实验报告

课程名称：物联网技术基础与应用开发

实验类型：操作实验

实验项目名称：实验二：基于 TinyML 的设备运动状态实时识别实验

姓名：蔡佳伟 学号：3220104519

QQ 号码：

（高校联合班成员填写：学校_____姓名_____ 学号_____ ）

实验日期：2024 年 10 月 11 日

一、 实验目的和要求：

- 掌握如何在资源有限的嵌入式设备进行模型推理
- 掌握自主创建简单模型的能力
- 掌握模型数据集准备，模型训练和模型存储等技术
- 掌握 TinyML 工作原理
- 使用神经网络识别设备运动状态并根据运动状态展示不同 LED 颜色的功能

二、 实验内容

基于 ESP32 硬件以及 RIOT 系统，根据实验手册 (https://gitee.com/emnets/emnets_experiment/blob/master/part1_tingml_motion_predict.md) 完成以下实验：

1. 设备运动状态实时识别实验
   - ✓ 通过 GPIO 控制 LED 灯状态，展示不同颜色
   - ✓ IMU 惯性传感器数据集收集，训练测试样本准备
   - ✓ 模型构造，模型训练，模型测试，生成可部署的模型文件
   - ✓ 设备模型导入，模型实时识别设备运动状态，根据运动状态展示不同灯颜色

三、 实验背景

- RIOT 操作系统

  RIOT(https://github.com/RIOT-OS/RIOT) 是一个开源的微控制器操作系统，旨在满足物联网(IoT)设备和其他嵌入式设备的需求。它支持一系列通常在物联网(IoT)中发现的设备:8 位，16 位和 32 位微控制器。RIOT 基于以下设计原则:节能、实时功能、内存占用小、模块化和统一的 API 访问，独立于底层硬件(该 API 提供部分

POSIX 遵从性)。

- TinyML 介绍

TinyML（Tiny Machine Learning）是一种专注于在超低功耗微控制器设备上开发和部署机器学习模型的领域。它允许机器学习在安全、低延迟、低功耗和低带宽的边缘设备上运行，通常功耗在毫瓦（mW）级别甚至更低。TinyML 的实现需要硬件、软件和算法的整体性协同设计，以支持各种始终在线的应用，特别是对电池驱动的设备非常适用 。

TinyML 的重要性在于其体积小、能耗低、成本效益高，适合大规模部署在嵌入式计算应用程序中。它结合了物联网设备、边缘计算和机器学习，为物联网设备的智能化提供了新的可能性 。此外，TinyML 的应用可以减少数据传输的需要，增强数据隐私保护，降低网络带宽的使用，并提高系统的可靠性和能源效率 。

开发 TinyML 应用程序通常涉及使用如 TensorFlow Lite for Microcontrollers (TF Lite Micro)这样的框架，它可以在资源受限的微控制器上实现机器学习任务。TF Lite Micro 提供了丰富的库和工具包，支持在多种微控制器板上进行开发，如 Arduino Nano 33 BLE Sense、SparkFun Edge 等 。

四、 主要仪器设备

- PC
- ESP32-WROOM-32、MPU6050 惯性传感器、LED RGB 灯。

五、 实验题目简答

a) 想象一下,TinyML 在生活中的应用场景有哪些？

有如下应用场景：

1. 智能家居

- 语音控制设备：通过 TinyML，智能音箱、智能灯、智能插座等设备可以实现离线语音识别，减少对云端的依赖，提升隐私和反应速度。例如，可以通过语音命令控制灯光、空调等电器。

- 智能温控器：基于传感器数据的学习，TinyML 可以根据房间的温度变化、时间和用户习惯来自动调节温度，提升舒适度和节能效果。

2. 健康监测

- 可穿戴设备：智能手表、健身追踪器等可穿戴设备通过 TinyML 实时分析心率、睡眠模式、运动状态等数据，提供更个性化的健康建议，而不需要持续连接到云端处理数据。

- 家庭医疗设备：一些智能健康设备（如血压计、血糖仪等）可以通过 TinyML 进行数据处理，帮助用户实时监控和分析自己的健康状况。

3. 工业物联网（IIoT）

- 设备状态监控：工厂设备或工业设施上的传感器可以使用 TinyML 模型检测振动、温度或其他参数，判断设备是否处于正常状态，并在出现异常时发出预警，防止设备损坏或停机。

