**Name:** Dhairya Patil **Class:** TY-15 (Batch C) **Roll Number:** 2223929

ECL Experiment 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.

1. 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.

1. 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.

1. 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.

1. 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.

1. Train the Model:

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

1. Test the Model:

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

1. Deploy the Model:

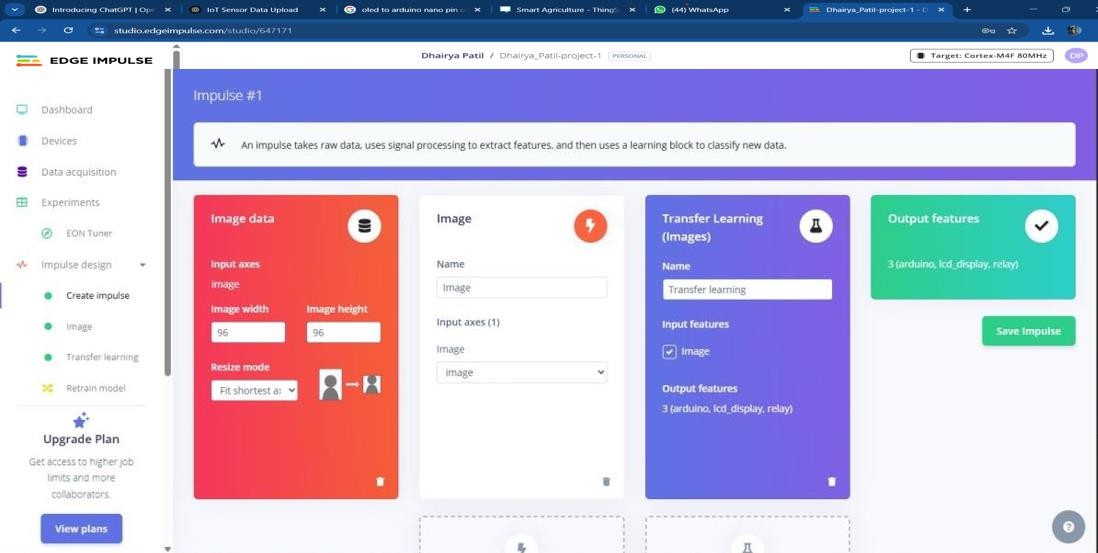
* Go to the 'Deployment' tab.
* Select the deployment method that suits your edge device (e.g., Arduino library, WebAssembly, container, etc.).
* Follow the instructions to deploy the model to your device.

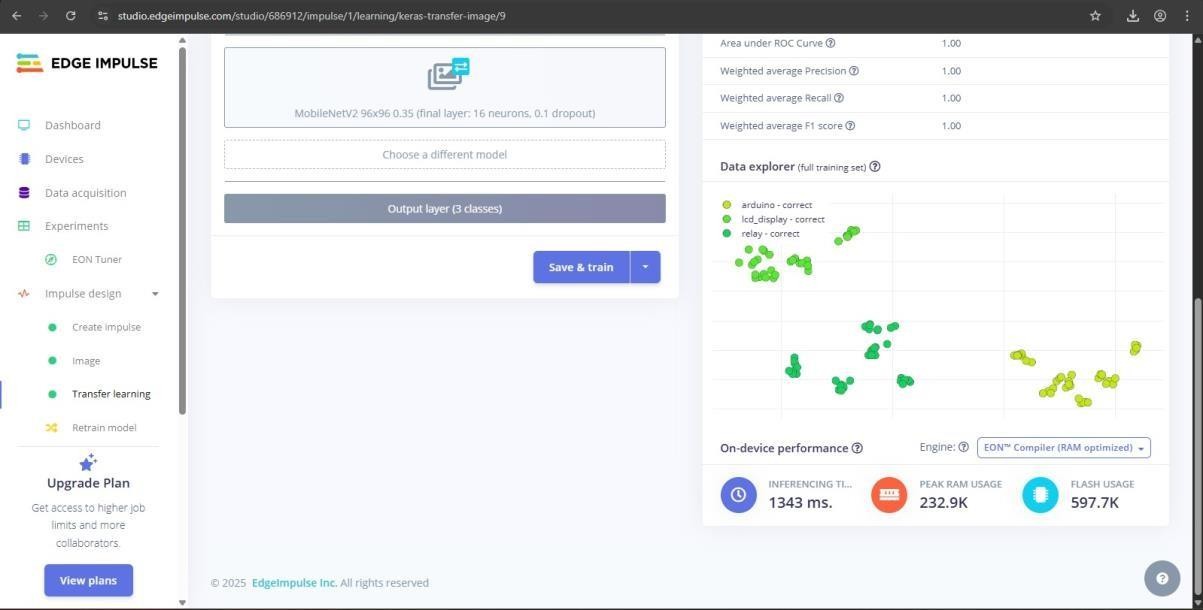
1. Run Inference:

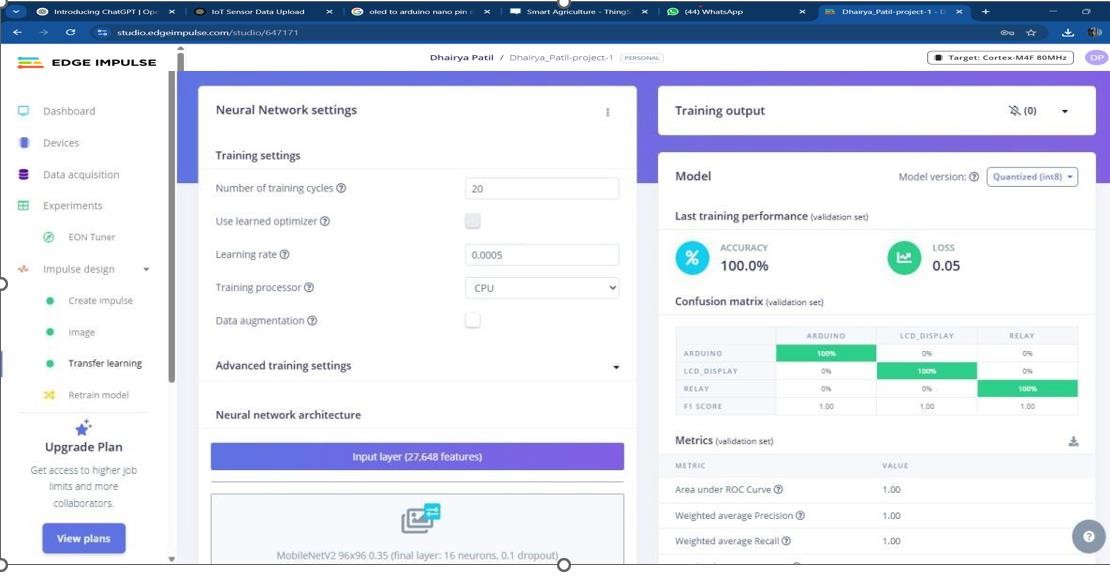
* With the model deployed, run inference on the edge device to see it classifying data in real-time.

1. Monitor:

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

1. **Dataset Image**
2. **Feature Extraction Image**



1. **Accuracy / Loss Confusion Matrix Image**
   1. Copy of the Arduino Code

/\* Edge Impulse ingestion SDK

* + - Copyright © 2022 EdgeImpulse Inc.

\*

* + - Licensed under the Apache License, Version 2.0 (the “License”);
    - you may not use this file except in compliance with the License.
    - You may obtain a copy of the License at
    - <http://www.apache.org/licenses/LICENSE-2.0>
    - Unless required by applicable law or agreed to in writing, software
    - distributed under the License is distributed on an “AS IS” BASIS,
    - WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    - See the License for the specific language governing permissions and
    - limitations under the License.

\*/

/\* Includes \*/

#include <camera\_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-](https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-the-installed-cores-located-) [located-](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 ½ 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 ½ 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

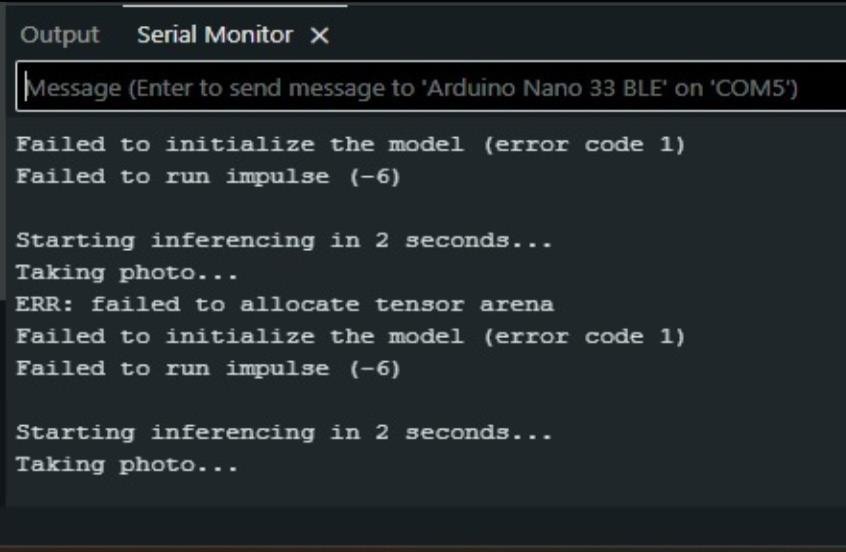
R00 = ((r00 \* ny\_frac) + (r01 \* y\_frac) + FRAC\_VAL/2) >> FRAC\_BITS; // combine top + bottom G00 = ((g00 \* nx\_frac) + (g10 \* x\_frac) + FRAC\_VAL/2) >> FRAC\_BITS; // top line

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\_f

* 1. **Output**

****