

Name-Deep Gorle

Roll no-22231068

Class-TY-AIEC Batch-B

Experiment No-9

1 .DATASET-

Dataset

Data explorer

Data sources

AI labeling

CSV Wizard

DATA COLLECTED

153 items

TRAIN / TEST SPLIT

78% / 22%

Collect data

[Connect a device](#) to start building your dataset.

Dataset

Training (120)

Test (33)

SAMPLE NAME	LABEL	ADDED
arduino.5ps08sfc	arduino	Apr 29 2025, 1...
arduino.5ps08j12	arduino	Apr 29 2025, 1...
arduino.5ps089hm	arduino	Apr 29 2025, 1...
arduino.5ps07mlt	arduino	Apr 29 2025, 1...
arduino.5ps07d64	arduino	Apr 29 2025, 1...
arduino.5ps073n8	arduino	Apr 29 2025, 1...

RAW DATA

arduino.5ps08sfc

Metadata

No metadata.

Dataset

Data explorer

Data sources

AI labeling

CSV Wizard

DATA COLLECTED

153 items

TRAIN / TEST SPLIT

78% / 22%

Collect data

[Connect a device](#) to start building your dataset.

Dataset

Training (120)

Test (33)

SAMPLE NAME	LABEL	ADDED
arduino.5psnp1r	arduino	Apr 29 2025, 1...
arduino.5psnp2jk	arduino	Apr 29 2025, 1...
arduino.5psnop4d	arduino	Apr 29 2025, 1...
arduino.5psnofok	arduino	Apr 29 2025, 1...
relay.5psnlk8l	relay	Apr 29 2025, 1...
relay.5psnlaqk	relay	Apr 29 2025, 1...

RAW DATA

relay.5psnknsm

Metadata

No metadata.

Dataset

Data explorer

Data sources

AI labeling

CSV Wizard

DATA COLLECTED

153 items

TRAIN / TEST SPLIT

78% / 22%

Collect data

[Connect a device](#) to start building your dataset.

Dataset

Training (120)

Test (33)

RAW DATA

lcd_display.5psmr2pv

Metadata

No metadata.

2. Feature Extraction Image

Image data

Input axes

image

Image width

96

Image height

96

Resize mode

Fit shortest

Image

Name

Image

Input axes (1)

Image

image

Transfer Learning (Images)

Name

Transfer learning

Input features

☒ Image

Output features

3 (arduino, lcd_display, relay)

Output features

3 (arduino, lcd_display, relay)

Save Impulse

Raw features


0xbfa0a8, 0xbfa0a8, 0xbfa0a8, 0xbfa0a8, 0xbfa0a8, 0xbfa0a8, 0xbe9fa7, 0xbfa0a8, ...

Parameters

Image

Color depth 

Grayscale 

Save parameters 

DSP result

Image



Processed features

0.6674, 0.6674, 0.6674, 0.6674, 0.6674, 0.6674, 0.6635, 0.6674, 0.6674, 0.6674, ...

On-device performance



PROCESSING TIME

11 ms.



PEAK RAM USAGE

4 KB

3. Accuracy / Loss Confusion Matrix Image

Model

Model version: ?

Quantized (int8) ▾

Last training performance (validation set)



ACCURACY
100.0%



LOSS
0.00

Confusion matrix (validation set)

