# IESTI01-Laboratório 2-Keyword Spotting

Igor Corrêa Nunes - 31367

# 1 Introdução

O objetivo do projeto é indentificar as palavras:

- Pato
- Tobias

Além disso , o modelo de Machine Learning também indentifica o som ambiente como "Environment". O dataset foi gravado a partir do Kit TinyML utilizando o microcontrolador Arduino Nano 33 BLE Sense e plataforma Edge Impulse para as etapas de:

- Pré-processamento do dataset
- Dvisão do dataset em treinamento e teste
- Definição do modelo de rede neural
- Treinamento do modelo
- Teste do modelo
- Otimização para reduzir espaço de memória
- Deploy do modelo Arduino Nano 33 BLE Sense

# 2 Desenvolvimento

O primeiro passo foi gravar áudios à uma taxa de amostragem de 16 kHz em amostras de 10 segundos.Utilizando a ferramenta de divisão de amostras do Edge Impulse, dividir em amostras de 1 segundo e separar em "Treinamento"e "Teste".

Após a curagem do dataset foi definido o modelo de o bloco de processamento MFCC(Mel Frequency Cepstral Coefficient), que é o recomendado para modelos utilizando voz, e em seguinda gerar os features que irão fazer a classificaçã das entradas.

O modelo de Rede Neural escolhido foi o de classificação utilizando a biblioteca Keras.

O design pode ser visto na imagem a seguir:

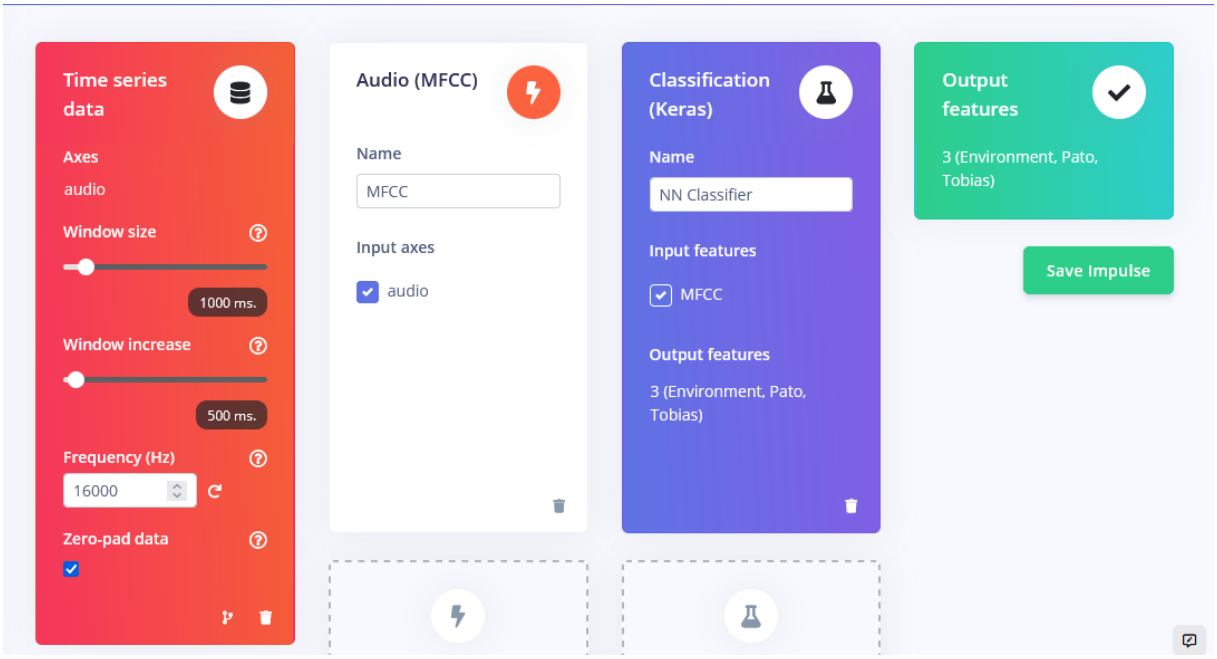


**Imagem 1 - Impulse design**

Em seguida foram definidos os parâmetros para o processamento do dataset para gerar os features:

