Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

Renato da Silva Vilela

Trabalho de Conclusão de Curso

Solução IoT para Digitalização de Medidores de Consumo por Visão Computacional e MicroML

Campina Grande, Paraíba
22 de outubro de 2021

Renato da Silva Vilela

Solução IoT para Digitalização de Medidores de Consumo por Visão Computacional e MicroML

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Departamento de Engenharia Elétrica - DEE

Orientador: Rafael Bezerra Correia Lima, D. Sc.

Campina Grande, Paraíba
22 de outubro de 2021

Renato da Silva Vilela

Solução IoT para Digitalização de Medidores de Consumo por Visão Computacional e MicroML

Trabalho de Conclusão de Curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Trabalho aprovado em: ____/____/2021.

Rafael Bezerra Correia Lima, D. Sc.
Orientador

George Acioli Júnior, D. Sc.
Convidado

Campina Grande, Paraíba
22 de outubro de 2021

Agradecimentos

Agradeço aos meus familiares, por todo o apoio e paciência.

Aos meus amigos de curso, pelo bons momentos na universidade, por todos os conselhos e dicas.

Ao meu orientador, Prof. Rafael Bezerra por sua orientação, disponibilidade e paciência. Este trabalho não teria sido concluído sem sua participação.

Por fim, agradeço a todos que contribuíram nessa parte da minha historia, à instituição UFCG, ao departamento de engenharia elétrica e aos professores.

Resumo

Neste trabalho é apresentada uma solução *IoT* (*Internet of things*) para a digitalização de medidores de consumo analógicos. A solução é baseado no uso de visão computacional e aprendizado de máquina. O trabalho proposto apresenta um protótipo para aquisição, classificação e envio de dados, baseado no módulo ESP32-CAM com Câmera OV2640 2MP. O protótipo realiza o reconhecimento do valor da unidade do consumo, atualiza o valor e envia o valor atualizado para um banco de dados. O trabalho proposto apresenta uma aplicação *web* para o monitoramento do consumo de forma gráfica. Por fim, foi realizada uma análise da viabilidade da aplicação.
Palavras-chaves: Internet das Coisas, Visão Computacional, Aprendizado de Máquina, Sistema Embarcado.

Abstract

In this work are presented an IoT (Internet of things) solution for digitizing analog consumption meters. The solution is based on the use of computer vision and machine learning. The proposed work presents a prototype for data acquisition, classification and sending, based on the ESP32-CAM module with OV2640 2MP Camera. The prototype performs the recognition of the consumption unit value, updates the value and sends the updated value to a database. The proposed work presents a web application for monitoring consumption graphically. Finally, an analysis of the feasibility of the application was carried out.
Key-words: Internet of Things, Computer Vision, Machine Learning, Embedded System.

Lista de Figuras

Lista de Tabelas

Lista de símbolos e abreviaturas

AD	Árvore de Decisão
AM	Aprendizado de Máquina
CART	*Classification and Regression Trees*
EQM	Erro Quadrático Médio
FA	Floresta Aleatória
IDE	*Integrated Development Environment*
IoT	*Internet of Things*

Sumário

1 Introdução

O procedimento de aferição da medição do consumo, seja de energia elétrica ou água, é realizado manualmente em medidores que, na maioria dos casos, não possuem *display* digital. Dessa forma, esse processo é suscetível a falhas, além disso, nem sempre os medidores possuem boa visibilidade ou estão em locais insalubres e de difícil acesso.

Com processo de aferição da medição automatizada, é possível capturar as informações em uma taxa muito maior, possibilitando a extração de tendências de consumo e possivelmente uma futura otimização. Dessa forma, possibilitando, por exemplo: detectar vazamentos de água, detectar consumo atípico.

Nesse contexto, a proposta deste trabalho é propor uma solução *IoT* (*Internet of things*) para a digitalização desses medidores e automatização do processo de aferição da medição do consumo. De uma forma geral, compreende-se por *IoT* a maneira com que objetos físicos do nosso cotidiano se interconectam na *web*, muitos destes equipamentos estão incorporados com inteligência ubíqua e são controlados pela Internet. Os avanços em tecnologias subjacentes permitiram que "coisas" possam ser identificadas, detectadas e controladas remotamente usando sensores e atuadores (FENG et al., 2012).

Em geral, os dados dos dispositivos *IoT* são processados em Nuvem (*Cloud Computing*), entretanto, é possível conseguir melhorias de desempenho realizando o processamento dos dados localmente, mais próximo da fonte, diminuindo o tempo de resposta.

Segundo SAKR et al. (2020), a mudança de Computação em Nuvem (*Cloud Computing*) para Computação de Borda (*Edge Computing*) tem vantagens em termos de latência na resposta, ocupação de largura de banda, consumo de energia, segurança e privacidade.

Essa mudança também inclui às técnicas de Aprendizado de Máquina (*Machine Learning*), principalmente para a tarefa de inferência (acionamento de atuador, envio de mensagem através de transmissor (*transceiver*) de comunicação), pois é necessário menos poder computacional nessa fase do que na fase de treinamento.

AM é uma área de pesquisa da Inteligência Artificial que visa ao desenvolvimento de programas de computador com a capacidade de aprender a executar uma dada tarefa com sua própria experiência (FACELI et al., 2011).

Portanto, para a solução proposta, a tarefa de inferência do algorítimo de AM, será realizada localmente por um microcontrolador (ESP32). Para o desenvolvido do modelo será utilizado MicroML.

MicroML é um projeto que tem como objetivo implementar algoritmos de

ML para microcontroladores. Ele nasceu como uma alternativa ao Tensorflow para Microcontroladores, que é exclusivamente dedicado a Redes Neurais Artificiais. MicroML oferece alternativas mais leves às redes neurais para executar inferência mesmo em microcontroladores de 8 bits.

Para o solução proposta o algoritmo de AM utilizado será o de Floresta Aleatória (*Random Forest*, em inglês), sua categoria é aprendizado supervisionado. O algoritmo de Floresta Aleatória cria várias Árvores de decisão e as combina para obter uma predição.

1.1 Objetivos

Este trabalho tem como objetivo geral propor uma solução IoT para a digitalização de medidores de consumo analógicos (energia, água e gás), e assim, automatizar o processo de aferição da medição do consumo.

Para que o objetivo geral seja alcançado, são definidos os seguintes objetivos específicos:

- Criação, treino e otimização de um modelo de AM com Scikit-Learn e MicroML;
- Utilização do modelo criado em um microcontrolador (ESP32) para o reconhecimento do dígito correspondente a unidade do consumo;
- Atualização do valor do consumo com o novo valor de unidade inferido;
- Envio das informações referente ao consumo para um banco de dados;
- Visualização dos dados de consumo;
- Validação dos resultados obtidos;
- Análise da viabilidade da aplicação.

1.2 Estruturação do Trabalho

O documento está dividido em 5 capítulos. No Capítulo 1 são apresentadas a motivação associada ao desenvolvimento deste trabalho, os objetivos e a organização do texto.

No Capítulo 2 é apresentada a fundamentação teórica sobre temas pertinentes ao trabalho.

No Capítulo 3 é apresentada a metologia utilizada para alcançar os objetivos do trabalho.

No Capítulo 4 é apresentada a análise dos resultados obtidos.

No Capítulo 5 são apresentadas as conclusões.

