Personal Trainer com TinyML

Gabriel B. Vargas - 2019004680
João Vitor Yukio Bordin Yamashita - 2017005768
Pedro Linhares Tristão Moreira - 2017006166
Rodrigo Pereira - 2020004403

Universidade Federal de Itajubá

8 de agosto de 2021

I. Introdução

Assistentes pessoais estão em alta crescente no mundo, com dispositivos focados em bem estar pessoal e auxílio em atividades. Portanto, o ramo de exercícios físicos não poderia ficar de fora dessa tendência, com dispositivos inteligentes para melhorar a experiência do usuário.

Logo, com o uso do *TinyML*, tarefas difíceis, como a classificação do movimento de exercícios físicos, tornam-se possíveis com baixíssimo custo, logo, permitindo uma ampla adoção em mercado.

i. Motivação

Este projeto em específico surgiu com o intuito de ajudar pessoas em academias e melhorar o serviço ofertado pelos estabelecimentos. É comum não se sentir confortável para chamar o responsável, logo, possuir uma maneira embutida de verificar a qualidade do exercício praticado melhora a experiência do usuário com a academia.

Dessa forma, verifica-se também a facilidade de realizar esse projeto, uma vez que halteres são equipamentos muito utilizados e possuem muita área para fixar os sensores necessários. Além disso, um sistema como esse, que depende apenas de um acelerômetro, representaria pouco custo para as academias, para um benefício extremo.

II. Objetivos

Sendo assim, com o projeto concebido em mente, os objetivos são de criar um produto capaz de ser aplicado amplamente nas academias e embarcado em halteres, de modo a verificar a qualidade do exercício praticado.

Especificamente, esses objetivos do projeto podem ser traduzidos em:

- Possuir baixo custo;
- Alta confiabilidade;
- Capacidade de classificar a atividade;
- Capacidade de detectar erros na execução.

III. Desenvolvimento

A execução do projeto se deu pela plataforma *Edge Impulse*, utilizada para a criação de projetos em *TinyML* de maneira extremamente facilitada. As etapas necessárias para a conclusão do projeto são as seguintes:

- Aquisição de dados;
- Processamento dos Dados;
- Criação e treinamento do modelo;
- *Deployment* do modelo.

Dessa forma, cada uma dessas etapas será abordada nessa seção, iniciando pela coleta dos dados necessários.

i. Coleta de Dados

O processo de coleta pode ser resumido em três etapas iniciais:

1. Dados do Arduino Nano 33 Ble Sense + Acelerômetro
2. *Upload* para a plataforma *Edge Impulse*;
3. Separação em classes.

Os dados foram coletados ao se fixar o Arduino em um haltere de academia, como pode ser visto na figura 1, e posteriormente sendo realizado o exercício rosca direta. Esse processo foi repetido para 5 pessoas, com faixa etária entre 15 e 23 anos. Já na figura 2, é possível visualizar o processo de coleta durante o exercício realizado.

(a) *Fixação do Arduino no haltere*

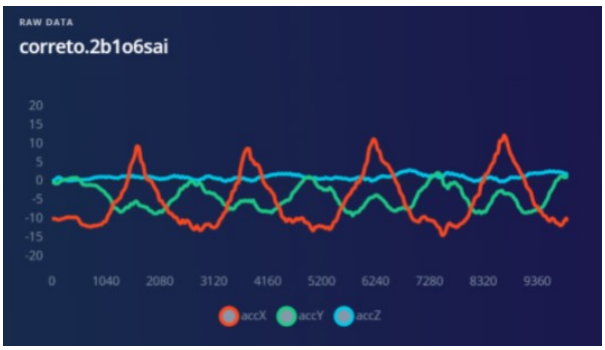
(b) *Conexão com a plataforma*

Figura 1: *Método de coleta de dados*

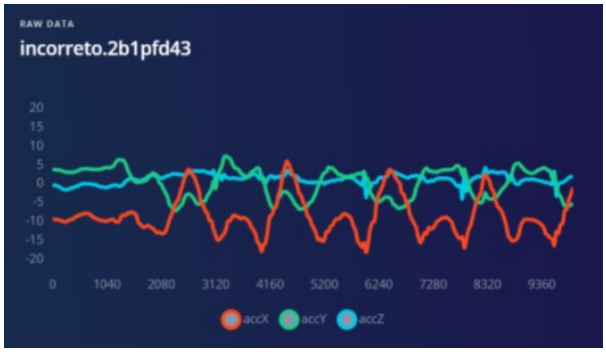
Figura 2: *Exercício de coleta de dados*

Por fim, esses dados puderam ser observados na plataforma do *Edge Impulse*, como demonstrado na figura 10. Nesses dados é notável uma variação periódica nas acelerações dos eixos X e Y, enquanto o eixo Z permanece praticamente constante.

