse  // 執行推理  EI\_IMPULSE\_ERROR res = run\_classifier(&features\_signal, &result, true);    ei\_printf("run\_classifier returned: %d\n", res);  ht32\_led\_toggle(LED2);  // 若推理過程有錯誤，則返回  if (res != 0) {  return 1;  }    ei\_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.): \n",result.timing.dsp, result.timing.classification, result.timing.anomaly);  // turn off LED2,3,4  ht32\_led\_off(LED2);  ht32\_led\_off(LED3);  ht32\_led\_off(LED4);  // print the predictions  ei\_printf("[");  for (size\_t ix = 0; ix < EI\_CLASSIFIER\_LABEL\_COUNT; ix++) {  //機率大於0.5時即推測為該手勢  if (result.classification[ix].value > 0.5)  {  switch (ix)  {  case 0: //往後揮  ht32\_led\_on(LED2);  ht32\_led\_on(LED4);  break;    case 1: //往前揮  ht32\_led\_on(LED2);  break;    case 2: //往左揮  ht32\_led\_on(LED3);  break;    case 3: //往右揮  ht32\_led\_on(LED4);  break;  }  }  ei\_printf("%.5f", result.classification[ix].value);  #if EI\_CLASSIFIER\_HAS\_ANOMALY == 1  ei\_printf(", ");  #else  if (ix != EI\_CLASSIFIER\_LABEL\_COUNT - 1) {  ei\_printf(", ");  }  #endif  }  #if EI\_CLASSIFIER\_HAS\_ANOMALY == 1  ei\_printf("%.3f", result.anomaly);  #endif  ei\_printf("]\n");  }  }  } |

7.6.5 測試推論結果

最後將程式燒錄至板子上，進行手勢辨識測試。如圖7.28所示，為測試結果圖。該圖是本範例測試畫圈叉推論結果機率數值。其中，判斷圈的手勢機率為16%，而叉的手勢機率為83.9%。

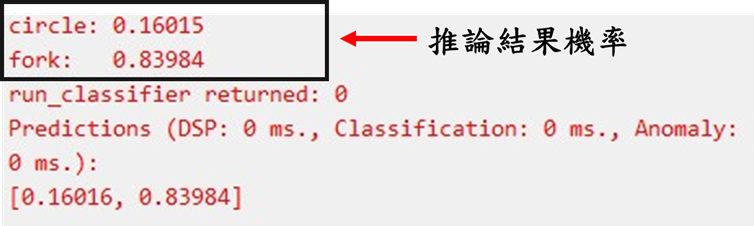


圖7.28、測試結果圖