UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Classificação para o Monitoramento Não-Intrusivo de Cargas em Sistema Embarcado com Rede Neural Convolucional

Dissertação de Mestrado

Hugo Menezes Tavares

São Cristóvão – Sergipe

2021

UNIVERSIDADE FEDERAL DE SERGIPE

CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Hugo Menezes Tavares

Classificação para o Monitoramento Não-Intrusivo de Cargas em Sistema Embarcado com Rede Neural Convolucional

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Sergipe como requisito parcial para a obtenção do título de mestre em Ciência da Computação.

Orientador: Bruno Otávio Piedade Prado
Coorientador: Kalil Araujo Bispo

São Cristóvão – Sergipe

2021

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL
UNIVERSIDADE FEDERAL DE SERGIPE

Tavares, Hugo Menezes
T231c Classificação para o monitoramento não-intrusivo de cargas em sistema embarcado com rede neural convolucional / Hugo Menezes Tavares ; orientador Bruno Otávio Piedade Prado. - São Cristóvão, 2021.
148 f. : il.

Dissertação (mestrado em Ciência da Computação) – Universidade Federal de Sergipe, 2021.

1. Computação. 2. Inteligência artificial. 3. Redes neurais (computação). 4. Sistemas embarcados (Computadores). I. Prado, Bruno Otávio Piedade orient. II. Título.

CDU 004

UNIVERSIDADE FEDERAL DE SERGIPE
PRÓ-REITORIA DE PÓS-GRADUAÇÃO E PESQUISA
COORDENAÇÃO DE PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Ata da Sessão Solene de Defesa da Dissertação do Curso de Mestrado em Ciência da Computação-UFS.
Candidato: HUGO MENEZES TAVARES

Em 25 dias do mês de novembro do ano de dois mil e vinte, com início às 10h00min, realizou-se na Sala virtual https://meet.google.com/you-rfrd-pxe. A Sessão Pública de Defesa de Dissertação de Mestrado do candidato **HUGO MENEZES TAVARES**, que desenvolveu o trabalho intitulado: **"Classificação para o Monitoramento Não-Intrusivo de Cargas em Sistema Embarcado com Rede Neural Convolucional"**, sob a orientação do Prof. Dr. **Bruno Otávio Piedade Prado**. A Sessão foi presidida pelo Prof. Dr. **Bruno Otávio Piedade Prado** (PROCC/UFS), que após a apresentação da dissertação passou a palavra aos outros membros da Banca Examinadora, Prof. Dr. **Edward David Moreno Ordonez** (PROCC/UFS), Prof. Dr. **Kalil Araujo Bispo** (PROCC/UFS), **Prof. Dr. Leonardo Nogueira Matos** (PROCC/UFS) e, em seguida, ao Prof. Dr. **Paulo Salgado Gomes de Mattos Neto** (UFPE). Após as discussões, a Banca Examinadora reuniu-se e considerou o mestrando (a) APROVADO "*(aprovado/reprovado)*". Atendidas as exigências da Instrução Normava 01/2017/PROCC, do Regimento Interno do PROCC (Resolução 67/2014/CONEPE), Resolução nº 25/2014/CONEPE e da Portaria nº 413 de 27 de maio de 2020 (Banca por videoconferência) que regulamentam a Apresentação e Defesa de Dissertação, e nada mais havendo a tratar, a Banca Examinadora elaborou esta Ata que será assinada pelos seus membros e pelo mestrando.

Cidade Universitária "*Prof. José Aloísio de Campos*", 25 de novembro de 2020.

Prof. Dr. Bruno Otávio Piedade Prado
(PROCC/UFS)
Presidente

Prof. Dr. Kalil Araujo Bispo
(PROCC/UFS)
Coorientador / Examinador Interno

Prof. Dr. Edward David Moreno Ordonez
(PROCC/UFS)
Examinador Interno

Prof. Dr. Leonardo Nogueira Matos
(PROCC/UFS)
Examinador Interno

Prof. Dr. Paulo Salgado Gomes de Mattos Neto
(UFPE)
Examinador Externo

Hugo Menezes Tavares
discente

Aos meus pais Luciene e Reginaldo, ao meu irmão Vicente, à minha noiva Mayara, aos professores idealizadores desse trabalho: Bruno Prado, Kalil Bispo e Leonardo Matos.

Agradecimentos

Agradeço à minha mãe Luciene, que me deu a vida, amor, carinho e sempre me apoiou com muita paciência. Ao meu pai Reginaldo e ao meu irmão Vicente, por sempre aguentar, pacientemente, meu estresse descontado injustamente. Devo aos meu pais tudo que sou, tudo que conquistei e tudo que poderei um dia conquistar.

Agradeço à minha noiva Mayara, por ser uma pessoa tão especial, a qual desejo conviver e amar até o fim da vida. Obrigado por todo amor, carinho, cuidado e compreensão. Ao meu tio João Vicente, aos primos João Victor e Rafael e à família em geral, que carrega por gerações as virtudes do trabalho e da honestidade, e da qual sinto muito orgulho em fazer parte. Aos meus sogros Maria Lúcia e Valdir Ferreira e à minha cunhada Lívia. Agradeço também, com um carinho especial, à todos os colaboradores da Casa São Vicente, que são minha segunda família.

Agradeço aos colegas bolsistas que tive a honra de conhecer durante a pós-graduação: Ademir Almeida, Alana Lúcia, Alef Menezes, Bruno Nunes, Cícero Gonçalves, Felipe Faustino, Mislene Nunes e Thauanne Moura. Aos velhos amigos de graduação e que também foram colegas de pós-graduação: Elias Rabêlo, Fabiano Andrade e em especial a Felipe Florêncio, que foi um grande incentivador do meu seguimento na carreira acadêmica. Agradeço a todos da secretaria do Programa de Pós-Graduação em Ciência da Computação (PROCC), em especial a Elaine Silva, a qual sou muito grato pelos conselhos, por sempre buscar as informações e facilitar a vida de nós mestrandos.

Agradeço aos professores, meus heróis, que embora injustamente não tenham o devido reconhecimento por parte da sociedade e dos governantes, lutam pela educação, pelo desenvolvimento, aprendizado de seus alunos e por um futuro melhor para o país. Agradeço a todos os professores, desde a que pegou na minha mão e me ensinou a escrever, assim como aos da pós-graduação, especialmente aos doutores Bruno Otávio, Kalil Araujo e Leonardo Matos, pela paciência, ajuda e por todos os ensinamentos repassados. Agradeço também aos doutores Paulo Salgado e Edward Moreno por terem aceitado os convites de avaliadores externo e interno, respectivamente, desta dissertação.

Agradeço o incentivo financeiro dado pela Fundação de Apoio à Pesquisa e à Inovação Tecnológica do Estado de Sergipe (FAPITEC) e agradeço também aos servidores, terceirizados e a todos que compõem a nossa majestosa e queridíssima Universidade Federal de Sergipe (UFS), que mesmo em tempos tenebrosos e adversos pelos quais a cultura e a educação brasileira vêm passando, agem nos bastidores pela melhoria do ensino, cientes da importância deste para o crescimento econômico, cultural e para a promoção de uma sociedade mais justa e igualitária.

“Quando tudo tiver parecendo ir contra você,
lembre-se que o avião decola contra o vento,
e não a favor dele."
(Henry Ford)

Resumo

A energia elétrica é de grande importância para o desenvolvimento econômico dos países e o seu consumo vem crescendo em um ritmo vertiginoso, mais rápido que os demais modais energéticos. Concomitantemente com o aumento do consumo, surge também a preocupação com o meio ambiente e a sustentabilidade, sendo que assegurar o acesso à energia elétrica de forma confiável, sustentável, moderna e a preço acessível para todos é um dos objetivos da Agenda 2030 proposta pela Organização das Nações Unidas (ONU). Além do incentivo ao uso de energias renováveis e de menor impacto ambiental, há também duas preocupações: criar dispositivos energeticamente cada vez mais eficientes e reduzir o desperdício de energia elétrica, buscando alternativas para um uso mais eficiente desta. O envolvimento ativo dos consumidores resulta, na maioria das vezes, em um uso mais eficiente da energia elétrica, aumentando o interesse no desenvolvimento de tecnologias que os conscientizem quanto aos seus hábitos. Estudos mostram que quanto maior o detalhamento de informações acerca do consumo elétrico, maior a quantidade de energia elétrica economizada pelos consumidores. Uma das técnicas mais utilizadas para esse detalhamento é o Monitoramento Não-Intrusivo de Cargas, que através da desagregação de cargas, faz a distinção entre as cargas elétricas e explora o consumo elétrico de cada uma delas individualmente. A fim de contribuir para essa técnica, e diante do crescente avanço nas áreas de eletrônica e aprendizado de máquina, este estudo propõe realizar a classificação de cargas em sistema embarcado utilizando um método de aprendizado profundo e, dessa forma, contribuir para um consumo mais eficiente de energia elétrica. Baseado na literatura e em experimentos realizados, adotamos como característica de distinção a imagem binária da trajetória tensão-corrente, que foi a que obteve melhores resultados. Para realizar a classificação dos aparelhos, utilizamos essas imagens como entrada para um método de aprendizado profundo, que foi a Rede Neural Convolucional (RNC), escolhido após obter melhores resultados nos testes que foram realizados. A contribuição deste trabalho é a quantização do modelo da RNC usando o *TensorFlow Lite* e a sua aplicação em um dispositivo embarcado, que foi o ESP32. Usando o método de validação cruzada *leave-one-out*, nosso modelo da RNC foi avaliado usando o *dataset* PLAID e obteve uma média macro F-Score de 74,76% para PLAID1, 56,48% para PLAID2 e 73,97% para PLAID1+2, resultados bastante próximos ao da literatura. Em testes realizados com todos os dados do PLAID1+2, a acurácia foi de 98,55% no ESP32, demostrando que o dispositivo embarcado pode ser utilizado para realizar a classificação de cargas com alta taxa de acurácia.

Palavras-chave: Monitoramento Não-Intrusivo de Cargas, *Non-Intrusive Load Monitoring* (NILM), Classificação de Cargas, Trajetória VI, Rede Neural Convolucional (RNC), Sistemas Embarcados, ESP32.

Abstract

Electric power is very important for the economic development of all countries, and its consumption has been growing at an impressive rate, doing so faster than other types of power. Along with the increase in consumption, there are also concerns about environmental sustainability; after all, ensuring access to electric power in a reliable, sustainable, modern, and affordable way for all is one of the objectives of the Sustainable Development Goals, an agenda proposed by the United Nations (UN). In addition to encouraging the usage of renewable and less environmentally impactful energy, there are also concerns to create increasingly more power efficient devices, and to reduce the waste of electric power by seeking alternatives for a more efficient use of it. The active involvement of consumers results, for the most part, in a more efficient use of electric power, which increases interest in the development of technologies that make them aware of their habits. Studies show that, the greater the detail of information about electrical power consumption, the greater the amount of electric power saved by consumers. One of the most used techniques to analyze such details is Non-Intrusive Load Monitoring (NILM), who, by disaggregating the loads, distinguishes between each of the appliances and explores the electric power consumption of each one individually. Therefore, in order to contribute to this technique, and considering the ever-growing progress in the electronic and machine-learning areas, this study proposes a set of training strategies using a deep-learning method for load classification in an embedded system, and therefore, contribute to a more efficient use of electric power. Based on literature and experiments, we adopted the binary image of the voltage-current as the distinguishing feature, as it obtained the best results. In order to classify the devices, we used said images as input to the Convolutional Neural Network (CNN), which was chosen after obtaining the best results in the tests that were performed. After we used the leave-one-out cross-validation method, our CNN model was evaluated using the PLAID dataset and obtained an F-Score macro-average of 74.76% for PLAID1, 56.48% for PLAID2, and 73.97% for PLAID1+2, and those results were very close to literature. The novelty of this study is the quantization of the CNN model using TensorFlow Lite, and its application in a resource-constrained embedded system (ESP32). The accuracy rate achieved in testes performed with all data from the PLAID1+2 dataset was 98.55%, which shows that the embedded device can be used to perform the load classification with a high accuracy rate.

Keywords: Non-Intrusive Load Monitoring (NILM), Load Classification, VI Trajectory, Convolutional Neural Network, CNN, Embedded Systems, ESP32.

Lista de ilustrações

Figura 1 – Gráfico da progressão do consumo mundial de energia elétrica 23
Figura 2 – Gráfico do consumo cativo de energia elétrica por classe 23
Figura 3 – Média de energia economizada por tipo de *feedback* 25
Figura 4 – Triângulo de potências 32
Figura 5 – Exemplo de abordagem NILM 35
Figura 6 – Exemplo de abordagem ILM 35
Figura 7 – Etapas do processo NILM 36
Figura 8 – Classificações dos dispositivos NILM 37
Figura 9 – Representação do neurônio artificial 39
Figura 10 – Redes *Perceptron* de multicamadas 44
Figura 11 – Exemplo de topologia da Rede Neural Convolucional 45
Figura 12 – Exemplo de entrada tridimensional 46
Figura 13 – Exemplo de operação de convolução usando dados literais 47
Figura 14 – Exemplo de operação de convolução usando números 47
Figura 15 – Exemplo de operação de *pooling* 49
Figura 16 – Exemplo de camada densa 49
Figura 17 – Matriz de confusão literal para classificação binária 50
Figura 18 – Gráfico de instâncias por tipo de dispositivo (PLAID1+2) 52
Figura 19 – Logotipo do *TensorFlow* 53
Figura 20 – Arquitetura do TensorFlow 2.0 53
Figura 21 – Módulo ESP-WROOM32 com SoC ESP32 55
Figura 22 – Gráfico dos *hardwares* mais utilizados 67
Figura 23 – Gráfico das grandezas mais mensuradas 68
Figura 24 – Gráfico das características mais utilizadas 69
Figura 25 – Gráfico das técnicas de aprendizado mais utilizadas 70
Figura 26 – Gráfico das métricas mais utilizadas 71
Figura 27 – Forma de onda da corrente de quatro diferentes dispositivos elétricos 73
Figura 28 – Plano P-Q de quatro diferentes dispositivos elétricos 73
Figura 29 – Harmônicas de quatro diferentes dispositivos elétricos 73
Figura 30 – Corrente quantizada de quatro diferentes dispositivos elétricos 74
Figura 31 – Trajetória V-I de quatro diferentes dispositivos elétricos 74
Figura 32 – Matriz de confusão para o *Random Forest* utilizando as imagens da trajetória V-I 76
Figura 33 – Acurácia em função da taxa de amostragem 77
Figura 34 – Estrutura da RNC 78
Figura 35 – Medidas-F e matriz de confusão do experimento de (BAETS et al., 2017b) . 79

Figura 36 – Medidas-F e matriz de confusão usando o PLAID1: Parte I 86
Figura 37 – Medidas-F e matriz de confusão usando o PLAID2: Parte I 87
Figura 38 – Medidas-F e matriz de confusão usando o PLAID1+2: Parte I 88
Figura 39 – Medidas-F e matriz de confusão usando o PLAID1: Parte II 90
Figura 40 – Medidas-F e matriz de confusão usando o PLAID2: Parte II 91
Figura 41 – Medidas-F e matriz de confusão usando o PLAID1+2: Parte II 92
Figura 42 – Medidas-F e matriz de confusão usando o PLAID1 no modelo proposto . . 96
Figura 43 – Medidas-F e matriz de confusão usando o PLAID2 no modelo proposto . . 98
Figura 44 – Medidas-F e matriz de confusão usando o PLAID1+2 no modelo proposto . 99
Figura 45 – Exemplo simplificado de quantização 106
Figura 46 – Representação do uso das portas seriais do sistema 109
Figura 47 – Fluxograma da função de leitura 111
Figura 48 – Fluxograma da função de inferência 112
Figura 49 – Fluxograma da função de retorno das inferências 113
Figura 50 – Fluxograma da função de repetição 114
Figura 51 – Vista simplificada, em árvore, do modelo *hello_world* gerado 142
Figura 52 – Vista em árvore do projeto do *PlatformIO* 143
Figura 53 – Vista em árvore do diretório *tfmicro* 145
Figura 54 – Vista em árvore do diretório *tfmicro* reorganizado 146
Figura 55 – Vista em árvore do projeto do *TensorFlow Lite* no *PlatformIO* 147

Lista de tabelas

Tabela 1 – Descrição dos tipo de *feedback* 25
Tabela 2 – Tabela resumida com as equações das métricas mais utilizadas 51
Tabela 3 – Evolução na distribuição dos dispositivos e instâncias no PLAID 52
Tabela 4 – Especificações técnicas do módulo ESP-WROOM32 55
Tabela 5 – Resultado numérico das seleções 58
Tabela 6 – Arquitetura da RNC utilizada 65
Tabela 7 – Medidas-F obtidas por dispositivo no experimento de Baptista et al. (2018) . 66
Tabela 8 – Tabela dos *hardwares* utilizados por artigos selecionados 67
Tabela 9 – Tabela de grandezas mensuradas por artigos selecionados 68
Tabela 10 – Tabela de características por artigos selecionados 69
Tabela 11 – Tabela de técnicas de aprendizado por artigos selecionados 70
Tabela 12 – Tabela de métricas por artigos selecionados 71
Tabela 13 – Classificadores e parâmetros utilizados 75
Tabela 14 – Acurácia dos classificadores nos experimentos 76
Tabela 15 – Tabela comparativa dos artigos selecionados 80
Tabela 16 – Classificadores e parâmetros utilizados 83
Tabela 17 – Acurácias dos classificadores e características testadas usando o PLAID1 . . 83
Tabela 18 – Acurácias dos classificadores e características testadas usando o PLAID2 . . 84
Tabela 19 – Acurácias dos classificadores e características testadas usando o PLAID1+2 85
Tabela 20 – Arquitetura da RNC usada por Baets et al. (2017b) 85
Tabela 21 – Relatório de classificação da RNC usando o PLAID1: Parte I 85
Tabela 22 – Relatório de classificação da RNC usando o PLAID2: Parte I 86
Tabela 23 – Relatório de classificação da RNC usando o PLAID1+2: Parte I 88
Tabela 24 – Relatório de classificação da RNC usando o PLAID1: Parte II 89
Tabela 25 – Relatório de classificação da RNC usando o PLAID2: Parte II 90
Tabela 26 – Relatório de classificação da RNC usando o PLAID1+2: Parte II 91
Tabela 27 – Arquitetura da RNC do modelo proposto 95
Tabela 28 – Relatório de classificação da RNC usando o PLAID1 no modelo proposto . 96
Tabela 29 – Relatório de classificação da RNC usando o PLAID2 no modelo proposto . 97
Tabela 30 – Relatório de classificação da RNC usando o PLAID1+2 no modelo proposto 98
Tabela 31 – Comparativo entre as medidas-F do modelo proposto e dos experimentos reproduzidos 99
Tabela 32 – Tabela comparativa dos artigos selecionados atualizada 100
Tabela 33 – Relatório de classificação da RNC utilizando o método *holdout* 103
Tabela 34 – Relatório de classificação da RNC para todas as instâncias do PLAID1+2 . . 104
Tabela 35 – Sumário com os parâmetros do modelo compilado 105

Tabela 36 – Relatório de classificação do modelo da RNC pós-quantização 107
Tabela 37 – Relatório de classificação do modelo carregado no ESP32 (PLAID1+2) . . . 117

Lista de códigos

Código 1 – Definição do modelo da RNC à nível de programação 102
Código 2 – Parâmetros de compilação do modelo 103
Código 3 – Divisão do conjunto de dados utilizando o método *holdout* 104
Código 4 – Conversão do modelo do *TensorFlow* em *TensorFlow Lite* 105
Código 5 – Função geradora de conjunto de dados representativo usando imagens . . . 106
Código 6 – Função geradora de conjunto de dados representativo usando um vetor . . . 107
Código 7 – Quantização do modelo do *TensorFlow Lite* 107
Código 8 – Bibliotecas necessárias para compilar o projeto do *TensorFlow Lite* 108
Código 9 – Inicialização dos parâmetros do *TensorFlow Lite* 108
Código 10 – Função de configuração do projeto do *TensorFlow Lite* 109
Código 11 – Declaração da matriz do modelo do *TensorFlow Lite* 110
Código 12 – Função que faz a leitura das imagens enviadas pela porta serial 111
Código 13 – Função responsável pela inferência do modelo do *TensorFlow Lite* 112
Código 14 – Função responsável pelo envio das inferências pela porta serial 112
Código 15 – Função de repetição do projeto do *TensorFlow Lite* 113
Código 16 – Função de inferência do modelo do *TensorFlow Lite* 114
Código 17 – Função de inferência do modelo carregado no ESP32 115
Código 18 – Função de avaliação do modelo do *TensorFlow Lite* 115
Código 19 – Função de avaliação do modelo carregado no ESP32 116
Código 20 – Inicialização da classe de avaliação do modelo 116
Código 21 – Funcionamento da classe de avaliação do modelo 117

Lista de abreviaturas e siglas

ADC	*Analog-to-Digital Converter*
ALM	*Appliance Load Monitoring*
AMPds	*Almanac of Minutely Power data set*
ANN	Artificial Neural Network
API	*Application Programming Interface*
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CBP	*Constructive Back Propagation*
CNN	*Convolutional Neural Network*
CPU	*Central Processing Unit*
CUDA	*Compute Unified Device Architecture*
CVD	*Continuously Variable Devices*
DAC	*Digital-to-Analog Converter*
DT	*Decision Tree*
ELM	*Extreme Learning Machine*
FC	*Fourier Coefficient*
FFT	*Fast Fourier Transform*
FHMM	*Factorial Hidden Markov Model*
FPGA	*Field Programmable Gate Array*
FSM	*Finite State Machine*
GNB	*Gaussian Naive Bayes*
GPIO	*General Purpose Input/Output*
GPU	*Graphics Processing Unit*
HMM	*Hidden Markov Model*
I2C	*Inter-Integrated Circuit*

I2S	*Inter-IC Sound*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
HTTP	*Hypertext Transfer Protocol*
kNN	*k-Nearest Neighbors*
LDA	*Linear Discriminant Analysis*
LED	*Light Emissor Diode*
LGC	*Logistic Regression Classifier*
LM	*Load Monitoring*
MCP	*McCulloch Pitts*
MLP	*Multi-Layer Perceptron*
MSL	Mapeamento Sistemático da Literatura
NILM	*Non-Intrusive Load Monitoring*
ODS	Objetivo de Desenvolvimento Sustentável
OTA	*Over-the-air*
PCA	*Principal Component Analysis*
PCD	*Permanent Consumer Devices*
PLAID	*Plug Load Appliance Dataset*
PNG	*Portable Network Graphics*
PSO	*Particle Swarm Optimization*
QDC	Quadro de Distribuição de Circuito
RAM	*Random-Access Memory*
RF	*Random Forest*
RMS	*Root Mean Square*
RNA	Rede Neural Artificial
RNC	Rede Neural Convolucional

ROM	*Read-Only Memory*
RTC	*Real Time Clock*
SAR	*Successive Approximation Register*
SD	*Secure Digital*
SoC	*System-on-Chip*
SPI	*Serial Peripheral Interface*
SQL	*Structured Query Language*
SRAM	*Static Random-Access Memory*
SVM	*Support Vector Machine*
TC	Transformador de Corrente
TP	Transformador de Potencial
TPU	*Tensor Processing Unit*
UART	*Universal Asynchrounous Receiver/Transmiter*
USB	*Universal Serial Bus*
WT	*Wavelet Transform*
ZCD	*Zero Crossing Detector*

Lista de símbolos

A	Ampere
Hz	Hertz
KB	Kilobytes
kHz	Quilohertz
MHz	Megahertz
MB	Megabytes
ms	Milissegundos
s	Segundos
μs	Microssegundos
V	Volts
VA	Volt-Ampere
VAr	Volt-Ampere-reativo
VArh	Volt-Ampere-reativo-hora
W	Watt
Wh	Watt-hora

Sumário

1 Introdução 22
1.1 Contextualização e Motivação 22
1.2 Justificativa 24
1.3 Objetivos 26
1.4 Metodologia Geral 27
1.5 Estrutura do Documento 27

2 Fundamentação Teórica 29
2.1 Cálculos das Potências 29
2.1.1 Potência Instantânea 29
2.1.2 Potência Média ou Ativa 30
2.1.3 Potência Reativa 31
2.1.4 Potência Aparente 32
2.1.5 Fator de Potência 32
2.1.6 Valor Eficaz 33
2.2 Desagregação de Cargas 34
2.2.1 Aquisição dos Dados 37
2.2.2 Detecção de Eventos 38
2.2.3 Extração de Características 38
2.2.4 Técnicas de Aprendizado 39
2.3 Redes Neurais Artificiais 39
2.3.1 Funções de Ativação 41
2.3.2 Arquiteturas de RNA 41
2.3.3 Tipos de Aprendizagem 42
2.3.4 Redes *Perceptron* de Multicamadas 43
2.3.5 Redes Neurais Convolucionais (RNC) 44
2.3.5.1 Entrada 45
2.3.5.2 Camada Convolucional 46
2.3.5.3 Camada de *Pooling* 48
2.3.5.4 Camada *Fully Connected* 48
2.3.6 Medidas de Desempenho 49
2.4 *Plug Load Appliance Identification Dataset* (PLAID) 51
2.5 *TensorFlow* 52
2.6 *Espressif* ESP32 54

3 Trabalhos Relacionados 56

3.1 Mapeamento Sistemático da Literatura 56
3.1.1 Objetivo do Mapeamento 56
3.1.2 Questões de Pesquisa 56
3.1.3 Termos de Busca 57
3.1.4 Seleção das Fontes 57
3.1.5 Critérios de Inclusão e Exclusão 57
3.1.6 Seleção dos Artigos 58
3.1.7 Análise dos Artigos Selecionados 58
3.1.7.1 Quais estudos existentes na literatura que desenvolveram um sistema de desagregação de cargas aplicado em um sistema embarcado, computador *single-board* ou plataforma de prototipagem? 59
3.1.7.1.1 *Leveraging Smart Meter Data to Recognize Home Appliances* 59
3.1.7.1.2 *Inspiring Energy Conservation Through Open Source Metering Hardware and Embedded Real-Time Load Disaggregation* 59
3.1.7.1.3 *A Non-intrusive Load Monitoring System Using an Embedded System for Applications to Unbalanced Residential Distribution Systems* 60
3.1.7.1.4 *Smart Meter Led Probe for Real-Time Appliance Load Monitoring* 60
3.1.7.1.5 *High Frequency Non-Intrusive Electric Device Detection and Diagnosis* 61
3.1.7.1.6 *SEADS: A Modifiable Platform for Real Time Monitoring of Residential Appliance Energy Consumption* 61
3.1.7.1.7 *Nonintrusive Load Monitoring (NILM) Using an Artificial Neural Network in Embedded System with Low Sampling Rate* 62
3.1.7.1.8 *Non-Intrusive Appliances Load Monitoring (NILM) for Energy Conservation in Household with Low Sampling Rate* 62
3.1.7.1.9 *Interpreting Human Activity From Electrical Consumption Data Using Reconfigurable Hardware and Hidden Markov Models* 63
3.1.7.1.10 *Smart Meter Based on Time Series Modify and Constructive Backpropagation Neural Network* 63
3.1.7.1.11 *Smart Meter Based on Time Series Modify and Extreme Learning Machine* 64

3.1.7.1.12 *Smart Meter Based on Time Series Modify and Neural Network for Online Energy Monitoring* 64
3.1.7.1.13 *Implementation Strategy of Convolution Neural Networks on Field Programmable Gate Arrays for Appliance Classification Using the Voltage and Current (V-I) Trajectory* 65
3.1.7.1.14 *An Online Non-Intrusive Load Monitoring Method Based on Hidden Markov Model* 66
3.1.7.2 Quais são os *hardwares* mais utilizados? 66
3.1.7.3 Quais são as grandezas elétricas comumente mensuradas? . . 67
3.1.7.4 Quais são as características mais utilizadas para distinção das cargas? 68
3.1.7.5 Quais são os classificadores mais utilizadas no reconhecimento das cargas? 69
3.1.7.6 Quais métricas mais utilizadas para avaliação dos classificadores? 70
3.1.7.7 Quais são os *datasets* mais utilizadas para avaliação dos classificadores? 71
3.1.7.8 As informações são transmitidas pelo dispositivo para o meio externo? 72
3.2 Outros Trabalhos 72
3.2.1 *A Feasibility Study of Automated Plug-Load Identification From High-Frequency Measurements* 72
3.2.2 *Appliance Classification Using VI Trajectories and Convolutional Neural Networks* 77
3.3 Tabela Comparativa 79

4 Escolha do Classificador e da Característica de Distinção 81
4.1 Metodologia 81
4.2 Experimentos 82
4.2.1 Análise da Imagem Binária da Trajetória V-I Como Característica de Distinção das Cargas 82
4.2.2 Análise da RNC aplicada a imagem binária da trajetória V-I 84
4.3 Considerações Finais do Capítulo 92

5 Proposta da Rede Neural Convolucional 94
5.1 Arquitetura 94
5.2 Metodologia 94
5.3 Experimento 95
5.4 Tabela Comparativa 99
5.5 Considerações Finais do Capítulo 101

6 *TensorFlow Lite* e ESP32: Rede Neural Convolucional Embarcada 102
6.1 Treinamento do Modelo Proposto 102
6.2 Conversão em Modelo do *TensorFlow Lite* 104
6.3 Quantização do Modelo do *TensorFlow Lite* 105
6.4 Aplicação do Modelo do *TensorFlow Lite* no ESP32 108
6.5 Avaliação do Modelo do *TensorFlow Lite* Aplicado no ESP32 114
6.6 Considerações Finais do Capítulo 117

7 Conclusões 119
7.1 Contribuições 120
7.2 Dificuldades e Limitações 121
7.3 Trabalhos Futuros 121

Referências 122

Apêndices 132

APÊNDICE A Guia Básico de Utilização do *Dataset* PLAID 133

APÊNDICE B Geração do Modelo do *TensorFlow Lite* para o ESP32 141

APÊNDICE C Programação no ESP32 usando o *VS Code* e o *PlatformIO* 143

APÊNDICE D Integração do modelo do *TensorFlow Lite* ao *PlatformIO* 145

1 Introdução

Este capítulo apresenta uma visão geral do trabalho, abordando a motivação e a justificativa para o tema escolhido, os objetivos gerais e específicos, a metodologia geral e a estrutura do documento.

1.1 Contextualização e Motivação

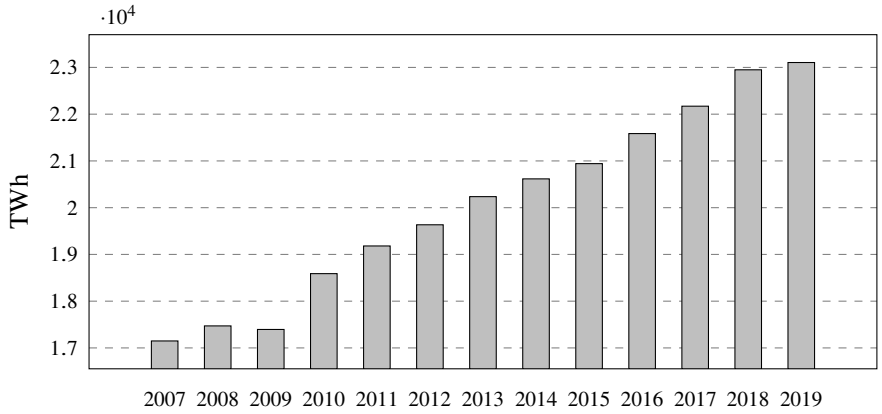
Durante a Primeira Revolução Industrial, no início do século XVIII, foi descoberta a aplicação do carvão mineral como meio de fonte de energia, trazendo como consequência as máquinas a vapor e as locomotivas. O carvão deu suporte ao crescimento e às transformações da economia nesta época, enquanto que o petróleo era usado somente para sistemas de iluminação (FREITAS, 2019).

Na segunda metade do século XIX, com o advento da Segunda Revolução Industrial, a energia elétrica, que já era conhecida e tinha seu uso restrito às pesquisas laboratoriais, passou a ser utilizada como um tipo de energia que tinha a vantagem de poder ser transmitida a longas distâncias e com custo menor, se comparado ao vapor utilizando carvão (SOUSA, 2019). Com a invenção da lâmpada incandescente, importante marco nos sistemas de iluminação da época por substituir o gás, tornou-se ainda mais atrativo o desenvolvimento da energia elétrica (SOUSA, 2017).

Em pouco tempo, a energia elétrica tornou-se de suma importância para o desenvolvimento econômico dos países (ESEN; BAYRAK, 2017; KHAN et al., 2016), sendo importante ainda hoje principalmente aos países em desenvolvimento (REILLY, 2015). Sua relevância é tamanha que levá-la a aproximadamente um bilhão e meio de pessoas que ainda não têm acesso a ela é um dos maiores desafios globais do século XXI (IEA, 2019).

Seu consumo vem crescendo em um ritmo vertiginoso, mais rápido do que os outros modais energéticos. Numericamente falando, como pode ser visto na Figura 1, esse consumo

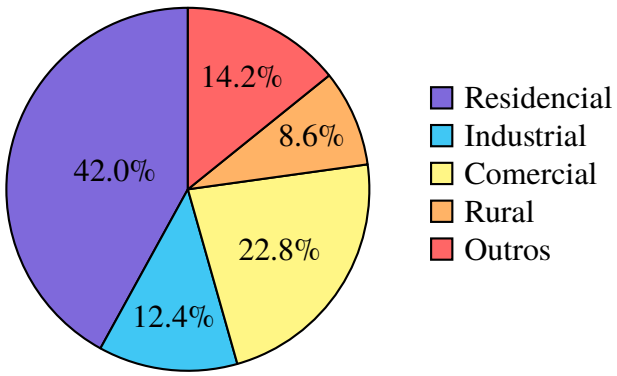
Figura 1 – Gráfico da progressão do consumo mundial de energia elétrica



Fonte – Adaptada de Enerdata (2019)

mundial cresceu em torno de 5000 TWh entre 2007 e 2019 (ENERDATA, 2019). Atualmente, com a popularização dos carros elétricos no mundo (IEA, 2018), estudos já revelam uma tendência no aumento do consumo de energia elétrica (ZHUK; BUZOVEROV, 2018).

Figura 2 – Gráfico do consumo cativo de energia elétrica por classe



Fonte – Adaptada de EPE (2018)

O Brasil não se comportou de forma diferente, obtendo um crescimento de aproximadamente 120 TWh (ENERDATA, 2019) entre 2007 e 2017, configurando-se como o oitavo maior consumidor em 2016 (CIA, 2016). Os setores residencial e comercial são os detentores da maior parcela de consumo cativo[1] no Brasil, sendo responsáveis, em 2017, por 42,0% e 22,8%, respectivamente, do total de energia elétrica consumida (EPE, 2018), como pode ser visualizado na Figura 2.

Concomitantemente com o aumento do consumo, surge também a preocupação com o meio ambiente e o desenvolvimento sustentável, principalmente em países que geram energia elétrica usando matrizes mais poluentes, como as que queimam combustíveis fósseis (EIA, 2018).

[1] "Consumidor ao qual só é permitido comprar energia elétrica da distribuidora detentora da concessão ou permissão na área onde se localizam as instalações do acessante, e, por isso, não participa do mercado livre e é atendido sob condições reguladas" (ANEEL, 2019).

Além do incentivo ao uso de energias renováveis e de menor impacto ambiental, há também duas preocupações: criar dispositivos cada vez mais eficientes (EPA, 2019) e reduzir o desperdício de energia elétrica, buscando alternativas para um uso mais eficiente desta (IEA, 2013).

O avanço da tecnologia, principalmente na área de eletrônica e comunicação, permitiu o surgimento da Rede Elétrica Inteligente (do inglês, *Smart "Electric" Grid*), que é um sistema avançado de energia elétrica com infraestrutura de comunicação integrada para permitir o fluxo bidirecional dessa energia e de informações (USMAN; SHAMI, 2013). Essa rede consegue integrar as ações de todos os usuários conectados a ela, sejam geradores, consumidores ou geradores-consumidores, com o objetivo de fornecer energia elétrica sustentável, segura e econômica (KHAN et al., 2016; ZAFAR et al., 2018).

O envolvimento ativo dos consumidores resulta, na maioria das vezes, em um uso mais eficiente da energia elétrica (USMAN; SHAMI, 2013), aumentando o interesse no desenvolvimento de tecnologias que os conscientizem quanto aos seus hábitos (TAVARES et al., 2018). Diante disso, vêm sendo desenvolvidos novos medidores de energia elétrica, os chamados medidores inteligentes (do inglês, *Smart Meters*), cujo foco passa a ser não somente a medição, mas também o armazenamento e envio de informações e a integração com sistemas de gerenciamento de energia elétrica (BARAI; KRISHNAN; VENKATESH, 2015).

Um dos temas importantes citado para os medidores inteligentes é o Monitoramento Não-Intrusivo de Cargas (do inglês, *Non-Intrusive Load Monitoring* – NILM), também conhecido como desagregação de cargas. Essa técnica permite, por exemplo, distinguir quais eletrodomésticos estão ligados, em determinado momento, usando apenas as leituras de tensão e corrente da residência. A esses medidores, que são vistos como medidores inteligentes aprimorados, dá-se o nome de medidor cognitivo (MAKONIN; POPOWICH; GILL, 2013).

O aumento no consumo de energia elétrica, ano após ano, faz com que pautas relacionadas ao uso mais eficiente da mesma sejam cada vez mais recorrentes. Sabendo que uma das formas conscientes de se economizar energia elétrica é levando informações detalhadas ao consumidor e sabendo também dos avanços tecnológicos na área da eletrônica e das redes de aprendizagem profunda, contribuiremos para o estado da arte fazendo a classificação de dispositivos elétricos para o problema de desagregação de cargas utilizando uma técnica de aprendizagem profunda em um dispositivo embarcado.

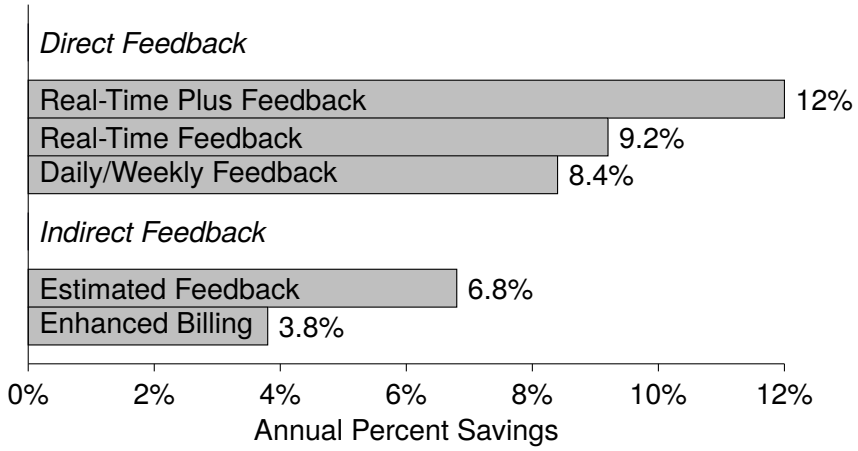
1.2 Justificativa

A Agenda 2030, proposta pela ONU (2019), é um plano de ação para as pessoas, o planeta e a prosperidade que busca fortalecer a paz universal (PLATAFORMA AGENDA 2030, 2020). São indicados 17 Objetivos de Desenvolvimento Sustentável (ODS), cujo cumprimento é esperado de todos os países, de acordo com suas próprias prioridades, até o ano de 2030. O sétimo ODS propõe assegurar o acesso à energia elétrica de forma confiável, sustentável,

moderno e a preço acessível para todos. Além de reduzir desperdícios dos recursos naturais que temos disponíveis, o uso racional da energia elétrica é importante também por economizar dinheiro com a redução do valor da fatura (CPFL, 2016).

Um estudo conduzido pelo *American Council for an Energy-Efficient Economy* (ACEEE) em vários países indicou que quanto maior o detalhamento de informações acerca do consumo elétrico, maior a quantidade de energia elétrica economizada pelos consumidores (EHRHARDT-MARTINEZ; DONNELLY; LAITNER, 2010). Esse estudo foi realizado em vários países e dividido em duas fases: uma anterior à crise de petróleo de 1973 e outra posterior à mesma. Considerando a fase posterior à crise do petróleo, com dados de 1995 até meados de 2010, notou-se que munir os consumidores com informações sobre seu consumo aumentou a economia de energia elétrica em 8,2% (EHRHARDT-MARTINEZ; DONNELLY; LAITNER, 2010).

Figura 3 – Média de energia economizada por tipo de *feedback*



Fonte – Adaptada de Ehrhardt-Martinez, Donnelly e Laitner (2010)

Tabela 1 – Descrição dos tipo de *feedback*

Tipo de *Feedback*	**Descrição**
Real-Time Plus Feedback	Informações em tempo-real, ou quase tempo-real, de custo e consumo de energia desagregada por aparelho elétrico.
Real-Time Feedback	Informações em tempo-real, ou quase tempo-real, de custo e consumo de energia de toda a residência.
Daily/Weekly Feedback	Utiliza algumas médias ou registros do consumo feitos pelos próprios consumidores, além de relatórios diários ou semanais de consumo de energia fornecidos pelas concessionárias.
Estimated Feedback	Utiliza técnicas estatísticas para desagregar o uso total de energia com base no tipo de família, informações dos aparelhos elétricos e nos dados da fatura, sendo geralmente oferecidas aos clientes pelas concessionárias de energia.
Enhanced Billing	Informações mais detalhadas sobre os padrões de consumo de energia, muitas das vezes incluindo estatísticas comparativas.

Fonte – Adaptada de Ehrhardt-Martinez, Donnelly e Laitner (2010)

Esse estudo também detalha a porcentagem de economia de acordo com o tipo de informação que o consumidor recebe, como observado na Figura 3 e na Tabela 1, sendo que

a economia mais expressiva é obtida através do provimento de informações em tempo real, com a discriminação de consumo por aparelho, resultando em uma economia média de 12%. Impulsionado pelos fatores ambientais, restrições internas de recursos energéticos, envelhecimento da infraestrutura e pela crescente demanda e custo da energia elétrica, estima-se que o mercado global de medidores inteligentes chegue a US$10,4 bilhões até 2022 (GLOBALDATA, 2018).

No mundo afora, já é comum o uso de *smart meters* por parte das concessionárias de energia elétrica (SEI, 2018a), enquanto que no Brasil ainda não há muito investimento na implementação dos mesmos (SEI, 2018b). O avanço tecnológico na área da eletrônica permitiu a criação de dispositivos cada vez menores, mais potentes (BOYLESTAD, 2013) e de baixo custo, como os SoC, tornando possível o desenvolvimento de medidores de energia elétrica com preços mais acessíveis.

Isso contribuiu para que a técnica de Monitoramento Não-Intrusivo de Cargas, que já era pesquisada há algumas décadas, inicialmente estudada por Hart (1992), tornasse-se um tema até hoje estudado e cada vez mais significativo (XU; MILANOVIĆ, 2015; C; Kumar; K, 2019). Tanto grandes empresas, como a Intel e a Belkin, quanto algumas pequenas *start-ups*, como a GetEmme (EUA) e Navetas (UK), investiram na pesquisa e desenvolvimento de medidores de energia elétrica cognitivos (ZEIFMAN; ROTH, 2011).

1.3 Objetivos

O objetivo deste trabalho é realizar a classificação de cargas elétricas (eletrodomésticos, eletroeletrônicos, eletroportáteis, dentre outros) para o problema do Monitoramento Não-Intrusivo de Cargas aplicando um método de aprendizagem profunda em um sistema embarcado. Para isso, foi escolhida uma característica de distinção de cargas, a trajetória tensão-corrente, como entrada de uma Rede Neural Convolucional, que foi embarcada em um Espressif ESP32.

Para facilitar a compreensão dos objetivos gerais e expor de forma mais detalhada as particularidades da abordagem, foram delimitados os objetivos específicos, que são elementares ao desenvolvimento do projeto. São eles:

1. Realizar uma revisão bibliográfica sobre o uso de dispositivos embarcados para desagregação de cargas;
2. Elencar conjuntos de dados populares na classificação de dispositivos elétricos;
3. Identificar a melhor característica de distinção dos dispositivos elétricos;
4. Verificar se a Rede Neural Convolucional apresenta alta acurácia para o problema;
5. Embarcar a rede neural em um dispositivo com limitações de memória e processamento, de baixo custo e acessível no mercado;

6. Tornar os resultados do trabalho reprodutíveis e comparáveis com a de trabalhos relacionados.

1.4 Metodologia Geral

Inicialmente, para que se pudesse ter uma compreensão do estado da arte nesta área de pesquisa, foi feito um Mapeamento Sistemático da Literatura (MSL). Foram identificadas as publicações científicas que utilizaram algum dispositivo embarcado para a realização da desagregação de cargas, abrangendo desde a fase de aquisição de dados até a fase de classificação dos dispositivos elétricos. Desta forma, foi obtido o conhecimento necessário para o desenvolvimento do trabalho. Foram identificados, também, os melhores classificadores e as melhores características de distinção para diferenciar os dispositivos elétricos, assim como o conjunto de dados mais utilizado na área do Monitoramento Não-Intrusivo de Cargas.

De posse dessas informações, foi proposta uma Rede Neural Convolucional, utilizando as imagens da trajetória V-I como entrada, para realizar a classificação dos dispositivos elétricos. Procuramos parâmetros em que conseguíssemos bons resultados sem aumentar demasiadamente a complexidade da rede, a fim de facilitar a aplicação da mesma no dispositivo embarcado. Os testes foram realizados utilizando um conjunto de dados conhecido para o problema, que foi o *Plug Load Appliance Identification Dataset* (PLAID).

Com o modelo da Rede Neural Convolucional já definido, fizemos a conversão desse modelo do *TensorFlow* para o *TensorFlow Lite*, possibilitando a aplicação em microcontroladores. Foi necessário realizar a quantização do modelo, processo no qual os pesos da rede foram convertidos e otimizados para que o tamanho do modelo pudesse ser reduzido e, dessa forma, pudesse ser embarcado no ESP32 [2]. Por fim, para a validação do trabalho, a última etapa foi avaliar o modelo após ter sido embarcado e compará-lo com os resultados do modelo previamente proposto, verificando se houve perda significativa de acurácia.