- 预测性维护：利用 TinyML 分析设备运行中的数据，预测设备的故障趋势，提前安排维修，以减少停机时间和维修成本。

4. 农业物联网

- 智能灌溉：通过传感器数据（如土壤湿度、温度、天气预报等）以及 TinyML 模型，自动调节灌溉系统，确保作物得到适量的水分，既节水又提高农业产量。

- 作物健康监测：利用 TinyML 分析农田传感器或无人机收集的数据，识别病虫害、作物营养不足等问题，帮助农民及时采取行动。

5. 智能城市

- 智能交通管理：通过在交通灯或监控设备中嵌入 TinyML，实时分析交通流量数据，优化交通灯的切换，减少交通拥堵，提升交通效率。

- 环境监测：TinyML 设备可以用于监控空气质量、噪音水平和温度变化等指标，帮助城市管理者及时采取措施改善环境。

6. 安防与监控

- 智能摄像头：TinyML 可以嵌入到家庭或商用摄像头中，实现本地化的人脸识别、异常行为检测等功能，保障隐私的同时减少数据传输延迟。

- 入侵检测：家用安防设备通过 TinyML 可以检测出异常的声响或振动（如玻璃破裂声），并在本地触发警报。

7. 个人助理与工具

- 运动追踪：TinyML 可以嵌入在手机或耳机中，通过传感器数据检测用户的运动状态，如跑步、骑行、步行等，提供更加精准的运动分析。

- 手势识别：利用 TinyML，智能手表或手机可以识别用户的手势动作，用于控制设

备或执行特定任务，如通过手势控制音乐播放。

8. 汽车行业

- 驾驶员监控：通过 TinyML，可以监控驾驶员的疲劳状态、注意力分散情况，并在危险时发出警告，从而提高驾驶安全性。

- 车辆传感器分析：车辆中的传感器数据可以通过 TinyML 进行实时处理，检测车辆异常，如刹车系统问题、胎压异常等，及时发出提醒。

9. 教育和玩具

- 智能玩具：一些智能玩具通过 TinyML 实现个性化互动，根据儿童的行为和语言进行反应，提升娱乐和教育的效果。

- 学习工具：基于 TinyML 的教育设备能够根据学生的学习习惯和反馈，调整学习内容和节奏，提高学习效率。

10. 环境保护

- 野生动物监测：安装在野外的小型传感器设备使用 TinyML 分析动物活动、声音等数据，实时监控野生动物的动向，帮助科研人员保护濒危物种。

- 自然灾害预警：通过 TinyML 分析环境传感器数据，如地震前的振动、火灾前的温度变化等，及时发出预警，减少灾害损失。

TinyML 的关键优势在于其能在资源受限的设备上运行，因此在对功耗、响应时间、隐私保护等有严格要求的场景中非常有用。随着物联网的进一步普及，TinyML 将会在更多领域带来变革性的应用，改善人们的日常生活。

b) 模型数据集训练集和测试集的作用是什么？请详细介绍数据训练集大小不足会导致模型训练出现什么情况?

数据集分为训练集和测试集，在机器学习模型的开发过程中各有其作用：

- 训练集：用于训练模型，即让模型学习数据中的模式和规律。模型通过训

练集不断调整参数，以提高预测能力。

- 测试集：用于评估模型的性能。它是模型从未见过的数据，用来检查模型的泛化能力，即它在处理新数据时的表现。

训练集大小不足的影响

如果训练集太小，可能导致以下问题：

1. 过拟合（Overfitting）：模型在训练数据上表现很好，但由于训练集规模小，模型可能过度记忆了训练数据的细节，而无法在新数据上做出准确的预测，泛化能力差。

2. 模型不够准确：由于数据样本太少，模型无法学习到足够多的模式，可能无法形成有效的决策边界，导致预测性能低下。

3. 不稳定性：训练集的代表性不强，导致模型对训练数据的微小变化特别敏感，结果不稳定。

因此，合适大小的训练集是确保模型能够有效学习并在新数据上表现良好的关键。

c) 什么是欠拟合，什么是过拟合？如何避免出现这两者情况，请各列举至少两点措施?

欠拟合：模型过于简单，无法捕捉数据中的复杂模式，导致在训练集和测试集上都表现不佳。

- 避免欠拟合的措施：

1. 增加模型复杂度：使用更复杂的模型（如增加神经网络层数、使用更多特征）。

2. 更多特征工程：提取或生成更有意义的特征，提高模型对数据的理解能力。

过拟合：模型过于复杂，过度记忆训练数据中的细节和噪声，导致在训练集上表现很好，但在测试集上表现差。

- 避免过拟合的措施：

1. 正则化：使用 L1、L2 正则化或 Dropout 来约束模型的复杂度。
2. 增加数据量：通过收集更多数据或使用数据增强技术，帮助模型更好地泛化。

d) tflite-micro 库中 MicroMutableOpResolver 和 AllOpsResolver 两者的区别? 为什么 AllOpsResolver 会被淘汰?

