**설계 과정**

**실패한 아이디어: On / Off 스위치가 달린 제스처를 인식해 밝기를 조정하는 스탠드 겸 무드등**

1. **실패한 아이디어 ( 첫번째 아이디어 )**

**“기존에 볼 수 없었던 기발한 아이디어의 적용”**이라는 평가 항목을 위해 위에서 정한 아이디어를 구현할 수 있는 기술에 대해 모색했다. 모색하던 중 이전에 배운 적 있는 tinyML을 우연히 볼 수 있었고 아두이노를 지원함을 알 수 있었다. 곧바로 동작을 학습시켜 아이디어를 구현하기 위해 이를 위한 다양한 레퍼런스를 탐색하게 되었다.

레퍼런스에는 tinyML을 위해 MCU(Micro Controller Unit)으로 nRF52840을 탑재한 NANO-FAMILY의 최상위 라인업이라고 할 수 있는 Arduino nano 33 ble sense이 필요했다.

테이블이(가) 표시된 사진

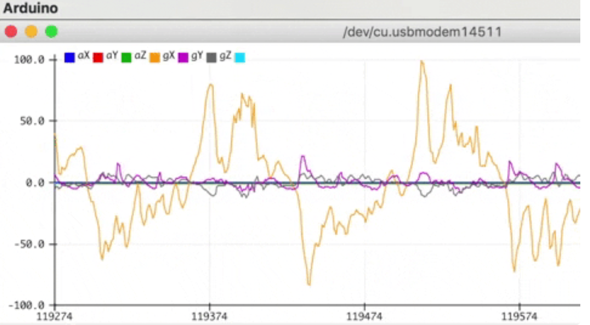
자동 생성된 설명

구입한 Arduino nano 33 ble sense을 바탕으로 Machine Learning을 위한 학습 데이터 셋을 설정하기 위해 아주 최적화된 센서 라인업을 가지고 있음을 확인했다. 위에서 볼 수 있듯이 각각의 Module을 설치해서 코드를 짜서 시리얼 보드에서 각각의 센서들의 작동 여부를 확인할 수 있었다.

텍스트이(가) 표시된 사진

자동 생성된 설명

센서가 정상 작동하는 것을 확인 했으니 이와 관련된 툴을 설치했다. 또한 이와 관련된 데이터(손동작)을 수집하고 전처리 과정을 거친 데이터를 Test Data-Set와 Validate Data-Set로 나누어 학습시키고 검증해서 정확도를 측정하는 과정을 진행하였다. 우선 TensorFlow Blog공식 홈페이지에 올라와 있는 Visualizing live sensor data log from the Arduino board글을 바탕으로 센서의 값을 손 쉽게 확인하기 위해 시각화 하였다.



이제 Training in Tensorflow글을 바탕으로 Google의 Colab으로 이미 완성되어 있는 신경망(모델)로 csv파일을 Python으로 학습시켜서 벤치마크를 진행하였다.

텍스트이(가) 표시된 사진

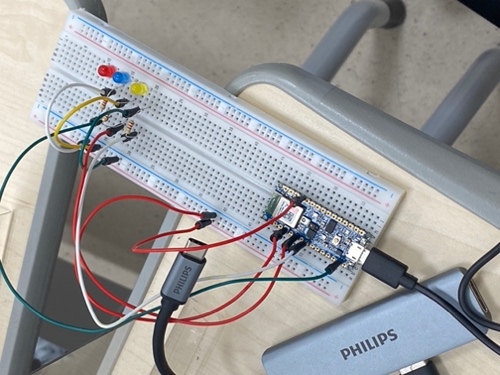
자동 생성된 설명

1. **차선책 ( 두번째 아이디어 )**

**차선책: 주변의 물체를 스마트하게 감지하고 확인할 수 있는 레이더**

**부제: 엄마 탐지 레이더**

하지만 학습을 다 시키고 큰 문제가 발생하였다. 바로 Arduino nano 33 ble sense와 Arduino Uno보드와의 통신 하는 법을 생각하지 않고 무작정 33 ble sense에서 학습만 시킨 것이었다. 물론 ble가 Bluetooth Low Energy를 뜻하는 것인데 블루투스 저전력 프로토콜을 뜻한다. 이를 활용하면 둘 간의 통신은 식은죽 먹기였다. 하지만 시간이 촉박한 상황에서 추가로 사야하는 모듈이 있었기 때문에 굳이 이러한 위험 부담을 지지 않기로 하였다.