2 Fundamentação Teórica

2.1 Aprendizado de Máquina

Aprendizado de Máquina (AM) é um ramo da Inteligência Artificial que visa o desenvolvimento de programas de computador que possam aprender com os dados.

Uma definição mais precisa: diz-se que um programa de computador aprende pela experiência E em relação a algum tipo de tarefa T e alguma medida de desempenho P se o seu desempenho em T, conforme medido por P, melhora com a experiência E (MITCHELL, 1997).

As técnicas de AM vêm sendo usadas na solução de problemas reais em que as técnicas de programação tradicionais tornam-se complexas, fazendo com que a manutenção do programa seja muito difícil. Entre as principais aplicações, podem ser citadas:

- Reconhecimento de palavras faladas;
- Reconhecimento ótico de caracteres (OCR);
- Detecção de *e-mails* como *spam*;
- Detecção do uso fraudulento de cartões de crédito;
- Condução de automóveis de forma autônoma;
- Predição de taxas de cura de pacientes;
- Diagnóstico de câncer pela análise de dados de expressão gênica;
- Programas de computador (*engines*) que jogam xadrez de forma melhor que humanos.

As técnicas de AM são classificados de acordo com a quantidade e o tipo de supervisão que recebem durante o treinamento. As principais categorias são: supervisionado, não supervisionado, semi-supervisionado e por reforço.

Nesse trabalho o foco será o aprendizado supervisionado.

2.1.1 Aprendizado Supervisionado

Nos algorítimos do tipo supervisionado, o modelo é treinado a partir de resultados desejados, ou seja, o modelo possui uma referência do que é certo e errado. Por exemplo: Em um filtro de *spam* o modelo é treinado com muitos

exemplos de *e-mails* classificados em *spam* e não *spam*, e deve aprender a classificar novos *e-mails*.

Alguns dos algorítimos de aprendizado supervisionado mais importantes:

- k-Nearest Neighbours;
- Regressão Linear;
- Regressão Logística;
- Máquina de Vetores de Suporte (SVM);
- Árvore de Decisão e Floresta Aleatória;
- Redes Neurais.

As Redes neurais podem ser não supervisionadas, e também podem ser semi-supervisionadas.

2.2 Árvore de Decisão

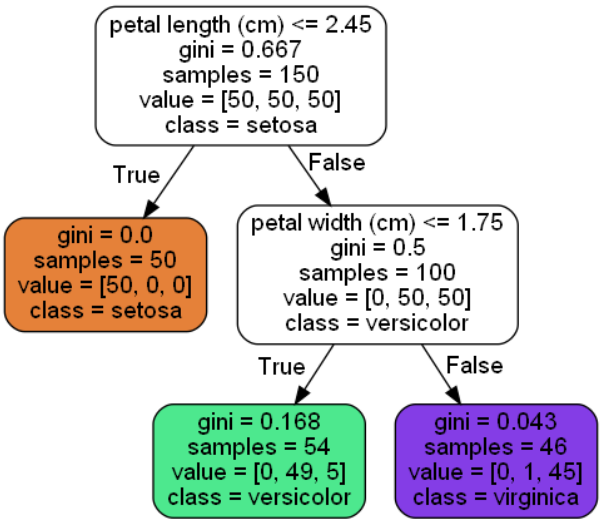
Uma árvore é uma coleção de elementos chamados nós, dentre os quais um é distinguido como uma raiz, juntamente com uma relação de "paternidade" que impõe uma estrutura hierárquica sobre os nós. Um único nó é uma árvore. Este nó é também a raiz da árvore (Lauretto, 2010).

Em uma Árvore de Decisão (AD), um nó folha (ou nó resposta) contém o nome de uma classe ou o símbolo nulo. Um nó interno (ou nó de decisão) que contém o nome de um atributo, para cada possível valor do atributo, corresponde um ramo para uma outra AD

AD é um algoritmo de AM que pode executar tarefas de classificação, regressão e tarefas *multioutput*. É o componente fundamental do algoritmo Floresta Aleatória (FA), que é formado por um agrupamento de Árvores de Decisão.

Na Figura 1, apresenta-se uma visualização de uma AD, com profundidade máxima de 2, desenvolvida a partir do conjunto de dados flor íris. O conjunto de dados consiste em 50 amostras de cada uma das três espécies de íris (íris setosa, íris virginica e íris versicolor), e possui quatro atributos, que são: o comprimento e a largura das sépalas e pétalas, em centímetros.

Figura 1: Árvore de Decisão do Conjunto de Dados Flor Íris.



Fonte: Próprio Autor.

2.2.1 Classificação

Tomando como base a Árvore de Decisão do conjunto de dados flor íris, apresentada na Figura 1, o fluxo de classificação começa no nó raiz (profundidade 0, na parte superior), este nó verifica se o comprimento da pétala (*petal length*) é menor igual a 2,45 cm, caso seja verdade, o fluxo se desloca para o nó filho esquerdo da raiz (profundidade 1, esquerda), esse nó é um nó folha, nele ocorre a classificação da flor em setosa.

No caso em que o comprimento da pétala é maior que 2,45 cm, o fluxo se desloca para o nó filho direito da raiz (profundidade 1, direita), Esse nó é um nó interno, nele verifica-se se a largura da pétala (*petal width*) é menor igual a 1,75 cm, caso seja verdade, o fluxo se desloca para o nó filho esquerdo (profundidade 2, esquerda), classificando a flor em versicolor. Para o caso falso, o fluxo se desloca para o nó filho direito (profundidade 2, direita), classificando a flor em virginica.

O atributo *samples* de um nó representa a quantidade de exemplos de treinamentos o qual este nó se aplica. No nó esquerdo de profundidade 1, 50 exemplos possuem o comprimento da pétala menor igual a 2,45.

O atributo *values* de um nó representa a quantidade de exemplos de treinamento pertencentes a cada classe este nó se aplica. No nó direito de profundidade 1, têm-se um total de 100 exemplos, com comprimento da pétala maior que 2,45, em que 0 pertencem à classe setosa, 50 à classe virginica e

50 à classe versicolor.

O atributo *gini* é coeficiente Gini, representa a impureza de um nó, esse coeficiente é utilizado na fase de treinamento da AD. O coeficiente Gini é calculado pela Equação 1, definida a seguir.

$$G_i = 1 - \sum_{k=1}^{n} {P_{i,k}}^2 \tag{1}$$

Em que:

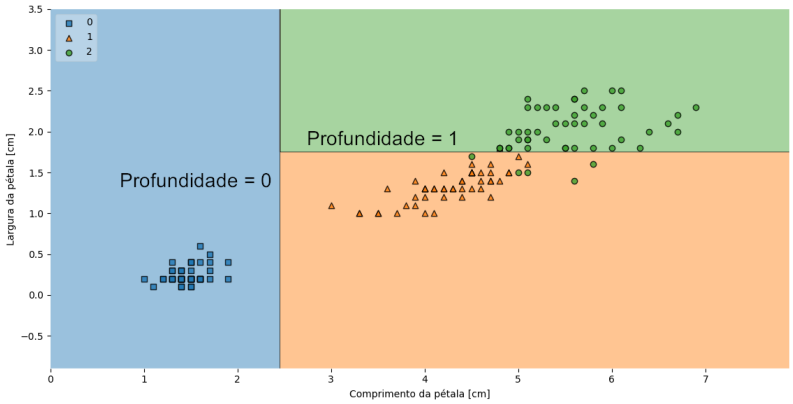
- $P_{i,k}$: é a razão entre a quantidade de exemplos pertencentes a classe k e o total de exemplos no nó i.

