

UNIFEI – UNIVERSIDADE FEDERAL DE ITAJUBÁ

TRABALHO FINAL – IESTI01



Henrique Vivaldi – 2021002922

João Pedro Fonseca – 2021015359

Matheus Paiva – 2021013720

Itajubá

2024

1 INTRODUÇÃO

Este trabalho apresenta um projeto que utiliza o microfone do Arduino Nano 33 BLE Sense para identificação de estilos de músicas em tempo real. Neste projeto, exploramos o microfone integrado à placa para capturar amostras de áudio, que são processadas para identificar os estilos musicais. Essa identificação envolve a análise dos padrões de cada estilo e a comparação com o banco de dados das amostras capturadas para o treinamento.

2 MATERIAIS E MÉTODOS

Para a execução do projeto, foi utilizada a placa Arduino Nano 33 BLE Sense que possui o microfone (MP34DT05) para a coleta de dados e identificação dos estilos musicais. Para isso, a placa foi posicionada próxima ao alto-falante de um celular que reproduzia os estilos musicais utilizados para formar os dados.

3 COLETA DE DADOS

A coleta foi feita apenas por um integrante do grupo o qual utilizou um celular para reproduzir os sons dos diferentes estilos musicais para compor os dados de treinamento e teste. A captura das amostras de áudio foi feita utilizando a placa Arduino conectada ao site Edge Impulse. Desta forma, foi possível obter diferentes formas de ondas para tais estilos.

Para início da coleta, foi feita de silêncio (1 minuto e 55 segundos) para que possa ser diferenciado quando há silêncio e quando há algum estilo musical sendo tocado e a forma de onda obtida foi a seguinte:

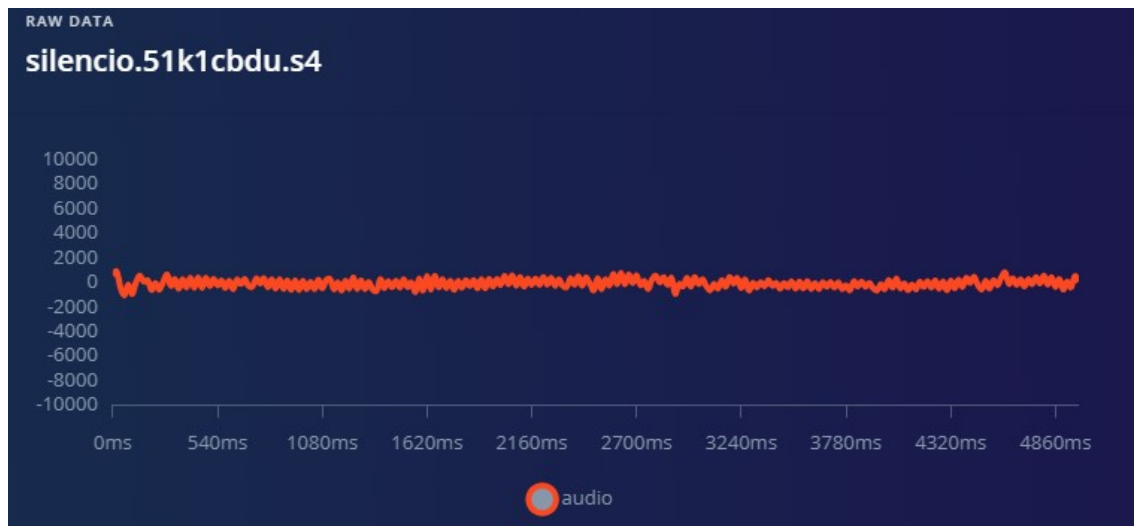


Figura 1: Forma de onda para silêncio

A segunda coleta foi do estilo musical eletrônico (7 minutos) e uma das formas de onda coletada foi a seguinte:

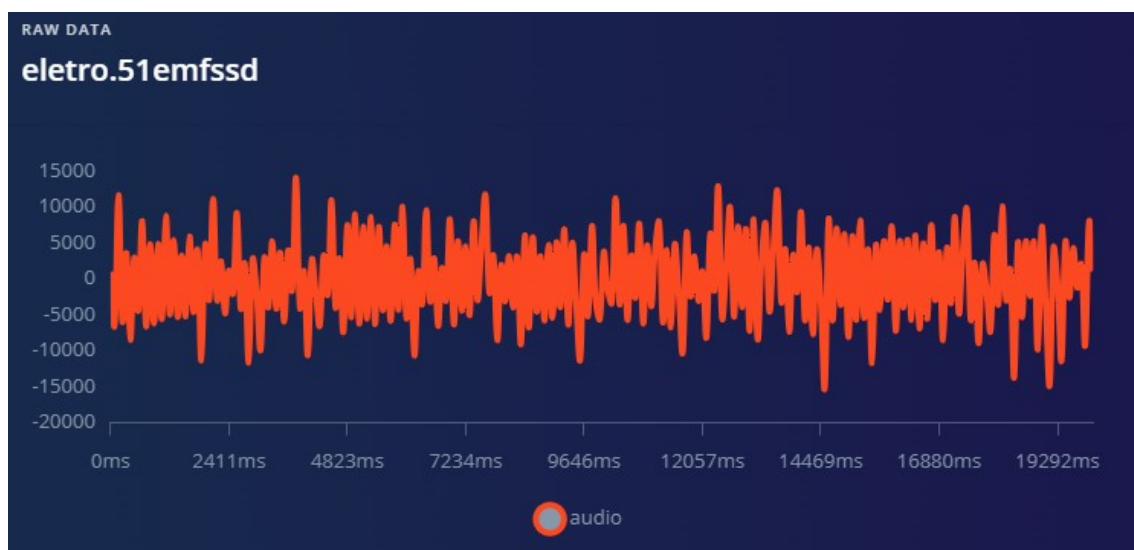


Figura 2: Forma de onda para o estilo eletrônico

A terceira coleta foi do estilo musical clássico (8 minutos e quinze segundos) e uma das formas de onda coletada foi a seguinte:

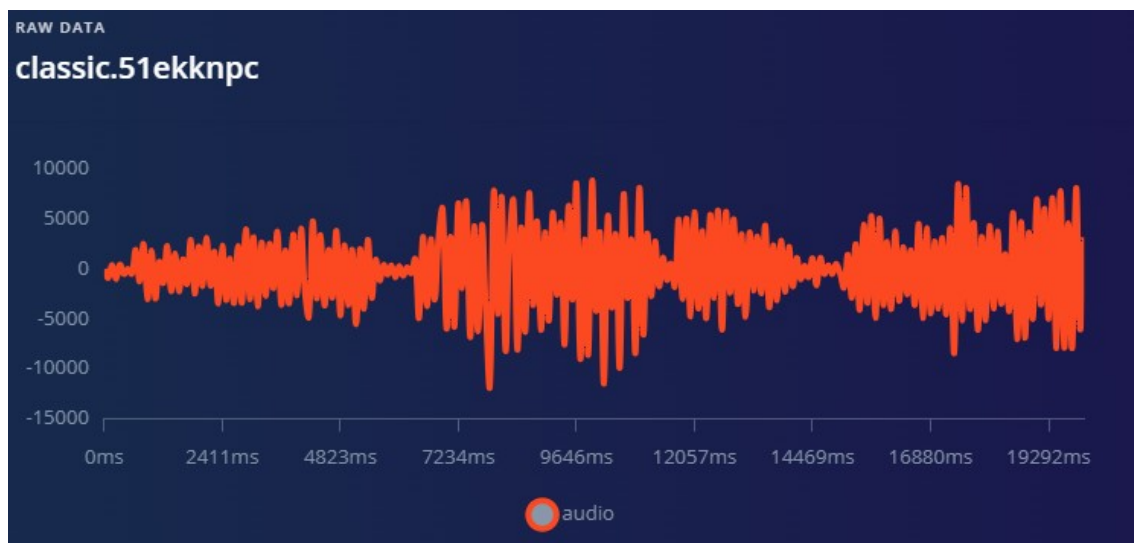


Figura 3: Forma de onda para o estilo clássico

A quarta coleta foi do estilo musical rock (6 minutos e 40 segundos) e uma das formas de onda coletada foi a seguinte:



Figura 4: Forma de onda para o estilo rock

Com isso, todos os dados de teste e treinamento foram coletados.

4 TREINAMENTO DA IA

Para o treinamento da IA é necessário a criação de um impulso no site do Edge Impulse. Para a criação do impulso, as seguintes configurações foram feitas.

4.1 Time series data

Não houve nenhuma alteração da já estabelecida como padrão de sugestão e mais apropriada pelo site.