Já na prática incorreta do exercício, como observada na figura 3b, a periodicidade das variações são diferentes, com o eixo X variando mais que o Y. Logo, esse será um ponto importante para a diferenciação das classes.



(a) *Execução correta*



(b) *Execução incorreta*

Figura 3: *Dados coletados no Edge Impulse*

No total, inicialmente foram coletados dados para três classes, sendo elas:

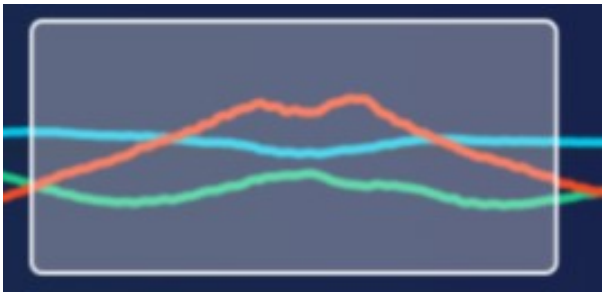
- Idle: equipamento em repouso (1 min 10 s);
- Correto: rosca direta executada corretamente (4 min 20 s);
- Incorreto: rosca direta executada incorretamente (3 min 30 s).

ii. Processamento dos Dados

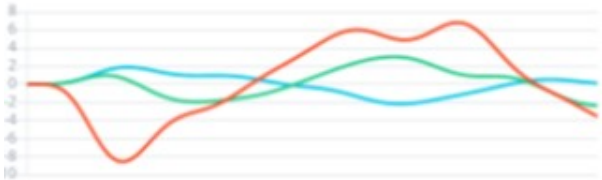
,

Com os dados já coletados, foi necessária a realização de um pré-processamento, de modo a reduzir a quantidade de *features* geradas. Essa etapa é de importância fundamental, pois caso haja uma grande quantidade de *features*, o modelo fica grande demais para utilização em um dispositivo embarcado.

Para a filtragem dos dados, foi aplicado um filtro passa baixas, de modo a cortar altas frequências que pudessem existir nos dados. A comparação dessa etapa pode ser vista na figura 4.



(a) *Sinal original*



(b) *Sinal filtrado*

Figura 4: *Processamento dos sinais*

Com os dados já filtrados, foi aplicado o *Fast Fourier Transform (FFT)*, de modo a se visualizar as frequências presentes na amostra de dados. O gráfico do domínio das frequências é observado na figura 5a. Com esses dados em mãos, foi encontrado o *Spectral Power Distribution (SPD)*, que consiste em encontrar a potência de cada frequência existente no sinal. Esses são de fato os dados passados para a próxima etapa, de treinamento. O gráfico com o SDP pode ser visto na figura 5b.

(a) *Domínio das frequências* **(b)** *Spectral Power Distribution*

Figura 5: *Características dos sinais*

iii. Modelo e Treinamento

Já com as *features* extraídas na última etapa, resta criar o modelo a ser treinado. Esse modelo de rede neural conta com uma camada de entrada com 33 neurônios, um para cada *feature*, e uma de saída com 3 neurônios, um por classe. Entre essas camadas, foram adicionadas três camadas densas, com 20 neurônios cada.

Figura 6: *Modelo de rede neural*

O modelo criado pode ser observado também pela plataforma *Netron*, como demonstrado na figura 7. Nele, também é possível ver as funções de ativação de cada camada, sendo a *ReLu* utilizada para as camadas densas, e a *Softmax* para a camada de saída.

