UNIVERSIDADE FEDERAL DE ITAJUBÁ

CAMPUS ITAJUBÁ

IESTI01

TINYML - APRENDIZADO DE MÁQUINA APLICADO PARA DISPOSITIVOS IOT EMBARCADOS

Teste de qualidade

Alice Bernardi Gottardello - 2019003913

João Pedro de Abreu Soares - 2020004833

Raul Gustavo de Oliveira Bortoloto - 2020012183

UNIFEI – Itajubá
Julho 2024

1. Objetivos do projeto

Este projeto tem como objetivo demonstrar os conhecimentos adquiridos nas aulas de TinyML, criando um modelo de machine learning que poderá ser utilizado em uma situação real de uma indústria. O modelo consiste em identificar a qualidade de um produto que chega em uma esteira de uma linha de produção, avaliando se ele está bom, defeituoso ou vazio pelo som que o produto faz quando é colocado na esteira e mostrando o resultado para o usuário pelo sistema de leds da placa Arduino Nano 33 BLE Sense.

2. Descrição do projeto

O projeto consiste na diferenciação de sons produzidos pela queda de pacotes de bolacha em diferentes situações: totalmente cheio de bolachas, parcialmente cheio de bolachas e vazio. A IA deve ser capaz de diferenciar os diferentes sons, que serão captados pelo microfone integrado ao Arduino Nano, e, assim, classificar o estado do pacote de bolacha com base no som de sua queda.

Foram utilizados diferentes tipos de bolacha na gravação dos sons, porém, todas as quedas foram padronizadas em 15cm de distância entre o pacote e a superfície. Ademais, o pacote parcialmente cheio também foi padronizado com metade da capacidade de bolachas em todos os casos.

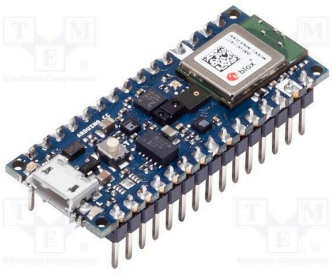
3. Referências (fonte de inspiração)

Como primeira fonte de inspiração, temos um projeto que foi divulgado durante aula ministrada da disciplina, que consistia na análise de garrafas a partir de **inspeções visuais**. Todavia, por conta da baixa qualidade das imagens captadas, imaginamos um cenário de **análise de embalagens através de inspeções sonoras**.

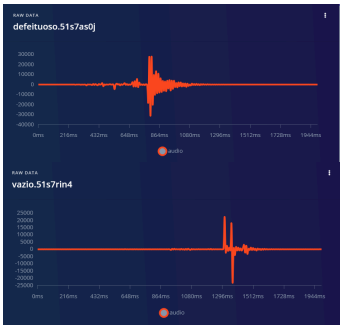
https://www.vision-systems.com/factory/consumer-packaged-goods/article/14277146/deep-learning-at-the-edge-simplifies-package-inspection (Acesso em 11/07/2024 às 21:00)

4. Organização

a. Diagrama de blocos



Aquisição dos dados sonoros produzidos pela queda da embalagem (Sensor MP34DT05-A)



