Universidade Federal do Pará

Campus Tucuruí

Faculdade de Engenharia de Computação

Kamilla Taiwhscki Barros Silva

Sistema de Classificação de Imagens Utilizando uma Rede Neural Squeezenet Embarcada em uma Raspberry Pi

UFPA / FECOMP

Campus Universitário de Tucuruí
Tucuruí-Pará-Brasil

2023

Universidade Federal do Pará

Campus Tucuruí

Faculdade de Engenharia de Computação

Kamilla Taiwhscki Barros Silva

Sistema de Classificação de Imagens Utilizando uma Rede Neural Squeezenet Embarcada em uma Raspberry Pi

Trabalho de Conclusão de Curso submetido ao colegiado da Faculdade de Engenharia de Computação da Universidade Federal do Pará, como requisito parcial para a obtenção do grau de bacharel em Engenharia de Computação.

UFPA / FECOMP

Campus Universitário de Tucuruí
Tucuruí-Pará-Brasil

2023

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

S586s Silva, Kamilla Taiwhscki Barros.
Sistema de Classificação de Imagens Utilizando uma Rede Neural Squeezenet Embarcada em uma Raspberry Pi / Kamilla Taiwhscki Barros Silva. — 2023.
72 f. : il. color.

Orientador(a): Prof. Dr. Cleison Daniel Silva
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Pará, Campus Universitário de Tucuruí, Faculdade de Engenharia da Computação, Tucuruí, 2023.

1. Aprendizado de Máquina. 2. Classificação. 3. Raspberry Pi. 4. MNIST. I. Título.

CDD 006.32

Kamilla Taiwhscki Barros Silva

Trabalho de Conclusão de Curso submetido ao colegiado da Faculdade de Engenharia de Computação da Universidade Federal do Pará, como requisito parcial para a obtenção do grau de bacharel em Engenharia de Computação.

gov.br
Documento assinado digitalmente
CLEISON DANIEL SILVA
Data: 16/07/2023 09:36:29-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Cleison Daniel Silva
Faculdade de Engenharia Elétrica - UFPA
Orientador

gov.br
Documento assinado digitalmente
Bruno Merlin
Data: 15/07/2023 16:27:30-0300
Verifique em https://validar.iti.gov.br

Prof. Dr. Bruno Merlin
Faculdade de Engenharia da Computação - UFPA
Membro da Banca Examinadora

gov.br
Documento assinado digitalmente
RAFAEL DE LIMA ROCHA
Data: 17/07/2023 08:57:38-0300
Verifique em https://validar.iti.gov.br

Me. Rafael Lima Rocha
Instituto Tecnológico Vale
Membro da Banca Examinadora

UFPA / FECOMP
Campus Universitário de Tucuruí
Tucuruí-Pará-Brasil
2023

Dedico à todas as mulheres de TI, em especial às mulheres negras, que entendem como é difícil estar nessa área, mas que mesmo assim não desistem e lutam todos os dias por respeito e reconhecimento. Não desistamos de lutar.

Agradecimentos

Primeiramente, agradeço à minha mãe, Maria José Pereira Barros, pelo seu esforço e perseverança, sem ela nada disso seria possível, ela é o motivo de eu nunca ter desistido dessa caminhada que por vezes foi muito árdua, obrigada por me apoiar e por sempre estar presente.

Ao meu parceiro de longa data que foi de amigo à amor da minha vida, Thiago da Silva, por muitas vezes você me fez perceber que existem forças em mim que não me deixam desistir e todos os dias você tenta me fazer enxergar a mulher incrível que eu posso ser. Obrigada por todas as noites de jogatinas e conversas, tudo isso foi importante para me manter sã todos esses anos.

Agradeço fortemente às minhas queridas amigas Amanda Brunelli Melo Pereira e Geiciane Lima Viana, vocês foram fundamentais em todos esses anos, obrigada pelos conselhos e por sempre me apoiarem, eu não sei o que eu seria sem vocês. Agradeço também a todos os meus amigos que fiz durante os anos de faculade, em especial ao Hugo Henrique da Silva Lima, que é um irmão que a vida me deu e que me ajudou nos momentos mais críticos da minha vida, obrigada por ainda ser tão presente nela. Também agradeço ao Eurialdo Mendes Ferreira e ao Henrique Luis Santos Barata que foram meus grandes parceiros em todos esses anos de universidade.

Ao meu orientador, Prof. Dr. Cleison Daniel Silva, por seu apoio e orientação. Obrigada por confiar em mim e acreditar no meu potencial, sei que demorei anos para concluir esse trabalho, mas sempre tive a confiança e senti que você nunca deixou de acreditar em mim. Foi uma honra poder fazer esse trabalho, o crescimento e aprendizado que obtive em todos esses anos eu levarei para a vida toda.

Não posso deixar de agradecer ao Mizael Alves da Silva e ao Rafael Luis Santos, sem vocês esse trabalho não teria sido concluído, sou extremamente grata por tudo o que vocês fizeram por mim, por todo o esforço e dedicação, vocês são incríveis e eu admiro muito o potencial que existe dentro de vocês. Muito obrigada.

Por fim, gostaria de agradecer a todos aqueles que não citei aqui, mas que de alguma forma fizeram parte da minha caminhada.

"Not all those who wander are lost."
(J.R.R. Tolkien, The Fellowship of the Ring)

Resumo

A Visão Computacional é um campo da Inteligência Artificial caracterizado pelo estudo das informações existentes em imagens, identificando suas características intrínsecas. O estudo da Visão Computacional tem como objetivo a criação de modelos artificiais que imitem as habilidades analíticas da visão humana, para isso são utilizados conceitos de Processamento Digital de Imagens para extrair informações a serem estudadas. Realizar essas operações exige uma grande quantidade de dados para ser efetiva e para isso necessitam-se de algoritmos capazes de processarem essas informações. Nesse contexto, algoritmos de Aprendizado Profundo são ideais para trabalharem com uma imensa quantidade de dados, visto estes possuem eficiência e eficácia para tal. Dessa forma, o uso de Redes Neurais para este propósito se torna bastante adequado, pois essa ferramenta permite que seja possível aprender a partir de um conjunto de exemplos de forma que a generalização dos dados seja adequada aos exemplos fornecidos. No caso de imagens, Redes Neurais Convolucionais são o estado da arte na área de Visão Computacional, sendo possível observar diversas aplicações que envolvem a classificação de imagens, identificação de objetos e reconhecimento de faces. Porém, esses algoritmos são robustos e apresentam uma complexa implementação, possuindo diversos parâmetros livres que são determinados durante a execução, exigindo que o *hardware* que o comporta possua elevada capacidade computacional para funcionar sem erros ou com tempo de execução exacerbado. Para o caso de sistemas embarcados que necessitam de baixo custo de implementação, computadores de placa única são comumente adotados, considerando que tais *hardwares* podem ser aplicados em diversos contextos e possuem baixo custo de execução. Todavia, esses dispositivos são restritos em relação ao poder computacional e é necessário um grande estudo das técnicas que permitam a execução de algoritmos complexos em seus *hardwares*. Dessa forma, este trabalho tem o intuito de apresentar um exemplo de implementação de um classificador de imagens em um Computador de Placa Única com uma Rede Neural Convolucional (CNN) sendo executada. São expostos os conceitos de CNNs e de Processamento Digital de Imagens utilizados durante o desenvolvimento do projeto. O classificador desenvolvido captura imagens de dígitos manuscritos e classifica-os em tempo real em 10 classes distribuídas de 0 a 9. Além disso, demonstra-se as técnicas de Processamento Digital de Imagens desenvolvidas, que utilizam o Filtro Gaussiano para aproximar as imagens utilizadas para o treinamento da CNN e as imagens utilizadas durante o teste do classificador embarcado. Os resultados da classificação do sistema demonstram-se razoáveis para o cenário estabelecido, sendo resultados relevantes para o trabalho em questão, em especial ao que diz respeito a acurácia de classificação do sistema de 76% e uma precisão de 80% ao classificar as imagens.

Palavras-chave: Aprendizado de Máquina, Classificação, Raspberry Pi, MNIST.