고심 끝에 우리가 생각한 방안은 Evaluation Matrix과정에서 선정되었던 차선책을 실행하는 것이었다. 바로 회의를 진행하였고 차선책인 레이더를 통한 물체 인식과 관련된 프로젝트를 하기로 결정했다. 이것 또한 tinyML을 적용하기에 아주 좋은 프로젝트였다. 그 이유는 물체가 접근하는 것을 학습시켜서 레이더의 정확성을 올려 줄 수 있을 것 이라 생각했기 때문이다.

또한 여기서 다른 레이더 프로젝트와의 차별점을 두기 위해 프로세싱으로 모니터링을 할 수 있게끔 하면 어떨까?라고 생각하게 되었다. 여기서 프로세싱이랑 컴퓨터 프로그래밍의 본질을 시각적 개념으로 교육하기 위해 개발된 오픈 소스 프로그래밍 언어이자 통합 개발 환경이다.

1. 코드 작성 - **각 코드에 대한 설명은 주석으로 대체하겠다.**

**3-1. Final\_Project.ino**



// 서브 모터와 관련된 헤더 파일

#include<Servo.h>

// TODO: 인식된 결과를 시각화 하기 위한 lcd패널 헤더

#include <LiquidCrystal.h>

// TODO: 비동기 처리를 위한 Timer 헤더

#include "Timer.h"

// 각각의 변수(서브모터, Timer) 생성

Servo myservo;

Timer t;

// 각도 변수

int pos = 0;

int angle = 0;

// 감지 결과를 담아줄 변수 4개

int value1, value2, value3, value4;

// led 깜빡거리기 위한 변수

int count = 0;

int count2 = 0;

void setup()

{

// 9600으로 Serial통신 시작

Serial.begin(9600);

// 9번핀에 서브모터 부착

myservo.attach(9);

pinMode(2, OUTPUT);

pinMode(3, OUTPUT);

pinMode(10, OUTPUT);

pinMode(11, OUTPUT);

pinMode(5, INPUT);

pinMode(6, INPUT);

pinMode(7, INPUT);

pinMode(8, INPUT);

// 부저 OUTPUT

pinMode(12, OUTPUT);

// 4 -> Red LED OUTPUT

pinMode(4, OUTPUT);

// 13 -> BLUE LED OUTPUT

pinMode(13, OUTPUT);

}

void loop() {

// ( 모터 회전 설정 )

// 90도 ~ -90도로 번갈아 가면서 레이더로 주변 사물을 인식해 주기 위한 반복문

for (pos = 0; pos <= 90; pos += 1)

{

// 서브 모터 로 값 들여 오기

myservo.write(pos);

delay(10);

// ultraSonicAll함수 호출

ultraSonicAll();

// 각도 1씩 증가 시키기

angle=angle+1;

}

// 위와 동일 방향만 반대

for (pos = 90; pos >= 0; pos -= 1)

{

myservo.write(pos);

delay(10);

ultraSonicAll();

angle=angle-1;

}

}

void ultraSonicAll() {

value1 = ultraSonic(2, 5);

value2 = ultraSonic(10, 8);

value3 = ultraSonic(11, 7);

value4 = ultraSonic(3, 6);

// 거리 한계점 일단 설정 ( main.cpp파일에서 머신러닝과 합쳐서 정확도 높임 )

if(value1 <= 10 || value2 <= 10 || value3 <= 10 || value4 <= 10) {

tone(12, 261, 100);

turnLed();

}

// Debuggig하기 위한 코드

Serial.print(ultraSonic(2, 5));

Serial.print(",");

Serial.print(ultraSonic(10, 8));

Serial.print(",");

Serial.print(ultraSonic(11, 7));

Serial.print(",");

Serial.print(ultraSonic(3, 6));

Serial.print(",");

Serial.print(-angle);

Serial.print(".");

}

// ( 초음파 센서 설정 )

// 각각의 초음파 센서로 값(거리 cm)읽어 들이기

int ultraSonic(int trigPin, int echoPin) {

unsigned long duration;