	ARDUINO	LCD_DISPLAY	RELAY
ARDUINO	100%	0%	0%
LCD_DISPLAY	0%	100%	0%
RELAY	0%	0%	100%
F1 SCORE	1.00	1.00	1.00

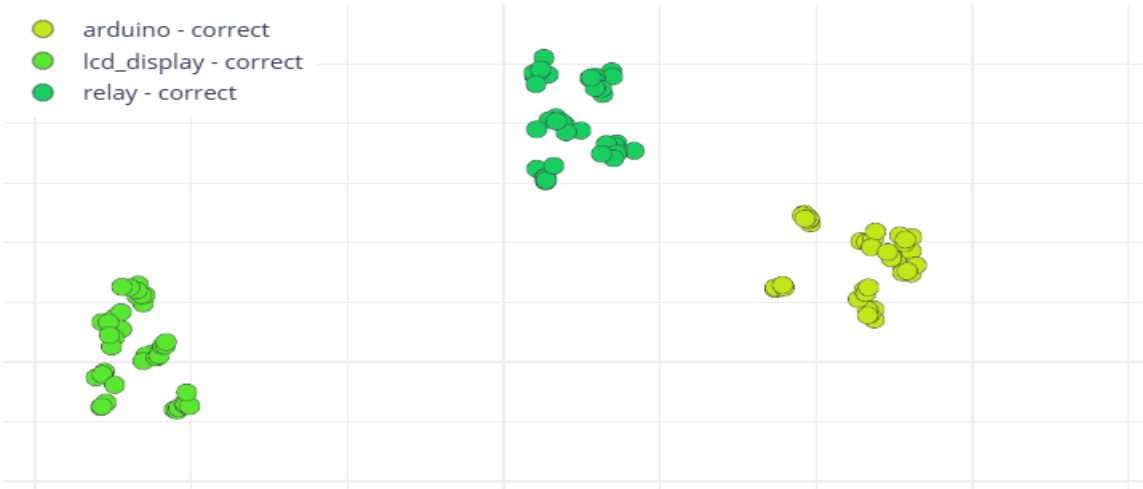
Metrics (validation set)



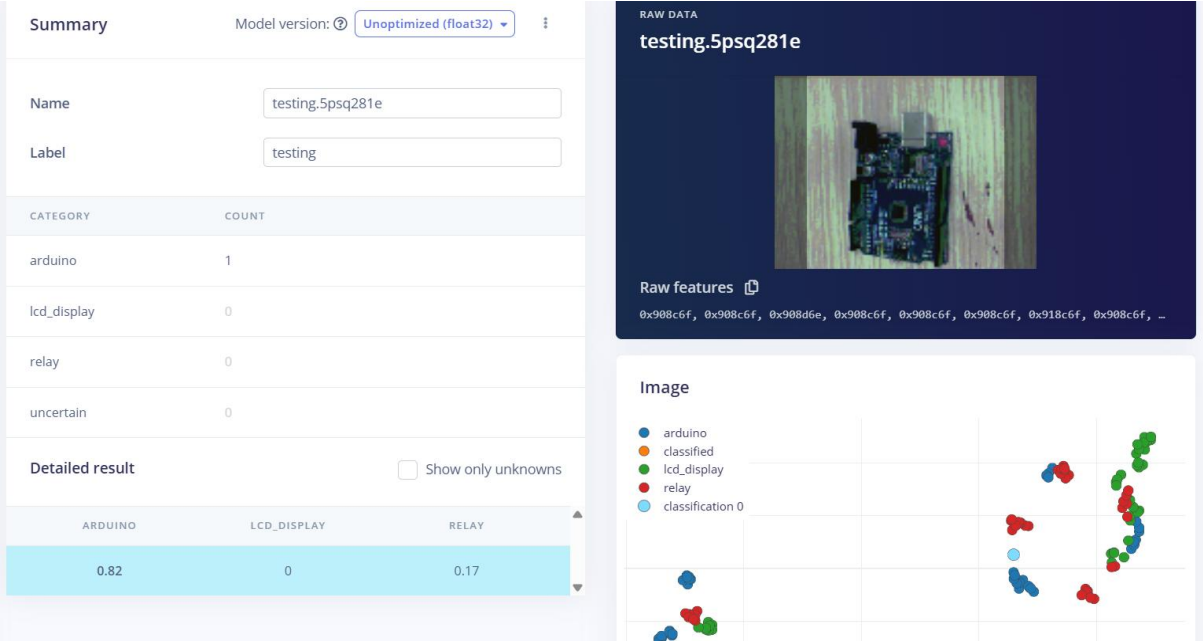
METRIC	VALUE
Area under ROC Curve ?	1.00
Weighted average Precision ?	1.00
Weighted average Recall ?	1.00
Weighted average F1 score ?	1.00

Data explorer (full training set) ?