Abstract

Computer Vision is a field of Artificial Intelligence characterized by the study of existing information in images, identifying their intrinsic characteristics. The study of Computer Vision aims to create artificial models that mimic the analytical skills of human vision, for this, concepts of Digital Image Processing are used to extract information to be studied. Performing these operations requires a large amount of data to be effective and for that, algorithms capable of processing this information are needed. In this context, Deep Learning algorithms are ideal for working with a huge amount of data, as they are efficient and effective. In this way, the use of Neural Networks for this purpose becomes quite suitable, as this tool allows it to be possible to learn from a set of examples so that the generalization of the data is adequate to the examples provided. In the case of images, Convolutional Neural Networks are the state of the art in the area of Computer Vision, and it is possible to observe several applications involving image classification, object identification and face recognition. However, these algorithms are robust and have a complex implementation, having several free parameters that are determined during execution, requiring that the *hardware* that supports it has high computational capacity to function without errors or with exacerbated execution time. For the case of embedded systems that require low implementation cost, single board computers are commonly adopted, considering that such *hardware* can be applied in different contexts and have low execution cost. However, these devices are restricted in terms of computational power and a major study of techniques that allow the execution of complex algorithms on their *hardware* is necessary. Thus, this work aims to present an example of implementation of an image classifier in a Single Board Computer with a Convolutional Neural Network (CNN) running. The concepts of CNNs and Digital Image Processing used during the development of the project are exposed. The developed classifier captures images of handwritten digits and classifies them in real time into 10 classes distributed from 0 to 9. In addition, the developed Digital Image Processing techniques are demonstrated, which use the Gaussian Filter to approximate the images used to the CNN training and the images used during the test of the embedded classifier. The results of the system's classification prove to be reasonable for the established scenario, being relevant results for the work in question, in particular with regard to the classification accuracy of the system of 76% and a precision of 80% when classifying the images.

Keywords: Machine Learning, Classification, Raspberry Pi, MNIST.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ANN	Artificial Neural Network
API	Application Programming Interface
BP	Backpropagation
CAPES	Portal de Periódicos da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CNN	Convolutional Neural Network
DFT	Discrete Fourier Transform
DL	Deep Learning
GB	Gigabyte
GHz	Gigahertz
GPU	Graphics Processing Unit
HDMI	High-Definition Multimedia Interface
LED	Light-Emitting Diode
MLP	Multilayer Perceptron
MP	Megapixel
MNIST	Modified National Institute of Standards and Technology
PDI	Processamento Digital de Imagens
RSL	Revisão Sistemática da Literatura
SO	Sistema Operational
SBC	Single Board Computer
SD	Secury Digital
SDRAM	Synchronous Dynamic Random-Access Memory
USB	Universal Serial Bus

Sumário

1 Introdução

A Visão Computacional é um campo da Inteligência Artificial que busca criar um modelo de visão artificial capaz de imitar a visão humana, possibilitando que máquinas tenham uma habilidade analítica similar às habilidades humanas tais como o reconhecimento de padrões e de imagens (GE et al., 2019). A Visão Computacional é caracterizada por estudar as informações que imagens possuem, identificando suas características intrínsecas. Para realizar a extração de tais características são utilizadas técnicas de Processamento Digital de Imagens (PDI) que consistem em processos dos quais a entrada e a saída são imagens (GONZALEZ; WOODS, 2010). O PDI extrai os dados e gera informações a serem estudadas na Visão Computacional.

Segundo Ponti e Costa (2017), o Aprendizado Profundo revolucionou a Visão Computacional devido a capacidade que seus algoritmos possuem para trabalhar com uma grande quantidade de dados e a de reduzir o tempo de processamento para realizar a identificação e classificação de imagens.

O uso das Redes Neurais Artificiais para esse tipo de tratamento é bastante adequado, pois essa ferramenta permite que se aprenda a partir de um conjunto de exemplos e, assim, seja possível realizar as transformações necessárias, permitindo também que se trabalhe com informações quantitativas, generalização dos conhecimentos, robustez das redes em relação ao ruído e aos erros, e o paralelismo no processamento de dados (OSÓRIO; BITTENCOURT, 2000).

Nesse âmbito, Redes Neurais Convolucionais (*Convolutional Neural Network*, CNN) são o estado da arte na área da visão computacional, sendo possível observar aplicações diversas como: a classificação de imagens, a identificação de objetos e o reconhecimento de faces. As CNNs são algoritmos robustos que apresentam uma implementação complexa, em especial, quanto ao número de parâmetros livres a serem determinados. O que implica a necessidade de disponibilidade computacional elevada dos *hardwares*, principalmente, para não incorrer a erros de memória, ou mesmo, tempo de execução elevado (KIM; JUNG; CHOI, 2019; WEI et al., 2019).

Ao considerar a implementação de CNNs em sistemas embarcados que envolvam Computadores de Placa Única (*Single Board Computer*, SBC), existe uma limitação quanto ao desempenho computacional desses *hardwares* visto que o incremento de novos recursos computacionais são limitação nesse contexto e CNNs exigem do ambiente em que estão sendo implementadas. (DENIZ et al., 2017).

Entretanto, atualmente existem técnicas para embarcar CNNs em sistemas embarcados de forma efetiva. O baixo custo que sistemas embarcados normalmente proporcionam

facilita o acesso para estudar tais arquiteturas e implementar diversas aplicações em diferentes ambientes, como, por exemplo, aplicações para o campo (MILIOTO; LOTTES; STACHNISS, 2017), tecnologias assistivas (CALABRESE et al., 2020) e sistemas de monitoramento (NIKODEM et al., 2020). Outra aplicação de CNNs amplamente realizada é a classificação da escrita humana (LECUN et al., 1989). Visto que existem diversas variações da escrita de uma pessoa para outra, classificar corretamente todas as letras e dígitos de um grupo de pessoas pode ser uma tarefa laboriosa e desafiadora.

A classificação de caracteres manuscritos em imagens é um desafio explorado há anos (LECUN et al., 1989; SERMANET; CHINTALA; LECUN, 2012; ASHIQUZZAMAN; TUSHAR, 2017) e, apesar de já ter sido uma aplicação extremamente desenvolvida, não é uma solução trivial, necessitando do entendimento de CNNs no contexto de Visão Computacional. Embarcar um classificador que utiliza CNNs pode levar à desafios que proporcionam uma compreensão acerca de sistemas que envolvem *hardware* e *software* nesse contexto.

Dessa forma, neste trabalho são abordados os conceitos de CNNs e SBCs e a arquitetura que os envolvem em um sistema embarcado. Procura-se compreender o comportamento de CNNs ao serem embarcadas em ambientes com recursos computacionais limitados e quais as medidas necessárias para desenvolver tal dispositivo de forma eficaz e eficiente no contexto de classificação de dígitos manuscritos em imagens.

1.1 Objetivo geral

Este trabalho tem como objetivo utilizar técnicas de aprendizado de máquina para desenvolver um classificador embarcado que realize a classificação de imagens que possuem dígitos manuscritos utilizando Redes Neurais Convolucionais e Computadores de Placa Única.

1.2 Objetivos específicos

Neste trabalho, além de utilizar as técnicas de aprendizado de máquina, também é necessário aplicar os conceitos do Processamento Digital de Imagens para realizar o pré-processamento dos exemplos que são utilizados para a classificação. Os objetivos específicos são definidos para que seja possível alcançar o objetivo geral deste trabalho e são descritos a seguir:

- Buscar e selecionar algoritmos codificados na linguagem de programação Python que realizam a classificação de caracteres utilizando Redes Neurais Convolucionais;
- Realizar a montagem do *hardware* de acordo com as necessidades do projeto;

- Capturar imagens utilizando o *hardware* para que seja possível o desenvolvimento de uma base de dados própria a ser utilizada para a classificação;
- Implementar o sistema embarcado que fará a leitura de dígitos escritos por um usuário que serão classificados entre as classes disponíveis (0 a 9);
- Comparar os resultados obtidos e avaliar a viabilidade do sistema embarcado no contexto em que foi proposto.

2 Revisão da literatura

Neste capítulo são apresentados os estudos que compõem a revisão sistemática da literatura, incluindo discussões acerca dos estudos selecionados com o objetivo de analisar aplicações que envolvam Computadores de Placa Única que implementam Redes Neurais Convolucionais, tendo como propósito auxiliar no desenvolvimento da pesquisa e na realização dos objetivos propostos para este trabalho.