在 TensorFlow Lite Micro (TFLite Micro) 库中，`MicroMutableOpResolver` 和 `AllOpsResolver` 是用于管理操作符（ops）解析的两个类，它们的区别在于如何选择模型中使用的操作符：

1. MicroMutableOpResolver

- 用途：它是一个可定制的操作解析器。开发者可以手动添加所需的操作符，仅包含模型实际需要的那些操作。
- 优点：由于只加载必要的操作符，内存占用更小，适合资源受限的设备。

2. AllOpsResolver

- 用途：它会加载 TensorFlow Lite 支持的所有操作符。
- 缺点：因为包含了所有可能的操作符，内存和存储占用大，超出嵌入式设备的资源限制。

由于资源受限的设备（如微控制器）通常具有非常有限的存储和内存，`AllOpsResolver` 会显著增加内存占用，不适合这些场景。而 `MicroMutableOpResolver` 只加载必要的操作符，能有效减少内存和存储需求，因此在嵌入式设备中更加高效和实用。

六、 实验数据记录和处理

1. 基础设备控制实验

1) 代码截图。

Ledcontroller.cpp:

直接复制第一次试验代码：

```
#include "ztimer.h"

#include "ledcontroller.hh"

#include "periph/gpio.h"

#include <stdio.h>

/**

 * Initialize the RGB pins as output mode, and initially, the lights should be off.

 * gpio_init(gpio_pin, GPIO_OUT);

 * gpio_write(pin, 0);

 */

LEDController::LEDController(uint8_t gpio_r, uint8_t gpio_g, uint8_t gpio_b){

    printf("LED Controller initialized with (RGB: GPIO%d, GPIO%d, GPIO%d)\n", gpio_r, gpio_g, gpio_b);

    // input your code

    led_gpio[0] = gpio_r;

    led_gpio[1] = gpio_g;

    led_gpio[2] = gpio_b;

    for(int i=0;i<3;i++){

        gpio_init(led_gpio[i],GPIO_OUT); // init all gpio GPIO_OUT

        gpio_write(led_gpio[i],0); // close all led default

    }

}



/**

 * Implement a light that displays at least 5 status colors through the RGB three pins.

 * -------------------------------------------------------------
```

```
 * @note Method 1
 * Utilizes the gpio_write function to set the GPIO pin connected to the LED to the current LED state.
 * void gpio_write(uint8_t pin, int value);
 * @param pin The GPIO pin connected to the LED.
 * @param value The value to set the GPIO pin to (0 for LOW, 1 for HIGH).
 * ------------------------------------------------------------
 * @note Method 2
 * Uses the gpio_set and gpio_clear functions to set the GPIO pin connected to the LED to the current LED state.
 * void gpio_set(uint8_t pin); void gpio_clear(uint8_t pin);
 * @param pin The GPIO pin connected to the LED.
 */
void LEDController::change_led_color(uint8_t color){
    // input your code
    switch (color)
    {
    case COLOR_NONE:
        gpio_write(led_gpio[0],0); //close red
        gpio_write(led_gpio[1],0); //close green
        gpio_write(led_gpio[2],0); //close blue
        break;
    case COLOR_RED:
        gpio_write(led_gpio[0],1); //open red
        gpio_write(led_gpio[1],0); //close green
        gpio_write(led_gpio[2],0); //close blue
        break;
    case COLOR_GREEN:
        gpio_write(led_gpio[0],0); //close red
        gpio_write(led_gpio[1],1); //open green
        gpio_write(led_gpio[2],0); //close blue
```

```
        break;
    case COLOR_YELLO:
        gpio_write(led_gpio[0],1); //open red
        gpio_write(led_gpio[1],1); //open green
        gpio_write(led_gpio[2],0); //close blue
        break;
    case COLOR_BLUE:
        gpio_write(led_gpio[0],0); //close red
        gpio_write(led_gpio[1],0); //close green
        gpio_write(led_gpio[2],1); //open blue
        break;
    case COLOR_MAGENTA:
        gpio_write(led_gpio[0],1); //open red
        gpio_write(led_gpio[1],0); //close green
        gpio_write(led_gpio[2],1); //open blue
        break;
    case COLOR_CYAN:
        gpio_write(led_gpio[0],0); //close red
        gpio_write(led_gpio[1],1); //open green
        gpio_write(led_gpio[2],1); //open blue
        break;
    case COLOR_WHITE:
        gpio_write(led_gpio[0],1); //open red
        gpio_write(led_gpio[1],1); //open green
        gpio_write(led_gpio[2],1); //open blue
        break;
    default:
        break;
    }
```

```
}
```