// 초음파 센서 값 읽기

digitalWrite(trigPin, LOW);

delayMicroseconds(2);

digitalWrite(trigPin, HIGH);

delayMicroseconds(10);

digitalWrite(trigPin, LOW);

// 음속과 관련된 공식으로 물체와의 거리 계산하기

duration = pulseIn(echoPin, HIGH,6000);

int distance = duration \* 17 / 1000;

// 범위 밖이면 고정된 값으로 바꾸어 주기

if(distance >= 40 || distance == 0) {

distance = 40;

}

return distance;

}

// ( LED 설정 )

void turnLed() {

// count2 가 짝수라면 키면 된다 -> 홀수라면 끄면 된다.

// 빨간 불을 킨다 ( 빨간불 차례여야 하고 { count가 짝수여야 한다 } 파란불이 꺼져있다 )

if(count % 2 == 0) {

if(count2 % 2 == 0) {

digitalWrite(4, HIGH);

} else {

digitalWrite(4, LOW);

count++;

}

count2++;

// 파란 불을 킨다 ( 파란불 차례여야 { count가 홀수여야 한다 } 하고 빨간불이 꺼져있다면 )

} else if(count % 2 != 0) {

if(count2 % 2 == 0) {

digitalWrite(13, HIGH);

} else {

digitalWrite(13, LOW);

count++;

}

count2++;

}

}

**3-2. Final\_Project.pde**

import processing.serial.\*;

Serial myPort;

String myString = null;

int distance\_up = 0;

int distance\_right = 0;

int distance\_down = 0;

int distance\_left = 0;

int angle = 0;

void setup(){//기본설정

size(700, 700);

background(0);

myPort = new Serial(this, "/dev/cu.usbmodem14201", 9600);

}

void draw(){//화면에 그림을 표시해줍니다.

noStroke();

fill(0,10);

rect(0,0,width,height);

drawRader();

drawLine();

}

void serialEvent(Serial p){//시리얼 입력이 생기면

try{

myString = p.readStringUntil('.');

String[] list = split(myString, ',');

distance\_up = int(list[0]);

distance\_right = int(list[1]);

distance\_down = int(list[2]);

distance\_left = int(list[3]); //각 방향마다 거리를 저장

angle = int(list[4]); //각도를 저장

//angle = 90;

}catch(Exception e){

}

}

void drawRader(){

pushMatrix();

translate(width/2, height); //line length

noFill();

strokeWeight(2); //line thicknes

stroke(98, 245, 31);

// draw the arc lines

arc(0, -width/2, width, width, 0, TWO\_PI);

arc(0, -width/2, width\*3/4, width\*3/4, 0, TWO\_PI);

arc(0, -width/2, width\*2/4, width\*2/4, 0, TWO\_PI);

arc(0, -width/2, width\*1/4, width\*1/4, 0, TWO\_PI);

arc(0, 0, width, width, PI, PI);

textSize(15);

fill(98, 245, 31);

textAlign(RIGHT);

text("10cm", width\*1/8, -350);

text("20cm", width\*2/8, -350);

text("30cm", width\*3/8, -350);

text("40cm", width\*4/8, -350);

popMatrix();

}

void drawLine(){

pushMatrix();

strokeWeight(10); //point thickness

stroke(255,10, 10); // red color

float d1 = (width/2.0/40.0)\*(float)distance\_up;

float d2 = (width/2.0/40.0)\*(float)distance\_right;

float d3 = (width/2.0/40.0)\*(float)distance\_down;

float d4 = (width/2.0/40.0)\*(float)distance\_left;

line(width/2,width/2,width/2-(d1\*cos(radians(angle))),width/2-(d1\*sin(radians(angle))));

line(width/2,width/2,width/2-(d2\*cos(radians(angle+90))),width/2-(d2\*sin(radians(angle+90))));

line(width/2,width/2,width/2-(d3\*cos(radians(angle+180))),width/2-(d3\*sin(radians(angle+180))));

line(width/2,width/2,width/2-(d4\*cos(radians(angle+270))),width/2-(d4\*sin(radians(angle+270))));

popMatrix();