2.1 Introdução

Computadores convencionais normalmente são empregados para a implementação de CNNs, visto que as limitações de desempenho são facilmente superadas com o incremento de recursos computacionais, sempre que necessário. O mesmo não costuma ser verdadeiro em aplicações que exijam sistemas embarcados com menor custo (DENIZ et al., 2017).

Nesse contexto, utilizar SBCs costuma ser a solução adotada, dado que os SBCs permitem a implementação de CNNs e são rotineiramente utilizados em sistemas embarcados, possibilitando um conjunto de soluções de baixo custo e alto desempenho (LEROUX et al., 2017; ALI; DUMAN; BETÜL, 2020).

Ainda que SBCs possuam limitações quanto ao seu poder computacional, com o devido planejamento e boa execução, é possível implementar aplicações complexas em seu *hardware*. Compreender como essas aplicações são realizadas pode ser fundamental para a execução de projetos de forma eficiente. Conhecer os SBCs comumente utilizados para este fim pode levar a uma escolha ideal dos *hardwares* e o entendimento das CNNs que são implementadas em SBCs também permite a proposição de novos caminhos para a resolução de problemas que envolvam ferramentas da área de visão computacional.

Nesse contexto, a revisão da literatura tem como objetivo entender o funcionamento de tais arquiteturas em aplicações envolvendo a resolução de problemas de visão computacional na classificação, identificação e reconhecimento de imagens para sua utilização no desenvolvimento do sistema embarcado proposto nesse trabalho.

2.2 Planejamento

Para entender o funcionamento de arquiteturas de CNNs aplicadas em SBCs é realizada uma revisão sistemática da literatura baseada no modelo sugerido em Kitchenham e Charters (2007). Tal modelo consiste em definir um protocolo para a especificação da pesquisa, seguindo passos definidos e explícitos, a fim de buscar estudos alinhados ao tema

e facilitar a inserção de trabalhos relevantes e que agreguem valor à pesquisa.

Inicialmente é preciso desenvolver uma ou mais questões de pesquisa (QP) que são utilizadas para delinear o desenvolvimento da revisão. Os estudos buscados e utilizados devem responder às questões de pesquisa aqui definidas. A revisão desse trabalho tem como objetivo responder as seguintes perguntas:

- **QP.1** Como é feita a avaliação do desempenho dos computadores de placa única que implementam redes neurais convolucionais ao realizar a classificação de imagens?
- **QP.2** É viável implementar algoritmos de redes neurais convolucionais em computadores de placa única?

Uma vez definidas as questões de pesquisa, é necessário elaborar uma *string* de busca que é utilizada para encontrar estudos alinhados ao tema da RSL nas bases científicas. Os estudos identificados como relevantes são chamados de estudos primários e respondem às questões de pesquisa (KITCHENHAM; CHARTERS, 2007).

A *string* de busca é composta por palavras-chaves que levam em consideração os temas principais que envolvem o objetivo da revisão da literatura. Nesse caso, os temas de interesse são os algoritmos de redes neurais convolucionais, logo, é utilizado o termo "*convolutional neural network*" como primeira parte da *string*. Similarmente é interessante buscar por estudos que realizam a classificação de imagens, então o termo "*classification*" compõe a segunda parte; e, por último, é utilizado o termo "*single board*" para se referir aos computadores de placa única.

O termo "*single board computer*" não é utilizado para compor a *string*, pois durante os testes de buscas nas bases científicas, os resultados encontrados não foram satisfatórios, ou seja, os estudos retornados com a *string* "*"convolutional neural network" AND "classification" AND "single board computer"*" não se adequam para responder as questões de pesquisa, em especial, por excluírem artigos relevantes e que são retornados com a *string* que possui o termo "*single board*". Sendo assim, a *string* de busca utilizada nessa pesquisa é a seguinte:

***String* de busca:** *"convolutional neural network" AND "classification" AND "single board"*

As pesquisas são realizadas no Portal de Periódicos da CAPES[1], o acervo científico virtual brasileiro de acesso a estudos científicos e bases de dados eletrônicas. Também são feitas pesquisas diretamente na biblioteca digital do IEEE Xplore[2]. As buscas foram realizadas no período de 04 de março de 2021 a 16 de abril de 2021 e retornaram estudos acadêmicos em formato de artigo que foram publicados entre os anos 2015 e 2021.

[1] <http://periodicos.capes.gov.br>
[2] <https://ieeexplore.ieee.org>

Critérios de inclusão e de exclusão são definidos para que sejam selecionados, dentre os estudos encontrados através da *string*, apenas aqueles que se alinham ao objetivo da revisão. Os critérios são diretos e restritos para garantir que os materiais obtidos sejam relevantes à revisão sistemática da literatura.

Os critérios de inclusão e exclusão são definidos como se segue:

- São incluídos estudos que abordem a classificação de imagens. Não excluindo aqueles que apresentem o reconhecimento e identificação de objetos em imagens;
- São incluídos trabalhos disponíveis na web, escritos em língua inglesa ou portuguesa do Brasil;
- São incluídos estudos que possuem avaliação por pares;
- São incluídos estudos que abordem aplicações em computadores de placa única;
- São incluídos estudos publicados entre os anos 2015 e 2021, excluindo-se todos os estudos anteriores a esse período;
- E são incluídos estudos que aplicam algoritmos de redes neurais convolucionais como, pelo menos, uma das técnicas de visão computacional.

A seleção dos estudos primários para a revisão sistemática da literatura é realizada em duas fases e considera os critérios de inclusão e de exclusão definidos acima. Na primeira fase são excluídos estudos através da leitura do título, das palavras-chave e dos resumos. A segunda fase é caracterizada pela leitura completa dos artigos, analisando cada elemento de suas estruturas com a intenção de avaliar se os estudos realmente se encaixam nos critérios de inclusão.

Através dos resultados obtidos, é gerada uma tabela com todos os artigos incluídos. Esses estudos passam por uma avaliação de qualidade conforme os critérios definidos no protocolo de revisão da literatura e descritos a seguir:

- Os estudos devem descrever adequadamente e de forma explícita a metodologia aplicada para a obtenção dos resultados;
- Os estudos devem ter objetivos claros e bem definidos;
- E os estudos devem ter a avalição do autor em relação à pesquisa.

A última etapa da revisão sistemática da literatura trata-se da extração e síntese dos dados dos estudos primários e tem como objetivo registrar com precisão as informações obtidas em formulários de forma qualitativa ou quantitativa através da leitura completa e detalhada dos estudos selecionados (KITCHENHAM; CHARTERS, 2007). Os dados

quantitativos das obras são coletados e utilizados para o desenvolvimento da revisão aqui apresentada.

Para cada um dos estudos é realizada a identificação dos pontos principais abordados nos trabalhos, a identificação das hipóteses, das variáveis dependentes e independentes e dos materiais utilizados. É apontado o planejamento de cada um dos estudos selecionados, os resultados obtidos, as ameaças à validade em relação aos resultados e ao método utilizado. Além disso, são selecionadas as referências consideradas relevantes em cada obra através do título e é dada preferência àquelas que se alinham a essa revisão. Ao concluir o preenchimento dos formulários de extração de dados é possível realizar conclusões e análises críticas acerca dos estudos primários e, assim, responder as questões de pesquisa. A Figura 1 resume os passos utilizados para desenvolver a revisão sistemática da literatura.

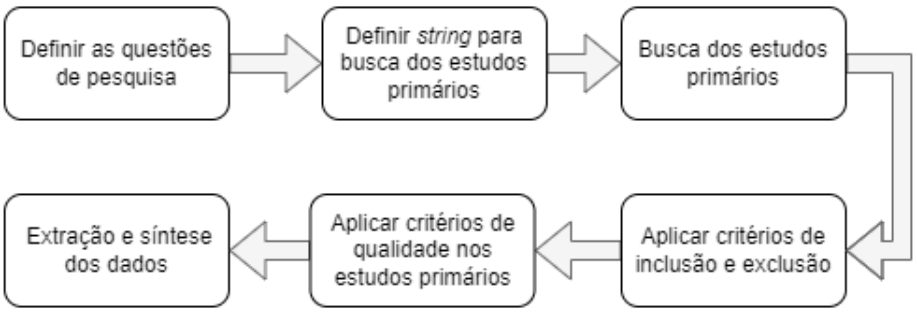

Figura 1 – Passos seguidos para a realização da Revisão Sistemática da Literatura. Fonte: Elaborada pelo autor.