Na Árvore de Decisão da flor íris, no nó de profundidade 2 esquerdo, o *gini* é dado por:

$$G_2 = 1 - \left(\frac{0}{54}\right)^2 - \left(\frac{49}{54}\right)^2 - \left(\frac{5}{54}\right)^2$$
$$G_2 = \frac{245}{1458}$$
$$G_2 = 0,168$$

Na Figura 2, apresenta-se as fronteiras de decisão da AD. A linha vertical representa a fronteira de decisão do nó raiz, profundidade 0, e a linha horizontal representa a fronteira de decisão do nó filho direito, profundidade 1.

Figura 2: Fronteiras de decisão da Árvore de Decisão da flor Íris.



Fonte: Próprio Autor.

2.2.2 Probabilidades de Classes

Uma AD possibilita estimar a probabilidade de um exemplo pertencer a uma classe específica k. Com base na Árvore de Decisão da flor íris, uma flor com pétalas de 5 cm e comprimento de 1,5 cm de largura, apresentaria as seguintes probabilidades:

$$P_0 = \frac{0}{54} \rightarrow P_0 = 0$$
$$P_1 = \frac{49}{54} \rightarrow P_1 = 0,907407$$
$$P_2 = \frac{5}{54} \rightarrow P_2 = 0,092592$$

Na Tabela 1, apresenta-se a probabilidade para cada classe.

Tabela 1: Estimativa de Probabilidades da Árvore de Decisão da Flor Íris.

íris setosa	0 %
íris virginica	90,74 %
íris versicolor	9,26 %

Fonte: Próprio Autor.

2.2.3 Algoritmo de Treinamento

O algoritmo de Treinamento Árvore de Classificação e Regressão (CART, em inglês), é um dos algoritmos utilizados no treinamento de uma ÁD, existindo outros, como: ID3, C4.5, CHAID, QUEST, GUIDE, CRUISE, e CTREE. Entretanto, será dado enfoque no algoritmo CART, pois ele é utilizado no *Scikit-Learn*.

O *Scikit-Learn* é uma biblioteca de AM de código aberto para a linguagem de programação Python (Pedregosa et al., 2011). É a biblioteca utilizada na fase de treinamento do modelo utilizado no protótipo proposto.

Para o treinamento, o algoritmo CART inicialmente divide o conjunto de treinamento em dois subconjuntos utilizando uma única característica k e um limiar t_k. A escolha de k e t_k é realizada pela busca de um par (k, t_k) que produz os subconjuntos mais puros (ponderados pelo tamanho). O algoritmo busca minimizar a função de custo dada pela Equação 2, definida a seguir.

$$J(k, t_k) = \frac{m_{esquerda}}{m} G_{esquerda} + \frac{m_{direita}}{m} G_{direita} \tag{2}$$

Em que:

- $G_{esquerda/direita}$: mede a impureza do subconjunto esquerdo/direito;
- $m_{esquerda/direita}$: é o número de exemplos no subconjunto esquerdo/direito.

O algoritmo então continua a divisão dos subconjuntos em sub-subconjuntos, recursivamente. finalizando as divisões ao atingir a profundidade máxima definida ou se não é mais possível encontrar uma divisão que reduza a impureza.

2.2.4 Complexidade Computacional

O algoritmo de treinamento compara todas as características em todas as amostras a cada nó. Resultando em uma complexidade de treinamento $O(n \times m \log(m))$ (Géron, 2019).

Para realizar uma previsão é necessário percorrer a AD da raiz até uma folha, realizando a verificação do valor de uma característica. Em geral, as árvores de decisão são equilibradas, ou seja, as subárvores esquerda e direita têm a mesma altura. Resultando em uma complexidade de previsão $O(\log_2(m))$ (Géron, 2019).

2.2.5 Entropia

Entropia é uma grandeza física da Termodinâmica que representa o grau de desordem de um sistema. Assim, quanto maior for a variação de entropia

de um sistema, maior será sua desordem. No AM, a entropia é frequentemente utilizada como uma medida de impureza de conjunto. Se um conjunto pertence a mesma classe, sua entropia é zero. A entropia de um nó em uma Arvore de Decisão é dada pela Equação 3, definida a seguir.

$$H_i = -\sum_{k=1}^{n} P_{i,k} \log_2(P_{i,k}), P_{i,k} \neq 0 \tag{3}$$

Em que:

- $P_{i,k}$: é a razão entre a quantidade de exemplos pertencentes a classe k e o total de exemplos no nó i.

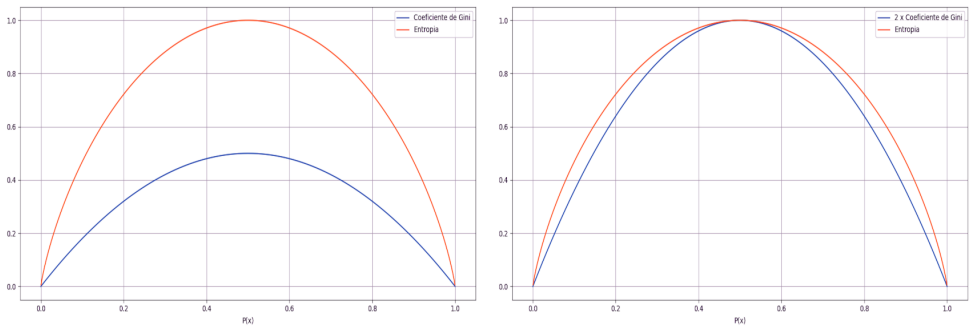
Para a AD da flor íris, a entropia do nó esquerdo de profundidade 2 será:

$$H_2 = -\left(\frac{49}{54}\right) \log_2 \left(\frac{49}{54}\right) - \left(\frac{5}{54}\right) \log_2 \left(\frac{5}{54}\right)$$
$$h_2 = 0,445$$

O coeficiente de Gini apresenta complexidade computacional menor, então geralmente é usado por padrão. A diferença entre os dois coeficientes é que o coeficiente de Gini tende a isolar a classe mais frequente em seu próprio ramo da árvore, enquanto a entropia tende a produzir árvores ligeiramente mais equilibradas.

Na Figura 3, apresenta-se os dois coeficientes graficamente, assim, permitindo notar suas similaridades.

Figura 3: Coeficiente de Gini e Entropia.

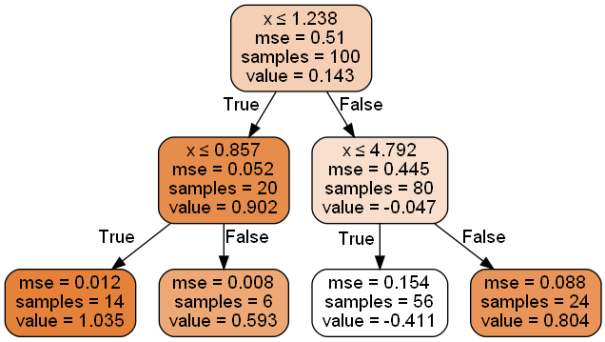


Fonte: Próprio Autor.

2.2.6 Regressão

Uma AD também possibilita realizar tarefas de regressão. Na Figura 4, apresenta-se a visualização de uma AD, com profundidade máxima de 2, construída a partir do conjunto de dados de uma função Cosseno com ruído, de distribuição uniforme em [0, 0,25).

