

Deteção de doenças em folhas de tomate

Introdução

A preservação das plantações de tomate é essencial para garantir a segurança alimentar e a sustentabilidade agrícola. Os tomateiros são suscetíveis a diversas doenças e pragas que podem afetar negativamente a produção e consequentemente a qualidade dos frutos. Portanto, é fundamental monitorar a saúde das plantações e identificar precocemente qualquer sinal de infestação. O projeto descrito neste documento consiste na concepção de um dispositivo que monitore uma plantação de tomates por meio de fotografias para a identificação de pragas existentes que tem como efeito a alteração das folhas dos tomateiros, com isso sendo possível mapear regiões de uma plantação onde haja a presença da praga e assim proporcionar a eliminação de maneira rápida e eficaz. O projeto é requisito parcial na avaliação da disciplina “TinyML - Aprendizado de Máquina Aplicado para Dispositivos IoT Embarcados” (IESTI01 - turma 01) do primeiro semestre de 2024 na Universidade Federal de Itajubá (UNIFEI - *campus* Itajubá).

Visando auxiliar os agricultores na manutenção da saúde das plantações e na tomada de decisões adequadas, foram mapeadas doenças por meio de fotografias de folhas de tomates que foram afetados pelas mesmas. Portanto, para isso utilizou-se o *dataset* “[Tomato leaf disease detection](#)” disponível na plataforma Kaggle pelo usuário Kaustubh B., o qual conta com 10 classes e cada uma dessas possui 1000 imagens para treinamento e mais 100 para validação/teste de exemplos de doenças presentes em tomateiros. As doenças analisadas são apresentadas a seguir.

- | | | |
|----------------------|----------------------|--------------------|
| • Bacterial spot | • Leaf mold | • Target spot |
| • Early blight | • Mosaic virus | • Yellow leaf curl |
| • Healthy (saudável) | • Septoria leaf spot | virus |
| • Late blight | • Spider mites | |

A execução do projeto se baseará nos conceitos de TinyML, da coleta dos dados até o *deployment* no dispositivo foi utilizado a plataforma Edge Impulse para desenvolvimento do projeto e assim então será usado como dispositivo o [Arduino Nano](#)

[33 BLE Sense](#) (Figura 1) em conjunto com a câmera [OmniVision OV7675](#) (Figura 2), os quais serão conectados por uma placa de desenvolvimento. O Arduino utilizado possui o processador de 32 bits ARM® Cortex™ M4 operando a 64 MHz, contando com 1MB de memória flash e 256KB de SRAM. A câmera VGA utilizada é um sensor de imagem CMOS que possui 0.3MP e suporte para resolução até 640x480 pixels.

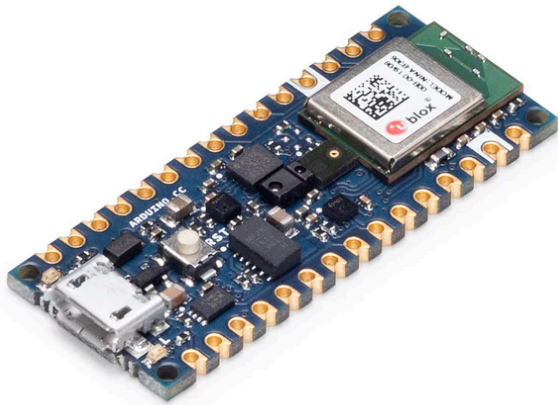


Figura 1

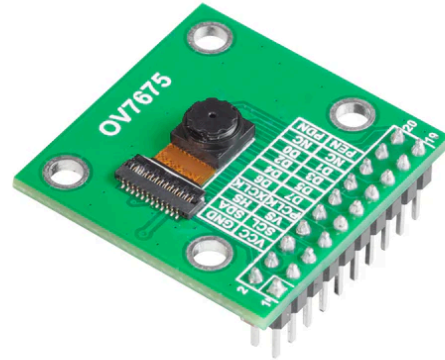


Figura 2

O sistema utilizará técnicas de aprendizado de máquina para análise das imagens capturadas, ou seja, um modelo de inferência previamente treinado será empregado para identificar padrões e características nas folhas que possam indicar a presença das pragas, gerando assim alertas para o usuário, permitindo uma intervenção rápida e direcionada. Aliado a isso, o dispositivo terá baixo consumo de energia e será de custo-benefício relativamente baixo, visto o custo dos dispositivos eletrônicos utilizados e as facilidades que o sistema pode trazer ao usuário.