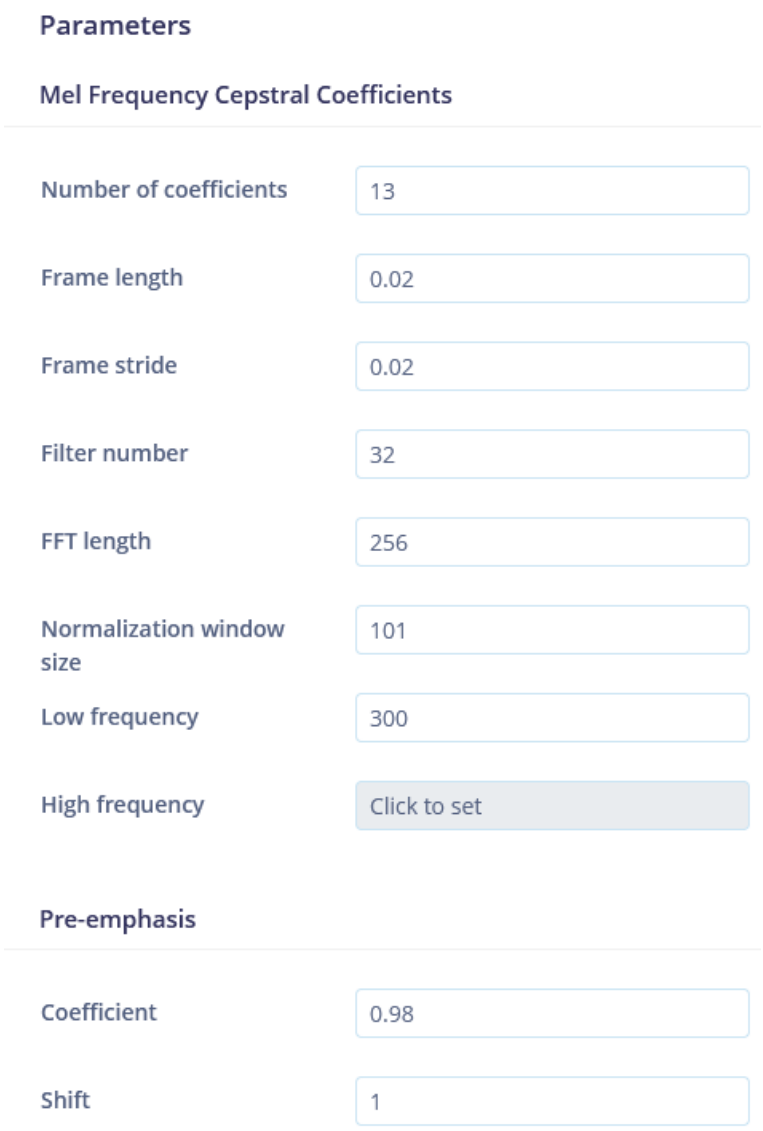


**Imagem 2 - Parameters**

Por fim é definido a quantidade de ciclos para treinamento, o Learning rate e a arquitetura da Rede Neural. O Edge Impulse também oferece a opção de data augmentation para o enriquecimento do dataset.