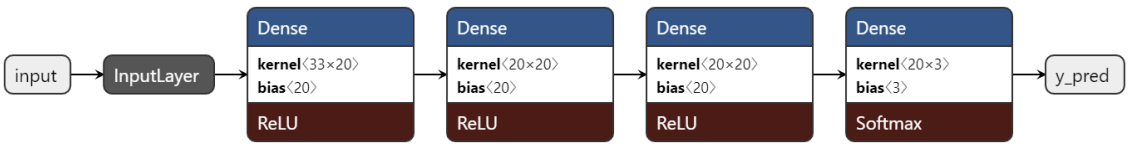
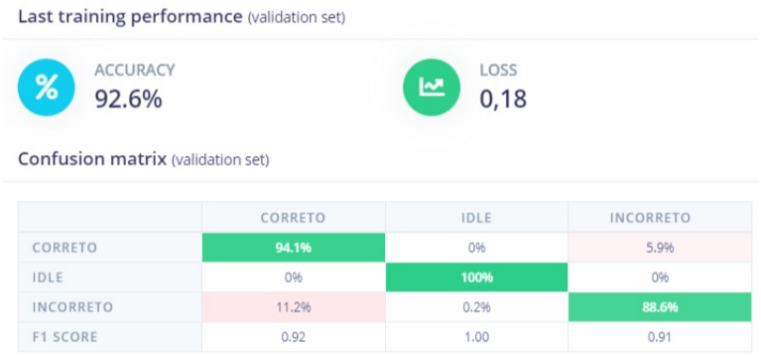


Figura 7: *Modelo visualizado no Netron*

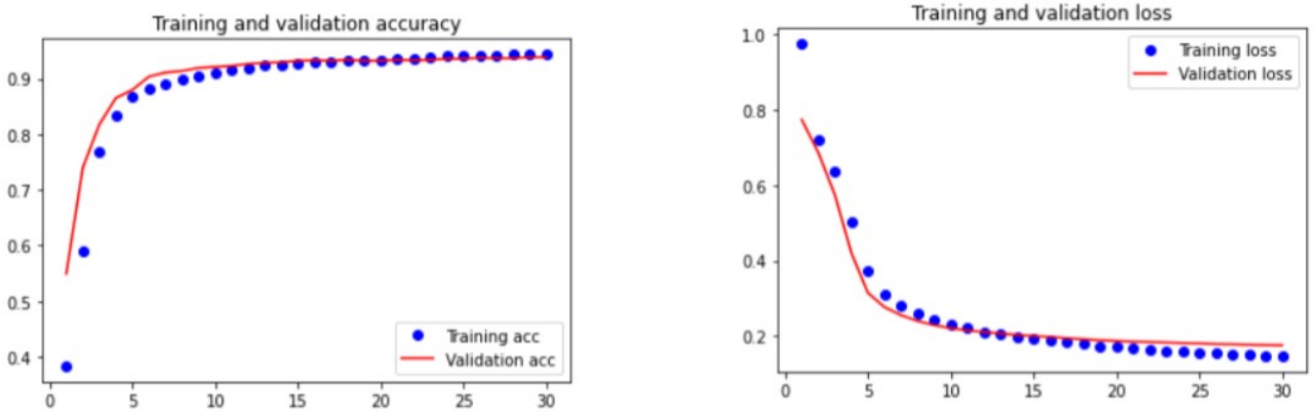
Após a realização do treinamento, em 30 épocas, o resultado obtido foi extremamente satisfatório, e pode ser observado na figura 8.



	CORRETO	IDLE	INCORRETO
CORRETO	94.1%	0%	5.9%
IDLE	0%	100%	0%
INCORRETO	11.2%	0.2%	88.6%
F1 SCORE	0.92	1.00	0.91

Figura 8: *Resultado de treinamento*

Em adição, o projeto foi exportado para o Google Colab, e, com isso, o treinamento pôde ser melhor observado, sendo plotados dois gráficos, que representam o ganho de acurácia e a diminuição da perda do modelo. Esses gráficos são observados nas figuras 9a e 9b. Nota-se que não houve *overfit* significativo no sistema, já que a curva de acurária dos dados de validação seguiu a curva dos dados de treino.

