Edge Computing Lab

Class: TY-AIEC

School of Computing, MIT Art Design Technology University

Academic Year: 2024-25

Experiment No. 9

Introduction

The Object Detection using Camera on Edge Computing Devices

Objective: Build a project to detect an object using Edge Computing

Tasks:

- Generate the dataset for customized object
- Configure Edge Impulse for Object Detection
- Building and Training a Model
- Deploy on Edge Computing Device

Introduction

Edge Impulse is a development platform for machine learning on edge devices, targeted at developers who want to create intelligent device solutions. The " Camera "sensor reading equivalent in Edge Impulse would typically involve creating a simple machine learning model that can run on an edge device, like classifying sensor data or recognizing a basic pattern.

Materials Required

- Nano BLE Sense Board

Theory

GPIO (General Purpose Input/Output) pins on the Raspberry Pi are used for interfacing with other electronic components. BCM numbering refers to the pin numbers in the Broadcom SOC channel, which is a more consistent way to refer to the GPIO pins across different versions of the

Here's a high-level overview of steps you'd follow to create a "Hello World" project on Edge Impulse:

Steps to Configure the Edge Impulse:

1. Create an Account and New Project:
    - Sign up for an Edge Impulse account.
    - Create a new project from the dashboard.
2. Connect a Device:

- You can use a supported development board or your smartphone as a sensor device.
- Follow the instructions to connect your device to your Edge Impulse project.

3. Collect Data:

   - Use the Edge Impulse mobile app or the Web interface to collect data from the onboard sensors.
   - For a "Hello World" project, you could collect accelerometer data, for instance.

4. Create an Impulse:

   - Go to the 'Create impulse' page.
   - Add a processing block (e.g., time-series data) and a learning block (e.g., classification).
   - Save the impulse, which defines the machine learning pipeline.

5. Design a Neural Network:

   - Navigate to the 'NN Classifier' under the 'Learning blocks'.
   - Design a simple neural network. Edge Impulse provides a default architecture that works well for most basic tasks.

6. Train the Model:

   - Click on the 'Start training' button to train your machine learning model with the collected data.

7. Test the Model:

   - Once the model is trained, you can test its performance with new data in the 'Model Testing' tab.

8. Deploy the Model:

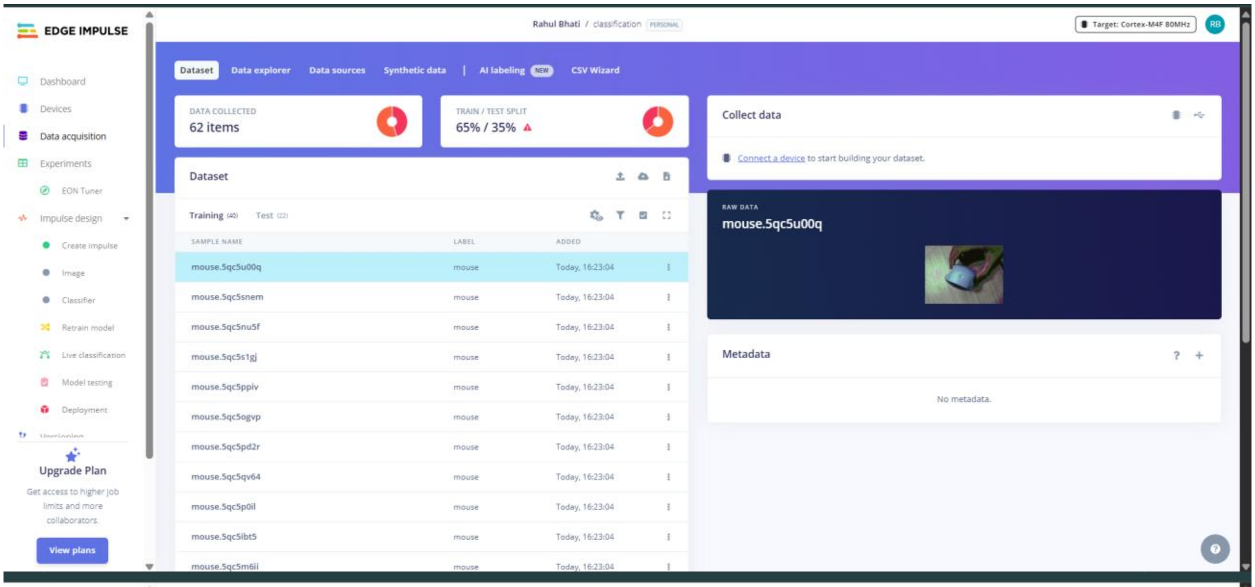
   - Go to the 'Deployment' tab.
   - Select the deployment method that suits your edge device (e.g., Arduino library, WebAssembly, container, etc.).
   - Follow the instructions to deploy the model to your device.

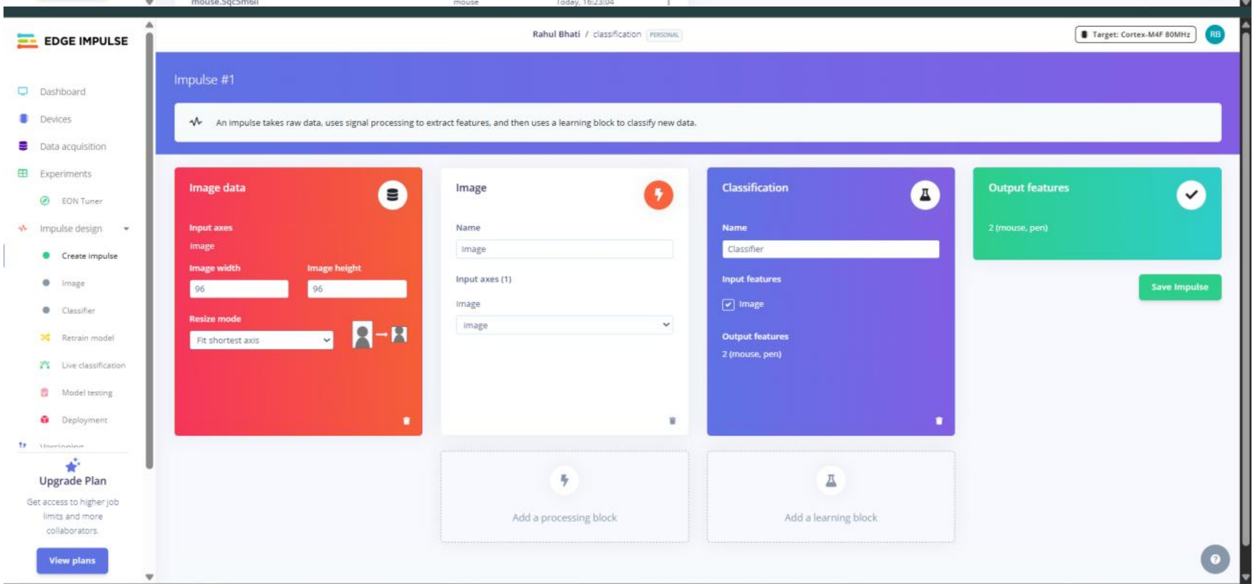
9. Run Inference:

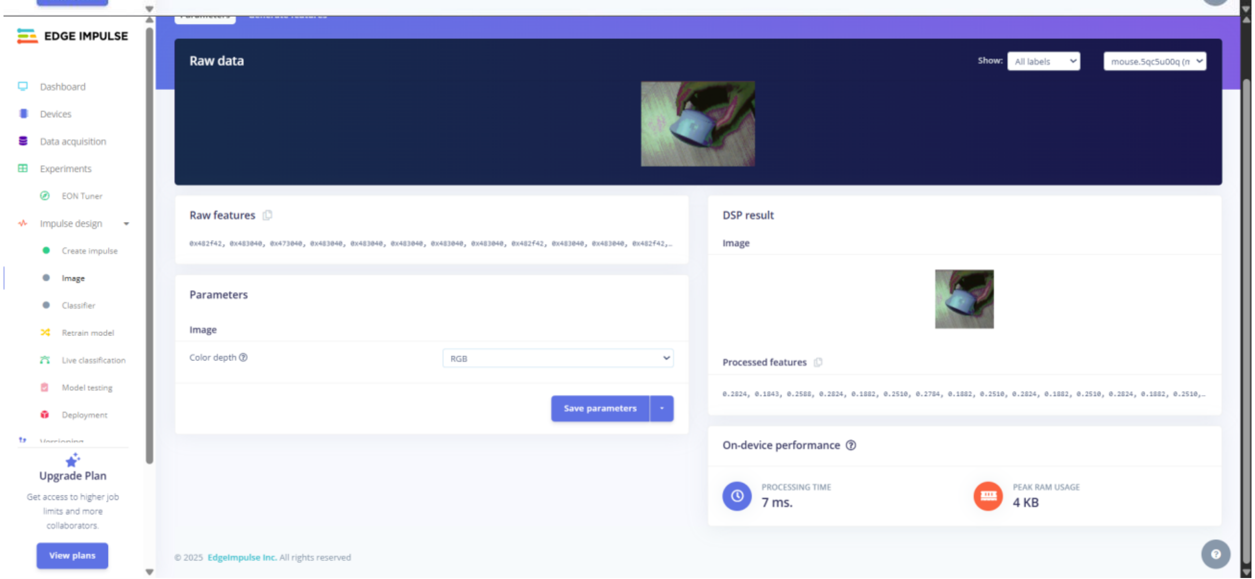
- With the model deployed, run inference on the edge device to see it classifying data in real-time.

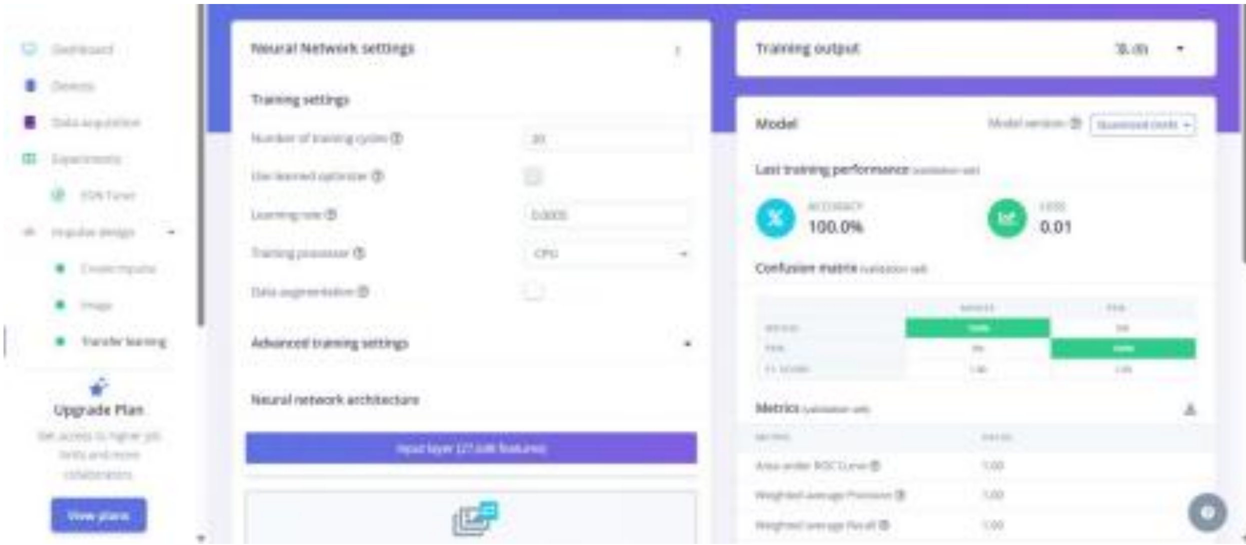
10. Monitor:

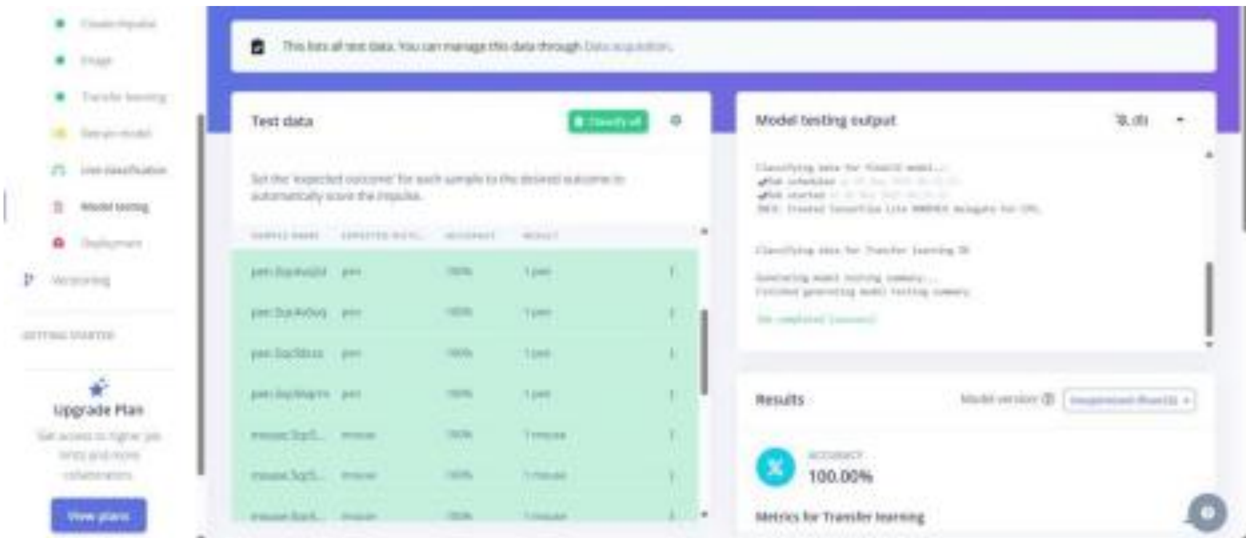
- You can monitor the performance of your device through the Edge Impulse studio.


EDGE IMPULSE
Rahul Bhati / classification PERSONAL
Target: Cortex-M4F 80MHz
Dashboard
Devices
Data acquisition
Experiments
EON Tuner
Impulse design
Create impulse
Image
Classifier
Retrain model
Live classification
Model testing
Deployment
Upgrade Plan
Get access to higher job limits and more collaborators.
View plans
Dataset
Data explorer
Data sources
Synthetic data
AI labeling
CSV Wizard
DATA COLLECTED
62 items
TRAIN / TEST SPLIT
65% / 35%
Collect data
Connect a device to start building your dataset.
Dataset
Training
Test
SAMPLE NAME
LABEL
ADDED
mouse.5qc5u00q mouse Today, 16:23:04
mouse.5qc5snem mouse Today, 16:23:04
mouse.5qc5nu5f mouse Today, 16:23:04
mouse.5qc5s1gj mouse Today, 16:23:04
mouse.5qc5ppiv mouse Today, 16:23:04
mouse.5qc5ogvp mouse Today, 16:23:04
mouse.5qc5pd2r mouse Today, 16:23:04
mouse.5qc5qv64 mouse Today, 16:23:04
mouse.5qc5p0il mouse Today, 16:23:04
mouse.5qc5ibt5 mouse Today, 16:23:04
RAW DATA
mouse.5qc5u00q
Metadata
No metadata.


EDGE IMPULSE
Rahul Bhati / classification PERSONAL
Target: Cortex-M4F 80MHz
Impulse #1
An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.
Image data
Input axes
image
Image width
96
Image height
96
Resize mode
Fit shortest axis
Image
Name
Image
Input axes (1)
image
Classification
Name
Classifier
Input features
Image
Output features
2 (mouse, pen)
Output features
2 (mouse, pen)
Save Impulse
Add a processing block
Add a learning block
Upgrade Plan
Get access to higher job limits and more collaborators.
View plans


EDGE IMPULSE
Raw data
Show:
All labels
Raw features
DSP result
Image
Parameters
Image
Color depth
RGB
Save parameters
Processed features
On-device performance
PROCESSING TIME
7 ms.
PEAK RAM USAGE
4 KB
Upgrade Plan
Get access to higher job limits and more collaborators.
View plans
© 2025 EdgeImpulse Inc. All rights reserved





```
/* Includes ---------------------------------------------------------------- */
#include <Rj_bhati-project-1_inferencing.h>
#include <Arduino_OV767X.h> //Click here to get the library: https://www.arduino.cc/reference/en/libraries/arduino_ov767x/

#include <stdint.h>
#include <stdlib.h>

/* Constant variables ------------------------------------------------------- */
#define EI_CAMERA_RAW_FRAME_BUFFER_COLS     160
#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS     120

#define DWORD_ALIGN_PTR(a)   ((a & 0x3) ?(((uintptr_t)a + 0x4) & ~(uintptr_t)0x3) : a)

/*
 ** NOTE: If you run into TFLite arena allocation issue.
 **
 ** This may be due to may dynamic memory fragmentation.
 ** Try defining "-DEI_CLASSIFIER_ALLOCATION_STATIC" in boards.local.txt (create
 ** if it doesn't exist) and copy this file to
 ** `<ARDUINO_CORE_INSTALL_PATH>/arduino/hardware/<mbed_core>/<core_version>/`.
 **
 ** See
 ** (https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-the-installed-cores-located-)
 ** to find where Arduino installs cores on your machine.
 **
 ** If the problem persists then there's not enough memory for this model and application.
 */

/* Edge Impulse ------------------------------------------------------------- */
class OV7675 : public OV767X {
    public:
        int begin(int resolution, int format, int fps);
        void readFrame(void* buffer);

    private:
```

Code:

/* Includes -- */

#include <Rj_bhati-project-1_inferencing.h>

```
#include <Arduino_OV767X.h> //Click here to get the library:
https://www.arduino.cc/reference/en/libraries/arduino_ov767x/


#include <stdint.h>

#include <stdlib.h>


/* Constant variables ------------------------------------------------------- */

#define EI_CAMERA_RAW_FRAME_BUFFER_COLS     160

#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS     120


#define DWORD_ALIGN_PTR(a)   ((a & 0x3) ?(((uintptr_t)a + 0x4) &
~(uintptr_t)0x3) : a)


/*

 ** NOTE: If you run into TFLite arena allocation issue.

 **

 ** This may be due to may dynamic memory fragmentation.

 ** Try defining "-DEI_CLASSIFIER_ALLOCATION_STATIC" in boards.local.txt (create

 ** if it doesn't exist) and copy this file to

 **
`<ARDUINO_CORE_INSTALL_PATH>/arduino/hardware/<mbed_core>/<core_versi
on>/`.

 **

 ** See

 ** (https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-the-
installed-cores-located-)

 ** to find where Arduino installs cores on your machine.

 **

 ** If the problem persists then there's not enough memory for this model and
application.

 */


/* Edge Impulse ------------------------------------------------------------- */
```

```
class OV7675 : public OV767X {
  public:
    int begin(int resolution, int format, int fps);
    void readFrame(void* buffer);

  private:
    int vsyncPin;
    int hrefPin;
    int pclkPin;
    int xclkPin;

    volatile uint32_t* vsyncPort;
    uint32_t vsyncMask;
    volatile uint32_t* hrefPort;
    uint32_t hrefMask;
    volatile uint32_t* pclkPort;
    uint32_t pclkMask;

    uint16_t width;
    uint16_t height;
    uint8_t bytes_per_pixel;
    uint16_t bytes_per_row;
    uint8_t buf_rows;
    uint16_t buf_size;
    uint8_t resize_height;
    uint8_t *raw_buf;
    void *buf_mem;
    uint8_t *intrp_buf;
    uint8_t *buf_limit;

    void readBuf();
```

```
    int allocate_scratch_buffs();
    int deallocate_scratch_buffs();
};

typedef struct {
  size_t width;
  size_t height;
} ei_device_resize_resolutions_t;

/**
 * @brief      Check if new serial data is available
 *
 * @return     Returns number of available bytes
 */
int ei_get_serial_available(void) {
  return Serial.available();
}

/**
 * @brief      Get next available byte
 *
 * @return     byte
 */
char ei_get_serial_byte(void) {
  return Serial.read();
}

/* Private variables ------------------------------------------------------- */
static OV7675 Cam;
static bool is_initialised = false;
```

```
/*
** @brief points to the output of the capture
*/
static uint8_t *ei_camera_capture_out = NULL;
uint32_t resize_col_sz;
uint32_t resize_row_sz;
bool do_resize = false;
bool do_crop = false;

static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal

/* Function definitions ------------------------------------------------------- */
bool ei_camera_init(void);
void ei_camera_deinit(void);
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf) ;
int calculate_resize_dimensions(uint32_t out_width, uint32_t out_height, uint32_t *resize_col_sz, uint32_t *resize_row_sz, bool *do_resize);
void resizeImage(int srcWidth, int srcHeight, uint8_t *srcImage, int dstWidth, int dstHeight, uint8_t *dstImage, int iBpp);
void cropImage(int srcWidth, int srcHeight, uint8_t *srcImage, int startX, int startY, int dstWidth, int dstHeight, uint8_t *dstImage, int iBpp);

/**
* @brief     Arduino setup function
*/
void setup()
{
  // put your setup code here, to run once:
  Serial.begin(115200);
  // comment out the below line to cancel the wait for USB connection (needed for native USB)
  while (!Serial);
```

```
  Serial.println("Edge Impulse Inferencing Demo");


  // summary of inferencing settings (from model_metadata.h)
  ei_printf("Inferencing settings:\n");
  ei_printf("\tImage resolution: %dx%d\n", EI_CLASSIFIER_INPUT_WIDTH,
EI_CLASSIFIER_INPUT_HEIGHT);
  ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
  ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories) /
sizeof(ei_classifier_inferencing_categories[0]));
}


/**
* @brief      Get data and run inferencing
*
* @param[in]  debug  Get debug info if true
*/
void loop()
{
  bool stop_inferencing = false;


  while(stop_inferencing == false) {
    ei_printf("\nStarting inferencing in 2 seconds...\n");


    // instead of wait_ms, we'll wait on the signal, this allows threads to cancel us...
    if (ei_sleep(2000) != EI_IMPULSE_OK) {
      break;
    }


    ei_printf("Taking photo...\n");


    if (ei_camera_init() == false) {
      ei_printf("ERR: Failed to initialize image sensor\r\n");
```

```
        break;
    }

    // choose resize dimensions
    uint32_t resize_col_sz;
    uint32_t resize_row_sz;
    bool do_resize = false;
    int res = calculate_resize_dimensions(EI_CLASSIFIER_INPUT_WIDTH,
EI_CLASSIFIER_INPUT_HEIGHT, &resize_col_sz, &resize_row_sz, &do_resize);
    if (res) {
        ei_printf("ERR: Failed to calculate resize dimensions (%d)\r\n", res);
        break;
    }

    void *snapshot_mem = NULL;
    uint8_t *snapshot_buf = NULL;
    snapshot_mem = ei_malloc(resize_col_sz*resize_row_sz*2);
    if(snapshot_mem == NULL) {