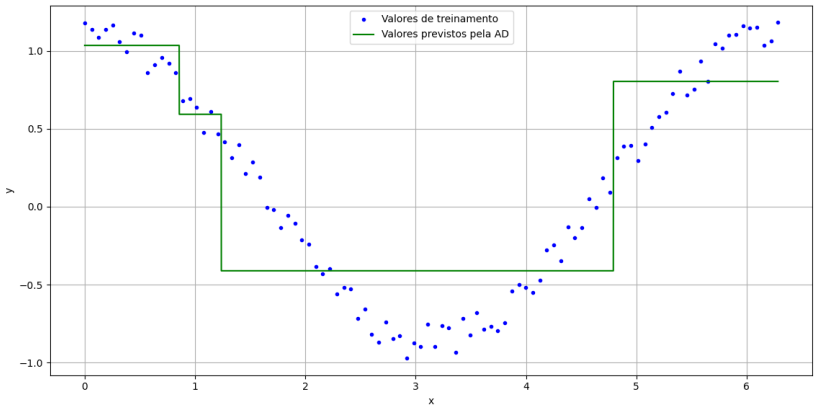
Figura 4: Árvore de Decisão de Regressão para o Conjunto de Dados da Função Cosseno com Ruído.



Fonte: Próprio Autor.

Na figura 5, apresenta-se valores previstos utilizando essa AD.

Figura 5: Valores Previstos com a Árvore de Decisão para o Conjunto de Dados da Função Cosseno com Ruído.



Fonte: Próprio Autor.

Para o treinamento da AD, o algorítimo CART divide o conjunto de dados buscando minimizar o erro quadrático médio (EQM, ou MSE em inglês). A função de custo que o algorítimo busca minimizar é dada pela Equação 4, definida a seguir.

$$J(k, t_k) = \frac{m_{esquerda}}{m} EQM_{esquerda} + \frac{m_{direita}}{m} EQM_{direita} \tag{4}$$

Em que:

$$EQM_{nó} = \sum_{i \in nó} \left(\hat{y}_{nó} - y^{(i)}\right)^2 \tag{5}$$

$$\hat{y}_{nó} = \frac{1}{m_{nó}} \sum_{i \in nó} y^{(i)} \tag{6}$$

2.3 Floresta Aleatória

Uma Floresta Aleatória é um conjunto de Árvores de Decisão, cada qual construída a partir de um subconjunto aleatório do conjunto de treinamento. Esse conjunto de árvores resulta em um preditor agregado, que pode ser usado para a predição da classe de novos objetos através de um sistema de votação (Lauretto, 2010).

Em geral, uma FA é treinada pelo método *bagging* (ou algumas vezes *pasting*).

2.3.1 *Bagging* e *Pasting*

Bootstrap é uma técnica estatística de redimensionamento, que consiste em retirar repetidamente, com reposição, amostras do conjunto de dados, formando diferentes subconjuntos (denominados amostras *bootstrap*).

Dessa forma, é possível treinar vários preditores com diferentes subconjuntos aleatórios do conjunto de treinamento. Quando a amostragem é realizada com reposição, esse método é chamado *bagging* (abreviação para *bootstrap aggregating*). Quando a amostragem é realizada sem reposição, é chamado *pasting*.

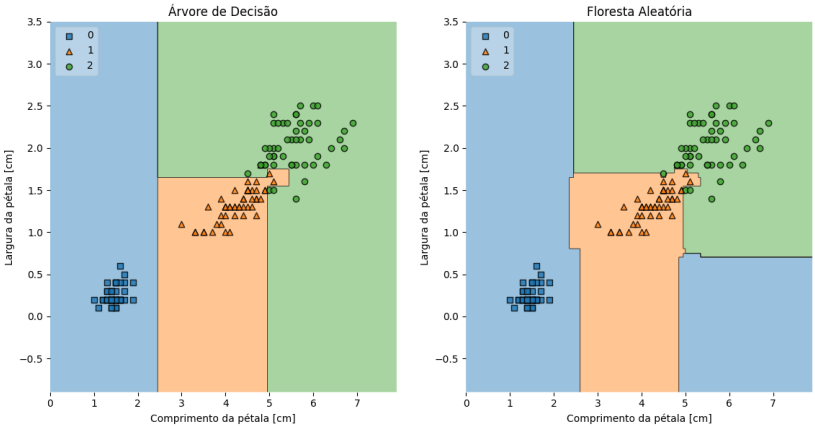
Uma vez que todos os classificadores são treinados, pode-se realizar a classificação para um novo exemplo agregando as estimativas de todos os classificadores. Para a classificação, a estimativa mais frequente é escolhida por um sistema de votação, para regressão, é calculada a média das estimativas.

Por se formada pelo agregado de Árvores de Decisão, uma FA realiza previsões que generalizam bem melhor do que as previsões de uma AD. Dessa forma, evitando sobreajuste (*overfitting*). Sobreajuste é um termo usado em

estatística para descrever quando um modelo estatístico se ajusta muito bem ao conjunto de dados de treinamento, mas se mostra ineficaz para generalizar para novos resultados.

Na Figura 6, apresenta-se as fronteiras de decisão de uma única AD e de uma FA formada pelo agregado de 100 Árvores de Decisão, construídas a partir do conjunto de dados flor íris. A profundidade máxima de cada arvore é de 5.

Figura 6: Árvores de Decisão e Floresta Aleatória, com Profundidade Máxima de 5.



Fonte: Próprio Autor.

2.3.2 Importância da Característica

Uma caraterística importante da FA é a capacidade de medir a importância relativa de cada característica. Para isso, calcula-se quanto cada característica reduz a impureza do modelo. De forma mais precisa, é uma média ponderada em que o peso de cada nó é igual ao número de amostras treinadas que são associadas a ela (Géron, 2019).

Na Tabela 2, apresenta-se a importância relativa de cada característica do conjunto de dados flor íris para uma FA formada pelo agregado de 100 Árvores de Decisão, com profundidade máxima de cada árvore de 5.

Tabela 2: Importância das Características do Conjunto de Dados Flor Íris para a Floresta Aleatória.

Comprimento da Sépala (cm)	11,31 %
Largura da Sépala (cm)	2,4 %
Comprimento da Pétala (cm)	39,31 %
Largura da Pétala (cm)	46,98 %

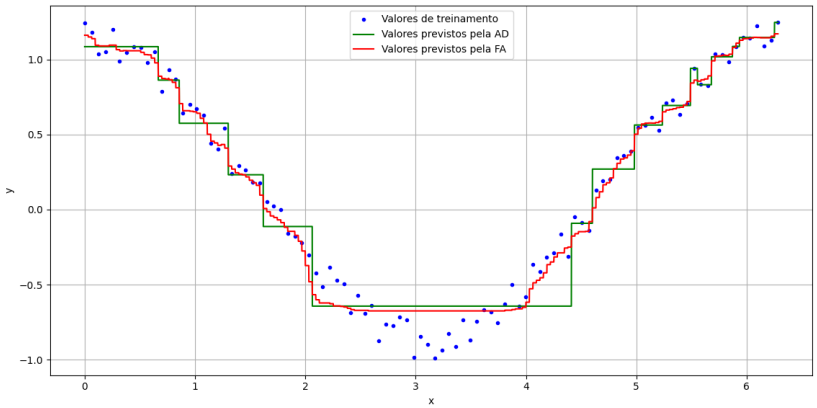
Fonte: Próprio Autor.