}

**3-3. Final\_Project\_main.cpp**

/\*

\* Adapted from Magic Wand TFLite demo

\*/

#include <Arduino.h>

#include <Arduino\_LSM9DS1.h>

#include "accelerometer\_handler.h"

#include "gesture\_predictor.h"

#include "magic\_wand\_model\_data.h"

#include "output\_handler.h"

#include "tensorflow/lite/micro/kernels/micro\_ops.h"

#include "tensorflow/lite/micro/micro\_error\_reporter.h"

#include "tensorflow/lite/micro/micro\_interpreter.h"

#include "tensorflow/lite/micro/micro\_mutable\_op\_resolver.h"

#include "tensorflow/lite/schema/schema\_generated.h"

#include "tensorflow/lite/version.h"

// Globals, used for compatibility with Arduino-style sketches.

namespace {

tflite::*ErrorReporter*\* error\_reporter = nullptr;

const tflite::*Model*\* model = nullptr;

tflite::*MicroInterpreter*\* interpreter = nullptr;

*TfLiteTensor*\* model\_input = nullptr;

int input\_length;

// Create an area of memory to use for input, output, and intermediate arrays.

// The size of this will depend on the model you're using, and may need to be

// determined by experimentation.

constexpr int kTensorArenaSize = 60 \* 1024;

uint8\_t tensor\_arena[kTensorArenaSize];

// Whether we should clear the buffer next time we fetch data

bool should\_clear\_buffer = false;

} // namespace

void setup()

{

// Set up logging. Google style is to avoid globals or statics because of

// lifetime uncertainty, but since this has a trivial destructor it's okay.

static tflite::*MicroErrorReporter* micro\_error\_reporter; // NOLINT

error\_reporter = &micro\_error\_reporter;

// Wait until we know the serial port is ready

while (!Serial) {

}

delay(500);

error\_reporter->Report("\n\r\n\rMagic Wand - TensorFlow Lite demo");

// Map the model into a usable data structure. This doesn't involve any

// copying or parsing, it's a very lightweight operation.

model = tflite::GetModel(g\_magic\_wand\_model\_data);

if (model->version() != TFLITE\_SCHEMA\_VERSION)

{

error\_reporter->Report(

"TFLite Model provided is schema version %d, which not equal "

"to supported version %d.",

model->version(), TFLITE\_SCHEMA\_VERSION);

return;

}

else {

error\_reporter->Report(

"TFLite Model provided is schema version %d.", model->version());

}

// Pull in only the operation implementations we need.

// This relies on a complete list of all the ops needed by this graph.

// An easier approach is to just use the AllOpsResolver, but this will

// incur some penalty in code space for op implementations that are not

// needed by this graph.

static tflite::*MicroMutableOpResolver* micro\_mutable\_op\_resolver; // NOLINT

micro\_mutable\_op\_resolver.AddBuiltin(

tflite::BuiltinOperator\_DEPTHWISE\_CONV\_2D,

tflite::ops::micro::Register\_DEPTHWISE\_CONV\_2D());

micro\_mutable\_op\_resolver.AddBuiltin(

tflite::BuiltinOperator\_MAX\_POOL\_2D,

tflite::ops::micro::Register\_MAX\_POOL\_2D());

micro\_mutable\_op\_resolver.AddBuiltin(tflite::BuiltinOperator\_CONV\_2D,

tflite::ops::micro::Register\_CONV\_2D());

micro\_mutable\_op\_resolver.AddBuiltin(

tflite::BuiltinOperator\_FULLY\_CONNECTED,

tflite::ops::micro::Register\_FULLY\_CONNECTED());

micro\_mutable\_op\_resolver.AddBuiltin(tflite::BuiltinOperator\_SOFTMAX,

tflite::ops::micro::Register\_SOFTMAX());

// Build an interpreter to run the model with

static tflite::*MicroInterpreter* static\_interpreter(

model, micro\_mutable\_op\_resolver, tensor\_arena, kTensorArenaSize,

error\_reporter);

interpreter = &static\_interpreter;

// Allocate memory from the tensor\_arena for the model's tensors

interpreter->AllocateTensors();