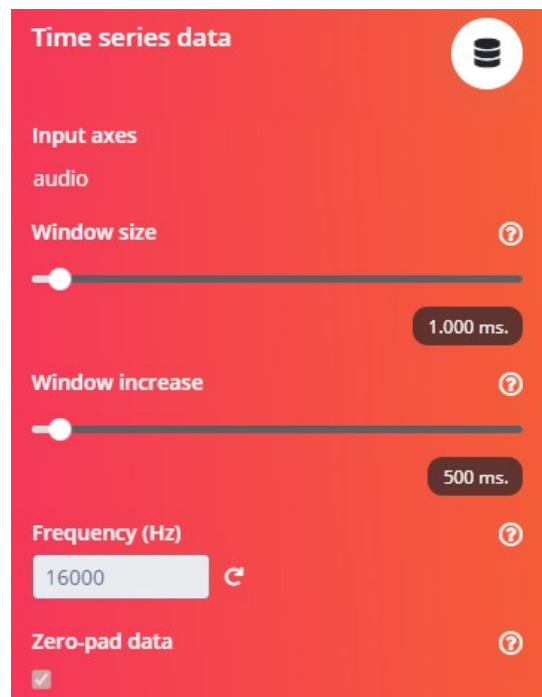



Figura 5: Configuração do time series data

4.2 Processing block

Para a configuração do processing block, a alteração feita foi que em vez de utilizar o MFCC, melhor na detecção de voz humana, que foi sugerido por padrão, utilizamos o MFE, melhor na detecção quando há ausência de voz humana, devido a testes realizados e precisão melhorada.

Audio (MFE)



Name

MFE

Input axes (1)

Signal ?

audio

Figura 6: Configuração do processing block

Para a configuração do processing block MFE escolhido os seguintes parâmetros foram utilizados:

Parameters

Mel-filterbank energy features

Frame length ?	0.02
Frame stride ?	0.01
Filter number ?	40
FFT length ?	256
Low frequency ?	0
High frequency ?	Not set

Normalization

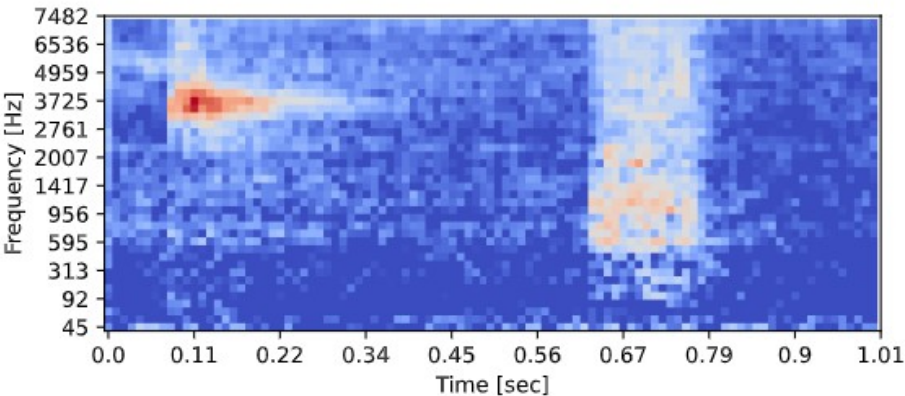
Noise floor (dB) ?	-52
--------------------	-----

Figura 7: Configuração do MFE

Os resultados obtidos de tempo de processamento e máximo uso da memória RAM foram 180ms e 20KB respectivamente. Os resultados de processamento digital de sinais foram os seguintes:

DSP result

Mel Energies (DSP Output)



FFT Bin Weighting

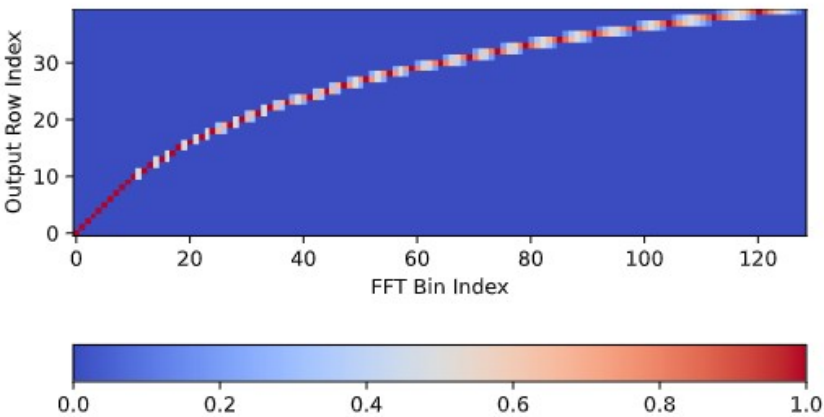
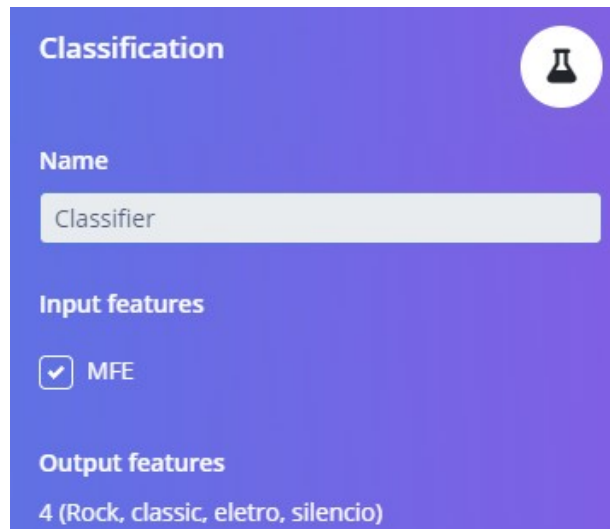