Main.cpp：结合第二个案例完成如下：

```
#include "periph_conf.h"

#include "periph/gpio.h"

#include "periph/i2c.h"

#include "shell.h"

#include <log.h>

#include <xtimer.h>

#include "ledcontroller.hh"

#include "ztimer.h"

#include "mpu6050.h"

#include <string>

#include <math.h>

#include "msg.h"



void setup();

int predict(float *imu_data, int data_len, float threashold, int class_num);

using namespace std;

#define THREAD_STACKSIZE                (THREAD_STACKSIZE_IDLE)

static char stack_for_motion_thread[THREAD_STACKSIZE];

static char stack_for_led_thread[THREAD_STACKSIZE];

static kernel_pid_t _led_pid;

#define LED_GPIO_R GPIO26

#define LED_GPIO_G GPIO25

#define LED_GPIO_B GPIO27

#define g_acc (9.8)

#define SAMPLES_PER_GESTURE (10)

#define ACC_THRESHOLD 1.5 //  加速度静止阈值
```

```
#define ROT_THRESHOLD 50 //  角速度静止阈值

struct MPU6050Data

{

    float ax, ay, az; // acceler_x_axis, acceler_y_axis, acceler_z_axis

    float gx, gy, gz; // gyroscope_x_axis, gyroscope_y_axis, gyroscope_z_axis

};



enum MoveState{Stationary, Tilted, Rotating, Moving, Moving_X, Moving_Y,

};



void delay_ms(uint32_t sleep_ms)

{

    ztimer_sleep(ZTIMER_USEC, sleep_ms * US_PER_MS);

    return;

}

/**

 * LED control thread function.

 * Then, it enters an infinite loop where it waits for messages to control the LED.

 * @param arg Unused argument.

 * @return NULL.

 */

void *_led_thread(void *arg)

{

    (void) arg;

    LEDController led(LED_GPIO_R, LED_GPIO_G, LED_GPIO_B);

    led.change_led_color(0);

    while(1){

        // Input your codes

        // Wait for a message to control the LED
```

```
            // Display different light colors based on the motion state of the device.
            msg_t msg;
            msg_receive(&msg);
            if (msg.content.value == Stationary) {
            led.change_led_color(COLOR_YELLO);
            } else if (msg.content.value == Moving_X) {
            led.change_led_color(COLOR_CYAN);
            } else if (msg.content.value == Moving_Y) {
            led.change_led_color(COLOR_WHITE);
            } else if (msg.content.value == Tilted) {
            led.change_led_color(COLOR_RED);
            } else if (msg.content.value == Rotating) {
            led.change_led_color(COLOR_BLUE);
            } else if (msg.content.value == Moving) {
            led.change_led_color(COLOR_GREEN);
            } else {
            led.change_led_color(COLOR_NONE);
            }
            delay_ms(10);
        }
        return NULL;
}

float gyro_fs_convert = 1.0;
float accel_fs_convert;

void get_imu_data(MPU6050 mpu, float *imu_data){
        int16_t ax, ay, az, gx, gy, gz;
        for(int i = 0; i < SAMPLES_PER_GESTURE; ++i)
```

```
    {
        /* code */
        delay_ms(20);
        mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
        imu_data[i*6 + 0] = ax / accel_fs_convert;
        imu_data[i*6 + 1] = ay / accel_fs_convert;
        imu_data[i*6 + 2] = az / accel_fs_convert;
        imu_data[i*6 + 3] = gx / gyro_fs_convert;
        imu_data[i*6 + 4] = gy / gyro_fs_convert;
        imu_data[i*6 + 5] = gz / gyro_fs_convert;
    }
}

void *_motion_thread(void *arg)
{
    (void) arg;
    // Initialize MPU6050 sensor
    MPU6050 mpu;
    // get mpu6050 device id
    uint8_t device_id = mpu.getDeviceID();
    printf("[IMU_THREAD] DEVICE_ID:0x%x\n", device_id);
    mpu.initialize();
    // Configure gyroscope and accelerometer full scale ranges
    uint8_t gyro_fs = mpu.getFullScaleGyroRange();
    uint8_t accel_fs_g = mpu.getFullScaleAccelRange();
    uint16_t accel_fs_real = 1;

    // Convert gyroscope full scale range to conversion factor
    if (gyro_fs == MPU6050_GYRO_FS_250)
```

```
        gyro_fs_convert = 131.0;
    else if (gyro_fs == MPU6050_GYRO_FS_500)
        gyro_fs_convert = 65.5;
    else if (gyro_fs == MPU6050_GYRO_FS_1000)
        gyro_fs_convert = 32.8;
    else if (gyro_fs == MPU6050_GYRO_FS_2000)
        gyro_fs_convert = 16.4;