2.3 Execução e resultados

A execução da *string* de busca dos estudos primários relacionados à revisão retorna um total de 70 artigos, sendo 64 estudos retornados através do Portal de Periódicos da CAPES e 6 do IEEE Xplore. A partir do resultado das buscas são aplicadas as fases de seleção das obras.

Na primeira fase de seleção dos estudos é identificado que 50 artigos não atendem às necessidades dessa revisão da literatura segundo os critérios de inclusão e de exclusão definidos na Seção 2.2. Desse total, 48 encontrados no Portal de Periódicos da CAPES. Restando 20 artigos para análise na segunda fase que se caracteriza pela leitura completa e avaliação das obras através dos critérios de qualidade definidos.

A segunda fase leva a exclusão de 3 artigos, os três recuperados do Portal de Periódicos da CAPES. A execução das duas fases resulta em 17 artigos incluídos. A evolução da seleção dos estudos em cada fase pode ser observada na Tabela 1.

Tabela 1 – Estudos selecionados nas Fases 1 e 2. Fonte: Elaborada pelo autor.

Base de dados	Artigos encontrados	Fase 1	Fase 2
Periódicos da CAPES	64	16	13
IEEE Xplore	6	4	4
Total	70	20	17

2.3.1 Síntese dos estudos primários

Como mencionado anteriormente, a extração dos dados dos estudos primários selecionados tem o objetivo de analisar elementos relevantes a revisão. Sendo assim, é possível identificar quais as aplicações utilizadas nos estudos para implementar os algoritmos de CNNs nos SBCs.

No trabalho de Curtin e Matthews (2019) é desenvolvido um dispositivo para realizar o reconhecimento de imagens. A intenção do dispositivo é ser uma solução de baixo custo para o reconhecimento de animais selvagens. É utilizado o Raspberry Pi[3] 3 B+ para a implementação da CNN desenvolvida, avaliando o tempo de execução da rede no *hardware*. A aplicação tem como objetivo trabalhar em tempo real e classificar as imagens capturadas entre três classes. Em um trabalho semelhante Shiddieqy *et al.* (2017) tem o objetivo de avaliar o desempenho do Raspberry Pi 3 B levando em consideração o tempo de execução, precisão e sensibilidade do sistema.

Por outro lado, o trabalho de Oliveira e Wehrmeister (2018) utiliza o Raspberry Pi 2 em uma aplicação para o reconhecimento de padrões em imagens aéreas, identificando e classificando as imagens. O SBC é avaliado em relação ao tempo de execução, acurácia e sensibilidade do sistema. Para fins de comparação, são avaliados outros algoritmos de classificação também aplicados na placa. No estudo é possível perceber como implementar CNNs em SBCs exige do poder computacional dos *hardwares* devido a complexidade das CNNs. Esse fato é peceptível quando é realizada a comparação com outros algoritmos utilizados no estudo considerados menos complexos que CNNs, como a cascata do tipo Haar. Tais algoritmos menos complexos conseguem ser executados exigindo menos do Raspberry Pi 2. Similarmente, em Hossain e Lee (2019) é desenvolvido um sistema de identificação de objetos em imagens aéreas e os autores avaliam dentre seis arquiteturas qual apresenta melhor desempenho em relação ao tempo de processamento.

No estudo de Lindner *et al.* (2020) é proposto um sistema de monitoramento que utiliza uma CNN multitarefa e tem como objetivo realizar o monitoramento de pessoas, dessa forma o sistema deve reconhecer, identificar e classificar imagens. O SBC utilizado na aplicação é o NVIDIA[4] Jetson Nano e é avaliado o seu tempo de processamento. A aplicação busca ter um bom desempenho em relação ao processamento do SBC, porém os

[3] <https://www.raspberrypi.org/>
[4] <https://www.nvidia.com/pt-br>

autores sugerem soluções com custos menores que utilizam o Raspberry Pi 3 B+. Outros trabalhos como o de Milioto *et al.* 2017 e Kim *et al.* 2019 também propõem sistemas de monitoramento utilizando CNNs e SBCs e os avaliando da mesma forma.

Alguns autores como Leroux *et al.* (2017) propõem o particionamento de redes neurais convolucionais como soluções que auxiliam o desempenho de SBCs, pois CNNs particionadas são menos complexas se comparadas a CNNs convencionais. O trabalho de Leroux *et al.* (2017) tem o objetivo de classificar imagens e três SBCs diferentes são utilizados: o NVIDIA Jeston TK1, o Raspberry Pi 2 e o Intel[5] Edison. A avaliação é realizada levando em consideração o tempo de processamento dos SBCs. A sugestão de particionamento de CNNs, que são chamadas pelos autores de "CNNs leves", se mostra eficiente e evita que os SBCs tenham suas capacidades sobrecarregadas por algoritmos complexos (OLIVEIRA; BORIN, 2019; LEROUX et al., 2017). CNNs leves são propostas para serem aplicadas em SBCs nos trabalhos de Nikodem *et al.* (2020), que avalia a memória utilizada, em Dayal *et al.* (2021) e Peine *et al.* (2019), que avaliam o tempo de processamento e o consumo de energia.

Aplicações de tecnologia assistivas também podem ser desenvolvidas utilizando CNNs e SBCs e podem ser soluções de baixo custo como o sugerido em Calabrese *et al.* (2020). No trabalho é apresentado uma tecnologia vestível que realiza o reconhecimento, identificação e classificação de imagens. Esses autores avaliam o tempo de processamento e o consumo de energia de um NVIDIA Jetson Nano. O estudo de Boschi *et al.* (2020) tem o objetivo de desenvolver um robô autônomo para o acompanhamento de pessoas idosas que é capaz de realizar tarefas semelhantes às do trabalho anterior, porém se focando no pouco consumo de energia em compensação com o desempenho que o sistema proposto pode oferecer. Por outro lado, Dayal *et. al* (2021) desenvolvem um sistema de autenticação que identifica, classifica e reconhece gestos da linguagem de sinais, avaliando o tempo de processamento do Raspberry Pi 4 B utilizado.

No trabalho de Khan (2020) a CNN é implementada em um Raspberry Pi 3 B e treinada na placa, o que não é uma aplicação comum, pois é sabido que o treino de uma CNN é a fase que exige um elevado poder computacional do *hardware* (LEROUX et al., 2017; NIKODEM et al., 2020). O treino é realizado dessa forma na aplicação, pois o sistema de classificação e identificação de imagens proposto receberá constantemente novas imagens a serem adicionadas ao conjunto de treino, logo a rede neural deve ser treinada novamente. E para evitar repetições de implementação de código na placa o treino é realizado no SBC em segundo plano e a classificação ocorre em paralelo. De forma similar, em seu trabalho, Oliveira e Borin (2019) também treinam a CNN diretamente no SBC.

O estudo de Sordillo *et al.* (2021) avalia o desempenho e o consumo de energia de

[5] <https://www.intel.com.br/>

SBCs executando uma CNN que realiza o reconhecimento de objetos em imagens, esse artigo traz informações relevantes sobre o uso de algoritmos complexos em SBCs. Os autores Cho, Choi e Kim (2020) em seu estudo propõem um *framework* em uma CNN com o objetivo de a tornarem mais viável para funcionar em SBCs. A CNN realiza o reconhecimento de objetos em imagens. E, por último, Nikodem *et. al* (2020) desenvolvem um sistema para identificação, classificação e reconhecimento de objetos, aplicando a CNN para o monitoramento de estacionamentos e cruzamentos de veículos, buscando elaborar um sistema de baixo custo, avaliando o tempo de processamento do SBC utilizado.

Com a síntese dos estudos primários é possível notar que 11 estudos abordam a classificação de imagens, aplicação que abrange a maioria dos artigos, seguida de reconhecimento de imagens, com 8 estudos. A identificação de objetos em imagens é realizada em 7 dos estudos primários.

Outro ponto importante é a motivação em utilizar os dispositivos escolhidos para realizar a classificação, identificação e reconhecimento de imagens. É identificado que a maioria dos estudos (52,9%) escolhem essa abordagem por se tratar de dispositivos que possuem um baixo custo de implementação. Vale ressaltar que SBCs são dispositivos com custos variados e possuem desempenho computacional diferente, com o seu valor variando conforme a capacidade computacional.