1.5 Estrutura do Documento

Para melhor entendimento, este documento está estruturado em capítulos, que são:

- Capítulo 2 - Fundamentação Teórica: A fundamentação teórica do projeto é abordada, tendo como objetivo prover as informações necessárias para o entendimento do mesmo;
- Capítulo 3 - Trabalhos Relacionados: Os resultados do Mapeamento Sistemático da Literatura são mostrados, tendo como objetivo revisar o estado da arte;

[2] <https://www.espressif.com/en/products/socs/esp32>

- Capítulo 4 - Escolha do Classificador e da Característica de Distinção: Alguns experimentos são apresentados para testar o uso da imagem binária da trajetória V-I como característica de distinção e a Rede Neural Convolucional como classificador;
- Capítulo 5 - Proposta da Rede Neural Convolucional: O modelo da Rede Neural Convolucional proposto para o problema é apresentado e avaliado, sendo também os resultados comparados com o de trabalhos relacionados.
- Capítulo 6 - *TensorFlow Lite* e ESP32: Rede Neural Convolucional Embarcada: O processo de conversão do modelo da Rede Neural Convolucional do *TensorFlow* para o *TensorFlow Lite* é apresentado, incluindo o processo de quantização do mesmo;
- Capítulo 7 - Conclusões: As considerações finais acerca do trabalho são apresentadas, incluindo as contribuições deste, suas limitações e sugestões para trabalhos futuros.

2
Fundamentação Teórica

Esse capítulo aborda os principais conceitos básicos para o entendimento desta dissertação. São abordados temas fundamentais e teóricos, como cálculo das potências, desagregação de cargas e redes neurais artificiais e, também, temas mais específicos, como o *Plug Load Appliance Identification Dataset*, o *TensorFlow* e o *Espressif ESP32*.

2.1 Cálculos das Potências

Segundo Silva (2019), a potência de uma força representa a velocidade com a qual ela realiza um trabalho ou é transformada em outras formas de energia. Os principais conceitos de potência são potência instantânea, potência média (ou ativa), potência reativa, potência aparente e fator de potência, que serão abordados nesta seção.

2.1.1 Potência Instantânea

A potência, por definição, é a taxa de variação temporal do gasto ou da absorção de energia (NILSSON; RIEDEL, 2009), sendo expressa em linguagem matemática através da Equação 2.1.

$$p = \frac{dw}{dt} \tag{2.1}$$

em que:
p é a potência em watts (W),
w é o trabalho em joules (J),
t é o tempo em segundos (s).

Aplicando a Regra da Cadeia a Equação 2.1, temos que potência e energia são relacionadas com a tensão e corrente, como mostra a Equação 2.2.

$$p = \frac{dw}{dt} = \left(\frac{dw}{dq}\right)\left(\frac{dq}{dt}\right) \tag{2.2}$$

em que:
p é a potência em watts (W),
$\frac{dw}{dq}$ é a definição de tensão em volts (V),
$\frac{dq}{dt}$ é a definição de corrente em amperes (A),

A potência instantânea é a velocidade com que se consome ou se absorve energia, medida em watts (ALEXANDER; SADIKU, 2008). A partir da Equação 2.2 e da definição dos seus termos, a potência instantânea pode ser representada pela Equação 2.3. A unidade de potência no Sistema Internacional de Unidades (SI) é o watt, cujo símbolo é W, em homenagem ao matemático e engenheiro escocês James Watt. Um watt corresponde a um joule por segundo (SEARS et al., 2008).

$$p = v \cdot i \tag{2.3}$$

em que:
p é a potência instantânea em watts (W),
v é a tensão em volts (V),
i é a corrente em amperes (A).

A potência fornecida ou consumida por um elemento é o produto da tensão no elemento pela corrente que passa por ele. Se a potência for positiva, ela está sendo fornecida para o elemento, caso contrário ela está sendo fornecida pelo mesmo. O sentido da corrente e a polaridade da tensão são de suma importância para a determinação do sinal da potência. Para isso, deve ser seguida a convenção do sinal passivo, que nos diz que o sinal é positivo quando a corrente entra pelo terminal positivo de um elemento e, obviamente, negativo quando a corrente entra pelo terminal negativo (ALEXANDER; SADIKU, 2008).

2.1.2 Potência Média ou Ativa

A potência média, também denominada potência ativa, é a potência que é convertida de uma forma elétrica para uma forma não elétrica (NILSSON; RIEDEL, 2009), ou seja, é a potência que realiza trabalho, gerando calor, luz, movimento, etc. É denominada potência média

por ser a média da potência instantânea ao longo de um período (ALEXANDER; SADIKU, 2008), sendo regida pela Equação 2.4.

$$P = \frac{V_m I_m}{2} cos(\theta_v - \theta_i) \tag{2.4}$$

em que:
P é a potência média em watts (W),
V_m é a tensão média em volts (V),
I_m é a corrente média em amperes (A),
θ_v é o ângulo de fase da tensão,
θ_i é o ângulo de fase da corrente.

2.1.3 Potência Reativa

A potência reativa apresenta potência média zero, portanto não transforma energia elétrica em nenhum outro tipo de energia, ou seja, não realiza trabalho útil (NILSSON; RIEDEL, 2009). Essa potência pode ser indutiva ou capacitiva e é regida pela Equação 2.5. A unidade de potência reativa é o volt-ampere reativo, cujo símbolo é var.

$$Q = \frac{V_m I_m}{2} sen(\theta_v - \theta_i) \tag{2.5}$$

em que:
Q é a potência reativa em volt-ampere reativo (var),
V_m é a tensão média em volts (V),
I_m é a corrente média em amperes (A),
θ_v é o ângulo de fase da tensão,
θ_i é o ângulo de fase da corrente.

Os aparelhos elétricos indutivos, por possuírem passagem de corrente pelos enrolamentos, geram um campo magnético interno vital para seu funcionamento. Quando esses elementos são alimentados em corrente alternada, é natural que haja uma tendência da energia armazenada de opor-se à variação da intensidade da corrente, que por consequência gera um atraso da corrente em relação à tensão (CHRISTO, 2005).

Para os aparelhos elétricos capacitivos, a potência é continuamente permutada entre a fonte que excita o circuito e o campo elétrico associado ao elemento capacitivo, gerando um adiantamento da corrente em relação à tensão (NILSSON; RIEDEL, 2009).

2.1.4 Potência Aparente

A potência aparente é uma grandeza mais importante do que a potência média, pois embora esta represente a potência que realiza trabalho, a potência aparente representa a potência total disponível necessária para fornecer a potência média desejada (NILSSON; RIEDEL, 2009).

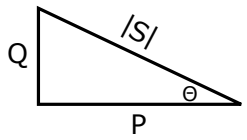


Figura 4 – Triângulo de potências

Essa potência é regida pela Equação 2.6, obtida aplicando o Teorema de Pitágoras ao triângulo de potências, visto na Figura 4 e que nos mostra a relação entre a potência aparente ($|S|$), a potência ativa (P) e a potência reativa (Q). A unidade de potência aparente é o volt-ampere, cujo símbolo é VA.

$$|S| = \sqrt{P^2 + Q^2} \tag{2.6}$$

em que:
$|S|$ é a potência aparente em volt-ampere (VA),
P é a potência ativa em watts (W),
Q é a potência reativa em volt-ampere reativo (var).

2.1.5 Fator de Potência

O fator de potência é a relação entre a potência ativa e a potência aparente (CHRISTO, 2005), como mostra a Equação 2.7. Esse valor pode ser também obtido a partir do cosseno do ângulo do fator de potência, que é $\theta_v - \theta_i$ (NILSSON; RIEDEL, 2009), de tal forma que o fator de potência é regido pela Equação 2.8. Por ser uma relação entre dois valores representados pela mesma unidade de potência, é um número adimensional.

$$pf = \frac{P}{|S|} \tag{2.7}$$

em que:
pf é o fator de potência,
P é a potência ativa em watts (W),
$|S|$ é a potência aparente em volt-ampere (VA).

$$pf = cos(\theta_v - \theta_i) \tag{2.8}$$

em que:
pf é o fator de potência,
θ_v é o ângulo de fase da tensão,
θ_i é o ângulo de fase da corrente.

Quando a onda de corrente está atrasada em relação à onda de tensão, o fator de potência é dito indutivo ou atrasado. Caso contrário, o fator de potência é dito capacitivo ou adiantado. Se as ondas de corrente e tensão estiverem em fase, o fator de potência é unitário e chamado de resistivo (FILHO, 2007; NILSSON; RIEDEL, 2009), sendo a carga constituída somente de potência ativa.

Nesse caso, toda a potência gerada pela fonte é absorvida pela carga, enquanto que aparelhos que envolvem motores elétricos e transformadores, por exemplo, necessitam de energia reativa para seu funcionamento, como visto na seção 2.1.3. Até mesmo as linhas de transmissão e distribuição de energia elétrica são fontes de energia reativa devido à sua reatância (FILHO, 2007).

As concessionárias de energia elétrica protegem-se de elevadas potências reativas em suas linhas impondo ao consumidor um fator de potência mínimo (CHRISTO, 2005), que atualmente é de 0,92 e foi determinado pelo Artigo nº 95 da Resolução ANEEL nº414 (ANEEL, 2012). Caso o consumidor apresente um fator de potência abaixo que o estipulado anteriormente, o mesmo será cobrado pelo excedente de energia reativa.

2.1.6 Valor Eficaz

A necessidade de medir a eficácia de uma fonte de tensão ou de corrente na liberação de potência para uma carga resistiva deu surgimento ao termo valor eficaz, que corresponde ao valor que uma corrente alternada deve ter para produzir a mesma potência em um resistor que a corrente contínua (ALEXANDER; SADIKU, 2008). O valor eficaz para qualquer função periódica cujo sinal seja contínuo no tempo é regido pela Equação 2.9.

Para sinais discretos no tempo, o valor eficaz é regido pela Equação 2.10 (WEISSTEIN, 2017), lembrando que a fim de evitar o fenômeno *aliasing*[1], o *Teorema de Nyquist* deve ser respeitado, ou seja, o valor da taxa de amostragem dever ser pelo menos duas vezes o valor da frequência analógica que será amostrada (IAZZETTA, 2019).

$$X_{RMS} = \sqrt{\frac{1}{T}\int_0^T x(t)^2 dt} \tag{2.9}$$

em que:
X_{RMS} é o valor eficaz,

[1] Segundo Iazzetta (2019), é o espelhamento da frequência para uma região mais grave do espectro.

T é o período,
$x(t)$ é uma função periódica qualquer.

$$X_{RMS} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}} \tag{2.10}$$

em que:
X_{RMS} é o valor eficaz,
n é o número total de amostras,
x é o valor da amostra.

A potência média dada pela Equação 2.4 e a potência reativa dada pela Equação 2.5 podem ser reescritas em termos dos valores eficazes (NILSSON; RIEDEL, 2009), sendo referenciadas pela Equação 2.11 e Equação 2.12, respectivamente.

$$P = V_{rms} I_{rms} cos(\theta_v - \theta_i) \tag{2.11}$$

em que:
P é a potência média em watts (W),
V_{rms} é a tensão eficaz em volts (V),
I_{rms} é a corrente eficaz em amperes (A),
θ_v é o ângulo de fase da tensão,
θ_i é o ângulo de fase da corrente.

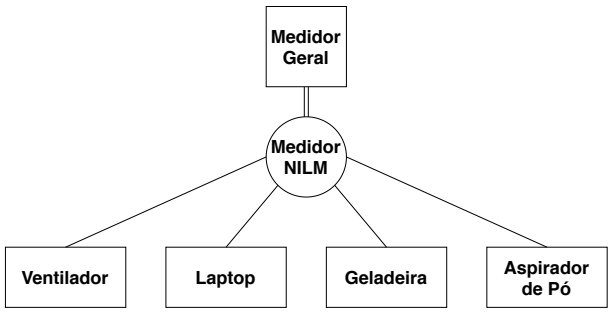
$$Q = V_{rms} I_{rms} sen(\theta_v - \theta_i) \tag{2.12}$$

em que:
Q é a potência reativa em volt-ampere reativo (var),
V_m é a tensão eficaz em volts (V),
I_m é a corrente eficaz em amperes (A),
θ_v é o ângulo de fase da tensão,
θ_i é o ângulo de fase da corrente.

2.2 Desagregação de Cargas

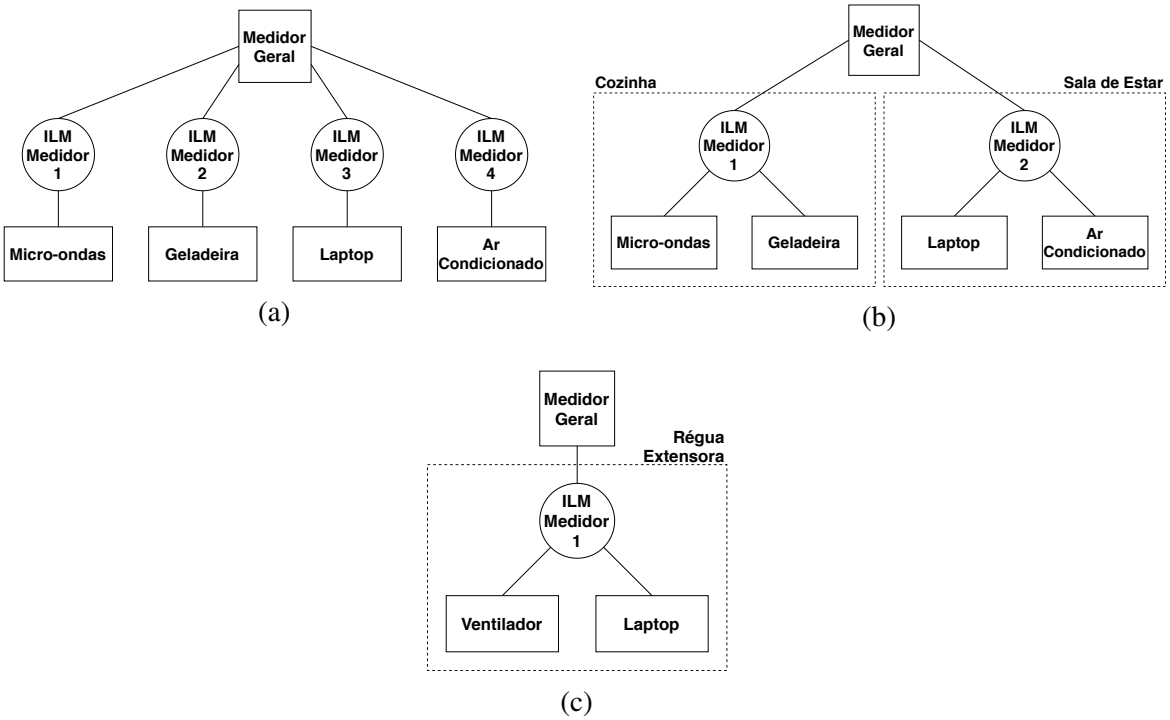
O Monitoramento de Cargas (do Inglês, *Load Monitoring* – LM), também chamado de *Appliance Load Monitoring* (ALM), tornou-se fundamental para o entendimento do consumo e economia de energia elétrica. Segundo Ridi, Gisler e Hennebert (2014), existem duas abordagens de monitoramento:

Figura 5 – Exemplo de abordagem NILM



- ***Non-Intrusive Load Monitoring* (NILM) ou Monitoramento Não-Intrusivo de Cargas:** A medição de consumo de energia é feita usando somente um medidor, geralmente instalado no quadro de distribuição principal da residência (ZEIFMAN; ROTH, 2011), como exemplificado na Figura 5. Usando essa técnica, as assinaturas dos dispositivos são sobrepostas, sendo necessária a desagregação das cargas. Devido a essa sobreposição, o termo desagregação de cargas virou sinônimo do Monitoramento Não-Intrusivo de Cargas (MAKONIN; POPOWICH; GILL, 2013).

Figura 6 – Exemplo de abordagem ILM



- ***Intrusive Load Monitoring* (ILM) ou Monitoramento Intrusivo de Cargas:** A medição de consumo de energia é feita utilizando mais de um medidor, sendo geralmente um para cada dispositivo a ser monitorado (Figura 6a), por região específica da residência (Figura 6b) ou por conjunto de dispositivos ligados a uma régua extensora (Figura 6c).

A abordagem NILM é estudada há mais tempo, sendo Sultanem (1991) e Hart (1992) os pioneiros deste estudo. Devido ao menor número de sensores, essa abordagem tem como vantagens a facilidade e velocidade na instalação e o custo reduzido. Já a abordagem ILM, possui como vantagens a maior precisão das medições, devido a quantidade de medidores utilizados, a redução da sobreposição, gerando maior detalhamento das assinaturas dos dispositivos e a melhor detecção de aparelhos de baixo consumo (RIDI; GISLER; HENNEBERT, 2014). Esses dispositivos podem ser classificados de acordo com os seus diferentes estados de funcionamento:

- **Tipo I (liga/desliga):** Dispositivos que possuem apenas dois estados de operação (LIGA/DESLIGA). Nessa classe estão incluídos dispositivos puramente resistivos, como lâmpadas incandescentes, chuveiros elétricos, sanduicheiras, entre outros (RIDI; GISLER; HENNEBERT, 2014). Um gráfico que exemplifica o funcionamento desse tipo de classe é mostrado na Figura 8a;
- **Tipo II (multi-estados):** Dispositivos que possuem finitos estados de operação, sendo que cada um deles possuem um consumo específico. Um método comum de representar essa classe é utilizando as Máquinas de Estados Finitos (do Inglês, *Finite State Machine* – FSM) (KLEMENJAK; GOLDSBOROUGH, 2016), sendo o gráfico de funcionamento mostrado na Figura 8b. Alguns dispositivos que se enquadram nessa classe são: máquina de lavar roupas e ventiladores multivelocidades;
- **Tipo III (infinitos estados):** Dispositivos continuamente variáveis, ou CVDs, possuem potência variável e não possuem um número finito de estados. A desagregação desses dispositivos é um enorme desafio devido à dificuldade na sua identificação (RIDI; GISLER; HENNEBERT, 2014). Alguns deles são: lâmpadas dimerizáveis e furadeiras. Um gráfico que exemplifica o funcionamento desse tipo de classe é mostrado na Figura 8c;
- **Tipo IV (*standby*):** Essa classe foi elencada por Zeifman e Roth (2011) e nela estão os dispositivos que estão continuamente consumindo energia, cujo gráfico de funcionamento é mostrado na Figura 8d. É típico de aparelhos que estão em *standby*, como detectores de incêndio e televisores, por exemplo. Segundo Klemenjak e Goldsborough (2016), essa classe pode ser entendida como uma subclasse do Tipo III.

De forma geral, as etapas do processo NILM seguem um padrão: aquisição de dados, detecção de eventos, extração das características singulares dos aparelhos e a aplicação de uma técnica de aprendizado para a identificação do dispositivo, como mostra a Figura 7.

Figura 7 – Etapas do processo NILM

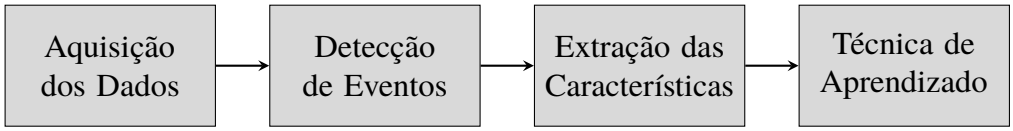
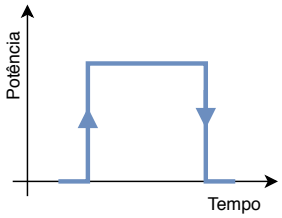
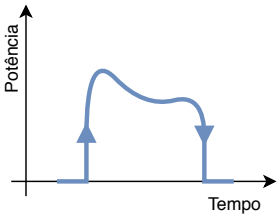


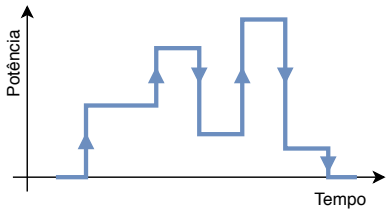
Figura 8 – Classificações dos dispositivos NILM



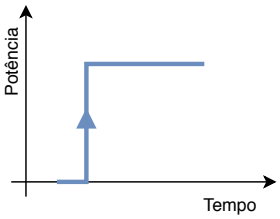
(a) Tipo I (liga/desliga)



(b) Tipo II (multi-estados)



(c) Tipo III (infinitos estados)



(d) Tipo IV (*standby*)

2.2.1 Aquisição dos Dados

Nesta etapa, as informações são obtidas a uma taxa de amostragem adequada para que as assinaturas de carga possam ser identificadas (ZOHA et al., 2012) de acordo com o nível de detalhamento requerido para o problema. Os sistemas de aquisição de dados podem ser divididos em duas categorias:

- **Baixa Frequência (alguns Hz):** É o medidor mais utilizado pelos fornecedores de energia elétrica devido ao baixo custo (ZHUANG; SHAHIDEHPOUR; LI, 2018). Nos últimos anos, devido ao alto custo dos medidores de alta-frequência, tanto a comunidade acadêmica quanto as empresas têm preferido esse tipo de medidor, visto que as grandezas mais comuns podem ser medidas a uma baixa frequência (ZHUANG; SHAHIDEHPOUR; LI, 2018). A precisão do sensor é algo crítico, pois afeta diretamente a qualidade das informações;
- **Alta Frequência (acima de KHz):** Para capturar eventos do estado transitório, ou algum ruído elétrico, a frequência de amostragem precisa ser muito alta, entre 10 e 100 MHz (ZOHA et al., 2012). De forma similar, para capturar as harmônicas, a taxa de amostragem deve estar entre 1 e 11 KHz (ZEIFMAN; ROTH, 2011). Esse tipo de medidor, devido ao *hardware* sofisticado, são mais caros e geralmente são adaptados ao tipo de informação que se deseja extrair (ZOHA et al., 2012).

Caso seja necessário capturar as harmônicas de ordem superior do sinal elétrico, que são múltiplas da frequência fundamental, a taxa de amostragem deve obedecer às regras de amostragem de Nyquist-Shannon, como citado na Seção 2.1.6. Por exemplo, sendo a frequência fundamental de 60 Hz, se for desejada leitura da décima harmônica, será necessária uma taxa de amostragem de pelo menos 1,2 kHz.

2.2.2 Detecção de Eventos

Um evento é definido como a mudança do sinal de um estado para outro. Os métodos de detecção de eventos referem-se a como detectar essa mudança de estado das cargas. Segundo Abubakar et al. (2015), detectores de eventos podem ser categorizados em:

- **Baseado em Eventos:** Esse método utiliza o algoritmo de detecção de borda para detectar uma alteração no sinal do dispositivo, como por exemplo, eventos de ligar e desligar o equipamento. As características de borda são classificadas de acordo com as regras do algoritmo de aprendizado;
- **Não-baseado em Eventos:** Esse método não precisa de detecção de borda para fazer a classificação, usando, em vez disso, inferência nas amostras do valor agregado.

Ainda segundo Abubakar et al. (2015): "métodos baseados em eventos são mais eficientes computacionalmente do que os métodos não-baseados em eventos".

2.2.3 Extração de Características

As assinaturas dos dispositivos elétricos são as características singulares que permitem distinguir, identificar e classificar os mesmos (KLEMENJAK; GOLDSBOROUGH, 2016). As assinaturas também podem ser definidas como os parâmetros mensuráveis da carga total que oferecem informações sobre o tipo e o estado de operação de cada dispositivo (ABUBAKAR et al., 2015). Na aplicação do NILM, os quatro principais tipos de assinaturas identificados são:

- **Assinaturas de Estado Estável:** A assinatura da carga é analisada quando a mesma está em estado estável. A potência real e reativa são as grandezas mais usadas na desagregação de cargas para esse tipo de assinatura, embora haja dificuldades na identificação de cargas do Tipo II e III quando há sobreposição (ABUBAKAR et al., 2015);
- **Assinaturas de Estado Transitório:** A assinatura da carga é analisada quando a mesma está em estado transitório, que ocorre sempre que o dispositivo é ligado, por um curto intervalo de tempo, até que o sinal seja estabilizado e opere em estado estável (BANERJEE, 2015). Por ter um comportamento distinto na maioria dos aparelhos, analisar o transitório torna-se uma opção adequada para a identificação de cargas;
- **Assinaturas Híbridas:** A assinatura da carga é analisada tanto no estado estável quanto em estado transitório, melhorando a precisão de reconhecimento (BANERJEE, 2015);
- **Assinaturas Não-tradicionais:** A assinatura da carga é analisada por outros métodos de identificação, como por exemplo, comportamentos ambientais (como vibração e ruído), frequência de uso do aparelho e duração entre o ligar e desligar (ZOHA et al., 2012; NAGHIBI; DEILAMI, 2014).

2.2.4 Técnicas de Aprendizado

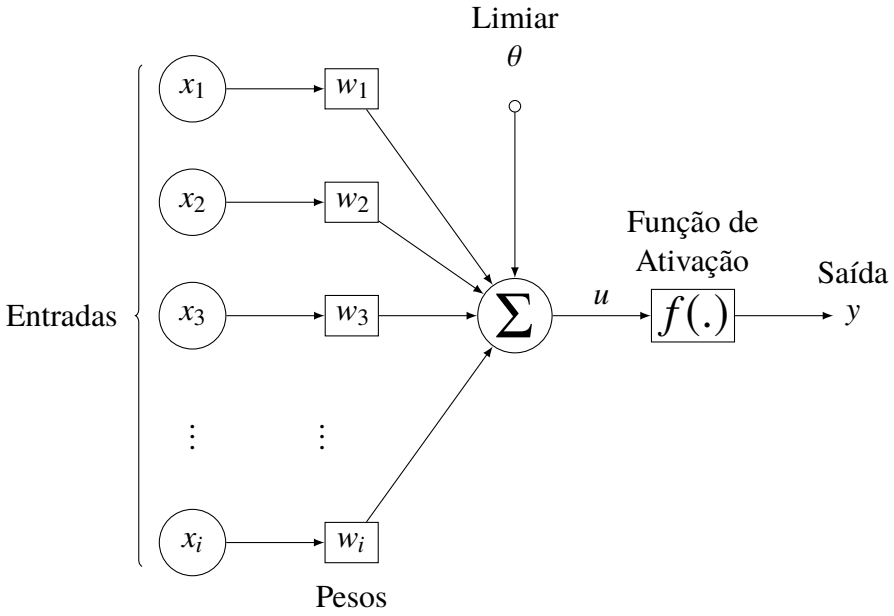
As técnicas de aprendizado podem ser divididas em duas: supervisionada e não-supervisionada. Esses dois tipos de aprendizagem serão abordados futuramente, na Seção 2.3.3.

2.3 Redes Neurais Artificiais

Segundo Haykin (2011): "uma rede neural é um processador maciçamente paralelamente distribuído constituído de unidades de processamento simples, que têm a propensão natural para armazenar conhecimento experimental e torná-lo disponível para o uso". Como o conhecimento é adquirido pela rede por meio de um processo de aprendizagem através da percepção do ambiente e o conhecimento adquirido é armazenado nos pesos sinápticos, as redes neurais assemelham-se ao cérebro humano (HAYKIN, 2011).

O primeiro modelo matemático a representar o neurônio biológico foi elaborado em 1943, em um estudo desenvolvido por McCulloch e Pitts (1943). O modelo proposto extraia apenas o essencial para representar o neurônio biológico com precisão, removendo todos os detalhes desnecessários (MARSLAND, 2014). Esse modelo artificial proposto, chamado *McCulloch Pitts* (MCP), pode ser visto na Figura 9.

Figura 9 – Representação do neurônio artificial



$$u = \sum_{i=1}^{n} x_i \cdot w_i - \theta \tag{2.13}$$

$$y = f(u) \tag{2.14}$$

Matematicamente, as expressões 2.13 e 2.14 descrevem o modelo MCP. O seu funcionamento é descrito da seguinte maneira:

- **Entradas:** São n terminais de entrada que representam os dendritos cerebrais (MARSLAND, 2014);
- **Pesos:** Representam as sinapses cerebrais, e por isso, também são chamados de pesos sinápticos. Os pesos podem ser positivos ou negativos, a depender do tipo de sinapse ao qual corresponde: excitatória ou inibitória, respectivamente (BRAGA, 2007);
- **Combinador Linear (Σ):** O somador dos valores de entrada corresponde à membrana da célula que recebe as descargas elétricas do cérebro (MARSLAND, 2014). O neurônio dispara o impulso quando a soma das entradas recebidas ultrapassam o limiar de excitação (BRAGA, 2007);
- **Função de Ativação:** A função de ativação corresponde à decisão tomada pelo neurônio cerebral em relação ao envio do impulso elétrico (MARSLAND, 2014), sendo também chamada de função de esmagamento, pois limita a amplitude da saída de um neurônio (HAYKIN, 2011). Sua função é ativar ou não a saída do neurônio, a depender do resultado da soma ponderada das entradas e do limiar de excitação (BRAGA, 2007);
- **Saída:** É o resultado único e final produzido pelo neurônio em relação às entradas, sendo geralmente a saída representada como 0 ou 1.

Segundo Braga (2007), a simplificação do modelo de McCulloch e Pitts (1943) considera que todos os nós da camada disparam de forma sincronizada. No sistema biológico não existe nenhum mecanismo sincronizador nem as saídas são disparadas em tempos discretos. Esse modelo também não apresenta técnicas de aprendizado.

Somente em 1958, com o modelo de Rosenblatt (1958), que o conceito de aprendizado em RNAs foi introduzido. Esse modelo, conhecido por *perceptron*, era basicamente um neurônio MCP com algoritmo de treinamento, capaz de ajustar os pesos de entrada. Quatro anos depois, Rosenblatt (1962) demonstrou o teorema de convergência do *perceptron*, provando que o *Perceptron* sempre converge caso o problema seja linearmente separável.

Durante a década de 70 e o início da década de 80, houve um grande desinteresse pela área, principalmente devido aos questionamentos de Minsky e Papert (1969) quanto à capacidade computacional do *perceptron* (BRAGA, 2007). O interesse pela área só aumentou a partir da década de 80, com a rede de Hopfield (1982) e o algoritmo *backpropagation*, proposto por Rumelhart, Hinton e Williams (1986), e que se tornou um dos mais importantes algoritmos de treinamento.

2.3.1 Funções de Ativação

A função de ativação limita, dentre um intervalo finito de valores, a saída do neurônio (SILVA; SPATTI; FLAUZINO, 2010). As principais funções de ativação são: a função linear, a função rampa, a função degrau e a função sigmoidal (BRAGA, 2007).

- **Função Linear:** É matematicamente definida pela equação 2.15;

$$f(u) = a \cdot u \tag{2.15}$$

- **Função Rampa:** É definida, matematicamente, pela equação 2.16;

$$f(u) = \begin{cases} +\alpha & \text{se } u \geq +\alpha \\ u & \text{se } |u| < +\alpha \\ -\alpha & \text{se } u \leq -\alpha \end{cases} \tag{2.16}$$

- **Função Degrau:** Também chamada de função de *Heaviside*, é matematicamente definida pela equação 2.17;

$$f(u) = \begin{cases} +\alpha & \text{se } u \geq 0 \\ -\alpha & \text{se } u < 0 \end{cases} \tag{2.17}$$

- **Função Sigmoidal:** Também chamada de função *S-shape*, é uma das mais utilizadas. Uma função sigmoidal bastante conhecida é a logística, matematicamente definida pela equação 2.18.

$$f(u) = \frac{1}{1 + e^{-u/T}} \tag{2.18}$$

2.3.2 Arquiteturas de RNA

A arquitetura de rede está diretamente ligada ao problema que será tratado e relacionada ao algoritmo de aprendizagem utilizado para o treinamento (SILVA; SPATTI; FLAUZINO, 2010). Os parâmetros que fazem parte da definição da arquitetura, segundo Braga (2007), são:

- **Número de camadas da rede:** Pode ser de uma única camada ou múltiplas camadas;
- **Número de nós em cada camada:** O número de nós não segue uma regra, sendo determinado experimentalmente;
- **Tipo de conexão entre os nós:** Os neurônios podem estar totalmente ou parcialmente conectados;
- **Topologia de rede:** A topologia divide-se em *feedfoward* e *feedback*:
  - ***Feedfoward*:** A saída de um nó numa determinada camada da rede não pode ser usada como entrada de nós na mesma camada ou em camadas anteriores;

  - ***Feedback*:** A saída de um nó numa determinada camada da rede pode ser usada como entrada de nós na mesma camada ou em camadas anteriores.

2.3.3 Tipos de Aprendizagem

A capacidade de aprender a partir de exemplos ou de conjuntos de treinamento é um dos destaques mais relevantes das RNAs. O processo de treinamento da rede consiste na aplicação de etapas bem definidas para ajustar parâmetros da RNA tendo como objetivo a generalização das respostas dadas pelas suas saídas (SILVA; SPATTI; FLAUZINO, 2010). Ao conjunto dessas etapas bem definidas, visando o treinamento da rede, dá-se o nome algoritmo de aprendizagem.

A etapa de aprendizagem é um processo iterativo de ajuste dos parâmetros da rede, como os pesos sinápticos, que guarda, ao final, o conhecimento adquirido pela rede sobre o ambiente que está operando (BRAGA, 2007). Existem vários métodos de aprendizado, que são agrupados em dois paradigmas:

- **Aprendizado Supervisionado:** É chamado de supervisionado porque a entrada e a saída desejadas são fornecidas por um supervisor externo. Tem como objetivo ajustar os parâmetros da rede relacionando a entrada com a saída. O supervisor informa a rede sobre o erro entre o valor de sua saída e o valor da saída desejado. Dessa forma, para cada entrada, a comparação é feita e os pesos vão sendo ajustados para minimizar o erro. A desvantagem desse aprendizado é que a rede não consegue aprender novas estratégias na ausência do supervisor (BRAGA, 2007), ou seja, ela só funcionará adequadamente para os casos em que foi treinada. O aprendizado supervisionado pode ser implementado de duas formas:
  - ***Offline*:** Os dados do conjunto de treinamento não variam, ou seja, quando a solução da rede é definida, ela se mantém fixa. Caso algum novo dado surja, um novo processo de treinamento deve ser realizado;
  - ***Online*:** Os dados do conjunto de treinamento variam continuamente, ou seja, a rede deve adaptar-se constantemente.
- **Aprendizado Não-supervisionado:** No aprendizado não-supervisionado não existe um supervisor que acompanhe o processo de aprendizado. Para essa tipo de aprendizado, somente as informações de entrada estão disponíveis para rede, ou seja, a própria rede deve se auto-organizar a partir do seu conjunto de entradas (SILVA; SPATTI; FLAUZINO, 2010). A abordagem só se torna possível quando existe redundância nesses dados de entrada, permitindo assim encontrar quaisquer padrões ou características para que agrupamentos por similaridade possam ser criados (BRAGA, 2007).

2.3.4 Redes *Perceptron* de Multicamadas

As redes de camada única resolvem apenas problemas que são linearmente separáveis (BRAGA, 2007). Para resolver os problema não-linearmente separáveis, são utilizadas redes multicamadas. Uma das redes multicamadas mais utilizadas é a MLP (*Multi-Layer Perceptron*) (MARSLAND, 2014).

A rede MLP é caracterizada pela presença de ao menos uma camada escondida de neurônios, situada entre as camadas de entrada e saída. Além disso, destaca-se pela versatilidade, sendo utilizada em várias áreas de conhecimento para tratar vários tipos diferentes de problemas (SILVA; SPATTI; FLAUZINO, 2010).

Uma rede com uma camada escondida, ou camada intermediária, permite a aproximação de qualquer função contínua (CYBENKO, 1989), enquanto que uma rede com duas camadas ocultas permite a aproximação de qualquer função (CYBENKO, 1988). A depender dos dados de entrada, o treinamento da rede pode convergir para um mínimo local ou demorar demais para encontrar uma solução (BRAGA, 2007).

Nesse tipo de rede, o processamento de cada nó está diretamente ligado ao processamento realizado pelos nós da camada anterior conectados a ele. Do ponto de vista da arquitetura da rede, sua topologia é do tipo *feedforward*, ou seja, não há realimentação. A rede possui uma camada de entrada, camadas intermediárias e a camada de saída. As camadas intermediárias servem como detectores de características, gerando codificações dos padrões de entrada que serão utilizadas na definição da saída da rede (BRAGA, 2007).

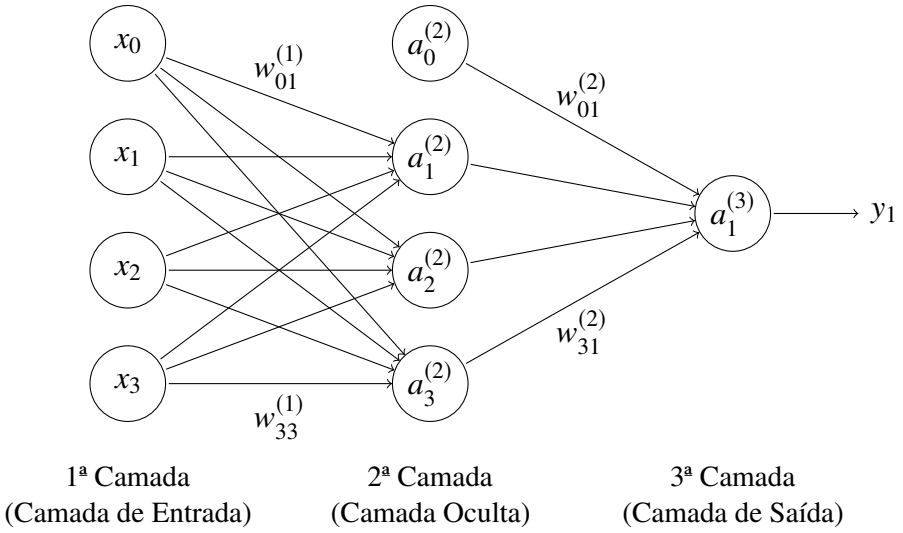
O uso de duas ou mais camadas intermediárias pode facilitar o treinamento da rede, porém o excesso dessas camadas não é recomendado, pois o erro medido torna-se menos preciso a cada propagação para a camada anterior (BRAGA, 2007). A quantidade de nós, de forma geral, é obtida de forma empírica, dependendo da distribuição dos padrões de treinamento e da validação da rede.

O algoritmo de aprendizado *back-propagation*, popularizado por Rumelhart, Hinton e Williams (1986), é o mais conhecido para treinamento das MLP. É um algoritmo supervisionado que utiliza os valores de entrada e saída desejados para ajustar os pesos da rede através da correção de erros (BRAGA, 2007). Esses pesos são ajustados baseados na regra delta, proposta por Widrow e Hoff (1988), que utiliza o método gradiente para tal.

Observando a rede da Figura 10, percebemos visualmente uma característica fundamental da MLP, que é o fato da mesma ser totalmente conectada, ou seja, todos os neurônios da camada n são ligados a todos os neurônios da camada $n + 1$. A rede MLP ilustrada é composta por 4 unidades na camada de entrada, 3 unidades na camada oculta e uma unidade na camada de saída.

Segundo Florencio et al. (2018), Gershenson (2003), o erro total da rede é definido pela

Figura 10 – Redes *Perceptron* de multicamadas



Fonte – Adaptada de Raschka e Mirjalili (2017)

Equação 2.19, enquanto que os pesos da camada são atualizados pela Equação 2.20.

$$E = \frac{1}{2}\sum_{j=1}^{n}(d_j - y_j)^2 \tag{2.19}$$

em que:
d_j é o valor desejado para a j-ésima posição,
y_j é o valor da j-ésima posição do vetor de saída da MLP.

$$\omega' = \omega - \eta\frac{\partial E}{\partial \omega} \tag{2.20}$$

em que:
ω' é o valor do peso atualizado da camada,
ω é o valor do peso da camada,
η é a taxa de aprendizado.

2.3.5 Redes Neurais Convolucionais (RNC)

Uma Rede Neural Convolucional – RNC (do inglês, *Convolutional Neural Network* – CNN), é uma classe de Rede Neural Artificial do tipo *feedforward* amplamente utilizada na área de reconhecimento de padrões (LI et al., 2018). Teve seu desenvolvimento inspirado na neurobiologia, tendo como base o estudo pioneiro de Hubel e Wiesel (1962), que estudaram a fisiologia do córtex visual de um gato, identificando as células visuais que respondiam a diferentes tipos de estímulo (HAYKIN, 2011).

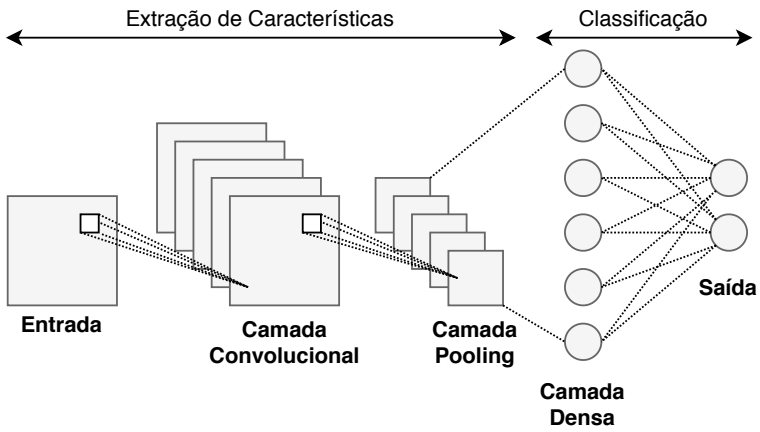
Baseado nesse estudo, Fukushima (1979) propôs uma rede neural multicamadas hierárquica chamada *neo-cognitron*, que introduziu conceitos de camadas semelhantes às camadas convolucionais e de *pooling* das RNCs atuais. Durante a década de 80, reconhecer dígitos escritos manualmente ou impressos, independente do tamanho ou da fonte, era um problema de grande interesse tanto prático quanto teórico (KAHAN; PAVLIDIS; BAIRD, 1987). Em 1987, com o estudo de Denker et al. (1989), essa abordagem foi aplicada no reconhecimento de dígitos manuscritos de códigos postais fornecidos pelo serviço postal dos EUA.

Em 1989, diferentemente dos resultados anteriores relatados pelo grupo de Denker et al. (1989) para o problema, LeCun et al. (1989) usaram o algoritmo de *back-propagation* e alimentaram a rede de aprendizado diretamente com imagens, em vez de vetores, obtendo os melhores resultados para a época e mostrando que as redes baseadas em retropropagação podiam lidar com grandes quantidades de informação.

Esse tipo automático de aprendizado foi adequado para vários tipos de problema de reconhecimento de imagens, sendo que quase uma década posteriormente, LeCun et al. (1998) propuseram a LeNet-5, uma rede convolucional pioneira com 7 camadas (3 convolucionais, 2 de *pooling*, 1 *fully-connected* e a saída) que foi aplicada em diversas instituições bancárias para o reconhecimento de dígitos manuscritos em cheques digitalizados como imagens de 32×32 pixels.

Segundo Haykin (2011), uma rede convolucional "é um perceptron multicamada projetado especificamente para reconhecer formas bidimensionais com um alto grau de invariância à translação, escalonamento, inclinação e outras formas de distorção". As RNCs são compostas, basicamente, pelos seguintes tipos de camadas: convolucional, de *pooling* e *fully-connected*, além de possuírem também as entradas, a função de ativação e a saída. Um exemplo de topologia da RNC pode ser visto na Figura 11.

Figura 11 – Exemplo de topologia da Rede Neural Convolucional

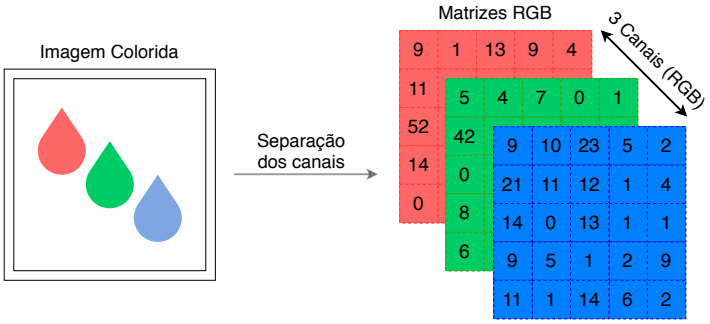


2.3.5.1 Entrada

As entradas são matrizes usualmente tridimensionais (com altura, largura e profundidade, sendo esta última determinada pela quantidade de canais de cores, como mostra a Figura 12),

ou bidimensionais (ALVES, 2018). Para exemplificar com *datasets* conhecidos, uma imagem do CIFAR10[2] é representada por uma matriz tridimensional, enquanto que uma do MNIST[3] é representada por uma matriz bidimensional.

Figura 12 – Exemplo de entrada tridimensional



2.3.5.2 Camada Convolucional

O termo Rede Neural Convolucional sugere o emprego da operação matemática de convolução, um tipo de operação linear que a partir de duas funções dadas produz uma terceira, que expressa como a forma de uma é modificada pela outra (ATANGANA, 2017).

De forma geral, matematicamente falando, a integral de convolução de duas funções, $x_1(t)$ e $x_2(t)$, é simbolicamente representada por $x_1(t) * x_2(t)$, sendo o asterisco $(*)$ usado como símbolo para definir esta operação (LATHI, 2007).

A integral de convolução é definida como a integral do produto de duas funções, depois que uma é revertida e deslocada, sendo essa integral aplicada para todos os valores de deslocamento existente, como mostra a Equação 2.21.

$$x_1(t) * x_2(t) \equiv \int_{-\infty}^{\infty} x_1(\tau)x_2(t-\tau)d\tau \tag{2.21}$$

Tratando-se das redes convolucionais, a primeira função, que denominamos de $x_1(t)$, é chamada de *input*, ou entrada, enquanto que a segunda função, a qual denominamos de $x_2(t)$, é chamada de *kernel*, ou filtro (GOODFELLOW; BENGIO; COURVILLE, 2016). A função resultante da convolução das funções anteriores é comumente chamada de *feature map*, ou mapa de características.

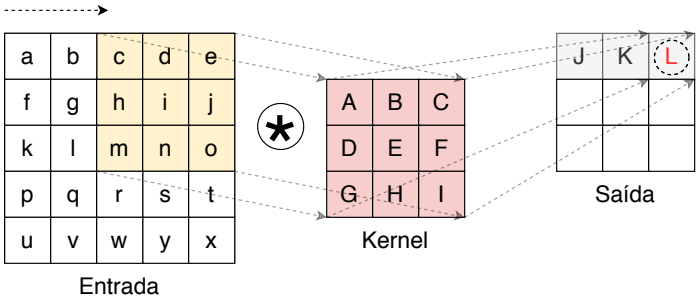
Porém, normalmente, essa operação de convolução utilizada em uma RNC não corresponde integralmente à definição de convolução utilizada em outras áreas, como a engenharia e a matemática (GOODFELLOW; BENGIO; COURVILLE, 2016). Do ponto de vista matemático, a convolução apresenta algumas propriedades, dentre elas: comutatividade, associatividade e

[2] <https://www.cs.toronto.edu/~kriz/cifar.html>
[3] <http://yann.lecun.com/exdb/mnist>

distributividade (LATHI, 2007). Nas convoluções da área de aprendizado de máquina, para que a propriedade da comutatividade seja atendida, o *kernel* deve ser invertido em relação à entrada (GOODFELLOW; BENGIO; COURVILLE, 2016).