Nota-se pela Tabela 2 que as características mais importantes são: o comprimento da pétala e a largura da pétala.

2.3.3 Regressão

Assim como a AD, a FA também possibilita realizar tarefas de regressão. Na Figura 7, apresenta-se valores previstos utilizando uma AD e uma FA, formada pelo agregado de 100 Árvores de Decisão, construídas a partir do conjunto de dados de uma função Cosseno com ruído, de distribuição uniforme em [0, 0,25), com a profundidade máxima de cada arvore de 4.

Figura 7: Regressão com Árvore de Decisão e Floresta Aleatória, para o Conjunto de Dados da Função Cosseno com Ruído.



Fonte: Próprio Autor.

Nota-se, pela Figura 7, que a FA generaliza bem melhor que a AD.

3 Metodologia

Para que a solução *IoT* de digitalização de medidores de consumo analógicos proposta seja possível, algumas etapas são necessárias.

A solução utiliza um modelo de AM aplicada em um módulo ESP32-CAM, que realizará a captura e reconhecimento do dígito correspondente a unidade do consumo apresentada pelo *display* do medidor.

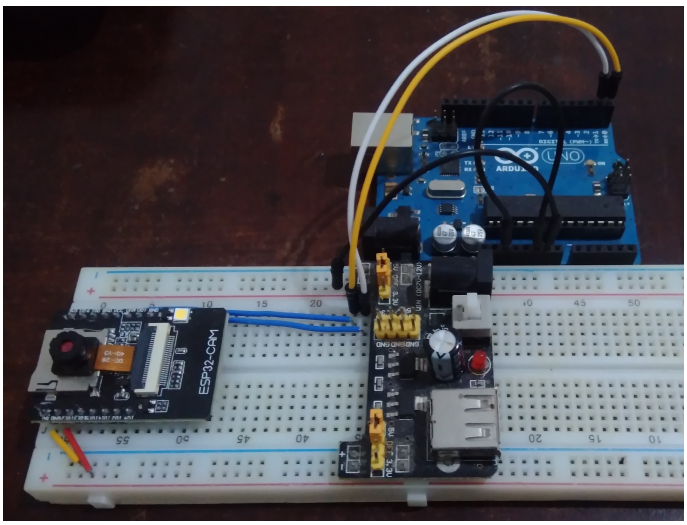
O módulo ESP32-CAM possui uma câmera, modelo OV2640 de 2MP. O módulo é o componente principal da solução proposta. A seguir, apresenta-se suas principais especificações.

- Modelo: ESP 32-CAM;
- Tensão de operação: 5V;
- CPU: Xtensa Dual-Core 32-bit LX6;
- ROM: 448 Kbytes;
- RAM: 520 Kbytes SRAM;
- Flash: 4 MB PSRAM;
- Resolução da foto: 2 Megapixels;
- *Clock* máximo: 240 MHz;
- *Wireless* padrão 802.11 b/g/n;
- Conexão: Wifi 2.4 Ghz (máximo de 150 Mbps);
- Modos de operação: STA/AP/STA+AP;
- Bluetooth: BLE 4.2;
- Portas GPIO: 16;
- Taxa de transferência: 110-460800 bps;
- Dimensões: 40x27x6mm (CxLxE);
- Peso: 7 g;

3.1 Arquitetura

Além do ESP32-CAM, foram utilizados uma fonte de alimentação de 5V e um Arduino UNO para a programação do ESP32-CAM e na comunicação serial. Dessa forma, para solução final o Arduino UNO pode ser removido e a fonte de alimentação pode ser substituída por uma bateria. Na Figura 8, apresenta-se a arquitetura da solução.

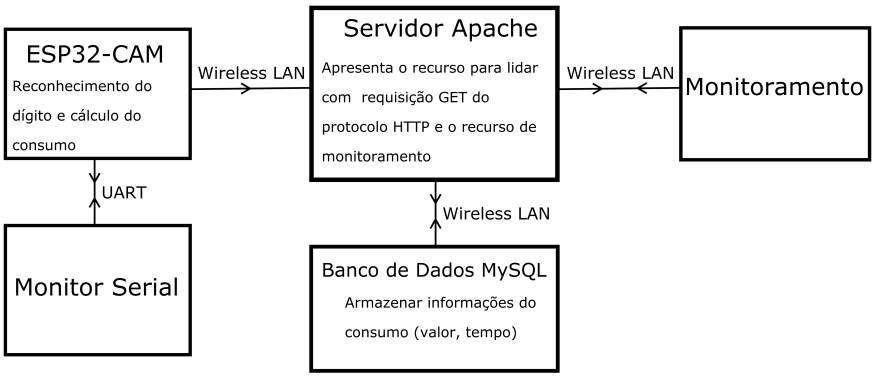
Figura 8: Arquitetura da Solução Proposta.



Fonte: Próprio Autor.

Na Figura 9, apresenta-se um diagrama de bloco da solução.

Figura 9: Diagrama de Bloco da Solução Proposta.



Fonte: Próprio Autor.

3.2 Processamento da Imagem

O processamento da imagem é uma etapa importante, pois facilita o treinamento do modelo, permitindo melhorar a sua interpretação e realçar características, além disso, reduz a quantidade de dados.

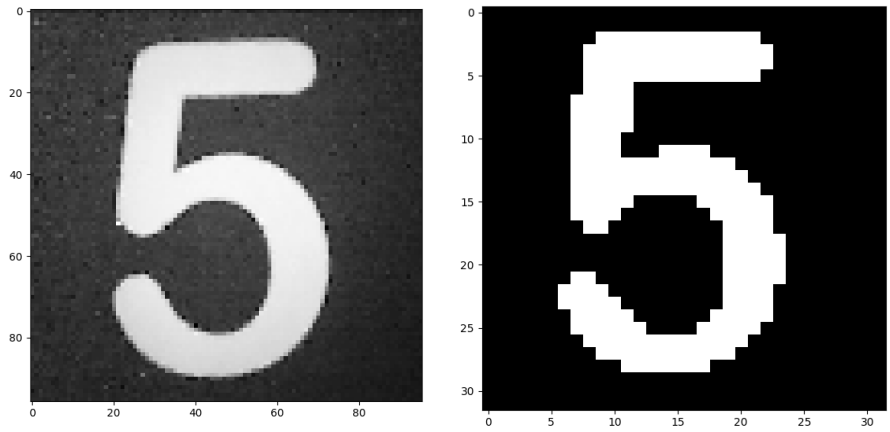
Para a aquisição das imagens, o ESP32-CAM foi configurado para realizar a captura em tons cinza e com um tamanho de imagem de 96 x 96 *pixels* (menor tamanho configurável disponível).

Os valores dos *pixels* (menor elemento da imagem) são salvos em um *buffer* de 8 bits de tamanho 9216. Após a aquisição da imagem, realizou-se a compressão para um tamanho de 32 x 32 *pixels*, salvando os valores em um vetor unidimensional. Além disso, realizou-se a conversão da imagem de tons cinza para o preto e branco, aplicando um limiar.

Na Figura 10a, apresenta-se a imagem sem processamento, na Figura 10b, apresenta-se a imagem com processamento.

Figura 10: Capturas com o ESP32-CAM.

(a) Imagem Sem Processamento. (b) Imagem Com Processamento.



Fonte: Próprio Autor.