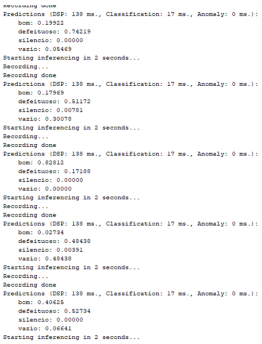
Classificação dos áudios captados

Pré-processamento dos dados e criação do modelo

Treinamento do modelo

TensorFlow Lite

Realização do deploy no arduino através do TensorFlow Lite



Inferência sobre as novas embalagens

b. Hardware utilizado

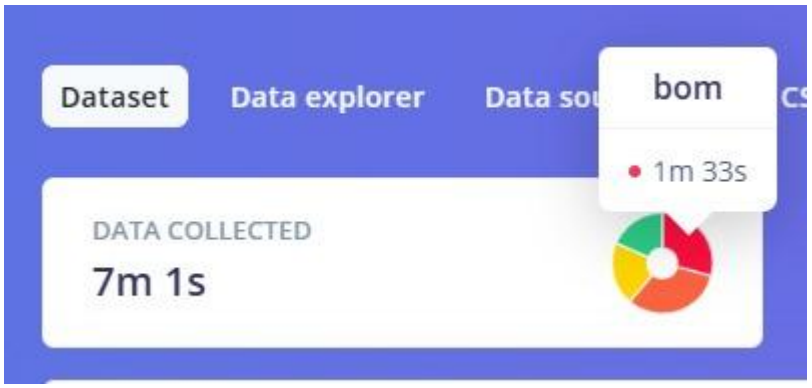
- Arduino Nano 33 BLE Sense;
- Cabo Micro USB (1m);
- Computador pessoal.

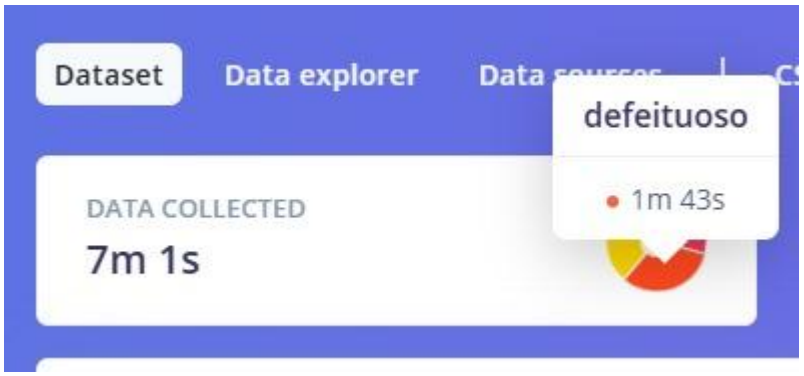
c. Coleta de dados

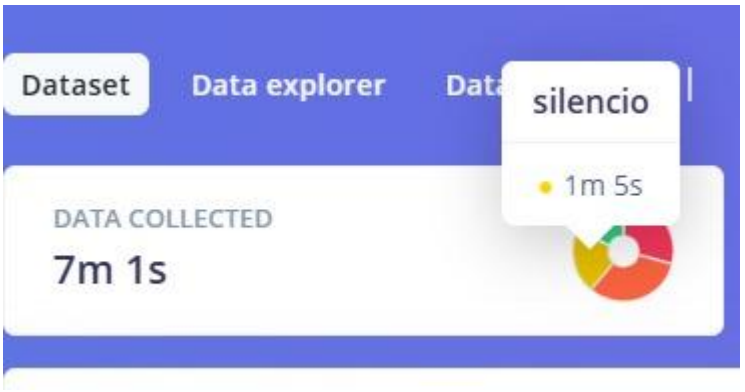
A coleta de dados para o treinamento da IA ocorreu através de sons captados pelo microfone acoplado ao Arduino Nano. Foram produzidos sons através da queda de pacotes de bolacha, de uma altura média de 15cm entre o pacote e a superfície.

Os sons foram produzidos em três situações distintas: pacote totalmente cheio de bolachas, pacote parcialmente cheio de bolachas e, por último, pacote vazio. Ademais, foram utilizados dois tipos distintos de bolacha: água e sal (de formato quadrado) e bolachas de chocolate de formato redondo. Além dos áudios em que era produzido algum som, foram coletados áudios de silêncio também.

A partir disso, foram gerados 7 minutos de áudio de bolachas caindo. As imagens do Edge Impulse mostram como se distribuiu esse tempo de áudio entre os labels pré-determinados:







d. Pré-processamento

Para o bloco de pré-processamento, foi utilizado o do tipo MFE que faz uso da escala Mel-Scale. Ele é recomendado para identificação de sons que não são falas e garante que o modelo final seja pequeno e rápido por causa de suas poucas entradas.

