```
#define EIDSP_QUANTIZE_FILTERBANK   0
#define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW    1
#define EI_CLASSIFIER_SLICE_SIZE (EI_CLASSIFIER_RAW_SAMPLE_COUNT /
EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)

#include <speech_modelR_inferencing.h>
#include <PDM.h>
#include "endR_words.h"

#include <Adafruit_GFX.h>    // Core graphics library
#include <Adafruit_ST7735.h> // Hardware-specific library for ST7735
#include <Adafruit_ST7789.h> // Hardware-specific library for ST7789
#include <SPI.h>

 #define TFT_CS        10
 #define TFT_RST        8 // Or set to -1 and connect to Arduino RESET pin
 #define TFT_DC         9

#define TFT_MOSI 11  // Data out
#define TFT_SCLK 13  // Clock out

// For ST7735-based displays, we will use this call
Adafruit_ST7735 tft = Adafruit_ST7735(TFT_CS, TFT_DC, TFT_MOSI, TFT_SCLK,
TFT_RST);

/** Audio buffers, pointers and selectors */
typedef struct {
   signed short *buffers[2];
   unsigned char buf_select;
   unsigned char buf_ready;
   unsigned int buf_count;
   unsigned int n_samples;
} inference_t;

static inference_t inference;
static volatile bool record_ready = false;
// static signed short *sampleBuffer;
static signed short sampleBuffer[2048];
static bool debug_nn = false; // Set this to true to see e.g. features
generated from the raw signal
```

```
static int print_results = -(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW);

//variables used
float max_value=0;
const char* max_label=0;
int buttonOnPin=2;
int buttonSkipPin=3;
int buttonScorePin=5;
int buttonTrainPin=6;
int buttonOnRead=1;
int buttonSkipRead=1;
int buttonScoreRead=1;
int buttonTrainRead=1;
int curidx=0;
int lenwordlist=5;
int petmode_on=0;
const char* curword = 0;
int ml_correct=0;
int ml_live=0;
bool show_display=false;
int wordmode_on=0;
int wordscreen_cleared=0;
int petscreen_cleared=0;
int point_count=0;

//list of words that represent the labels in the AI
const char* ml_wordlist[5]={"bear", "color", "number", "water", "water"};
//list that tracks the number of words spoken that are detected correctly
int ml_count_right_list[5]={0,0,0,0,0};


/**
* @brief      Arduino setup function
*/
void setup()
{
   // put your setup code here, to run once:
   Serial.begin(9600);
   //sets up the buttons that are connected to microchip
   pinMode(buttonOnPin, INPUT_PULLUP);
   pinMode(buttonSkipPin, INPUT_PULLUP);
   pinMode(buttonScorePin, INPUT_PULLUP);
   pinMode(buttonTrainPin, INPUT_PULLUP);
   tft.initR(INITR_144GREENTAB);

   // clears audio buffer
   run_classifier_init();
   //check if there is enough memory on the microcontroller to inferencing
   if (microphone_inference_start(EI_CLASSIFIER_SLICE_SIZE) == false) {
       ei_printf("ERR: Could not allocate audio buffer (size %d), this
could be due to the window length of your model\r\n",
EI_CLASSIFIER_RAW_SAMPLE_COUNT);
       return;
   }
}

/**
* @brief      Arduino main function. Runs the inferencing loop.
*/
void loop()
{  //read signals from buttons connected to chip
   buttonOnRead=digitalRead(buttonOnPin);
   buttonSkipRead=digitalRead(buttonSkipPin);
   buttonScoreRead=digitalRead(buttonScorePin);
   buttonTrainRead=digitalRead(buttonTrainPin);
   //flag to activate pet mode
   if (buttonOnRead==0) {
     //Condition: on button is pressed
     //Action: keep lcd on
     petmode_on=1;
     wordmode_on=0;
     Serial.println("button on");
   }
   //flag to activate training mode
   if (buttonTrainRead==0) {
     wordmode_on=1;
     petmode_on=0;
   }

   //activates if pet mode is on
```

```
  if (petmode_on==1) {
    Serial.println("petmode_on=1");
    wordmode_on=0;
    petscreen_cleared=0;
    //clear screen before showing pet
    if (wordscreen_cleared==0) {
      tft.fillScreen(ST77XX_BLACK);
      wordscreen_cleared=1;
    }
    //shows the pet
    show_graphic();
    default_face();
    //shows the score
    score_on();
    //value depends on how many positive nonzero numbers are in
    //ml_count_right_list. This number determines which stage of the pet
    //will be shown on screen.