As funções que realizam essas atividades foram desenvolvidas utilizando a Arduino *Integrated Development Environment* (IDE). Na Figura 11, apresenta-se a função que realiza a captura de imagem, realiza o processo de compressão e conversão em preto e branco, retornando *true* (verdade) ao finalizar todo o processo.

Figura 11: Função de Processamento de Imagem.

```
bool capture_still()
{
  camera_fb_t *frame = esp_camera_fb_get();

  if (!frame) return false;

  // reseta o array da imagem
  for (uint16_t i = 0; i < H * W; i++) features[i] = 0;

  // redução da imagem (Downsampling)
  for (uint16_t i = 0; i < frame->len; i++)
  {
    const uint16_t x = i % WIDTH;
    const uint16_t y = floor(i / WIDTH);              // floor() arredonda para baixo no inteiro mais próximo
    const uint8_t block_x = floor(x / BLOCK_SIZE);
    const uint8_t block_y = floor(y / BLOCK_SIZE);
    const uint16_t j = block_y * W + block_x;

    features[j] += frame->buf[i];
  }

  // Aplica limiar
  for (uint16_t i = 0; i < H * W; i++) features[i] = (features[i] / (BLOCK_SIZE * BLOCK_SIZE) < THRESHOLD) ? 1 : 0;

  return true;
}
```

Fonte: Próprio Autor.

3.3 Criação do Conjunto de Dados

Com os dados de entrada no formato desejado, passou-se para a próxima etapa, que consiste na criação do modelo de classificação. Para criação de um modelo de classificação é necessário um conjunto de dados de treinamento, o módulo ESP32-CAM foi utilizado para a criação do conjunto de dados.

Devido as dificuldades em encontrar dígitos variados, o conjunto de dados de treinamento foi composto de 3 imagens de cada dígito e o conjunto de dados de teste com 1 imagem de cada dígito. Em geral, os medidores de consumo possuem dígitos bastante semelhantes, dessa forma, foi possível obter uma boa precisão para o modelo.

O procedimento para criação do conjunto foi da seguinte forma: foram impressos dígitos de 0 a 9, realizaram-se a captura e processamento desses dígitos, os valores dos *pixels* foram impressos no monitor serial da Arduino IDE e, por fim, foram copiados e salvos em um arquivo formato CSV. Na Figura 12, apresenta-se a função desenvolvida na Arduino IDE que realiza a impressão dos valores no monitor serial.

Figura 12: Função para Imprimir Valores no Monitor Serial.

```
115 void print_features()
116 {
117   for (uint16_t i = 0; i < H * W; i++)
118   {
119     Serial.print(features[i]);
120
121     if (i != H * W - 1)
122     Serial.print(',');
123   }
124
125   Serial.println();
126 }
```

Fonte: Próprio Autor.

Dessa forma, foi possível obter uma arquivo formato CSV para cada classe com os dados para o treinamento e teste do modelo, com o nome de cada arquivo correspondendo à sua classe.

3.4 Criação do Modelo

Com a criação do conjunto de dados, passou-se para etapa de criação do modelo. Para isso, utilizou-se o Visual Studio Code no desenvolvimento das funções na linguagem de programação Python, e a biblioteca *scikit-learn* para criação e teste do modelo de uma FA de classificação, com o objetivo de realizar o reconhecimento dos dígitos. Na Figura 13, apresenta-se as funções utilizadas.

Figura 13: Funções para Criação da Floresta Aleatória de Classificação de Dígitos.

```
1  from sklearn.ensemble import RandomForestClassifier
2  from micromlgen import port
3  import numpy as np
4  from glob import glob
5  from os.path import basename
6
7  def load_features(folder):
8      dataset = None
9      classmap = {}
10     for class_idx, filename in enumerate(glob('%s/*.csv' % folder)):
11         class_name = basename(filename)[:-4]
12         classmap[class_idx] = class_name
13         samples = np.loadtxt(filename, dtype=float, delimiter=',')
14         labels = np.ones((len(samples), 1)) * class_idx
15         samples = np.hstack((samples, labels))
16         dataset = samples if dataset is None else np.vstack((dataset, samples))
17
18     return dataset, classmap
19
20 def get_classifier(features):
21     X, y = features[:, :-1], features[:, -1]
22
23     return RandomForestClassifier(n_estimators=50, max_depth=100).fit(X, y)
24
25 if __name__ == '__main__':
26     features, classmap = load_features('./dataset/')
27     classifier = get_classifier(features)
28     c_code = port(classifier, classmap=classmap)
29     file = open("model.h","w")
30     file.write(c_code)
31     file.close()
```

Fonte: Próprio Autor.

Na Figura 13, a função *load_features()* carrega e prepara o conjunto de dados para o treinamento. A função *get_classifier()* cria a FA, formada pelo agregado de 50 Árvores de Decisão, em que cada AD possui profundidade máxima de 100. Por fim, a função *port()* do pacote de MicroML *micromlgen* converte e otimiza a FA para a linguagem de programação C. O modelo em C da FA foi salvo em um arquivo H para utilização no ESP32-CAM.

Utilizando o conjunto de dados de teste, foi possível averiguar a precisão do modelo para novos dados. Na Figura 14, apresenta-se a função utilizada para o teste de precisão e o resultado obtido de 90%.

Figura 14: Funções para o Teste de Precisão do Modelo.

Fonte: Próprio Autor.

3.5 Aplicação do Modelo

O modelo criado possui uma função *predict()* que recebe os valores dos *pixels* da imagem em um vetor unidimensional e retorna uma classe estimada para os valores.

Com a aplicação do modelo, foi possível realizar o reconhecimento dos dígitos, dessa forma, seria possível estimar o valor do consumo apresentada por um *display* de um medidor. Na Figura 15, apresenta-se a função desenvolvida na Arduino IDE que aplica o modelo para estimar o valor da unidade e determina o novo valor do consumo.

Figura 15: Função de Determinação do Consumo a Partir do Valor de Unidade Estimado pelo Modelo.

```
void get_value()
{
  unit_value = predict(features);  // Reconhecimneto do dígito da unidade

  // Incrementa a dezena do consumo e calcula o novo valor
  if(value%10 == 9 && unit_value == 0) value = (value/10)*10 + unit_value + 10;

   // Decrementa a dezena do consumo e calcula o novo valor
  else if(value%10 == 0 && unit_value == 9) value = (value/10)*10 + unit_value - 10;

  // Calcula o novo valor do consumo
  else value = (value/10)*10 + unit_value;
}
```

Fonte: Próprio Autor.

3.6 Bancos de Dados

Com o consumo determinado, seu valor foi enviado para um servidor local de banco de dados MySQL, por meio do método de requisição GET do protocolo HTTP. Utilizou-se o programa XAMPP para criação dos servidores de banco de dados MySQL e Apache localmente em um computador pessoal.

Na Figura 16, apresenta-se a função desenvolvida na Arduino IDE que realiza requisição GET.

Figura 16: Função de Requisição GET.

```
void send_value()
{
  HTTPClient http;

  http.begin(HOST_NAME + PATH_NAME + queryString + value); //HTTP
  int httpCode = http.GET();

  // httpCode will be negative on error
  if(httpCode > 0)
  {
    // file found at server
    if(httpCode == HTTP_CODE_OK)
    {
      String payload = http.getString();
      Serial.println(payload);
    }

    // HTTP header has been send and Server response header has been handled
    else Serial.printf("[HTTP] GET... code: %d\n", httpCode);
  }

  else Serial.printf("[HTTP] GET... failed, error: %s\n", http.errorToString(httpCode).c_str());

  http.end();
}
```

Fonte: Próprio Autor.