    else
        printf("[IMU_THREAD] Unknown GYRO_FS: 0x%x\n", gyro_fs);

    // Convert accelerometer full scale range to real value
    if (accel_fs_g == MPU6050_ACCEL_FS_2)
        accel_fs_real = g_acc * 2;
    else if (accel_fs_g == MPU6050_ACCEL_FS_4)
        accel_fs_real = g_acc * 4;
    else if (accel_fs_g == MPU6050_ACCEL_FS_8)
        accel_fs_real = g_acc * 8;
    else if (accel_fs_g == MPU6050_ACCEL_FS_16)
        accel_fs_real = g_acc * 16;
    else
        printf("[IMU_THREAD] Unknown ACCEL_FS: 0x%x\n", accel_fs_g);

    // Calculate accelerometer conversion factor
    accel_fs_convert = 32768.0 / accel_fs_real;
    float imu_data[SAMPLES_PER_GESTURE * 6] = {0};
    int data_len = SAMPLES_PER_GESTURE * 6;
    delay_ms(200);
    // Main loop
    int predict_interval_ms = 200;
```

```
    int ret = 0;
#define class_num (4)
    float threshold = 0.7;
    string motions[class_num] = {"Stationary", "Tilted", "Rotating", "Moving"};
    while (1) {
        delay_ms(predict_interval_ms);
        // Read sensor data
        get_imu_data(mpu, imu_data);
        ret = predict(imu_data, data_len, threshold, class_num);
        // tell the led thread to do some operations
        // input your code
        // Print result
        MoveState state;
        for(int i = 0; i < SAMPLES_PER_GESTURE; ++i){
        i += 1;
        // 计算加速度在 X 轴、Y 轴和 Z 轴上的大小
        float acceleration_x = fabs(imu_data[i*6 + 0]);
        float acceleration_y = fabs(imu_data[i*6 + 1]);
        // 检测设备的总加速度
        float total_acceleration = sqrt(imu_data[i*6 + 0] * imu_data[i*6 + 0]
        + imu_data[i*6 + 1] * imu_data[i*6 + 1] + imu_data[i*6 + 2] * imu_data[i*6 + 2]);
        // 计算陀螺仪总角速度
        float total_rotation = sqrt(imu_data[i*6 + 3] * imu_data[i*6 + 3] +
        imu_data[i*6 + 4] * imu_data[i*6 + 4] + imu_data[i*6 + 5] * imu_data[i*6 + 5]);
        // 判断水平静止：Z 轴加速度接近 g，X 轴和 Y 轴加速度接近 0，且没有旋转
        if (fabs(total_acceleration - g_acc) < ACC_THRESHOLD*3 &&
        acceleration_x < ACC_THRESHOLD*3 && acceleration_y < ACC_THRESHOLD*3 &&
        total_rotation < ROT_THRESHOLD/2) {
        state = Stationary;
```

```
}
// 判断倾斜静止：设备的加速度在 X、Y、Z 轴上较为稳定，且没有旋转
else if (fabs(total_acceleration - g_acc) < ACC_THRESHOLD*3 &&
total_rotation < ROT_THRESHOLD/2) {
state = Tilted;
}
// 判断 X 轴平移：X 轴加速度大于阈值，且没有旋转
else if (acceleration_x > ACC_THRESHOLD && acceleration_y <
ACC_THRESHOLD && total_rotation < ROT_THRESHOLD) {
state = Moving_X;
}
// 判断 Y 轴平移：Y 轴加速度大于阈值，且没有旋转
else if (acceleration_y > ACC_THRESHOLD && acceleration_x <
ACC_THRESHOLD && total_rotation < ROT_THRESHOLD) {
state = Moving_Y;
}
// 判断旋转：总角速度大于阈值
else if (total_rotation > ROT_THRESHOLD) {
state = Rotating;
}
// 判断移动状态:总加速度明显偏离重力加速度
else if (fabs(total_acceleration - g_acc) > ACC_THRESHOLD) {
state = Moving;
}
else {
state = Moving;
}
}
// 向 LED 线程发送消息通知当前的运动状态
```

```
            msg_t msg;
            msg.content.value = (int)state;
            msg_send(&msg, _led_pid); // 发送消息到 LED 控制线程
            printf("Predict: %d, %s\n", ret, motions[ret].c_str());
        }
        return NULL;
}









int main(int argc, char* argv[])
{
        (void)argc;
        (void)argv;
        setup();
        _led_pid = thread_create(stack_for_led_thread, sizeof(stack_for_led_thread), THREAD_PRIORITY_MAIN - 2,
                                 THREAD_CREATE_STACKTEST, _led_thread, NULL,
                                 "led_controller_thread");
        if (_led_pid <= KERNEL_PID_UNDEF) {
            printf("[MAIN] Creation of receiver thread failed\n");
            return 1;
        }
        thread_create(stack_for_motion_thread, sizeof(stack_for_motion_thread), THREAD_PRIORITY_MAIN - 1,
                                 THREAD_CREATE_STACKTEST, _motion_thread, NULL,
                                 "imu_read_thread");
        printf("[Main] Initialization successful - starting the shell now\n");
        while(1);
```

```
    return 0;


}
```