- arduino - correct
- lcd_display - correct
- relay - correct



4. Validation Result



5. Copy of the Arduino Code-

```
17  /* Includes ----- */
18  #include <camera_inferencing.h>
19  #include <Arduino_OV767X.h> //Click here to get the library: https://www.arduino.cc/reference/en/libraries/arduino\_ov767x/
20
21  #include <stdint.h>
22  #include <stdlib.h>
23
24  /* Constant variables ----- */
25  #define EI_CAMERA_RAW_FRAME_BUFFER_COLS    160
26  #define EI_CAMERA_RAW_FRAME_BUFFER_ROWS    120
27
28  #define DWORD_ALIGN_PTR(a)    ((a & 0x3) ?(((uintptr_t)a + 0x4) & ~(uintptr_t)0x3) : a)
29
30  /*
31  ** NOTE: If you run into TFLite arena allocation issue.
32  **
33  ** This may be due to may dynamic memory fragmentation.
34  ** Try defining "-DEI_CLASSIFIER_ALLOCATION_STATIC" in boards.local.txt (create
35  ** if it doesn't exist) and copy this file to
36  ** <ARDUINO_CORE_INSTALL_PATH>/arduino/hardware/<mbed_core>/<core_version>/.
37  **
38  ** See
39  ** (https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-the-installed-cores-located-)
40  ** to find where Arduino installs cores on your machine.
41  **
42  ** If the problem persists then there's not enough memory for this model and application.
43  */
44
45  /* Edge Impulse ----- */
46  class OV7675 : public OV767X {
47  public:
48      int begin(int resolution, int format, int fps);
49      void readFrame(void* buffer);
50
51  private:
52      int vsyncPin;
53
54  private:
55      int vsyncPin;
56      int hrefPin;
57      int pclkPin;
58      int xclkPin;
59
60      volatile uint32_t* vsyncPort;
61      uint32_t vsyncMask;
62      volatile uint32_t* hrefPort;
63      uint32_t hrefMask;
64      volatile uint32_t* pclkPort;
65      uint32_t pclkMask;
66
67      uint16_t width;
68      uint16_t height;
69      uint8_t bytes_per_pixel;
70      uint16_t bytes_per_row;
71      uint8_t buf_rows;
72      uint16_t buf_size;
73      uint8_t resize_height;
74      uint8_t *raw_buf;
75      void *buf_mem;
76      uint8_t *intrap_buf;
77      uint8_t *buf_limit;
78
79      void readBuf();
80      int allocate_scratch_buffs();
81      int deallocate_scratch_buffs();
82  };
83
84  typedef struct {
85      size_t width;
86      size_t height;
87  } ei_device_resize_resolutions_t;
```

```

84 } ei_device_resize_resolutions_t;
85
86 /**
87  * @brief      Check if new serial data is available
88  *
89  * @return     Returns number of available bytes
90  */
91 int ei_get_serial_available(void) {
92     return Serial.available();
93 }
94
95 /**
96  * @brief      Get next available byte
97  *
98  * @return     byte
99  */
100 char ei_get_serial_byte(void) {
101     return Serial.read();
102 }
103
104 /* Private variables ----- */
105 static OV7675 Cam;
106 static bool is_initialised = false;
107
108 /**
109  * @brief points to the output of the capture
110  */
111 static uint8_t *ei_camera_capture_out = NULL;
112 uint32_t resize_col_sz;
113 uint32_t resize_row_sz;
114 bool do_resize = false;
115 bool do_crop = false;
116
117 static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
118
119 static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
120
121 /* Function definitions ----- */
122 bool ei_camera_init(void);
123 void ei_camera_deinit(void);
124 bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t *out_buf);
125 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);
126 void resizeImage(int srcWidth, int srcHeight, uint8_t *srcImage, int dstWidth, int dstHeight, uint8_t *dstImage, int iBpp);
127 void cropImage(int srcWidth, int srcHeight, uint8_t *srcImage, int startX, int startY, int dstWidth, int dstHeight, uint8_t *dstImage, int iBpp);
128
129 /**
130  * @brief      Arduino setup function
131  */
132 void setup()
133 {
134     // put your setup code here, to run once:
135     Serial.begin(115200);
136     // comment out the below line to cancel the wait for USB connection (needed for native USB)
137     while (!Serial);
138     Serial.println("Edge Impulse Inferencing Demo");
139
140     // summary of inferencing settings (from model_metadata.h)
141     ei_printf("Inferencing settings:\n");
142     ei_printf("\tImage resolution: %dx%d\n", EI_CLASSIFIER_INPUT_WIDTH, EI_CLASSIFIER_INPUT_HEIGHT);
143     ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
144     ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories) / sizeof(ei_classifier_inferencing_categories[0]));
145 }
146
147 /**
148  * @brief      Get data and run inferencing
149  *
150  * @param[in]  debug  Get debug info if true
151  */
152 void loop()
153 {