        ei_printf("failed to create snapshot_mem\r\n");
        break;
    }
    snapshot_buf = (uint8_t *)DWORD_ALIGN_PTR((uintptr_t)snapshot_mem);

    if (ei_camera_capture(EI_CLASSIFIER_INPUT_WIDTH,
EI_CLASSIFIER_INPUT_HEIGHT, snapshot_buf) == false) {
        ei_printf("Failed to capture image\r\n");
        if (snapshot_mem) ei_free(snapshot_mem);
        break;
    }

    ei::signal_t signal;
```

```
        signal.total_length = EI_CLASSIFIER_INPUT_WIDTH *
EI_CLASSIFIER_INPUT_HEIGHT;
        signal.get_data = &ei_camera_cutout_get_data;


        // run the impulse: DSP, neural network and the Anomaly algorithm
        ei_impulse_result_t result = { 0 };


        EI_IMPULSE_ERROR ei_error = run_classifier(&signal, &result, debug_nn);
        if (ei_error != EI_IMPULSE_OK) {
            ei_printf("Failed to run impulse (%d)\n", ei_error);
            ei_free(snapshot_mem);
            break;
        }


        // print the predictions
        ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.):
\n",
                result.timing.dsp, result.timing.classification, result.timing.anomaly);
#if EI_CLASSIFIER_OBJECT_DETECTION == 1
        ei_printf("Object detection bounding boxes:\r\n");
        for (uint32_t i = 0; i < result.bounding_boxes_count; i++) {
            ei_impulse_result_bounding_box_t bb = result.bounding_boxes[i];
            if (bb.value == 0) {
                continue;
            }
            ei_printf("  %s (%f) [ x: %u, y: %u, width: %u, height: %u ]\r\n",
                    bb.label,
                    bb.value,
                    bb.x,
                    bb.y,
                    bb.width,
                    bb.height);
```

```
        }

    // Print the prediction results (classification)
#else
        ei_printf("Predictions:\r\n");
        for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
            ei_printf("  %s: ", ei_classifier_inferencing_categories[i]);
            ei_printf("%.5f\r\n", result.classification[i].value);
        }
#endif

    // Print anomaly result (if it exists)
#if EI_CLASSIFIER_HAS_ANOMALY
        ei_printf("Anomaly prediction: %.3f\r\n", result.anomaly);
#endif

#if EI_CLASSIFIER_HAS_VISUAL_ANOMALY
        ei_printf("Visual anomalies:\r\n");
        for (uint32_t i = 0; i < result.visual_ad_count; i++) {
            ei_impulse_result_bounding_box_t bb = result.visual_ad_grid_cells[i];
            if (bb.value == 0) {
                continue;
            }
            ei_printf("  %s (%f) [ x: %u, y: %u, width: %u, height: %u ]\r\n",
                    bb.label,
                    bb.value,
                    bb.x,
                    bb.y,
                    bb.width,
                    bb.height);
        }
```

```
#endif

        while (ei_get_serial_available() > 0) {
            if (ei_get_serial_byte() == 'b') {
                ei_printf("Inferencing stopped by user\r\n");
                stop_inferencing = true;
            }
        }
        if (snapshot_mem) ei_free(snapshot_mem);
    }
    ei_camera_deinit();
}

/**
 * @brief      Determine whether to resize and to which dimension
 *
 * @param[in]  out_width     width of output image
 * @param[in]  out_height    height of output image
 * @param[out] resize_col_sz       pointer to frame buffer's column/width value
 * @param[out] resize_row_sz       pointer to frame buffer's rows/height value
 * @param[out] do_resize     returns whether to resize (or not)
 *
 */
int calculate_resize_dimensions(uint32_t out_width, uint32_t out_height, uint32_t *resize_col_sz, uint32_t *resize_row_sz, bool *do_resize)
{
    size_t list_size = 2;
    const ei_device_resize_resolutions_t list[list_size] = { {42,32}, {128,96} };

    // (default) conditions
    *resize_col_sz = EI_CAMERA_RAW_FRAME_BUFFER_COLS;
```

```
    *resize_row_sz = EI_CAMERA_RAW_FRAME_BUFFER_ROWS;
    *do_resize = false;

    for (size_t ix = 0; ix < list_size; ix++) {
        if ((out_width <= list[ix].width) && (out_height <= list[ix].height)) {
            *resize_col_sz = list[ix].width;
            *resize_row_sz = list[ix].height;
            *do_resize = true;
            break;
        }
    }

    return 0;
}

/**
 * @brief   Setup image sensor & start streaming
 *
 * @retval  false if initialisation failed
 */
bool ei_camera_init(void) {
    if (is_initialised) return true;

    if (!Cam.begin(QQVGA, RGB565, 1)) { // VGA downsampled to QQVGA (OV7675)
        ei_printf("ERR: Failed to initialize camera\r\n");
        return false;
    }
    is_initialised = true;

    return true;
}
```

```
/**
 * @brief      Stop streaming of sensor data
 */
void ei_camera_deinit(void) {
    if (is_initialised) {
        Cam.end();
        is_initialised = false;
    }
}

/**
 * @brief      Capture, rescale and crop image
 *
 * @param[in]  img_width     width of output image
 * @param[in]  img_height    height of output image
 * @param[in]  out_buf       pointer to store output image, NULL may be used
 *                           when full resolution is expected.
 *
 * @retval     false if not initialised, image captured, rescaled or cropped failed
 *
 */