2) 模型结构图(summary 方法打印的结果图)。

CNN 模型：

```
cjw@cjw-virtual-machine:~/RIOT/examples/emnets_experiment/12_tingml_gesture_pred
ict_experiment/external_modules/gesture$ python3.8 train.py
(784, 60, 1) (196, 60, 1) (784,) (196,)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 60, 8)             32
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 60, 8)             200
_________________________________________________________________
global_average_pooling1d (Gl (None, 8)                 0
_________________________________________________________________
dense (Dense)                (None, 8)                 72
_________________________________________________________________
dropout (Dropout)            (None, 8)                 0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 36
=================================================================
Total params: 340
Trainable params: 340
Non-trainable params: 0
_________________________________________________________________
```

MLP 模型：

```
(784, 60) (196, 60) (784,) (196,)
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                3904
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 260
=================================================================
Total params: 4,164
Trainable params: 4,164
Non-trainable params: 0
```

3) 案例 CNN 模型和 MLP 模型量化后，预测精度及混淆矩阵(主机端和 ESP32 设备端预测结果)。

通过 model.evaluate 方法来测试训练模型的准确度，

```
test_loss, test_accuracy = model.evaluate(xTest, yTest, verbose=0)
print("Test Accuracy for MLP/CNN model: {:.2f}%".format(test_accuracy * 100))
```

使用 confusion_matrix 来显示训练模型的混淆矩阵

```
from sklearn.metrics import confusion_matrix
predictions = model.predict(xTest)
predictions = np.argmax(predictions, axis=1)
cm = confusion_matrix(yTest, predictions)
print(cm)
```

我还测试了训练时间，一并展示：

CNN 模型：

```
Test Accuracy for MLP/CNN model: 94.39%
```

```
Epoch 00050: val_sparse_categorical_accuracy did not improve from 0.95918
Training time for model: 7.64 seconds
[[48  0  0  0]
 [ 0 70  0  2]
 [ 0  0 22  2]
 [ 1  0  3 48]]
```

MLP 模型：

```
Test Accuracy for MLP/CNN model: 98.98%
```

```
Epoch 00050: val_sparse_categorical_accuracy did not improve from 0.98469
Training time for model: 5.34 seconds
[[48  0  0  0]
 [ 0 72  0  0]
 [ 0  0 22  2]
 [ 0  0  1 51]]
```

可以看出，MLP 模型的精度更好，混淆矩阵更接近对角矩阵，说明效果更好，训练时间也更小，效果更好。

4) 改善后或自定义模型结构图，主机端训练测试报告(精度和混淆矩阵)，ESP32 设备端预测结果(不同 label 的预测概率)，可自行统计,给出设备端大致推理精度。