O Raspberry Pi 3 B é o mais utilizado nos estudos, modelo que aparece em um total de 7 estudos primários. Os SBCs da família Raspberry Pi são uma das plataformas favoritas para o desenvolvimento de sistemas para a Internet das Coisas devido ao seu tamanho físico pequeno e a placa acessível para novos usuários e entusiastas da área (LEROUX et al., 2017). Modelos mais antigos do Raspberry Pi, como a versão 2, ainda são utilizados e apresentam um bom funcionamento com aplicações de CNNs de poucas camadas (LEROUX et al., 2017; LINDNER et al., 2020).

Os modelos com maior poder computacional, como o Raspberry Pi 3 B+ e o Raspberry Pi 4 B, são opções de baixo custo que oferecem um bom desempenho para aplicações de visão computacional. Muitos estudos comparam esses dispositivos, principalmente o modelo 3 B+, com outros SBCs mais poderosos, como é o caso dos SBCs que pertencem à família dos NVIDIA Jetson e que possuem GPU. O NVIDIA Jetson Nano aparece em 5 dos estudos primários, sendo uma solução poderosa para a execução de CNNs de forma eficiente, porém é um SBC com custo elevado se comparado às placas da família Raspberry, podendo ser uma opção quando o tempo de processamento é mais importante do que o baixo custo do sistema (LINDNER et al., 2020).

Aplicações que focam no baixo custo (50% dos estudos primários) escolhem SBCs para fazer parte dos seus sistemas, pois com tais *hardwares* é possível realizar aplicações como controle de acesso, autenticação de usuários e sistemas de monitoramento (CURTIN; MATTHEWS, 2019; OLIVEIRA; WEHRMEISTER, 2018; LINDNER et al., 2020;

NIKODEM et al., 2020) mantendo um custo factível, além da fácil replicação. Também é interessante notar em aplicações que envolvem sistemas que promovem a acessibilidade de usuários (CALABRESE et al., 2020) ou que fazem parte de sistemas que já possuem um custo elevado (MILIOTO; LOTTES; STACHNISS, 2017; NIKODEM et al., 2020; KIM; JUNG; CHOI, 2019; BOSCHI et al., 2020) o baixo custo do sistema é uma preocupação dos autores.

O desempenho dos SBCs também é considerado um fator importante na escolha desses *hardwares*, pois as opções são variadas e, dependendo da finalidade da aplicação, utilizar um SBC que possui menos poder computacional do que a aplicação necessita, pode levar a custos desnecessários e impossibilidade da execução da aplicação do modo esperado. Dessa forma, alguns estudos fazem o experimento de vários SBCs para que seja possível escolher o ideal para a aplicação que propõem (LINDNER et al., 2020; SORDILLO et al., 2021; LEROUX et al., 2017; HOSSAIN; LEE, 2019; DAYAL et al., 2021). O desempenho dos SBCs é mandatório para sua escolha em 40% dos estudos primários. O pouco consumo de energia e o tamanho do SBC é motivação em 10% e 5% dos estudos primários, respectivamente.

2.3.2 Respostas as questões de pesquisa

Como definido anteriormente, essa revisão sistemática da literatura tem como objetivo responder as questões de pesquisas definidas para esta pesquisa. Tais perguntas são respondias a seguir.

QP.1 Como é feita a avaliação do desempenho dos computadores de placa única que implementam redes neurais convolucionais ao realizar a classificação de imagens?

Com a síntese dos estudos primários (EPs), é possível perceber que a maioria (88,24%) avalia os SBCs de acordo com o tempo de processamento desses *hardwares* uma vez que as aplicações exigem resultados em tempo real. As avaliações da acurácia (29,41% dos EPs) e do consumo de energia (17,65% dos EPs) aparecem em estudos que realizam aplicações que envolvem reconhecimento de padrões, identificação e classificação de imagens. A sensibilidade (*recall*) e a memória utilizada são avaliações que aparecem poucas vezes nos estudos, abrangendo apenas 11,76% e 5,88% dos EPs, respectivamente.

QP.2 É viável implementar algoritmos de redes neurais convolucionais em computadores de placa única?

A síntese dos estudos primários traz uma relação extensa de aplicações que envolvem CNNs e SBCs de forma eficiente. Autores também sugerem novas arquiteturas de CNNs com a proposição de novos algoritmos que podem aumentar a eficiência das aplicações. Então, conclui-se que tais implementações são viáveis de serem executadas no contexto de classificação de imagens.

3 Fundamentação teórica

Neste capítulo são abordados os conceitos teóricos utilizados como base para a produção desta pesquisa. São abordados os temas fundamentais relacionados ao aprendizado de máquina, mais especificamente redes neurais artificiais e aprendizado profundo. Também é abordado o processamento digital de imagens, visto que é uma técnica fundamental para a realização deste trabalho.

3.1 Redes Neurais Artificiais

3.1.1 Introdução

De acordo com Osório e Bittenccourt (2000), as Redes Neurais Artificiais (*Artificial Neural Networks*, ANN) são métodos conexionistas inspirados na organização e funcionamento estrutural do cérebro humano, simulando os elementos nele existentes, como os neurônios e as suas conexões. Uma ANN é uma técnica de aprendizado de máquina (*Machine Learning*, ML) constituída por um grafo orientado e ponderado que forma um autômato complexo chamado rede neural, sendo este composto por autômatos simples que são os neurônios artificias da rede.

O conhecimento de uma ANN é representado através das conexões entre os neurônios, sendo codificados na estrutura da rede. As ANNs associam cada conexão a um valor numérico chamado peso sináptico que são os parâmetros livres da rede (HAYKIN, 2001), tais pesos geram um conjunto de valores que descrevem o comportamento da rede e que armazenam o conhecimento adquirido. O aprendizado de uma ANN é feito de forma gradual em um processo de adaptação dos parâmetros livres que a rede possui, adquirindo conhecimento através da experiência obtida a partir do aprendizado automático proporcionado pelos algoritmos que as implementam (OSÓRIO; BITTENCOURT, 2000).

A capacidade de uma ANN aprender a solucionar um determinado problema e gerar saídas adequadas para entradas diferentes das contidas no treinamento ou fase de aprendizagem é o que faz com que elas sejam algoritmos extremamente robustos e capazes de solucionar problemas complexos (HAYKIN, 2001).

3.1.2 O neurônio artificial

O neurônio é a unidade de processamento de uma ANN e é parte fundamental para o seu funcionamento, pois é ele quem processa a entrada e a saída da rede de acordo com seus parâmetros. Um modelo de um neurônio artificial e as suas conexões é demonstrado

na Figura 2.

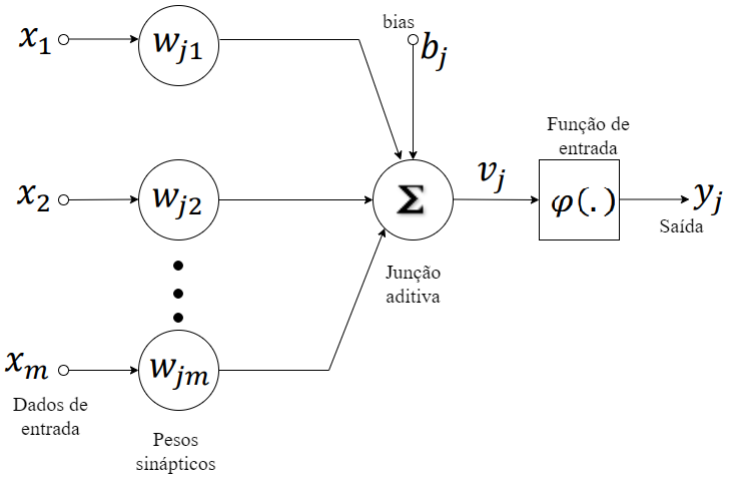

Figura 2 – Modelo de um neurônio artificial e suas conexões. Composto pelas sinapses, o somador e a função de ativação. Fonte: Adaptada de Haykin, 2001.