// Obtain pointer to the model's input tensor

model\_input = interpreter->input(0);

if ((model\_input->dims->size != 4) || (model\_input->dims->data[0] != 1) ||

(model\_input->dims->data[1] != 128) ||

(model\_input->dims->data[2] != kChannelNumber) ||

(model\_input->type != kTfLiteFloat32)) {

error\_reporter->Report("Bad input tensor parameters in model");

return;

}

input\_length = model\_input->bytes / sizeof(float);

*TfLiteStatus* setup\_status = SetupAccelerometer(error\_reporter);

if (setup\_status != kTfLiteOk) {

error\_reporter->Report("Set up failed\n");

}

else {

//error\_reporter->Report("Magic starts!\n");

*String* magicstr = R"(

\*엄마 탐지 레이더 시작!!\*)";

error\_reporter->Report(magicstr.c\_str());

error\_reporter->Report("\r\nPredicted gestures:\n\r");

}

}

void loop()

{

// Attempt to read new data from the accelerometer

bool got\_data = ReadAccelerometer(error\_reporter, model\_input->data.f,

input\_length, should\_clear\_buffer);

if (should\_clear\_buffer) {

error\_reporter->Report("\r\nPredicted gestures:\n\r");

}

// Don't try to clear the buffer again

should\_clear\_buffer = false;

// If there was no new data, wait until next time

if (!got\_data) {

return;

}

// Run inference, and report any error

*TfLiteStatus* invoke\_status = interpreter->Invoke();

if (invoke\_status != kTfLiteOk) {

error\_reporter->Report("Invoke failed on index: %d\n", begin\_index);

return;

}

// Analyze the results to obtain a prediction

int gesture\_index = PredictGesture(interpreter->output(0)->data.f);

// Clear the buffer next time we read data

should\_clear\_buffer = gesture\_index < 3;

// Produce an output

HandleOutput(error\_reporter, gesture\_index);

}

Final\_Project\_main.cpp파일은 TensorFlow lite에서 제공하는 데모용 코드를 git에서 clone 하여 다양한 동작 학습 데이터를 바탕으로 활용한 것이다. Git에서 Arduino nano 33 ble sense로 이미 측정된 \*사람이 다가올때의 소리나 \*사람이 다가올 때의 거리 변화 등이 학습된 데이터 셋을 다운로드 하여 Colab의 데모용 코드에 학습을 시켰다. ( 시간이 없었던 이유도 있고 수집하는데 다양한 이상치들이 존재할 수 있었기 때문에 직접 측정하지 않았고 대량의 데이터를 다운받았다.)

이 3개의 파일 외에도 의존성 트리를 만드는데 필요한 파일은 방대해서 다 서술하기에는 한계점이 존재한다. 또한 기존의 오픈소스를 이용해서 코딩을 한 것이기 때문에 주요 코드 외에는 이해하기 힘든 부분도 있었다.