```

```

150 void loop()
151 {
152     bool stop_inferencing = false;
153
154     while(stop_inferencing == false) {
155         ei_printf("\nStarting inferencing in 2 seconds...\n");
156
157         // instead of wait_ms, we'll wait on the signal, this allows threads to cancel us...
158         if (ei_sleep(2000) != EI_IMPULSE_OK) {
159             break;
160         }
161
162         ei_printf("Taking photo...\n");
163
164         if (ei_camera_init() == false) {
165             ei_printf("ERR: Failed to initialize image sensor\r\n");
166             break;
167         }
168
169         // choose resize dimensions
170         uint32_t resize_col_sz;
171         uint32_t resize_row_sz;
172         bool do_resize = false;
173         int res = calculate_resize_dimensions(EI_CLASSIFIER_INPUT_WIDTH, EI_CLASSIFIER_INPUT_HEIGHT, &resize_col_sz, &resize_row_sz, &do_resize);
174         if (res) {
175             ei_printf("ERR: Failed to calculate resize dimensions (%d)\r\n", res);
176             break;
177         }
178
179         void *snapshot_mem = NULL;
180         uint8_t *snapshot_buf = NULL;
181         snapshot_mem = ei_malloc(resize_col_sz*resize_row_sz*2);
182         if(snapshot_mem == NULL) {
183             ei_printf("failed to create snapshot_mem\r\n");
184             break;
185         }
186
187         snapshot_buf = (uint8_t *)DWORD_ALIGN_PTR((uintptr_t)snapshot_mem);
188
189         if (ei_camera_capture(EI_CLASSIFIER_INPUT_WIDTH, EI_CLASSIFIER_INPUT_HEIGHT, snapshot_buf) == false) {
190             ei_printf("Failed to capture image\r\n");
191             if (snapshot_mem) ei_free(snapshot_mem);
192             break;
193         }
194
195         ei::signal_t signal;
196         signal.total_length = EI_CLASSIFIER_INPUT_WIDTH * EI_CLASSIFIER_INPUT_HEIGHT;
197         signal.get_data = &ei_camera_cutout_get_data;
198
199         // run the impulse: DSP, neural network and the Anomaly algorithm
200         ei_impulse_result_t result = { 0 };
201
202         EI_IMPULSE_ERROR ei_error = run_classifier(&signal, &result, debug_nn);
203         if (ei_error != EI_IMPULSE_OK) {
204             ei_printf("Failed to run impulse (%d)\n", ei_error);
205             ei_free(snapshot_mem);
206             break;
207         }
208
209         // print the predictions
210         ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly: %d ms.): \n",
211             result.timing.dsp, result.timing.classification, result.timing.anomaly);
212
213         #if EI_CLASSIFIER_OBJECT_DETECTION == 1
214             ei_printf("Object detection bounding boxes:\r\n");
215             for (uint32_t i = 0; i < result.bounding_boxes_count; i++) {
216                 ei_impulse_result_bounding_box_t bb = result.bounding_boxes[i];
217                 if (bb.value == 0) {
218                     continue;
219                 }

```



```

215         if (bb.value == 0) {
216             continue;
217         }
218         ei_printf(" %s (%f) [ x: %u, y: %u, width: %u, height: %u ]\r\n",
219             bb.label,
220             bb.value,
221             bb.x,
222             bb.y,
223             bb.width,
224             bb.height);
225     }
226
227     // Print the prediction results (classification)
228 #else
229     ei_printf("Predictions:\r\n");
230     for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
231         ei_printf(" %s: ", ei_classifier_inferencing_categories[i]);
232         ei_printf("%.5f\r\n", result.classification[i].value);
233     }
234 #endif
235
236     // Print anomaly result (if it exists)
237 #if EI_CLASSIFIER_HAS_ANOMALY
238     ei_printf("Anomaly prediction: %.3f\r\n", result.anomaly);
239 #endif
240
241 #if EI_CLASSIFIER_HAS_VISUAL_ANOMALY
242     ei_printf("Visual anomalies:\r\n");
243     for (uint32_t i = 0; i < result.visual_ad_count; i++) {
244         ei_impulse_result_bounding_box_t bb = result.visual_ad_grid_cells[i];
245         if (bb.value == 0) {
246             continue;
247         }