Nessa figura é possível perceber que cada sinal de entrada é ponderado por seus respectivos pesos sinápticos e somados na junção aditiva. Esse somatório recebe um valor chamado *bias* que pode assumir valores positivos ou negativos e tem a função de diminuir ou aumentar a entrada da função de ativação da rede. A função de ativação irá restringir a amplitude da saída do neurônio (HAYKIN, 2001), ou seja, irá limitar os valores da saída em um determinado intervalo. Essa estrutura compõe uma rede neural de uma única camada e o seu modelo estruturado por Rosenblatt (1958) é chamado de Perceptron. O aprendizado dessa rede neural é feito através da alimentação para frente (*feed-forward*), significando que os dados de entrada são projetados para a camada de saída, sendo que o contrário não acontece (HAYKIN, 2001).

O neurônio e suas conexões podem ser representados matematicamente como descrito a seguir:

$$u_k = \sum_{j=1}^{m} w_{k_j} x_j \tag{3.1}$$

$$y_k = \varphi(u_k + b_k) \tag{3.2}$$

A equação 3.1 descreve a combinação linear das entradas ponderadas, onde x_j são os sinais de entrada; w_{k_j} são os pesos do neurônio k; e u_k é a saída do combinador linear (somatório). A equação 3.2 é a saída do neurônio y_k, sendo $\varphi(.)$ a função de ativação e b_k o *bias*, que tem a finalidade de aplicar uma função afim na saída do combinador linear (HAYKIN, 2001).

3.1.3 Redes neurais de múltiplas camadas

Uma rede neural de múltiplas camadas é caracterizada por possuir camadas de neurônios ocultas entre as camadas de entrada e de saída da rede. As camadas ocultas permitem que a rede extraia características dos sinais de entrada de forma progressiva, obtendo informações mais detalhadas desses dados (HAYKIN, 2001).

A Figura 3 representa uma rede de Perceptrons de múltiplas camadas (*Multilayer Perceptron*, MLP) que possui uma camada de entrada, uma camada oculta e uma camada de saída, onde cada um dos neurônios de uma camada estão conectados a todos os neurônios da camada seguinte (HAYKIN, 2001). Essas estruturas são chamadas de totalmente conectadas (*fully connected*). O sinal de saída de uma MLP proporciona uma resposta global para toda a rede, sendo possível a classificação da saída em mais de duas classes.

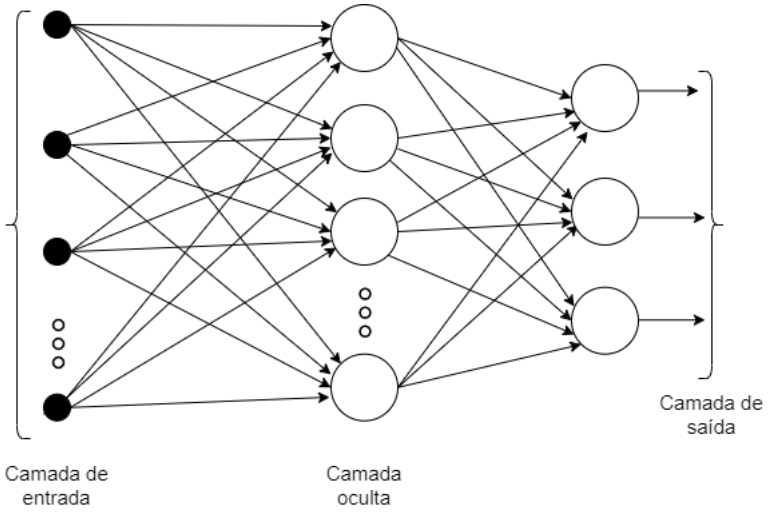

Figura 3 – Arquitetura de um Perceptron de múltiplas camadas possuindo uma camada oculta. Fonte: Elaborada pelo autor.

Uma MLP, assim como toda rede neural, tem o objetivo de aprender a classificar exemplos que englobam um determinado problema, os exemplos podem ser rotulados ou não rotulados. O aprendizado para tipos de exemplos rotulados é chamado de aprendizado supervisionado e o aprendizado de exemplos não rotulados, recebe o nome de aprendizado não-supervisionado.

No aprendizado supervisionado, a rede recebe exemplos de referência e adapta seus pesos sinápticos de acordo com os exemplos dados. Dessa forma, tenta-se obter o melhor classificador para o problema em questão (MELLO; PONTI, 2018). O algoritmo de retropropagação do erro (*Backpropagation*, BP) é amplamente utilizado para a correção do erro de classificação em redes de múltiplas camadas.

3.1.4 Funções de ativação

Uma função de ativação é uma função matemática aplicada na saída de cada neurônio da rede e é responsável por restringir a amplitude da sua saída, limitando os seus valores em um intervalo (HAYKIN, 2001). A função de ativação introduz a não-linearidade, permitindo que o modelo aprenda representações complexas e não lineares dos dados de entrada durante o treinamento (BISHOP, 2006; TABIK et al., 2017).

A função Unidade Linear Retificada (*Rectified Linear Unit*, ReLU) é uma função de ativação amplamente utilizada no contexto de redes neurais. Tal função retorna zero para valores negativos e o próprio valor de entrada para valores positivos (PONTI et al., 2017). A equação 3.3 é a sua definição matemática, onde x é a entrada de um neurônio.

$$f(x) = max(0, x) \tag{3.3}$$

A função *softmax*, também conhecida como a exponencial normalizada, é outra função de ativação comumente utilizada em redes neurais e é geralmente aplicada na camada de saída de redes neurais convolucionais para problemas de classificação multiclasse. A *softmax* transforma as saídas em uma distribuição de probabilidades e as normaliza, garantindo que a soma de todas as probabilidades seja igual a 1 (BISHOP, 2006).

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3.4}$$

A equação 3.4 é a definição matemática da *softmax*, onde $(\vec{z})$ é o vetor de entrada da função; z_i são todos os elementos do vetor de entrada da função e podem assumir qualquer valor real; e^{z_i} é uma função aplicada a todos os elementos do vetor de entrada; $\sum_{j=1}^{K} e^{z_j}$ é o termo de normalização que garante que a soma de todos os valores da saída seja igual a 1 constituindo uma distribuição de probabilidade válida; K é o número de classes do classificador multiclasse.

3.2 Aprendizado Profundo

O aprendizado profundo (*Deep Learning*, DL) é um método de Inteligência Artificial que visa resolver problemas que são complexos no contexto de aprendizado de máquina. A capacidade de trabalhar com grandes quantidades de dados que algoritmos de DL possuem e a de reduzir o tempo de processamento desses dados, permite que algoritmos de DL sejam amplamente utilizados em diversas áreas de conhecimentos, inclusive na visão computacional (PONTI; COSTA, 2017).

O DL possibilita que modelos computacionais de múltiplas camadas aprendam e representem dados com vários níveis de abstração (VOULODIMOS et al., 2018), realizando

de forma eficiente tarefas complexas que envolvam o processamento de sinais com objetivo de aprender sucessivamente as diferentes representações dos dados de entrada de um dado problema. O aprendizado de forma sucessiva é obtido através da hierarquia das representações do sinal de entrada, de forma que o algoritmo aprende as características mais simples às mais complexas, de modo que, após o treinamento do algoritmo de DL, ele seja capaz de responder aos exemplos apresentados, sendo esses exemplos não pertencentes ao conjunto de dados de treino utilizado pelo algoritmo (OSÓRIO; BITTENCOURT, 2000).

Como visto anteriormente, o MLP consiste em ser uma função matemática que recebe dados na sua entrada e gera uma saída através do mapeamento de tais dados. É um dos método de DL em que a sua função f é obtida através da composição de funções mais simples que geram uma nova representação da entrada (GOODFELLOW; BENGIO; COURVILLE, 2016).

Cada função f_l utiliza parâmetros W para realizar a transformação dos dados do vetor de entrada v_1 de tal forma que a saída de uma função inicial f_1 gere um novo vetor $v_{(i+1)}$ que se tornará a entrada da próxima função f_2 e que terá como parâmetro os dados processados na função anterior (f_1). O conjunto de parâmetros é denotado por W_l, sendo l o índice da camada e x_i os dados de entrada. Dessa forma a função f pode ser descrita como:

$$f_L(\dots f_2(f_1(x_1, W_1); W_2)\dots), W_L) \tag{3.5}$$

Os métodos de DL aprendem a função f por meio da composição de funções, onde cada uma delas toma como entrada um vetor de dados que foi gerado pela função anterior, como representado na equação 3.5, fazendo com que a hierarquia das representações seja importante no contexto de DL, considerando que alterar as entradas afeta diretamente na saída gerada (PONTI; COSTA, 2017).