Aquisição de dados

O projeto foi desenvolvido majoritariamente na plataforma Edge Impulse, com o *deploy* e pós-processamento sendo executado na Arduino IDE. Inicialmente idealizou-se o projeto para que o sistema conseguisse diferenciar as 10 classes de doenças de tomate presentes no *dataset*, contudo, após diversos testes na fase de treinamento utilizando diferentes arquiteturas de redes neurais e dimensionamento de hiperparâmetros, não foi possível obter métricas satisfatórias o suficiente para o prosseguimento do projeto com essas características. Portanto, reduziu-se a quantidade de classes para seis, uma delas contendo imagens de folhas saudáveis (classe *healthy*) e as outras cinco contendo imagens de diferentes doenças. A seleção das doenças teve como base a publicação "[Guia de identificação das doenças do tomateiro](#)" de LOPES, C. A. no portal da Empresa

Brasileira de Pesquisa Agropecuária (EMBRAPA) e na publicação [Tomato Mosaic Virus](#) de BLANCARD D. no portal Ephytia do *Institut national de la recherche agronomique* (INRAE). A definição dos sintomas dessas doenças pelos autores e imagens de exemplos do *dataset* são apresentadas a seguir.

- Mancha bacteriana (Bacterial spot - *Xanthomonas campestris* pv. *vesicatoria*): Manchas necróticas, concentrando-se nas bordas das folhas, que evoluem para uma "queima" da folhagem de baixo para cima. Caule, pecíolo e pedúnculo também ficam manchados. No fruto, as lesões são inicialmente esbranquiçadas, evoluindo para uma cor marrom, podendo atingir até 5 mm de diâmetro.



Figura 3

- Pinta preta (Early blight - *Alternaria solani*): Lesões escuras, com as bordas bem definidas, frequentemente com anéis concêntricos, mais evidentes nas folhas mais velhas, afetando também o caule. A doença provoca desfolha pela seca das folhas baixas. Nos frutos, causa um apodrecimento escuro, de consistência firme, na região do pedúnculo.



Figura 4

- Requeima (Late blight - *Phytophthora infestans*): Lesões grandes, marrom-escuras, encharcadas, aparecendo primeiro nas folhas mais novas, evoluindo para uma

"queima" ou "mela" geral da planta. No início do ataque da doença, é comum observar-se um escurecimento do caule no topo da planta, que se torna quebradiço. Frutos afetados apodrecem, ficam amarronzados, porém permanecem com a consistência firme.



Figura 5

- Mosaico do tomateiro (Mosaic virus - *Tomato mosaic virus ToMV*): Aparecem manchas amarelas e deformação das folhas, que ficam em forma de colher, já os frutos apresentam manchas marrons, onde é comum aparecerem anéis mais claros. As plantas infectadas ficam subdesenvolvidas e com crescimento atrofiado.



Figura 6

- Septoriose (Septoria leaf spot - *Septoria lycopersici*): Aparecem inicialmente nas folhas mais velhas e consistem de manchas arredondadas, marrom escuras, com o centro cinza-claro, onde pontuações negras podem ser visíveis. O coalescimento das lesões provoca seca das folhas. Lesões podem aparecer também no caule, pedúnculo e cálice, mas raramente nos frutos.



Figura 7

Sendo assim, no Edge Impulse (Figura 8), as 1000 imagens de treinamento do *dataset* foram carregadas e separadas em 80% para treinamento e 20% para teste mantendo o balanceamento das classes, e as 100 imagens de validação/teste do *dataset* foram destinadas para futuro teste com o dispositivo.