Como a propriedade comutativa geralmente não é relevante para a implementação da rede, muitas bibliotecas implementam uma função relacionada, chamada de correlação-cruzada, que é o mesmo que a convolução, porém sem inverter o *kernel* (GOODFELLOW; BENGIO; COURVILLE, 2016). Muitas dessas bibliotecas implementam a correlação-cruzada e as chamam de convolução (GOODFELLOW; BENGIO; COURVILLE, 2016), logo, nesse contexto de aprendizado de máquina, seguiremos a convenção e chamaremos ambas as operações de convolução.

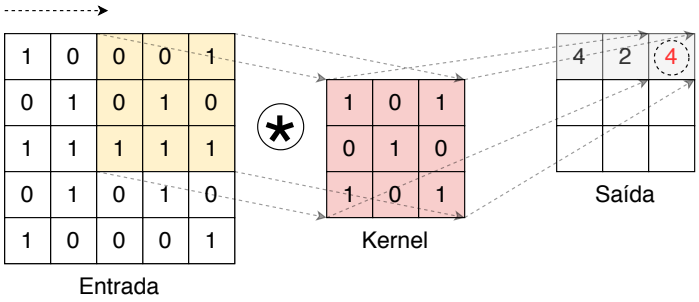
Figura 13 – Exemplo de operação de convolução usando dados literais



A camada convolucional consiste, então, em um conjunto de mapas de características, que são gerados a partir de convoluções sobre os dados de entrada ou outro mapa. A região da entrada onde o filtro é aplicado é conhecida como área receptiva (ALVES, 2018). Um exemplo da operação de convolução pode ser observado na Figura 13, que mostra o *kernel* deslizando sobre a área receptiva e o valor resultante da operação sendo salvo em um mapa de características. Para esse exemplo literal, temos que:

$$J = Aa + Bb + Cc + Df + Eg + Fh + Gk + Hl + Im$$
$$K = Ab + Bc + Cd + Dg + Eh + Fi + Gl + Hm + In$$
$$L = Ac + Bd + Ce + Dh + Ei + Fj + Gm + Hn + Io$$

Figura 14 – Exemplo de operação de convolução usando números



Com a finalidade de tornar mais fácil a compreensão, as letras foram substituídas por números aleatórios no exemplo da Figura 14, na qual temos que:

$$J = (1)(1) + (0)(0) + (1)(0) + (0)(0) + (1)(1) + (0)(0) + (1)(1) + (0)(1) + (1)(1) = 4$$
$$K = (1)(0) + (0)(0) + (1)(0) + (0)(1) + (1)(0) + (0)(1) + (1)(1) + (0)(1) + (1)(1) = 2$$
$$L = (1)(0) + (0)(0) + (1)(1) + (0)(0) + (1)(1) + (0)(0) + (1)(1) + (0)(1) + (1)(1) = 4$$

2.3.5.3 Camada de *Pooling*

A camada de *pooling* tem como objetivo alcançar certo nível de invariância ao deslocamento da entrada, reduzindo a resolução dos mapas de características (GU et al., 2018), ou seja, caso a entrada sofra um pequeno deslocamento, os valores da maioria das saídas não mudam. Segundo Goodfellow, Bengio e Courville (2016), a função de *pooling* substitui a saída da rede em um determinado local por uma estatística resumida das saídas próximas, ou seja, de uma forma geral, ela serve para simplificar a informação da camada anterior.

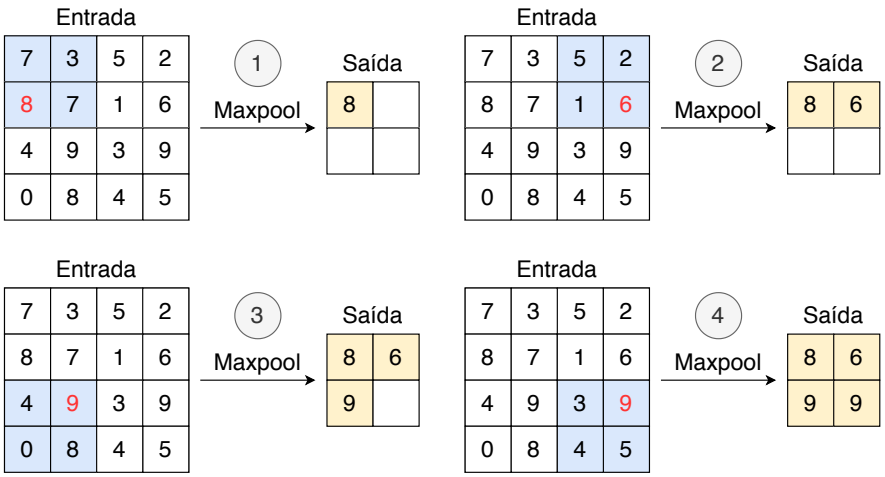
Apesar de existirem algumas outras funções populares de *pooling* envolvendo a média ou a normalização L2 da vizinhança retangular ou a média ponderada baseada na distância do pixel central (GOODFELLOW; BENGIO; COURVILLE, 2016), a sumarização dos dados geralmente é feita utilizando a função de *maxpooling* (ZHOU; CHELLAPPA, 1988), na qual apenas o maior número de um campo retangular é passado para a saída (ALVES, 2018). Um exemplo de sumarização dos dados utilizando essa função pode ser observado na Figura 15.

Nesse exemplo, a unidade de área escolhida foi de 2×2, correspondente aos quadrados azuis, logo o tamanho da entrada será reduzido pela metade. A informação dessa unidade de área será resumida em um único valor, que de acordo com a função *maxpooling* será o número de maior valor dessa área. Como a entrada escolhida possui tamanho 4×4, após a sumarização possuirá tamanho 2×2. Cada uma das quatro etapas mostra o passo-a-passo da operação, sendo a saída final correspondente ao quadrado 2×2 amarelo.

2.3.5.4 Camada *Fully Connected*

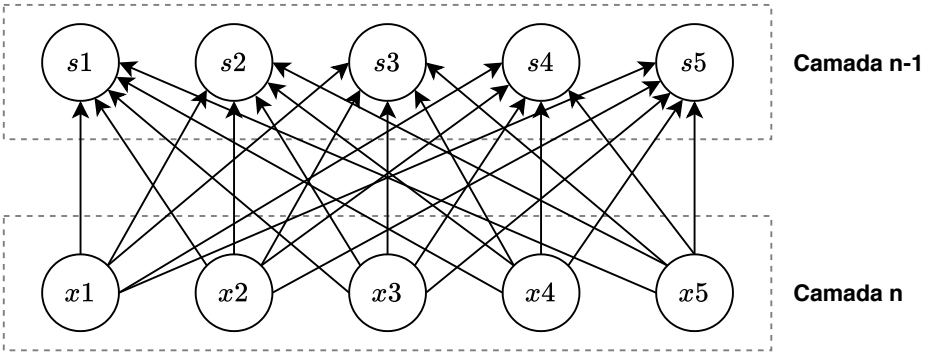
A camada totalmente conectada, do inglês *fully connected*, também conhecida como camada densa, funciona como uma rede neural multicamadas de arquitetura *feedforward*, como a MLP, sendo responsável pela interpretação das características extraídas pelas camadas iniciais (EBERMAM; KROHLING, 2018). Seu nome se deve ao fato de todos os nós da camada serem conectados aos nós da camada anterior (GOODFELLOW; BENGIO; COURVILLE, 2016), como mostrado na Figura 16. A saída dessa camada são N neurônios, sendo N a quantidade de classes do modelo (ALVES, 2018).

Figura 15 – Exemplo de operação de *pooling*



Fonte – Adaptado de Alves (2018)

Figura 16 – Exemplo de camada densa



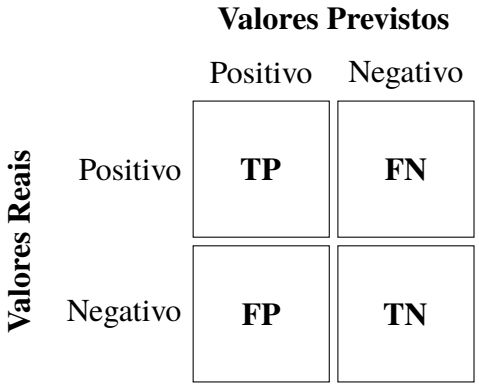
Fonte – Adaptado de Goodfellow, Bengio e Courville (2016)

2.3.6 Medidas de Desempenho

Na área de aprendizado de máquina, tipicamente no aprendizado supervisionado, é comum o uso de matrizes de confusão (KOHAVI; PROVOST, 1998), ou matrizes de erro, como também são conhecidas, para auxiliar na avaliação do desempenho de um sistema. A matriz de confusão contém informações sobre classificações reais e previstas, como é ilustrado na Figura 17, que representa uma matriz literal e de duas classes, sendo as notações utilizadas explicadas posteriormente.

- **Verdadeiro Positivo (TP):** Indica a proporção de casos positivos que foram previstos corretamente.
- **Verdadeiro Negativo (TN):** Indica a proporção de casos negativos que foram previstos corretamente.
- **Falso Positivo (FP):** Indica a proporção de casos negativos que foram previstos incorretamente como positivos.

Figura 17 – Matriz de confusão literal para classificação binária



- **Falso Negativo (FN):** Indica a proporção de casos positivos que foram previstos incorretamente como negativos.

Por meio da matriz de confusão, as seguintes métricas, cujas equações podem ser vistas na Tabela 2, podem ser calculadas:

- **Acurácia:** Taxa geral de acerto, ou seja, a divisão de todos os acertos em relação ao conjunto total de dados:
$$A_c = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}}$$
- **Precisão:** Indica a relação entre as predições positivas acertadas e todas as previsões positivas:
$$P_r = \frac{\text{TP}}{\text{TP} + \text{FP}}$$
- **Revocação:** Também conhecida como *Recall*, Cobertura ou Sensitividade. É a proporção dos casos positivos que são corretamente preditos positivos:
$$R_e = \frac{\text{TP}}{\text{TP} + \text{FN}}$$
- **Medida-F:** Também conhecida como F_1*-Score*, *F-Score* ou *F-Measure*. Facilita a visualização das métricas Precisão e Revocação juntas, sendo uma média harmônica entre as duas:
$$F_1 = \frac{2P_rR_e}{P_r + R_e} = \frac{2\text{TP}}{2\text{TP} + \text{FP} + \text{FN}}$$
- **Medida-G:** Também conhecida como *G-Measure* ou *Fowlkes-Mallows Score*. É definida como a média geométrica entre a Precisão e a Revocação:
$$G = \sqrt{P_rR_e} = \frac{\text{TP}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})}}$$

Para todas as métricas citadas anteriormente, quanto maior o valor das mesmas, melhor é o resultado.

Tabela 2 – Tabela resumida com as equações das métricas mais utilizadas

Símbolo	**Métrica**	**Equação**
A_c	Acurácia	$A_{cc} = \frac{\text{TP+TN}}{\text{TP+FP+TN+FN}}$
P_r	Precisão	$P_r = \frac{\text{TP}}{\text{TP+FP}}$
R_e	Revocação	$R_e = \frac{\text{TP}}{\text{TP+FN}}$
F_1	Medida-F	$F_1 = \frac{2P_rR_e}{P_r+R_e} = \frac{2\text{TP}}{2\text{TP+FP+FN}}$
G	Medida-G	$G = \sqrt{P_rR_e} = \frac{\text{TP}}{\sqrt{(\text{TP+FP})(\text{TP+FN})}}$

2.4 *Plug Load Appliance Identification Dataset* (PLAID)

O *Plug Load Appliance Identification Dataset* (PLAID) (GAO et al., 2014) está disponível publicamente em <http://www.plaidplug.com>. Este conjunto de dados inclui medições de tensão e corrente amostradas a uma frequência de 30 kHz, de 11 diferentes tipos de dispositivos elétricos em 55 residências de Pittsburgh, no estado estadunidense da Pennsylvania. Para cada dispositivo elétrico, existem dezenas de instâncias diferentes, variando a marca e o modelo do mesmo.

Essas medições passaram por um pré-processamento para extração de janelas de 1 segundo de largura, contendo tanto uma parte do sinal em estado estável quanto os transientes, quando disponíveis, ao ligar o aparelho. Inicialmente, contava com 235 modelos diferentes de dispositivo e com 1094 instâncias, porém as medições que apresentavam muitos ruídos ou erros foram removidas do conjunto de dados, do qual sobraram 1074 instâncias no total, como pode ser visto na Tabela 3.

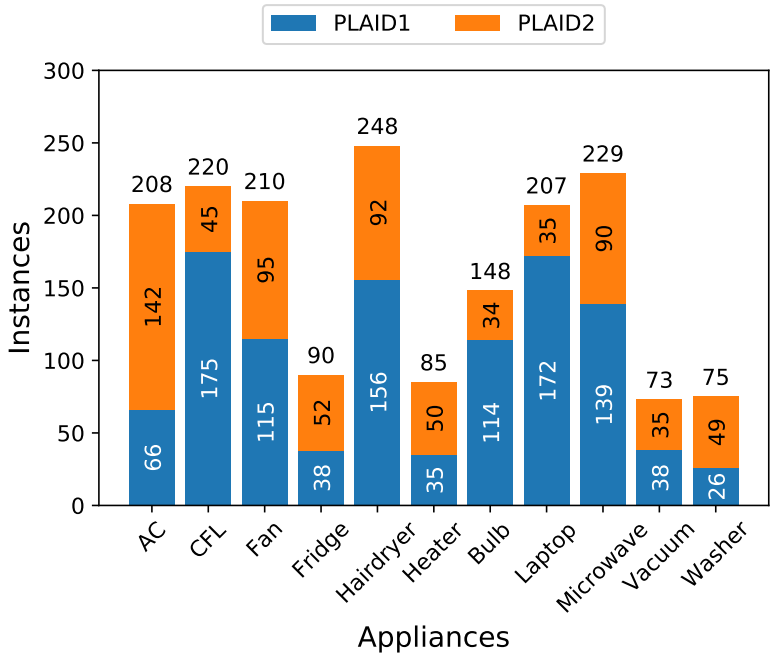
Em 2017, o PLAID2 (BAETS et al., 2017a) foi anunciado como uma extensão do PLAID, que acrescentou 719 novas instâncias, como pode ser visto na Tabela 3. Nessa extensão, as medições de tensão e corrente também foram amostradas a uma frequência de 30 kHz, dos mesmos 11 diferentes tipos de dispositivos elétricos e realizada em mais 9 residências de Pittsburgh. Além de estender numericamente o número de instâncias, o PLAID2 passou também a monitorar o estado de operação do dispositivo, por exemplo, a transição de velocidade do ventilador do ar-condicionado (alto, médio e baixo).

Dessa forma, o PLAID1+2 possui um total de 1793 instâncias, com medições de 11 diferentes tipos de dispositivos elétricos e realizado em 64 residências de Pittsburgh. Essas instâncias foram divididas de maneira tal qual é mostrada na Tabela 3 e na Figura 18.

Tabela 3 – Evolução na distribuição dos dispositivos e instâncias no PLAID

	PLAID1		PLAID2		PLAID1 + PLAID 2	
Dispositivo Elétrico	**Modelos**	**Instâncias**	**Modelos**	**Instâncias**	**Modelos**	**Instâncias**
Air Conditioner (AC)	14	66	7	142	21	208
Compact Fluorescent Lamp (CFL)	31	175	9	45	40	220
Fan	23	115	7	95	30	210
Fridge	21	38	9	52	30	90
Hairdryer	32	156	5	92	37	248
Heater	7	35	6	50	13	85
Incandescent Light Bulb (Bulb)	23	114	7	34	30	148
Laptop	34	172	7	35	41	207
Microwave	28	139	9	90	37	229
Vacuum	8	38	7	35	15	73
Washing Machine (Washer)	9	26	9	49	18	75
Total	**230**	**1074**	**82**	**719**	**312**	**1793**

Figura 18 – Gráfico de instâncias por tipo de dispositivo (PLAID1+2)



2.5 *TensorFlow*

A plataforma para aprendizado de máquina *TensorFlow*, cujo logotipo pode ser viso na Figura 19, é uma biblioteca de código aberto para computação numérica e criada pela equipe do *Google Brain* (ABADI et al., 2015), que se dedica à pesquisa de inteligência artificial de aprendizagem profunda no *Google*. A biblioteca foi escrita a partir de uma outra biblioteca de redes de aprendizado profundo (*deep-learning*), chamada *DistBelief*, que é um sistema distribuído para treinamento de redes neurais que o *Google* usa desde 2011 (DEAN et al., 2012).

O *TensorFlow* é baseado em grafos computacionais, que é um grafo direcionado em que

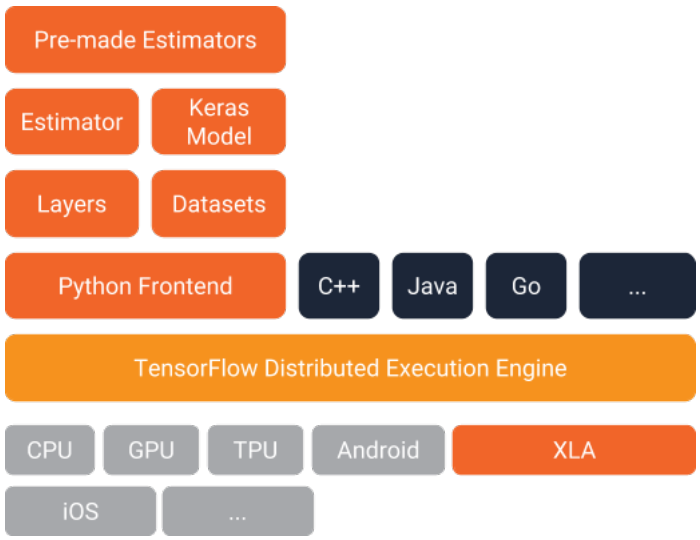
Figura 19 – Logotipo do *TensorFlow*

Fonte – Wikipedia (2020)

os nós representam as operações matemáticas e as arestas representam o fluxo de dados entre os nós (ABADI et al., 2016). Essa característica torna a biblioteca utilizável em qualquer domínio em que a computação possa ser modelada como um grafo de fluxo de dados. O nome da biblioteca, *TensorFlow*, é dado baseado em como os tensores, que são um tipo de vetor multidimensional, fluem na rede (PARVAT et al., 2017). Segundo Parvat et al. (2017), o *TensorFlow* foi escrito com uma API *Python* na linguagem C/C++.

Por ser multiplataforma, o *TensorFlow* pode ser usado no *Windows*, *macOS* e *Linux*. Com o *TensorFlow Lite*, está disponível também em dispositivos móveis e plataformas embarcadas, como o *Android OS*, *Raspberry Pi* e ESP32. Dessa forma, tem a capacidade de criar e implantar os modelos de aprendizado de máquina em uma variedade de plataformas, desde *smartphones* a grandes *clusters* (PARVAT et al., 2017). Pode ser executado em CPUs, GPUs e TPUs (*hardwares* desenvolvidos pela *Google* especificamente para acelerar o processamento do aprendizado de máquina de redes neurais).

Figura 20 – Arquitetura do TensorFlow 2.0



Fonte – Sharma (2019)

Visualizando a arquitetura resumida do *TensorFlow*, Figura 20, é notável a presença dos

muitos dispositivos suportados; do *TensorFlow Distributed Execution Engine*, que é o núcleo de alto desempenho implementado em C++; *Frontend*, com as APIs de desenvolvimento em *Python*, C++, Java e Go; as *Layers*, para construção dos modelos (DATA SCIENCE ACADEMY, 2018); os *Datasets*, com alguns conjuntos de dados prontos para uso; *Estimator* e *Keras Model*, que auxiliam no treino e avaliação dos modelos; e os *Pre-made Estimators*, que são modelos de aprendizado comuns e já prontos para o uso (LINHARES, 2017).

2.6 *Espressif* ESP32

A Internet das Coisas (do Inglês, *Internet of Things* – IoT) é um paradigma que se tornou popular rapidamente no cenário das modernas telecomunicações sem fio. Seu conceito está relacionado com a presença generalizada de vários objetos ao nosso redor (como sensores, atuadores, *smartphones*, dentre outros) que são capazes de interagir entre si (ATZORI; IERA; MORABITO, 2010). De forma geral, a IoT é considerada parte da Internet do futuro que incluirá bilhões de dispositivos inteligentes a rede (LI; XU; ZHAO, 2015).

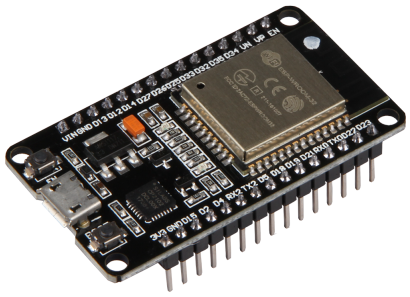
O mercado da IoT expandiu-se rapidamente nos últimos anos, trazendo cada vez mais a necessidade de soluções poderosas, contando com dispositivos de alto processamento, de baixo custo e baixo consumo de energia (MAIER; SHARP; VAGAPOV, 2017). Além disso, para aumentar a aplicabilidade do dispositivo, um outro requisito é possuir tamanho diminuto. Segundo Maier, Sharp e Vagapov (2017), cada unidade baseada em IoT é composta por um microcontrolador e um módulo de comunicação sem fio, ou uma combinação dos dois.

Um dos microcontroladores mais famosos é o ESP8266[4], fabricado pela empresa chinesa *Espressif Systems*[5] e comercializado a partir de 2014. O produto foi lançado inicialmente com quase nenhuma documentação em inglês e o seu atrativo foi atribuído ao seu preço, inicialmente um valor inferior a 10 dólares, bastante similar aos microcontroladores sem interface de rede (BENCHOFF, 2017). O módulo originalmente vem pré-programado com o conjunto de comandos *Hayes*, também conhecido como comandos AT, linguagem de comando desenvolvida em 1981 por Dennis Hayes para estabelecer comunicação em modems Hayes (DALAKOV, 2020).

Em 2016, a Espressif lançou o *System-on-Chip* (SoC) ESP32[6], que comparado ao antecessor, teve uma melhora significativa de processamento, de comunicação e de entrada/saída (SINGH; KAPOOR, 2017), tornando-se uma solução bastante atrativa para a IoT. O ESP-WROOM 32, visto na Figura 21, por exemplo, conta com uma unidade central de processamento *dual-core* de 32 bits, modelo Xtensa LX6 que opera entre 160 e 240 MHz, memória RAM de 520 KB e memória *flash* de 4 MB. Suas especificações podem ser vistas com mais detalhes na Tabela 4.

[4] <https://www.espressif.com/en/products/socs/esp8266>
[5] <https://www.espressif.com>
[6] <https://www.espressif.com/en/products/socs/esp32>

Figura 21 – Módulo ESP-WROOM32 com SoC ESP32



Fonte – Joy-It (2020)

Tabela 4 – Especificações técnicas do módulo ESP-WROOM32

ESP32 (Módulo ESP-WROOM 32)	
CPU / Memória	
CPU:	Dual-core 32-bit Xtensa LX6
ROM:	448 KB
SRAM:	520 KB
FLASH:	4 MB
Conectividade	
Wi-Fi:	802.11 b/g/n
Bluetooth:	4.2 BR/EDR + BLE
UART:	3
I/O	
GPIO:	34
ADC:	18 (12-bit) SAR
DAC:	2 (8-bit)
SPI:	4
I2S:	5
I2C:	5
Outras	
Tamanho (CxLxA):	52 mm x 28 mm x 5 mm
Preço Aproximado:	US$4

3
Trabalhos Relacionados

O objetivo deste capítulo é contextualizar o trabalho proposto com trabalhos contemporâneos e relevantes a fim de confirmar a importância do tema proposto.

3.1 Mapeamento Sistemático da Literatura

Foi adotado como método de pesquisa um Mapeamento Sistemático da Literatura (MSL), que segundo Petersen et al. (2008), fornece uma estrutura do tipo de pesquisa, assim como seus resultados publicados categorizados.

3.1.1 Objetivo do Mapeamento

O objetivo deste mapeamento é analisar as publicações científicas e descobrir as suas contribuições e limitações. Foram selecionadas publicações que desenvolveram um sistema de desagregação de cargas que compreenda alguma das etapas desde a aquisição dos dados até a aplicação de algum algoritmo de desagregação de cargas, ou de inferência, utilizando sistemas embarcados, computadores *single-board* ou plataformas de prototipagem.

3.1.2 Questões de Pesquisa

Segundo Petersen et al. (2008), a definição das questões de pesquisa é o primeiro passo do processo de mapeamento, fornecendo uma visão geral da área de pesquisa, inclusive identificando os resultados disponíveis dentro dela. Foram definidas cinco questões para guiar a pesquisa, que são:

(Q1) Quais estudos existentes na literatura que desenvolveram um sistema de desagregação de cargas aplicado em um sistema embarcado, computador *single-board* ou plataforma de prototipagem?

(Q2) Quais são os *hardwares* mais utilizados?

(Q3) Quais são as grandezas elétricas comumente mensuradas?

(Q4) Quais são as características mais utilizadas para distinção das cargas?

(Q5) Quais são os classificadores mais utilizadas no reconhecimento das cargas?

(Q6) Quais métricas mais utilizadas para avaliação dos classificadores?

(Q7) Quais são os *datasets* mais utilizadas para avaliação dos classificadores?

(Q8) As informações são transmitidas pelo dispositivo para o meio externo?

3.1.3 Termos de Busca

Como forma de identificar os artigos científicos nas bases de dados, foi necessário definir os termos de busca que seriam utilizados. As principais palavras-chave identificadas foram: "*load disaggregation*" ou "*non intrusive load monitoring*", respectivamente, em português, "desagregação de cargas" e "monitoramento não invasivo de cargas". Para reduzir os resultados a publicações mais práticas, foi utilizado também o termo "*meter*". Dessa forma, o resultado da *string* é apresentado a seguir: ***("load disaggregation" OR "non*intrusive load monitoring") AND ("meter")***.

3.1.4 Seleção das Fontes

Após a definição das questões de pesquisa e dos termos de busca, foram selecionadas as seguintes bases: *Elsevier Scopus*[1], *ACM Digital Library*[2], *IEEE Xplore Digital Library*[3] e *Science Direct*[4]. Para a seleção das mesmas, foram definidos os seguintes critérios: disponibilidade por meio do Portal de Periódicos da CAPES[5] e disponibilidade de artigos por meio do domínio da Universidade Federal de Sergipe (UFS).

3.1.5 Critérios de Inclusão e Exclusão

Os critérios de inclusão e exclusão devem ser definidos para guiarem os pesquisadores na seleção de estudos que foram coletados das bases de pesquisa. Esses critérios tornam a pesquisa mais confiável e permitem uma padronização e impessoalidade dos pesquisadores durante a seleção dos artigos. Foram definidos os seguintes critério de inclusão (CI) e de exclusão (CE):

CI1 - Serão selecionadas publicações que possuam um sistema de classificação de cargas aplicado em sistema embarcado, computador *single-board* ou plataforma de prototipagem;

[1] <https://www.scopus.com>
[2] <https://dl.acm.org>
[3] <https://ieeexplore.ieee.org>
[4] <https://www.sciencedirect.com>
[5] <https://www.periodicos.capes.gov.br>

CE1 - Não serão selecionadas publicações que não satisfaçam os critérios de inclusão;

CE2 - Não serão selecionadas publicações incompletas, ou que tornem o entendimento do todo incompreensível e o sistema irreprodutível;

CE3 - Não serão selecionadas publicações apenas conceituais;

CE4 - Não serão selecionadas publicações duplicadas;

CE5 - Não serão selecionadas publicações escritas em idiomas diferente do Inglês ou Português.

3.1.6 Seleção dos Artigos

As duas fases de seleção adotadas por Petersen et al. (2008) foram aplicadas aos 335 artigos resultantes da pesquisa. Após a primeira etapa de seleção, que consiste na leitura dos títulos, resumos e palavras-chave e em verificar se os mesmos estão relacionados à questão da pesquisa principal, foram selecionados 124 artigos.

A segunda etapa de seleção consiste na identificação e extração dos dados aplicando os critérios de inclusão e exclusão aos 124 artigos selecionados na etapa anterior. Ao fim desta segunda etapa, foram identificados 14 artigos relevantes.

Com a seleção finalizada, foi feita uma análise detalhada dos principais artigos, que serviram para responder às questões de pesquisa. O resultado numérico dessa seleção pode ser visto na Tabela 5.

Tabela 5 – Resultado numérico das seleções

		Resultados	
Base	**Pesquisa**	**1º Filtro**	**2º Filtro**
Scopus	222	81	9
ACM	19	10	0
IEEEXplore	47	15	3
ScienceDirect	47	18	2
Total	335	124	14

3.1.7 Análise dos Artigos Selecionados

Esta seção apresenta a análise dos artigos selecionados após a fase de extração dos dados. Serão apresentadas, detalhadamente, as respostas baseadas nos resultados encontrados para as questões de pesquisa. Os artigos estão ordenados de forma crescente por ano de publicação.

3.1.7.1 Quais estudos existentes na literatura que desenvolveram um sistema de desagregação de cargas aplicado em um sistema embarcado, computador *single-board* ou plataforma de prototipagem?

Ao fim do mapeamento foram selecionados 14 artigos relevantes para esta dissertação, cujos pontos mais relevantes são detalhados e explicados nesta subseção.

3.1.7.1.1 *Leveraging Smart Meter Data to Recognize Home Appliances*

Weiss et al. (2012) utilizaram o medidor comercializável e certificado *Landis + Gyr* E750[6], que registrava a potência aparente, potência reativa e potência real a uma frequência de 1 Hz para cada uma das três fases. Uma interface de comunicação integrada ao medidor possibilitou a conexão do mesmo, por meio de comandos HTTP, a um *gateway*, que foi um dispositivo embarcado com CPU de 600 MHz, 256 MB de memória RAM e dotado de módulos Ethernet e Wi-Fi, e que ficou responsável por receber as grandezas mensuradas e salvá-las em um banco de dados SQLite3.

O algoritmo desenvolvido, chamado *AppliSense*, automaticamente desagregava as cargas baseado em mudanças nos valores da potência real. Para isso, o algoritmo identificava pontos no tempo em que mudanças significativas entre dois níveis de consumo de energia ocorriam, computava a diferença de potência entre esses níveis e classificava essa variação como um evento de chaveamento de algum dispositivo. Após, comparava cada uma dessas variações de níveis com as informações já conhecidas gravadas no banco de dados para realizar a identificação da carga.

A classificação dos dispositivos foi feita usando o algoritmo kNN aplicado no plano euclidiano dQ/dP. Houve também o desenvolvimento de uma interface amigável, por meio da qual, usando o *smartphone*, o usuário verificava o consumo desagregado por carga e até mesmo cadastrava novas cargas no banco de dados. Ao ser testado, o algoritmo conseguiu detectar corretamente 125 de 144 eventos testados, tendo uma acurácia de 86,8%. A comparabilidade dos resultados torna-se difícil devido a não utilização de um conjunto de dados público conhecido.

3.1.7.1.2 *Inspiring Energy Conservation Through Open Source Metering Hardware and Embedded Real-Time Load Disaggregation*

Makonin et al. (2013) utilizaram como medidor de corrente um TC ligado ao ADC do *Arduino Due*[7]. As amostras de tensão, obtidas a uma frequência de 1 kHz, foram armazenados em um cartão SD a uma frequência máxima de 1 Hz. O algoritmo de desagregação, denominado *µDisagg*, foi o primeiro, segundo os autores, a ser implementado em um sistema embarcado de baixa potência e que funcionava em tempo real. Ele conseguiu desagregar as cargas a partir de

[6] <https://www.landisgyr.com.br/product/e750>
[7] <https://store.arduino.cc/usa/due>

comportamentos complexos, diferentemente do comportamento simples de ligar e desligar um dispositivo.

Para cada carga, foi construído um modelo de conhecimento prévio usando função massa de probabilidade. A classificação dos dispositivos foi feita por meio da análise dos estados, baseando-se nos Modelos Escondidos de Markov (HMM) (MAKONIN, 2014). Para a validação do trabalho, foram escolhidas nove cargas diferentes do *Almanac of Minutely Power data set* (AMPds)[8] para os experimentos. Embora o *μDisagg* ainda estivesse nos estágios iniciais de desenvolvimento, analisando cada uma das nove cargas individualmente, seis tiveram uma acurácia superior a 90%. De forma geral, obteve-se acurácia de 79,7%, após a retirada do dispositivo que concentrou a maioria dos erros, que foi a geladeira, pois sempre alternava entre os estados DESLIGADO e LIGADO durante a fase de refrigeração.

3.1.7.1.3 *A Non-intrusive Load Monitoring System Using an Embedded System for Applications to Unbalanced Residential Distribution Systems*

O sistema de Chang, Wiratha e Chen (2014) utilizou TPs como sensores de tensão e TCs como sensores de corrente. O dispositivo embarcado era o *Portweel* PQ7-C100XL[9], com processador *Intel Atom* E660[10], que também ficou responsável por armazenar os dados. O reconhecimento dos dispositivos foi feito usando MLP. A fim de melhorar o desempenho, os autores propuseram o uso de PSO para otimizar os parâmetros de retro-propagação da RNA.

O algoritmo foi treinado para identificar três cargas reais, que foram: (1) aspirador de pó (900 W), (2) secador de cabelo e (3) aspirador de pó (1200 W). Ao realizarem os experimentos, com poucos testes, inicialmente ligando cada uma das cargas separadamente (1, 2 e 3) e posteriormente as possíveis combinações entre elas (1 + 2, 1 + 3, 2 + 3 e 1 + 2 + 3), obtiveram a acurácia de 100%.

3.1.7.1.4 *Smart Meter Led Probe for Real-Time Appliance Load Monitoring*

O sistema de Barsocchi et al. (2014) detectava as piscadas de dois LEDs do medidor de energia da concessionária. Um LED indicava o consumo de energia ativa e o outro de energia reativa, sendo uma piscada por Wh e VArh, respectivamente, consumidos. Essas informações foram obtidas por meio de dois foto-resistores, um para cada LED, conectados a um Arduino. Para a detecção das cargas foram utilizadas FSMs.

Para uma carga em específico, um micro-ondas, implementaram no *Arduino* uma FSM de cinco estados. O estado inicial era o ocioso, que tinha duas entradas que afetavam o estado da FSM, sendo a primeira entrada a diferença entre a última potência medida e a potência atual

[8] <http://ampds.org>
[9] <https://portwell.com/products/detail.php?CUSTCHAR1=PQ7-C100XL>
[10] <https://ark.intel.com/content/www/us/en/ark/products/52495/intel-atom-processor-e660-512k-cache-1-30-ghz.html>

(ΔP) e a segunda entrada a potência atual (P). Os resultados mostraram uma precisão de 84%, revocação de 95%, medida-F de 89% e medida-G de 87%.

3.1.7.1.5 *High Frequency Non-Intrusive Electric Device Detection and Diagnosis*

O sistema de Jonetzko et al. (2015) utilizou sensores não-invasivos do tipo TC (YDHC SCT-013-005) e TP (transformador 3:1). O *chip Atmel* 90E36A[11] foi utilizado para a leitura dos valores de corrente e tensão, realizando essa leitura em até três fases a uma taxa de amostragem de 7,324 kHz. Para o processamento em tempo-real dos dados foi utilizada a placa de desenvolvimento *Teensy* 3.1[12]. A comunicação entre a placa Teensy e o *chip* da *Atmel* ocorreu usando o protocolo SPI. As informações foram transferidas para um *Raspberry Pi* pela porta USB, e este, por sua vez, transmitiu as informações para um servidor por meio da Wi-Fi.

Utilizando a tensão e corrente, foi aplicada a FFT para a obtenção dos coeficientes de Fourier, analisando assim as harmônicas do sinal. O algoritmo Radix-4 de ponto fixo foi utilizado para o cálculo da FFT, possibilitando a extração dos valores complexos dos coeficientes para auxiliarem na caracterização das cargas. Outra possibilidade abordada pelos autores foi a utilização somente da corrente, sem capturar a mudança de fase entre tensão e corrente. Para isso, foi necessário calcular os descritores de Fourier.

Para o reconhecimento dos dispositivos, os testes foram divididos em duas etapas. A primeira aplicou uma análise de agrupamento e uma MLP, enquanto que a segunda etapa aplicou um kNN. Realizando os testes da primeira etapa, com apenas um dispositivo, que foi um ar-condicionado portátil, tanto o algoritmo de agrupamento quanto a MLP obtiveram as mesmas acurácias: 93% medindo tensão e corrente e 82% medindo somente corrente.

Na segunda etapa do teste, que contou com uma versão protótipo do medidor, com comunicação *ZigBee* e medição da tensão e corrente a uma frequência de 4 kHz, nove diferentes dispositivos foram medidos individualmente para a criação de um padrão de reconhecimento antes de serem combinados. O classificador KNN obteve uma acurácia de 27% para toda a combinação de cargas. Separando os aparelhos de comportamento similar, a acurácia subiu para 91%.

3.1.7.1.6 *SEADS: A Modifiable Platform for Real Time Monitoring of Residential Appliance Energy Consumption*

O sistema de Adabi, Manovi e Mantey (2015) pode ser dividido em quatro camadas. A primeira, de *hardware*, foi capaz de medir tensão e corrente, sendo o medidor desenvolvido pelos autores e chamado de SEADS. Para medição de corrente, utilizaram sensores não-invasivos do tipo TC, cuja saída foi digitalizada pelo ADC a uma frequência de até 65 kHz. O microprocessador

[11] <https://www.microchip.com/wwwproducts/en/ATM90E36A>
[12] <https://www.hobbytronics.co.uk/teensy-v31>

de 32 bits ficou responsável pela aplicação da FFT, pela obtenção das harmônicas e pela execução do classificador de cargas, que era o kNN.

Após a aplicação do algoritmo de classificação, as informações mais importantes foram enviadas para um servidor, que consistiu na segunda camada. Este, por sua vez, armazenou os dados e forneceu-os externamente por meio de uma API, que foi a terceira camada. Além disso, pode também aplicar mais análise de dados, provavelmente complementando o processamento caso o classificador falhasse. A quarta e última camada, que foi de aplicação, consistiu em um aplicativo para visualização dos dados, suporte para sistemas responsivos e automação.

Os testes foram realizados utilizando alguns eletrodomésticos, como um aspirador de pó, um soprador e um micro-ondas. A amostragem foi feita entre 4 e 8 kHz, sendo percebido pelos autores que para uma residência típica, frequências maiores que essas não forneciam um ganho de precisão significativo. Utilizando as 6 primeiras harmônicas, o classificador obteve uma acurácia de 72%. Usando a combinação de todas as 50 harmônicas, essa acurácia subia para 92%

3.1.7.1.7 *Nonintrusive Load Monitoring (NILM) Using an Artificial Neural Network in Embedded System with Low Sampling Rate*

Biansoongnern e Plungklang (2016b) não especificavam qual medidor foi utilizado, informando apenas que as grandezas medidas foram a potência real e reativa, e que a frequência de medição foi de 1 Hz. Por meio do padrão de comunicação RS485, o medidor comunicou-se com um microcontrolador, que foi associado a um módulo RTC, um módulo de cartão SD e um módulo de conexão Wi-Fi. Este último, por sua vez, permitiu que as informações fossem enviadas para a plataforma *ThingSpeak*, na qual o usuário monitorava os dados com facilidade.

O sistema embarcado foi responsável pela detecção de eventos, que ocorreu por meio do monitoramento de seis pontos do sinal, detectando qualquer alteração da potência real. A desagregação também ocorreu no sistema embarcado, utilizando MLP. Para validar o modelo, foram realizados dois experimentos, um simulado e outro real. Foram utilizados os seguintes aparelhos: ar-condicionado, televisor, refrigerador, e panela elétrica. No experimento simulado, a acurácia do classificador foi de 98%, enquanto que no experimento real foi de 95%.

3.1.7.1.8 *Non-Intrusive Appliances Load Monitoring (NILM) for Energy Conservation in Household with Low Sampling Rate*

O sistema de Biansoongnern e Plungklang (2016a) foi basicamente o mesmo sistema citado anteriormente, parecendo ser um protótipo do que viria a ser o sistema, sendo o *hardware* completamente idêntico, sem informar especificamente o medidor utilizado, medindo potência real e reativa a uma taxa de amostragem de 1 Hz e com as mesmas características de conectividade.

O sistema embarcado continuou sendo responsável pela detecção de eventos, mas o monitoramento foi simplificado para três pontos do sinal, que detectavam qualquer alteração

da potência real. O método de reconhecimento de padrões para desagregar as cargas não foi especificado. Em um teste realizado para identificar um ar-condicionado e refrigerador, as acurácias do classificador foram de 90,71% e 89,95%, respectivamente, o que gerou uma acurácia geral de 90,33%.

3.1.7.1.9 *Interpreting Human Activity From Electrical Consumption Data Using Reconfigurable Hardware and Hidden Markov Models*

Ferrández-Pastor et al. (2017) utilizaram o *National Instrument myRIO*[13], com processador ARM Cortex-A9 *dual-core*, para processamento em tempo real, e uma FPGA Xilinx para obtenção dos dados. Os sensores de corrente e de tensão foram acoplados ao quadro elétrico da residência e conectados ao ADC da FPGA, que lia as informações a cada 155 μs. O sistema conseguia capturar os quadros sincronizados com o sinal da tensão quando estava cruzando o zero, ou seja, aplicava um ZCD, o que possibilitava o cálculo de ângulo da fase entre a tensão e corrente, e facilitava a detecção de eventos do tipo LIGA/DESLIGA.

Para a desagregação de cargas foi aplicada a transformada *Wavelet* nas fases de transientes e de estado estável do sinal. Uma tabela de características que representavam os eventos de cada carga foi armazenada, para a posterior aplicação do classificador kNN, que determinava qual dispositivo era ligado. Os autores realizaram um experimento utilizando oito diferente cargas e conseguiram 95% de acurácia.

3.1.7.1.10 *Smart Meter Based on Time Series Modify and Constructive Backpropagation Neural Network*

Adiatmoko et al. (2017) utilizaram sensor de tensão do tipo ZMPT e sensor de corrente do tipo ACS712. O processamento foi realizado por um *Intel Galileo*[14]. Em caso de cargas simultâneas, para o sistema recuperar as informações mais rapidamente sem ter a combinação das cargas, foi utilizado o método de regressão lag-1. Para o reconhecimento de cargas foi utilizada a MLP como classificador. Os parâmetros utilizados para a identificação de cada aparelho foram os sinais, após a regressão lag-1, de *overshoot* / *undershoot* máximo do sinal e o tempo de estabilização do sinal.

A rede foi treinada para detectar as seguintes cargas: secador de cabelos, liquidificador, ferro de passar roupa e panela de arroz. Nessa situação, a rede tinha como características: cinco neurônios, uma camada oculta, duas entradas (sinal de *overshoot* / *undershoot* máximo e tempo de estabilização) e quatro saídas (uma para cada carga). O experimento do modelo foi realizado com apenas 5 testes, conseguindo detectar todas as cargas corretamente, inclusive quando ligadas de forma sobreposta.

[13] <https://www.ni.com/en-us/shop/hardware/products/myrio-student-embedded-device.html>
[14] <https://www.intel.com.br/content/www/br/pt/support/articles/000005912/boards-and-kits/intel-galileo-boards.html>

3.1.7.1.11 *Smart Meter Based on Time Series Modify and Extreme Learning Machine*

O *hardware* utilizado por Arrachman et al. (2017) foi basicamente o mesmo citado anteriormente, com sensor de tensão ZMPT e sensor de corrente ACS712. A diferença estava no processamento, que neste sistema foi realizado por um *Intel Galileo Gen2*[15]. O método de regressão lag-1 também foi usado, porém o reconhecimento de cargas foi feito usando como classificador uma rede neural do tipo ELM. Para o treinamento desta, cujas entradas foram o sinal de *overshoot* / *undershoot* máximo e o tempo de estabilização, foram utilizadas cinco amostras para o estado LIGADO e cinco para o estado DESLIGADO da carga.

Para quatro cargas foram necessárias 40 amostras, gerando uma matriz H 40×40, visto que a estrutura da rede era de 40 neurônios com uma única camada escondida. De posse da matriz H, os valores dos pesos β foram calculados. Os experimentos foram realizados com as seguintes quatro cargas: secador de cabelos, liquidificador, ferro de passar roupa e panela de arroz. Realizaram dois experimentos no total. No primeiro, cujas cargas eram ligadas e desligadas uma por uma, apenas 5 testes foram realizados. No segundo, cujas cargas eram ligadas ou desligadas simultaneamente, foram realizados 40 testes, 10 para cada carga. Em ambos os experimentos, o modelo conseguiu detectar todas as cargas corretamente.

3.1.7.1.12 *Smart Meter Based on Time Series Modify and Neural Network for Online Energy Monitoring*

Syai'in et al. (2018) utilizaram basicamente o mesmo *hardware* de Adiatmoko et al. (2017), usando o sensor de tensão ZMPT, o sensor de corrente ACS712 e o Intel Galileo para processamento. O que diferiu nos dois artigos foi a presença do módulo ESP8266, com Wi-Fi, que permitiu o monitoramento online das cargas no *smartphone*. O aprendizado continuou sendo feito usando a mesma MLP, que possuía cinco neurônios, uma camada oculta, as duas mesmas entradas e quatro saídas, uma para cada carga, que eram: torradeira, liquidificador, forno e panela de arroz.

O experimento foi realizado com apenas 5 testes, conseguindo detectar todas as cargas corretamente. Após a detecção, os dados foram enviados ao *NodeMCU* ESP8266 por meio da comunicação serial. As informações foram enviadas ao banco de dados online do *Google*, o *Firebase*[16], que as disponibilizaram para o aplicativo *Android* desenvolvido pelos autores. O aplicativo, por sua vez, exibia as informações de desagregação na tela do *smartphone* do usuário.

[15] <https://ark.intel.com/content/www/br/pt/ark/products/83137/intel-galileo-gen-2-board.html>
[16] <https://firebase.google.com>

3.1.7.1.13 *Implementation Strategy of Convolution Neural Networks on Field Programmable Gate Arrays for Appliance Classification Using the Voltage and Current (V-I) Trajectory*

Baptista et al. (2018) utilizaram o SoC Xilinx Zynq-7000[17] para realizar a classificação das cargas implementando RNC. Esse SoC tem um processador ARM Cortex-A9 e uma FPGA Artix-7, possibilitando a conexão do sistema de aquisição de dados, que não foi desenvolvido pelos autores. Em contrapartida, foi utilizado o conjunto de dados PLAID, facilitando as comparações dos resultados obtidos. A característica singular para identificação de cada dispositivo utilizada foi a imagem binária, de tamanho 50×50, da trajetória de tensão e corrente.

A arquitetura da RNC utilizada possuía 6 camadas, como mostrado na Tabela 6. A primeira (C1) foi uma camada convolucional com 4 filtros de área 3×3. A segunda camada (P1) foi de *pooling*, com *pool* de área 2×2. A terceira camada (C3) foi convolucional com 6 filtros de área 3×3. A quarta camada (P2) foi de *pooling*, igual a camada P1. A quinta camada foi convolucional, com 18 filtros de área 3×3. A sexta e última camada (F1), foi uma camada densa. O algoritmo de otimização para computar os parâmetros treináveis utilizado ao longo das 200 épocas de treinamento foi o Adam, com taxa de aprendizado de 0,001.