3.3 Redes Neurais Convolucionais

3.3.1 Introdução

Redes Neurais Convolucionais são um tipo de rede neural que possuem várias camadas e que fazem o uso da operação linear de convolução. São redes amplamente utilizadas em aplicações de visão computacional, pois sua estrutura foi feita especificamente para atender a dados bidimensionais, além de possuírem alto grau de invariância a diversas formas de distorções que os dados de entrada possam sofrer (HAYKIN, 2001).

Uma CNN é estruturada de tal forma que seja possível extrair as características relevantes dos dados de entrada, transmitindo todo esse conhecimento através da rede para que assim seja possível classificar ou reconhecer os dados. Cada camada de uma CNN

é composta por neurônios artificiais que estão totalmente conectados à camada anterior através de um vetor de pesos sinápticos. Tal organização permite que o valor de um neurônio seja computado como a soma dos pesos dos neurônios da camada anterior, sendo que cada neurônio pode aplicar uma função de ativação para introduzir a não linearidade ao problema (TABIK et al., 2017).

Em uma CNN cada uma de suas camadas possui níveis de abstração diferentes e cada camada transforma gradualmente um sinal de entrada. As primeiras camadas decompõem o sinal de entrada de forma a abstrair as características mais simples e as camadas mais profundas abstraem os elementos mais complexos do sinal (TABIK et al., 2017; ALMEIDA, 2019).

A classificação de imagens utilizando CNNs funciona propagando o sinal de entrada em todas as camadas da rede, sendo a saída da rede a resposta que informa a qual classe essa imagem pertence. Quanto maior a quantidade de camadas que uma CNN possui, maior a capacidade desta rede aprender características mais complexas e maior será a capacidade de generalização dessa rede (TABIK et al., 2017).

Para que uma CNN seja capaz de generalizar a sua saída para qualquer entrada fornecida, por vezes é necessário que seja utilizado uma grande quantidade de dados durante o treinamento, dependendo da complexidade da rede (OSÓRIO; BITTENCOURT, 2000; PONTI; COSTA, 2017). A quantidade de dados utilizados precisa ser coerente com a complexidade da CNN, visto que treinar uma rede complexa com uma pequena quantidade de dados pode levar ao *overfitting*, ou seja, a rede pode acabar se ajustando ao conjunto de dados de treinamento e sendo ineficaz para classificar entradas que não pertencem a este conjunto de dados (SRIVASTAVA et al., 2014).

3.3.2 Camadas características de Redes Neurais Convolucionais

As CNNs utilizadas atualmente são variações da LeNet-5, CNN proposta por LeCun *et al.* em seu estudo de 1998 intitulado "*Gradient-based learning applied to document recognition*", um tipo de rede neural que utiliza o gradiente descendente (AJIT; ACHARYA; SAMANTA, 2020; BENGIO, 2009). Existem três diferentes camadas que uma CNN possui e que a caracterizam (TABIK et al., 2017; LECUN; KAVUKCUOGLU; FARABET, 2010; AJIT; ACHARYA; SAMANTA, 2020) e elas são descritas a seguir.

3.3.2.1 Camadas Convolucionais

Uma camada convolucional é composta por filtros que são aplicados em cada pixel da imagem de entrada. Os filtros das camadas convolucionais são matrizes (kxk) de pesos e cada dos pesos é um parâmetro do modelo a ser aprendido. Os filtros produzem uma transformação da entrada, ou seja, uma combinação linear de todos os valores dos *pixels*

de uma vizinhança definida pelo tamanho do filtro e gerando uma matriz dos valores de ativação locais (PONTI; COSTA, 2017). Os filtros podem possuir tamanhos diversos e as regiões que eles processam são chamadas de campos receptivos locais (LECUN; KAVUKCUOGLU; FARABET, 2010), o valor de saída desses campos receptivos locais é a combinação linear de todos os *pixels* da vizinhança que foram transformados pelo filtro. A Figura 4 exemplifica um filtro 3x3. A quantidade de vezes que o filtro passa por um *pixel* pode ser definido pelo passo (*stride*). Um *stride* igual a 3 faz com que o filtro se desloque, sempre, três colunas da esquerda para a direita.

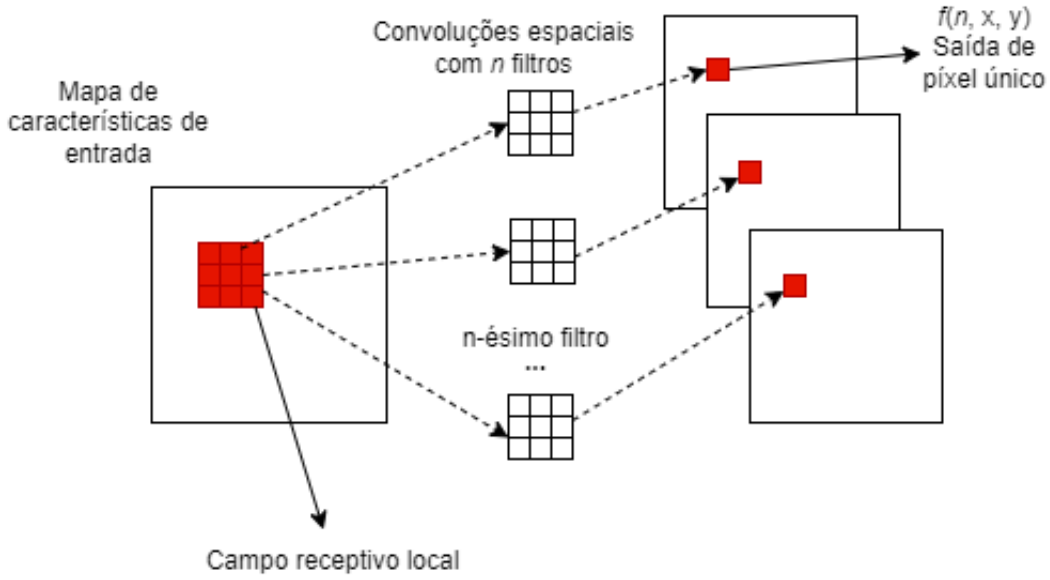

Figura 4 – Processo de convolução realizada em uma imagem de entrada da rede genérica. O filtro itera pela imagem e gera mapas de características que serão entradas para a camada seguinte. Fonte: Adaptada de Ponti *et. al*, 2017.

Cada neurônio da camada convolucional produz um novo vetor que recebe a função de ativação, esses vetores são chamados de mapas de ativação. Os mapas de características são empilhados de forma que servirão de entrada para a próxima camada.

Uma CNN pode ter tantas camadas convolucionais quanto o arquiteto que a planejou desejar, porém quanto maior a quantidade de camadas, maior a quantidade de parâmetros e, consequentemente, maior o tamanho da CNN e da disponibilidade computacional necessária para processá-la.

3.3.2.2 Camadas de agrupamento (*pooling*)

A camada de *pooling* é aplicada após as camadas convolucionais e tem a função de diminuir a dimensão espacial do vetor, reescalando a imagem de entrada. Essa operação é realizada visto que a saída das camadas convolucionais geram vetores com grandes profundidades.

O *max-pooling* é a técnica de *pooling* mais popular e consiste em gerar uma ativação máxima na matriz de entrada. (AJIT; ACHARYA; SAMANTA, 2020). A Figura 5 demonstra um exemplo de um *max-pooling* sendo aplicado a uma matriz em uma região de entrada 2x2.