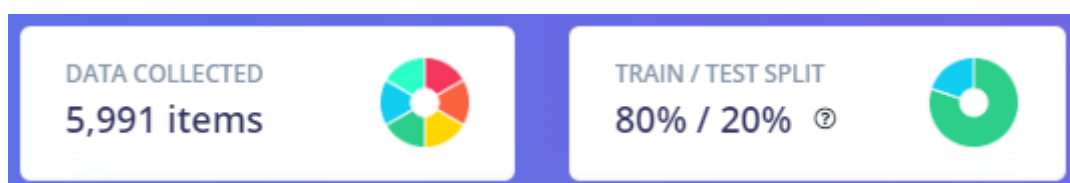


Figura 8

Desenho do impulso

Após diversos testes manuais, alterando a arquitetura da rede neural entre Redes Neurais Convolucionais (CNN) e seu parâmetros e o uso de *Transfer Learning* com as versões do MobileNet presentes na plataforma, os resultados não foram satisfatórios o suficiente e fez-se necessário o uso portanto da ferramenta EON Turner da plataforma. A arquitetura ótima encontrada pela ferramenta com base na acurácia, na latência de inferência e na memória RAM utilizada pelo modelo foi uma CNN que será apresentada posteriormente. Redes neurais com *Transfer Learning* não lidaram muito bem com esse projeto, portanto decidiu-se utilizar a CNN indicada. Sendo assim, o desenho do impulso conforme recomendado pela plataforma, é apresentado na Figura 9.

Na etapa de pré-processamento das imagens, as imagens que inicialmente são fotografadas em escala RGB serão normalizadas para escala de cinza (*grayscale*), conforme definido na plataforma. Em seguida, os *features* para o modelo foram gerados e 4787 janelas foram criadas. A Figura 10 apresenta a respectiva visualização em 2D dos *features* gerados a partir dos dados de treinamento.

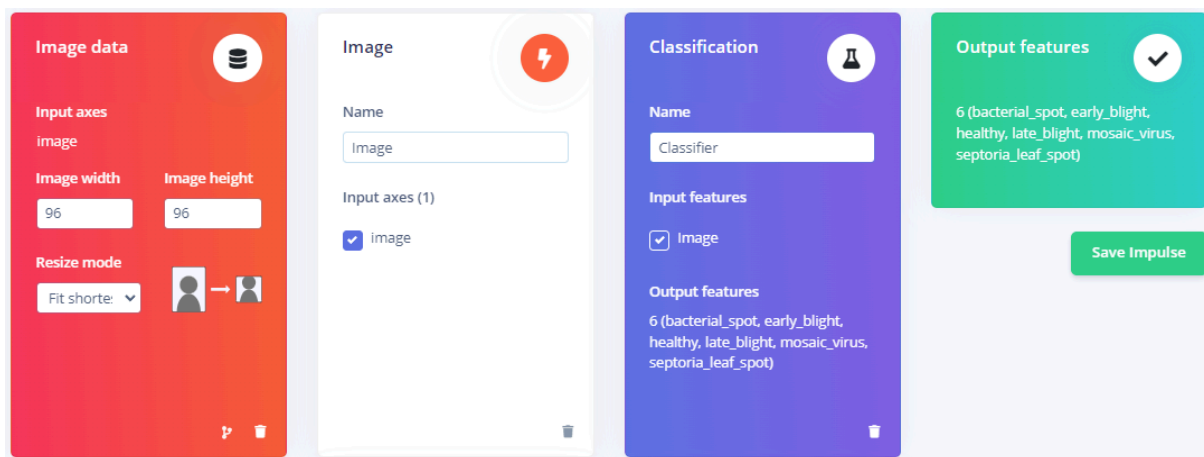


Figura 9

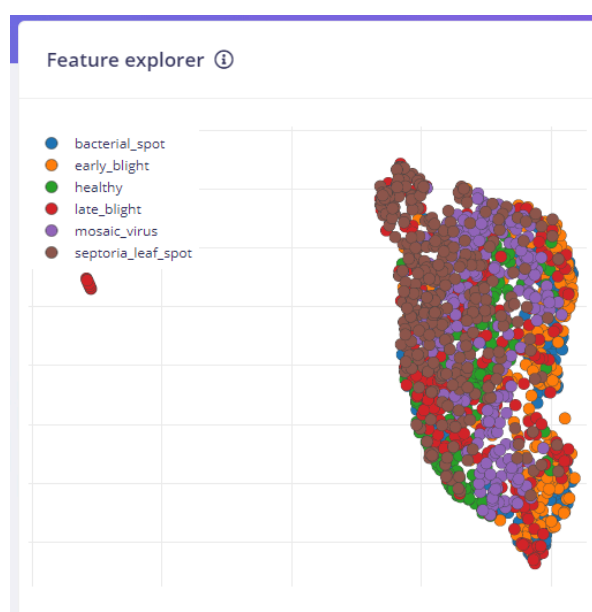


Figura 10

Treinamento

Na etapa de treinamento, com o modelo indicado pelo EON Turner já aplicado, alterou-se os parâmetros de treinamento até se obter um modelo ótimo para a rede neural apresentada na Figura 11. Portanto, com 25 ciclos de treinamento, taxa de aprendizado igual a 0.001, largura do *batch* igual a 32 e 20% dos dados de treinamento destinados ao *set* de validação, obteve-se os resultados apresentados na Figura 12.