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf)
{
    if (!is_initialised) {
        ei_printf("ERR: Camera is not initialized\r\n");
        return false;
    }

    if (!out_buf) {
        ei_printf("ERR: invalid parameters\r\n");
```

```
    return false;
  }

  // choose resize dimensions
  int res = calculate_resize_dimensions(img_width, img_height, &resize_col_sz,
&resize_row_sz, &do_resize);
  if (res) {
    ei_printf("ERR: Failed to calculate resize dimensions (%d)\r\n", res);
    return false;
  }

  if ((img_width != resize_col_sz)
    || (img_height != resize_row_sz)) {
    do_crop = true;
  }

  Cam.readFrame(out_buf); // captures image and resizes

  if (do_crop) {
    uint32_t crop_col_sz;
    uint32_t crop_row_sz;
    uint32_t crop_col_start;
    uint32_t crop_row_start;
    crop_row_start = (resize_row_sz - img_height) / 2;
    crop_col_start = (resize_col_sz - img_width) / 2;
    crop_col_sz = img_width;
    crop_row_sz = img_height;

    //ei_printf("crop cols: %d, rows: %d\r\n", crop_col_sz,crop_row_sz);
    cropImage(resize_col_sz, resize_row_sz,
        out_buf,
```

```
            crop_col_start, crop_row_start,
            crop_col_sz, crop_row_sz,
            out_buf,
            16);
    }

    // The following variables should always be assigned
    // if this routine is to return true
    // cutout values
    //ei_camera_snapshot_is_resized = do_resize;
    //ei_camera_snapshot_is_cropped = do_crop;
    ei_camera_capture_out = out_buf;

    return true;
}

/**
 * @brief      Convert RGB565 raw camera buffer to RGB888
 *
 * @param[in]   offset       pixel offset of raw buffer
 * @param[in]   length       number of pixels to convert
 * @param[out]  out_buf      pointer to store output image
 */
int ei_camera_cutout_get_data(size_t offset, size_t length, float *out_ptr) {
    size_t pixel_ix = offset * 2;
    size_t bytes_left = length;
    size_t out_ptr_ix = 0;

    // read byte for byte
    while (bytes_left != 0) {
        // grab the value and convert to r/g/b
```

```
        uint16_t pixel = (ei_camera_capture_out[pixel_ix] << 8) |
ei_camera_capture_out[pixel_ix+1];
        uint8_t r, g, b;
        r = ((pixel >> 11) & 0x1f) << 3;
        g = ((pixel >> 5) & 0x3f) << 2;
        b = (pixel & 0x1f) << 3;

        // then convert to out_ptr format
        float pixel_f = (r << 16) + (g << 8) + b;
        out_ptr[out_ptr_ix] = pixel_f;

        // and go to the next pixel
        out_ptr_ix++;
        pixel_ix+=2;
        bytes_left--;
    }

    // and done!
    return 0;
}

// This include file works in the Arduino environment
// to define the Cortex-M intrinsics
#ifdef __ARM_FEATURE_SIMD32
#include <device.h>
#endif
// This needs to be < 16 or it won't fit. Cortex-M4 only has SIMD for signed
multiplies
#define FRAC_BITS 14
#define FRAC_VAL (1<<FRAC_BITS)
#define FRAC_MASK (FRAC_VAL - 1)
//
```

```
// Resize
//
// Assumes that the destination buffer is dword-aligned
// Can be used to resize the image smaller or larger
// If resizing much smaller than 1/3 size, then a more rubust algorithm should average all of the pixels
// This algorithm uses bilinear interpolation - averages a 2x2 region to generate each new pixel
//
// Optimized for 32-bit MCUs
// supports 8 and 16-bit pixels
void resizeImage(int srcWidth, int srcHeight, uint8_t *srcImage, int dstWidth, int dstHeight, uint8_t *dstImage, int iBpp)
{
  uint32_t src_x_accum, src_y_accum; // accumulators and fractions for scaling the image
  uint32_t x_frac, nx_frac, y_frac, ny_frac;
  int x, y, ty, tx;

  if (iBpp != 8 && iBpp != 16)
    return;
  src_y_accum = FRAC_VAL/2; // start at 1/2 pixel in to account for integer downsampling which might miss pixels
  const uint32_t src_x_frac = (srcWidth * FRAC_VAL) / dstWidth;
  const uint32_t src_y_frac = (srcHeight * FRAC_VAL) / dstHeight;
  const uint32_t r_mask = 0xf800f800;
  const uint32_t g_mask = 0x07e007e0;
  const uint32_t b_mask = 0x001f001f;
  uint8_t *s, *d;
  uint16_t *s16, *d16;
  uint32_t x_frac2, y_frac2; // for 16-bit SIMD
  for (y=0; y < dstHeight; y++) {
    ty = src_y_accum >> FRAC_BITS; // src y
```

```
        y_frac = src_y_accum & FRAC_MASK;
        src_y_accum += src_y_frac;
        ny_frac = FRAC_VAL - y_frac; // y fraction and 1.0 - y fraction
        y_frac2 = ny_frac | (y_frac << 16); // for M4/M4 SIMD
        s = &srcImage[ty * srcWidth];
        s16 = (uint16_t *)&srcImage[ty * srcWidth * 2];
        d = &dstImage[y * dstWidth];
        d16 = (uint16_t *)&dstImage[y * dstWidth * 2];
        src_x_accum = FRAC_VAL/2; // start at 1/2 pixel in to account for integer
downsampling which might miss pixels
        if (iBpp == 8) {
        for (x=0; x < dstWidth; x++) {
            uint32_t tx, p00,p01,p10,p11;
            tx = src_x_accum >> FRAC_BITS;
            x_frac = src_x_accum & FRAC_MASK;
            nx_frac = FRAC_VAL - x_frac; // x fraction and 1.0 - x fraction
            x_frac2 = nx_frac | (x_frac << 16);
            src_x_accum += src_x_frac;
            p00 = s[tx]; p10 = s[tx+1];
            p01 = s[tx+srcWidth]; p11 = s[tx+srcWidth+1];
    #ifdef __ARM_FEATURE_SIMD32