Figura 8: Resultados de DSP

4.3 Learning block

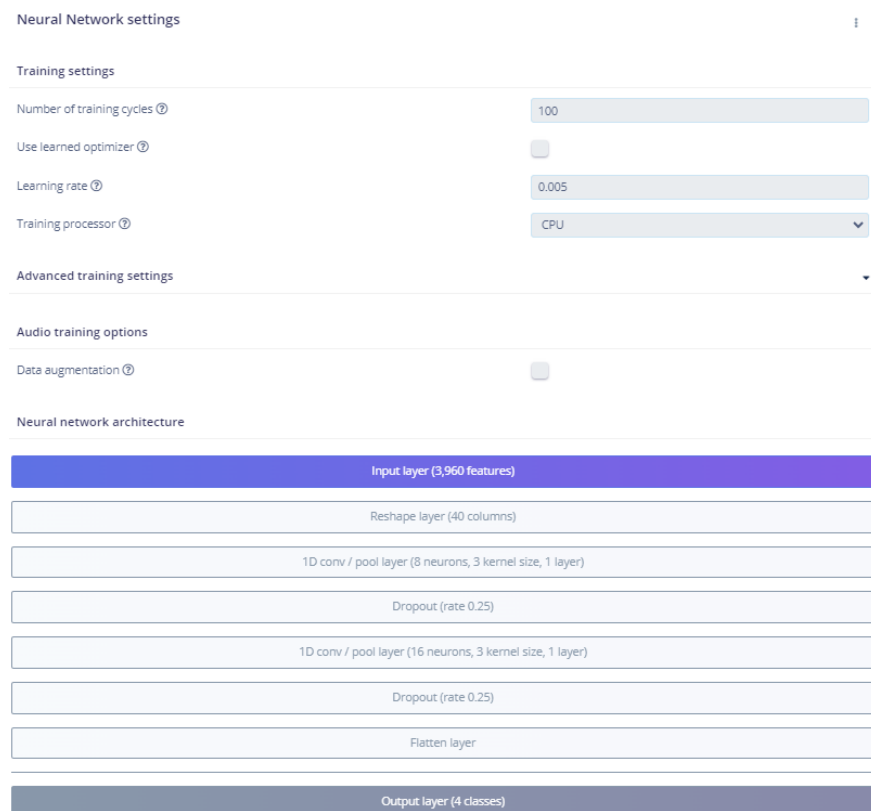
Não houve nenhuma alteração da já estabelecida como padrão de sugestão e mais apropriada pelo site.



The image shows a configuration window titled 'Classification' with a flask icon. It contains three sections: 'Name' with a text input field containing 'Classifier'; 'Input features' with a checked checkbox labeled 'MFE'; and 'Output features' with the text '4 (Rock, classic, eletro, silencio)'.

Figura 9: Configuração do learning block

Para a configuração do classificador escolhido para o learning block, nada foi alterado nos padrões sugeridos devido à alta acurácia alcançada com eles.



The image shows a 'Neural Network settings' configuration window. It includes sections for 'Training settings' (Number of training cycles: 100, Use learned optimizer: unchecked, Learning rate: 0.005, Training processor: CPU), 'Advanced training settings' (collapsed), 'Audio training options' (Data augmentation: unchecked), and 'Neural network architecture'. The architecture is a vertical stack of layers: 'Input layer (3,960 features)' (blue), 'Reshape layer (40 columns)', '1D conv / pool layer (8 neurons, 3 kernel size, 1 layer)', 'Dropout (rate 0.25)', '1D conv / pool layer (16 neurons, 3 kernel size, 1 layer)', 'Dropout (rate 0.25)', 'Flatten layer', and 'Output layer (4 classes)' (dark blue).

Figura 10: Configuração do classifier

Com o modelo feito acima, obteve-se uma alta precisão de 97,1% e pequena perda de 0,08. Além disso, o tempo de inferência foi de 10ms, o pico de uso de memória RAM foi 10,6KB e uso de memória flash foi de 33,4KB.