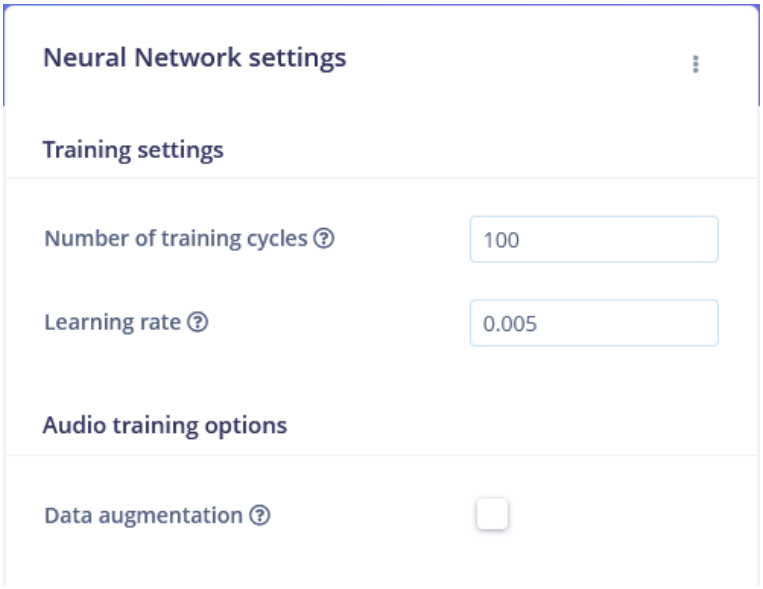


**Imagem 3 - Training settings**

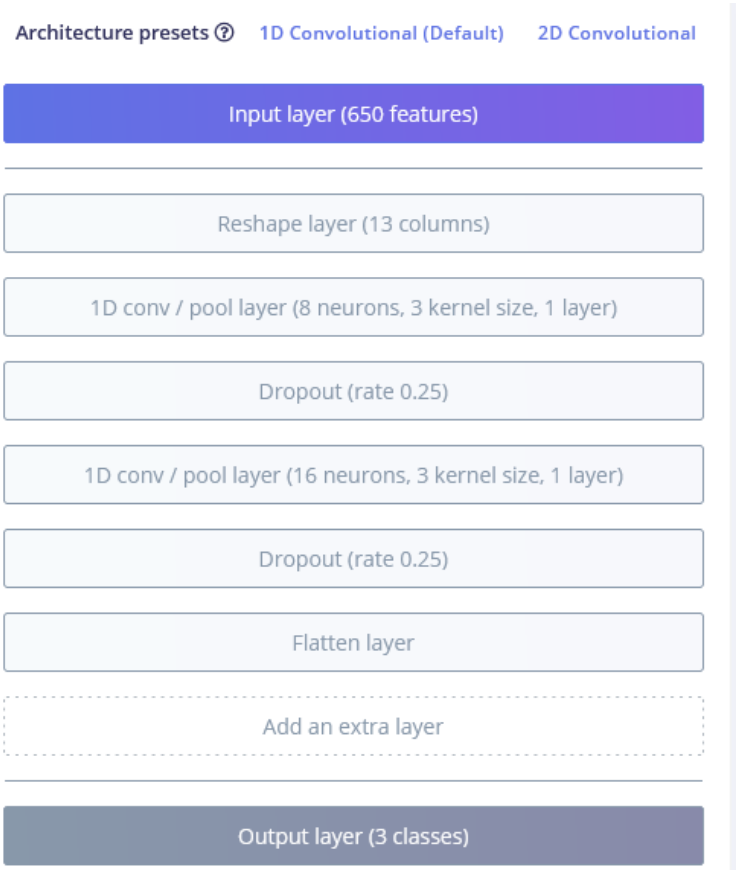


**Imagem 4 - Architecture**

O modelo então é treinado e resulta em Accuracy = 92.5% com Loss = 0.79

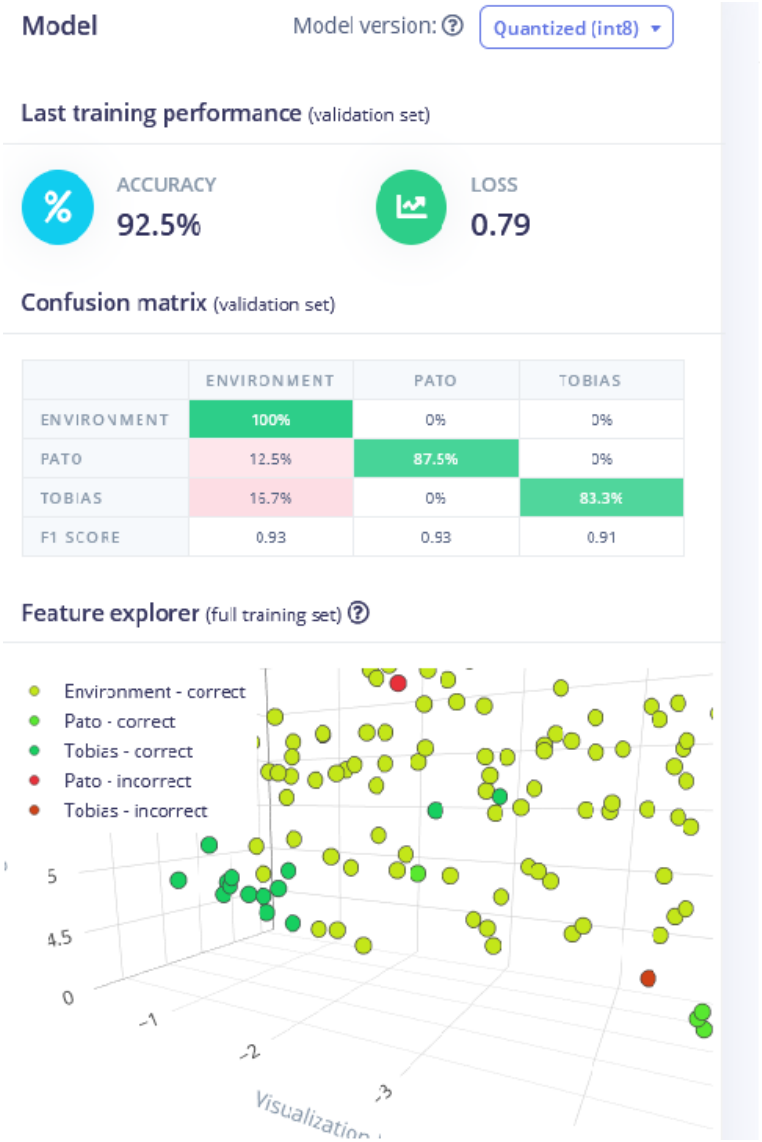


**Imagem 5 - Model**

É feito o teste a partir do dataset de teste e finalmente o deploy do modelo na placa Arduino Nano 33 BLE Sense. Para o deploy geramos uma biblioteca do modelo e, a partir de um código feito pela própria Edge Impulse para análise contínua de amostras de áudio do microfone,o sistema é implementado na placa. O último passo é implementar a ação de ligar determinada cor do Led conforme o resultado da classificação.