Figura 11



Figura 12

O valor para a acurácia de validação igual a 86,8% é satisfatória porém a *loss* igual a 0,39 pode afetar negativamente o desempenho do sistema. Embora os resultados apresentem erros, eles não são maioria, o que pode ser observado pela Figura 13 na exploração dos dados. Na estimativa da performance do modelo no dispositivo, a latência de inferência não é significativa assim como o uso da memória RAM do dispositivo.

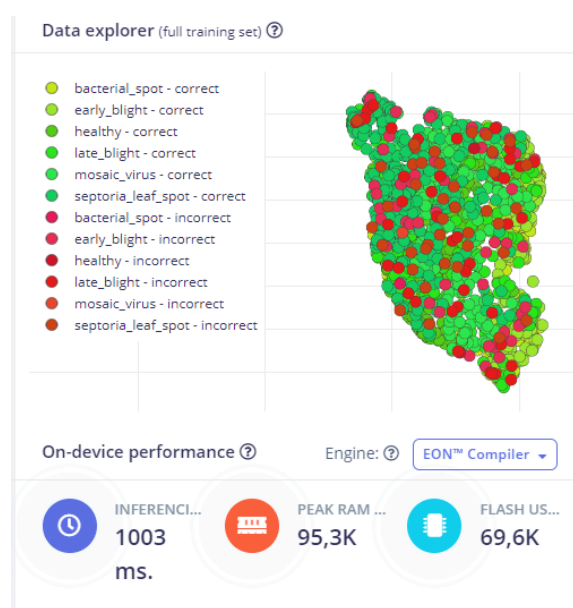


Figura 13

Teste do modelo

Na etapa de teste do modelo, com os dados separados para teste e uso do modelo não otimizado (float32) para o dispositivo, obteve-se 80,73% de acurácia nas inferências, o que motivou o prosseguimento do projeto. Os resultados do teste podem ser observados nas imagens apresentadas a seguir, junto à matriz de confusão do mesmo.

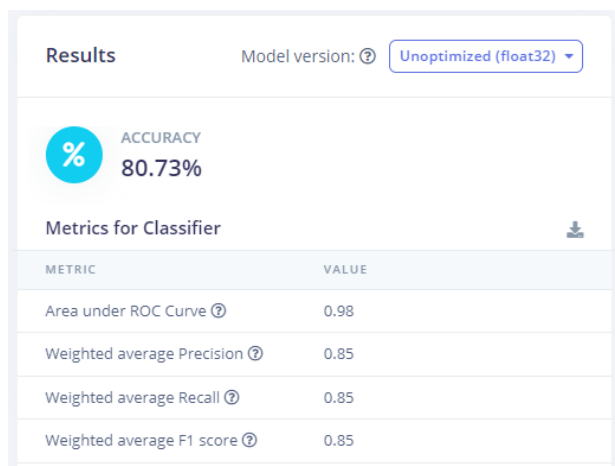


Figura 14

Confusion matrix

	BACTER	EARLY_E	HEALTH	LATE_BL	MOSAIC	SEPTOR	UNCERT
BACTERIAL	92.6%	1.0%	0%	0%	0%	1.5%	5.0%
EARLY_BLI	4.5%	62.8%	0%	9.5%	0.5%	4.0%	18.6%
HEALTHY	0%	0%	97.1%	0%	0%	0.5%	2.5%
LATE_BLI	1.5%	7.5%	3.0%	62.8%	0.5%	6.0%	18.6%
MOSAIC_V	0%	0.5%	0%	0%	96.5%	1%	2%
SEPTORIA	2.5%	2.5%	1%	2.5%	3.5%	72%	16%
F1 SCORE	0.92	0.72	0.97	0.72	0.96	0.78	