改善模型如下：

增加卷积层和滤波器数量，调整卷积核大小，调整 dropout 比例，防止过拟合。

```
def improved_cnn():


    model = keras.Sequential()
    # 增加卷积层和滤波器数量
    model.add(
        keras.layers.Conv1D(
            16, 5, padding="same", activation="relu", input_shape=(6 * SAMPLES_PER_GESTURE, 1)
        )
    )
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Conv1D(32, 5, padding="same", activation="relu"))
    model.add(keras.layers.BatchNormalization())
    model.add(keras.layers.Conv1D(64, 3, padding="same", activation="relu"))
    model.add(keras.layers.BatchNormalization())

    # 全局平均池化
    model.add(keras.layers.GlobalAveragePooling1D())

    # 添加更多的全连接层
    model.add(keras.layers.Dense(64, activation="relu"))
    model.add(keras.layers.Dropout(0.3))  # 调整 Dropout 比例，防止过拟合
    model.add(keras.layers.Dense(32, activation="relu"))
    model.add(keras.layers.Dropout(0.3))

    # 输出层
    model.add(keras.layers.Dense(len(LABELS), activation="softmax"))
```

```
return model
```

精确率和混淆矩阵如下：

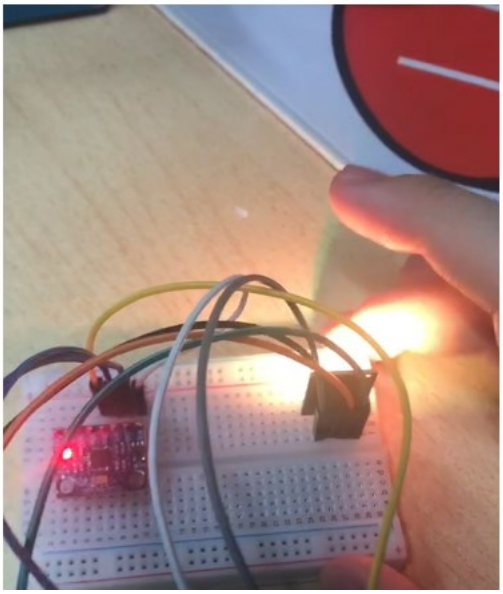
```
Training time for model: 16.38 seconds
[[48  0  0  0]
 [ 0 72  0  0]
 [ 0  0 21  3]
 [ 1  1  0 50]]
```

```
Test Accuracy for MLP/CNN model: 97.45%
[[48  0  0  0]
 [ 0 72  0  0]
 [ 0  0 21  3]
 [ 1  1  0 50]]
```

虽然训练时间更长（这是因为模型做出了改进，步骤更多），精确率对比普通的 CNN 模型有显著提升，混淆矩阵也更接近对角矩阵，表明确实对 CNN 模型进行了改进优化。

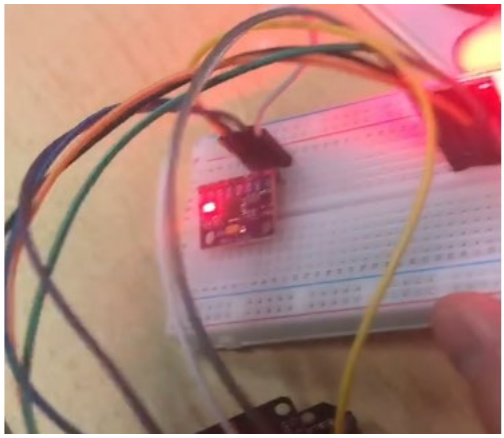
静止时颜色：

```
msg_receive(&msg);
if (msg.content.value == Stationary) {
led.change_led_color(COLOR_YELLO);
```



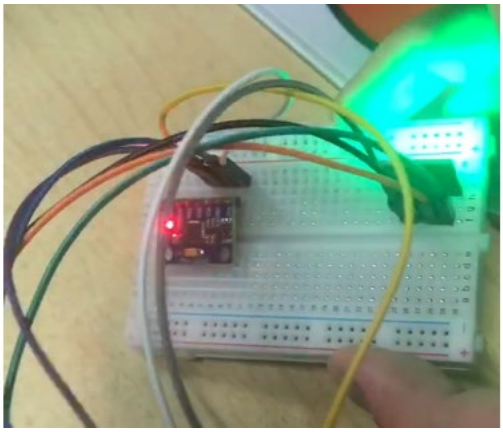
倾斜时颜色：

```
} else if (msg.content.value == Tilted) {
led.change_led_color(COLOR_RED);
```