# 3 Código

```
1
2
3 * Edge Impulse Arduino examples
4  * Copyright (c) 2021 EdgeImpulse Inc.
5  *
6  * Permission is hereby granted, free of charge, to any person obtaining a
       copy
7  * of this software and associated documentation files (the "Software"), to
       deal
8  * in the Software without restriction, including without limitation the
       rights
9  * to use, copy, modify, merge, publish, distribute, sublicense, and/or
       sell
10 * copies of the Software, and to permit persons to whom the Software is
11 * furnished to do so, subject to the following conditions:
12 *
13 * The above copyright notice and this permission notice shall be included
       in
14 * all copies or substantial portions of the Software.
15 *
16 * THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
       OR
17 * IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
18 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL
       THE
19 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
20 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING
       FROM,
21 * OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS
       IN THE
22 * SOFTWARE.
23 */
24
25 // If your target is limited in memory remove this macro to save 10K RAM
26 #define EIDSP_QUANTIZE_FILTERBANK   0
27
28 /**
29  * Define the number of slices per model window. E.g. a model window of
       1000 ms
30  * with slices per model window set to 4. Results in a slice size of 250 ms
       .
31  * For more info: https://docs.edgeimpulse.com/docs/continuous-audio-
       sampling
```

```
 */
#define EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW 3

/* Includes
   ---------------------------------------------------------------- */
#include <PDM.h>
#include <IESTI01_Key_Word_Spotting_-1.0.2_inferencing.h>

/** 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 bool record_ready = false;
static signed short *sampleBuffer;
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);

/**
 * @brief      Arduino setup function
 */
void setup()
{
    // put your setup code here, to run once:
    Serial.begin(115200);

    Serial.println("Edge Impulse Inferencing Demo");
  //Setup leds
    pinMode(LED_BUILTIN, OUTPUT);
    pinMode(LEDR, OUTPUT);
    pinMode(LEDG, OUTPUT);
    pinMode(LEDB, OUTPUT);

   //ensure the LED is off by default
   digitalWrite(LED_BUILTIN, LOW);

   digitalWrite(LEDR,HIGH);
   digitalWrite(LEDG,HIGH);
   digitalWrite(LEDB,HIGH);

    // summary of inferencing settings (from model_metadata.h)
```

```
77      ei_printf("Inferencing settings:\n");
        ei_printf("\tInterval: %.2f ms.\n", (float)EI_CLASSIFIER_INTERVAL_MS);
79      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);
81      ei_printf("\tNo. of classes: %d\n", sizeof(
            ei_classifier_inferencing_categories) /
                                                sizeof(
                                                    ei_classifier_inferencing_categories
                                                    [0]));
83
        run_classifier_init();
85      if (microphone_inference_start(EI_CLASSIFIER_SLICE_SIZE) == false) {
            ei_printf("ERR: Failed to setup audio sampling\r\n");
87          return;
        }
89  }

91  void turn_off_leds(){
      digitalWrite(LEDR,HIGH);
93    digitalWrite(LEDG,HIGH);
      digitalWrite(LEDB,HIGH);
95  }

97  void turn_on_leds(int pred_index){
      switch(pred_index)
99    {
        case 0:
101       turn_off_leds();
          digitalWrite(LEDG,LOW);
103       break;

105       case 1:
          turn_off_leds();
107       digitalWrite(LEDR,LOW);
          break;
109
          case 2:
111       turn_off_leds();
          digitalWrite(LEDB,LOW);
113       break;

115   }
  }
117

  /**
119  * @brief      Arduino main function. Runs the inferencing loop.
```

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

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

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

    if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)) {
        // 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);
        }
    int pred_index = 0;
    float pred_value = result.classification[0].value;

    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 (result.classification[ix].value > pred_value){
          pred_index = ix;
          pred_value = result.classification[ix].value;
        }
    }
    ei_printf("  Prediction: %s with probability %.2f\n", result.
        classification[pred_index].label, pred_value);
    turn_on_leds(pred_index);
#if EI_CLASSIFIER_HAS_ANOMALY == 1
```

```
163             ei_printf("    anomaly score: %.3f\n", result.anomaly);
#endif
165
            print_results = 0;
167         }
}
169
/**
171  * @brief      Printf function uses vsnprintf and output using Arduino
         Serial
 *
173  * @param[in]  format     Variable argument list
 */
175 void ei_printf(const char *format, ...) {
        static char print_buf[1024] = { 0 };
177
        va_list args;
179     va_start(args, format);
        int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
181     va_end(args);

183     if (r > 0) {
            Serial.write(print_buf);
185     }
}
187
/**
189  * @brief      PDM buffer full callback
 *             Get data and call audio thread callback
191  */
static void pdm_data_ready_inference_callback(void)
193 {
        int bytesAvailable = PDM.available();
195
        // read into the sample buffer
197     int bytesRead = PDM.read((char *)&sampleBuffer[0], bytesAvailable);

199     if (record_ready == true) {
            for (int i = 0; i<bytesRead>> 1; i++) {
201             inference.buffers[inference.buf_select][inference.buf_count++]
                    = sampleBuffer[i];

203             if (inference.buf_count >= inference.n_samples) {
                    inference.buf_select ^= 1;
205                 inference.buf_count = 0;
                    inference.buf_ready = 1;
207             }
```

```
        }
    }
}

/**
 * @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.buffers[0] = (signed short *)malloc(n_samples * sizeof(signed
        short));

    if (inference.buffers[0] == NULL) {
        return false;
    }

    inference.buffers[1] = (signed short *)malloc(n_samples * sizeof(signed
        short));

    if (inference.buffers[1] == NULL) {
        free(inference.buffers[0]);
        return false;
    }

    sampleBuffer = (signed short *)malloc((n_samples >> 1) * sizeof(signed
       short));

    if (sampleBuffer == NULL) {
        free(inference.buffers[0]);
        free(inference.buffers[1]);
        return false;
    }

    inference.buf_select = 0;
    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((n_samples >> 1) * sizeof(int16_t));

```