    counter();

    if (point_count==1) {
      if (wordscreen_cleared==0) {
        tft.fillScreen(ST77XX_BLACK);
        wordscreen_cleared=1;
      }
      reward_1_point();
    }
    else if (point_count==2) {
      if (wordscreen_cleared==0) {
        tft.fillScreen(ST77XX_BLACK);
        wordscreen_cleared=1;
      }
        reward_2_point();
    }
    else if (point_count==3) {
      if (wordscreen_cleared==0) {
        tft.fillScreen(ST77XX_BLACK);
        wordscreen_cleared=1;
      }
      reward_3_point();
    }
    else if (point_count==4) {
      if (wordscreen_cleared==0) {
        tft.fillScreen(ST77XX_BLACK);
        wordscreen_cleared=1;
      }
      reward_4_point();
    }
    else if (point_count==5) {
      if (wordscreen_cleared==0) {
        tft.fillScreen(ST77XX_BLACK);
        wordscreen_cleared=1;
      }
      reward_5_point();
    }

  }
  //activates if training mode is on
  if (wordmode_on==1) {
    Serial.println("wordmode_on=1");
    wordscreen_cleared=0;
    petmode_on=0;
    if (petscreen_cleared==0) {
      tft.fillScreen(ST77XX_BLACK);
      petscreen_cleared=1;
    }
    worddisplay();
  }
  //flag to show score breakdown
  if (buttonScoreRead==0) {
    show_display=true;
  }
  //show score breakdown
  if (show_display==true) {
    clearscreen();
    display_trainscore();
    delay(5000);
    clearscreen();
    clearscreen();
    show_display=false;
  }
```

```
   //skip to the next word
   if (buttonSkipRead==0){
     //Condition: button is pressed
     skipword();
     Serial.println("skipped word");
     Serial.println(curidx);
   }
   //starts recording
   bool m = microphone_inference_record();
   //check if the microphone has recorded
   if (!m) {
       ei_printf("ERR: Failed to record audio...\n");
       return;
   }

   signal_t signal;
   signal.total_length = EI_CLASSIFIER_SLICE_SIZE; //size of buffer
   signal.get_data = &microphone_audio_signal_get_data; //get data from
   //buffer
   ei_impulse_result_t result = {0}; //store result here

   EI_IMPULSE_ERROR res = run_classifier_continuous(&signal, &result,
debug_nn); //continuously classify data
   if (res != EI_IMPULSE_OK) {
       ei_printf("ERR: Failed to run classifier (%d)\n", res);
       return;
   }
   // read and process the result once all slices have been collected
   if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)) {
       // print inference return code
       ei_printf("run_classifier returned: %d\r\n", res);
       print_inference_result(result); //print results
       get_prediction(result); //get the prediction from the result
       set_cur_wordlist(); //set the current word as the current index
      //condition to increment score
       if (wordmode_on==1) {
         increment_traincount();
       }
       print_results = 0;
```

```
    }
}
//add to point_count for every item in ml_count_right list that is greater
than 1
void counter() {
 point_count=0;
 for (int i=0; i<lenwordlist; i+=1) {
   if (ml_count_right_list[i]>0) {
     point_count+=1;
   }
 }
}
//clearscreen on the tft display
void clearscreen() {
 tft.fillScreen(ST77XX_BLACK);
}

//move onto the next word
void skipword(){
 if (curidx==(lenwordlist-1)){
   //condition: current i = length of wordlist-1
   //action: reset the current i to 0
   curidx=0;
   clearscreen();
 }
 else{
   //condition: current i != lengeth of wordlist-1
   //action: go onto the next index
   curidx+=1;
   clearscreen();
 }
}

//display words on the tft screen
void worddisplay(){
 //Method: show word based on current i
 if (curidx==0){
   tftBear();
 }
 else if (curidx==1){
   tftColor();
 }
 else if (curidx==2){
   tftNumber();
 }
 else if (curidx==3){
   tftWater();
 }
 else if (curidx==4){
   tftYear();
 }
}

//display the word bear and its respective image
void tftBear(){
 tft.setTextWrap(false);
 tft.setCursor(0, 0);
 tft.setTextSize(2);
 tft.println("bear");
 tft.drawBitmap(30, 30, bear_bmp, 64, 64, ST77XX_BLUE);

}

//display the word color and its respective image