O modelo e seus resultados de performance na placa podem ser vistos na seguinte imagem:

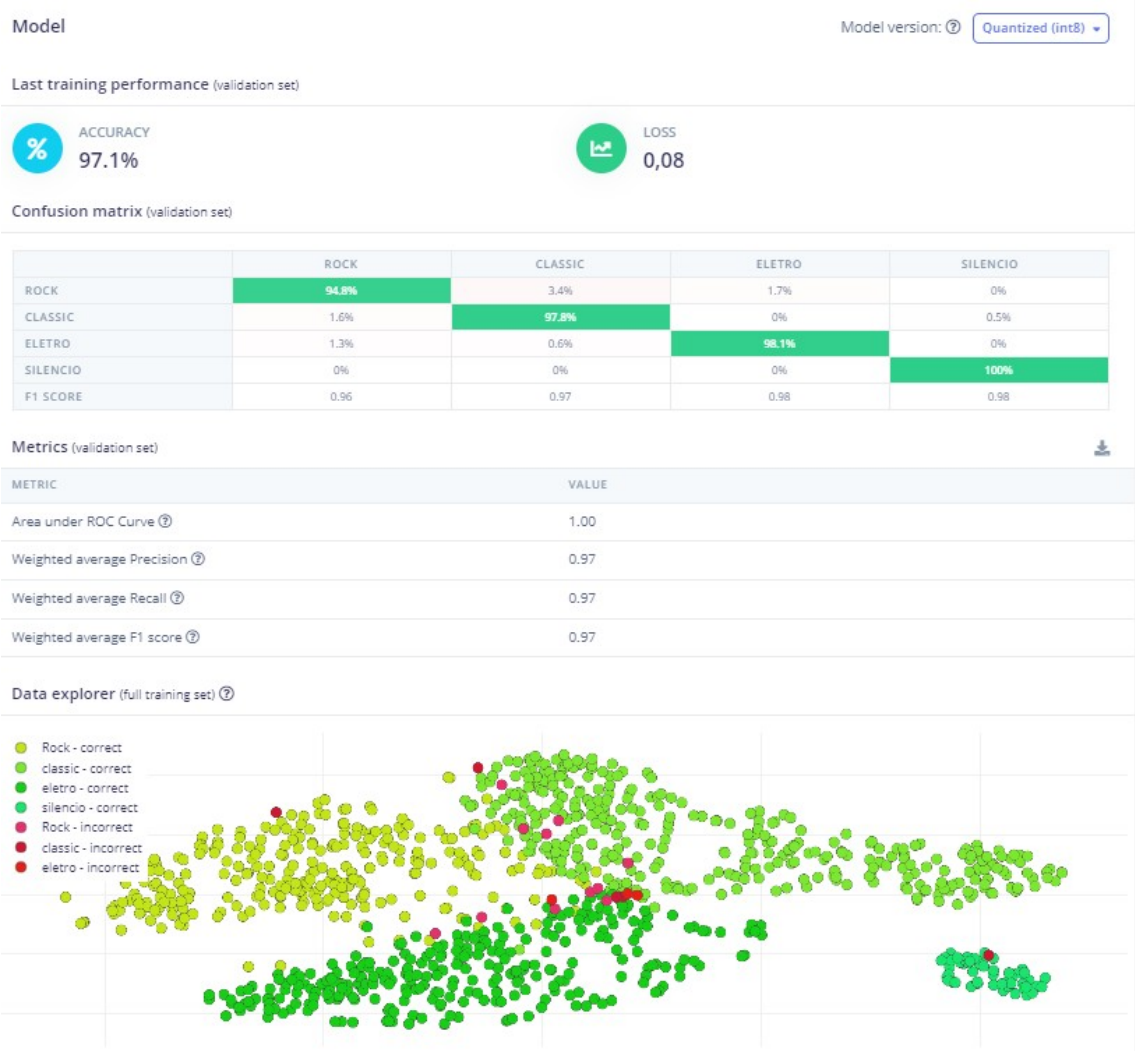


Figura 11: Resultados do modelo

5 TESTE DO MODELO

Com o término do treinamento do modelo criado, é necessário testar os dados coletados anteriormente para garantir que o modelo foi eficiente. Nesta etapa, o site Edge Impulse faz uma classificação de todas as amostras separadas no Data acquisition para serem usadas como teste de modelo.

Para os testes, foram separadas amostras para que estas sejam classificadas. Para silêncio foram 6 amostras de 5 segundos. Para clássica foram 5 amostras de 20 segundos. Para eletrônica foram 4 amostras de 20 segundos. Para rock foram 5 amostras de 20 segundos.

Com isso, obteve-se uma acurácia de 82,17% para o modelo criado, concluindo que foi de bom aproveitamento. Os resultados podem ser observados abaixo.