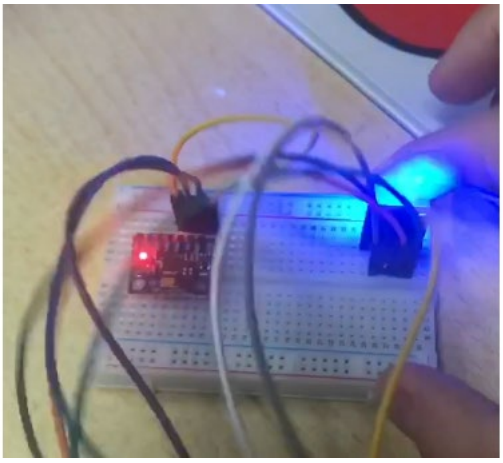
移动时颜色：

```
} else if (msg.content.value == Moving) {
led.change_led_color(COLOR_GREEN);
```



旋转时颜色：

```
} else if (msg.content.value == Rotating) {
led.change_led_color(COLOR_BLUE);
```



表明了补充代码的效果很好。

5) 对实验结果视频的补充？若没有，可忽略。

视频另附，这个是第二部分的结果。

```
2024-10-11 16:02:10,586 # ------------------------------
2024-10-11 16:02:10,588 # [0] value: 0.50
2024-10-11 16:02:10,588 # [1] value: 0.29
2024-10-11 16:02:10,590 # [2] value: 0.04
2024-10-11 16:02:10,591 # [3] value: 0.17
2024-10-11 16:02:10,593 # No match found
2024-10-11 16:02:10,594 # Predict: 0, Stationary
```

```
2024-10-11 16:02:07,161 # ------------------------------
2024-10-11 16:02:07,164 # [0] value: 0.00
2024-10-11 16:02:07,165 # [1] value: 0.97
2024-10-11 16:02:07,166 # [2] value: 0.00
2024-10-11 16:02:07,166 # [3] value: 0.02
2024-10-11 16:02:07,169 # Motion prediction: 1
2024-10-11 16:02:07,170 # Predict: 1, Tilted
```

```
2024-10-11 16:01:58,599 # [0] value: 0.00
2024-10-11 16:01:58,600 # [1] value: 0.00
2024-10-11 16:01:58,602 # [2] value: 0.83
2024-10-11 16:01:58,602 # [3] value: 0.17
2024-10-11 16:01:58,605 # Motion prediction: 2
2024-10-11 16:01:58,607 # Predict: 2, Rotating
```

```
2024-10-11 16:02:09,304 # [0] value: 0.00
2024-10-11 16:02:09,304 # [1] value: 0.00
2024-10-11 16:02:09,307 # [2] value: 0.26
2024-10-11 16:02:09,307 # [3] value: 0.74
2024-10-11 16:02:09,309 # Motion prediction: 3
2024-10-11 16:02:09,310 # Predict: 3, Moving
```

七、 实验结果与分析

- 基于该实验，总结开发一个可部署的 AI 模型应用基本流程。

1. 问题定义与需求分析

   - 明确业务需求和应用场景，确定要解决的问题，以及评估可用的数据和计算资源。

2. 数据收集与准备

   - 数据收集：从不同来源收集与问题相关的数据。

- 数据清洗：处理缺失值、异常值、去除噪声，确保数据质量。
- 特征工程：提取、转换或创建有助于模型训练的特征。

3. 选择模型

- 根据问题类型（分类、回归、目标检测等）选择合适的算法或架构（如决策树、神经网络等）。
- 使用预训练模型或从头开始构建模型。

4. 模型训练

- 使用训练集来训练模型，让模型从数据中学习模式和规律。
- 调整模型参数和超参数，提升模型性能。

5. 模型评估与验证

- 在验证集和测试集上评估模型的准确性、泛化能力和性能指标（如精度、召回率等）。
- 通过交叉验证或调整超参数来避免欠拟合或过拟合。

6. 优化与压缩

- 模型优化：通过剪枝、量化等技术减小模型大小，提升推理速度，降低内存和计算需求。
- 轻量化：特别是嵌入式设备或移动设备上，使用如 TensorFlow Lite、TinyML 等库进行模型轻量化处理。

7. 部署

- 选择部署环境（云端、移动设备、嵌入式设备）。
- 将优化后的模型集成到应用程序中，进行推理和预测。
- 确保部署后的模型能实时或近实时处理数据。

8. 监控与维护

- 持续监控模型在实际应用中的表现，确保其准确性和性能稳定。

- 根据需要对模型进行重新训练或更新，以适应新的数据或需求。

总结

这个流程从数据收集到模型部署，确保了模型在实际环境中的高效应用，同时能够根据需求灵活调整和优化。