Tabela 6 – Arquitetura da RNC utilizada

Camada	**Filtros**	**Área do Filtro/*Pool***	***Stride***	**Dimensão da Camada**
Entrada	-	-	-	1@50×50
Convolucional (C1)	4	3×3	1	4@48×48
Pooling (P1)	-	2×2	2	4@24×24
Convolucional (C2)	6	3×3	1	6@22×22
Pooling (P2)	-	2×2	2	6@11×11
Convolucional (C3)	18	3×3	1	18@9×9
Densa (F1)	-	-	-	11

Fonte – Adaptado de Baptista et al. (2018)

Os autores testaram 3 diferentes tamanhos de janela de amostragem para cada casa e dispositivo, sendo que o tamanho mais promissor foi o de 333 ms, equivalente a 20 períodos. Ao aplicar a RNC ao PLAID 1 e PLAID2, respectivamente, as medidas-F obtidas foram de 78,16% e 66,01%, como mostrado na Tabela 7. O sistema apresentou uma diferença de quase 10% quando comparado ao trabalho de Barsim, Mauch e Yang (2018), que utilizaram conjuntos de redes neurais com formas de onda corrente e tensão brutas como entrada, em vez da trajetória V-I. Em relação ao trabalho de Baets et al. (2017b), os resultados foram bastante similares, com diferença menor que 1%.

[17] <https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html>

Tabela 7 – Medidas-F obtidas por dispositivo no experimento de Baptista et al. (2018)

Dispositivo Elétrico	**PLAID1**	**PLAID2**
Air Conditioner (AC)	61,20%	57,55%
Compact Fluorescent Lamp (CFL)	90,86%	83,96%
Fan	54,18%	30,04%
Fridge	58,91%	54,32%
Hairdryer	84,70%	68,40%
Heater	71,92%	70,67%
Incandescent Light Bulb (Bulb)	84,83%	61,63%
Laptop	88,01%	71,01%
Microwave	86,98%	76,54%
Vacuum	97,55%	94,94%
Washing Machine (Washer)	80,62%	57,02%
Total	**78,16%**	**66,01%**

Fonte – Adaptado de Baptista et al. (2018)

3.1.7.1.14 *An Online Non-Intrusive Load Monitoring Method Based on Hidden Markov Model*

Huang et al. (2019) utilizaram como processador um *STMicroelectronics* STM32[18], ao qual foram acoplados os sensores de tensão e corrente. A frequência de amostragem desses sinais foi de aproximadamente 4 kHz. Inicialmente, o sinal foi pré-processado para a retirada de ruídos, e para isso, foi usado o algoritmo DB9 modificado, que além de filtrar os ruídos em alta frequência, como o DB9 original, também filtrava ruídos em baixa frequência.

A classificação das cargas foi realizada utilizando uma HMM melhorada, usando o algoritmo *Forward*. Para avaliar o sistema, foram coletados os sinais de seis cargas: lâmpada, televisor, radiador elétrico, geladeira, secador de cabelo e chaleira. Os experimentos mostraram que o algoritmo DB9 consegue filtrar os ruídos de baixa potência de forma eficiente. A acurácia obtida foi de aproximadamente 93%.

3.1.7.2 Quais são os *hardwares* mais utilizados?

Esta questão de pesquisa tem como objetivo identificar, entre os artigos selecionados, os principais *hardwares* utilizados para realizar a desagregação de cargas. Os artigos foram mapeados por *hardware* utilizado, como é possível verificar na Tabela 8.

O gráfico mostrado na Figura 22 permite visualizar que, entre os *hardwares* informados pelos autores, a placa de desenvolvimento *Intel Galileo* foi a mais utilizada, sendo seguida pela plataforma de prototipagem *Arduino*.

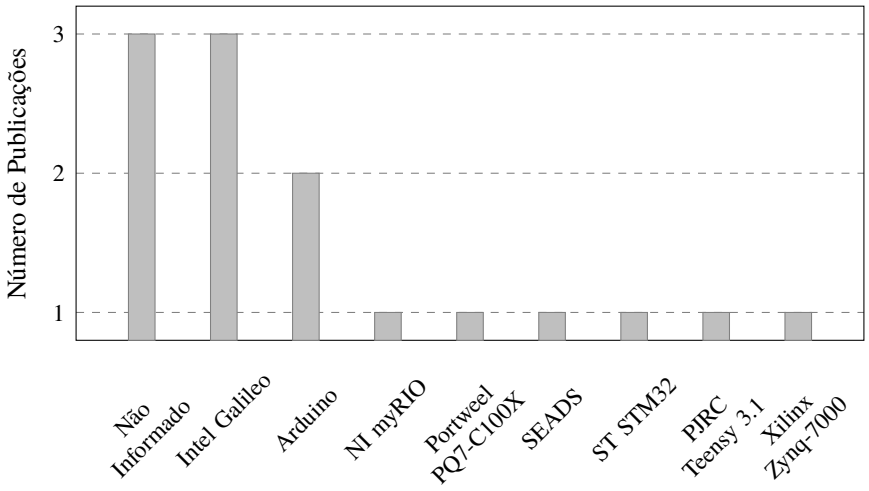
[18] <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>

Tabela 8 – Tabela dos *hardwares* utilizados por artigos selecionados

Artigos Selecionados	Arduino	Intel Galileo	NI myRIO	Portweel PQ7-C100X	SEADS *	ST STM32	Teensy 3.1	Xilinx Zynq-7000	Não Informado
Weiss et al. (2012)									✓
Makonin et al. (2013)	✓								
Chang, Wiratha e Chen (2014)				✓					
Barsocchi et al. (2014)	✓								
Jonetzko et al. (2015)							✓		
Adabi, Manovi e Mantey (2016)					✓				
Biansoongnern e Plungklang (2016b)									✓
Biansoongnern e Plungklang (2016a)									✓
Ferrández-Pastor et al. (2017)			✓						
Adiatmoko et al. (2017)		✓							
Arrachman et al. (2017)		✓							
Syai'in et al. (2018)		✓							
Baptista et al. (2018)								✓	
Huang et al. (2019)						✓			

* Desenvolvido por Adabi, Manovi e Mantey (2016).

Figura 22 – Gráfico dos *hardwares* mais utilizados



3.1.7.3 Quais são as grandezas elétricas comumente mensuradas?

O objetivo desta questão de pesquisa é identificar, entre os artigos selecionados, quais as principais grandezas elétricas que foram mensuradas e utilizadas, direta ou indiretamente, no processo de desagregação de cargas. Os artigos foram mapeados por grandezas mensuradas, como é possível verificar na Tabela 9.

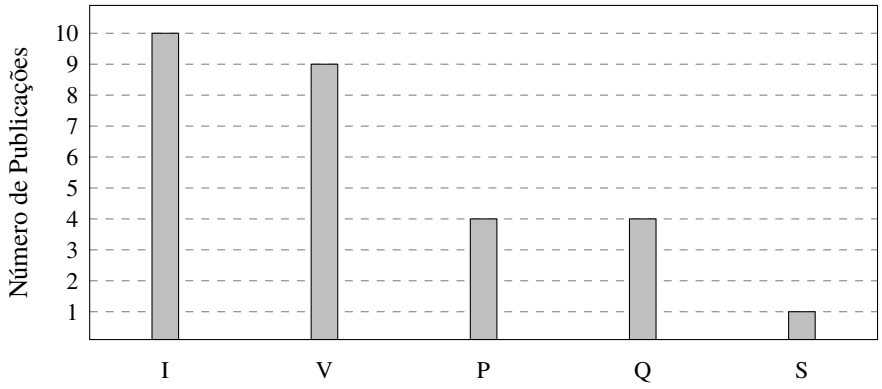
O gráfico mostrado na Figura 23 permite visualizar que a grandeza mais medida foi a corrente (I), seguida pela tensão (V), depois pelas potências ativa (P) e reativa (Q) e, por fim,

pela potência aparente (S).

Tabela 9 – Tabela de grandezas mensuradas por artigos selecionados

Artigos Selecionados	Corrente	Tensão	Potência Ativa	Potência Reativa	Potência Aparente
Weiss et al. (2012)			✓	✓	✓
Makonin et al. (2013)	✓				
Chang, Wiratha e Chen (2014)	✓	✓			
Barsocchi et al. (2014)			✓	✓	
Jonetzko et al. (2015)	✓	✓			
Adabi, Manovi e Mantey (2016)	✓	✓			
Biansoongnern e Plungklang (2016b)			✓	✓	
Biansoongnern e Plungklang (2016a)			✓	✓	
Ferrández-Pastor et al. (2017)	✓	✓			
Adiatmoko et al. (2017)	✓	✓			
Arrachman et al. (2017)	✓	✓			
Syai'in et al. (2018)	✓	✓			
Baptista et al. (2018)	✓	✓			
Huang et al. (2019)	✓	✓			

Figura 23 – Gráfico das grandezas mais mensuradas



3.1.7.4 Quais são as características mais utilizadas para distinção das cargas?

Identificar, entre os artigos selecionados, quais as principais características utilizadas pelo classificador para realizar a classificação dos dispositivos é o objetivo desta questão de pesquisa. Os artigos foram mapeados por características, como é possível verificar na Tabela 10.

O gráfico mostrado na Figura 24 permite visualizar que a característica mais utilizada foi o plano da potência real e aparente (P/Q), seguido da corrente (I), harmônicas (Har), imagens da trajetória binária da tensão e corrente (Bin V/I), potência aparente (S) e tensão e corrente (V/I).

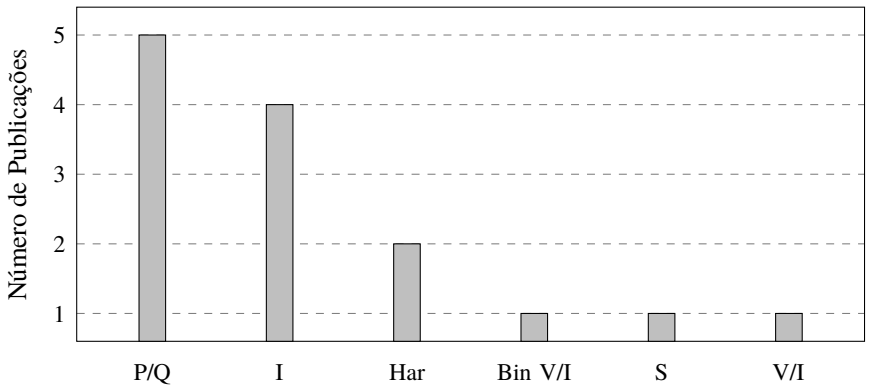
Tabela 10 – Tabela de características por artigos selecionados

Artigos Selecionados	Potências Real e Aparente	Harmônicas	Tensão e Corrente	Corrente	Trajetória Binária V/I	Potência Aparente
Weiss et al. (2012)	✓					
Makonin et al. (2013)				✓		
Chang, Wiratha e Chen (2014)	✓					
Barsocchi et al. (2014)	✓					
Jonetzko et al. (2015)		✓				
Adabi, Manovi e Mantey (2016)		✓				
Biansoongnern e Plungklang (2016b)	✓					
Biansoongnern e Plungklang (2016a)	✓					
Ferrández-Pastor et al. (2017)			✓[a]			
Adiatmoko et al. (2017)				✓[b]		
Arrachman et al. (2017)				✓[b]		
Syai'in et al. (2018)				✓[b]		
Baptista et al. (2018)					✓	
Huang et al. (2019)						✓

[a] A corrente serviu para o cálculo do máximo *undershoot*/*overshoot* dos dados após a regressão lag-1. Aliado ao tempo de estabilização do sinal, essas foram as características utilizadas na distinção.

[b] A tensão e a corrente serviram para o cálculo dos coeficientes de Wavelet, que foi a característica utilizada na distinção.

Figura 24 – Gráfico das características mais utilizadas



3.1.7.5 Quais são os classificadores mais utilizadas no reconhecimento das cargas?

Esta questão de pesquisa tem como objetivo identificar, entre os artigos selecionados, quais os principais classificadores utilizados para realizar a desagregação de cargas no sistema embarcado. Os artigos foram mapeados por classificadores, como é possível verificar na Tabela 11.

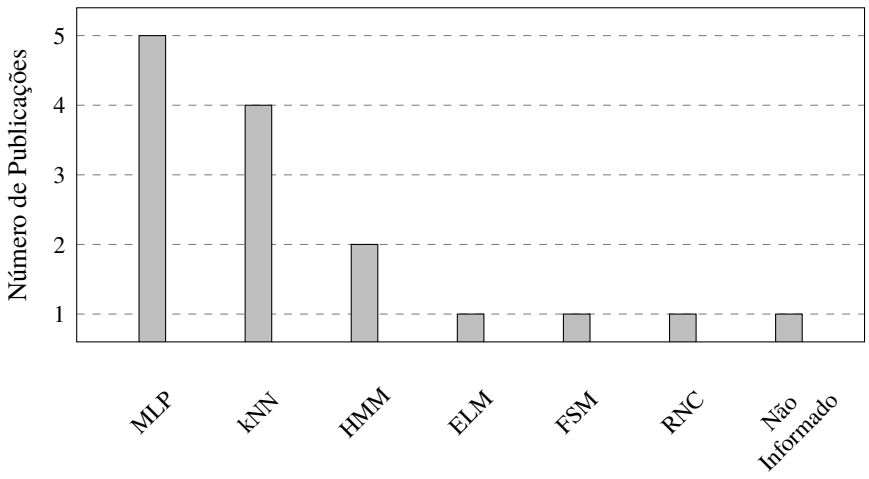
O gráfico mostrado na Figura 25 permite visualizar que a maioria dos artigos utilizaram Redes Neurais Artificiais, especificamente MLPs, como classificador. Somente Biansoongnern e Plungklang (2016b) não informaram qual classificador foi utilizado. Como o artigo explica o que

parece ser uma prévia do que viria a ser o sistema, explicado na Seção 3.1.7.1.7, provavelmente esse classificador era uma MLP.

Tabela 11 – Tabela de técnicas de aprendizado por artigos selecionados

Artigos Selecionados	MLP	kNN	HMM	ELM	FSM	RNC	Não Informado
Weiss et al. (2012)		✓					
Makonin et al. (2013)			✓				
Chang, Wiratha e Chen (2014)	✓						
Barsocchi et al. (2014)					✓		
Jonetzko et al. (2015)	✓	✓					
Adabi, Manovi e Mantey (2016)		✓					
Biansoongnern e Plungklang (2016b)	✓						
Biansoongnern e Plungklang (2016a)							✓
Ferrández-Pastor et al. (2017)		✓					
Adiatmoko et al. (2017)	✓						
Arrachman et al. (2017)				✓			
Syai'in et al. (2018)	✓						
Baptista et al. (2018)						✓	
Huang et al. (2019)			✓				

Figura 25 – Gráfico das técnicas de aprendizado mais utilizadas



3.1.7.6 Quais métricas mais utilizadas para avaliação dos classificadores?

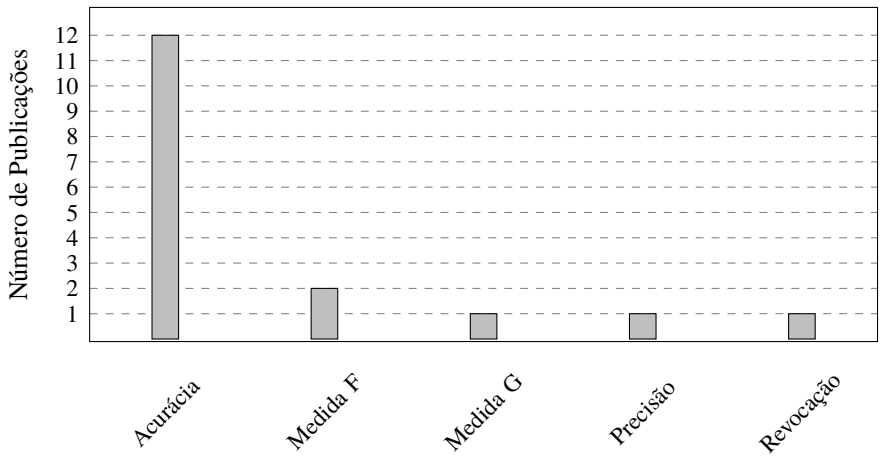
O objetivo desta questão de pesquisa é identificar, entre os artigos selecionados, quais as principais métricas utilizadas pelos autores, a fim de facilitar a comparação direta e justa entre os resultados obtidos. Os artigos foram mapeados por métricas utilizadas, como é possível verificar na Tabela 12.

O gráfico mostrado na Figura 26 permite visualizar que a maioria dos artigos utilizaram a acurácia como métrica.

Tabela 12 – Tabela de métricas por artigos selecionados

Artigos Selecionados	Acurácia	Precisão	Medida F	Medida G	Revocação
Weiss et al. (2012)	✓				
Makonin et al. (2013)	✓				
Chang, Wiratha e Chen (2014)	✓				
Barsocchi et al. (2014)		✓	✓	✓	✓
Jonetzko et al. (2015)	✓				
Adabi, Manovi e Mantey (2016)	✓				
Biansoongnern e Plungklang (2016b)	✓				
Biansoongnern e Plungklang (2016a)	✓				
Ferrández-Pastor et al. (2017)	✓				
Adiatmoko et al. (2017)	✓				
Arrachman et al. (2017)	✓				
Syai'in et al. (2018)	✓				
Baptista et al. (2018)			✓		
Huang et al. (2019)	✓				

Figura 26 – Gráfico das métricas mais utilizadas



3.1.7.7 Quais são os *datasets* mais utilizadas para avaliação dos classificadores?

Identificar, entre os artigos selecionados, quais são os *datasets* mais utilizados é o objetivo desta questão de pesquisa. Essa informação auxiliará, futuramente, na escolha de um conjunto de dados em comum, possibilitando que os resultados dos trabalhos possam ser comparáveis.

Apenas dois artigos utilizaram *datasets*, que foram os de Makonin et al. (2013) e Baptista et al. (2018), utilizando o AMPds e o PLAID, respectivamente. Os trabalhos apresentados nos demais artigos testavam os classificadores com dados obtidos pelos próprios autores, sem disponibilizá-los.

3.1.7.8 As informações são transmitidas pelo dispositivo para o meio externo?

Esta questão de pesquisa tem como objetivo identificar, entre os artigos selecionados, se houve uma preocupação dos autores em transmitir as informações remotamente, por exemplo, informando as grandezas medidas, quais as cargas que estão ligadas no momento, ou quais cargas consumiram mais energia em um determinado tempo, dentre outras.

Os artigos foram mapeados por tecnologia de comunicação utilizada, sendo que somente os artigos de Weiss et al. (2012), Jonetzko et al. (2015), Adabi, Manovi e Mantey (2015), Biansoongnern e Plungklang (2016b), Biansoongnern e Plungklang (2016a), Syai'in et al. (2018) conseguem externar as informações. Todos eles utilizaram a tecnologia Wi-Fi, excetuando Adabi, Manovi e Mantey (2015), que não informaram a tecnologia.

3.2 Outros Trabalhos

Analisando as referências bibliográficas de alguns artigos encontrados com o auxílio do Mapeamento Sistemático da Literatura (MSL), outros trabalhos importantes foram elencados, como o de Gao et al. (2015) e o de Baets et al. (2017b). Embora não tenham sido selecionados no MSL por não terem aplicado a desagregação em um sistema embarcado, são relevantes para este trabalho, pois também utilizam o conjunto de dados PLAID, tornando os resultados comparáveis.

3.2.1 *A Feasibility Study of Automated Plug-Load Identification From High-Frequency Measurements*

O trabalho de Gao et al. (2015) tem como objetivo comparar a performance de alguns classificadores em algumas características (*features*) populares para o problema de desagregação de carga. Além disso, explora a relação entre a acurácia do classificador e a frequência de amostragem do sinal, para um melhor entendimento das limitações de hardware ao implementar os medidores. As 6 características utilizadas são:

- **Corrente:** Medições diretas de corrente de um período, como mostrado na Figura 27;
- **Potência Real e Reativa:** Os cálculos de potência real e reativa são obtidos por meio das amostras de tensão e corrente, explicado em 2.1.2 e 2.1.3. Com essas potências pode-se gerar gráficos no plano P-Q, como o mostrado na Figura 28;
- **Harmônicas:** Quando a carga é não-linear, existem componentes harmônicos no sinal, que são múltiplos da frequência fundamental. Alguns exemplos dessa característica são mostrados na Figura 29;
- **Tensão e Corrente Quantizados:** Os sinais de tensão e corrente do estado estável são normalizados em amplitude e posteriormente quantizados, como mostrado na Figura 30.

Figura 27 – Forma de onda da corrente de quatro diferentes dispositivos elétricos

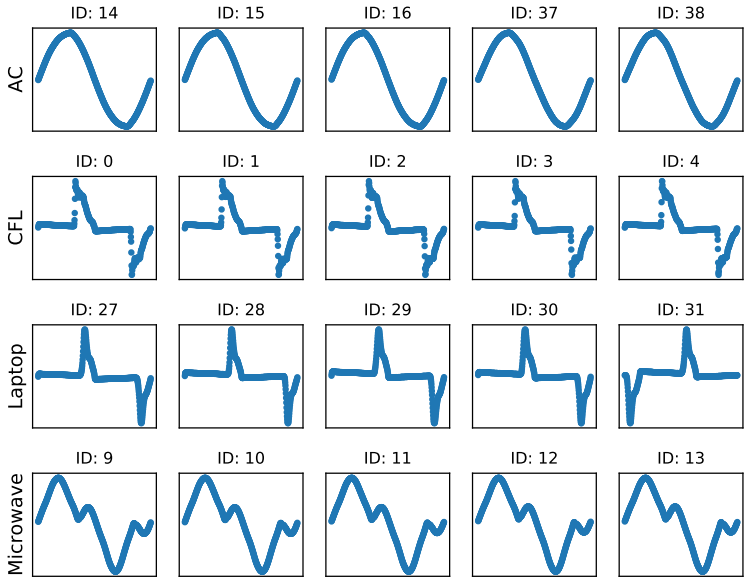


Figura 28 – Plano P-Q de quatro diferentes dispositivos elétricos

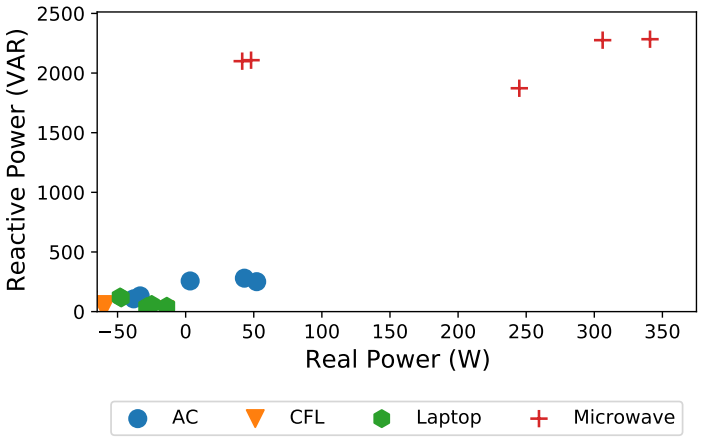
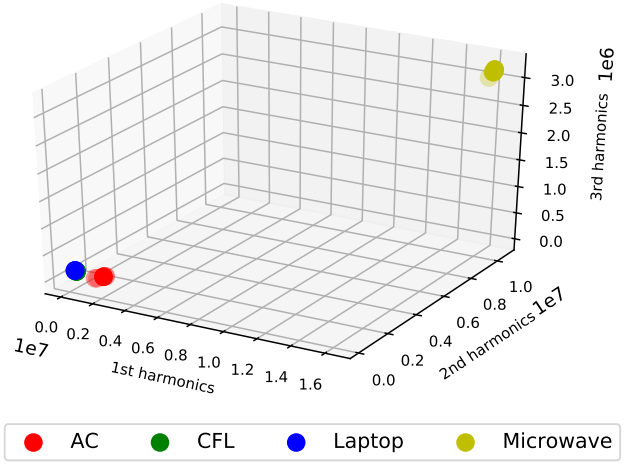
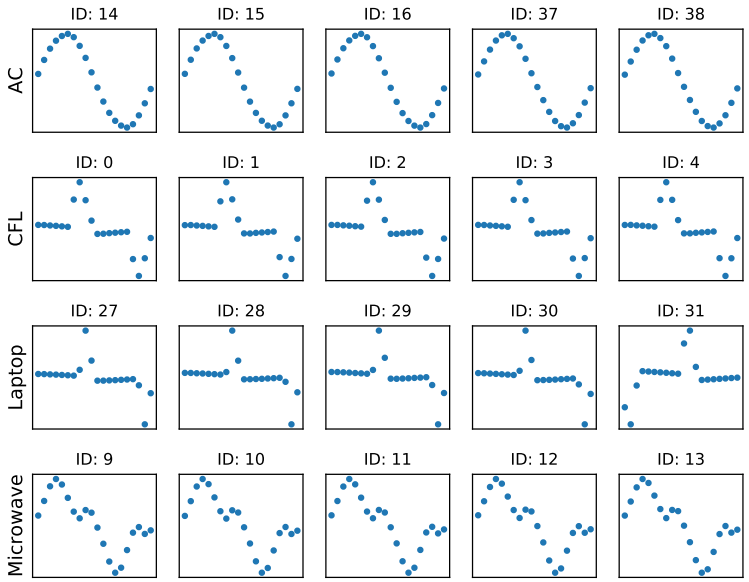


Figura 29 – Harmônicas de quatro diferentes dispositivos elétricos



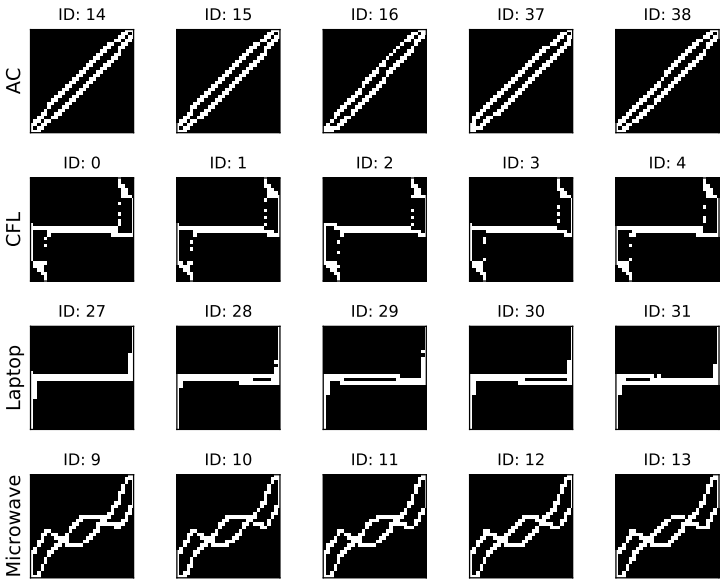
Como resultado, os vetores de tensão e corrente são concatenados, formando um vetor de característica única;

Figura 30 – Corrente quantizada de quatro diferentes dispositivos elétricos



- **Imagem Binária da Trajetória V-I:** A imagem da trajetória da tensão (V) e corrente (I) é normalizada na amplitude e convertida em uma imagem binária 16×16, na qual um pixel ativo é representado por 1 e os demais por 0, como mostrado na Figura 31;

Figura 31 – Trajetória V-I de quatro diferentes dispositivos elétricos



- ***Principal Component Analysis* (PCA):** Para reduzir a dimensão de algumas dessas características acima, foi adotada a Análise de Componentes Principais na corrente, tensão e corrente quantizados e também na trajetória V-I binária;

- **Combinação:** Por fim, quatro características (potência real e reativa, harmônicas, corrente e tensão quantizadas e trajetória V-I binária) foram agrupadas em um único vetor.

Para avaliar as características citadas anteriormente, Gao et al. (2015) usaram nove diferentes algoritmos de aprendizado supervisionado (classificadores), cujos nomes e parâmetros usados são mostrados na Tabela 13. Os testes foram conduzidos utilizando os pacotes *scikit-learn*[19] (PEDREGOSA et al., 2011) e *matplotlib*[20] (HUNTER, 2007) do *Python*. O *Jupyter Notebook*[21] (KLUYVER et al., 2016) que eles compartilharam está disponível em <https://github.com/jingkungao/PLAID/blob/master/ApplianceIdentificationCodeforGlobalSip.ipynb>.

Tabela 13 – Classificadores e parâmetros utilizados

Classificadores	**Parâmetros**
k-Nearest-Neighbors (kNN)	n_neighbors = 1
Gaussian Naive Bayes (GNB)	-
Logistic Regression Classifier (LGC)	C = 1e5
Support Vector Machine (SVM)	kernel = 'rbf', gamma = 0.7, C = 1.0
Linear Discriminant Analysis (LDA)	solver = 'lsqr', shrinkage = 'auto'
Quadratic Discriminant Analysis (QDA)	-
Decision Tree (DT)	max_depth = 10
Random Forest (RF)	max_depth = 10, n_estimators = 20
Adaptive Boosting (AB)	-

Para discutir a viabilidade dos testes no mundo real e obter resultados mais convincentes, Gao et al. (2015) acharam pertinente utilizar o método de validação cruzada *leave-one-out*. Desta forma, as medições de 1 residência foram utilizadas como conjunto de teste enquanto que as medições das outras 54 casas foram utilizadas como conjunto de treinamento, sendo esse processo repetido para cada uma das 55 casas.

Esse método de validação, apesar de investigar de forma mais profunda a variação do modelo em relação aos dados de entrada, possui um alto custo computacional, não sendo recomendado em situações de muitos dados. A métrica utilizada para comparação entre os classificadores foi a acurácia, que é a razão entre as predições corretas e o total de predições, como mostrado na Tabela 2. Para a exibição dos dados, foram eliminados os classificadores que apresentaram precisão inferior a 50%, sendo os resultados obtidos no experimento mostrados na Tabela 14.

A maior acurácia, de 86,03%, foi alcançada utilizando a combinação de características. Analisando as médias das acurácias das características individuais, a que obteve melhor resultado foi a imagem da trajetória V-I binária. O uso da PCA para redução da dimensionalidade mostrou desempenho aceitável apenas quando aplicada a imagem da trajetória V-I. Nas situações citadas

[19] <https://scikit-learn.org>
[20] <https://matplotlib.org/>
[21] <https://jupyter.org>

anteriormente, analisando também as médias de acurácia dos classificadores, o classificador que obteve o melhor desempenho foi o *Random Forest*.

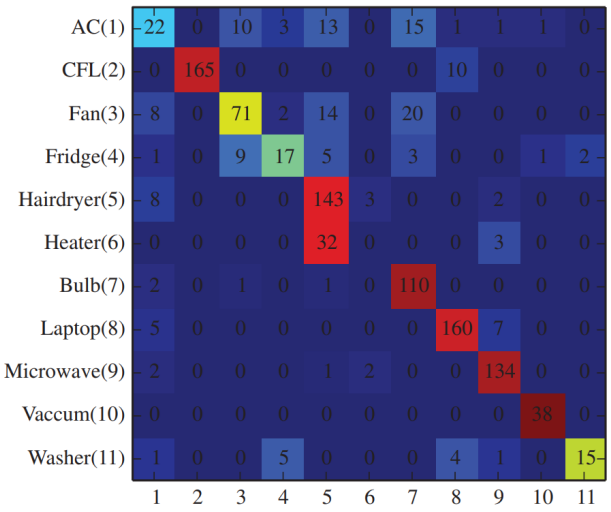
Tabela 14 – Acurácia dos classificadores nos experimentos

	kNN	**GNB**	**LGC**	**DT**	**RF**	**MÉDIA**
Corrente	75,98%	61,73%	69,83%	70,67%	76,26%	**70,89%**
Potência Real/Reativa	55,40%	27,19%	29,14%	49,07%	51,58%	**42,48%**
Harmônicas	45,25%	18,72%	30,45%	42,18%	49,63%	**37,25%**
V-I Quantizados	60,06%	57,17%	60,06%	73,09%	80,63%	**66,20%**
Imagem V-I Binária	78,96%	51,96%	74,49%	76,07%	81,75%	**72,65%**
Corrente (PCA)	44,13%	52,14%	46,37%	48,14%	45,07%	**47,17%**
V-I Quantizados (PCA)	24,30%	18,06%	11,08%	25,98%	27,28%	**21,34%**
Imagem V-I (PCA)	69,93%	60,34%	64,53%	70,67%	77,65%	**68,62%**
Combinação	62,10%	59,22%	49,44%	74,49%	86,03%	**66,26%**
MÉDIA	**57,35%**	**45,17%**	**48,38%**	**58,93%**	**63,99%**	

Fonte – Adaptado de Gao et al. (2015)

Como as imagens da trajetória V-I são normalizadas, os autores perceberam que as informações de amplitude são completamente irrelevantes, sugerindo que as entradas talvez não precisem ser calibradas. Além disso, usar essa característica economiza o esforço de alinhar sinais de correntes de diferentes instâncias para iniciar a partir de mesmo ponto de passagem por zero, que nem sempre é fácil de ser alcançado automaticamente.

Figura 32 – Matriz de confusão para o *Random Forest* utilizando as imagens da trajetória V-I



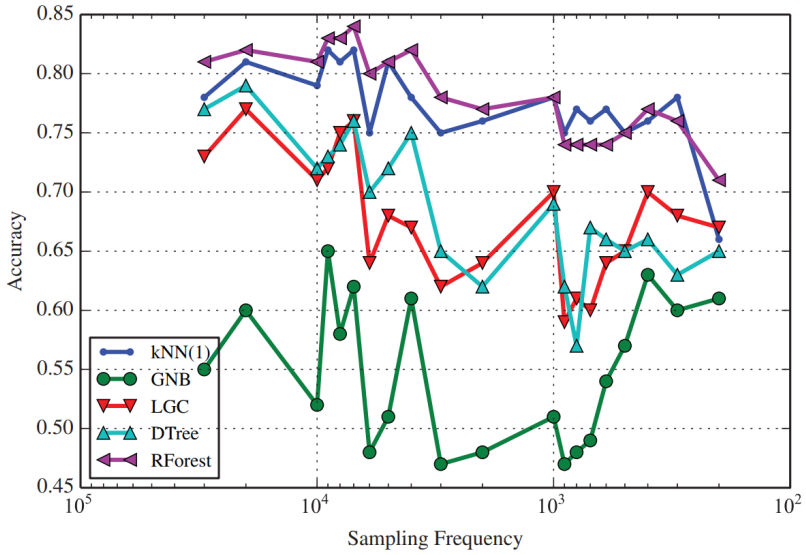
Fonte – Gao et al. (2015)

Eles perceberam que apesar da melhor acurácia ter sido obtida a partir da combinação de recursos, a imagem V-I binária era mais fácil de processar e apresentava menor custo computacional, fazendo com que essa característica fosse a escolhida para uma melhor análise. Foram explorados os resultados da classificação para cada tipo de dispositivo individualmente e apresentada a matriz de confusão, que pode ser vista na Figura 32.

Cada linha da matriz representa a contagem em valores reais de instâncias de um dispositivo, enquanto que a coluna representa essa contagem de valores previstos pelo classificador. Pode ser observado na matriz que o maior erro obtido foi o de 32 aquecedores classificados erroneamente como secadores de cabelo, sendo um erro compreensível pela natureza resistiva desses dispositivos.

Para investigar a viabilidade de implementar essas soluções em plataformas embarcadas, Gao et al. (2015) conduziram um novo experimento para analisar o efeito da frequência de amostragem na precisão do classificador. Pelas vantagens da imagem V-I binária mencionadas anteriormente, eles concentraram o novo experimento nessa característica. As taxas de amostragem variaram na escala logarítmica de 30 KHz a 200 Hz.

Figura 33 – Acurácia em função da taxa de amostragem



Fonte – Gao et al. (2015)

Os resultados obtidos, mostrados na Figura 33, mostram que taxas superiores a 4kHz apresentaram acurácias relativamente constantes dos classificadores, sendo *Random Forest* e kNN os que obtiveram desempenho relativamente superior ao resto. Desta forma, os autores mostraram que a possibilidade de usar um hardware com taxa de amostragem maior que 4 kHz é possível utilizando a imagem V-I binária como característica de distinção. Além disso, atentaram-se que como apenas os dados do estado estável são usados, uma curta duração, inferior a 1 minuto, de ativação do dispositivo, pode ser suficiente para identificá-lo.

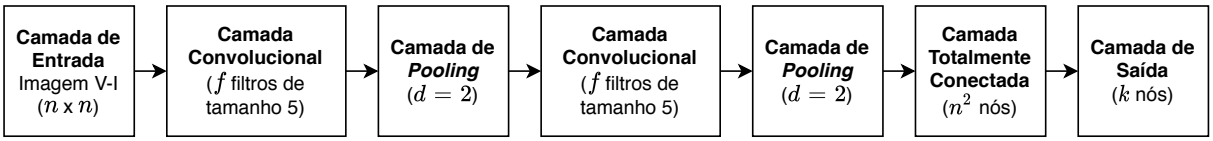
3.2.2 *Appliance Classification Using VI Trajectories and Convolutional Neural Networks*

O trabalho de Baets et al. (2017b), por sua vez, utilizou, além do PLAID, o *Worldwide Household and Industry Transient Energy Dataset* (WHITED) (KAHL et al., 2016). Este último

inclui amostras de tensão e corrente obtidas em 6 diferentes regiões da Europa e a uma frequência de amostragem de 44 KHz para 110 diferentes dispositivos elétricos, que podem ser agrupados em 47 diferentes classes. Para cada dispositivo, foram amostrados 10 eventos de inicialização dos mesmos, o que garantiu um total de 1100 amostras.

Como as amostras do WHITED não possuem a anotação do local correspondente, Baets et al. (2017b) criaram uma anotação de forma artificial e randômica, atribuindo aleatoriamente cada dispositivo de cada classe a uma casa. O número total de casas foi definido como 9, que é o número máximo de dispositivos por classe para esse conjunto de dados. Como consequência, classes que tinham somente amostras de um dispositivo foram eliminadas, o que reduziu o número de classes para 22.

Figura 34 – Estrutura da RNC



Para o experimento, os autores utilizaram como característica somente as imagens binárias da trajetória V-I. O classificador utilizado foi a RNC, cuja arquitetura pode ser vista na Figura 34. Foram utilizadas imagens de dimensão 50×50 (n = 50), 50 filtros (f = 50) e 11 nós de saída (k = 11) para o PLAID e 22 nós de saída (k = 22) para o WHITED, que correspondem aos respectivos números de classes desses *datasets*. Os testes foram conduzidos utilizando os pacotes *theano*[22] (AL-RFOU et al., 2016) e *matplotlib* do *Python*. O *Jupyter Notebook* compartilhado pelos autores está disponível em <https://github.com/LeenDB/ApplianceClassificationUsingVITrajectoriesAndCNN>.

A função de custo utilizada foi a entropia cruzada (do Inglês, *cross-entropy*), com o método de gradiente descendente para minimização. O conjunto de dados foi dividido seguindo o mesmo critério de Baets et al. (2017b), ou seja, o método de validação cruzada também foi o *leave-one-out*. Como métrica de desempenho, baseado no artigo de Makonin e Popowich (2014), os autores utilizaram a medida-F.

Como ambos os *datasets* são desbalanceados, a medida-F foi calculada para cada classe (ou seja, cada tipo de aparelho) separadamente. Os verdadeiros positivos (TP_a), falsos positivos (FP_a) e falsos negativos (FN_a) foram calculados para cada teste por tipo de aparelho a. Para combinar os resultados dos valores da medida-F (Equação 3.1) por classe em um único número, que seria o resultado geral obtido pelo classificador, foi utilizada a macro-média da medida-F,

[22] <http://deeplearning.net/software/theano>

mostrada na Equação 3.2.

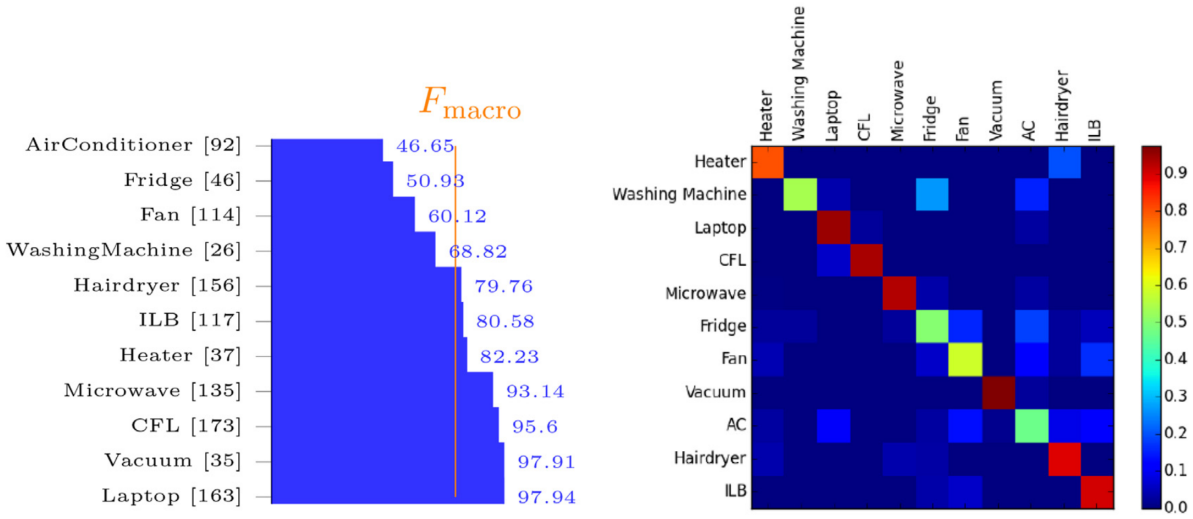
$$F_a = 2 \cdot \frac{\text{precisão}_a \cdot \text{revocação}_a}{\text{precisão}_a + \text{revocação}_a} \tag{3.1}$$

$$F_{macro} = \frac{1}{A} \sum_a F_a \tag{3.2}$$

Em que A é o número total de classes diferentes (tipos diferentes de aparelhos).

Para o PLAID, conjunto de dados alvo de nosso interesse, visto que o WHITED não foi encontrado disponível para ser baixado, as medidas-F por dispositivo podem ser vistas na Figura 35a. Além disso, também é exibida a macro-média da medida-F, que foi 77,60%. Os autores perceberam que para todos os dispositivos, excetuando a máquina de lavar, o ventilador, a geladeira e o ar-condicionado, a medida-F foi maior que a macro-média.

Figura 35 – Medidas-F e matriz de confusão do experimento de (BAETS et al., 2017b)



(a) Medida-F por classes (b) Matriz de confusão

Fonte – Baets et al. (2017b)

Ao investigarem a matriz de confusão, vista na Figura 35b, perceberam também que há uma confusão entre esses aparelhos, logo apresentaram medidas-F mais baixas que os demais. Esse fenômeno pode ser explicado pelo fato desses dispositivos serem eletricamente parecidos, pois todos utilizam motor elétrico.

3.3 Tabela Comparativa

O comparativo entre todos os artigos selecionados, levando em consideração todas as perguntas definidas no mapeamento, pode ser observado na Tabela 15.

Tabela 15 – Tabela comparativa dos artigos selecionados

Trabalhos Selecionados	*Hardware* Utilizado	Grandezas Mensuradas	Transmissão de Dados	*Feature*	Classificador	*Dataset*	Métricas (%)				
							A[a]	P[b]	R[c]	F[d]	G[e]
Weiss et al. (2012)	Não Informado	P, Q e S	Wi-Fi	ΔP/ΔQ	kNN		97,0				
Makonin et al. (2013)	Arduino Due	I		I	HMM	AMPds	79,7				
Chang, Wiratha e Chen (2014)	Portweel PQ7-C100X	V e I		ΔP/ΔQ	MLP		100,0				
Barsocchi et al. (2014)	Arduino Uno	P e Q		P e Q	FSM			84,0	95,0	89,0	87,0
Jonetzko et al. (2015)	PJRC Teensy 3.1	V e I	Wi-Fi	Harmônicas	Agrupamento MLP kNN		93,0 82,0 27,0				
Adabi, Manovi e Mantey (2016)	SEADS [f]	V e I		Harmônicas	kNN		92,0				
Biansoongnern e Plungklang (2016b)	Não Informado	P e Q	Wi-Fi	ΔP/ΔQ	MLP		95,0				
Biansoongnern e Plungklang (2016a)	Não Informado	P e Q	Wi-Fi	ΔP/ΔQ	Não Informado		90,3				
Ferrández-Pastor et al. (2017)	NI myRIO	V e I		V e I [g]	kNN		95,0				
Adiatmoko et al. (2017)	Intel Galileo	V e I		I [h]	MLP		100,0				
Arrachman et al. (2017)	Intel Galileo Gen2	V e I		I [h]	ELM		100,0				
Baets et al. (2017b)		V e I		Bin V/I	RNC	PLAID1				77,6	
Syai'in et al. (2018)	Intel Galileo	V e I	Wi-Fi	I [h]	MLP		100,0				
Baptista et al. (2018)	Xilinx Zynq-7000	V e I		Bin V/I	RNC	PLAID1 PLAID2				78,2 66,0	
Huang et al. (2019)	ST STM32	V e I		S	HMM		93,0				

[a] Acurácia [b] Precisão [c] Revocação [d] Medida F [e] Medida G (Índice de Fowlkes-Mallows)
[f] Desenvolvido pelos próprios autores do artigo, ou seja, por Adabi, Manovi e Mantey (2016)
[g] A corrente serviu para o cálculo do máximo *undershoot* / *overshoot* dos dados após a regressão lag-1. Aliado ao tempo de estabilização do sinal, essas foram as características utilizadas na distinção
[h] A tensão e a corrente serviram para o cálculo dos coeficientes de Wavelet, que foi a característica utilizada na distinção

4

Escolha do Classificador e da Característica de Distinção

Este capítulo tem por objetivo esclarecer as razões das escolhas das imagens binárias da trajetória V-I e da Rede Neural Convolucional como característica de distinção e como classificador, respectivamente. Para isso, alguns experimentos são realizados, sendo a metodologia e os resultados apresentados.

4.1 Metodologia

Para a escolha da melhor característica de distinção de cargas e do melhor classificador, refizemos o experimento de Gao et al. (2015), visto na Seção 3.2.1. No experimento original, cada um dos nove diferentes classificadores tiveram suas acurácias comparadas para cada uma das nove diferentes características de distinção de cargas. Ao reproduzirmos o experimento, acrescentamos um novo classificador, que foi a rede neural MLP.

A Rede Neural Convolucional não foi acrescentada na reprodução deste experimento porque era necessário possuir uma dimensão de entrada fixa e uma arquitetura de rede previamente definida para os dados que seriam trabalhados, o que não era possível, visto que as características de distinção abordadas possuíam dimensões variadas.