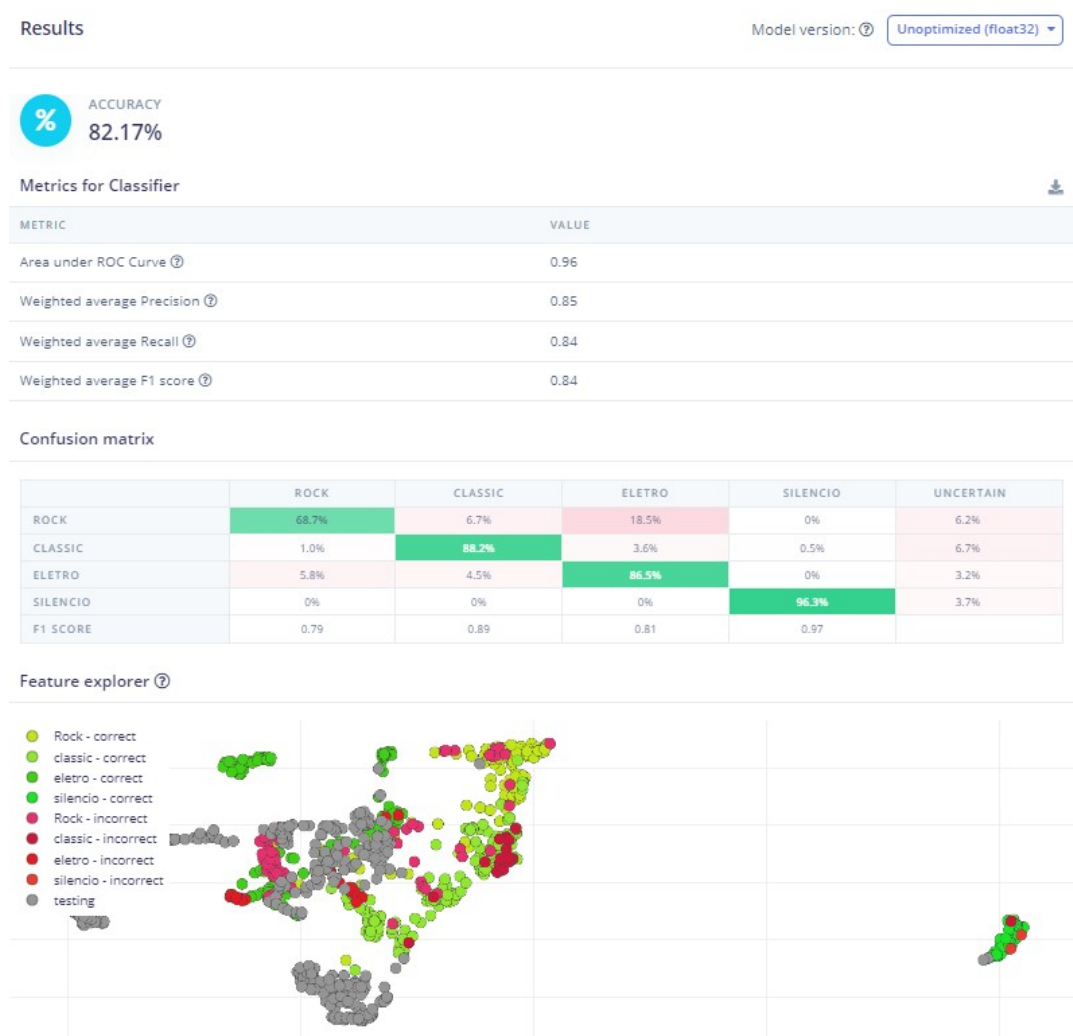


Figura 12: Resultados do teste do modelo

6 DEPLOYMENT

Terminada a sessão de testes e classificações ao vivo, foi feito o deployment da biblioteca que o site disponibiliza do modelo criado. Dessa forma é possível descarregar, por meio da interface da Arduino IDE, na placa. Para isso, a biblioteca baixada foi incluída na IDE e foi gerado o código por meio do exemplo de funcionamento do microfone.

O código gerado foi alterado para inclusão de LED RGB para demonstrar qual estilo musical está sendo tocado. Além disso, foi acrescentado um contador que avalia, durante 3 ciclos de 2 segundos cada, qual foi a classificação realizada para enfim classificar com maior certeza qual é o estilo musical avaliado.