Na tabela abaixo estão os valores usados para o pré-processamento, utilizando as recomendações da plataforma Edge Impulse:

Frame length	0.02
Frame stride	0.01
Filter number	40
FFT length	256
Low frequency	0

e. Design do modelo

Na imagem abaixo estão os parâmetros, usados para criar o design do modelo de machine learning. Ele é composto por uma camada com os dados, uma camada de *reshape*, uma camada unidimensional com 8 neurônios, seguido de uma camada de *dropout* que evita o *overfitting*, uma camada unidimensional de 16 neurônios, mais uma camada de *dropout* e por fim uma camada que transforma a saída em um vetor de uma dimensão.

imagem - Parâmetros de design

f. Treinamento

Na etapa de treinamento, observa-se a dificuldade do modelo de diferenciar o som das labels defeituoso e bom, fazendo com que a acurácia do modelo seja menor que o esperado.

imagem - Resultados do treinamento

g. Deploy

O deployment será feito em um *Arduino Nano 33 BLE Sense*, com o software da placa sendo projetado no *Arduino IDE* junto com o modelo de keyword spotting criado, treinado e validado no *TensorFlow Lite*.

Segue abaixo o código implementado:

```
#include <PDM.h>
#include <Automoveis_inferencing.h>

/** Audio buffers, pointers and selectors */
typedef struct {
    int16_t *buffer;
    uint8_t buf_ready;
    uint32_t buf_count;
    uint32_t n_samples;
} inference_t;

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


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("\tInterval: %.2f ms.\n",
(float)EI_CLASSIFIER_INTERVAL_MS);
    ei_printf("\tFrame size: %d\n",
EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
    ei_printf("\tSample length: %d ms.\n",
EI_CLASSIFIER_RAW_SAMPLE_COUNT / 16);
    ei_printf("\tNo. of classes: %d\n",
sizeof(ei_classifier_inferencing_categories) /
sizeof(ei_classifier_inferencing_categories[0]));

    if
(microphone_inference_start(EI_CLASSIFIER_RAW_SAMPLE_COUNT) ==
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;
        }
          //Initialize RGB LED pins
        pinMode(LEDR, OUTPUT);
        pinMode(LEDG, OUTPUT);
        pinMode(LEDB, OUTPUT);

       //Turn off the RGB LEDs (they are active LOW)
        digitalWrite(LEDR, HIGH);
        digitalWrite(LEDG, HIGH);
        digitalWrite(LEDB, HIGH);
    }


    void loop()
    {
        ei_printf("Starting inferencing in 2 seconds...\n");

        delay(2000);

        ei_printf("Recording...\n");

        bool m = microphone_inference_record();
        if (!m) {
            ei_printf("ERR: Failed to record audio...\n");
            return;
        }

        ei_printf("Recording done\n");

        signal_t signal;
        signal.total_length = EI_CLASSIFIER_RAW_SAMPLE_COUNT;
        signal.get_data = &microphone_audio_signal_get_data;
        ei_impulse_result_t result = { 0 };

        EI_IMPULSE_ERROR r = run_classifier(&signal, &result,
    debug_nn);
        if (r != EI_IMPULSE_OK) {
            ei_printf("ERR: Failed to run classifier (%d)\n", r);
            return;
        }

        // print the predictions
        ei_printf("Predictions ");
        ei_printf("(DSP: %d ms., Classification: %d ms., Anomaly:
    %d ms.)",
            result.timing.dsp, result.timing.classification,
    result.timing.anomaly);
        ei_printf(": \n");
        for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++)
```

```
        {
                ei_printf("    %s: %.5f\n",
        result.classification[ix].label,
        result.classification[ix].value);
            }
        #if EI_CLASSIFIER_HAS_ANOMALY == 1
            ei_printf("    anomaly score: %.3f\n", result.anomaly);
        #endif

        float max_value = 0.0;
            const char* max_label = "";
            for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++)
        {
                if (result.classification[ix].value > max_value) {
                    max_value = result.classification[ix].value;
                    max_label = result.classification[ix].label;
                }
            }
            digitalWrite(LEDR, HIGH);
            digitalWrite(LEDG, HIGH);
            digitalWrite(LEDB, HIGH);
        if (strcmp( max_label, "vazio") == 0) {
                // Set RGB to RED