Figura 15

Deployment e pós-processamento

Na etapa do *deploy* do modelo para o dispositivo, gerou-se a biblioteca Arduino na plataforma, e em seguida ela foi incluída na Arduino IDE. A biblioteca gerada pelo Edge Impulse contém um exemplo para teste do modelo com a câmera selecionada, e seguindo os devidos passos o exemplo foi carregado no Arduino. O exemplo retorna no monitor serial as probabilidades da imagem capturada em cada período de tempo de pertencer a alguma das 6 classes do modelo. Em seguida é apresentado um exemplo do resultado de uma inferência.

```
Starting inferencing in 2 seconds...
Taking photo...
Predictions (DSP: 26 ms., Classification: 1077 ms., Anomaly: 0 ms.):
Predictions:
bacterial_spot: 0.06641
early_blight: 0.00391
healthy: 0.00781
late_blight: 0.89453
mosaic_virus: 0.00000
septoria_leaf_spot: 0.02734
```

Tabela 1

Para facilitar a leitura de qual é a classe da doença inferida pelo modelo e assim então criar uma aplicação *standalone*, na etapa de pós-processamento elaborou-se uma lógica para que a luz do *LED* RGB do Arduino acenda de uma cor diferente para a classe

com a maior probabilidade entre as seis presentes no modelo, sendo que cada classe assim possui uma cor referente. As cores designadas são apresentadas a seguir.

- Bacterial spot (Mancha bacteriana) → *LED* rosa aceso ◐
- Early blight (Pinta preta) → *LED* azul aceso ◑
- Healthy (Saudável) → *LED* verde aceso ◑
- Late blight (Requeima) → *LED* vermelho aceso ◑
- Mosaic virus (Mosaico do tomateiro) → *LED* ciano aceso ◑
- Septoria leaf spot (Septoriose) → *LED* amarelo aceso ◑

Para isso, modificou-se o exemplo gerado pela plataforma para que tal lógica ocorresse ao modelo inferir uma nova entrada. Devido a extensão do código, no Anexo A é possível conferir as função *setup()* e *loop()* já com as devidas modificações, e as duas funções *turn_off_leds()* e *turn_on_leds()* que foram elaboradas. O código completo pode ser obtido na pasta do Google Drive a seguir: [nano_ble33_sense_camera.ino](https://drive.google.com/drive/folders/1U33333333333333333333333333333333). Com a modificação, o código ainda retorna no monitor serial as probabilidades mencionadas anteriormente, com o acréscimo de uma linha (Tabela 2) que fala qual é a classe com a maior probabilidade entre as seis do modelo, sendo assim aquela que o modelo inferiu estar presente na imagem capturada.

```
Starting inferencing in 2 seconds...
Taking photo...
Predictions (DSP: 26 ms., Classification: 1076 ms., Anomaly: 0 ms.):
Predictions:
bacterial_spot: 0.19922
early_blight: 0.00391
healthy: 0.01172
late_blight: 0.73047
mosaic_virus: 0.00000
septoria_leaf_spot: 0.05469
:
Prediction is late_blight with probability 0.73047
:
```

Tabela 2

Teste com o dispositivo

Por fim, o teste do modelo com o pós-processamento implementado operando no Arduino foi executado com o auxílio da Arduino IDE. Para o teste, selecionou-se para cada uma das seis classes duas imagens do set de validação/teste do *dataset*, imagens as quais são desconhecidas para o modelo. Em seguida imprimiu-se em uma folha A4 as 12 imagens resultantes, e colou-se as folhas na parede para que a câmera em conjunto o Arduino possa capturar e inferir a doença com o modelo carregado. As imagens impressas podem ser conferidas a seguir (Tabela 3) e a Figura 16 demonstra como ficou a

montagem do ambiente para o teste com o dispositivo.













 bacterial_spot	 bacterial_spot	 early_blight	 early_blight
 healthy	 healthy	 late_blight	 late_blight
 mosaic_virus	 mosaic_virus	 septoria_leaf_spot	 septoria_leaf_spot