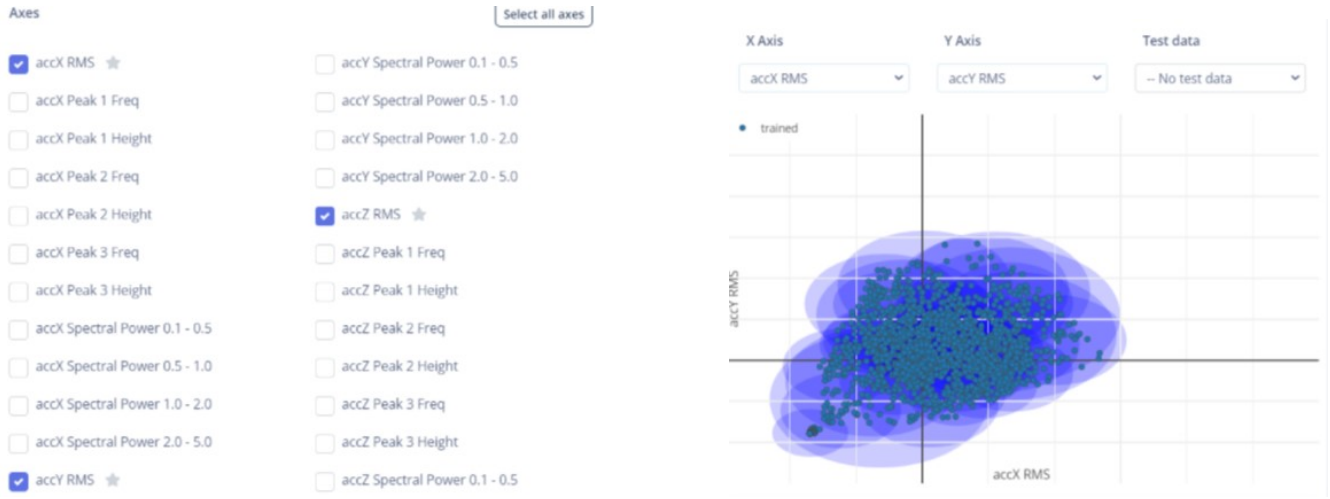


(a) *Acurácia do modelo*

(b) *Perda do modelo*

Figura 9: *Acurácia e perda a cada época*

Por fim, com o intuito de detectar outras possíveis falhas na execução dos exercícios, foi adicionado ao modelo um detector de anomalias, utilizando o método *K-means*. Esse método agrupa os dados em *clusters*, se baseando, como visto na figura 10a, nos valores RMS de cada eixo. Os *clusters* criados são vistos na figura 10b.



(a) *Valores utilizados no K-means* **(b)** *Clusters criados*

Figura 10: *Modelo de detecção de anomalia*

Tendo sido realizado o treinamento desse modelo, os resultados obtidos de anomalia podem ser observados na figura 11, obtendo uma acurácia de 93,07%.

	CORRETO	IDLE	INCORRETO	ANOMALY	UNCERTA...
CORRETO	92.3%	0%	4.2%	0%	3.5%
IDLE	0%	100%	0%	0%	0%
INCORRETO	5.4%	0%	91.1%	0%	3.5%
ANOMALY	-	-	-	-	-
F1 SCORE	0.95	1.00	0.91	0.00	

Figura 11: *Resultado de detecção de anomalia*

iv. Embarcar

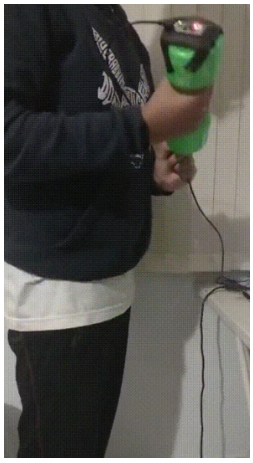
Para o teste com dispositivo embarcado, foi utilizado um haltere diferente, entretanto, a boa performance fora mantida. Na figura 12a é notável a maneira que o Arduino foi fixado no haltere, enquanto a 12b demonstra a detecção para a classe *idle*. Neste caso, os LEDs azul e verde acendem.

(a)

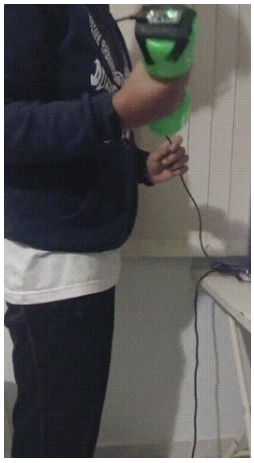
(b)

Figura 12: *Teste com dispositivo embarcado*

Quando o dispositivo detecta um movimento incorreto, o LED vermelho acende, como mostra a figura 13a. Já no caso de movimento correto o LED verde acende (figura 13b). Além disso, são enviadas mensagens para o computador através da comunicação serial. Em toda inferência são mostrados os valores das predições para cada classe, além do valor de anomalia. Se o movimento detectado for classificado como correto, o sistema também incrementa um contador que indica o número de repetições feitas pelo usuário (figura 14b).