7 RESULTADOS

Com a utilização do código, foi possível identificar qual estilo musical está sendo tocado baseado no LED RGB presentes na placa. Para silêncio, o LED não acende. Para clássico, aciona o verde. Para rock, aciona o vermelho. Para eletrônico aciona o azul. Esses resultados podem ser vistos nas imagens abaixo:



Figura 13: Resultado para silêncio

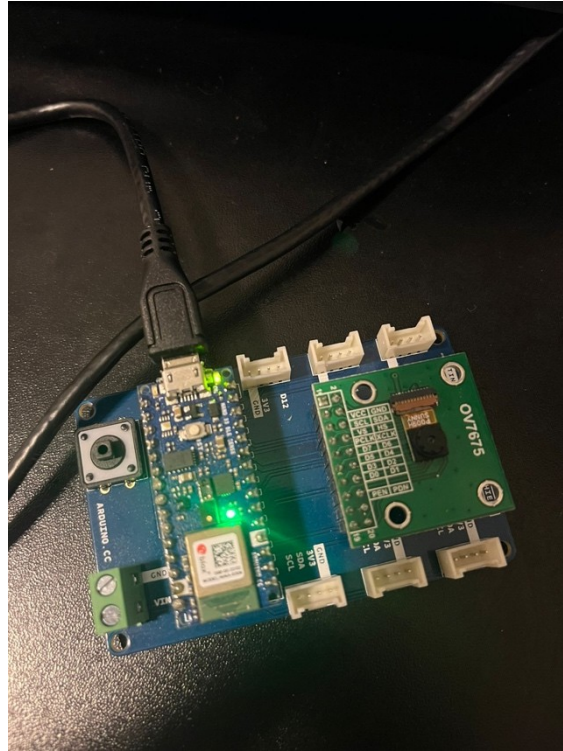


Figura 13: Resultado para estilo clásico

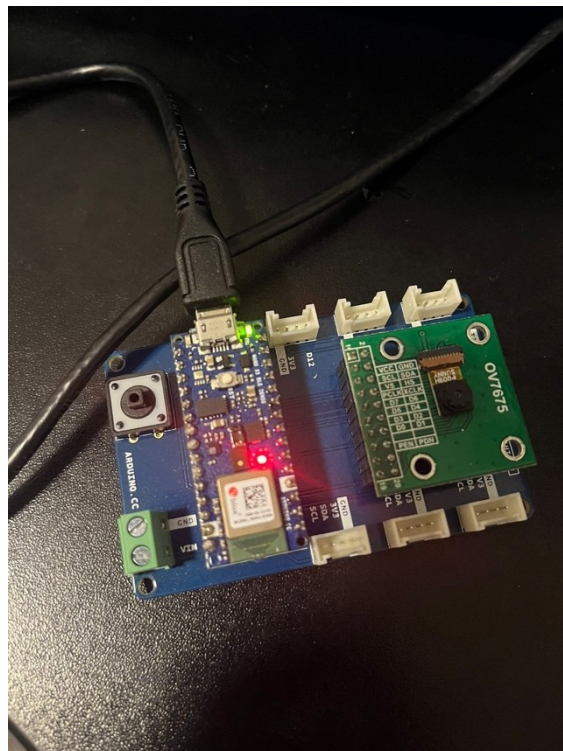


Figura 14: Resultado para estilo rock

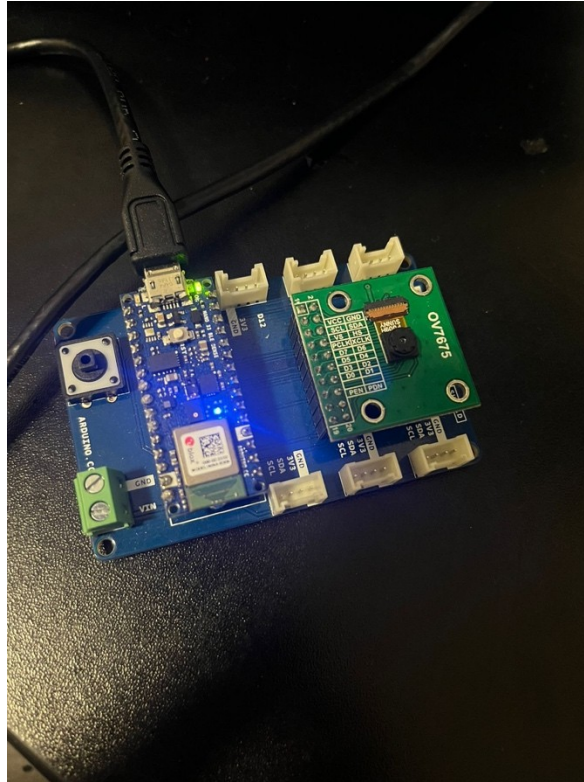


Figura 15: Resultado para estilo eletrônico

8 CÓDIGO

```
/* Edge Impulse ingestion SDK
 * Copyright (c) 2022 EdgImpulse 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.
 *
```

```

*/

// If your target is limited in memory remove this macro to save 10K
RAM
#define EIDSP_QUANTIZE_FILTERBANK 0

/*
** 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.
*/

/* Includes ----- */
#include <PDM.h>
#include <proj_final_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

/**
 * @brief   Arduino setup function
 */
void setup()
{
    // led
    pinMode(LED_R, OUTPUT);
    pinMode(LED_G, OUTPUT);
    pinMode(LED_B, OUTPUT);
    digitalWrite(LED_R, HIGH); // Apaga o LED vermelho
    digitalWrite(LED_G, HIGH); // Apaga o LED verde
    digitalWrite(LED_B, HIGH); // Apaga o LED azul
    // 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;
        }
    }

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

// Variável global para contar os ciclos
int cycle_count = 0;

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

    delay(2000);

    ei_printf("Recording...\n");
    int maxciclo = 3;
    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;
    }

    // Criar um vetor para contar classificações
    static int classification_counts[EI_CLASSIFIER_LABEL_COUNT] =
{0};

    // 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);

        bool is_max = true;
        for (size_t jx = 0; jx < EI_CLASSIFIER_LABEL_COUNT; jx++) {
            if (result.classification[jx].value > result.classification[ix].value) {

```

```

        is_max = false;
        break;
    }
}

if (is_max) {
    classification_counts[ix]++;
}
}

```

```

// Incrementar o contador de ciclos
cycle_count++;

```

```

// Acender o LED com base na classificação com o maior valor em
classification_counts a cada 5 ciclos

```

```

if (cycle_count >= maxciclo) {
    // Encontrar o índice com o maior valor em classification_counts
    size_t max_index = 0;
    int max_count = classification_counts[0];