Tabela 3

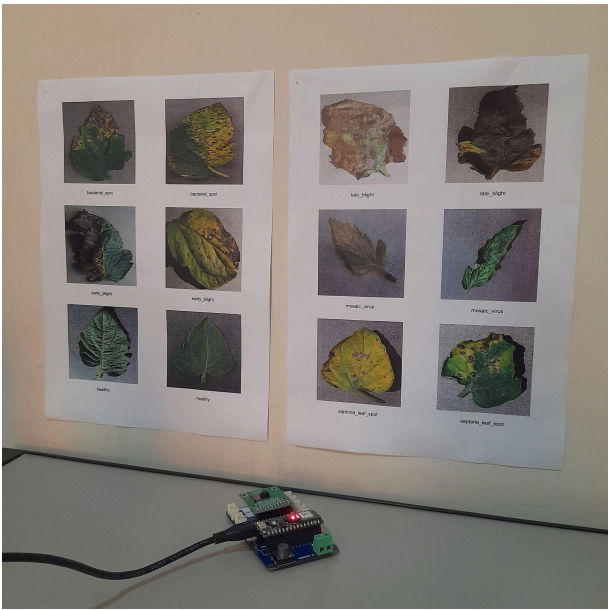


Figura 16

Com os testes realizados, verificou-se que o dispositivo não conseguiu obter bom desempenho com as imagens impressas. As classes *late_blight*, *bacterial_spot* e *healthy* obtiveram bom desempenho com o dispositivo reconhecendo as suas características, porém com as outras classes o desempenho não foi o mesmo. A resposta no *LED* divergiu entre *late_blight* e *bacterial_spot* para as classes *early_blight*, *mosaic_virus* e *septoria_leaf_spot*. As falhas podem ter sido ocasionadas pelos seguintes fatores:

1. A impressão das imagens possui um tom mais escuro do que as imagens virtuais, o que pode levar ao modelo inferir erroneamente, e assim a resposta para folhas de tom mais escuro.
2. A falta de iluminação no local também pode afetar o reconhecimento, porém testou-se o modelo com as folhas ao sol, e mesmo assim o desempenho foi semelhante.
3. A câmera utilizada possui pequena abertura, o que pode levar ao escurecimento da imagem, e assim ao erro na inferência.
4. A métrica *loss* do modelo também não é o ideal, quanto mais baixo o valor, maiores as chances da identificação correta das doenças.
5. Capturar uma fotografia de uma outra fotografia e ainda impressa pode afetar a qualidade da entrada do modelo, sendo assim a ausência de exemplares de folhas de tomates adoecidos pode ter impactado negativamente também.

Contudo, melhorias em todas as etapas do projeto podem ser empregadas para que se obtenha resultados melhores.

1. Embora a Edge Impulse forneça muitas facilidades, realizar o desenvolvimento do projeto fora da plataforma, como por exemplo no Google Colab, e usar a plataforma apenas para a realização do *deploy* no dispositivo pode levar a melhores resultados, pois assim terá maior liberdade para o desenvolvimento como por exemplo a restrição de tempo (20 minutos) de treinamento que é imposta na plataforma.
2. Embora o Arduino utilizado seja capaz de trabalhar com imagens, existem outros dispositivos que operam melhor com esse tipo de projeto, como por exemplo o XIAO ESP32S3. A utilização de uma câmera de resolução maior pode levar a entrada de dados mais precisos no modelo, e dispositivos com maior capacidade de memória e maior velocidade de *clock* podem armazenar redes neurais mais complexas e processar a inferência de maneira mais rápida e eficaz.

3. Ter em mãos exemplares de folhas de tomate adoecidos também pode auxiliar no desenvolvimento devido a praticidade de testar e aprimorar o modelo, assim como levar a resultados mais fidedignos.

Conclusão

Tendo em vista os dados apresentados, com a realização do projeto foi possível ter uma experiência de como ocorre a execução de um projeto real abordando os conceitos de TinyML e que possui uma finalidade bem definida. As listas de exercícios e os laboratórios realizados na disciplina auxiliaram também a elucidar questões e ter uma visão mais ampla sobre projetos da área mencionada, entretanto as circunstâncias desse projeto trouxeram novas dificuldades e esclarecimentos os quais proporcionaram uma visão ampliada de um projeto efetivo de TinyML. Embora no final os testes com o dispositivo não tenham ocorrido da maneira como era esperado, os conhecimentos acerca do TinyML foram aprimorados.

O projeto com as devidas melhorias indicadas pode ser capaz de se tornar uma ferramenta útil a um produtor agrícola que deseja um auxílio na identificação e combate de pragas em sua plantação de tomates. Com a inclusão de uma bateria ao dispositivo o sistema poderá operar em qualquer ambiente de maneira integral.