Em que:

- HOST_NAME é a *string* (cadeia de caracteres) do endereço de IP do servidor Apache.

- PATH_NAME é a *string* do recurso utilizado do servidor.
- queryString é a *string* para requisição GET do consumo.

O código em linguagem de programação PHP *insert_value.php* é o recurso do servidor local que lida com a requisição GET e insere o novo valor de consumo no servidor de banco de dados local. Para o desenvolvimento do código foi utilizado o Visual Studio Code, na Figura 17, apresenta-se seu conteúdo.

Figura 17: Código que Lida com a Requisição GET.

```php
<?php

include_once "config.php";

if(isset($_GET["value"])) {
   $value = $_GET["value"];

   // Create connection
   $conn = new mysqli($servername, $username, $password, $dbname);
   // Check connection
   if ($conn->connect_error) {
      die("Connection failed: " . $conn->connect_error);
   }

   $sql = "INSERT INTO tbl_value (value) VALUES ($value)";

   if ($conn->query($sql) === TRUE) {
      echo "New record created successfully";
   } else {
      echo "Error: " . $sql . " => " . $conn->error;
   }

   $conn->close();
} else {
   echo "value is not set";
}
?>
```

Fonte: Próprio Autor.

3.7 Monitoramento do Consumo

Com os valores de consumo enviados para o banco de dados, foi possível realizar o monitoramento do consumo acessando uma aplicação *Web* do servidor local. Essa aplicação *Web*, foi desenvolvida no Visual Studio Code e realiza a leitura dos 10 últimos valores de consumo, informa o último valor e gera um gráfico de consumo.

O monitoramento do consumo corresponde a etapa final da solução *IoT* de digitalização de medidores de consumo analógicos. Na Figura 18, apresenta-se os principais trechos de código da aplicação *Web*.

Figura 18: Trechos de Código da Aplicação *Web* de Monitoramento do Consumo.

```php
$sql = "SELECT * FROM (SELECT * FROM tbl_value ORDER BY id DESC LIMIT 10)sub ORDER BY id ASC";

$result = $conn->query($sql);
$dataset = array();
$id = 0;
$value = 0;
$date = 0;

if ($result->num_rows > 0) {
    while($row = $result->fetch_assoc()) {
        $id = $row["id"];
        $value = $row["value"];
        $date = $row["date"];
        $time = strtotime($date);
        $dataset[] = array($time, intval($row["value"]));
    }
} else {
    echo "0 results";
}

echo "<table>
    <tr>
        <th> ID </th>
        <th> Consumo </th>
        <th> Data </th>
    </tr>
    <tr>
        <td>" .$id. "</td>
        <td>" .$value. "</td>
        <td>" .$date. "</td>
    </tr>
</table>";
```

```php
<h3><b>Gráfico de Consumo<b></h2>
<div id="placeholder" style="width:1000px;height:200px"></div>
<script>
    var dataset = <?php echo json_encode($dataset); ?>;
    var options = {
        colors: ['#01DFD7','#01DFD7'],
        legend: {
            show: true,
            margin: 10,
            backgroundOpacity: 1
        },
        grid: {
            hoverable: true,
            borderWidth: 1,
            borderColor: '#000000',
            labelMargin: 15,
            backgroundColor: '#FFFFFF'
        },
        points: {
            show: true
        },
        lines: {
            show: true,
            lineWidth: 3,
        }
    };
```

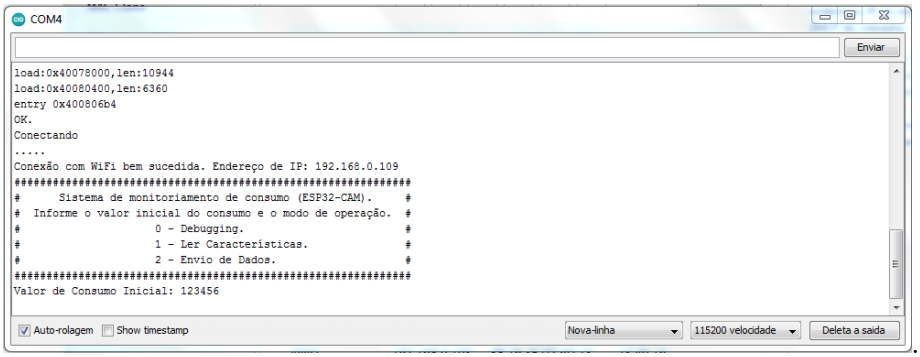
Fonte: Próprio Autor.

4 Resultados

A solução *IoT* proposta, calcula o valor do consumo a partir da estimativa do valor da unidade. Dessa forma, o valor inicial do consumo precisa de informado.

O valor inicial do consumo é informado via comunicação serial no monitor serial da Arduino IDE pelo operador da solução. Assim, foi criado um menu informativo para o operador com algumas informações de uso. Na Figura 19, apresenta-se esse menu de operação.

Figura 19: Menu do Sistema de Monitoramento de Consumo.



Fonte: Próprio Autor.

4.1 Reconhecimento dos Dígitos

Com a opção de *Debugging* ativa, é possível visualizar os valores do vetor de armazenamento da imagem capturada e processada pelo ESP32-CAM no monitor serial da Arduino IDE, dessa forma, permitindo ajustar o enquadramento da câmera e verificar se as estimativas estão corretas.

Na Figura 20, apresenta-se os resultados obtidos na classificação de dígitos pela FA no ESP32-CAM. Nota-se que as estimativas estão corretas, de acordo com a precisão obtida.

Figura 20: Reconhecimento dos Dígitos pelo ESP32-CAM.

Fonte: Próprio Autor.

Na utilzação do modelo da FA na estimativa dos dígitos, o ESP32-CAM leva, em torno, de 160 ms. O código completo da solução usa 1030354 bytes (32%) de espaço de armazenamento para programas, o máximo são 3145728 bytes. As variáveis globais usam 52076 bytes (15%) de memória dinâmica, deixando 275604 bytes para variáveis locais, o máximo são 327680 bytes.

4.2 Importância de Cada *Pixel*

No processo de classificação, a importância relativa de cada *pixel* pode ser determinada utilizando os valores de impureza do modelo. Na criação do modelo é atribuído um valor de importância para cada *pixel*, dessa forma, é possível acessar esses valores e apresenta-los no formato 32 x 32 *pixels*. Na Figura 21, apresenta-se o código utilizado na geração do gráfico da importância relativa.