                digitalWrite(LEDR, LOW);
            }
            else if (strcmp( max_label, "bom") == 0) {
                // Set RGB to GREEN
                digitalWrite(LEDG, LOW);
            }
            else if (strcmp( max_label, "defeituoso") == 0) {
                // Set RGB to BLUE
                digitalWrite(LEDB, LOW);
            }
        }

        /**
         * @brief      PDM buffer full callback
         *             Get data and call audio thread callback
         */
        static void pdm_data_ready_inference_callback(void)
        {
            int bytesAvailable = PDM.available();

            // read into the sample buffer
            int bytesRead = PDM.read((char *)&sampleBuffer[0],
        bytesAvailable);

            if (inference.buf_ready == 0) {
                for(int i = 0; i < bytesRead>>1; i++) {
                    inference.buffer[inference.buf_count++] =
```

```
        sampleBuffer[i];

                if(inference.buf_count >= inference.n_samples) {
                    inference.buf_count = 0;
                    inference.buf_ready = 1;
                    break;
                }
            }
        }
    }

    /**
     * @brief      Init inferencing struct and setup/start PDM
     *
     * @param[in]  n_samples  The n samples
     *
     * @return     { description_of_the_return_value }
     */
    static bool microphone_inference_start(uint32_t n_samples)
    {
        inference.buffer = (int16_t *)malloc(n_samples *
    sizeof(int16_t));

        if(inference.buffer == NULL) {
            return false;
        }

        inference.buf_count  = 0;
        inference.n_samples  = n_samples;
        inference.buf_ready  = 0;

        // configure the data receive callback
        PDM.onReceive(&pdm_data_ready_inference_callback);

        PDM.setBufferSize(4096);

        // initialize PDM with:
        // - one channel (mono mode)
        // - a 16 kHz sample rate
        if (!PDM.begin(1, EI_CLASSIFIER_FREQUENCY)) {
            ei_printf("Failed to start PDM!");
            microphone_inference_end();

            return false;
        }

        // set the gain, defaults to 20
        PDM.setGain(127);

        return true;
```

```
}

/**
 * @brief      Wait on new data
 *
 * @return     True when finished
 */
static bool microphone_inference_record(void)
{
    inference.buf_ready = 0;
    inference.buf_count = 0;

    while(inference.buf_ready == 0) {
        delay(10);
    }

    return true;
}

/**
 * Get raw audio signal data
 */
static int microphone_audio_signal_get_data(size_t offset,
size_t length, float *out_ptr)
{
    numpy::int16_to_float(&inference.buffer[offset], out_ptr,
length);

    return 0;
}

/**
 * @brief      Stop PDM and release buffers
 */
static void microphone_inference_end(void)
{
    PDM.end();
    free(inference.buffer);
}

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

5. Problemas e possíveis soluções

- Houve problema com a padronização da coleta de dados. Como cada membro do projeto colheu dados em sua própria casa, vários dados possuem grandes

diferenças entre dados de uma mesma *label*. Para resolver este problema, deve-se coletar novos dados, utilizando o mesmo produto, caindo em uma superfície de mesmo material e de uma altura comum entre todos os dados.

- Notavelmente, o modelo tem dificuldade de diferenciar as *labels*: bom e defeituoso. Para que o modelo diferencie os produtos com mais facilidade, será necessário coletar mais dados dos produtos bons e defeituosos, de forma que suas diferenças fiquem mais evidentes.

- O microfone presente na placa não possui resolução suficiente para captações sensíveis, como o impacto do produto. Na coleta de novos dados, é recomendado usar um microfone diferente para que se possa captar mais detalhes sonoros.

6. Conclusão

Com o modelo finalizado e implementado, obtivemos uma acurácia de 62,7% nos testes realizados, o que não é um resultado totalmente satisfatório. O resultado obtido faz com que este modelo dificilmente possa ser implementado dentro de uma linha de produção. Resolvendo os problemas apontados no tópico anterior, espera-se que esta porcentagem suba para uma valor acima de 75% para que o modelo seja mais preciso.