(a)



(b)

Figura 13: *Sistema embarcado funcionando*

```
Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 22 ms., Classification: 0 ms., Anomaly: 2 ms.):
    correto: 0.00000
    idle: 0.00000
    incorreto: 0.99609
    anomaly score: -0.434
```

(a)

```
Starting inferencing in 2 seconds...
Sampling...
Predictions (DSP: 21 ms., Classification: 1 ms., Anomaly: 1 ms.):
    correto: 0.82422
Quantidade de movimentos corretos: 3
    idle: 0.00000
    incorreto: 0.17578
    anomaly score: -0.349
```

(b)

Figura 14: *Mensagens enviadas na serial*

IV. Resultados

Foi possível criar um modelo satisfatório, com aproximadamente 94% de acurácia, pequeno o suficiente para ser utilizado em dispositivos embarcados como o Arduino Nano 33 BLE. A figura 15 mostra a utilização de memória Flash, RAM e latência do modelo. Nota-se também que o modelo foi capaz de generalizar para outro tipo de halter mantendo uma boa precisão.

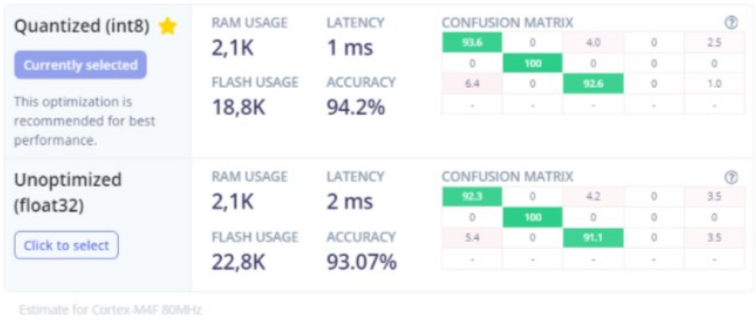
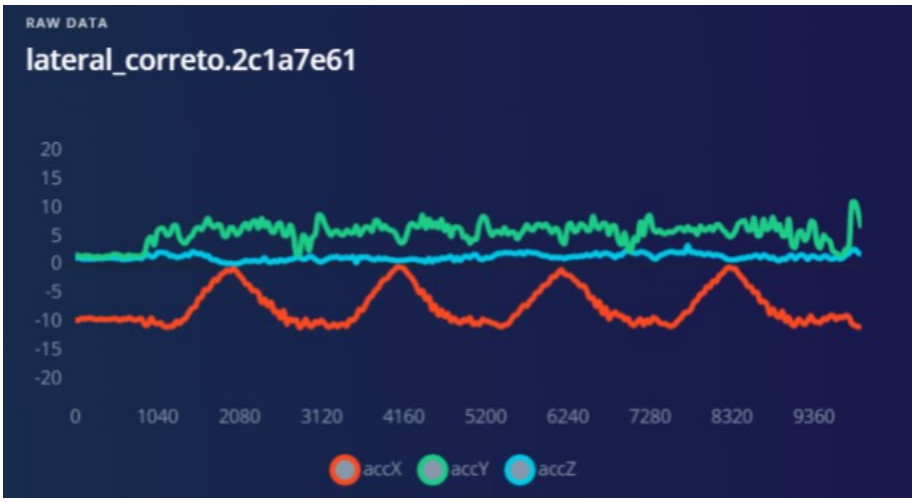


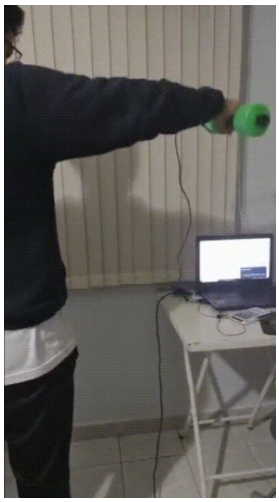
Figura 15: *Resultados do teste*

i. Expansão

Para generalizar o sistema é necessário adicionar novos exercícios. Nesse projeto, o exercício adicionado foi a Elevação lateral. Na figura 16a é mostrado o sinal capturado neste exercício, que é demonstrado na figura 16b.