            p00 = __SMLAD(p00 | (p10<<16), x_frac2, FRAC_VAL/2) >> FRAC_BITS; // top
line
            p01 = __SMLAD(p01 | (p11<<16), x_frac2, FRAC_VAL/2) >> FRAC_BITS; //
bottom line
            p00 = __SMLAD(p00 | (p01<<16), y_frac2, FRAC_VAL/2) >> FRAC_BITS; //
combine
    #else // generic C code
            p00 = ((p00 * nx_frac) + (p10 * x_frac) + FRAC_VAL/2) >> FRAC_BITS; // top
line
            p01 = ((p01 * nx_frac) + (p11 * x_frac) + FRAC_VAL/2) >> FRAC_BITS; //
bottom line
            p00 = ((p00 * ny_frac) + (p01 * y_frac) + FRAC_VAL/2) >> FRAC_BITS; //
combine top + bottom
```

```
#endif // Cortex-M4/M7
        *d++ = (uint8_t)p00; // store new pixel
    } // for x
    } // 8-bpp
    else
    { // RGB565
    for (x=0; x < dstWidth; x++) {
        uint32_t tx, p00,p01,p10,p11;
        uint32_t r00, r01, r10, r11, g00, g01, g10, g11, b00, b01, b10, b11;
        tx = src_x_accum >> FRAC_BITS;
        x_frac = src_x_accum & FRAC_MASK;
        nx_frac = FRAC_VAL - x_frac; // x fraction and 1.0 - x fraction
        x_frac2 = nx_frac | (x_frac << 16);
        src_x_accum += src_x_frac;
        p00 = __builtin_bswap16(s16[tx]); p10 = __builtin_bswap16(s16[tx+1]);
        p01 = __builtin_bswap16(s16[tx+srcWidth]); p11 =
__builtin_bswap16(s16[tx+srcWidth+1]);
#ifdef __ARM_FEATURE_SIMD32
        {
        p00 |= (p10 << 16);
        p01 |= (p11 << 16);
        r00 = (p00 & r_mask) >> 1; g00 = p00 & g_mask; b00 = p00 & b_mask;
        r01 = (p01 & r_mask) >> 1; g01 = p01 & g_mask; b01 = p01 & b_mask;
        r00 = __SMLAD(r00, x_frac2, FRAC_VAL/2) >> FRAC_BITS; // top line
        r01 = __SMLAD(r01, x_frac2, FRAC_VAL/2) >> FRAC_BITS; // bottom line
        r00 = __SMLAD(r00 | (r01<<16), y_frac2, FRAC_VAL/2) >> FRAC_BITS; //
combine
        g00 = __SMLAD(g00, x_frac2, FRAC_VAL/2) >> FRAC_BITS; // top line
        g01 = __SMLAD(g01, x_frac2, FRAC_VAL/2) >> FRAC_BITS; // bottom line
        g00 = __SMLAD(g00 | (g01<<16), y_frac2, FRAC_VAL/2) >> FRAC_BITS; //
combine
        b00 = __SMLAD(b00, x_frac2, FRAC_VAL/2) >> FRAC_BITS; // top line
```

```
        b01 = __SMLAD(b01, x_frac2, FRAC_VAL/2) >> FRAC_BITS; // bottom line
        b00 = __SMLAD(b00 | (b01<<16), y_frac2, FRAC_VAL/2) >> FRAC_BITS; //
combine
        }
  #else // generic C code
        {
        r00 = (p00 & r_mask) >> 1; g00 = p00 & g_mask; b00 = p00 & b_mask;
        r10 = (p10 & r_mask) >> 1; g10 = p10 & g_mask; b10 = p10 & b_mask;
        r01 = (p01 & r_mask) >> 1; g01 = p01 & g_mask; b01 = p01 & b_mask;
        r11 = (p11 & r_mask) >> 1; g11 = p11 & g_mask; b11 = p11 & b_mask;
        r00 = ((r00 * nx_frac) + (r10 * x_frac) + FRAC_VAL/2) >> FRAC_BITS; // top
line
        r01 = ((r01 * nx_frac) + (r11 * x_frac) + FRAC_VAL/2) >> FRAC_BITS; //
bottom line
        r00 = ((r00 * ny_frac) + (r01 * y_frac) + FRAC_VAL/2) >> FRAC_BITS; //
combine top + bottom
        g00 = ((g00 * nx_frac) + (g10 * x_frac) + FRAC_VAL/2) >> FRAC_BITS; // top
line
        g01 = ((g01 * nx_frac) + (g11 * x_frac) + FRAC_VAL/2) >> FRAC_BITS; //
bottom line
        g00 = ((g00 * ny_frac) + (g01 * y_frac) + FRAC_VAL/2) >> FRAC_BITS; //
combine top + bottom
        b00 = ((b00 * nx_frac) + (b10 * x_frac) + FRAC_VAL/2) >> FRAC_BITS; // top
line
        b01 = ((b01 * nx_frac) + (b11 * x_frac) + FRAC_VAL/2) >> FRAC_BITS; //
bottom line
        b00 = ((b00 * ny_frac) + (b01 * y_frac) + FRAC_VAL/2) >> FRAC_BITS; //
combine top + bottom
        }
  #endif // Cortex-M4/M7
        r00 = (r00 << 1) & r_mask;
        g00 = g00 & g_mask;
        b00 = b00 & b_mask;
        p00 = (r00 | g00 | b00); // re-combine color components
        *d16++ = (uint16_t)__builtin_bswap16(p00); // store new pixel
```

```
      } // for x
      } // 16-bpp
   } // for y
} /* resizeImage() */
//
// Crop
//
// Assumes that the destination buffer is dword-aligned
// optimized for 32-bit MCUs
// Supports 8 and 16-bit pixels
//
void cropImage(int srcWidth, int srcHeight, uint8_t *srcImage, int startX, int startY, int dstWidth, int dstHeight, uint8_t *dstImage, int iBpp)
{
   uint32_t *s32, *d32;
   int x, y;

   if (startX < 0 || startX >= srcWidth || startY < 0 || startY >= srcHeight || (startX + dstWidth) > srcWidth || (startY + dstHeight) > srcHeight)
      return; // invalid parameters
   if (iBpp != 8 && iBpp != 16)
      return;

   if (iBpp == 8) {
     uint8_t *s, *d;
     for (y=0; y<dstHeight; y++) {
       s = &srcImage[srcWidth * (y + startY) + startX];
       d = &dstImage[(dstWidth * y)];
       x = 0;
       if ((intptr_t)s & 3 || (intptr_t)d & 3) { // either src or dst pointer is not aligned
         for (; x<dstWidth; x++) {
           *d++ = *s++; // have to do it byte-by-byte
```

```
      }
    } else {
      // move 4 bytes at a time if aligned or alignment not enforced
      s32 = (uint32_t *)s;
      d32 = (uint32_t *)d;
      for (; x<dstWidth-3; x+= 4) {
        *d32++ = *s32++;
      }
      // any remaining stragglers?
      s = (uint8_t *)s32;
      d = (uint8_t *)d32;
      for (; x<dstWidth; x++) {
        *d++ = *s++;
      }
    }
  } // for y
} // 8-bpp
else
{
  uint16_t *s, *d;
  for (y=0; y<dstHeight; y++) {