    for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++) {
        if (classification_counts[ix] > max_count) {
            max_count = classification_counts[ix];
            max_index = ix;
        }
        classification_counts[ix] = 0;
    }
}

```

```

// Acender o LED com base no índice com o maior valor
if (max_index == 0) {
    ei_printf(" Classificacao do ciclo: ROCK\n");
    digitalWrite(LED_R, LOW); // Acende o LED vermelho
}

```

```

        digitalWrite(LEDG, HIGH); // Apaga o LED verde
        digitalWrite(LEDDB, HIGH); // Apaga o LED azul
    } else if (max_index == 1) {
        ei_printf(" Classificacao do ciclo: CLASSICA\n");
        digitalWrite(LEDRL, HIGH); // Apaga o LED vermelho
        digitalWrite(LEDG, LOW); // Acende o LED verde
        digitalWrite(LEDDB, HIGH); // Apaga o LED azul
    } else if (max_index == 2) {
        ei_printf(" Classificacao do ciclo: ELETRO\n");
        digitalWrite(LEDRL, HIGH); // Apaga o LED vermelho
        digitalWrite(LEDG, HIGH); // Apaga o LED verde
        digitalWrite(LEDDB, LOW); // Acende o LED azul
    } else if (max_index == 3) {
        ei_printf(" Classificacao do ciclo: SILENCIO\n");
        digitalWrite(LEDRL, HIGH); // Apaga o LED vermelho
        digitalWrite(LEDG, HIGH); // Apaga o LED verde
        digitalWrite(LEDDB, HIGH); // Apaga o LED azul
    }

    // Redefinir o contador de ciclos
    cycle_count = 0;
}

#if EI_CLASSIFIER_HAS_ANOMALY == 1
    ei_printf("  anomaly score: %.3f\n", result.anomaly);
#endif
}

/**
 * @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

```

9 CONCLUSÃO

Este projeto demonstrou que a placa Arduino Nano 33 BLE Sense, com seu microfone integrado, pode identificar estilos musicais em tempo real. Usando Edge Impulse para coleta de dados, treinamento e teste, alcançamos alta precisão (97,1% no treinamento e 82,17% nos testes). O processamento com MFE, ajustado para ausência de voz humana, foi essencial para o sucesso.

No deployment final, adicionamos um sistema de feedback visual com LEDs RGB, onde cada cor representava um estilo musical diferente, facilitando a demonstração dos resultados. Este projeto não só atingiu seus objetivos, mas também abriu portas para futuras melhorias, como a expansão do banco de dados de estilos musicais e aprimoramentos nos algoritmos de classificação. A combinação de hardware acessível e software de IA mostrou-se poderosa para reconhecimento de áudio.

10 LINKS

Link para o Edge Impulse: [PROJETO FINAL](#)

Link para o vídeo de apresentação: [PROJETO FINAL](#)