```

```

249         bb.label,
250         bb.value,
251         bb.x,
252         bb.y,
253         bb.width,
254         bb.height);
255     }
256 #endif
257
258     while (ei_get_serial_available() > 0) {
259         if (ei_get_serial_byte() == 'b') {
260             ei_printf("Inferencing stopped by user\r\n");
261             stop_inferencing = true;
262         }
263     }
264     if (snapshot_mem) ei_free(snapshot_mem);
265 }
266 ei_camera_deinit();
267 }
268
269 /**
270  * @brief Determine whether to resize and to which dimension
271  *
272  * @param[in] out_width width of output image
273  * @param[in] out_height height of output image
274  * @param[out] resize_col_sz pointer to frame buffer's column/width value
275  * @param[out] resize_row_sz pointer to frame buffer's rows/height value
276  * @param[out] do_resize returns whether to resize (or not)
277  */
278
279 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)
280 {
281     size_t list_size = 2;
282     const ei_device_resize_resolutions_t list[list_size] = { {42,32}, {128,96} };
283
284     // (default) conditions

```

```

286     *resize_row_sz = EI_CAMERA_RAW_FRAME_BUFFER_ROWS;
287     *do_resize = false;
288
289     for (size_t ix = 0; ix < list_size; ix++) {
290         if ((out_width <= list[ix].width) && (out_height <= list[ix].height)) {
291             *resize_col_sz = list[ix].width;
292             *resize_row_sz = list[ix].height;
293             *do_resize = true;
294             break;
295         }
296     }
297
298     return 0;
299 }
300
301 /**
302  * @brief Setup image sensor & start streaming
303  *
304  * @retval false if initialisation failed
305  */
306 bool ei_camera_init(void) {
307     if (is_initialised) return true;
308
309     if (!Cam.begin(QQVGA, RGB565, 1)) { // VGA downsampled to QQVGA (OV7675)
310         ei_printf("ERR: Failed to initialize camera\r\n");
311         return false;
312     }
313     is_initialised = true;
314
315     return true;
316 }
317
318 /**
319  * @brief Stop streaming of sensor data
320  */

```

```

739 //
740 // Extends the OV767X library function. Reads buf_rows VGA rows from the
741 // image sensor.
742 //
743 void OV7675::readBuf()
744 {
745     int offset = 0;
746
747     uint32_t ulPin = 33; // P1.xx set of GPIO is in 'pin' 32 and above
748     NRF_GPIO_Type * port;
749
750     port = nrf_gpio_pin_port_decode(&ulPin);
751
752     for (int i = 0; i < buf_rows; i++) {
753         // rising edge indicates start of line
754         while ((*hrefPort & hrefMask) == 0); // wait for HIGH
755
756         for (int col = 0; col < bytes_per_row; col++) {
757             // rising edges clock each data byte
758             while ((*pclkPort & pclkMask) != 0); // wait for LOW
759
760             uint32_t in = port->IN; // read all bits in parallel
761
762             in >>= 2; // place bits 0 and 1 at the "bottom" of the register
763             in &= 0x3f03; // isolate the 8 bits we care about
764             in |= (in >> 6); // combine the upper 6 and lower 2 bits
765
766             raw_buf[offset++] = in;
767
768             while ((*pclkPort & pclkMask) == 0); // wait for HIGH
769         }
770
771         while ((*hrefPort & hrefMask) != 0); // wait for LOW
772     }
773 } // OV7675::readBuf() */
774

```

6. Output

```
12:08:33.252 -> Taking photo...
12:08:36.032 -> ERR: failed to allocate tensor arena
12:08:36.032 -> Failed to initialize the model (error code 1)
12:08:36.032 -> Failed to run impulse (-6)
12:08:36.032 ->
12:08:36.032 -> Starting inferencing in 2 seconds...
12:08:38.035 -> Taking photo...
12:08:40.821 -> ERR: failed to allocate tensor arena
12:08:40.821 -> Failed to initialize the model (error code 1)
12:08:40.821 -> Failed to run impulse (-6)
12:08:40.821 ->
```