Vale ressaltar que na proposta do projeto foi solicitado a elaboração de um vídeo de apresentação da realização do projeto, porém o vídeo não foi criado devido a má gestão do tempo pelos integrantes do grupo perante a execução de todo o projeto.

Anexos

Anexo A

```
/**
 * @brief      Arduino setup function
 */
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("Tomato leaf disease detection");
}
```



```

pinMode(LED_R, OUTPUT);
pinMode(LED_G, OUTPUT);
pinMode(LED_B, OUTPUT);

digitalWrite(LED_R, HIGH);
digitalWrite(LED_G, HIGH);
digitalWrite(LED_B, HIGH);

// summary of inferencing settings (from model_metadata.h)
ei_printf("Inferencing settings:\n");
ei_printf("\tImage resolution: %dx%d\n", EI_CLASSIFIER_INPUT_WIDTH,
EI_CLASSIFIER_INPUT_HEIGHT);
ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
ei_printf("\tNo. of classes: %d\n",
sizeof(ei_classifier_inferencing_categories) /
sizeof(ei_classifier_inferencing_categories[0]));
}

```

Função *setup()*

```

void turn_off_leds() {
    digitalWrite(LED_R, HIGH);
    digitalWrite(LED_G, HIGH);
    digitalWrite(LED_B, HIGH);
}

void turn_on_leds(int pred_index) {
    switch(pred_index) {
        case 0:
            turn_off_leds();
            digitalWrite(LED_R, LOW);
            digitalWrite(LED_B, LOW);
            break;
        case 1:
            digitalWrite(LED_B, LOW);
            turn_off_leds();
            break;
        case 2:
            turn_off_leds();
            digitalWrite(LED_G, LOW);
            break;
        case 3:
            turn_off_leds();
            digitalWrite(LED_R, LOW);

```

```

        break;
    case 4:
        turn_off_leds();
        digitalWrite(LEDG, LOW);
        digitalWrite(LEDDB, LOW);
        break;
    case 5:
        turn_off_leds();
        digitalWrite(LEDRA, LOW);
        digitalWrite(LEDG, LOW);
        break;
}
}

```

Funções *turn_off_leds()* e *turn_on_leds()*

```

/**
 * @brief      Get data and run inferencing
 *
 * @param[in]  debug  Get debug info if true
 */
void loop()
{
    bool stop_inferencing = false;

    while(stop_inferencing == false) {
        ei_printf("\nStarting inferencing in 2 seconds...\n");

        // instead of wait_ms, we'll wait on the signal, this allows
        threads to cancel us...
        if (ei_sleep(2000) != EI_IMPULSE_OK) {
            break;
        }

        ei_printf("Taking photo...\n");

        if (ei_camera_init() == false) {
            ei_printf("ERR: Failed to initialize image sensor\r\n");
            break;
        }

        // choose resize dimensions
        uint32_t resize_col_sz;
        uint32_t resize_row_sz;
    }
}

```

```

        bool do_resize = false;
        int res = calculate_resize_dimensions(EI_CLASSIFIER_INPUT_WIDTH,
        EI_CLASSIFIER_INPUT_HEIGHT, &resize_col_sz, &resize_row_sz, &do_resize);
        if (res) {
            ei_printf("ERR: Failed to calculate resize dimensions
        (%d)\r\n", res);
            break;
        }

        void *snapshot_mem = NULL;
        uint8_t *snapshot_buf = NULL;
        snapshot_mem = ei_malloc(resize_col_sz*resize_row_sz*2);
        if(snapshot_mem == NULL) {
            ei_printf("failed to create snapshot_mem\r\n");
            break;
        }
        snapshot_buf = (uint8_t
        *)DWORD_ALIGN_PTR((uintptr_t) snapshot_mem);

        if (ei_camera_capture(EI_CLASSIFIER_INPUT_WIDTH,
        EI_CLASSIFIER_INPUT_HEIGHT, snapshot_buf) == false) {
            ei_printf("Failed to capture image\r\n");
            if (snapshot_mem) ei_free(snapshot_mem);
            break;
        }

        ei::signal_t signal;
        signal.total_length = EI_CLASSIFIER_INPUT_WIDTH *
        EI_CLASSIFIER_INPUT_HEIGHT;
        signal.get_data = &ei_camera_cutout_get_data;

        // run the impulse: DSP, neural network and the Anomaly algorithm
        ei_impulse_result_t result = { 0 };

        EI_IMPULSE_ERROR ei_error = run_classifier(&signal, &result,
        debug_nn);
        if (ei_error != EI_IMPULSE_OK) {
            ei_printf("Failed to run impulse (%d)\n", ei_error);
            ei_free(snapshot_mem);
            break;
        }

        // print the predictions
        ei_printf("Predictions (DSP: %d ms., Classification: %d ms.,