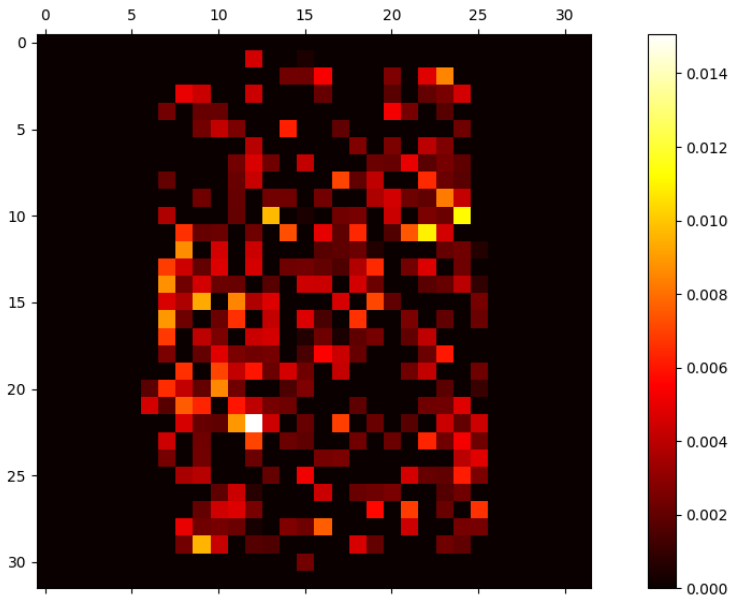
Figura 21: Código para Geração da Importância de Cada *Pixel*.

```
30    importance = classifier.feature_importances_
31    image_array = np.asfarray(importance).reshape((32,32))
32    plt.matshow(image_array, cmap=plt.cm.hot)
33    plt.colorbar()
34    plt.show()
```

Fonte: Próprio Autor.

Na Figura 22, apresenta-se a importância relativa de cada *pixel* no modelo da FA para a classificação dos dígitos.

Figura 22: Importância de Cada *Pixel* Utilizando Valores de Impureza.

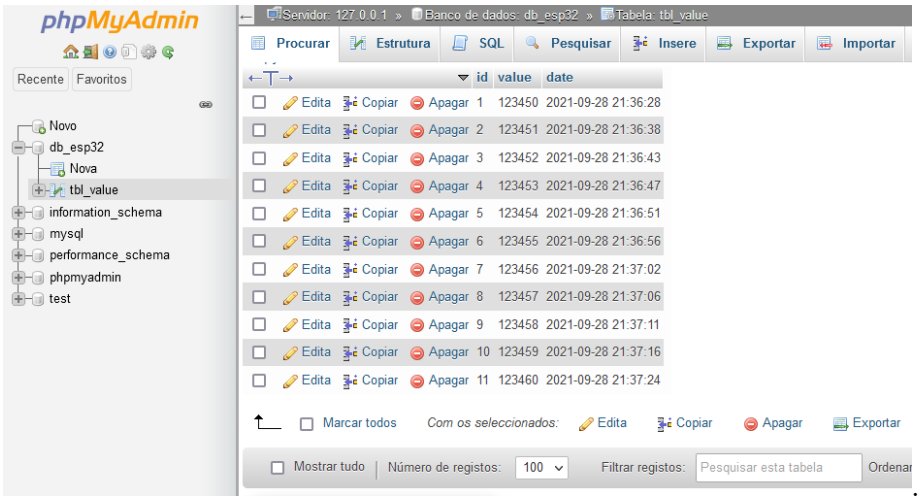


Fonte: Próprio Autor.

4.3 Envio dos Dados

Com a opção de *Envio de Dados* ativa, o valor do consumo é enviado para o servidor de banco de dados, por meio do método de requisição GET do protocolo HTTP. Na Figura 23, apresenta-se o banco de dados do consumo pelo aplicativo *Web* phpMyAdmin no servidor local.

Figura 23: Banco de Dados do Consumo.

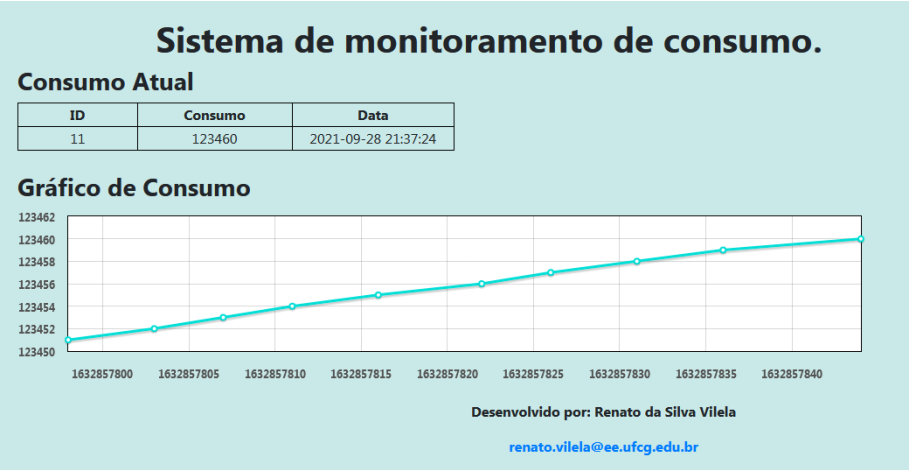


Fonte: Próprio Autor.

4.4 Sistema de Monitoramento

Com os valores de consumo determinados e enviados para o banco de dados, é possível realizar o monitoramento em tempo real do consumo acessando a aplicação *Web* do servidor local. Na Figura 24, apresenta-se a aplicação de monitoramento do consumo.

Figura 24: Sistema de Monitoramento de Consumo.



Fonte: Próprio Autor.

O gráfico de consumo permite a extração de tendências de consumo, possibilitando uma futura otimização. Além disso, é possível, por exemplo: detectar vazamentos, detectar consumo atípico. Outra vantagem é a possibilitada de realizar o monitoramento de forma remota em uma taxa muito maior.

5 Conclusões

A partir do desenvolvimento desse trabalho, foi possível aplicar múltiplos conhecimentos adquiridos ao longo do curso, além disso, confrontou-se com a tarefa da escolha de uma arquitetura para solução de determinado problema. Dessa forma, foi possível desenvolver uma solução *IoT* de digitalização de medidores de consumo analógicos. Além disso, foi possível aplicar o conceito de Computação de Borda (*Edge Computing*), que está em foco no contexto atual de sistemas *IoT*.

Não foi possível a realização de testes da solução em medidores reais. Uma das dificuldades atuais é a necessidade de uma lente de aumento para câmera do ESP32-CAM. Assim, os testes foram realizados com dígitos impressos com tamanho maior que os dígitos reais. Com os testes realizados foi possível comprovar que o sistema é viável e suscetível de implementação. Outra dificuldade, é obtenção de mais dados para aprimoramento do modelo.

Para trabalhos futuros, sugere-se utilizar mais dados para o treinamento e teste, o desenvolvimento de uma lente para câmera do ESP32-CAM, permitindo foco em objetos próximos, o desenvolvimento de uma PCI (placa de circuito impresso), e a possibilidade da aplicação do TensorFlow, que no momento está em fase experimental para o ESP32, permitindo, assim, a utilização de Redes Neurais Artificiais para a tarefa de inferência.

Referências

K. FACELI, A. C. LORENA, J. GAMA, and A. C. P. L. F. d. CARVALHO. *Inteligência artificial: uma abordagem de aprendizado de máquina*. LTC, Rio de Janeiro, 2 edition, 2011.

X. FENG, T. Y. LAURENCE, and W. LIZHE. Internet of things. *International Journal of Communication Systems*, 25(9):1101–1102, 2012.

A. Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, 2019.

M. S. Lauretto. Árvores de decisão, 2010.

T. M. MITCHELL. *Machine Learning*. McGraw-Hill Science, New York, 1 edition, 1997.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

F. SAKR, F. BELLOTTI, R. BERTA, and A. DE GLORIA. Machine learning on mainstream microcontrollers. *Sensors*, 20(9), 2020. ISSN 1424-8220. URL `https://www.mdpi.com/1424-8220/20/9/2638`.