A fim de testarmos a performance da RNC para o mesmo conjunto de dados, um segundo experimento foi realizado, baseado em Baets et al. (2017b) e Baptista et al. (2018), com o objetivo de termos um meio de comparar a performance dos classificadores do experimento anterior com a da RNC para uma característica de distinção em específico.

Como era necessário obter uma representação do estado estável para cada dispositivo elétrico, em ambos experimentos foram extraídas as 10.000 últimas amostras de cada instância do conjunto de dados. Essas amostras de tensão e corrente foram obtidas a uma frequência de amostragem (f_s) de 30 kHz.

Sabendo que a frequência da rede elétrica do local onde as amostras foram tiradas (f_r) é de 60 Hz e que o número de amostras por período é igual a razão entre a frequência de amostragem e a frequência da rede, temos um total de 500 amostras por período de cada dispositivo elétrico amostrado.

Os experimentos foram conduzidos na plataforma *Google Colab*[1], utilizando a linguagem de programação *Python*, cuja versão do pacote instalado era a 3.6.9. No primeiro experimento, para fins de reprodutibilidade, foram utilizados os classificadores e métricas provenientes do pacote *scikit-learn*. Para o segundo experimento, aproveitando a disponibilidade de GPU[2] na plataforma do *Google*, utilizamos a computação paralela proveniente da integração entre a biblioteca *TensorFlow* (cuja versão do pacote instalado era a 2.3.0) e a API CUDA para avaliarmos a RNC. Os parâmetros utilizados para cada classificador serão explicados posteriormente, na Seção 4.2.

Para garantirmos a reprodutibilidade de ambos os experimentos, foi utilizada uma semente fixa, que sempre garante a geração de uma mesma sequência de números pseudo-aleatórios. Foi usado também o pacote *tensorflow-determinism*, versão 0.3.0, que não é aplicado por padrão e serve, como o próprio nome já diz, para aumentar o determinismo durante as reproduções da RNC, fazendo com que os resultados mantenham-se similares. Desta forma, cada experimento foi repetido apenas algumas dezenas de vezes, visto que os resultados eram bastantes similares.

Além disso, para a característica trajetória V-I, foram utilizadas imagens binárias de dimensão igual a 50×50 *pixels*. Apesar do conjunto de dados utilizado pelos autores citados anteriormente ser o mesmo, em nossos experimentos, usamos não somente o PLAID1 (Seção 2.4) e o PLAID2, mas também o PLAID1+2, que foi como nomeamos a junção das amostras do PLAID1 e PLAID2.

4.2 Experimentos

Nesta seção são apresentadas informações relevantes e os resultados de ambos os experimentos, fazendo também uma comparação com os resultados dos experimentos primários, explicados na Seção 3.2.

4.2.1 Análise da Imagem Binária da Trajetória V-I Como Característica de Distinção das Cargas

Os parâmetros dos classificadores foram mantidos idênticos ao do experimento original, como observado na Tabela 16. Quanto aos parâmetros das características, apenas a dimensão das imagens das trajetórias V-I foi alterada. No experimento de Gao et al. (2015), a dimensão

[1] <https://colab.research.google.com>

[2] Para isso, deve ser alterado o tipo do ambiente de execução na plataforma, selecionado o acelerador de *hardware* GPU, como explicado no tutorial disponível em: <https://colab.research.google.com/notebooks/gpu.ipynb>.

dessas imagens era de 16×16, sendo modificada para 50×50 neste novo experimento. Essa decisão foi tomada para analisar se o aumento da dimensão provocaria alguma mudança significativa na acurácia e para tornar os resultados compatíveis com o segundo experimento, que utiliza a dimensão 50×50. O método de validação cruzada utilizado na avaliação também foi o *leave-one-out*[3].

Utilizando apenas os dados do PLAID1, os dez classificadores foram avaliados. Seguindo o roteiro de Gao et al. (2015), foram removidos os classificadores cujas médias de acurácia foram mais baixas. Dessa forma, as acurácias dos seis classificadores que obtiveram maior média podem ser observadas na Tabela 17.

As médias de cada classificador e de cada característica foram destacadas em negrito, sendo as inferiores referentes a cada coluna (classificador) e as da direita referentes a cada linha (característica). Além disso, a melhor acurácia por linha foi sublinhada.

Tabela 16 – Classificadores e parâmetros utilizados

Classificadores	**Parâmetros**
k-Nearest-Neighbors (kNN)	n_neighbors = 1
Gaussian Naive Bayes (GNB)	-
Logistic Regression Classifier (LGC)	C = 1e5
Support Vector Machine (SVM)	kernel = 'rbf', gamma = 0.7, C = 1.0
Linear Discriminant Analysis (LDA)	solver = 'lsqr', shrinkage = 'auto'
Quadratic Discriminant Analysis (QDA)	-
Decision Tree (DT)	max_depth = 10
Random Forest (RF)	max_depth = 10, n_estimators = 20
Adaptive Boosting (AB)	-
Multilayer Perceptron (MLP)	-

Tabela 17 – Acurácias dos classificadores e características testadas usando o PLAID1

	RF	**DT**	**kNN**	**GNB**	**MLP**	**LGC**	**MÉDIA**
Corrente	77,56%	70,67%	76,07%	61,73%	71,23%	66,20%	**70,58%**
Potência Real/Reativa	70,02%	67,88%	71,88%	57,36%	54,93%	47,11%	**61,53%**
Harmônicas	72,63%	66,11%	71,97%	51,30%	48,79%	48,04%	**59,81%**
V-I Quantizado	80,82%	70,86%	71,60%	71,42%	66,29%	56,24%	**69,54%**
Trajetória V-I	77,19%	68,90%	79,98%	64,06%	78,49%	77,28%	**74,32%**
Corrente (PCA)	72,81%	61,64%	22,25%	66,11%	12,48%	45,62%	**46,82%**
V-I Quantizado (PCA)	49,63%	47,77%	47,11%	51,21%	51,68%	43,67%	**48,51%**
Trajetória V-I (PCA)	74,77%	71,42%	76,63%	63,50%	61,73%	64,15%	**68,70%**
Combinação	85,85%	74,58%	72,07%	64,90%	60,80%	46,74%	**67,49%**
MÉDIA	**73,47%**	**66,65%**	**65,51%**	**61,29%**	**56,27%**	**55,01%**	

Comparado com o resultado do experimento original, visto na Tabela 14, percebemos que a melhor acurácia, de 85,85%, continuou sendo obtida usando o classificador *Random Forest*

[3] As medições de 1 residência foram utilizadas como conjunto de teste enquanto que as medições das outras *n* casas foram utilizadas como conjunto de treinamento, sendo esse processo repetido para cada uma das *n* casas.

com a combinação de características[4]. Analisando as médias, podemos notar que dentre os classificadores avaliados, o que obteve melhor média foi o *Random Forest* (73,47%). De forma análoga, a característica que obteve melhor média (74,32%) foi a trajetória V-I, corroborando com o que foi observado por Gao et al. (2015). Além disso, ao realizar a comparação das tabelas, foi percebido que o aumento da dimensão das imagens não aumentou a acurácia de forma significativa.

Tabela 18 – Acurácias dos classificadores e características testadas usando o PLAID2

	RF	**kNN**	**DT**	**LDA**	**GNB**	**MLP**	**MÉDIA**
Corrente	48,96%	49,37%	44,78%	48,26%	38,25%	47,15%	**46,13%**
Potência Real/Reativa	54,94%	52,02%	53,41%	40,19%	42,28%	40,61%	**47,24%**
Harmônicas	51,60%	55,35%	39,08%	40,47%	45,20%	36,16%	**44,65%**
V-I Quantizado	56,88%	41,31%	45,90%	58,55%	57,02%	36,58%	**49,37%**
Trajetória V-I	61,47%	59,94%	48,96%	51,60%	40,33%	57,02%	**53,22%**
Corrente (PCA)	50,07%	15,30%	47,98%	29,62%	47,57%	9,87%	**33,40%**
V-I Quantizado (PCA)	31,57%	32,13%	31,85%	38,39%	36,02%	38,25%	**34,70%**
Trajetória V-I (PCA)	52,99%	51,46%	51,32%	37,69%	37,69%	50,63%	**46,96%**
Combinação	68,57%	55,77%	45,76%	60,78%	55,91%	49,65%	**56,07%**
MÉDIA	**53,01%**	**45,85%**	**45,45%**	**45,06%**	**44,48%**	**40,66%**	

Repetindo o mesmo teste para o PLAID2, ao compararmos os resultados, vistos na Tabela 18, percebemos que a melhor acurácia obtida, de 68,57%, persistiu sendo do classificador *Random Forest* com a característica combinada. Fazendo a análise das médias, o melhor classificador persistiu sendo o *Random Forest* (53,01%). Já na análise das características, a que obteve maior média de acurácia foi a combinação de características (56,07%). Das características singulares, a maior média continuou sendo da trajetória V-I (53,22%).

Por fim, aplicando o mesmo teste para a junção do PLAID1 ao PLAID2, ao qual nomeamos PLAID1+2, obtivemos os resultados exibidos na Tabela 19. A melhor acurácia obtida, de 79,31%, mais uma vez continuou sendo do classificador *Random Forest* com a característica combinada. Analisando as médias, o melhor classificador permaneceu sendo o *Random Forest* (64,72%). Observando as médias das características, percebemos que com o aumento de amostras do conjunto de dados, a trajetória V-I foi a que obteve maior média de acurácia (69,39%).

4.2.2 Análise da RNC aplicada a imagem binária da trajetória V-I

Para verificarmos a acurácia da RNC, reproduzimos o experimento de Baets et al. (2017b) utilizando a mesma arquitetura, mostrada na Tabela 20, e também imagens da trajetória V-I de dimensão 50×50. Visto que os autores não especificaram os parâmetros de compilação, utilizamos o otimizador *Adam*, com a entropia cruzada como função de perda e 30 épocas de treinamento.

[4] Como explicado na Seção 3.2, a combinação de características engloba a potência real/reativa, harmônicas, V-I quantizado e trajetória V-I.

Tabela 19 – Acurácias dos classificadores e características testadas usando o PLAID1+2

	RF	**DT**	**kNN**	**GNB**	**MLP**	**LDA**	**MÉDIA**
Corrente	66,93%	59,23%	65,53%	48,58%	65,70%	51,92%	**59,65%**
Potência Real/Reativa	58,28%	56,27%	57,56%	51,42%	50,86%	29,67%	**50,68%**
Harmônicas	61,46%	51,92%	62,19%	43,17%	45,18%	49,69%	**52,27%**
V-I Quantizado	73,56%	68,10%	58,67%	61,18%	57,78%	55,83%	**62,52%**
Trajetória V-I	73,06%	65,42%	73,84%	58,78%	73,68%	71,56%	**69,39%**
Corrente (PCA)	67,71%	61,29%	16,95%	57,39%	16,95%	58,39%	**46,45%**
V-I Quantizado (PCA)	43,84%	43,39%	37,65%	42,39%	44,45%	37,53%	**41,54%**
Trajetória V-I (PCA)	58,34%	57,50%	61,46%	53,99%	59,34%	35,30%	**54,32%**
Combinação	79,31%	68,38%	62,52%	58,45%	58,23%	74,34%	**66,87%**
MÉDIA	**64,72%**	**59,06%**	**55,15%**	**52,82%**	**52,46%**	**51,58%**	

Tabela 20 – Arquitetura da RNC usada por Baets et al. (2017b)

Camada	**Filtros**	**Área do Filtro/*Pool***	**Dimensão da Camada**
Entrada	-	-	1@50×50
Convolucional (C1)	50	5×5	50@46×46
Pooling (P1)	-	2×2	50@23×23
Convolucional (C2)	50	5×5	50@19×19
Pooling (P2)	-	2×2	50@9×9
Densa (F1)	-	-	11

Usando o mesmo método de validação cruzada, *leave-one-out*, comum tanto no trabalho de Gao et al. (2015) quanto no de Baets et al. (2017b) e de Baptista et al. (2018), já explicado anteriormente na Seção 3.2, a acurácia obtida foi de 81,28%, como pode ser observado na Tabela 21. Ao compararmos com as acurácias da Tabela 17, percebemos que a da RNC foi maior que a de todos os outros classificadores para essa característica de distinção.

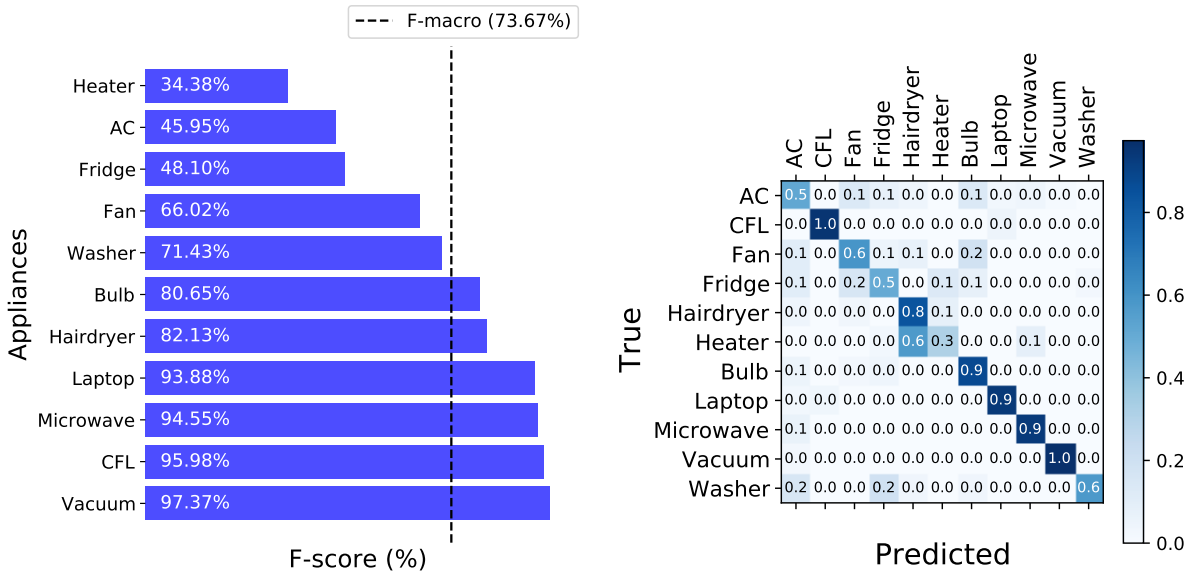
Tabela 21 – Relatório de classificação da RNC usando o PLAID1: Parte I

Classes	**Precisão**	**Revocação**	**Medida-F**	**Suporte**[a]
Air Conditioner (00)	41,46%	51,52%	45,95%	66
Compact Fluorescent Lamp (01)	96,53%	95,43%	95,98%	175
Fan (02)	74,73%	59,13%	66,02%	115
Fridge (03)	46,34%	50,00%	48,10%	38
Hairdryer (04)	80,37%	83,97%	82,13%	156
Heater (05)	37,93%	31,43%	34,38%	35
Incandescent Light Bulb (06)	74,63%	87,72%	80,65%	114
Laptop (07)	94,15%	93,60%	93,88%	172
Microwave (08)	95,59%	93,53%	94,55%	139
Vacuum (09)	97,37%	97,37%	97,37%	38
Washing Machine (10)	93,75%	57,69%	71,43%	26
Acurácia			81,28%	1074
Média Macro[b]	75,71%	72,85%	73,67%	1074
Média Ponderada[c]	81,91%	81,28%	81,29%	1074

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

As medidas-F, para cada classe treinada, podem ser observadas na Figura 36a. O aquecedor (*heater*), o ar-condicionado (AC), a geladeira (*fridge*), o ventilador (*fan*) e a lavadora de roupas (*washer*) ficaram abaixo da macro-média da medida-F, que foi de 73,67%. Na matriz de confusão, exibida na Figura 36b, percebemos que a confusão mais acentuada foi do aquecedor (*heater*) com o secador de cabelos (*hairdryer*), que possuem características elétricas similares, pois ambos são cargas resistivas[5].

Figura 36 – Medidas-F e matriz de confusão usando o PLAID1: Parte I



(a) Medida-F por classes

(b) Matriz de confusão

Tabela 22 – Relatório de classificação da RNC usando o PLAID2: Parte I

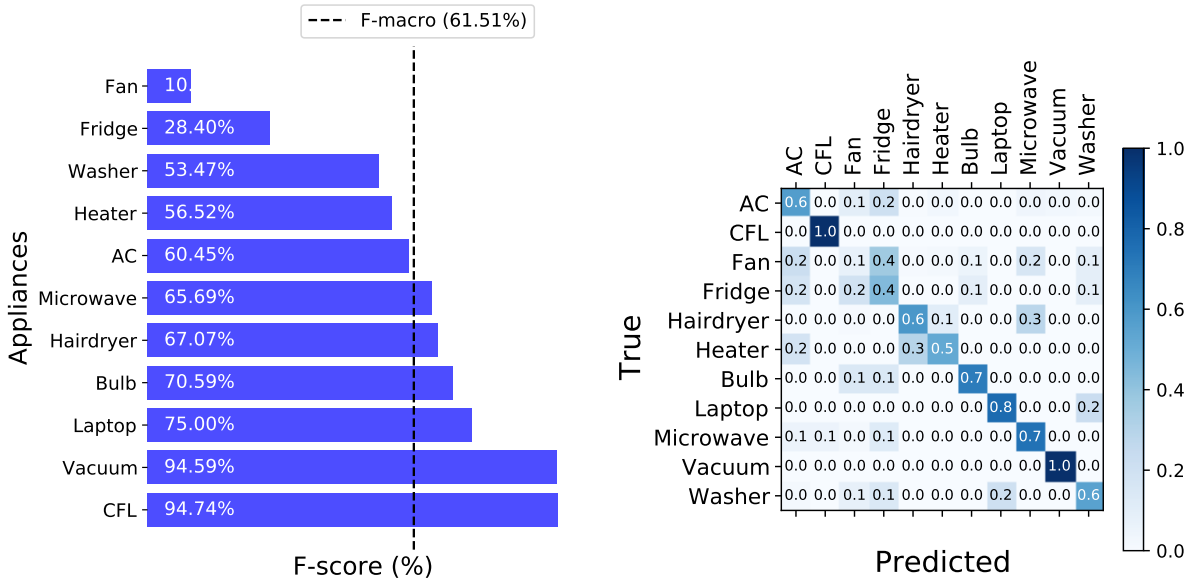
Classes	Precisão	Revocação	Medida-F	Suporte[a]
Air Conditioner (00)	64,29%	57,04%	60,45%	142
Compact Fluorescent Lamp (01)	90,00%	100,00%	94,74%	45
Fan (02)	16,28%	7,37%	10,14%	95
Fridge (03)	20,91%	44,23%	28,40%	52
Hairdryer (04)	76,39%	59,78%	67,07%	92
Heater (05)	61,90%	52,00%	56,52%	50
Incandescent Light Bulb (06)	70,59%	70,59%	70,59%	34
Laptop (07)	72,97%	77,14%	75,00%	35
Microwave (08)	58,77%	74,44%	65,69%	90
Vacuum (09)	89,74%	100,00%	94,59%	35
Washing Machine (10)	51,92%	55,10%	53,47%	49
Acurácia			58,00%	719
Média Macro[b]	61,25%	63,43%	61,51%	719
Média Ponderada[c]	58,23%	58,00%	57,23%	719

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

[5] Cargas em que as ondas de tensão e corrente estão em fase, ou seja, o fator de potência é unitário e toda a potência gerada pela fonte é absorvida pela carga, como explicado na Seção 2.1.5

De forma análoga, repetindo o experimento para o PLAID2, observamos na Tabela 22 que a acurácia da RNC foi de 58%, figurando-se em terceiro lugar, aproximadamente 6% abaixo da melhor acurácia obtida pelos classificadores avaliados no experimento anterior, cujos resultados estão disponíveis na Tabela 18. Uma possível explicação para isso é o fato do PLAID2 possuir menos amostras que os demais, além de ser ainda mais desbalanceado.

Figura 37 – Medidas-F e matriz de confusão usando o PLAID2: Parte I



(a) Medida-F por classes

(b) Matriz de confusão

As medidas-F, para cada classe treinada, podem ser observadas na Figura 37a. O ventilador (*fan*), a geladeira (*fridge*), a lavadora de roupas (*washer*), o aquecedor (*heater*) e o ar-condicionado (*AC*), assim como no experimento realizado para o PLAID1, ficaram abaixo da macro-média da medida-F, que foi de 61,51%.

Na matriz de confusão, exibida na Figura 37b, percebemos que as maiores confusões foram do ventilador com a geladeira, com o ar-condicionado e com o micro-ondas (microwave), e do secador de cabelo (*hairdryer*) com o micro-ondas (*microwave*) e com o aquecedor.

Embora o ventilador, a geladeira e o ar-condicionado possuam funcionamentos parecidos, assim como o secador de cabelo e o aquecedor possuem, o fato de nem todas as residências do PLAID2 possuírem pelo menos um representante de cada classe fez com que o classificador não tivesse dados suficientes para apresentar bons resultados, contribuindo para essas confusões.

Refazendo o experimento para o PLAID1+2, podemos observar na Tabela 23 que a acurácia obtida pela RNC foi de 76,69%, maior que a de todos os classificadores testados no primeiro experimento, com resultados disponíveis na Tabela 19.

As medidas-F, para cada classe treinada, podem ser observadas na Figura 38a. A geladeira (*fridge*), ar-condicionado (AC), o ventilador (*fan*), a lavadora de roupas (*washer*) e o aquecedor

(*heater*) ficaram abaixo da macro-média da medida-F, que foi de 73,91%.

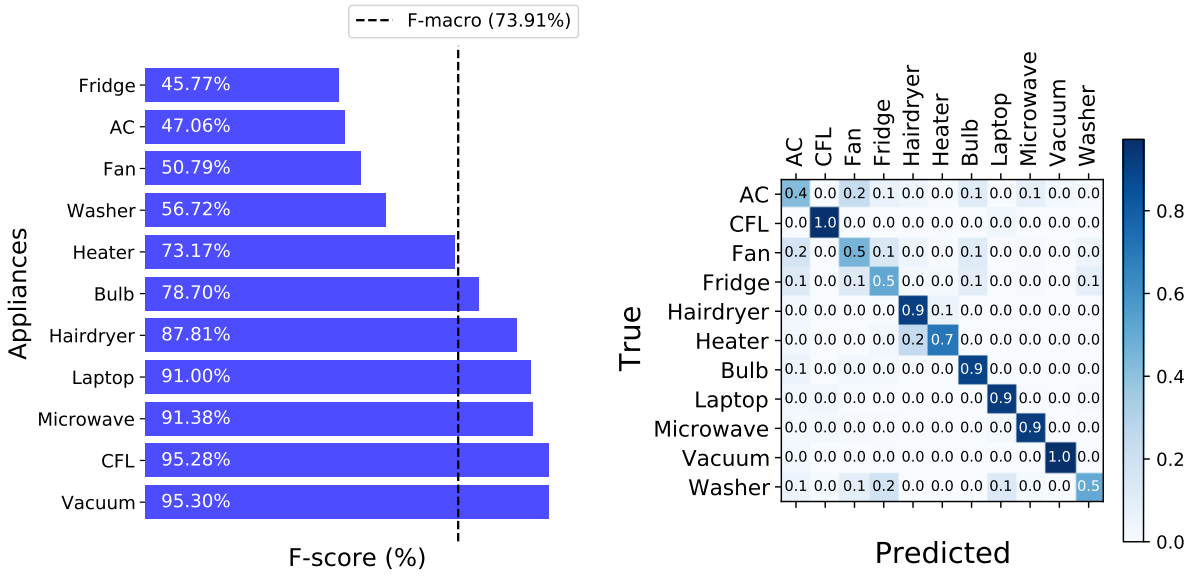
Tabela 23 – Relatório de classificação da RNC usando o PLAID1+2: Parte I

Classes	Precisão	Revocação	Medida-F	Suporte[a]
Air Conditioner (00)	53,01%	42,31%	47,06%	208
Compact Fluorescent Lamp (01)	94,22%	96,36%	95,28%	220
Fan (02)	57,14%	45,71%	50,79%	210
Fridge (03)	41,44%	51,11%	45,77%	90
Hairdryer (04)	84,39%	91,53%	87,81%	248
Heater (05)	75,95%	70,59%	73,17%	85
Incandescent Light Bulb (06)	70,00%	89,86%	78,70%	148
Laptop (07)	89,30%	92,75%	91,00%	207
Microwave (08)	90,21%	92,58%	91,38%	229
Vacuum (09)	93,42%	97,26%	95,30%	73
Washing Machine (10)	64,41%	50,67%	56,72%	75
Acurácia			76,69%	1793
Média Macro[b]	73,95%	74,61%	73,91%	1793
Média Ponderada[c]	75,86%	76,69%	75,94%	1793

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

Na matriz de confusão, exibida na Figura 38b, percebemos que a maior confusão foi do ar-condicionado (AC) com o ventilador (*fan*). Ambos os dispositivos possuem funcionamento parecido, pois o ar-condicionado quando não está com o compressor ligado, ou seja, quando não está gelando ou esquentando, está apenas com a evaporadora funcionando, cujo funcionamento é similar ao de um ventilador.

Figura 38 – Medidas-F e matriz de confusão usando o PLAID1+2: Parte I



(a) Medida-F por classes

(b) Matriz de confusão

Já o experimento de Baptista et al. (2018), que utilizou o PLAID1 e PLAID2, foi

reproduzido utilizando a arquitetura original, mostrada na Tabela 6, e também imagens da trajetória V-I de dimensão 50×50 como característica de distinção. Assim como no experimento original, utilizaremos o otimizador *Adam*, com taxa de aprendizagem padrão de 0,001, a entropia cruzada como função de perda e 200 épocas de treinamento.

Usando o mesmo método de validação cruzada, *leave-one-out*, a acurácia obtida foi de 80,45% para o PLAID1, como pode ser observado na Tabela 24. Ao compararmos com as acurácias da Tabela 17, da reprodução do experimento de Gao et al. (2015), percebemos que a acurácia da RNC continua sendo maior que a de todos os outros classificadores para essa característica, mesmo utilizando outra arquitetura.

Tabela 24 – Relatório de classificação da RNC usando o PLAID1: Parte II

Classes	**Precisão**	**Revocação**	**Medida-F**	**Suporte**[a]
Air Conditioner (00)	36,76%	37,88%	37,31%	66
Compact Fluorescent Lamp (01)	97,09%	95,43%	96,25%	175
Fan (02)	79,27%	56,52%	65,99%	115
Fridge (03)	33,33%	44,74%	38,20%	38
Hairdryer (04)	77,71%	82,69%	80,12%	156
Heater (05)	40,00%	28,57%	33,33%	35
Incandescent Light Bulb (06)	74,45%	89,47%	81,27%	114
Laptop (07)	93,10%	94,19%	93,64%	172
Microwave (08)	95,74%	97,12%	96,43%	139
Vacuum (09)	94,87%	97,37%	96,10%	38
Washing Machine (10)	78,95%	57,69%	66,67%	26
Acurácia			80,45%	1074
Média Macro[b]	72,84%	71,06%	71,39%	1074
Média Ponderada[c]	80,81%	80,45%	80,24%	1074

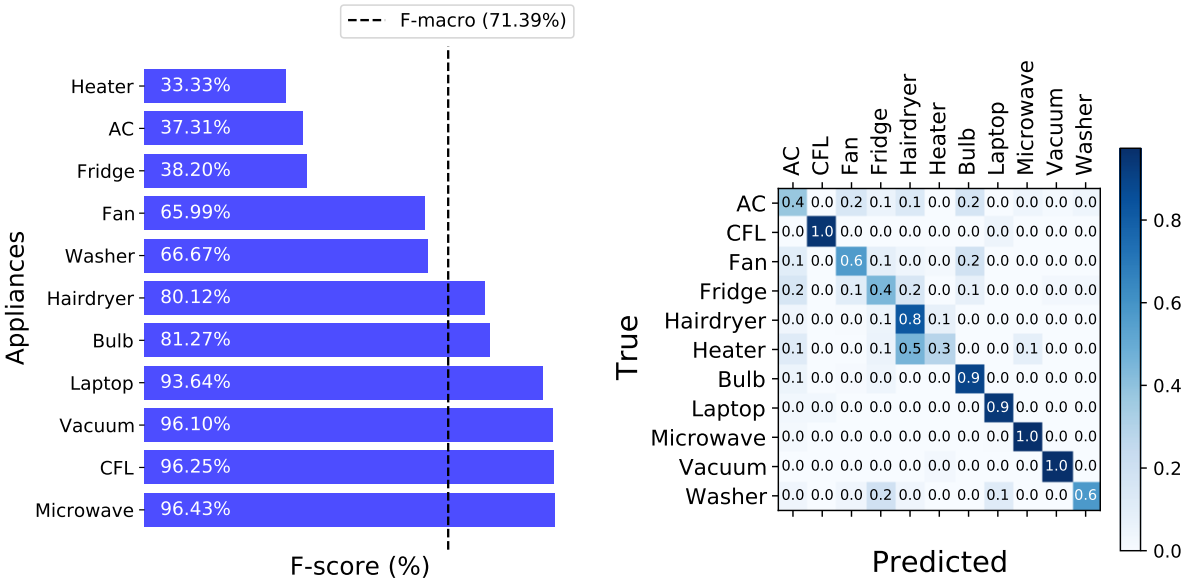
[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

Enquanto a macro-média da medida-F obtida na reprodução do experimento de Baets et al. (2017b) foi de 73,67%, a de Baptista et al. (2018), ao refazermos o experimento, foi de 71,39%, apresentando uma variação percentual de aproximadamente 4% entre os resultados.

As medidas-F, para cada classe treinada, podem ser observadas na Figura 39a. Assim como na reprodução do experimento de Baets et al. (2017b), o aquecedor (*heater*), o ar-condicionado (AC), a geladeira (*fridge*), o ventilador (*fan*) e a lavadora de roupas (*washer*) ficaram abaixo da macro-média da medida-F. O mesmo comportamento pode ser visto na matriz de confusão, exibida na Figura 39b, na qual também percebemos que a confusão mais acentuada foi do aquecedor (*heater*) com o secador de cabelos (*hairdryer*).

Repetindo o experimento para o PLAID2, observamos na Tabela 25 que a acurácia da RNC foi de 56,88%, figurando-se em quarto lugar (aproximadamente 8% abaixo da melhor acurácia) entre os classificadores avaliados na reprodução do experimento de Gao et al. (2015), cujos resultados estão disponíveis na Tabela 18.

Figura 39 – Medidas-F e matriz de confusão usando o PLAID1: Parte II



(a) Medida-F por classes

(b) Matriz de confusão

Tabela 25 – Relatório de classificação da RNC usando o PLAID2: Parte II

Classes	Precisão	Revocação	Medida-F	Suporte[a]
Air Conditioner (00)	59,12%	57,04%	58,06%	142
Compact Fluorescent Lamp (01)	91,84%	100,00%	95,74%	45
Fan (02)	17,86%	10,53%	13,25%	95
Fridge (03)	23,08%	40,38%	29,37%	52
Hairdryer (04)	75,00%	55,43%	63,75%	92
Heater (05)	48,98%	48,00%	48,48%	50
Incandescent Light Bulb (06)	61,11%	64,71%	62,86%	34
Laptop (07)	71,43%	71,43%	71,43%	35
Microwave (08)	61,68%	73,33%	67,01%	90
Vacuum (09)	92,11%	100,00%	95,89%	35
Washing Machine (10)	54,72%	59,18%	56,86%	49
Acurácia			56,88%	719
Média Macro[b]	59,72%	61,82%	60,25%	719
Média Ponderada[c]	56,76%	56,88%	56,24%	719

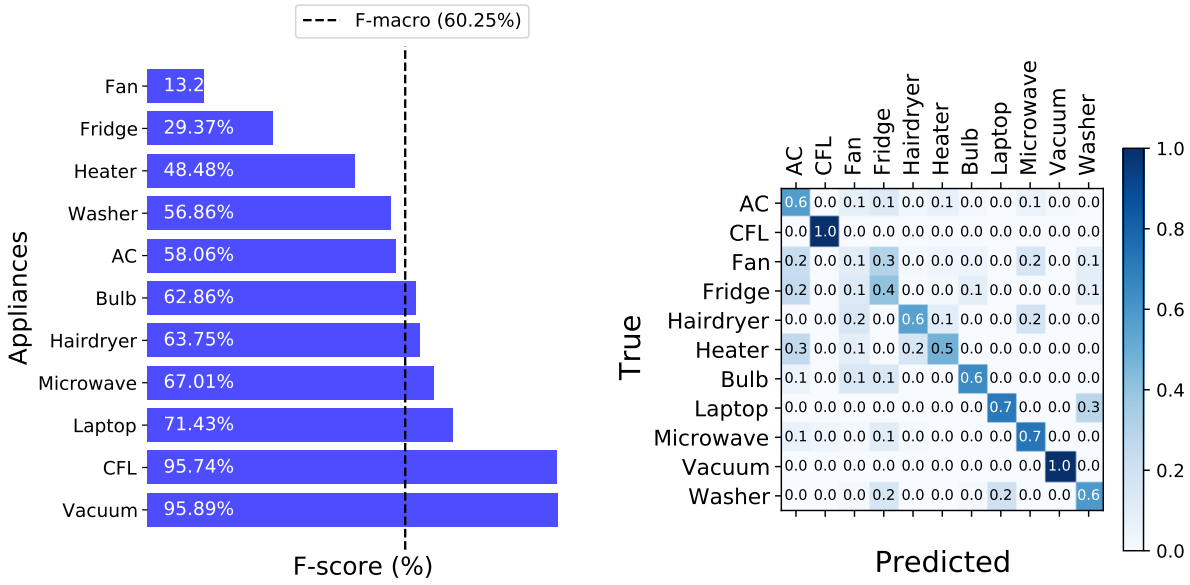
[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

A macro-média da medida-F obtida na reprodução do experimento de Baets et al. (2017b) foi de 61,51%, enquanto que a de Baptista et al. (2018), ao refazermos o experimento, foi de 60,25%, apresentando uma variação percentual de aproximadamente 2%.

As medidas-F para cada classe treinada podem ser observadas na Figura 40a. O ventilador (*fan*), a geladeira (*fridge*), o aquecedor (*heater*), a lavadora de roupas (*washer*) e o ar-condicionado (AC) ficaram abaixo da macro-média da medida-F. Na matriz de confusão, exibida na Figura 40b, percebemos que as maiores confusões foram do ventilador com a geladeira, com o ar-condicionado

e com o micro-ondas (microwave).

Figura 40 – Medidas-F e matriz de confusão usando o PLAID2: Parte II



(a) Medida-F por classes

(b) Matriz de confusão

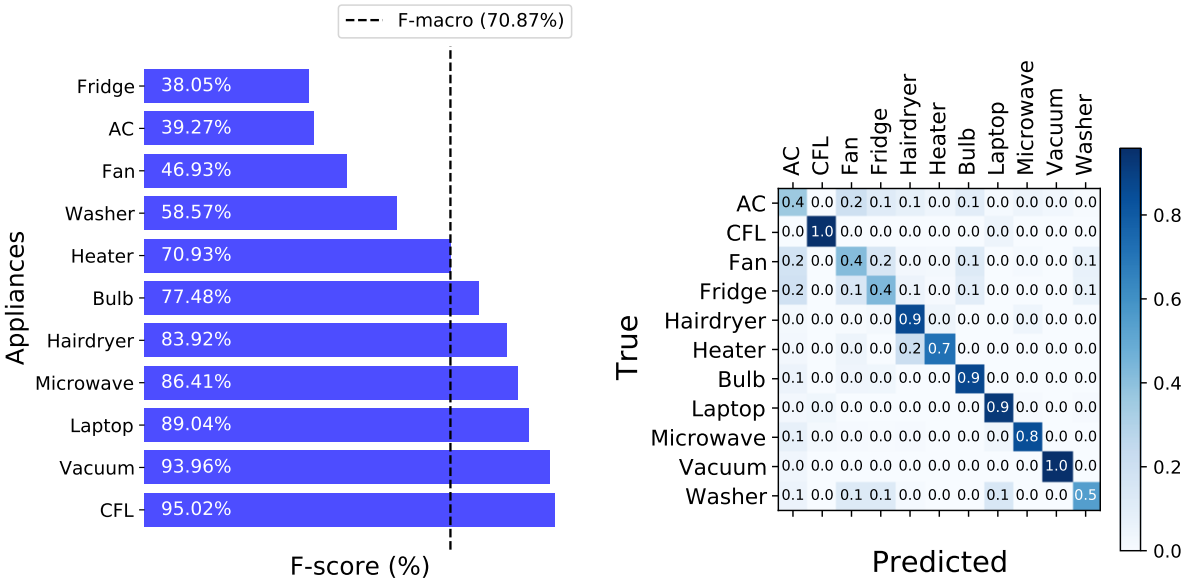
Por fim, refazendo o experimento para o PLAID1+2, podemos observar na Tabela 26 que a acurácia obtida pela RNC foi de 70,87%, maior que a de todos os classificadores testados no experimento da Seção 4.2.1, com resultados disponíveis na Tabela 19. A macro-média da medida-F obtida na reprodução do experimento de Baets et al. (2017b) foi de 73,91%, já a de Baptista et al. (2018), ao refazermos o experimento, foi de 70,87%, apresentando uma variação percentual de aproximadamente 4%.

Tabela 26 – Relatório de classificação da RNC usando o PLAID1+2: Parte II

Classes	Precisão	Revocação	Medida-F	Suporte[a]
Air Conditioner (00)	43,10%	36,06%	39,27%	208
Compact Fluorescent Lamp (01)	94,59%	95,45%	95,02%	220
Fan (02)	53,33%	41,90%	46,93%	210
Fridge (03)	33,91%	43,33%	38,05%	90
Hairdryer (04)	81,68%	86,29%	83,92%	248
Heater (05)	70,11%	71,76%	70,93%	85
Incandescent Light Bulb (06)	69,73%	87,16%	77,48%	148
Laptop (07)	86,04%	92,27%	89,04%	207
Microwave (08)	88,18%	84,72%	86,41%	229
Vacuum (09)	92,11%	95,89%	93,96%	73
Washing Machine (10)	63,08%	54,67%	58,57%	75
Acurácia			73,17%	1793
Média Macro[b]	70,53%	71,77%	70,87%	1793
Média Ponderada[c]	72,52%	73,17%	72,58%	1793

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

Figura 41 – Medidas-F e matriz de confusão usando o PLAID1+2: Parte II



(a) Medida-F por classes

(b) Matriz de confusão

As medidas-F, para cada classe treinada, podem ser observadas na Figura 41a. A geladeira (*fridge*), o ar-condicionado (AC), o ventilador (*fan*) e a lavadora de roupas (*washer*) ficaram abaixo da macro-média da medida-F. Na matriz de confusão, exibida na Figura 41b, percebemos que as maiores confusões foram do ar-condicionado com o ventilador (*fan*) e da geladeira com o ar-condicionado.

4.3 Considerações Finais do Capítulo

No primeiro experimento, após esses exaustivos testes, ficou percebido que a característica individual que obteve os melhores resultados foi a trajetória V-I, logo foi a característica adotada para as próximas etapas desta dissertação. Quanto a dimensão das imagens dessa trajetória, percebemos que ao aumentá-la de 16×16 para 50×50 *pixels*, houve também um aumento da acurácia, porém não foi muito significativo para esse experimento. Dentre os classificadores testados nessa etapa, o que obteve os melhores resultados foi o *Random Forest*.

Vale salientar que a escolha dos parâmetros dos classificadores adotada nesta reprodução, que foi a mesma definida por Gao et al. (2015), vista na Tabela 13, pode ter influenciado nos resultados obtidos. Talvez, otimizando os parâmetros de alguns classificadores, os resultados obtidos pudessem ser diferentes. Como a intenção era verificar a característica de distinção mais promissora, isso não foi caracterizado como um problema.

No segundo experimento, já utilizando como característica a trajetória V-I, testamos a RNC separadamente e percebemos que este foi o classificador que apresentou melhores acurácias em relação aos demais testados no primeiro experimento. Aliado ao fato de trabalhos recentes,

como o de Baets et al. (2017b) e Baptista et al. (2018) terem preferido o uso de redes neurais em vez de outros classificadores, a Rede Neural Convolucional foi o classificador adotado para as próximas etapas desta dissertação.

Ao reproduzirmos os experimentos de Baets et al. (2017b) e Baptista et al. (2018), percebemos também que os resultados foram diferentes dos resultados originais, apresentados pelos respectivos autores. Isso não quer dizer que os resultados originais estejam errados, o que não foi a finalidade desta reprodução aqui realizada, porém foram estes os resultados obtidos sem a disponibilização dos códigos-fonte por parte dos autores.

Diferentemente dos experimentos originais, que utilizaram as amostras do PLAID1 e PLAID2 somente, em nossos experimentos foram utilizados o PLAID1, com 1074 amostras, o PLAID2, com 719 amostras, e o PLAID 1+2, com 1793 amostras. Dessa forma, pudemos testar o comportamento dos classificadores e das características para um número maior de situações.

Os documentos *Jupyter Notebook* no qual reproduzimos estes experimentos encontram-se disponíveis no seguinte repositório: <https://github.com/201110008710/NILM_ESP32/tree/master/Jupyter_Notebooks>.

5

Proposta da Rede Neural Convolucional

De acordo com os resultados apresentados no Capítulo 4, foi decidido usar como classificador a Rede Neural Convolucional e as imagens binárias da trajetória VI como característica de distinção de cargas. Desta forma, neste capítulo são abordados a arquitetura definida para a RNC, os resultados obtidos pela mesma e uma análise comparativa em relação aos trabalhos correlatos.

5.1 Arquitetura

Diferentemente da dimensão das imagens da trajetória VI utilizadas nos experimentos do Capítulo 4, que foi de 50×50, utilizamos imagens de dimensão 30×30. Como Gao et al. (2015) e Baets et al. (2017b) utilizaram imagens de dimensão 16×16 e 50×50, respectivamente, escolhemos um valor dentro desse intervalo, que foi 30×30. Além disso, essa redução foi motivada pelos dois seguintes fatores: redução do tamanho da rede, o que é um fator positivo visto que queremos embarcá-la, e a análise dos efeitos dessa redução comparando os resultados obtidos com os dos trabalhos relacionados.

Após exaustivos testes, a melhor arquitetura obtida, exibida na Tabela 27, foi escolhida. Essa arquitetura possui uma camada convolucional (C1) com 64 filtros de tamanho 5×5, uma camada de *pooling* (P1) com janela de tamanho 2×2, outra camada convolucional (C2) com 32 filtros de tamanho 5×5, uma camada de *dropout* (D1) com taxa de 0,26 e, por fim, a camada densa, sendo a saída de tamanho 11, que é quantidade de classes para a qual a rede foi treinada.

5.2 Metodologia

Assim como explicado no Capítulo 4, para a obtenção de uma representação do estado estável para cada dispositivo elétrico, foram extraídas as 10.000 últimas amostras de cada instância do conjunto de dados, sendo essas amostras de tensão e corrente obtidas a uma frequência de

Tabela 27 – Arquitetura da RNC do modelo proposto

Camada	**Filtros**	**Área do Filtro/*Pool***	**Taxa**	**Dimensão da Camada**
Entrada	-	-	-	1@30×30
Convolucional (C1)	64	5×5	-	64@26×26
Pooling (P1)	-	2×2	-	64@13×13
Convolucional (C2)	32	5×5	-	32@9×9
Dropout (D1)	-	-	0,26	32@9×9
Densa (F1)	-	-	-	11

amostragem (f_s) de 30 kHz.

Pelo fato da frequência da rede elétrica do local onde as amostras foram tiradas (f_r) ser de 60 Hz e, sabendo que o número de amostras por período é igual a razão entre a frequência de amostragem e a frequência da rede, temos um total de 500 amostras por período de cada dispositivo elétrico amostrado.

Para a realização dos experimentos, foi utilizada a plataforma *Google Colab*[1]. Utilizamos a linguagem de programação *Python*, cuja versão do pacote instalado era a 3.6.9. Aproveitamos, também, a disponibilidade de GPU dessa plataforma para acelerarmos o treinamento da RNC, utilizando a computação paralela proveniente da integração entre a biblioteca *TensorFlow* (cuja versão do pacote instalado era a 2.3.0) e da API CUDA.

Para garantirmos a reprodutibilidade de ambos os experimentos, foi utilizada uma semente fixa e o pacote *tensorflow-determinism*, versão 0.3.0. Logo, cada experimento foi repetido apenas algumas dezenas de vezes, visto que os resultados eram bastantes similares. Para a característica da trajetória VI, que é a entrada da nossa RNC, foram utilizadas imagens binárias de dimensão igual a 30×30 *pixels*, como explicado na Seção 5.1.

Os experimentos foram realizados usando o PLAID1, PLAID2 e PLAID1+2 como conjunto de dados. Em todos os experimentos, a rede foi treinada por 30 épocas utilizando o otimizador *Adam*, com a taxa de aprendizado padrão de 0,001, e a entropia cruzada como função de perda. Nesses experimentos, também foi utilizado o método *leave-one-out*[2] de validação cruzada. O documento *Jupyter Notebook* deste experimento encontra-se disponível no seguinte repositório: <https://github.com/201110008710/NILM_ESP32/tree/master/Jupyter_Notebooks>

5.3 Experimento

A acurácia obtida para o PLAID1 foi de 82,50%, como mostra a Tabela 28. Comparando com os resultados que obtivemos ao reproduzir o experimento de Baets et al. (2017b), que foi de 81,28% (Tabela 21), e o de Baptista et al. (2018), que foi de 80,45% (Tabela 24), percebemos

[1] <https://bit.ly/3hGNjql>

[2] As medições de 1 residência foram utilizadas como conjunto de teste enquanto que as medições das outras n casas foram utilizadas como conjunto de treinamento, sendo esse processo repetido para cada uma das n casas.

que houve uma pequena melhora, mesmo reduzindo a dimensionalidade das imagens. De forma análoga, a média macro da medida-F também aumentou, partindo de 73,67% e de 71,39%, respectivamente na ordem dos autores citados anteriormente, para 74,76%.