void tftColor(){
 tft.setTextWrap(false);
 tft.setCursor(0, 0);
 tft.setTextSize(2);
 tft.println("color");
 tft.drawCircle(30, 60, 15, ST77XX_RED);
 tft.drawCircle(40, 60, 15, ST77XX_ORANGE);
 tft.drawCircle(50, 60, 15, ST77XX_YELLOW);
 tft.drawCircle(60, 60, 15, ST77XX_GREEN);
 tft.drawCircle(70, 60, 15, ST77XX_CYAN);
 tft.drawCircle(80, 60, 15, ST77XX_BLUE);
 tft.drawCircle(90, 60, 15, ST77XX_MAGENTA);
}

//display the word number and its respective image
void tftNumber() {
 tft.setTextWrap(false);
 tft.setCursor(0, 0);
 tft.setTextSize(2);
 tft.println("number");
 tft.setCursor(40, 60);
 tft.println("1 2 3");
}

//display the word water and its respective image
void tftWater(){
 tft.setTextWrap(false);
 tft.setCursor(0, 0);
 tft.setTextSize(2);
 tft.println("water");
 tft.drawBitmap(30, 30, water_bmp, 64, 64, ST77XX_BLUE);
}

//display the word year and its respective image
void tftYear() {
 tft.setTextWrap(false);
 tft.setCursor(0, 0);
 tft.setTextSize(2);
 tft.println("year");
 tft.drawBitmap(30, 30, year_bmp, 64, 64, ST77XX_BLUE);
}

//get the label that the model predicted
void get_prediction(ei_impulse_result_t result){
 max_label=0;
 max_value=0;
 for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
   if (result.classification[i].value>max_value) {
     max_value=result.classification[i].value;
     max_label=ei_classifier_inferencing_categories[i];
   }
 }
 ei_printf("\n  %s: ", max_label);
 ei_printf("%.5f\r\n", max_value);
}
```

```
//get the current word that is on training mode
void set_cur_wordlist() {
 curword=ml_wordlist[curidx];
 ei_printf("\n  %d  ", curidx);
 ei_printf("\n %s  ", curword);
}

// show the total score and score per word
void display_trainscore() {
 tft.setTextWrap(false);
 tft.setCursor(0, 0);
 tft.setTextSize(2);
 tft.println("Training");
 tft.print(ml_correct);
 tft.println(" correct");
 tft.print(ml_count_right_list[0]);
 tft.print(" ");
 tft.println(ml_wordlist[0]);
 tft.print(ml_count_right_list[1]);
 tft.print(" ");
 tft.println(ml_wordlist[1]);
 tft.print(ml_count_right_list[2]);
 tft.print(" ");
 tft.println(ml_wordlist[2]);
 tft.print(ml_count_right_list[3]);
 tft.print(" ");
 tft.println(ml_wordlist[3]);
 tft.print(ml_count_right_list[4]);
 tft.print(" ");
 tft.println(ml_wordlist[4]);
}

//show the total score when the pet is on the screen
void score_on() {
 tft.setTextWrap(false);
 tft.setCursor(0, 0);
 tft.setTextSize(2);
 tft.println(ml_correct);
}
```

```
//increase the score when a word spoken matches the word on the screen
void increment_traincount() {
 if (max_label==curword) {
   ml_count_right_list[curidx]+=1;
   ml_correct+=1;
   clearscreen();
   tft.setTextWrap(false);
   tft.setTextColor(ST77XX_GREEN);
   tft.setCursor(0, 0);
   tft.setTextSize(2);
   tft.println("Correct");
   delay(2000);
   clearscreen();
   tft.setTextColor(ST77XX_WHITE);
 }
 Serial.print("ml_correct=");
 Serial.println(ml_correct);
}

//show body of first stage
void show_graphic() {
 tft.drawRoundRect(30, 40, 70, 50, 80, ST77XX_BLUE);
 //tft.fillRoundRect(30, 40, 70, 50, 80, ST77XX_BLUE);
}

//show eyes of first stage
void default_face() {
 tft.fillCircle(60, 65, 7, ST77XX_BLUE);
 tft.fillCircle(80, 65, 7, ST77XX_BLUE);
}

//show 2nd stage of pet
void reward_1_point() {
 tft.fillRoundRect(30, 40, 70, 50, 80, ST77XX_BLUE);
 tft.fillCircle(60, 65, 7, ST77XX_BLACK);
 tft.fillCircle(80, 65, 7, ST77XX_BLACK);
}

//show 3rd stage of pet
void reward_2_point() {
 reward_1_point();
 tft.drawFastVLine(55, 80, 35, ST77XX_BLUE);
 tft.drawFastVLine(80, 80, 35, ST77XX_BLUE);
}

//show 4th stage of pet
void reward_3_point() {
 reward_2_point();
 tft.drawFastHLine(20, 80, 35, ST77XX_BLUE);
 tft.drawFastHLine(80, 80, 35, ST77XX_BLUE);
}
//show 5th stage of pet
void reward_4_point() {
 reward_3_point();
 tft.fillTriangle(50, 40, 60, 40, 55, 20, ST77XX_GREEN);
}

//show the pet when fully evolved
void reward_5_point() {
 reward_4_point();
 tft.fillTriangle(70, 40, 80, 40, 75, 20, ST77XX_GREEN);
}
```