```
    // 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!");
    }

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

    record_ready = true;

    return true;
}

/**
 * @brief      Wait on new data
 *
 * @return     True when finished
 */
static bool microphone_inference_record(void)
{
    bool ret = true;

    if (inference.buf_ready == 1) {
        ei_printf(
            "Error sample buffer overrun. Decrease the number of slices per model window "
            "(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)\n");
        ret = false;
    }

    while (inference.buf_ready == 0) {
        delay(1);
    }

    inference.buf_ready = 0;

    return ret;
}

/**
 * Get raw audio signal data
 */
static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr)
{
```

```
297     numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][
            offset], out_ptr, length);

299     return 0;
}
301
/**
303  * @brief      Stop PDM and release buffers
 */
305 static void microphone_inference_end(void)
{
307     PDM.end();
     free(inference.buffers[0]);
309     free(inference.buffers[1]);
     free(sampleBuffer);
311 }

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

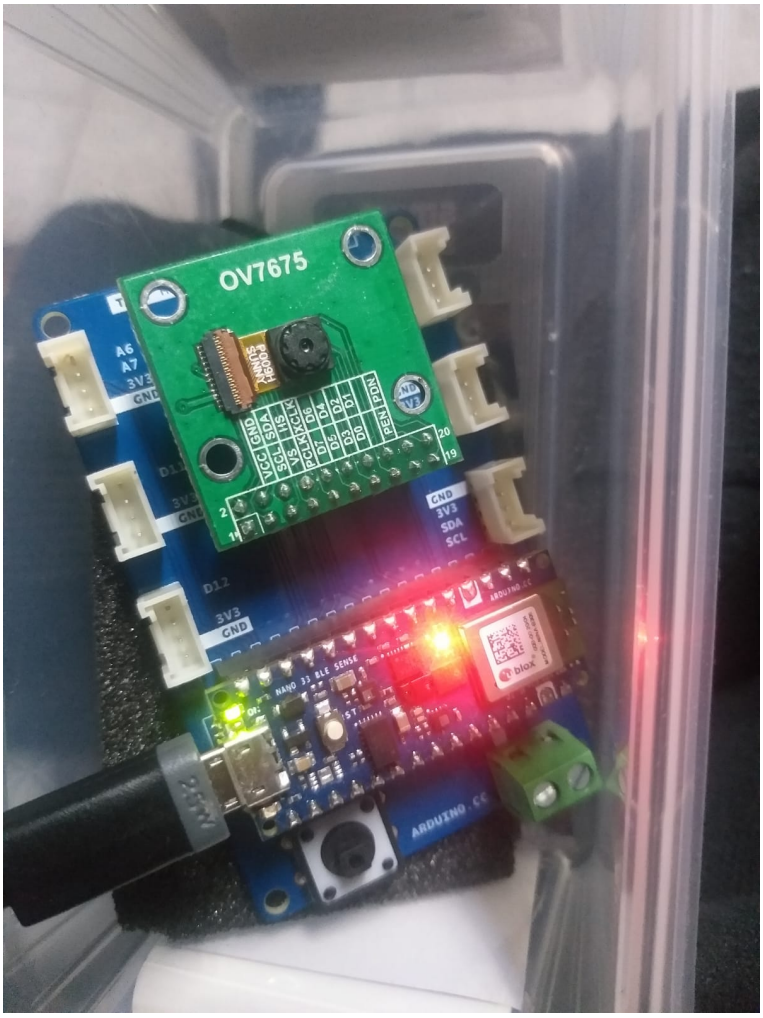


**Imagem 6 -Pato**


**Imagem 7 - Tobias**


**Imagem 8 - Environment**

# 4 Conclusão

Fazendo os testes é possível constatar que o modelo funciona, porém para aumentar a confiabilidade e evitar falsos positivos é interessante expandir o dataset com outras palavras.