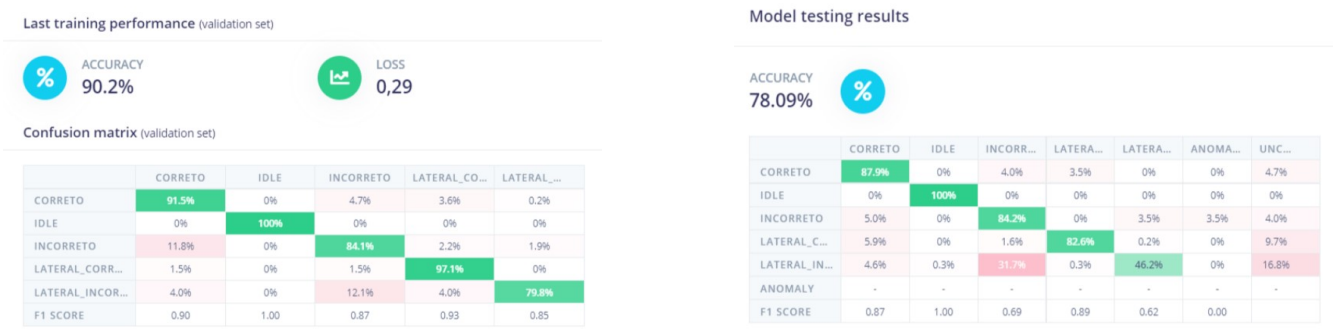
(a)



(b)

Figura 16: *Aquisição de dados para novo exercício*

Após o treinamento do modelo com as novas classes, percebeu-se uma diminuição considerável nas acurácias de treino e validação. Isto é um *trade off* comum na maioria dos sistemas de *machine learning*, e deve ser considerado na hora do desenvolvimento do sistema.



(a) **(b)**

Figura 17: *Resultados do novo exercício*

Além disso, pode-se perceber pelas figuras 17a e 17b que tanto nos dados de validação quanto nos dados de treinamento o modelo teve dificuldades em diferenciar os movimentos incorretos dos dois movimentos, mas teve uma boa precisão em diferenciar os movimentos corretos; ao olhar os *scores* individuais de cada classe pode-se ter uma melhor noção do desempenho de cada classe e dependendo da aplicação pode-se considerar os movimentos errados de uma forma única, já que, de forma geral, não importa se o classificado foi uma rosca direta incorreta ou uma elevação lateral incorreta, ambos estão incorretos.

V. Conclusão

TinyML é um campo sensacional. O que pode-se aprender ao longo das aulas é um conteúdo diferente de todos os outros durante a graduação, desde a primeira aula, pela gama de conceitos que foram apresentados até a apresentação dos projetos.

Este projeto foi escolhido entre diversas ideias levantadas durante alguns dias, entre elas a análise de expressões faciais de tristeza e um sistema de detecção vozes e responder a perguntas simples, mas os halteres eram a opção mais viável, levando em conta a conversa que tivemos com o Daniel Situnayake que nos motivou a ir em direção da experiência da criação de um *dataset* do "zero"e da aquisição de dados "em campo".

Desta maneira, o grupo se organizou e fez todo o processo descrito anteriormente, para que o relatório e o vídeo estivessem o mais bem estruturados possível dentro do prazo estipulado.

É importante ressaltar também o *bias* existente em nosso *dataset*, já que os dados foram coletados apenas usando homens na faixa etária de 15 à 23 anos. Para reduzir o *bias* é necessário coletar dados com pessoas de sexo, faixas etárias e condições físicas diferentes, além de pessoas com possíveis lesões e com movimentos restritos.

Por fim, o grupo formado por Gabriel, João, Pedro e Rodrigo agradece todo o conteúdo disponibilizado pelo professor Marcelo Rovai, assim como o grande incentivo ao estudo, respostas rápidas e respeitosas e o carinho pelos alunos. Todos foram aprendizes e pretendem utilizar estes conhecimentos em seus trabalhos e projetos futuros.