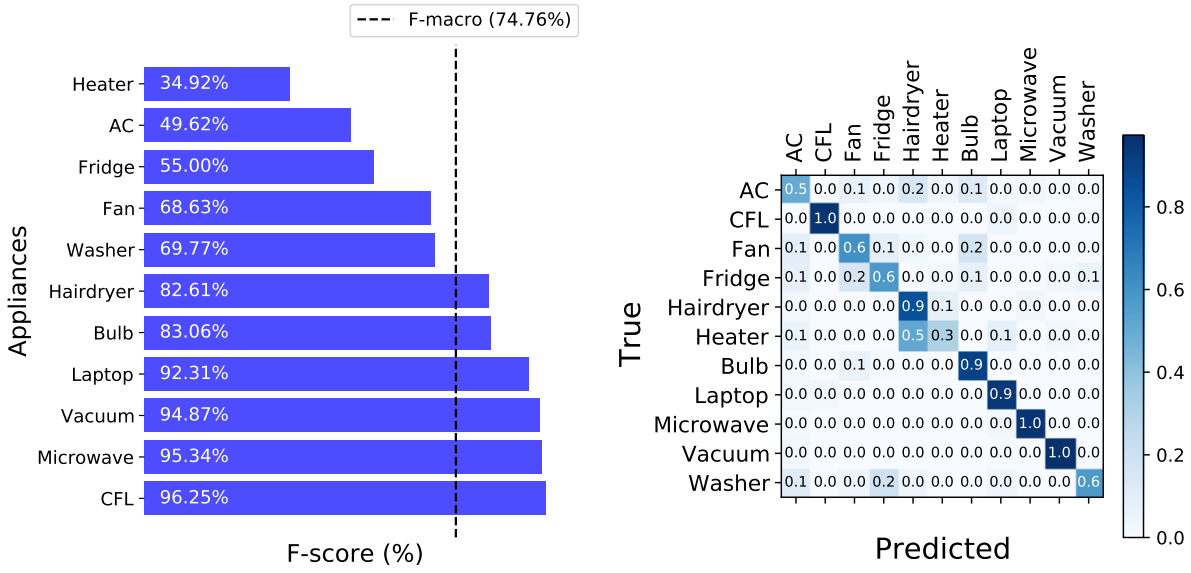
As medidas-F para cada classe treinada podem ser observadas na Figura 42a. Os dispositivos que ficaram abaixo da média macro da medida-F foram: aquecedor (*heater*), ar-condicionado (AC), geladeira (*fridge*), ventilador (*fan*) e lavadora de roupa (*washer*).

Tabela 28 – Relatório de classificação da RNC usando o PLAID1 no modelo proposto

Classes	Precisão	Revocação	Medida-F	Suporte[a]
Air Conditioner (00)	49,25%	50,00%	49,62%	66
Compact Fluorescent Lamp (01)	97,09%	95,43%	96,25%	175
Fan (02)	78,65%	60,87%	68,63%	115
Fridge (03)	52,38%	57,89%	55,00%	38
Hairdryer (04)	80,12%	85,26%	82,61%	156
Heater (05)	39,29%	31,43%	34,92%	35
Incandescent Light Bulb (06)	76,87%	90,35%	83,06%	114
Laptop (07)	90,50%	94,19%	92,31%	172
Microwave (08)	95,00%	95,68%	95,34%	139
Vacuum (09)	92,50%	97,37%	94,87%	38
Washing Machine (10)	88,24%	57,69%	69,77%	26
Acurácia			82,50%	1074
Média Macro[b]	76,35%	74,20%	74,76%	1074
Média Ponderada[c]	82,40%	82,50%	82,15%	1074

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

Figura 42 – Medidas-F e matriz de confusão usando o PLAID1 no modelo proposto



(a) Medida-F por classes

(b) Matriz de confusão

Na matriz de confusão, exibida na Figura 42b, percebemos que a confusão mais acentuada

foi do aquecedor (*heater*) com o secador de cabelos (*hairdryer*), que possuem características elétricas similares, pois ambos são cargas resistivas. Os resultados foram bastante similares aos mostrados na Seção 4.2.2.

A acurácia obtida para o PLAID2 foi de 53,41%, como mostra a Tabela 29. Comparando com os resultados que obtivemos ao reproduzir o experimento de Baets et al. (2017b), que foi de 58,00% (Tabela 22), e o de Baptista et al. (2018), que foi de 56,88% (Tabela 25), percebemos que com a redução da dimensionalidade das imagens, houve uma redução de aproximadamente 10% no pior caso. De forma análoga, a média macro da medida-F também diminuiu, partindo de 61,51% e de 60,25%, respectivamente na ordem dos autores citados anteriormente, para 56,48%.

Tabela 29 – Relatório de classificação da RNC usando o PLAID2 no modelo proposto

Classes	**Precisão**	**Revocação**	**Medida-F**	**Suporte**[a]
Air Conditioner (00)	56,56%	48,59%	52,27%	142
Compact Fluorescent Lamp (01)	88,24%	100,00%	93,75%	45
Fan (02)	16,88%	13,68%	15,12%	95
Fridge (03)	23,96%	44,23%	31,08%	52
Hairdryer (04)	71,83%	55,43%	62,58%	92
Heater (05)	68,75%	44,00%	53,66%	50
Incandescent Light Bulb (06)	48,89%	64,71%	55,70%	34
Laptop (07)	61,54%	45,71%	52,46%	35
Microwave (08)	58,88%	70,00%	63,96%	90
Vacuum (09)	81,40%	100,00%	89,74%	35
Washing Machine (10)	51,02%	51,02%	51,02%	49
Acurácia			53,41%	719
Média Macro[b]	57,09%	57,94%	56,48%	719
Média Ponderada[c]	54,74%	53,41%	53,21%	719

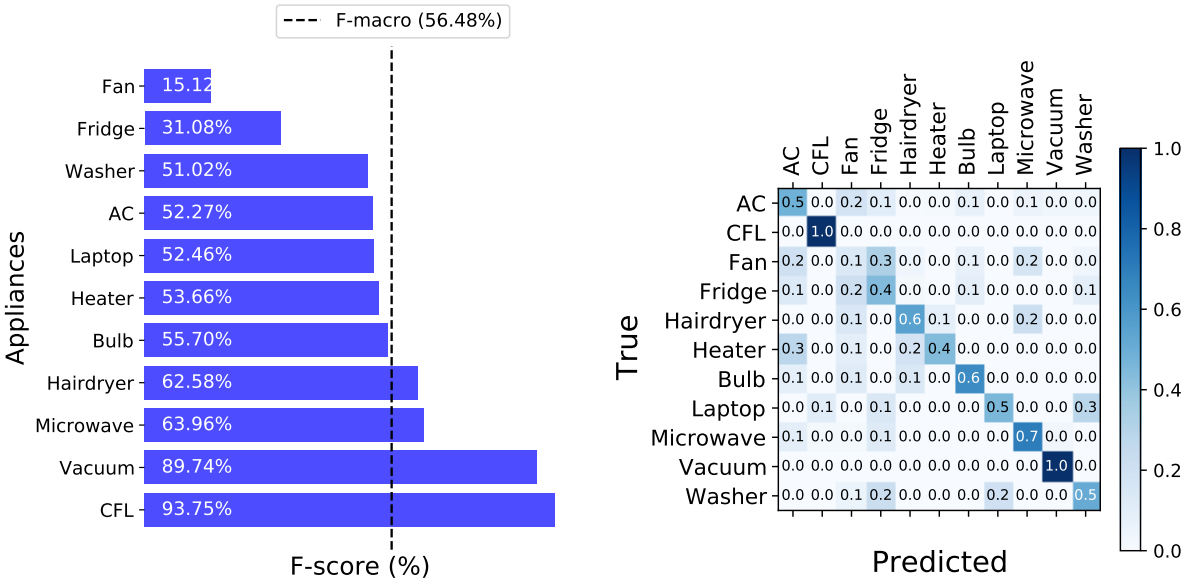
[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

As medidas-F para cada classe treinada podem ser observadas na Figura 43a. Dessa vez, a maioria dos dispositivos ficaram abaixo da média macro da medida-F, sendo eles: ventilador (*fan*), geladeira (*fridge*), lavadora de roupa (*washer*), ar-condicionado (AC), computador portátil (*laptop*), aquecedor (*heater*) e lâmpada incandescente (bulb).

Na matriz de confusão, exibida na Figura 43b, percebemos que as confusões mais acentuadas foram do ventilador com o a geladeira, ar-condicionado e micro-ondas (*microwave*), do aquecedor com o ar-condicionado e do computador portátil com a lavadora de roupa.

A acurácia obtida para o PLAID1+2 foi de 76,58%, como mostra a Tabela 30. Comparando com os resultados que obtivemos ao reproduzir o experimento de Baets et al. (2017b), que foi de 76,69% (Tabela 23), e o de Baptista et al. (2018), que foi de 73,17% (Tabela 26), percebemos que com a redução da dimensionalidade das imagens, houve uma redução insignificante entre as acurácias.

Figura 43 – Medidas-F e matriz de confusão usando o PLAID2 no modelo proposto



(a) Média macro da Medida-F

(b) Matriz de confusão

Já a média macro da medida-F, comportou-se de maneira contrária, aumentando, mas também de forma insignificante, de 73,91% e de 70,87%, respectivamente na ordem dos autores citados anteriormente, para 73,97%.

Tabela 30 – Relatório de classificação da RNC usando o PLAID1+2 no modelo proposto

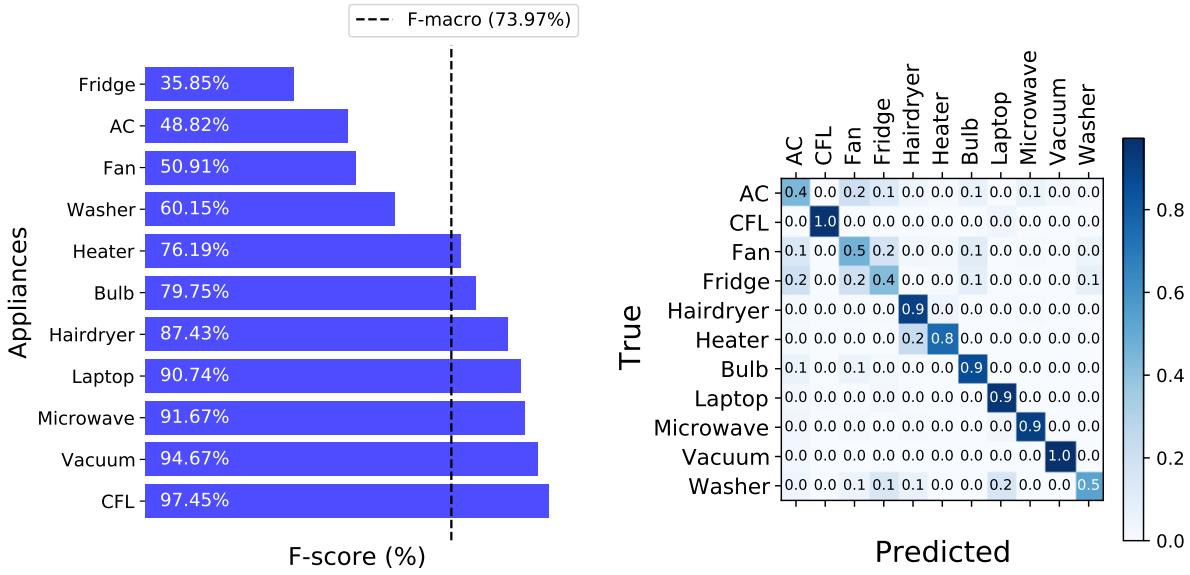
Classes	Precisão	Revocação	Medida-F	Suporte[a]
Air Conditioner (00)	53,76%	44,71%	48,82%	208
Compact Fluorescent Lamp (01)	99,53%	95,45%	97,45%	220
Fan (02)	56,00%	46,67%	50,91%	210
Fridge (03)	31,15%	42,22%	35,85%	90
Hairdryer (04)	84,01%	91,13%	87,43%	248
Heater (05)	77,11%	75,29%	76,19%	85
Incandescent Light Bulb (06)	73,99%	86,49%	79,75%	148
Laptop (07)	87,11%	94,69%	90,74%	207
Microwave (08)	92,07%	91,27%	91,67%	229
Vacuum (09)	92,21%	97,26%	94,67%	73
Washing Machine (10)	68,97%	53,33%	60,15%	75
Acurácia			76,58%	1793
Média Macro[b]	74,17%	74,41%	73,97%	1793
Média Ponderada[c]	76,41%	76,58%	76,22%	1793

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

As medidas-F para cada classe treinada podem ser observadas na Figura 44a. Os únicos dispositivos que ficaram abaixo da média macro da medida-F foram: geladeira (*fridge*), ar-condicionado (AC), ventilador (*fan*), e lavadora de roupa (*washer*).

Na matriz de confusão, exibida na Figura 44b, percebemos que as confusões mais acentuadas foram do ar-condicionado com o ventilador e da geladeira com o ar-condicionado e ventilador. Esses dispositivos, como explicado na Seção 4.2.2, possuem funcionamento parecido.

Figura 44 – Medidas-F e matriz de confusão usando o PLAID1+2 no modelo proposto



(a) Medida-F por classes

(b) Matriz de confusão

A compilação com os resultados tanto das reproduções dos experimentos de Baets et al. (2017a) e de Baptista et al. (2018), obtidos no Capítulo 4, quanto do modelo proposto neste capítulo, podem ser vistos na Tabela 31.

Tabela 31 – Comparativo entre as medidas-F do modelo proposto e dos experimentos reproduzidos

Dispositivo	Baets et al. (2017b)			Baptista et al. (2018)			**Modelo Proposto**		
Elétrico	**PLAID1**	**PLAID2**	**PLAID1+2**	**PLAID1**	**PLAID2**	**PLAID1+2**	**PLAID1**	**PLAID2**	**PLAID1+2**
AC	45,95%	60,45%	47,06%	37,31%	58,06%	39,27%	49,62%	52,27%	48,82%
CFL	95,98%	94,74%	95,28%	96,25%	95,74%	95,02%	96,25%	93,75%	97,45%
Fan	66,02%	10,14%	50,79%	65,99%	13,25%	46,93%	68,63%	15,12%	50,91%
Fridge	48,10%	28,40%	45,77%	38,20%	29,37%	38,05%	55,00%	31,08%	35,85%
Hairdryer	82,13%	67,07%	87,81%	80,12%	63,75%	83,92%	82,61%	62,58%	87,43%
Heater	34,38%	56,52%	73,17%	33,33%	48,48%	70,93%	34,92%	53,66%	76,19%
Bulb	80,65%	70,59%	78,70%	81,27%	62,86%	77,48%	83,06%	55,70%	79,75%
Laptop	93,88%	75,00%	91,00%	93,64%	71,43%	89,04%	92,31%	52,46%	90,74%
Microwave	94,55%	65,69%	91,38%	96,43%	67,01%	86,41%	95,34%	63,96%	91,67%
Vacuum	97,37%	94,59%	95,30%	96,10%	95,89%	93,96%	94,87%	89,74%	94,67%
Washer	71,43%	53,47%	56,72%	66,67%	56,86%	58,57%	69,77%	51,02%	60,15%
Total	**73,67%**	**61,51%**	**73,91%**	**71,39%**	**60,25%**	**70,87%**	**74,76%**	**56,48%**	**73,97%**

5.4 Tabela Comparativa

O comparativo entre os resultados do modelo proposto e dos trabalhos relacionados, considerando os questionamentos definidos na Seção 3.1.2, pode ser visto na Tabela 32.

Tabela 32 – Tabela comparativa dos artigos selecionados atualizada

Trabalhos Selecionados	*Hardware* Utilizado	Grandezas Mensuradas	Transmissão de Dados	*Feature*	Classificador	*Dataset*	Métricas (%) A[a]	P[b]	R[c]	F[d]	G[e]
Weiss et al. (2012)	Não Informado	P, Q e S	Wi-Fi	ΔP/ΔQ	kNN		97,0				
Makonin et al. (2013)	Arduino Due	I		I	HMM	AMPds	79,7				
Chang, Wiratha e Chen (2014)	Portweel PQ7-C100X	V e I		ΔP/ΔQ	MLP		100,0				
Barsocchi et al. (2014)	Arduino Uno	P e Q		P e Q	FSM			84,0	95,0	89,0	87,0
Jonetzko et al. (2015)	PJRC Teensy 3.1	V e I	Wi-Fi	Harmônicas	Agrupamento MLP kNN		93,0 82,0 27,0				
Adabi, Manovi e Mantey (2016)	SEADS [f]	V e I		Harmônicas	kNN		92,0				
Biansoongnern e Plungklang (2016b)	Não Informado	P e Q	Wi-Fi	ΔP/ΔQ	MLP		95,0				
Biansoongnern e Plungklang (2016a)	Não Informado	P e Q	Wi-Fi	ΔP/ΔQ	Não Informado		90,3				
Ferrández-Pastor et al. (2017)	NI myRIO	V e I		V e I [g]	kNN		95,0				
Adiatmoko et al. (2017)	Intel Galileo	V e I		I [h]	MLP		100,0				
Arrachman et al. (2017)	Intel Galileo Gen2	V e I		I [h]	ELM		100,0				
Baets et al. (2017b)[i]		V e I		Trajetória V-I	RNC	PLAID1				77,6[i]	
Syai'in et al. (2018)	Intel Galileo	V e I	Wi-Fi	I [h]	MLP		100,0				
Baptista et al. (2018)	Xilinx Zynq-7000	V e I		Trajetória V-I	RNC	PLAID1 PLAID2				78,2[i] 66,0[i]	
Huang et al. (2019)	ST STM32	V e I		S	HMM		82,0				
Este Trabalho	Espressif ESP32	V e I		Trajetória V-I	RNC	PLAID1 PLAID2 PLAID1+2	82,5 53,4 76,6	76,4 57,1 74,2	74,2 57,9 74,4	74,8 56,5 74,0	

[a] Acurácia [b] Precisão [c] Revocação [d] Medida F [e] Medida G (Índice de Fowlkes-Mallows)
[f] Desenvolvido pelos próprios autores do artigo, ou seja, por Adabi, Manovi e Mantey (2016)
[g] A corrente serviu para o cálculo do máximo *undershoot* / *overshoot* dos dados após a regressão lag-1. Aliado ao tempo de estabilização do sinal, essas foram as características utilizadas na distinção
[h] A tensão e a corrente serviram para o cálculo dos coeficientes de Wavelet, que foi a característica utilizada na distinção
[i] Foram consideradas as métricas originais extraídas dos respectivos artigos, e não as métricas obtidas nas reproduções dos experimentos realizadas nesta dissertação

5.5 Considerações Finais do Capítulo

Após a definição da arquitetura da RNC, os testes realizados neste capítulo nos fez compreender os efeitos da redução da dimensionalidade das imagens binárias da trajetória VI na acurácia. Para o experimento utilizando o PLAID1, conseguimos resultados superiores tanto em acurácia quanto em medida-F se comparados aos obtidos reproduzindo o experimento de Baets et al. (2017b) e de Baptista et al. (2018) mostrado no Capítulo 4.

Para o PLAID2, os resultados foram piores tanto em acurácia quanto em medida-F, o que nos fez perceber que caso seja desejável treinar uma rede somente para esta parte menor do conjunto de dados, devem ser levados em consideração uma nova arquitetura e o uso de imagens de maior dimensionalidade.

Já para o PLAID1+2, os resultados obtidos foram bastante próximos aos obtidos nos experimentos do Capítulo 4. A acurácia obtida foi ligeiramente menor, porém a medida-F foi ligeiramente maior. Dessa forma, visto que os resultados foram bastante parecidos, percebemos que foi viável trabalhar com imagens de dimensão reduzida para atacarmos o problema, já que precisamos otimizar o tamanho da rede para embarcá-la no ESP32.

6

TensorFlow Lite e ESP32: Rede Neural Convolucional Embarcada

Neste capítulo é explicado como o modelo proposto da RNC, visto no Capítulo 5, é aplicado em um sistema embarcado. Para isso, explicamos as etapas do treinamento desse modelo, como é feita a conversão para um modelo do *TensorFlow Lite* e o processo de quantização para adequá-lo ao dispositivo embarcado. Por fim, avaliamos o desempenho do modelo da RNC embarcado no ESP32 e comparamos com o desempenho do modelo proposto.

6.1 Treinamento do Modelo Proposto

Para a conversão do modelo da RNC mostrado no Capítulo 5 em um modelo do *TensorFlow Lite*, o primeiro passo foi treinarmos esse modelo utilizando o conjunto de dados PLAID1+2 (explicado na Seção 2.4), que possui mais amostras e é mais balanceado. A definição das camadas do modelo, à nível de programação, pode ser observada no Código 1.

Na primeira linha do código, o bloco sequencial é declarado. Na segunda e quarta linhas as camadas convolucionais são declaradas, enquanto que terceira e quinta linhas são declaradas as camadas de *pooling* e o *dropout*, respectivamente. Por fim, na sétima linha do Código 1, temos a declaração da camada densa.

Código 1 – Definição do modelo da RNC à nível de programação

```
1 model = keras.Sequential([
2   keras.layers.Conv2D(filters = 64, kernel_size = 5, activation='relu',
        ↪ kernel_initializer = keras.initializers.glorot_uniform(seed = SEED),
        ↪ input_shape=(30, 30, 1)),
3   keras.layers.MaxPool2D(pool_size=2),
4   keras.layers.Conv2D(filters = 32, kernel_size = 5, activation='relu',
        ↪ kernel_initializer = keras.initializers.glorot_uniform(seed = SEED)),
5   keras.layers.Dropout(0.26),
6   keras.layers.Flatten(),
7   keras.layers.Dense(11, activation='softmax', kernel_initializer =
        ↪ keras.initializers.glorot_uniform(seed = SEED))
8 ])
```

Como explicado no Apêndice A, o parâmetro kernel_initializer foi utilizado para reduzir a variabilidade durante a inicialização do *kernel*, possibilitando avaliações do modelo com resultados mais constantes. O otimizador utilizado foi o *Adam*, com taxa de aprendizado padrão (0,001) e utilizando a entropia cruzada como função de perda. Como visto no Código 2, foram utilizadas 40 épocas para o treinamento.

Código 2 – Parâmetros de compilação do modelo

```
1 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
      ↪ metrics=['accuracy'])
2 model.fit(x_train, y_train, epochs = 40,verbose = 0)
```

Dessa vez não foi utilizado o método de validação cruzada *leave-one-out*, mas sim o método *holdout*, no qual o conjunto de dados é dividido de forma fixa em conjunto de treino e de teste. O método *leave-one-out*, que foi utilizado para fins estatísticos e para comparação com os demais trabalhos correlatos, treinava o modelo n vezes, sendo n o número de residências do conjunto de dados.

Dessa forma, eram realizados 64 treinamentos, sendo os resultados para cada um deles armazenados e depois realizado o cálculo estatístico. Além desse método ser mais lento, era necessário uma forma invariável de divisão do conjunto de dados para que o modelo pudesse ser treinado e embarcado, justificando a escolha do método *holdout*.

Tabela 33 – Relatório de classificação da RNC utilizando o método *holdout*

Classes	**Precisão**	**Revocação**	**Medida-F**	**Suporte**[a]
Air Conditioner (00)	94,74%	90,00%	92,31%	40
Compact Fluorescent Lamp (01)	100,00%	97,92%	98,95%	48
Fan (02)	95,00%	97,44%	96,20%	39
Fridge (03)	93,75%	75,00%	83,33%	20
Hairdryer (04)	97,96%	94,12%	96,00%	51
Heater (05)	85,71%	100,00%	92,31%	18
Incandescent Light Bulb (06)	93,55%	100,00%	96,67%	29
Laptop (07)	97,62%	100,00%	98,80%	41
Microwave (08)	97,92%	100,00%	98,95%	47
Vacuum (09)	95,00%	100,00%	97,44%	19
Washing Machine (10)	71,43%	71,43%	71,43%	7
Acurácia			95,54%	359
Média Macro[b]	92,97%	93,26%	92,94%	359
Média Ponderada[c]	95,63%	95,54%	95,47%	359

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

Ademais, como analisamos o modelo utilizando o método *leave-one-out* no Capítulo 5 e analisaremos utilizando o método *holdout* neste capítulo, teremos um parâmetro de comparação ao analisarmos os resultados, visto que é o mesmo modelo. O conjunto de dados foi analisado

com a divisão da seguinte forma: 80% treinamento e 20% de teste, como mostrado na primeira linha do Código 3. É possível visualizar na segunda linha que essas amostras foram divididas de forma aleatória, utilizando a função randômica do *Numpy*.

Código 3 – Divisão do conjunto de dados utilizando o método *holdout*

```
1 TRAIN_SPLIT =  int(0.8 * n)
2 indices = np.random.permutation(n)
3 train_idx, test_idx = indices[:TRAIN_SPLIT], indices[TRAIN_SPLIT:]
4 x_train, x_test = imgVI[train_idx], imgVI[test_idx]
5 y_train, y_test = appliancesIDs[train_idx], appliancesIDs[test_idx]
```

Ao realizarmos a avaliação do modelo, obtivemos uma acurácia de 95,54%, conforme mostrado na Tabela 33. Avaliando para todo o conjunto de dados, para posteriormente compará-lo com o modelo quantizado embarcado, obtivemos uma acurácia de 98,55%, conforme mostrado na Tabela 34.

Tabela 34 – Relatório de classificação da RNC para todas as instâncias do PLAID1+2

Classes	**Precisão**	**Revocação**	**Medida-F**	**Suporte**[a]
Air Conditioner (00)	99,02%	97,60%	98,31%	208
Compact Fluorescent Lamp (01)	100,00%	99,55%	99,77%	220
Fan (02)	99,05%	99,52%	99,29%	210
Fridge (03)	98,84%	94,44%	96,59%	90
Hairdryer (04)	99,59%	98,79%	99,19%	248
Heater (05)	96,59%	100,00%	98,27%	85
Incandescent Light Bulb (06)	98,01%	100,00%	99,00%	148
Laptop (07)	95,39%	100,00%	97,64%	207
Microwave (08)	99,57%	100,00%	99,78%	229
Vacuum (09)	98,65%	100,00%	99,32%	73
Washing Machine (10)	96,97%	85,33%	90,78%	75
Acurácia			98,55%	1793
Média Macro[b]	98,34%	97,75%	97,99%	1793
Média Ponderada[c]	98,57%	98,55%	98,53%	1793

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

6.2 Conversão em Modelo do *TensorFlow Lite*

Para convertermos o modelo do *TensorFlow* em um modelo do *TensorFlow Lite*, como mostrado no Código 4, inicialmente, nas duas primeiras linhas de código, convertemos o modelo sem nenhuma quantização. O próximo passo, mostrado na terceira linha, foi salvar o modelo, já convertido, em arquivo. Na quarta linha, o comando xxd[1] é instalado, caso ainda não esteja. Com o uso desse comando, na sexta linha, o hexadecimal correspondente ao modelo do *TensorFlow*

[1] O comando xxd, ou hexdump, gera um *dump* hexadecimal de um determinado arquivo. Ele também pode converter esse *dump* hexadecimal de volta à sua forma binária original (MERTZ, 2020).

Lite é salvo como um arquivo da linguagem de programação C, que será vital para a transferência do modelo para o dispositivo embarcado.

Embora o modelo proposto tenha sido bem resumido, com um total de 81.419 parâmetros, como exibido na Tabela 35, ainda assim, após a conversão, ocupou 1,92 MB de espaço em disco, inviabilizando a sua implementação no ESP32. Para ser capaz de embarcar este modelo, precisamos fazer uma quantização pós-treinamento.

Código 4 – Conversão do modelo do *TensorFlow* em *TensorFlow Lite*

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
open("30x30_PLAID1_2.tflite", "wb").write(tflite_model)
!apt-get -qq install xxd
!xxd -i 30x30_PLAID1_2.tflite > 30x30_PLAID1_2.cc
```

Tabela 35 – Sumário com os parâmetros do modelo compilado

Camada	Dimensão da Saída	Parâmetros
Conv2D	(26, 26, 64)	1664
MaxPooling2D	(13, 13, 64)	0
Conv2D	(9, 9, 32)	51232
Dropout	(9, 9, 32)	0
Flatten	2592	0
Dense	11	28523
	Parâmetros Totais:	**81.419**
	Parâmetros Treináveis:	**81.419**
	Parâmetros Não-Treináveis:	**0**

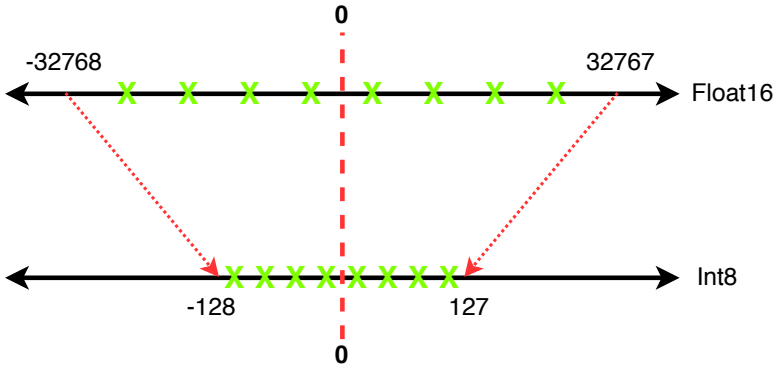
6.3 Quantização do Modelo do *TensorFlow Lite*

A partir da versão 2.3.0 do *TensorFlow Lite*, as conversões híbridas não foram mais aceitas no ESP32. Este tipo de conversão quantizava apenas os pesos da rede neural, deixando outros dados variáveis[2] em ponto flutuante inalterados. Cientes disso, tentamos realizar a conversão usando quantização apenas de inteiros, porém ao embarcarmos o modelo no dispositivo, o mesmo não foi aceito por incompatibilidade.

O processo de quantização consiste em mapear os valores mínimo e máximo do intervalo flutuante para o mínimo e máximo do intervalo quantizado (NEURAL NETWORK DISTILLER, 2020), como mostrado na Figura 45. Dessa forma, consequentemente, há uma redução da precisão da variável (TENSORFLOW, 2020c). Como a saída da rede neural é dada utilizando a codificação *One-Hot*, essa redução da precisão geralmente não interfere no resultado, pois ao obtermos o argumento máximo desse vetor de saída, teremos apenas um valor próximo de 1 e o restante próximo de 0.

[2] Dados como entrada/saída do modelo e intermediários entre camadas (TENSORFLOW, 2020c).

Figura 45 – Exemplo simplificado de quantização



Diante da impossibilidade citada anteriormente, optamos pela conversão por meio da quantização de *fallback* de ponto flutuante, onde todos os pesos e dados variáveis são quantizados, tornando o modelo significativamente menor em relação ao modelo original. Para esse tipo de conversão, o conversor do *TensorFlow Lite* deixa os tensores de entrada e saída na forma de ponto flutuante, a fim de manter a compatibilidade com aplicativos que tradicionalmente os usam desta forma.

Para quantizar os dados variáveis, é necessário fornecer um conjunto de dados representativo, que permite ao conversor estimar uma faixa dinâmica para esses valores. Para isso é necessário criar uma função geradora que forneça um conjunto de dados de entrada grande o suficiente para representar os valores típicos (TENSORFLOW, 2020c).

Alguns exemplos dessa função são mostrados no Código 5, cuja função gera o conjunto de dados representativo a partir do próprio vetor de entrada da RNC, contendo as matrizes binárias da trajetória V-I, e no Código 6, cuja função gera as representações a partir das imagens binárias da trajetória V-I salvas em formato de imagem.

Código 5 – Função geradora de conjunto de dados representativo usando imagens

```
1  def representative_data_generator():
2    dataset_list = tf.data.Dataset.list_files("/content/drive/My Drive/Colab
         ↪ Notebooks/PLAID/Images/30x30/*")
3    for i in range(100):
4      image = next(iter(dataset_list))
5      image = tf.io.read_file(image)
6      image = tf.io.decode_jpeg(image, channels=1)
7      image = tf.image.resize(image, [30, 30])
8      image = tf.cast(image / 255., tf.float32)
9      image = tf.expand_dims(image, 0)
10     yield [image]
```

A quantização do modelo, como mostrado no Código 7, é realizada seguindo basicamente os mesmos comandos da conversão do modelo do *TensorFlow* pro *TensorFlow Lite*, explicado no Código 4. A diferença está nos comandos da segunda e terceira linhas.

Ainda no Código 7, o comando da segunda linha define o tipo de otimização a ser utilizada, sendo de três tipos diferentes: padrão, latência ou tamanho. O tipos de otimização por latência

Código 6 – Função geradora de conjunto de dados representativo usando um vetor

```
1 def representative_data_generator():
2   for input_value in tf.data.Dataset.from_tensor_slices(
        ↪ imgVI.astype(np.float32)).batch(1).take(100):
3     yield [input_value]
```

ou tamanho fazem a quantização parcial, além de terem se tornado obsoletas (TENSORFLOW, 2020d), logo preferimos utilizar o tipo padrão. O comando da terceira linha é onde a função geradora do conjunto de dados representativo é passada como parâmetro.

Código 7 – Quantização do modelo do *TensorFlow Lite*

```
1 converter = tf.lite.TFLiteConverter.from_keras_model(model)
2 converter.optimizations = [tf.lite.Optimize.DEFAULT]
3 converter.representative_dataset = representative_data_generator
4 tflite_model_quant = converter.convert()
5 open("fallback_quant_30x30_PLAID1_2.tflite", "wb").write(tflite_model_quant)
6 !apt-get -qq install xxd
7 !xxd -i fallback_quant_30x30_PLAID1_2.tflite > fallback_quant_30x30_PLAID1_2.cc
```

Após a quantização, nosso modelo passou a ocupar apenas 531 KB de espaço em disco, ou seja, aproximadamente quatro vezes menor que o modelo original. Ao avaliarmos o modelo quantizado para todo o conjunto de dados, a acurácia obtida foi de 98,55%, conforme mostrado na Tabela 36. Em comparação com o modelo original, sendo que o modelo quantizado comportou-se de maneira exatamente igual, não houve perda de acurácia.

Tabela 36 – Relatório de classificação do modelo da RNC pós-quantização

Classes	**Precisão**	**Revocação**	**Medida-F**	**Suporte**[a]
Air Conditioner (00)	99,02%	97,60%	98,31%	208
Compact Fluorescent Lamp (01)	100,00%	99,55%	99,77%	220
Fan (02)	99,05%	99,52%	99,29%	210
Fridge (03)	98,84%	94,44%	96,59%	90
Hairdryer (04)	99,59%	98,79%	99,19%	248
Heater (05)	96,59%	100,00%	98,27%	85
Incandescent Light Bulb (06)	98,01%	100,00%	99,00%	148
Laptop (07)	95,39%	100,00%	97,64%	207
Microwave (08)	99,57%	100,00%	99,78%	229
Vacuum (09)	98,65%	100,00%	99,32%	73
Washing Machine (10)	96,97%	85,33%	90,78%	75
Acurácia			98,55%	1793
Média Macro[b]	98,34%	97,75%	97,99%	1793
Média Ponderada[c]	98,57%	98,55%	98,53%	1793

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

6.4 Aplicação do Modelo do *TensorFlow Lite* no ESP32

Para embarcar o modelo no ESP32, os passos descritos nos Apêndices B, C e D foram inicialmente seguidos. O código fonte foi então, aos poucos, sendo adaptado para o nosso problema para posteriormente ser carregado no dispositivo embarcado. Todos os códigos apresentados nesta seção foram executados no ESP32. Na função principal, incluímos todas as bibliotecas necessárias, incluindo a biblioteca *Arduino*, como mostrado no Código 8, para elevar o nível de abstração e facilitar a reprodutibilidade do trabalho.

Código 8 – Bibliotecas necessárias para compilar o projeto do *TensorFlow Lite*

```
1 #include "tensorflow/lite/micro/all_ops_resolver.h"
2 #include "tensorflow/lite/micro/micro_error_reporter.h"
3 #include "tensorflow/lite/micro/micro_interpreter.h"
4 #include "tensorflow/lite/schema/schema_generated.h"
5 #include "tensorflow/lite/version.h"
6 #include "model.h"
7 #include <Arduino.h>
```

Ainda analisando as bibliotecas exibidas no Código 8, a *all_ops_resolver.h* é responsável pelas operações usadas pelo interpretador para executar o modelo, *micro_error_reporter.h* exibe informações de depuração, *micro_interpreter.h* carrega e executa os modelos, *schema_generated.h* é responsável pelo arquivo do modelo *FlatBuffer Lite* e *version.h* fornece informações da versão para o esquema do *TensorFlow Lite* (TENSORFLOW, 2020a). O cabeçalho da classe *model* também é incluído ao projeto.

Para evitar um conflito de nomes entre classes de bibliotecas do *TensorFlow Lite*, do ESP32 e do Arduino, foi criada uma região declarativa, como mostrado no Código 9. Na terceira, quarta e quinta linhas são declarados e inicializados, respectivamente, o relator de erros durante as inferências, o modelo do *TensorFlow Lite* e o interpretador responsável por carregar e executar os modelos. Na sexta e sétima linhas são inicializados os tensores de entrada e saída do modelo da RNC.

Código 9 – Inicialização dos parâmetros do *TensorFlow Lite*

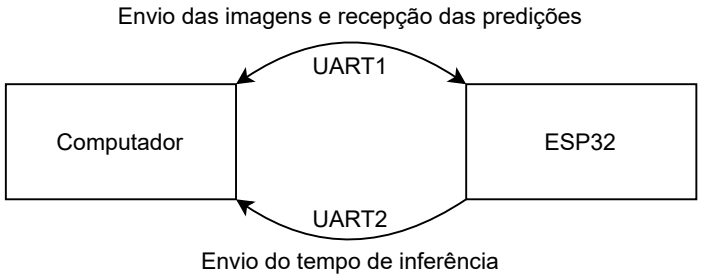
```
1  namespace
2  {
3      tflite::ErrorReporter* error_reporter = nullptr;
4      const tflite::Model* model = nullptr;
5      tflite::MicroInterpreter* interpreter = nullptr;
6      TfLiteTensor* input = nullptr;
7      TfLiteTensor* output = nullptr;
8
9      const int kModelArenaSize = 55 * 1024;
10     const int kExtraArenaSize = 1024;
11     const int kTensorArenaSize = kModelArenaSize + kExtraArenaSize;
12     alignas(16) uint8_t tensor_arena[kTensorArenaSize];
13 }
```

O espaço de memória usado para alocar os tensores de entrada, saída e os vetores intermediários, conhecido como *Arena Size*, mostrado na décima primeira linha do Código 9, foi de 57344 bytes no total, incluindo o tamanho necessário para o modelo e um espaço extra. Durante a compilação, um aviso recomendou o alinhamento do *tensor_arena* a 16 bytes, o que feito na declaração do mesmo, visto na décima segunda linha.

Código 10 – Função de configuração do projeto do *TensorFlow Lite*

```
1  void setup()
2  {
3      Serial.begin(115200);
4      Serial2.begin(115200);
5
6      static tflite::MicroErrorReporter micro_error_reporter;
7      error_reporter = &micro_error_reporter;
8
9      model = tflite::GetModel(tf_model);
10     if (model->version() != TFLITE_SCHEMA_VERSION)
11     {
12         TF_LITE_REPORT_ERROR(error_reporter, "Model provided is schema version %d
               ↪ not equal to supported version %d.", model->version(),
               ↪ TFLITE_SCHEMA_VERSION);
13         return;
14     }
15
16     static tflite::AllOpsResolver resolver;
17
18     static tflite::MicroInterpreter static_interpreter(model, resolver,
           ↪ tensor_arena, kTensorArenaSize, error_reporter);
19     interpreter = &static_interpreter;
20
21     TfLiteStatus allocate_status = interpreter->AllocateTensors();
22     if (allocate_status != kTfLiteOk)
23     {
24         TF_LITE_REPORT_ERROR(error_reporter, "AllocateTensors() failed");
25         return;
26     }
27     input = interpreter->input(0);
28     output = interpreter->output(0);
29 }
```

Figura 46 – Representação do uso das portas seriais do sistema



A função de configuração setup(), cujo nome é em razão a compatibilidade com o Arduino, é mostrada no Código 10. Inicialmente são configuradas duas portas seriais, como visto na terceira e quarta linhas, uma que receberá as imagens de entrada e enviará a inferência como saída (UART1) e outra que serve para depuração do código (UART2), informando, por exemplo,

o tempo de latência de cada inferência, como mostrado na Figura 46. Na sexta e sétima linhas é configurado o registro de gravação de erros.

Na nona linha, o modelo é instanciado usando os dados do vetor *tf_model*, que é declarado na classe *model*, como mostrado no Código 11, sendo esse vetor alinhado a 8 bytes para garantir acessos alinhados de 64 bits. Para preencher esse vetor, deve ser copiado o vetor do arquivo *fallback_quant_*30*x*30*_PLAID*1_2*.cc*, que foi gerado por meio do Código 7, a partir da conversão do modelo proposto em modelo do *TensorFlow Lite* com a pós-quantização.

Da décima à décima quarta linha, o modelo é verificado para garantir que a versão do esquema seja compatível com versão que está sendo utilizada. Na décima sexta linha, o resolvedor de operações é instanciado. Ele é responsável por carregar todas as operações disponíveis no *TensorFlow Lite* para microcontroladores. Na décima oitava e décima nona linhas, o interpretador é instanciado com as variáveis já explicadas.

Da vigésima primeira à vigésima sexta linha, é feita a alocação dos tensores e a verificação do êxito dessa alocação. Caso o tamanho do *tensor_arena* tenha sido declarado de forma insuficiente, a alocação dos tensores falhará. Por fim, na vigésima sétima e vigésima oitava linhas, são obtidos os ponteiros para os tensores de entrada e saída do modelo.

Código 11 – Declaração da matriz do modelo do *TensorFlow Lite*

```
1 alignas(8) const unsigned char tf_model[] = {
2     0x20, 0x00, 0x00, 0x00, 0x54, 0x46, 0x4c, 0x33, 0x00, 0x00, 0x00, 0x00,
3     0x00, 0x00, 0x12, 0x00, 0x1c, 0x00, 0x04, 0x00, 0x08, 0x00, 0x0c, 0x00,
4     ....., ....., ....., ....., ....., ....., ....., ....., ....., ....., .....,
5     0x00, 0x00, 0x00, 0x11, 0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x0a, 0x00,
6     0x0c, 0x00, 0x07, 0x00, 0x00, 0x00, 0x08, 0x00, 0x0a, 0x00, 0x00, 0x00,
7     0x00, 0x00, 0x00, 0x03, 0x03, 0x00, 0x00, 0x00
8 };
9 const int tf_model_len = 88304;
```

Para realizar a leitura das imagens e o envio das predições pela porta serial, adaptamos algumas funções do projeto desenvolvido por Korablyov (2020). As imagens são lidas por intermédio da função mostrada no Código 12, exibida também na Figura 47 por meio de fluxograma. Os parâmetros que devem ser passados para a função são: o tensor de entrada; a largura, o comprimento e o número de canais da imagem de entrada; e o tamanho do *buffer* que receberá as informações da porta serial.

Na terceira linha, declaramos um *buffer* com memória alocada dinamicamente, com o tamanho passado por parâmetro ao chamar a função. Esse *buffer* é necessário para receber os dados da serial em pequenos pedaços, visto que para imagens de dimensões maiores e de maior número de canais pode não ser possível o recebimento de todos os dados de uma única vez.

Da sexta à vigésima primeira linha, temos um laço, cuja saída só ocorre após o recebimento de todos as posições da matriz que formam a imagem. À medida que cada posição é recebida, seu valor já vai sendo normalizado e armazenado no tensor de entrada do modelo do *TensorFlow Lite*.

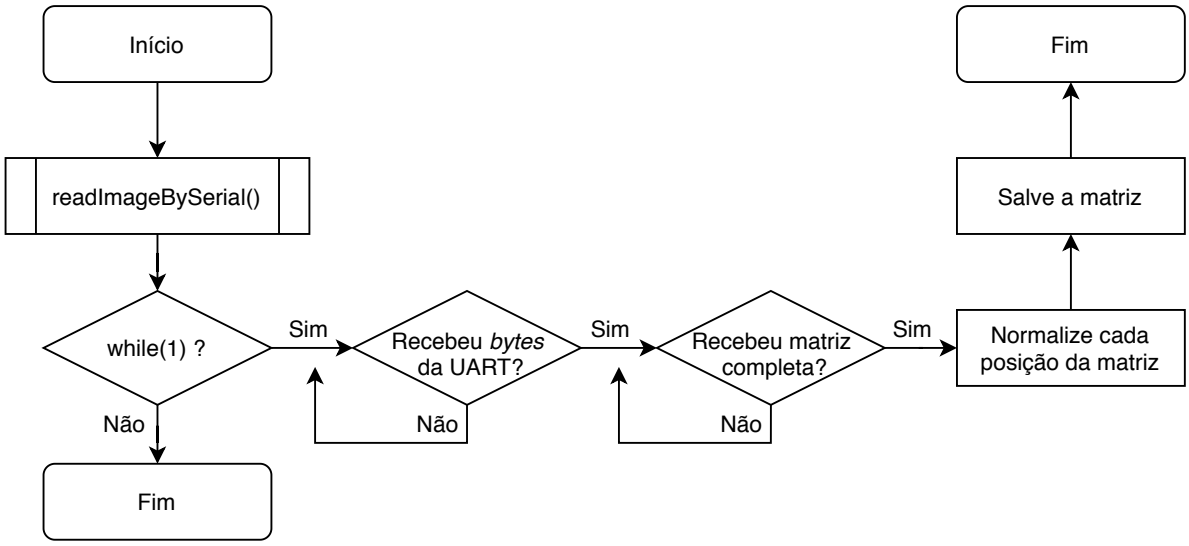
A quantidade de posições dessa matriz é dada pela multiplicação da largura, do comprimento e do número de canais da imagem de entrada.

Dessa forma, caso a imagem possua a dimensão 30×30×1, por exemplo, sua matriz terá 900 posições. Caso o tamanho do *buffer* definido seja 100, por exemplo, quando a imagem for enviada pela serial, serão realizadas 9 leituras para que essa imagem seja completamente carregada no tensor de entrada.

Código 12 – Função que faz a leitura das imagens enviadas pela porta serial

```
1  void readImageBySerial(TfLiteTensor* input, int img_width, int img_height, int
       ↪ img_channel, uint8_t bufferLen)
2  {
3      uint8_t *rxBuffer = (uint8_t *)malloc(bufferLen + 1);
4      int imgBytes = 0;
5      int streamBytes = 0;
6      while(1)
7      {
8          streamBytes = Serial.readBytes(rxBuffer, bufferLen);
9          if (streamBytes > 0)
10         {
11             for (int i = 0; i < streamBytes; imgBytes++, i++)
12             {
13                 input->data.f[imgBytes] = static_cast<float>(rxBuffer[i]) / 255.0f;
14             }
15         }
16         if (imgBytes >= img_width*img_height*img_channel - 1)
17         {
18             imgBytes = 0;
19             break;
20         }
21     }
22     return;
23 }
```

Figura 47 – Fluxograma da função de leitura



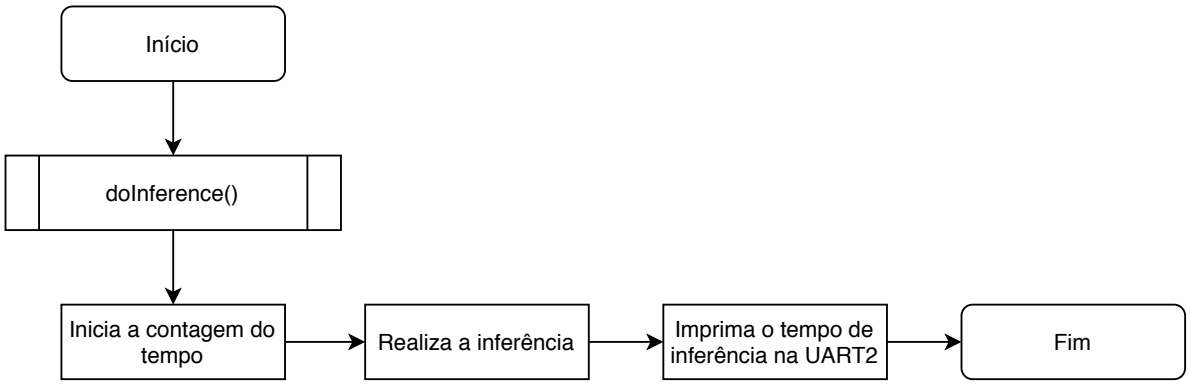
A inferência é realizada utilizando a função mostrada no Código 13, representada também por fluxograma, exibido na Figura 48. Nas duas primeiras e duas últimas linhas, temos variáveis

responsáveis pela medição do tempo, em microssegundos, em que a inferência foi realizada. Da sexta à décima linha, o interpretador é invocado para executar o modelo, sendo verificado também se essa execução foi bem-sucedida.

Código 13 – Função responsável pela inferência do modelo do *TensorFlow Lite*