    s = (uint16_t *)&srcImage[2 * srcWidth * (y + startY) + startX * 2];
    d = (uint16_t *)&dstImage[(dstWidth * y * 2)];
    x = 0;
    if ((intptr_t)s & 2 || (intptr_t)d & 2) { // either src or dst pointer is not aligned
      for (; x<dstWidth; x++) {
        *d++ = *s++; // have to do it 16-bits at a time
      }
    } else {
      // move 4 bytes at a time if aligned or alignment no enforced
      s32 = (uint32_t *)s;
```

```
        d32 = (uint32_t *)d;
        for (; x<dstWidth-1; x+= 2) { // we can move 2 pixels at a time
          *d32++ = *s32++;
        }
        // any remaining stragglers?
        s = (uint16_t *)s32;
        d = (uint16_t *)d32;
        for (; x<dstWidth; x++) {
          *d++ = *s++;
        }
      }
    } // for y
  } // 16-bpp case
} /* cropImage() */

#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR !=
EI_CLASSIFIER_SENSOR_CAMERA
#error "Invalid model for current sensor"
#endif

// OV767X camera library override
#include <Arduino.h>
#include <Wire.h>

#define digitalPinToBitMask(P) (1 << (digitalPinToPinName(P) % 32))
#define portInputRegister(P) ((P == 0) ? &NRF_P0->IN : &NRF_P1->IN)

//
// OV7675::begin()
//
// Extends the OV767X library function. Some private variables are needed
```

```
// to use the OV7675::readFrame function.
//
int OV7675::begin(int resolution, int format, int fps)
{
  pinMode(OV7670_VSYNC, INPUT);
  pinMode(OV7670_HREF, INPUT);
  pinMode(OV7670_PLK, INPUT);
  pinMode(OV7670_XCLK, OUTPUT);

  vsyncPort = portInputRegister(digitalPinToPort(OV7670_VSYNC));
  vsyncMask = digitalPinToBitMask(OV7670_VSYNC);
  hrefPort = portInputRegister(digitalPinToPort(OV7670_HREF));
  hrefMask = digitalPinToBitMask(OV7670_HREF);
  pclkPort = portInputRegister(digitalPinToPort(OV7670_PLK));
  pclkMask = digitalPinToBitMask(OV7670_PLK);

  // init driver to use full image sensor size
  bool ret = OV767X::begin(VGA, format, fps);
  width = OV767X::width(); // full sensor width
  height = OV767X::height(); // full sensor height
  bytes_per_pixel = OV767X::bytesPerPixel();
  bytes_per_row = width * bytes_per_pixel; // each pixel is 2 bytes
  resize_height = 2;

  buf_mem = NULL;
  raw_buf = NULL;
  intrp_buf = NULL;
  //allocate_scratch_buffs();

  return ret;
} /* OV7675::begin() */
```

```
int OV7675::allocate_scratch_buffs()
{
    //ei_printf("allocating buffers..\r\n");
    buf_rows = height / resize_row_sz * resize_height;
    buf_size = bytes_per_row * buf_rows;

    buf_mem = ei_malloc(buf_size);
    if(buf_mem == NULL) {
        ei_printf("failed to create buf_mem\r\n");
        return false;
    }
    raw_buf = (uint8_t *)DWORD_ALIGN_PTR((uintptr_t)buf_mem);

    //ei_printf("allocating buffers OK\r\n");
    return 0;
}

int OV7675::deallocate_scratch_buffs()
{
    //ei_printf("deallocating buffers...\r\n");
    ei_free(buf_mem);
    buf_mem = NULL;

    //ei_printf("deallocating buffers OK\r\n");
    return 0;
}

//
// OV7675::readFrame()
//
```

```
// Overrides the OV767X library function. Fixes the camera output to be
// a far more desirable image. This image utilizes the full sensor size
// and has the correct aspect ratio. Since there is limited memory on the
// Nano we bring in only part of the entire sensor at a time and then
// interpolate to a lower resolution.
//
void OV7675::readFrame(void* buffer)
{
  allocate_scratch_buffs();

  uint8_t* out = (uint8_t*)buffer;
  noInterrupts();

  // Falling edge indicates start of frame
  while ((*vsyncPort & vsyncMask) == 0); // wait for HIGH
  while ((*vsyncPort & vsyncMask) != 0); // wait for LOW

  int out_row = 0;
  for (int raw_height = 0; raw_height < height; raw_height += buf_rows) {
    // read in 640xbuf_rows buffer to work with
    readBuf();

    resizeImage(width, buf_rows,
          raw_buf,
          resize_col_sz, resize_height,
          &(out[out_row]),
          16);

    out_row += resize_col_sz * resize_height * bytes_per_pixel; /* resize_col_sz * 2 *
2 */
  }
```

```
    interrupts();

    deallocate_scratch_buffs();
} /* OV7675::readFrame() */

//
// OV7675::readBuf()
//
// Extends the OV767X library function. Reads buf_rows VGA rows from the
// image sensor.
//
void OV7675::readBuf()
{
    int offset = 0;

    uint32_t ulPin = 33; // P1.xx set of GPIO is in 'pin' 32 and above
    NRF_GPIO_Type * port;

    port = nrf_gpio_pin_port_decode(&ulPin);

    for (int i = 0; i < buf_rows; i++) {
        // rising edge indicates start of line
        while ((*hrefPort & hrefMask) == 0); // wait for HIGH

        for (int col = 0; col < bytes_per_row; col++) {
            // rising edges clock each data byte
            while ((*pclkPort & pclkMask) != 0); // wait for LOW

            uint32_t in = port->IN; // read all bits in parallel
```

```
        in >>= 2; // place bits 0 and 1 at the "bottom" of the register
        in &= 0x3f03; // isolate the 8 bits we care about
        in |= (in >> 6); // combine the upper 6 and lower 2 bits

        raw_buf[offset++] = in;

        while ((*pclkPort & pclkMask) == 0); // wait for HIGH
      }

      while ((*hrefPort & hrefMask) != 0); // wait for LOW
    }
} /* OV7675::readBuf() */
```