VI. Anexos

i. Projeto Edge Studio

O projeto criado no Edge Studio pode ser encontrado neste link.

ii. Jupyter Notebook

O Jupyter Notebook utilizado durante o desenvolvimento do projeto pode ser acessado aqui.

iii. Código Arduino

O código criado para este projeto é mostrado abaixo e pode ser encontrado aqui.

```
/* Edge Impulse Arduino examples
   Copyright (c) 2021 EdgeImpulse Inc.

   Permission is hereby granted, free of charge, to any person obtaining a copy
   of this software and associated documentation files (the "Software"), to deal
   in the Software without restriction, including without limitation the rights
   to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
   copies of the Software, and to permit persons to whom the Software is
   furnished to do so, subject to the following conditions:

   The above copyright notice and this permission notice shall be included in
   all copies or substantial portions of the Software.

   THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
   IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
   FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
   AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
   LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
   OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
   SOFTWARE.
*/
#include <dumbell_class_inferencing.h>
#include <Arduino_LSM9DS1.h>

#define CONVERT_G_TO_MS2    9.80665f
#define RED 22
#define BLUE 24
#define GREEN 23

static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
int count = 0;

void setup()
{
  Serial.begin(115200);
  Serial.println("Edge Impulse Inferencing Demo");


  pinMode(RED, OUTPUT);
  pinMode(BLUE, OUTPUT);
  pinMode(GREEN, OUTPUT);
  // Inicia os leds desligados (LOW -> ligado; HIGH -> desligado)
  digitalWrite(RED, HIGH);
  digitalWrite(GREEN, HIGH);
  digitalWrite(BLUE, HIGH);

  if (!IMU.begin()) {
    ei_printf("Failed to initialize IMU!\r\n");
  }
  else {
    ei_printf("IMU initialized\r\n");
  }

  if (EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME != 3) {
```

```
    ei_printf("ERR: EI_CLASSIFIER_RAW_SAMPLES_PER_FRAME should be equal to 3 (the 3 sensor axes)\n");
    return;
  }
}

void ei_printf(const char *format, ...) {
  static char print_buf[1024] = { 0 };

  va_list args;
  va_start(args, format);
  int r = vsnprintf(print_buf, sizeof(print_buf), format, args);
  va_end(args);

  if (r > 0) {
    Serial.write(print_buf);
  }
}

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

  delay(2000);

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

  // Allocate a buffer here for the values we'll read from the IMU
  float buffer[EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE] = { 0 };

  for (size_t ix = 0; ix < EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE; ix += 3) {
    // Determine the next tick (and then sleep later)
    uint64_t next_tick = micros() + (EI_CLASSIFIER_INTERVAL_MS * 1000);

    IMU.readAcceleration(buffer[ix], buffer[ix + 1], buffer[ix + 2]);

    buffer[ix + 0] *= CONVERT_G_TO_MS2;
    buffer[ix + 1] *= CONVERT_G_TO_MS2;
    buffer[ix + 2] *= CONVERT_G_TO_MS2;

    delayMicroseconds(next_tick - micros());
  }

  // Turn the raw buffer in a signal which we can the classify
  signal_t signal;
  int err = numpy::signal_from_buffer(buffer, EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE, &signal);
  if (err != 0) {
    ei_printf("Failed to create signal from buffer (%d)\n", err);
    return;
  }

  // Run the classifier
  ei_impulse_result_t result = { 0 };

  err = run_classifier(&signal, &result, debug_nn);
```

```
  if (err != EI_IMPULSE_OK) {
    ei_printf("ERR: Failed to run classifier (%d)\n", err);
    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 (result.classification[ix].value >= 0.8) // Tem mais de 80% de certeza sobre esse movimento?
    {
      if (result.classification[ix].label == "correto")
      {
        digitalWrite(RED, HIGH);
        digitalWrite(GREEN, LOW);
        digitalWrite(BLUE, HIGH);
        count++;
        Serial.println("Quantidade de movimentos corretos: " + String(count));
      } else if (result.classification[ix].label == "incorreto")
      {
        digitalWrite(RED, LOW);
        digitalWrite(GREEN, HIGH);
        digitalWrite(BLUE, HIGH);
      } else
      {
        digitalWrite(RED, HIGH);
        digitalWrite(GREEN, LOW);
        digitalWrite(BLUE, LOW);
      }
    }
  }
#if EI_CLASSIFIER_HAS_ANOMALY == 1
  ei_printf("    anomaly score: %.3f\n", result.anomaly);
  if (result.anomaly >= 0.3)
  {
    digitalWrite(RED, HIGH);
    digitalWrite(GREEN, HIGH);
    digitalWrite(BLUE, LOW);
  }
#endif
}

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