```
1  void doInference()
2  {
3      unsigned long startTime = millis();
4      unsigned long elapsedTime = 0;
5      TfLiteStatus invoke_status = interpreter->Invoke();
6      if (invoke_status != kTfLiteOk)
7      {
8          TF_LITE_REPORT_ERROR(error_reporter, "Invoke failed");
9          return;
10     }
11     elapsedTime = millis() - startTime;
12     Serial2.printf("Inference Time: %lu\n", elapsedTime);
13 }
```

Figura 48 – Fluxograma da função de inferência



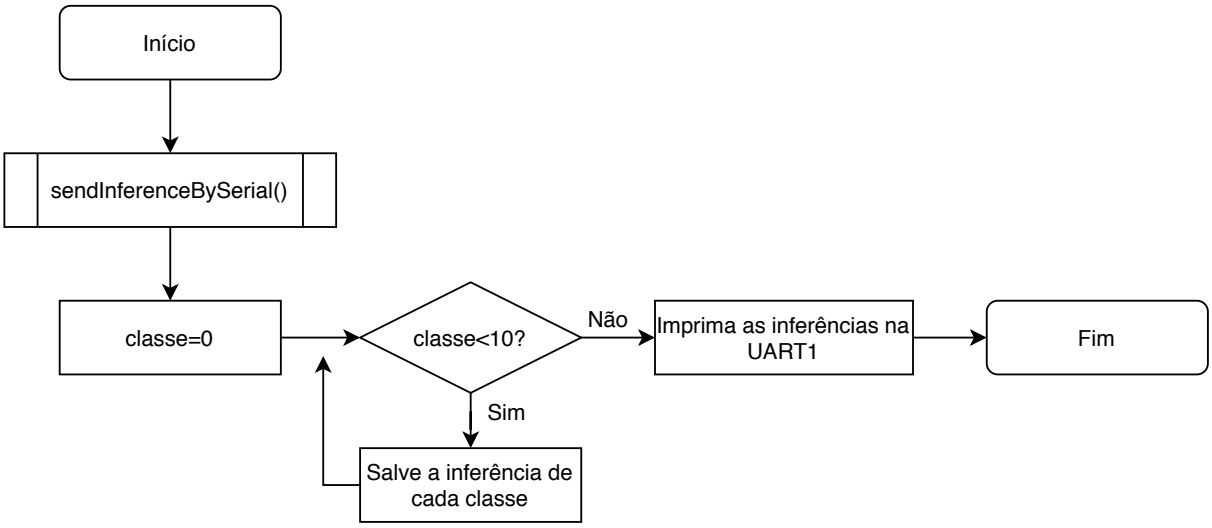
Caso a execução do modelo tenha sido bem-sucedida, as inferências podem ser obtidas acessando o tensor de saída. Para enviar essas inferências pela porta serial, a função exibida no Código 14 foi utilizada. O fluxograma desta pode ser observado na Figura 49.

Código 14 – Função responsável pelo envio das inferências pela porta serial

```
1  void sendInferenceBySerial(TfLiteTensor *output)
2  {
3      char str[250] = {0};
4      char buf[20] = {0};
5      int numElements = output->dims->data[1];
6      for (int i = 0; i < numElements; i++)
7      {
8        sprintf(buf, "%e,", static_cast<float>(output->data.f[i]));
9        strcat(str, buf);
10     }
11     strcat(str, "\n");
12     Serial.println(str);
13     Serial2.println(str);
14 }
```

Na quinta linha do código, obtivemos o número de elementos de saída, ou seja, o número de classes do modelo, que são 11 para o PLAID. Como a saída utiliza a codificação *one-hot*, na oitava linha, a inferência para cada uma dessas 11 classes são armazenadas em uma cadeia de caracteres, sendo cada inferência separada por vírgula. Na décima segunda e décima terceira linhas do código, essa cadeia é enviada por meio das portas seriais.

Figura 49 – Fluxograma da função de retorno das inferências



Por fim, temos a função de repetição *loop*(), cujo nome é essencial para a compatibilidade com o *Arduino*. Nesta função, mostrada no Código 15, são declarados a largura, comprimento e número de canais da imagem de entrada e o tamanho do *buffer* de leitura da porta serial. É nela também em que as funções explicadas anteriormente, responsáveis pela leitura das imagens, pela inferência e pelo envio dessas inferências, são chamadas. O fluxograma que representa esta função pode ser visto na Figura 50.

Código 15 – Função de repetição do projeto do *TensorFlow Lite*

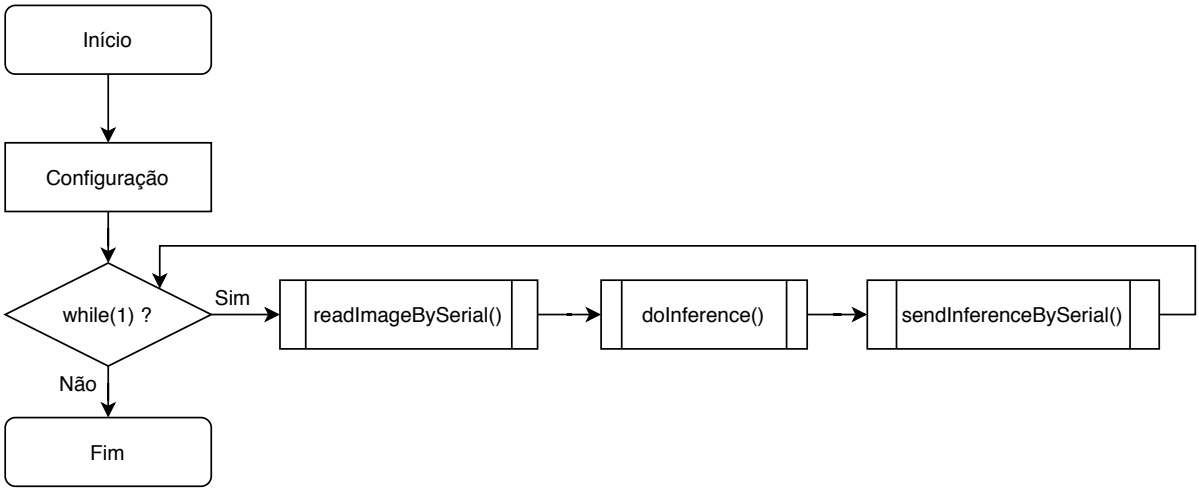
```
1  void loop()
2  {
3      int img_width = 30;
4      int img_height = 30;
5      int img_channel = 1;
6      uint8_t serialBuffer = 100;
7      readImageBySerial(input, img_width, img_height, img_channel, serialBuffer);
8      doInference();
9      sendInferenceBySerial(output);
10 }
```

Após a compilação do modelo, notamos que o mesmo ocupou cerca de 23,9% (78156 de 327680 bytes) da memória RAM e 35,2% (460990 de 1310720 bytes) da memória *flash* do ESP32. Embora tenha sobrado um espaço considerável de memória, não adiantaria aumentarmos a complexidade do modelo da RNC, visto que acarretaria em um tempo de inferência maior.

Porém, esses recursos que sobraram podem ser utilizados para outras finalidades, como por exemplo, o processamento de dados de sensores e atuadores, e para a transmissão de informações

por *Wi-Fi* ou *Bluetooth*. O código completo está disponível no seguinte repositório: <https://github.com/201110008710/NILM_ESP32/tree/master/ESP32_TensorFlowLite_PLAID>.

Figura 50 – Fluxograma da função de repetição



6.5 Avaliação do Modelo do *TensorFlow Lite* Aplicado no ESP32

Para testarmos o modelo diretamente no ESP32, criamos um documento *Jupyter Notebook* para fazer o envio das imagens por meio da porta serial. Os códigos apresentados nesta seção foram executados no computador. Cada uma das 1793 imagens da união do conjunto de dados PLAID1 e PLAID2 foram enviadas, sendo a inferência recebida em seguida e armazenada para a obtenção das métricas. Para o registro de outras informações, como a latência, por exemplo, uma outra porta serial do dispositivo embarcado foi utilizada.

Código 16 – Função de inferência do modelo do *TensorFlow Lite*

```
1  def predict_TFLite_Model(self, tflite_model, raw_image):
2      interpreter = tf.lite.Interpreter(model_path=str(tflite_model))
3      interpreter.allocate_tensors()
4      input_details = interpreter.get_input_details()
5      output_details = interpreter.get_output_details()
6      input_shape = input_details[0]['shape']
7      norm_image = cv2.normalize(raw_image, None, norm_type=cv2.NORM_MINMAX)
8      norm_image = norm_image.reshape(input_shape)
9      input_data = np.array(norm_image, dtype=np.float32)
10     interpreter.set_tensor(input_details[0]['index'], input_data)
11     interpreter.invoke()
12     output_data = interpreter.get_tensor(output_details[0]['index'])[0]
13     return output_data
```

Inicialmente, utilizamos a função mostrada no Código 16, que foi adaptada de TensorFlow (2020b), para fazer a inferência de cada imagem individualmente. Na segunda e terceira linhas, o modelo é carregado e os tensores são alocados, respectivamente. Na quarta e quinta linhas,

são obtidos os detalhes dos tensores de entrada e saída do modelo. Na sexta linha, é obtida a dimensão da imagem de entrada requerida pelo modelo, sendo a imagem que será carregada ajustada a essa dimensão, na oitava linha, caso já não esteja.

A sétima linha é responsável pela normalização da imagem, enquanto que a nona e décima linhas carregam a imagem no tensor de entrada. Na décima primeira linha, o interpretador do modelo é invocado, realizando a inferência, que é retornada posteriormente, por meio do comando da décima segunda linha.

Código 17 – Função de inferência do modelo carregado no ESP32

```
1 def predict_ESP32(self, raw_image):
2     self.init_serial(self.serial_port)
3     input_data = np.array(raw_image, dtype=np.uint8)
4     bytes_sent = self.send_to_ESP32(input_data)
5     response_str = self.read_from_ESP32()
6     response_str = response_str.decode("utf-8")
7     predictions = np.fromstring(response_str, dtype=np.float32, sep=',')
8     self.close_serial()
9     return predictions
```

Já a função mostrada no Código 17, serve para fazer a inferência de cada imagem, individualmente, no ESP32. Na segunda linha, a porta serial é inicializada para fazer o envio dos dados. Na terceira linha, a imagem é convertida em um vetor cuja dimensão é a multiplicação da largura, comprimento e do número de canais dessa imagem. A quarta linha é responsável pelo envio do vetor para o ESP32 por meio da porta serial.

Já na quinta linha, é aguardada a inferência enviada de volta pelo dispositivo embarcado. O comando da sexta linha fica responsável pela decodificação da inferência que foi enviada. Na sétima linha, acontece a conversão da cadeia de caracteres, contendo a inferência para cada classe separa por vírgula, em um vetor, o qual é retornado na nona linha após o fechamento da porta serial, que ocorre na oitava linha.

Código 18 – Função de avaliação do modelo do *TensorFlow Lite*

```
1  def evaluate_TFLite(self, tflite_model:str, images: list, true_Y: list, verbose:
       ↪ int = 1):
2      print("\nEvaluating TFLite model...")
3      start_time = time.time()
4      n = len(images)
5      tflite_predicts = []
6      for i, img in enumerate(images, start=1):
7          if verbose:
8              elapsed_time = time.time()-start_time
9              print('\rProgress: %4d/%4d (%.2f%%)\t Time elapsed: %s' %
                   ↪ (i,n,100*(i/n), time.strftime("%H:%M:%S",
                   ↪ time.gmtime(elapsed_time))), end="")
10         raw_image = cv2.imread(img,0)
11         tflite_predicts.append(np.argmax(self.predict_TFLite_Model(tflite_model,
               ↪ raw_image)))
12     print("\nAccuracy of TFLite Model: {:.4}".format(accuracy_score(true_Y,
           ↪ tflite_predicts)))
13     return tflite_predicts
```

Para carregar todas as imagens e fazer a avaliação do modelo do *TensorFlow Lite*, foi utilizada a função mostrada no Código 18. São passados como parâmetros o modelo, o vetor de imagens e o vetor com os rótulos das imagens. A função nada mais é que uma chamada recursiva da função mostrada no Código 16, sendo iterada pelo número total de imagens e retornado um vetor com as predições realizadas para cada uma delas. A função também exibe o tempo levado para toda a execução e exibe também a acurácia do modelo.

Código 19 – Função de avaliação do modelo carregado no ESP32

```
1  def evaluate_ESP32(self, images: list, true_Y: list, verbose: int = 1):
2      print("\nEvaluating ESP32...")
3      start_time = time.time()
4      n = len(images)
5      esp32_predicts = []
6      for i, img in enumerate(images, start=1):
7          if verbose:
8              elapsed_time = time.time()-start_time
9              print('\rProgress: %4d/%4d (%.2f%%)\t Time elapsed: %s' %
                   ↪ (i,n,100*(i/n), time.strftime("%H:%M:%S",
                   ↪ time.gmtime(elapsed_time))), end="")
10         raw_image = cv2.imread(img,0)
11         esp32_predicts.append(np.argmax(self.predict_ESP32(raw_image)))
12     print("\nAccuracy of ESP32: {:.4}".format(accuracy_score(true_Y,
           ↪ esp32_predicts)))
13     return esp32_predicts
```

A função mostrada no Código 19 avalia o modelo carregado no ESP32. São passados como parâmetros o vetor de imagens e o vetor com os rótulos das imagens. A função é análoga à descrita anteriormente, sendo uma chamada recursiva da função mostrada no Código 17, sendo iterada pelo número total de imagens e retornado um vetor com as predições realizadas para cada uma delas. A função também exibe o tempo levado para toda a execução e a acurácia do modelo.

Os rótulos das imagens foram salvos utilizando o pacote *numpy*, do Python (OLIPHANT, 2006; WALT; COLBERT; VAROQUAUX, 2011), através do comando *save*() durante o treinamento do modelo do *TensorFlow*, visto no Capítulo 5. Para avaliar esse modelo antes e depois de ser embarcado no ESP32, carregamos os rótulos utilizando o comando *load*(), como mostrado na primeira linha do Código 20.

Código 20 – Inicialização da classe de avaliação do modelo

```
1  true_Y = np.load('Outputs_PLAID1_2.npy')
2  tflite_model = 'fallback_quant_30x30_PLAID1_2.tflite'
3  images_path = '/PLAID/30x30/'
4  serial_port = 'COM4'
5  model = ESP32_TFLite_TF_Comparer(serial_port)
6  images = model.load_images(images_path,['.png'])
```

Na segunda e terceira linhas, os caminhos do modelo e das imagens, respectivamente, são declarados. A variável presente na quarta linha deve indicar a porta serial que o ESP32 está utilizando. Por fim, na quarta e quinta linhas, respectivamente, a classe desenvolvida é instanciada (passando como parâmetro a porta serial), e as imagens são carregadas.

Por fim, como mostrado no Código 21, as funções para realizar a predição do modelo tanto antes de ser embarcado, como mostrado na primeira linha, quanto após ter sido embarcado, mostrado na segunda linha, são chamadas. Após a predição no ESP32, a porta serial deve ser fechada, como mostrado na terceira linha.

Código 21 – Funcionamento da classe de avaliação do modelo

```
1 tflite_predictions = model.evaluate_TFLite(tflite_model, images, true_Y)
2 esp32_predictions = model.evaluate_ESP32(images, true_Y)
3 model.close_serial()
```

Após o envio de todas as 1793 imagens do conjunto de dados e a recepção das inferências para cada uma delas, evento com duração total de 2 horas e 5 minutos, foi calculada uma acurácia de 98,55%, como mostrado na Tabela 37. Ao analisarmos o tempo de latência, ou seja, o tempo em que o dispositivo embarcado levou para fazer a inferência, obtivemos uma média de 4077 ms.

O documento *Jupyter Notebook* deste experimento encontra-se disponível no seguinte repositório: <https://github.com/201110008710/NILM_ESP32/tree/master/ESP32_Avaliacao_PLAID/ESP32_TFLite_TF_Comparer.ipynb>

Tabela 37 – Relatório de classificação do modelo carregado no ESP32 (PLAID1+2)

Classes	**Precisão**	**Revocação**	**Medida-F**	**Suporte**[a]
Air Conditioner (00)	99,02%	97,60%	98,31%	208
Compact Fluorescent Lamp (01)	100,00%	99,55%	99,77%	220
Fan (02)	99,05%	99,52%	99,29%	210
Fridge (03)	98,84%	94,44%	96,59%	90
Hairdryer (04)	99,59%	98,79%	99,19%	248
Heater (05)	96,59%	100,00%	98,27%	85
Incandescent Light Bulb (06)	98,01%	100,00%	99,00%	148
Laptop (07)	95,39%	100,00%	97,64%	207
Microwave (08)	99,57%	100,00%	99,78%	229
Vacuum (09)	98,65%	100,00%	99,32%	73
Washing Machine (10)	96,97%	85,33%	90,78%	75
Acurácia			98,55%	1793
Média Macro[b]	98,34%	97,75%	97,99%	1793
Média Ponderada[c]	98,57%	98,55%	98,53%	1793

[a] Suporte é o número de instâncias verdadeiras por cada classe.
[b] Uma média macro calcula a métrica independentemente para cada classe, obtendo em seguida a média, logo trata todas as classes de forma igualitária.
[c] A média ponderada é calculada utilizando os suportes como pesos.

6.6 Considerações Finais do Capítulo

Comparando os resultados mostrados anteriormente com a avaliação do modelo quantizado antes de ser implementado no ESP32, os resultados foram iguais, ou seja, não houve perda de acurácia após embarcarmos a RNC. A latência obtida, de 4074 ms, atende aos requisitos do

sistema, pois em termos práticos, podemos encará-la como o atraso entre um dispositivo elétrico ser ligado e ser reconhecido.

Dessa forma, por não ter tido alteração nos resultados das métricas de avaliação no processo de quantização e nem após a rede ter sido embarcado, como visto nas Tabelas 34, 36 e 37, podemos deduzir que caso tivéssemos dividido o conjunto de dados utilizando o método de validação cruzada *leave-one-out*, os resultados seriam os mesmos que obtivemos ao avaliarmos o modelo proposto para o conjunto de dados PLAID1+2, no Capítulo 5 e mostrados na Tabela 30. Logo, a comparação dos resultados deste trabalho com os trabalhos relacionados, feita na Tabela 15, continua sendo válida.

7

Conclusões

Este trabalho trouxe como inovação o uso de uma Rede Neural Convolucional embarcada em um dispositivo de baixo custo (ESP32), para realizar a classificação de dispositivos elétricos, contribuindo para o estudo da área de Monitoramento Não-Intrusivo de Cargas. Desta forma, esperamos contribuir, indiretamente, para a redução do consumo elétrico por meio de um uso mais consciente da energia elétrica.

Imagens binárias da trajetória V-I foram utilizadas como característica de distinção, visto que a mesma apresentou bons resultados ao replicarmos os estudos de outros autores. Além disso, todos os nossos experimentos foram realizados utilizando os conjunto de dados PLAID1, PLAID2 e PLAID1+2, com mais de uma métrica de avaliação, a fim de tornar os nossos resultados comparáveis com os de outros estudos. Os códigos-fonte também foram disponibilizados[1] a fim de tornarmos o estudo reprodutível.

O modelo da Rede Neural Convolucional que treinamos utilizando a validação cruzada do tipo *Leave-One-Out* obteve uma macro média da medida-F de 74,76% para o PLAID1, 56,48% para o PLAID2 e 73,97% para o PLAID1+2. Investigando essas medidas-F por classes, percebemos que o modelo apresentou bons resultados para a maioria dos eletrodomésticos no PLAID1 e PLAID1+2, ou seja, quanto maior o número de instâncias do conjunto de dados, melhores foram os resultados.

Para aplicarmos o modelo no dispositivo embarcado, foi necessário quantizá-lo, pois originalmente não havia espaço de memória suficiente no ESP32. Após a quantização (tipo *fallback* de ponto flutuante) do nosso modelo do *TensorFlow Lite*, percebemos que não houve perda de acurácia e conseguimos embarcá-lo. Com esse mesmo modelo já embarcado, realizamos as inferências de todas as 1793 amostras da união dos conjuntos de dados PLAID1 e PLAID2 e obtivemos uma acurácia de 98,55%.

[1] <https://github.com/201110008710/NILM_ESP32>

Dessa forma, mostramos que a classificação de dispositivos elétricos para o problema da desagregação de cargas pode ser feita em um dispositivo embarcado, utilizando a trajetória da tensão-corrente como característica de distinção e a Rede Neural Convolucional como classificador, obtendo baixa latência (~4s) e alta acurácia (>98%).

7.1 Contribuições

Com os novos experimentos realizados, utilizando tanto o PLAID1, quanto o PLAID2 e o PLAID1+2, que são conjuntos de dados populares para o problema, contribuímos reproduzindo a análise de alguns classificadores e mostrando que a escolha da Rede Neural Convolucional para o problema da classificação de cargas elétricas é bastante viável, visto que a mesma obteve excelentes resultados em comparação aos demais classificadores.

De forma análoga, contribuímos também comparando algumas características de distinção dessas cargas, sendo que a imagem binária da trajetória V-I foi a que obteve, individualmente, melhores resultados, ratificando os resultados mostrados em alguns trabalhos relacionados. Além das citadas anteriormente, consideramos que a nossa principal contribuição foi o uso da Rede Neural Convolucional embarcada em um dispositivo embarcado de baixo custo para a classificação dos dispositivos elétricos.

Outra contribuição é termos uma solução embarcada usando o ESP32, o que incluiu: a seleção de uma arquitetura da rede neural convolucional, sendo escolhida a que obteve o melhor resultado após a realização de vários experimentos com parâmetros diferentes; todo o processo de quantização da rede neural utilizando o TensorFlow Lite para microcontroladores, processo pelo qual conseguimos uma redução de aproximadamente 73% no tamanho da rede, possibilitando que a mesma fosse embarcada; e a programação do ESP32 para que o mesmo recebesse as imagens da trajetória V-I e retornasse as respectivas predições usando a interface serial.

Por fim, contribuímos com a publicação de um estudo preliminar que focava no desenvolvimento de um medidor inteligente de potência, cujo artigo foi intitulado *A Non-intrusive Approach for Smart Power Meter* na décima sexta versão da *IEEE International Conference on Industrial Informatics* (INDIN), de 2018, com *qualis* B1, disponível eletronicamente em: <https://ieeexplore.ieee.org/document/8471960/>.

Um outro artigo, cujo título é *Implementation of Convolutional Neural Network on ESP32 for Appliance Classification*, foi submetido ao periódico *Energy and Buildings*[2] (ISSN: 0378-7788), que ainda está em análise pelos revisores.

[2] <https://www.journals.elsevier.com/energy-and-buildings>

7.2 Dificuldades e Limitações

Devido a capacidade limitada do dispositivo embarcado escolhido, não conseguimos desenvolver um modelo mais complexo para a Rede Neural Convolucional, o que poderia melhorar ainda mais os resultados sem comprometer, possivelmente, o tempo de inferência.

Uma outra dificuldade é quanto ao desbalanceamento das classes do conjunto de dados PLAID, que pode ter afetado as acurácias do modelo proposto e dos testes realizados.

7.3 Trabalhos Futuros

Para trabalhos futuros, recomendamos novos testes utilizando outros conjuntos de dados, para fins de comparação, e outros dispositivos embarcados, visto que a complexidade da rede pode ser aumentada e o tempo de inferência pode ser reduzido usando dispositivos mais potentes.

Escolhendo dispositivos de maior potência, é interessante verificar também a influência do aumento da dimensão da imagem binária da trajetória V-I na acurácia e no tempo de latência. Além disso, é interessante obter a relação entre os parâmetros de rede × tamanho × tempo de inferência.

Uma outra boa possibilidade seria acoplar sensores de tensão e corrente ao dispositivo embarcado e realizar a desagregação e a classificação das cargas neste mesmo dispositivo, que se tornaria uma solução viável e completa para o monitoramento não-intrusivo de cargas.

Para as classes que obtiveram resultados inferiores na classificação feita pela RNC, pode ser avaliada a utilização de técnicas de distinção agregadas, como a trajetória V-I e harmônicas juntas, por exemplo. Poderia também ser feita uma comparação da RNC com outros classificadores, classe a classe, para verificar se as dificuldades são inerentes a classe ou ao classificador utilizado.

Apesar dos conjuntos de dados já existentes e disponíveis, seria interessante utilizar o conjunto de dados brasileiro nilmbr, disponível no seguinte repositório: <https://github.com/wesleyangelino/nilmbr>, ou até criar um novo conjunto de dados brasileiro.

Um outro classificador que pode ser utilizado para o problema, principalmente quando utilizado somente o PLAID2, é o *gcForecast* (ZHOU; FENG, 2017)[3], que é uma alternativa de rede neural profunda capaz de funcionar bem mesmo quando há poucos dados de treinamento da rede.

[3] Disponível no seguinte repositório: <https://github.com/kingfengji/gcForest>

Referências

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>. Citado na página 52.

ABADI, M. et al. Tensorflow: A system for large-scale machine learning. In: *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*. USA: USENIX Association, 2016. (OSDI'16), p. 265–283. ISBN 9781931971331. Citado na página 53.

ABUBAKAR, I. et al. An overview of non-intrusive load monitoring methodologies. In: *2015 IEEE Conference on Energy Conversion (CENCON)*. [S.l.: s.n.], 2015. p. 54–59. Citado na página 38.

ADABI, A.; MANOVI, P.; MANTEY, P. Seads: A modifiable platform for real time monitoring of residential appliance energy consumption. In: *2015 Sixth International Green and Sustainable Computing Conference (IGSC)*. [S.l.: s.n.], 2015. p. 1–4. Citado 2 vezes nas páginas 61 e 72.

ADABI, A.; MANOVI, P.; MANTEY, P. Cost-effective instrumentation via nilm to support a residential energy management system. In: *2016 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.: s.n.], 2016. p. 107–110. ISSN 2158-4001. Citado 7 vezes nas páginas 67, 68, 69, 70, 71, 80 e 100.

ADIATMOKO, M. F. et al. Smart meter based on time series modify and constructive backpropagation neural network. In: *2017 4th International Conference on Information Technology, Computer, and Electrical Engineering (ICITACEE)*. [S.l.: s.n.], 2017. p. 147–153. Citado 9 vezes nas páginas 63, 64, 67, 68, 69, 70, 71, 80 e 100.

AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA. *Resolução Normativa Nº414/2010*. [S.l.], 2012. 202 p. Citado na página 33.

AGÊNCIA NACIONAL DE ENERGIA ELÉTRICA. *Consumidor cativo*. ANEEL, 2019. Disponível em: <http://www.aneel.gov.br/busca?p_p_id=101&p_p_lifecycle=0&p_p_state=maximized&p_p_mode=view&_101_struts_action=/asset_publisher/view_content&_101_returnToFullPageURL=/web/guest/busca&_101_assetEntryId=15046283&_101_type=content&_101_groupId=656835&_101_urlTitle=consumidor-cativo&inheritRedirect=true>. Acesso em: 09 jun. 2019. Citado na página 23.

AL-RFOU, R. et al. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, maio 2016. Disponível em: <http://arxiv.org/abs/1605.02688>. Citado na página 78.

ALEXANDER, C.; SADIKU, M. *Fundamentos de circuitos elétricos*. 4ª. ed. [S.l.]: McGraw Hill Higher Education, 2008. ISBN 9780071284417. Citado 3 vezes nas páginas 30, 31 e 33.

ALVES, G. *Entendendo Redes Convolucionais (CNNs)*. Medium, 2018. Disponível em: <https://medium.com/neuronio-br/entendendo-redes-convolucionais-cnns-d10359f21184>. Acesso em: 17 dez. 2019. Citado 4 vezes nas páginas 46, 47, 48 e 49.

ARRACHMAN, S. R. et al. Smart meter based on time series modify and extreme learning machine. In: *2017 2nd International Conference on Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*. [S.l.: s.n.], 2017. p. 86–92. Citado 8 vezes nas páginas 64, 67, 68, 69, 70, 71, 80 e 100.

ATANGANA, A. *Fractional Operators with Constant and Variable Order with Application to Geo-hydrology*. [S.l.]: Elsevier Science, 2017. ISBN 9780128097960. Citado na página 46.

ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. *Computer Networks*, v. 54, n. 1, p. 2787 – 2805, 2010. ISSN 1389-1286. Citado na página 54.

BAETS, L. D. et al. Handling imbalance in an extended plaid. *2017 Sustainable Internet and ICT for Sustainability (SustainIT)*, p. 1–5, 2017. Citado 2 vezes nas páginas 51 e 99.

BAETS, L. D. et al. Appliance classification using vi trajectories and convolutional neural networks. *Energy and Buildings*, v. 158, p. 32 – 36, 2017. ISSN 0378-7788. Citado 22 vezes nas páginas 9, 11, 65, 72, 77, 78, 79, 80, 81, 84, 85, 89, 90, 91, 93, 94, 95, 97, 99, 100, 101 e 133.

BANERJEE, R. Development of pc based transient current analysis system using microcontroller and hall effect sensor. *International Journal of Engineering Research and General Science*, v. 3, p. 321–326, 01 2015. Citado na página 38.

BAPTISTA, D. et al. Implementation strategy of convolution neural networks on field programmable gate arrays for appliance classification using the voltage and current (v-i) trajectory. *Energies*, v. 11, n. 9, 2018. ISSN 1996-1073. Citado 21 vezes nas páginas 11, 65, 66, 67, 68, 69, 70, 71, 80, 81, 85, 88, 89, 90, 91, 93, 95, 97, 99, 100 e 101.

BARAI, G. R.; KRISHNAN, S.; VENKATESH, B. Smart metering and functionalities of smart meters in smart grid - a review. In: *2015 IEEE Electrical Power and Energy Conference (EPEC)*. [S.l.: s.n.], 2015. p. 138–145. Citado na página 24.

BARSIM, K. S.; MAUCH, L.; YANG, B. Neural network ensembles to real-time identification of plug-level appliance measurements. *CoRR*, abs/1802.06963, 2018. Disponível em: <http://arxiv.org/abs/1802.06963>. Citado na página 65.

BARSOCCHI, P. et al. Smart meter led probe for real-time appliance load monitoring. In: *SENSORS, 2014 IEEE*. [S.l.: s.n.], 2014. p. 1451–1454. ISSN 1930-0395. Citado 8 vezes nas páginas 60, 67, 68, 69, 70, 71, 80 e 100.

BENCHOFF, B. *New Chip Alert: The ESP8266 WiFi Module (It's $5)*. 2017. Disponível em: <https://hackaday.com/2014/08/26/new-chip-alert-the-esp8266-wifi-module-its-5/>. Acesso em: 15 jun. 2020. Citado na página 54.

BIANSOONGNERN, S.; PLUNGKLANG, B. Non-intrusive appliances load monitoring (nilm) for energy conservation in household with low sampling rate. *Procedia Computer Science*, v. 86, p. 172 – 175, 2016. ISSN 1877-0509. 2016 International Electrical Engineering Congress, iEECON2016, 2-4 March 2016, Chiang Mai, Thailand. Citado 9 vezes nas páginas 62, 67, 68, 69, 70, 71, 72, 80 e 100.

BIANSOONGNERN, S.; PLUNGKLANG, B. Nonintrusive load monitoring (nilm) using an artificial neural network in embedded system with low sampling rate. In: *2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)*. [S.l.: s.n.], 2016. p. 1–4. Citado 9 vezes nas páginas 62, 67, 68, 69, 70, 71, 72, 80 e 100.

BOYLESTAD, R. *Electronic Devices and Circuit Theory, 11e*. [S.l.]: Prentice-Hall, 2013. ISBN 9788564574212. Citado na página 26.

BRAGA, A. de P. *Redes neurais artificiais: teoria e aplicações*. 2. ed. Rio de Janeiro, RJ, Brasil: LTC Editora, 2007. ISBN 9788521615644. Citado 4 vezes nas páginas 40, 41, 42 e 43.

C, S.; Kumar, M.; K, I. Design and implementation of non-intrusive load monitoring using machine learning algorithm for appliance monitoring. In: *2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS)*. [S.l.: s.n.], 2019. p. 1–6. Citado na página 26.

CENTRAL INTELLIGENCE AGENCY. *The World Factbook 2016-17*. 2016. Disponível em: <https://www.cia.gov/library/publications/the-world-factbook/geos/br.html>. Acesso em: 05 jun. 2019. Citado na página 23.

CHANG, H.-H.; WIRATHA, P. W.; CHEN, N. A non-intrusive load monitoring system using an embedded system for applications to unbalanced residential distribution systems. *Energy Procedia*, v. 61, p. 146 – 150, 2014. ISSN 1876-6102. International Conference on Applied Energy, ICAE2014. Citado 8 vezes nas páginas 60, 67, 68, 69, 70, 71, 80 e 100.

CHRISTO, E. da S. *Previsão de Potência Reativa*. Tese (Doutorado) — Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brasil, julho 2005. Citado 3 vezes nas páginas 31, 32 e 33.

CPFL ENERGIA. *Cartilha de utilização consciente da energia elétrica*. CPFL Energia, 2016. Disponível em: <https://www.cpfl.com.br/energias-sustentaveis/eficiencia-energetica/uso-consciente/calculo-de-consumo/Documents/cartilha-da-utilizacao-consciente-de-energia-eletrica.pdf>. Acesso em: 10 jun. 2019. Citado na página 25.

CYBENKO, G. *Continuous Valued Neural Networks with Two Hidden Layers are Sufficient*. [S.l.: s.n.], 1988. Citado na página 43.

CYBENKO, G. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, v. 2, n. 4, p. 303–314, Dec 1989. ISSN 1435-568X. Citado na página 43.

DALAKOV, G. *The Modem of Dennis Hayes and Dale Heatherington*. 2020. Disponível em: <https://history-computer.com/ModernComputer/Basis/modem.html>. Acesso em: 15 jun. 2020. Citado na página 54.

DATA SCIENCE ACADEMY. *O Que é o TensorFlow Machine Intelligence Platform?* 2018. Disponível em: <datascienceacademy.com.br/blog/o-que-e-o-tensorflow-machine-intelligence-platform/>. Acesso em: 13 jun. 2020. Citado na página 54.

DEAN, J. et al. Large scale distributed deep networks. In: PEREIRA, F. et al. (Ed.). *Advances in Neural Information Processing Systems 25*. [S.l.]: Curran Associates, Inc., 2012. p. 1223–1231. Citado na página 52.

DENKER, J. S. et al. Neural network recognizer for hand-written zip code digits. In: *Advances in neural information processing systems*. [S.l.: s.n.], 1989. p. 323–331. Citado na página 45.

EBERMAM, E.; KROHLING, R. A. Uma introdução compreensiva às redes neurais convolucionais: Um estudo de caso para reconhecimento de caracteres alfabéticos. *Revista de Sistemas de Informação da FSMA*, v. 22, p. 49–59, 2018. ISSN 1983-5604. Citado na página 48.

EHRHARDT-MARTINEZ, K.; DONNELLY, K.; LAITNER, J. Advanced metering initiatives and residential feedback programs: a meta-review for household electricity-saving opportunities. *American Council for an Energy-Efficient Economy*, 01 2010. Citado na página 25.

EMPRESA DE PESQUISA ENERGÉTICA. *Anuário Estatístico de Energia Elétrica 2018 (ano base 2017)*. Rio de Janeiro, RJ, Brasil: EPE, 2018. 249 p. Citado na página 23.

ENERDATA. *Electricity domestic consumption*. Enerdata, 2019. Disponível em: <https://yearbook.enerdata.net/electricity/electricity-domestic-consumption-data.html>. Acesso em: 05 jun. 2019. Citado na página 23.

ESEN, . A.; BAYRAK, M. Does more energy consumption support economic growth in net energy-importing countries. *Journal of Economics, Finance and Administrative Science*, scielo, v. 22, p. 75 – 98, 06 2017. ISSN 2077-1886. Citado na página 22.

FERRÁNDEZ-PASTOR, F. J. et al. Interpreting human activity from electrical consumption data using reconfigurable hardware and hidden markov models. *Journal of Ambient Intelligence and Humanized Computing*, v. 8, n. 4, p. 469–483, Aug 2017. ISSN 1868-5145. Citado 8 vezes nas páginas 63, 67, 68, 69, 70, 71, 80 e 100.

FILHO, J. M. *Instalações elétricas industriais*. 7ª. ed. Rio de Janeiro, RJ, Brasil: Livros Técnicos e Científicos, 2007. ISBN 9788521615200. Citado na página 33.

FLORENCIO, F. D. A. et al. Intrusion detection via mlp neural network using an arduino embedded system. In: *2018 VIII Brazilian Symposium on Computing Systems Engineering (SBESC)*. [S.l.: s.n.], 2018. p. 190–195. ISSN 2324-7894. Citado na página 43.

FREITAS, E. de. *Primeira Revolucao Industrial*. Brasil Escola, 2019. Disponível em: <https://brasilescola.uol.com.br/geografia/primeira-revolucao-industrial.htm>. Acesso em: 05 jun. 2019. Citado na página 22.

FUKUSHIMA, K. Self-organization of a neural network which gives position-invariant response. In: *Proceedings of the 6th International Joint Conference on Artificial Intelligence - Volume 1*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1979. (IJCAI'79), p. 291–293. ISBN 0934613478. Citado na página 45.

GAO, J. et al. Plaid: A public dataset of high-resolution electrical appliance measurements for load identification research: Demo abstract. In: *Proceedings of the 1st ACM Conference on Embedded Systems for Energy-Efficient Buildings*. New York, NY, USA: ACM, 2014. (BuildSys '14), p. 198–199. ISBN 978-1-4503-3144-9. Citado na página 51.

Gao, J. et al. A feasibility study of automated plug-load identification from high-frequency measurements. In: *2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. [S.l.: s.n.], 2015. p. 220–224. Citado 13 vezes nas páginas 72, 75, 76, 77, 81, 82, 83, 84, 85, 89, 92, 94 e 133.

GERSHENSON, C. Artificial neural networks for beginners. *arXiv preprint cs/0308031*, 2003. Citado na página 43.

GLOBALDATA. *Global smart meter market expected to reach $10.4bn by 2022*. Globaldata, 2018. Disponível em: <https://www.globaldata.com/global-smart-meter-market-expected-reach-10-4bn-2022>. Acesso em: 10 jun. 2019. Citado na página 26.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. Citado 4 vezes nas páginas 46, 47, 48 e 49.

GU, J. et al. Recent advances in convolutional neural networks. *Pattern Recognition*, v. 77, p. 354 – 377, 2018. ISSN 0031-3203. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0031320317304120>. Citado na página 48.

HART, G. W. Nonintrusive appliance load monitoring. *Proceedings of the IEEE*, v. 80, n. 12, p. 1870–1891, Dec 1992. ISSN 0018-9219. Citado 2 vezes nas páginas 26 e 36.

HAYKIN, S. *Neural Networks and Learning Machines*. Upper Saddle River, NJ, EUA: Pearson Education, 2011. ISBN 9780133002553. Citado 4 vezes nas páginas 39, 40, 44 e 45.

HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, National Acad Sciences, v. 79, n. 8, p. 2554–2558, 1982. Citado na página 40.

HUANG, X. et al. An online non-intrusive load monitoring method based on hidden markov model. *Journal of Physics: Conference Series*, IOP Publishing, v. 1176, p. 042036, mar 2019. Citado 8 vezes nas páginas 66, 67, 68, 69, 70, 71, 80 e 100.

HUBEL, D. H.; WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, v. 160, n. 1, p. 106–154, 1962. Citado na página 44.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. Citado na página 75.

IAZZETTA, F. *Áudio Digital*. 2019. Disponível em: <http://www2.eca.usp.br/prof/iazzetta/tutor/audio/a_digital/a_digital.html>. Acesso em: 10 jun. 2019. Citado na página 33.

INTERNATIONAL ENERGY AGENCY. *Technology Roadmap - Energy Efficient Building Envelopes*. [S.l.]: IEA, 2013. 68 p. Citado na página 24.

INTERNATIONAL ENERGY AGENCY. *Global EV Outlook 2018*. [S.l.]: OECD, 2018. 141 p. Citado na página 23.

INTERNATIONAL ENERGY AGENCY. *World Energy Outlook 2019*. 2019. Disponível em: <https://www.iea.org/reports/world-energy-outlook-2019>. Acesso em: 19 out. 2020. Citado na página 22.

JONETZKO, R. et al. High frequency non-intrusive electric device detection and diagnosis. In: *2015 International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*. [S.l.: s.n.], 2015. p. 1–8. Citado 9 vezes nas páginas 61, 67, 68, 69, 70, 71, 72, 80 e 100.

JOY-IT. *NodeMCU ESP32*. 2020. Disponível em: <https://joy-it.net/en/products/SBC-NodeMCU-ESP32>. Acesso em: 13 jun. 2020. Citado na página 55.

KAHAN, S.; PAVLIDIS, T.; BAIRD, H. S. On the recognition of printed characters of any font and size. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, n. 2, p. 274–288, 1987. Citado na página 45.

KAHL, M. et al. Whited – a worldwide household and industry transient energy data set. In: *3rd International Workshop on Non-Intrusive Load Monitoring*. [S.l.: s.n.], 2016. Citado na página 77.

KHAN, A. R. et al. Load forecasting, dynamic pricing and dsm in smart grid: A review. *Renewable and Sustainable Energy Reviews*, v. 54, p. 1311 – 1322, 2016. ISSN 1364-0321. Citado 2 vezes nas páginas 22 e 24.

KLEMENJAK, C.; GOLDSBOROUGH, P. Non-intrusive load monitoring: A review and outlook. *CoRR*, abs/1610.01191, 2016. Citado 2 vezes nas páginas 36 e 38.

KLUYVER, T. et al. Jupyter notebooks – a publishing format for reproducible computational workflows. In: LOIZIDES, F.; SCHMIDT, B. (Ed.). *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. [S.l.], 2016. p. 87 – 90. Citado na página 75.

KOHAVI, R.; PROVOST, F. Glossary of terms. *Editorial for the Special Issue on Applications of Machine Learning and the Knowledge Discovery Process*, Kluwer Academic Publishers, v. 30, n. 2-3, p. 271–274, 1998. Citado na página 49.

KORABLYOV, D. *First steps with ESP32 and TensorFlow Lite for Microcontrollers*. 2020. Disponível em: <https://medium.com/@dmytro.korablyov/first-steps-with-esp32-and-tensorflow-lite-for-microcontrollers-c2d8e238accf>. Acesso em: 17 set. 2020. Citado na página 110.

LATHI, B. *Sinais e Sistemas Lineares*. 2ª. ed. [S.l.]: Bookman, 2007. ISBN 9788560031139. Citado 2 vezes nas páginas 46 e 47.

LECUN, Y. et al. Backpropagation applied to handwritten zip code recognition. *Neural computation*, MIT Press, v. 1, n. 4, p. 541–551, 1989. Citado na página 45.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, IEEE, v. 86, n. 11, p. 2278–2324, 1998. Citado na página 45.

LI, H. et al. Patternnet: Visual pattern mining with deep neural network. In: *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*. New York, NY, USA: Association for Computing Machinery, 2018. (ICMR '18), p. 291–299. ISBN 9781450350464. Citado na página 44.

LI, S.; XU, L. D.; ZHAO, S. The internet of things: A survey. *Information Systems Frontiers*, Kluwer Academic Publishers, USA, v. 17, n. 2, p. 243–259, abr. 2015. ISSN 1387-3326. Citado na página 54.

LINHARES, M. *TensorFlow v1.4: Estimators (Parte 1)*. 2017. Disponível em: <https://medium.com/@mariannelinharesm/tensorflow-v1-4-0-estimators-parte-1-1a58bbfc13ae>. Acesso em: 13 jun. 2020. Citado na página 54.

MAIER, A.; SHARP, A.; VAGAPOV, Y. Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things. In: *2017 Internet Technologies and Applications (ITA)*. [S.l.: s.n.], 2017. p. 143–148. Citado na página 54.

MAKONIN, S.; POPOWICH, F. Nonintrusive load monitoring (NILM) performance evaluation. *Energy Efficiency*, Springer Science and Business Media LLC, v. 8, n. 4, p. 809–814, out. 2014. Citado na página 78.

MAKONIN, S.; POPOWICH, F.; GILL, B. The cognitive power meter: Looking beyond the smart meter. In: *2013 26th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. [S.l.: s.n.], 2013. p. 1–5. ISSN 0840-7789. Citado 2 vezes nas páginas 24 e 35.

MAKONIN, S. et al. Inspiring energy conservation through open source metering hardware and embedded real-time load disaggregation. In: *2013 IEEE PES Asia-Pacific Power and Energy Engineering Conference (APPEEC)*. [S.l.: s.n.], 2013. p. 1–6. ISSN 2157-4839. Citado 8 vezes nas páginas 59, 67, 68, 69, 70, 71, 80 e 100.

MAKONIN, S. W. *Real-Time Embedded Low-Frequency Load Disaggregation*. Tese (Doutorado) — School of Computing Science Faculty of Applied Sciences, Simon Fraser University, 2014. Citado na página 60.

MARSLAND, S. *Machine Learning: An Algorithmic Perspective*. 2. ed. [S.l.]: Chapman and Hall/CRC, 2014. ISBN 1466583282. Citado 3 vezes nas páginas 39, 40 e 43.

MCCULLOCH, W. S.; PITTS, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, v. 5, n. 4, p. 115–133, Dec 1943. ISSN 1522-9602. Citado 2 vezes nas páginas 39 e 40.

MERTZ, D. *The GNU Text Utilities*. 2020. Disponível em: <https://gnosis.cx/publish/programming/text_utils.html>. Acesso em: 17 set. 2020. Citado na página 104.

MINSKY, M.; PAPERT, S. Perceptron: an introduction to computational geometry. *The MIT Press, Cambridge, expanded edition*, v. 19, n. 88, p. 2, 1969. Citado na página 40.

NAGHIBI, B.; DEILAMI, S. Non-intrusive load monitoring and supplementary techniques for home energy management. In: *2014 Australasian Universities Power Engineering Conference (AUPEC)*. [S.l.: s.n.], 2014. p. 1–5. Citado na página 38.

NEURAL NETWORK DISTILLER. *Quantization Algorithms*. 2020. Disponível em: <https://intellabs.github.io/distiller/algo_quantization.html>. Acesso em: 17 set. 2020. Citado na página 105.

NILSSON, J. W.; RIEDEL, S. A. *Circuitos Elétricos*. 8ª. ed. São Paulo, SP, Brasil: Pearson Prentice Hall, 2009. ISBN 9788576051596. Citado 6 vezes nas páginas 29, 30, 31, 32, 33 e 34.

OLIPHANT, T. E. *A guide to NumPy*. [S.l.]: Trelgol Publishing USA, 2006. v. 1. Citado na página 116.

ORGANIZAÇÃO DAS NAÇÔES UNIDAS. *The 17 Goals*. ONU, 2019. Disponível em: <https://www.globalgoals.org>. Acesso em: 07 jun. 2019. Citado na página 24.

PARVAT, A. et al. A survey of deep-learning frameworks. In: *2017 International Conference on Inventive Systems and Control (ICISC)*. [S.l.: s.n.], 2017. p. 1–7. Citado na página 53.

PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, v. 12, p. 2825–2830, 2011. Citado na página 75.

PETERSEN, K. et al. Systematic mapping studies in software engineering. In: *EASE*. [S.l.: s.n.], 2008. v. 8, p. 68–77. Citado 2 vezes nas páginas 56 e 58.

PLATAFORMA AGENDA 2030. *A Agenda 2030 para o Desenvolvimento Sustentável*. 2020. Disponível em: <http://www.agenda2030.com.br/sobre/>. Acesso em: 17 set. 2020. Citado na página 24.

RASCHKA, S.; MIRJALILI, V. *Python Machine Learning, 2nd Ed.* 2. ed. Birmingham, UK: Packt Publishing, 2017. ISBN 978-1787125933. Citado na página 44.

REILLY, J. Energy and development in emerging countries. *Revue d'économie du développement*, CAIRN, v. 23, n. HS, p. 19, 2015. Citado na página 22.

RIDI, A.; GISLER, C.; HENNEBERT, J. A survey on intrusive load monitoring for appliance recognition. In: *2014 22nd International Conference on Pattern Recognition*. [S.l.: s.n.], 2014. p. 3702–3707. ISSN 1051-4651. Citado 2 vezes nas páginas 34 e 36.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, American Psychological Association, v. 65, n. 6, p. 386, 1958. Citado na página 40.

ROSENBLATT, F. *Perceptions and the theory of brain mechanisms*. [S.l.]: Spartan books, 1962. Citado na página 40.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, v. 323, n. 6088, p. 533–536, Oct 1986. ISSN 1476-4687. Citado 2 vezes nas páginas 40 e 43.

SEARS, F. et al. *Física I: Mecânica*. 14ª. ed. São Paulo, SP, Brasil: Pearson Education do Brasil, 2008. ISBN 9788588639300. Citado na página 30.

SHARMA, S. *Explained: Deep Learning in Tensorflow — Chapter 1*. 2019. Disponível em: <https://dimensionless.in/whats-new-in-tensorflow-2/>. Acesso em: 13 jun. 2020. Citado na página 53.

SILVA, D. C. M. da. *Potência de uma força*. Mundo Educação, 2019. Disponível em: <https://mundoeducacao.uol.com.br/fisica/potencia-uma-forca.htm>. Acesso em: 13 jun. 2019. Citado na página 29.

SILVA, I. da; SPATTI, D.; FLAUZINO, R. *Redes Neurais Artificiais para Engenharia e Ciências Aplicadas: Curso Prático*. 1. ed. São Paulo, SP, Brasil: ArtLiber Editora, 2010. ISBN 9788588098534. Citado 3 vezes nas páginas 41, 42 e 43.

SINGH, K. J.; KAPOOR, D. S. Create your own internet of things: A survey of iot platforms. *IEEE Consumer Electronics Magazine*, v. 6, n. 2, p. 57–68, 2017. Citado na página 54.

SMART ENERGY INTERNATIONAL. *Global trends in smart metering*. Smart Energy International, 2018. Disponível em: <https://www.smart-energy.com/magazine-article/global-trends-in-smart-metering/>. Acesso em: 10 jun. 2019. Citado na página 26.

SMART ENERGY INTERNATIONAL. *Smart grid development in Brazil and South American counterparts*. Smart Energy International, 2018. Disponível em: <https://www.smart-energy.com/industry-sectors/policy-regulation/smart-grid-brazil-south-america-frost-sullivan>. Acesso em: 10 jun. 2019. Citado na página 26.

SOUSA, R. G. *As lâmpadas de Edison*. Mundo Educação, 2017. Disponível em: <https://brasilescola.uol.com.br/historiag/segunda-revolucao-industrial.htm>. Acesso em: 05 jun. 2019. Citado na página 22.

SOUSA, R. G. *Segunda Revolução Industrial*. Brasil Escola, 2019. Disponível em: <https://brasilescola.uol.com.br/historiag/segunda-revolucao-industrial.htm>. Acesso em: 05 jun. 2019. Citado na página 22.

SULTANEM, F. Using appliance signatures for monitoring residential loads at meter panel level. *IEEE Transactions on Power Delivery*, v. 6, n. 4, p. 1380–1385, Oct 1991. ISSN 0885-8977. Citado na página 36.

SYAI'IN, M. et al. Smart meter based on time series modify and neural network for online energy monitoring. *International Journal of Mechanical and Mechatronics Engineering*, v. 18, p. 46–53, 06 2018. Citado 9 vezes nas páginas 64, 67, 68, 69, 70, 71, 72, 80 e 100.

TAVARES, H. et al. A non-intrusive approach for smart power meter. In: *2018 IEEE 16th International Conference on Industrial Informatics (INDIN)*. [S.l.: s.n.], 2018. p. 605–610. ISSN 2378-363X. Citado na página 24.

TENSORFLOW. *Comece com microcontroladores*. 2020. Disponível em: <https://www.tensorflow.org/lite/microcontrollers/get_started>. Acesso em: 17 set. 2020. Citado na página 108.

TENSORFLOW. *Inferência do TensorFlow Lite*. 2020. Disponível em: <https://www.tensorflow.org/lite/guide/inference>. Acesso em: 17 set. 2020. Citado na página 114.

TENSORFLOW. *Quantização inteira pós-treinamento*. 2020. Disponível em: <https://www.tensorflow.org/lite/performance/post_training_integer_quant?hl=pt-br>. Acesso em: 17 set. 2020. Citado 2 vezes nas páginas 105 e 106.

TENSORFLOW. *tf.lite.Optimize*. 2020. Disponível em: <https://www.tensorflow.org/api_docs/python/tf/lite/Optimize>. Acesso em: 17 set. 2020. Citado na página 107.

UNITES STATES ENERGY INFORMATION ADMINISTRATION. *Electricity and the Environment*. EIA, 2018. Disponível em: <https://www.eia.gov/energyexplained/electricity/electricity-and-the-environment.php>. Acesso em: 05 jun. 2019. Citado na página 23.

UNITES STATES ENVIRONMENTAL PROTECTION AGENCY. *Reduce the Environmental Impact of Your Energy Use*. EPA, 2019. Disponível em: <https://www.epa.gov/energy/reduce-environmental-impact-your-energy-use>. Acesso em: 05 jun. 2019. Citado na página 24.

USMAN, A.; SHAMI, S. H. Evolution of communication technologies for smart grid applications. *Renewable and Sustainable Energy Reviews*, v. 19, p. 191 – 199, 2013. ISSN 1364-0321. Citado na página 24.

WALT, S. V. D.; COLBERT, S. C.; VAROQUAUX, G. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, IEEE Computer Society, v. 13, n. 2, p. 22, 2011. Citado na página 116.

WEISS, M. et al. Leveraging smart meter data to recognize home appliances. In: *2012 IEEE International Conference on Pervasive Computing and Communications*. [S.l.: s.n.], 2012. p. 190–197. Citado 9 vezes nas páginas 59, 67, 68, 69, 70, 71, 72, 80 e 100.

WEISSTEIN, E. W. *Root-Mean-Square*. MathWorld - A Wolfram Web Resource, 2017. Disponível em: <https://mathworld.wolfram.com/Root-Mean-Square.html>. Acesso em: 11 jun. 2019. Citado na página 33.

WIDROW, B.; HOFF, M. E. Neurocomputing: Foundations of research. In: ANDERSON, J. A.; ROSENFELD, E. (Ed.). Cambridge, MA, USA: MIT Press, 1988. cap. Adaptive Switching Circuits, p. 123–134. ISBN 0-262-01097-6. Citado na página 43.

WIKIPEDIA. *TensorFlow*. 2020. Disponível em: <https://pt.wikipedia.org/wiki/TensorFlow>. Acesso em: 13 jun. 2020. Citado na página 53.

XU, Y.; MILANOVIĆ, J. V. Artificial-intelligence-based methodology for load disaggregation at bulk supply point. *IEEE Transactions on Power Systems*, v. 30, n. 2, p. 795–803, March 2015. ISSN 0885-8950. Citado na página 26.

ZAFAR, R. et al. Prosumer based energy management and sharing in smart grid. *Renewable and Sustainable Energy Reviews*, v. 82, p. 1675 – 1684, 2018. ISSN 1364-0321. Citado na página 24.

ZEIFMAN, M.; ROTH, K. Nonintrusive appliance load monitoring: Review and outlook. *IEEE Transactions on Consumer Electronics*, v. 57, n. 1, p. 76–84, February 2011. ISSN 0098-3063. Citado 4 vezes nas páginas 26, 35, 36 e 37.

ZHOU; CHELLAPPA. Computation of optical flow using a neural network. In: *IEEE 1988 International Conference on Neural Networks*. [S.l.: s.n.], 1988. p. 71–78 vol.2. Citado na página 48.

ZHOU, Z.-H.; FENG, J. Deep forest: Towards an alternative to deep neural networks. In: *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*. [s.n.], 2017. p. 3553–3559. Disponível em: <https://doi.org/10.24963/ijcai.2017/497>. Citado na página 121.

ZHUANG, M.; SHAHIDEHPOUR, M.; LI, Z. An overview of non-intrusive load monitoring: Approaches, business applications, and challenges. In: *2018 International Conference on Power System Technology (POWERCON)*. [S.l.: s.n.], 2018. p. 4291–4299. Citado na página 37.

ZHUK, A.; BUZOVEROV, E. The impact of electric vehicles on the outlook of future energy system. *IOP Conference Series: Materials Science and Engineering*, IOP Publishing, v. 315, p. 012032, feb 2018. Citado na página 23.

ZOHA, A. et al. Non-intrusive load monitoring approaches for disaggregated energy sensing: A survey. *Sensors*, MDPI AG, v. 12, n. 12, p. 16838–16866, dec 2012. Citado 2 vezes nas páginas 37 e 38.

Apêndices

APÊNDICE A – Guia Básico de Utilização do *Dataset* PLAID

A classe desenvolvida, baseada nos trabalhos de Gao et al. (2015) e Baets et al. (2017b), possibilita a extração das características bem como a plotagem de gráficos e de informações do *dataset*. Foi desenvolvida na linguagem de programação Python e o documento Jupyter Notebook está disponível no seguinte repositório: <https://github.com/201110008710/NILM_ESP32/blob/master/Jupyter_Notebooks/Hugo_Dataset_Graphs_Infos.ipynb>. Dentre as 48 funções disponíveis, as principais para a compreensão do uso básico são listadas e comentadas abaixo:

```
#Inicializa as principais variáveis da classe passando como parâmetro o diretório
    ↪ onde o dataset pode ser encontrado.
__init__(data_path:str)
```

```
#Responsáveis pela leitura dos metadados do PLAID.
read_meta_PLAID1()          #Lê os metadados do PLAID1
read_meta_PLAID2()          #Lê os metadados do PLAID2
read_meta_PLAID1_and_2()    #Lê os metadados do PLAID1+2
```

```
#Faz a leitura das instâncias do dataset a partir dos metadados lidos e salva um
    ↪ arquivo pickle para facilitar uma posterior leitura.
#Parâmetros:
#file_name: nome do arquivo pickle que será salvo.
#last_points: quantidade de pontos, ou amostras, do estado estável que serão lidas.
    ↪ (padrão: 10.000)
#verbose: 1 para ter o log mais detalhado e 0 para não. (padrão: 0)
read_and_save_all_data(file_name:str, npts:int, verbose:int)
```

```
#Retorna a lista com o nome do dispositivo elétrico para cada instância
    ↪ correspondente.
get_appliances(self)
```

```
#Retorna a lista de inteiros correspondentes à classe a qual cada instância
    ↪ corresponde.
get_appliancesIDs()
```

```
#Retorna a lista com os nomes, não repetidos, dos dispositivos elétricos.
#Como são 11 diferentes dispositivos, os nomes deles serão retornados.
get_uniqueAppliances()
```

```
#Retorna a lista com os nomes abreviados, não repetidos, dos dispositivos elétricos.
#Como são 11 diferentes dispositivos, os nomes abreviados deles serão retornados.
get_uniqueAppliances_Short()
```

```
#Retorna a lista de inteiros correspondentes à casa a qual cada instância foi
    ↪ amostrada.
get_housesIDs()
```

```
#Retorna a lista com as casas, não repetidas, onde os dispositivos elétricos foram
    ↪ amostrados.
get_uniqueHouses()
```

```
# Retorna um dicionário com os dispositivos elétricos e o identificador de cada
    ↪ instância pertencente a eles.
get_instances_by_appliances()
```

```
# Retorna um vetor com o número de instâncias para cada dispositivo elétrico, na
    ↪ mesma ordem em que os nomes são listados pela função get_uniqueAppliances().
get_instances_per_appliances()
```

```
#Extrai um período representativo do estado estável para cada instância.
#Lembrete:
#fs/f0: número de amostras por período.
#npts/(fs/f0): número de períodos por número de pontos extraídos.
#Parâmetros:
#fs: frequência de amostragem do dataset. (padrão: 30.000)
#f0: frequência fundamental da rede elétrica. (padrão: 60)
get_rep_period_SS(fs:int, f0:int)
```

```
#Retorna, para cada instância, o respectivo ponto no plano PQ.
#Lembrete:
#fs/f0: número de amostras por período.
#npts/(fs/f0): número de períodos por número de pontos extraídos.
#Parâmetros:
#fs: frequência de amostragem do dataset. (padrão: 30.000)
#f0: frequência fundamental da rede elétrica. (padrão: 60)
get_PQ_SS(fs:int, f0:int)
```

```
#Retorna, para cada instância, as n ordens de harmônicas e a fundamental.
#Lembrete:
#fs/f0: número de amostras por período.
#npts/(fs/f0): número de períodos por número de pontos extraídos.
#Parâmetros:
#order: ordem da maior harmônica que será extraída. (padrão: 21)
get_harmonicsF_SS(order:int)
```

```
#Retorna, para cada instância, um número reduzido de amostras de tensão e corrente.
#Parâmetros:
#bins: número de amostras, ou pontos, desejados para cada instância. (padrão: 20)
get_downSampled_SS(bins:int)
```

```
#Retorna a imagem binária da trajetória V-I para uma determinada instância.
#Parâmetros:
#V: valor da tensão da instância.
#I: valor da corrente da instância.
#width: dimensão, quadrada, da imagem.
#hard_threshold: caso verdadeiro, valores abaixo do 'para' são 0 e os demais 1.
    ↪ (padrão: True)
#para: valor que determina o cut off ou a redução de intensidade da imagem.
    ↪ (padrão: .5)
get_img_from_VI(V:float, I:float, width:int, hard_threshold:bool, para:float)
```

```
#Retorna a imagem binária, para cada instância, da trajetória V-I.
#Faz uma chamada recursiva da função get_img_from_VI(V,I,width,hard_threshold,para)
    ↪ para cada uma das instâncias do dataset.
#Parâmetros:
#width: dimensão, quadrada, da imagem.
#hard_threshold: se verdadeiro, valores abaixo do 'para' são 0 e os demais 1.
    ↪ (padrão: True)
#para: determina o cut off ou a redução de intensidade da imagem. (padrão: 1)
get_imgs_from_VI(width:int, hard_threshold:bool, para:float)
```

```
#Função para reduzir a dimensionalidade da característica utilizando a Análise de
    ↪ Componentes Principais.
#Parâmetros:
#feature: vetor da característica de distinção.
#variation: taxa, em porcentagem, de variação para o componente.
get_PCAs(feature:list, variation:float)
```

```
#Retorna algumas informações do dataset, incluindo os números de instâncias e de
    ↪ diferentes modelos por dispositivo elétrico.
dataset_info()
```

```
#Função salvar as imagens binárias da trajetória V-I em um formato de imagem.
#Parâmetros:
#width: dimensão, quadrada, da imagem.
#extension: extensão do formato da imagem. (padrão: pdf)
#path: diretório onde as imagens serão salvas.
#verbose: 1 para ter o log mais detalhado e 0 para não.
save_imgs(width:int, extension:str, path:str, verbose:int):
```

Para realizar as plotagens dos gráficos, as seguintes funções podem ser utilizadas:

```
#Plota o gráfico do plano P-Q para uma determinada lista de dispositivos elétricos.
#Parâmetros:
#appliancesTypes: lista com os inteiros correspondentes aos dispositivos
    ↪ escolhidos. Caso a lista não seja declarada, todos serão considerados.
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_PQ_SS(appliancesTypes:list, savePath:str, extension:str)
```

```
#Faz uma sub-plotagem com gráficos da corrente para uma determinada lista de
    ↪ dispositivos elétricos.
#Parâmetros:
#nCols: número de gráficos por coluna
#rand: caso verdadeiro, as instâncias dos dispositivos serão aleatórias.
#appliancesTypes: lista com os inteiros correspondentes aos dispositivos
    ↪ escolhidos. Caso a lista não seja declarada, todos serão considerados.
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_rawI_SS(nCols:int, rand:bool, appliancesTypes:list, savePath:str,
    ↪ extension:str)
```

```
#Faz uma sub-plotagem com gráficos da tensão para uma determinada lista de
    ↪ dispositivos elétricos.
#Parâmetros:
#nCols: número de gráficos por coluna
#rand: caso verdadeiro, as instâncias dos dispositivos serão aleatórias.
#appliancesTypes: lista com os inteiros correspondentes aos dispositivos
    ↪ escolhidos. Caso a lista não seja declarada, todos serão considerados.
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_rawV_SS(nCols:int, rand:bool, appliancesTypes:list, savePath:str,
    ↪ extension:str)
```

```
#Faz uma sub-plotagem com gráficos subamostrados da corrente para uma determinada
    ↪ lista de dispositivos elétricos.
#Parâmetros:
#nCols: número de gráficos por coluna
#rand: caso verdadeiro, as instâncias dos dispositivos serão aleatórias.
#bins: número de amostras, ou pontos, desejados para cada instância. (padrão: 20)
#appliancesTypes: lista com os inteiros correspondentes aos dispositivos
    ↪ escolhidos. Caso a lista não seja declarada, todos serão considerados.
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_dSI_SS(nCols:int, rand:bool, bins:int, appliancesTypes:list, savePath:str,
    ↪ extension:str)
```

```
#Faz uma sub-plotagem com gráficos subamostrados da tensão para uma determinada
    ↪ lista de dispositivos elétricos.
#Parâmetros:
#nCols: número de gráficos por coluna
#rand: caso verdadeiro, as instâncias dos dispositivos serão aleatórias.
#bins: número de amostras, ou pontos, desejados para cada instância. (padrão: 20)
#appliancesTypes: lista com os inteiros correspondentes aos dispositivos
    ↪ escolhidos. Caso a lista não seja declarada, todos serão considerados.
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_dSV_SS(nCols:int, rand:bool, bins:int, appliancesTypes:list, savePath:str,
    ↪ extension:str)
```

```
#Plota o gráfico das n primeiras harmônicas para uma determinada instância do
    ↪ dispositivo.
#Parâmetros:
#applianceID: inteiro correspondente à intância do dispositivo
#nHarmonics: número de harmônicas a serem plotadas, incluindo a fundamental.
    ↪ (padrão: 11)
#fs: frequência de amostragem do dataset. (padrão: 30.000)
#f0: frequência fundamental da rede elétrica. (padrão: 60)
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_Harmonics_SS(applianceID:int, nHarmonics:int, fs:int, f0:int, savePath:str,
    ↪ extension:str)
```

```
#Plota o gráfico tridimensional das harmônicas para uma determinada lista de
    ↪ dispositivos elétricos.
#Parâmetros:
#nPoints: número de pontos, por dispositivo elétrico, a ser plotado.
#appliancesTypes: lista com os inteiros correspondentes aos dispositivos
    ↪ escolhidos. Caso a lista não seja declarada, todos serão considerados.
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_3D_Harmonics_SS(nPoints:int, appliancesTypes:list, savePath:str, extension:str)
```

```
#Faz uma sub-plotagem com imagens binárias da trajetória V-I para uma determinada
    ↪ lista de dispositivos elétricos.
#Parâmetros:
#nCols: número de gráficos por coluna
#rand: caso verdadeiro, as instâncias dos dispositivos serão aleatórias.
#width: dimensão, quadrada, da imagem.
#appliancesTypes: lista com os inteiros correspondentes aos dispositivos
    ↪ escolhidos. Caso a lista não seja declarada, todos serão considerados.
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_binVI_SS(nCols:int, rand:bool, width:int, appliancesTypes:list, savePath:str,
    ↪ extension:str)
```

```
#Faz uma sub-plotagem com os gráficos da trajetória V-I para uma determinada lista
    ↪ de dispositivos elétricos.
#Parâmetros:
#nCols: número de gráficos por coluna
#rand: caso verdadeiro, as instâncias dos dispositivos serão aleatórias.
#width: dimensão, quadrada, da imagem.
#appliancesTypes: lista com os inteiros correspondentes aos dispositivos
    ↪ escolhidos. Caso a lista não seja declarada, todos serão considerados.
#savePath: diretório onde as imagens serão salvas. Caso não seja informado, a
    ↪ imagem não será salva.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_VI_SS(nCols:int, rand:bool, appliancesTypes:list, savePath:str, extension:str)
```

```
#Plota o gráfico de das instâncias por dispositivos elétricos.
#Parâmetros:
#savePath: diretório onde as imagens serão salvas.
#extension: extensão do formato da imagem. (padrão: pdf)
plot_instances_per_appliances(savePath:str, extension:str)
```

Agora que as principais funções foram explicadas, para começar a usar a classe para o PLAID1+2, por exemplo, os comandos iniciais são descritos abaixo:

```
#Instancia a classe
pld = PLAID(plaid_path)
#Carrega os metadados do PLAID1+2
pld.read_meta_PLAID1_and_2()
#Faz a leitura dos arquivos contendo as amostras de cada instância do PLAID1+2
pld.read_and_save_all_data(file_name = 'PLAID1_2.pkl', last_points = 10000)
#Imprime algumas informações do dataset para conferência
pld.dataset_info()
```

Para carregar cada uma das características de distinção possíveis, os comandos abaixo devem ser seguidos:

```
#Obtém os pontos de tensão e corrente de um período do estado estável.
repV, repI = pld.get_rep_period_SS()
#Obtém os pontos do plano PQ, no estado estável, para cada instância.
PQ = pld.get_PQ_SS()
#Obtém, para cada instância, as harmônicas e a frequência fundamental.
har = pld.get_harmonicsF_SS()
#Obtém os pontos subamostrados de tensão e corrente de um período do estado estável.
dsV, dsI = pld.get_downSampled_SS()
#Obtém as imagens binárias, de tamanho 50x50, da trajetória V-I.
imgVI = pld.get_imgs_from_VI(width = 50)
#Caso seja necerráio utilizar todas as características, podemos concatená-las:
allF = np.concatenate((PQ, har, dsF, binVI),axis=1)
#Reshape das imagens binárias para aplicação da PCA.
binVI = np.reshape(imgVI,(imgVI.shape[0],imgVI.shape[1]*imgVI.shape[2]))
PCA_BinVI = pld.get_PCAs(binVI, 0.99)
#Rearranja os vetores subamostrados de tensão e corrente para aplicação da PCA.
dsF = np.hstack([dsI,dsV])
PCA_dsF = pld.get_PCAs(dsF, 0.99)
#Aplicação da PCA ao vetor dos valores da corrente.
PCA_repI = pld.get_PCAs(repI, 0.99)
```

Após a familiarização com a extração das características de distinção, será mostrado um exemplo prático. Para realizarmos a avaliação de um modelo de uma RNC, utilizando imagens binárias da trajetória V-I como entrada e dividindo o *dataset* usando o método de validação cruzada *leave-one-out*, os seguintes comandos devem ser usados:

```
#Instancia a classe
pld = PLAID(plaid_path)
#Carrega os metadados do PLAID1+2
pld.read_meta_PLAID1_and_2()
#Faz a leitura dos arquivos contendo as amostras de cada instância do PLAID1+2
pld.read_and_save_all_data(file_name = 'PLAID1_2.pkl', last_points = 10000)
#Obtém a classe a qual o dispositivo de cada instância pertence.
appliancesIDs = pld.get_appliancesIDs() #É usado como rótulo da CNN
#Obtém a lista com o identificador da casa para cada instância amostrada.
housesIDs = pld.get_housesIDs()
#Obtém, de forma abreviada, o nome dos dispositivos elétricos.
appliancesLabelsShort = pld.get_uniqueAppliances_Short()
#Obtém a lista, não repetida, das casas.
uniqueHouses = pld.get_uniqueHouses()
#Gera as imagens binárias, de tamanho 30x30, da trajetória V-I.
imgVI = pld.get_imgs_from_VI(width = 30)
#Chama a função que fará o teste da CNN para o método leave one out.
y_true_leave_one_out, y_pred_leave_one_out = test_cnn_leave_one_out(imgVI,
    ↪ appliancesIDs, housesIDs, uniqueHouses)
#Plota o gráfico de barra com as medidas-F para cada dispositivo elétrico.
plot_f_score(true=y_true_leave_one_out, pred=y_pred_leave_one_out,
    ↪ appliancesLabels=appliancesLabelsShort, savePath=save_imgs)
#Plota a matriz de confusão.
plot_confusion_matrix(true=y_true_leave_one_out, pred=y_pred_leave_one_out,
    ↪ normalize = True, appliancesLabels=appliancesLabelsShort, savePath=save_imgs)
```

Detalhando os principais pontos da função test_cnn_leave_one_out(), percebemos na sétima linha a presença de um um laço, cuja função é fazer a iteração entre todas as residências disponíveis no *dataset*. A cada vez que o laço é iterado, visto que o método de validação cruzada *leave-one-out* foi utilizado, o modelo da RNC é novamente declarado, como mostrado da décima à décima quinta linhas, a fim de que todos os pesos sejam reinicializados.

```
1  def test_cnn_leave_one_out(inputs:list, labels:list, housesIDs:list,
       ↪ uniqueHouses:list, epochs:int = 40, verbose:bool = False):
2      start_time = time.time()
3      predicted = []
4      true = []
5      kLen = len(uniqueHouses)
6      count = 1
7      for k, house in enumerate(range(kLen), start = 1):
8          elapsed_time = time.time()-start_time
9          model = keras.Sequential([
10           keras.layers.Conv2D(filters = 64, kernel_size = 5, activation='relu',
                 ↪ kernel_initializer = keras.initializers.glorot_uniform(seed =
                 ↪ SEED), input_shape=(30, 30, 1)),
11           keras.layers.MaxPool2D(pool_size=2),
12           keras.layers.Conv2D(filters = 32, kernel_size = 5, activation='relu',
                 ↪ kernel_initializer = keras.initializers.glorot_uniform(seed =
                 ↪ SEED)),
13           keras.layers.Dropout(0.26),
14           keras.layers.Flatten(),
15           keras.layers.Dense(11, activation='softmax', kernel_initializer =
                 ↪ keras.initializers.glorot_uniform(seed = SEED))
16         ])
17         ix_train = np.where(housesIDs!=house)[0]
18         ix_test = np.where(housesIDs==house)[0]
19         print('\rProgress: House %2d/%2d (%.2f%%)\t Time elapsed: %s' %
               ↪ (k,kLen,100*(count/kLen), time.strftime("%H:%M:%S",
               ↪ time.gmtime(elapsed_time))), end="")
20         X_train, Y_train = inputs[ix_train], labels[ix_train]
21         X_test, Y_test = inputs[ix_test], labels[ix_test]
22         model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
               ↪ metrics=['accuracy'])
23         model.fit(X_train, Y_train, epochs = 30,verbose = 0)
24         predicted.extend(model.predict(X_test))
25         true.extend(Y_test)
26         count += 1
27     y_pred = []
28     for x in predicted:
29         y_pred.append(np.argmax(x))
30     print('\n')
31     print(classification_report(true, y_pred, digits=4))
32     return true, y_pred
```

Como mostrado na décima sétima e décima oitava linhas, as instâncias de uma residência foram utilizadas como conjunto de teste enquanto que as instâncias das outras casas foram utilizadas como conjunto de treinamento, sendo esse processo repetido para cada uma das casas. Por fim, para cada iteração feita, tanto o vetor com as entradas quanto o vetor com as predições são salvos. Ao fim de todas as iterações, as métricas de avaliação são calculadas.

O parâmetro kernel_initializer, visto em cada camada convolucional e na camada densa, é responsável por uniformizar a inicialização dos filtros, garantindo uma maior estabilidade nos resultados. Dessa forma, aliado a outros artifícios como a escolha de uma semente fixa onde permitido e o uso do pacote *tensorflow-determinism*, faz que os resultados dos experimentos sofram poucas variações quando são repetidos.

APÊNDICE B – Geração do Modelo do *TensorFlow Lite* para o ESP32

Primeiramente, caso o sistema operacional seja o *Windows*, deve-se baixar e instalar o *Git*, disponível em: <https://gitforwindows.org>. Após a instalação, deve-se procurar e baixar o arquivo a seguir, disponível em: <https://sourceforge.net/projects/ezwinports/files>.

```
make-4.3-without-guile-w32-bin.zip
```

Após a descompactação do arquivo baixado, deve-se copiar e colar todo o seu conteúdo, mesclando os diretórios **SEM** sobrescrever nenhum outro arquivo existente, no seguinte local:

```
%programfiles%\Git\mingw64\
```

Para testar se o passo anterior foi seguido corretamente, deve-se abrir o *bash* do *Git* e digitar o comando *make*. Se tudo estiver correto, aparecerá a seguinte mensagem:

```
make: *** No targets specified and no makefile found.  Stop.
```

Caso o sistema operacional seja o *Linux*, basta abrir o terminal e digitar o seguinte comando para iniciar a instalação do *Git*:

```
sudo apt-get install git
```

O passo inicial para a geração do modelo é a clonagem do repositório do *TensorFlow*. Para isso, os seguintes comandos devem ser executados no *Git*:

```
cd <Diretorio>
git clone https://github.com/tensorflow/tensorflow.git
```

Após a clonagem do repositório, para gerar o modelo *hello-world*, por exemplo, deve-se executar os seguintes comandos:

```
cd <Diretorio>/tensorflow
make -f tensorflow/lite/micro/tools/make/Makefile TARGET=esp
    ↪ generate_hello_world_esp_project
```

O modelo será gerado com estrutura semelhante à mostrada na Figura 51 e ficará disponível no seguinte local:

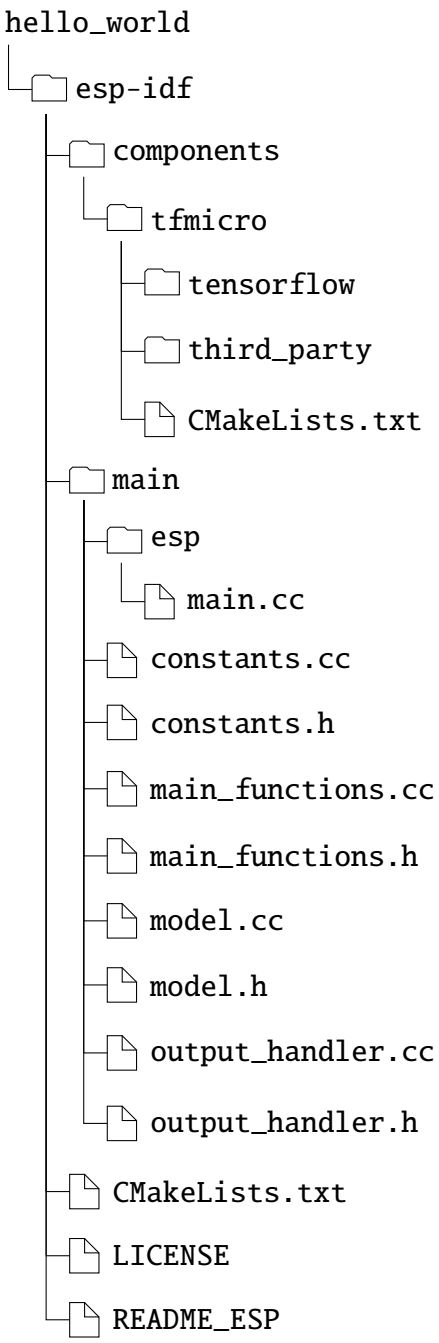
Windows:

```
<Diretorio>\tensorflow\lite\micro\tools\make\gen\esp_xtensa-esp32\prj\hello_world\
```

Linux:

```
<Diretorio>/tensorflow/lite/micro/tools/make/gen/esp_xtensa-esp32/prj/hello_world/
```

Figura 51 – Vista simplificada, em árvore, do modelo *hello_world* gerado

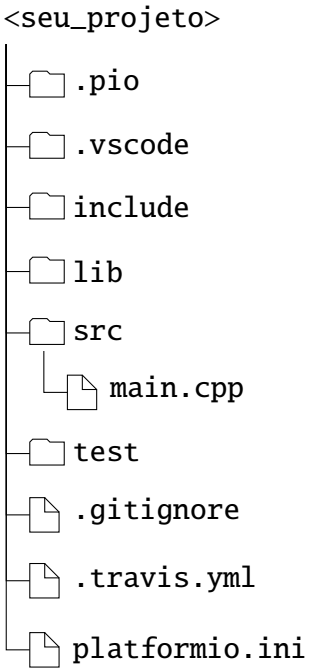


APÊNDICE C – Programação no ESP32 usando o *VS Code* e o *PlatformIO*

Inicialmente, deve-se baixar o *Microsoft Visual Studio Code*, que está disponível gratuitamente em: <https://code.visualstudio.com> para *Windows*, *Linux* ou *macOS*. O próximo passo é instalar a extensão do *PlatformIO*[1] para o *VS Code*, devendo-se abrir o Gerenciador de Extensões do editor da Microsoft, buscar por *PlatformIO IDE* e clicar em Instalar.

O próximo passo é criar um projeto do *PlatformIO* para o ESP32. Para isso, deve-se escolher o nome do projeto, a placa do ESP32 para a qual o projeto será desenvolvido e escolher o *framework Arduino*. O modelo gerado terá uma estrutura semelhante ao mostrado na Figura 52.

Figura 52 – Vista em árvore do projeto do *PlatformIO*



Como foi escolhido o *framework Arduino* no passo anterior, a biblioteca *Arduino.h* estará incluída no arquivo *main.cpp* gerado, que nada mais é que o conhecido arquivo *Sketch* do *Arduino*, no qual temos as duas funções básicas *setup()* e *loop()*. Isso significa que podemos utilizar algumas funções desse *framework* em nosso programa, como *delay()*, *pinMode()*, *digitalWrite()*, dentre outras, aumentando bastante o nível de abstração no desenvolvimento. O arquivo main.cpp gerado é mostrado abaixo:

[1] <https://platformio.org>

```
#include <Arduino.h>

void setup() {
    // put your setup code here, to run once:
}

void loop() {
    // put your main code here, to repeatedly:
}
```

Antes de compilar o projeto, caso deseje salvar a tabela de partições do ESP32 na pasta raiz do projeto do PlarformIO, a qual denominamos <seu_projeto> na Figura 52, basta formatá-la com base nas necessidades do projeto a ser desenvolvido e nas limitações do tamanho da memória *flash* do ESP32. A formatação padrão de fábrica, disponível em: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/partition-tables.html>, é a seguinte:

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000,
phy_init, data, phy,     0xf000,  0x1000,
factory,  app,  factory, 0x10000, 1M,
```

Já uma outra formatação, comumente utilizada, que é a padrão utilizada pelo *Arduino* e pelo *PlatformIO*, disponível em <https://docs.platformio.org/en/latest/platforms/espressif32.html>, é a seguinte:

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x5000,
otadata,  data, ota,     0xe000,  0x2000,
app0,     app,  ota_0,   0x10000, 0x140000,
app1,     app,  ota_1,   0x150000,0x140000,
spiffs,   data, spiffs,  0x290000,0x170000,
```

Para carregar a tabela de partição, basta criar o arquivo partitions.csv com a formatação adequada e salvá-lo. Posteriormente, para registrar a formatação da tabela de partição no arquivo de configuração do projeto do *PlatformIO*, que é o platformio.ini, basta acrescentar e salvar a configuração da linha 5 mostrada abaixo:

```
1 [env:<sua\_placa>]
2 platform = espressif32
3 board = <sua\_placa>
4 framework = arduino
5 board_build.partitions = partitions.csv
6 monitor_speed = 115200
```

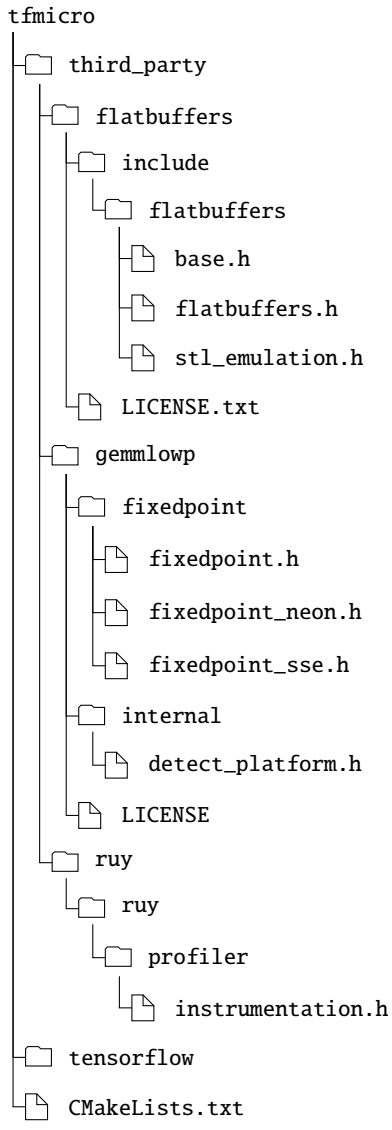
O código completo está disponível no seguinte repositório: <https://github.com/201110008710/NILM_ESP32/tree/master/Modelo_PlatformIO_VS_Code_ESP32>.

APÊNDICE D – Integração do modelo do *TensorFlow Lite* ao *PlatformIO*

Sabendo gerar o modelo do *TensorFlow Lite* para o ESP32, explicado no Apêndice B e sabendo gerar um projeto no *PlatformIO* para o ESP32, explicado no Apêndice D, resta-nos unir esse conhecimento e integrarmos o modelo do *TensorFlow Lite* ao *PlatformIO*, podendo utilizar o *framework* do *Arduino* e elevarmos o nível de abstração, facilitando o processo.

O passo inicial é reorganizar os arquivos do modelo do *TensorFlow Lite*. Baseado na estrutura da Figura 51, deve-se acessar o diretório tfmicro, que possui a estrutura mostrada na Figura 53.

Figura 53 – Vista em árvore do diretório *tfmicro*



Os arquivos devem ser reorganizados de acordo com a estrutura mostrada na Figura 54. Com a reorganização da estrutura concluída, deve-se abrir o arquivo base.h, disponível no seguinte local:

```
/tfmicro/flatbuffers
```

No arquivo base.h, substituir os comandos da linha 35 a 39 exibidos a seguir:

```
35 #if defined(ARDUINO) && !defined(ARDUINOSTL_M_H)
36   #include <utility.h>
37 #else
38   #include <utility>
39 #endif
```

Pelo seguinte comando:

```
#include <utility>
```

O diretório tfmicro deve ser copiado para o diretório lib do projeto do *platformIO*. Todos os arquivos do diretório main do *TensorFlow Lite* devem ser também copiados para o diretório src do projeto do *PlatformIO*. Dentre esses arquivos, copiar os *headers* para o diretório *include*. No fim, a estrutura do projeto deverá ficar semelhante à apresentada na Figura 55.

Figura 54 – Vista em árvore do diretório *tfmicro* reorganizado

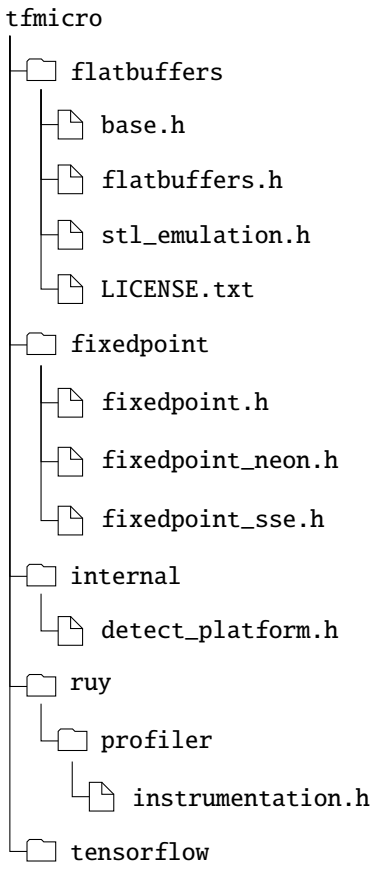
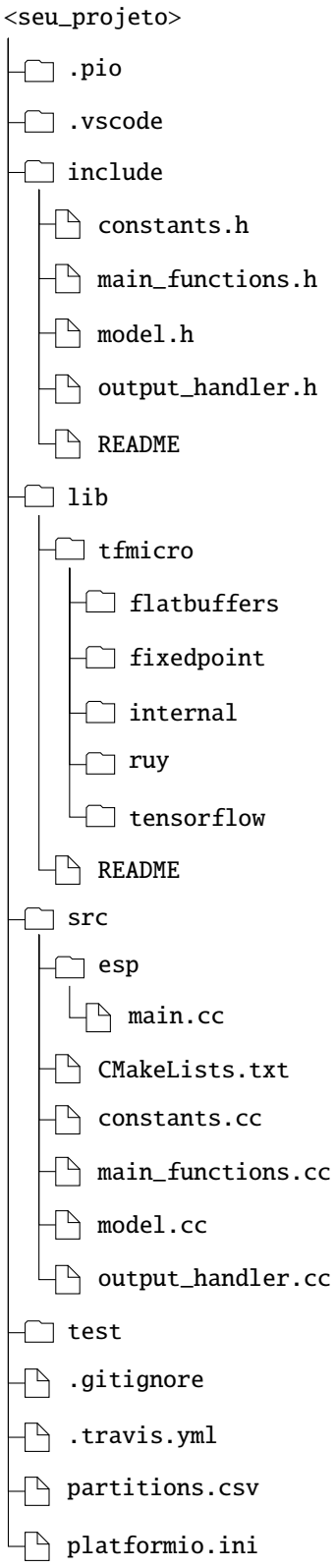


Figura 55 – Vista em árvore do projeto do *TensorFlow Lite* no *PlatformIO*



Após a cópia dos *headers* para a pasta include, deve-se alterar a linha 16 do aquivo main.cc, vista a seguir:

```
#include "..\main_functions.h"
```

Pelo seguinte comando:

```
#include "main_functions.h"
```

É comum que durante a primeira tentativa de compilação apareçam alguns erros. Para corrigir um dos possíveis erros, é necessário fazer algumas alterações nos arquivos max.h e min.h, encontrados no seguinte local:

Windows:

```
<dir\_projeto\_platformio>\lib\tfmicro\tensorflow\lite\kernels\internal
```

Linux:

```
<dir\_projeto\_platformio>/lib/tfmicro/tensorflow/lite/kernels/internal
```

Para o arquivo max.h, o comando da linha 29, visto a seguir:

```
22 #if defined(TF_LITE_USE_GLOBAL_MAX) || defined(__ZEPHYR__)
23 inline float TfLiteMax(const float& x, const float& y) {
24   return std::max(x, y);
25 }
26 #else
27 template <class T>
28 inline T TfLiteMax(const T& x, const T& y) {
29   return std::fmax(x, y);
30 }
31 #endif
```

Deve ser substituído pelo seguinte comando:

```
return fmax(x, y);
```

De forma análoga, para o arquivo min.h, o comando da linha 29, visto a seguir:

```
22 #if defined(TF_LITE_USE_GLOBAL_MIN) || defined(__ZEPHYR__)
23 inline float TfLiteMin(const float& x, const float& y) {
24   return std::min(x, y);
25 }
26 #else
27 template <class T>
28 inline T TfLiteMin(const T& x, const T& y) {
29   return std::fmin(x, y);
30 }
31 #endif
```

Deve ser substituído pelo seguinte comando:

```
return fmin(x, y);
```

Caso apareça um outro erro, notado *à priori* apenas na *release* 2.3.0 do *Tensorflow Lite*, pode ser necessário fazer algumas alterações no arquivo micro_allocator.cc, disponível no seguinte local:

Windows:

```
<dir\_projeto\_platformio>\lib\tfmicro\tensorflow\lite\micro
```

Linux:

```
<dir\_projeto\_platformio>/lib/tfmicro/tensorflow/lite/micro
```

O erro está localizado no comando da linha 406, visto a seguir:

```
404 // Only two conversions are supported - float and int32 - ensure that these
405 // match at compile time instead of duplicating functions here:
406 static_assert((std::is_same<kFlatBufferVectorType, int32_t>() &&
407                 std::is_same<kTfLiteArrayType, TfLiteIntArray>()) ||
408                (std::is_same<kFlatBufferVectorType, float>() &&
409                 std::is_same<kTfLiteArrayType, TfLiteFloatArray>()));
```

Deve ser substituído pelo seguinte comando:

```
404 // Only two conversions are supported - float and int32 - ensure that these
405 // match at compile time instead of duplicating functions here:
406 static_assert((std::is_same<kFlatBufferVectorType, int32_t>() &&
407                 std::is_same<kTfLiteArrayType, TfLiteIntArray>()) ||
408                (std::is_same<kFlatBufferVectorType, float>() &&
409                 std::is_same<kTfLiteArrayType, TfLiteFloatArray>()),"Not supported
                     ↪ conversion");
```

Após essas correções, o modelo do *TensorFlow Lite* poderá ser utilizado no projeto do *PlatformIO* de forma compatível com o *framework* do *Arduino*. O código completo está disponível no seguinte repositório: <https://github.com/201110008710/NILM_ESP32/tree/master/ESP32_TensorFlowLite_PLAID>.