```

```

Anomaly: %d ms.): \n",
        result.timing.dsp, result.timing.classification,
result.timing.anomaly);
#if EI_CLASSIFIER_OBJECT_DETECTION == 1
    ei_printf("Object detection bounding boxes:\r\n");
    for (uint32_t i = 0; i < result.bounding_boxes_count; i++) {
        ei_impulse_result_bounding_box_t bb =
result.bounding_boxes[i];
        if (bb.value == 0) {
            continue;
        }
        ei_printf("  %s (%f) [ x: %u, y: %u, width: %u, height: %u
]\r\n",
                bb.label,
                bb.value,
                bb.x,
                bb.y,
                bb.width,
                bb.height);
    }

    // Print the prediction results (classification)
#else
    int pred_index = 0;
    float pred_value = 0;
    ei_printf("Predictions:\r\n");
    for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
        ei_printf("  %s: ", ei_classifier_inferencing_categories[i]);
        ei_printf("%.5f\r\n", result.classification[i].value);
        if(result.classification[i].value > pred_value) {
            pred_index = i;
            pred_value = result.classification[i].value;
        }
    }
    ei_printf(": \n");
    ei_printf(" Prediction is %s with probability %.5f\n",
        result.classification[pred_index].label, pred_value);
    ei_printf(": \n");
    turn_on_leds(pred_index);

#endif

    // Print anomaly result (if it exists)
#if EI_CLASSIFIER_HAS_ANOMALY

```



```

        ei_printf("Anomaly prediction: %.3f\r\n", result.anomaly);
#endif

#if EI_CLASSIFIER_HAS_VISUAL_ANOMALY
    ei_printf("Visual anomalies:\r\n");
    for (uint32_t i = 0; i < result.visual_ad_count; i++) {
        ei_impulse_result_bounding_box_t bb =
result.visual_ad_grid_cells[i];
        if (bb.value == 0) {
            continue;
        }
        ei_printf("  %s (%f) [ x: %u, y: %u, width: %u, height: %u
]\r\n",
                bb.label,
                bb.value,
                bb.x,
                bb.y,
                bb.width,
                bb.height);
    }
#endif

    while (ei_get_serial_available() > 0) {
        if (ei_get_serial_byte() == 'b') {
            ei_printf("Inferencing stopped by user\r\n");
            stop_inferencing = true;
        }
    }
    if (snapshot_mem) ei_free(snapshot_mem);
}
ei_camera_deinit();
}

```

Função *loop()*

Referências

- [1] ARDUINO. **Datasheet: Arduino Nano 33 BLE Sense**, 2024. Disponível em: <https://docs.arduino.cc/resources/datasheets/ABX00031-datasheet.pdf>. Acesso em: 3 de maio de 2024.
- [2] BLANCARD, D. **Tomato mosaic virus**. Ephytia, 2023. Disponível em: <https://ephytia.inra.fr/pt/C/5020/Tomate-Tomato-mosaic-virus-ToMV>. Acesso em: 27 de junho de 2024.

- [3] KAUSTUBH, B. **Tomato leaf disease detection**. Kaggle, 2020. Disponível em: <https://www.kaggle.com/datasets/kaustubhb999/tomatoleaf/data>. Acesso em: 3 de maio de 2024.
- [4] LOPES, Carlos Alberto. **Guia de identificação das doenças do tomateiro**. Embrapa, 2018. Brasília: Embrapa Hortaliças. Zeneca, 2000. Disponível em: <https://www.embrapa.br/busca-de-publicacoes/-/publicacao/768979/guia-de-identificacao-das-doencas-do-tomateiro>. Acesso em: 27 de junho de 2024.
- [5] OMNIVISION. **Product guide: OmniVision OV7675**, 2024. Disponível em: <https://www.ovt.com/products/ov7675/>. Acesso em: 3 de maio de 2024.