* **Platform IO Dependency Tree. (참고용)**

├── README.md

├── docs

│ ├── IMG\_1736.jpg

│ └── timl\_1109.png

├── include

│ └── README

├── lib

│ ├── Arduino\_LSM9DS1

│ │ ├── CHANGELOG

│ │ ├── README.adoc

│ │ ├── examples

│ │ │ ├── SimpleAccelerometer

│ │ │ │ └── SimpleAccelerometer.ino

│ │ │ ├── SimpleGyroscope

│ │ │ │ └── SimpleGyroscope.ino

│ │ │ └── SimpleMagnetometer

│ │ │ └── SimpleMagnetometer.ino

│ │ ├── keywords.txt

│ │ ├── library.properties

│ │ └── src

│ │ ├── Arduino\_LSM9DS1.h

│ │ ├── LSM9DS1.cpp

│ │ └── LSM9DS1.h

│ ├── Arduino\_TensorFlowLite

│ │ ├── LICENSE

│ │ ├── library.properties

│ │ └── src

│ │ ├── TensorFlowLite.h

│ │ ├── tensorflow

│ │ │ ├── core

│ │ │ │ └── public

│ │ │ │ └── version.h

│ │ │ └── lite

│ │ │ ├── c

│ │ │ │ ├── builtin\_op\_data.h

│ │ │ │ ├── common.c

│ │ │ │ └── common.h

│ │ │ ├── core

│ │ │ │ └── api

│ │ │ │ ├── error\_reporter.cpp

│ │ │ │ ├── error\_reporter.h

│ │ │ │ ├── flatbuffer\_conversions.cpp

│ │ │ │ ├── flatbuffer\_conversions.h

│ │ │ │ ├── op\_resolver.cpp

│ │ │ │ ├── op\_resolver.h

│ │ │ │ ├── tensor\_utils.cpp

│ │ │ │ └── tensor\_utils.h

│ │ │ ├── kernels

│ │ │ │ ├── internal

│ │ │ │ │ ├── common.h

│ │ │ │ │ ├── compatibility.h

│ │ │ │ │ ├── optimized

│ │ │ │ │ │ └── neon\_check.h

│ │ │ │ │ ├── quantization\_util.cpp

│ │ │ │ │ ├── quantization\_util.h

│ │ │ │ │ ├── reference

│ │ │ │ │ │ ├── add.h

│ │ │ │ │ │ ├── arg\_min\_max.h

│ │ │ │ │ │ ├── binary\_function.h

│ │ │ │ │ │ ├── ceil.h

│ │ │ │ │ │ ├── comparisons.h

│ │ │ │ │ │ ├── concatenation.h

│ │ │ │ │ │ ├── conv.h

│ │ │ │ │ │ ├── depthwiseconv\_float.h

│ │ │ │ │ │ ├── depthwiseconv\_uint8.h

│ │ │ │ │ │ ├── dequantize.h

│ │ │ │ │ │ ├── floor.h

│ │ │ │ │ │ ├── fully\_connected.h

│ │ │ │ │ │ ├── integer\_ops

│ │ │ │ │ │ │ ├── add.h

│ │ │ │ │ │ │ ├── conv.h

│ │ │ │ │ │ │ ├── depthwise\_conv.h

│ │ │ │ │ │ │ ├── fully\_connected.h

│ │ │ │ │ │ │ ├── mul.h

│ │ │ │ │ │ │ ├── pooling.h

│ │ │ │ │ │ │ └── softmax.h

│ │ │ │ │ │ ├── logistic.h

│ │ │ │ │ │ ├── maximum\_minimum.h

│ │ │ │ │ │ ├── mul.h

│ │ │ │ │ │ ├── neg.h

│ │ │ │ │ │ ├── pad.h

│ │ │ │ │ │ ├── pooling.h

│ │ │ │ │ │ ├── prelu.h

│ │ │ │ │ │ ├── process\_broadcast\_shapes.h

│ │ │ │ │ │ ├── quantize.h

│ │ │ │ │ │ ├── round.h

│ │ │ │ │ │ ├── softmax.h

│ │ │ │ │ │ └── strided\_slice.h

│ │ │ │ │ ├── round.h

│ │ │ │ │ ├── scoped\_profiling\_label\_wrapper.h

│ │ │ │ │ ├── strided\_slice\_logic.h

│ │ │ │ │ ├── tensor.h

│ │ │ │ │ ├── tensor\_ctypes.h

│ │ │ │ │ └── types.h

│ │ │ │ ├── kernel\_util.cpp

│ │ │ │ ├── kernel\_util.h

│ │ │ │ ├── op\_macros.h

│ │ │ │ └── padding.h

│ │ │ ├── micro

│ │ │ │ ├── arduino

│ │ │ │ │ └── debug\_log.cpp

│ │ │ │ ├── compatibility.h

│ │ │ │ ├── debug\_log.h

│ │ │ │ ├── debug\_log\_numbers.cpp

│ │ │ │ ├── debug\_log\_numbers.h

│ │ │ │ ├── kernels

│ │ │ │ │ ├── activation\_utils.h

│ │ │ │ │ ├── activations.cpp

│ │ │ │ │ ├── add.cpp

│ │ │ │ │ ├── all\_ops\_resolver.cpp

│ │ │ │ │ ├── all\_ops\_resolver.h

│ │ │ │ │ ├── arg\_min\_max.cpp

│ │ │ │ │ ├── ceil.cpp

│ │ │ │ │ ├── comparisons.cpp

│ │ │ │ │ ├── concatenation.cpp

│ │ │ │ │ ├── conv.cpp

│ │ │ │ │ ├── dequantize.cpp

│ │ │ │ │ ├── elementwise.cpp

│ │ │ │ │ ├── floor.cpp

│ │ │ │ │ ├── fully\_connected.cpp

│ │ │ │ │ ├── logical.cpp

│ │ │ │ │ ├── logistic.cpp

│ │ │ │ │ ├── maximum\_minimum.cpp

│ │ │ │ │ ├── micro\_ops.h

│ │ │ │ │ ├── micro\_utils.h

│ │ │ │ │ ├── mul.cpp

│ │ │ │ │ ├── neg.cpp

│ │ │ │ │ ├── pack.cpp

│ │ │ │ │ ├── pad.cpp

│ │ │ │ │ ├── pooling.cpp

│ │ │ │ │ ├── portable\_optimized

│ │ │ │ │ │ └── depthwise\_conv.cpp

│ │ │ │ │ ├── prelu.cpp

│ │ │ │ │ ├── quantize.cpp

│ │ │ │ │ ├── reshape.cpp

│ │ │ │ │ ├── round.cpp

│ │ │ │ │ ├── softmax.cpp

│ │ │ │ │ ├── split.cpp

│ │ │ │ │ ├── strided\_slice.cpp

│ │ │ │ │ ├── svdf.cpp

│ │ │ │ │ └── unpack.cpp

│ │ │ │ ├── memory\_helpers.cpp

│ │ │ │ ├── memory\_helpers.h

│ │ │ │ ├── memory\_planner

│ │ │ │ │ ├── greedy\_memory\_planner.cpp

│ │ │ │ │ ├── greedy\_memory\_planner.h

│ │ │ │ │ ├── linear\_memory\_planner.cpp

│ │ │ │ │ ├── linear\_memory\_planner.h

│ │ │ │ │ └── memory\_planner.h

│ │ │ │ ├── micro\_allocator.cpp

│ │ │ │ ├── micro\_allocator.h

│ │ │ │ ├── micro\_error\_reporter.cpp

│ │ │ │ ├── micro\_error\_reporter.h

│ │ │ │ ├── micro\_interpreter.cpp

│ │ │ │ ├── micro\_interpreter.h

│ │ │ │ ├── micro\_mutable\_op\_resolver.cpp

│ │ │ │ ├── micro\_mutable\_op\_resolver.h

│ │ │ │ ├── micro\_optional\_debug\_tools.cpp

│ │ │ │ ├── micro\_optional\_debug\_tools.h

│ │ │ │ ├── micro\_utils.cpp

│ │ │ │ ├── micro\_utils.h

│ │ │ │ ├── simple\_memory\_allocator.cpp

│ │ │ │ ├── simple\_memory\_allocator.h

│ │ │ │ ├── test\_helpers.cpp

│ │ │ │ ├── test\_helpers.h

│ │ │ │ └── testing

│ │ │ │ ├── micro\_test.h

│ │ │ │ └── test\_utils.h

│ │ │ ├── schema

│ │ │ │ └── schema\_generated.h

│ │ │ ├── string\_type.h

│ │ │ ├── string\_util.h

│ │ │ ├── type\_to\_tflitetype.h

│ │ │ └── version.h

│ │ └── third\_party

│ │ ├── flatbuffers

│ │ │ ├── LICENSE.txt

│ │ │ └── include

│ │ │ └── flatbuffers

│ │ │ ├── base.h

│ │ │ ├── flatbuffers.h

│ │ │ └── stl\_emulation.h

│ │ └── gemmlowp

│ │ ├── LICENSE

│ │ ├── fixedpoint

│ │ │ ├── fixedpoint.h

│ │ │ └── fixedpoint\_sse.h

│ │ └── internal

│ │ └── detect\_platform.h

│ └── README

├── platformio.ini

├── src

│ ├── accelerometer\_handler.cpp

│ ├── accelerometer\_handler.h

│ ├── constants.cpp

│ ├── constants.h

│ ├── gesture\_predictor.cpp

│ ├── gesture\_predictor.h

│ ├── magic\_wand\_model\_data.cpp

│ ├── magic\_wand\_model\_data.h

│ ├── Final\_Project\_main.cpp

│ ├── output\_handler.cpp

│ ├── output\_handler.h

│ └── xmas\_demo.hpp

│ └── Final\_Project.ino ( cpp )

└── test

└── README

└── Final\_Project.pde