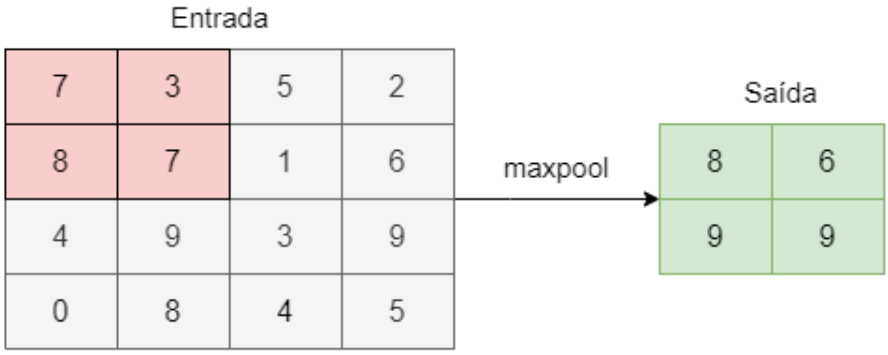

Figura 5 – *Max-pooling* de um mapa de características. Fonte: Elaborada pelo autor.

O *Global Average Pooling* é outra técnica de *pooling* utilizada quando deseja-se realizar a diminuição da taxa de amostragem para o mesmo tamanho do *kernel* dos mapas de características . Esse agrupamento é realizado calculando o valor médio de todos os elementos dos mapas de características. A camada de agrupamento em que é realizado o *Global Average Pooling* pode substituir a camada totalmente conectada na estrutura convencional de CNNs, dessa forma, reduzindo o armazenamento necessário para suportar as matrizes de pesos das camadas totalmente conectadas (HSIAO et al., 2019). A Figura 6 demonstra uma aplicação do *Global Average Pooling*.

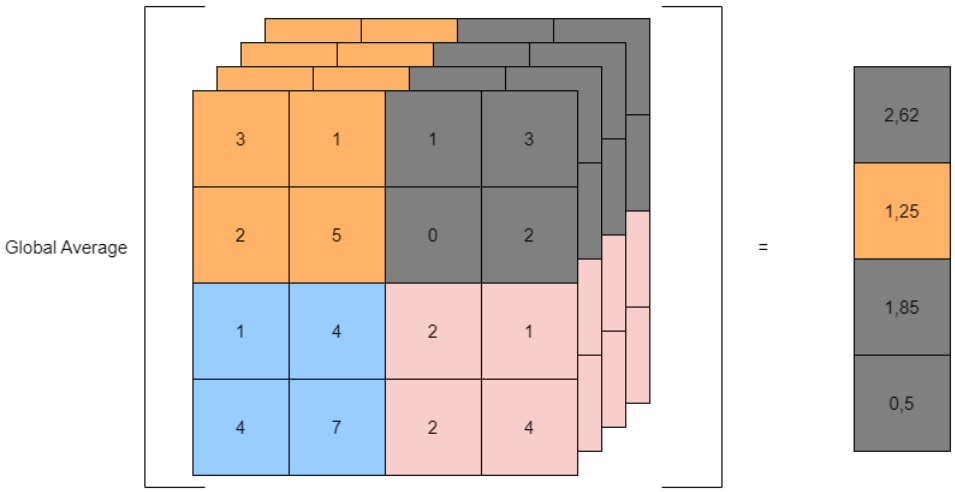

Figura 6 – *Global Average Pooling* de quatro mapas de características. Fonte: Elaborada pelo autor.

3.3.2.3 Camadas totalmente conectadas

As camadas totalmente conectadas possuem esse nome pois seus neurônios estão conectados a todos os neurônios da camada anterior. Essa camada geralmente se encontra na parte final da arquitetura e é responsáveis por realizar a classificação com base nas características extraídas pela parte convolucional da rede. Diferentemente das camadas convolucionais, as camadas totalmente conectadas identificam a entrada que recebem da camada anterior e produzem um valor escalar, ou seja, a camada totalmente conectada recebe um vetor de entrada e produz uma combinação linear desse vetor, de forma a gerar uma única saída (PONTI; COSTA, 2017).

3.3.3 Regularização de Redes Neurais Convolucionais

A regularização de CNNs é uma técnica utilizada para ajudar a controlar a complexidade do modelo, evitando que ele memorize os dados de treinamento e melhore sua capacidade de generalização através da minimização do erro (BISHOP, 2006). Dessa forma, evitando que ocorra o *overfitting* (SRIVASTAVA et al., 2014).

Existem várias técnicas de regularização que podem ser aplicadas a CNNs. O *Dropout* é uma das mais utilizadas e consiste em reduzir a co-adaptação dos neurônios desativando uma quantidade deles aleatoriamente, ou seja, durante a fase de treinamento, alguns neurônios e suas conexões são temporariamente removidos da rede de acordo com uma determinada probabilidade (SRIVASTAVA et al., 2014). Dessa forma, a rede é forçada a aprender recursos mais robustos, reduzindo a dependência excessiva de neurôrios específicos e consequentemente evitando o *overfitting*.

3.3.4 Redes Neurais Convolucionais e sua aplicação em sistemas embarcados

Sistemas embarcados possuem limitações computacionais que não são facilmente superadas como em computadores convencionais, logo, modelos de CNNs com muitos parâmetros podem não ser viáveis de serem implementados nesse tipo de *hardware*. Dessa forma, arquiteturas menores se tornam ideais, visto que quanto menor a quantidade de parâmetros, menor a quantidade de disponibilidade computacional exigida pela CNN.

Arquiteturas com poucos parâmetros podem oferecer desempenho similar à arquiteturas convencionais que possuem uma grande quantidade de parâmetros e ainda oferecem a flexibilidade de serem implementadas em ambientes diferenciados (IANDOLA et al., 2016).

Considerando que a comunicação entre servidores em arquiteturas de CNNs é um fator limitante para a escalabilidade de treinamento distribuído das redes, arquiteturas com poucos parâmetros possuem maior eficiência em treinamentos distribuídos. Modelos menores treinam mais rapidamente, pois requerem menos comunicação entre servidores. E,

como essas arquiteturas exigem menos comunicação entre os servidores, a realização de atualizações frequentes é mais viável, o que torna esse tipo de rede ideal para sistemas embarcados que geralmente exigem atualizações constantes (IANDOLA et al., 2016).

3.4 Processamento digital de imagens

3.4.1 Introdução

Uma imagem pode ser representada como uma matriz bidimensional constituída dos valores que correspondem a cada um de seus elementos. Tais elementos são chamados de *pixels* e cada *pixel* possui sua coordenada x, y para que seja possível identificar sua localização na imagem. Cada par de coordenadas forma uma função f em que sua amplitude indica a intensidade ou nível de cinza da imagem nesse ponto. Quando a intensidade da imagem e os *pixels* possuem valores discretos, tem-se uma imagem digital (GONZALEZ; WOODS, 2010).

O PDI refere-se ao campo de estudo em que computadores digitais processam imagens digitais e as manipulam para um determinado fim. É um campo de estudo de interesse crescente, pois viabiliza a aplicação em diversas áreas como na medicina, biologia, geografia e na visão computacional (ALMEIDA, 2019). Em geral, o processamento de imagens diz respeito a operações que são realizadas em imagens e que resultam em imagens (GONZALEZ; WOODS, 2010) e essas operações podem ser feitas utilizando conhecimentos de aprendizado de máquina aplicadas ao escopo de visão computacional.

Para que uma imagem possa ser utilizada digitalmente, primeiramente é preciso capturá-la do mundo real, o que pode ser feito por um equipamento qualquer, e então convertê-la para o formato digital. A conversão de imagens analógicas em imagens digitais é realizada em um processo de amostragem da função das coordenadas x e y da imagem e na sua amplitude (intensidade), esse processo é chamado de amostragem e transforma os valores contínuos da imagem em valores discretos (GONZALEZ; WOODS, 2010).

Entretanto, durante a captura da imagem e do processo de amostragem, é possível que a imagem possua ruídos que podem afetar negativamente as aplicações das operações de processamento de imagens. Para tentar reverter o máximo possível de imperfeições que as imagens adquiridas tenham, e afim de melhorar a qualidade delas, realiza-se o pré-processamento das imagens (ALMEIDA, 2019).

Para realizar o pré-processamento é feito o uso da filtragem espacial das imagens amostradas que é dada pela variação da intensidade por unidade de distância, criando um novo *pixel* com coordenadas iguais as coordenadas do centro da distância dos *pixels* de uma vizinhança, o resultado é o valor da filtragem (GONZALEZ; WOODS, 2010). Os filtros aceitam ou deixam passar certos tipos de frequência de modo a eliminar os ruídos

nas imagens.

A aplicação dos filtros gera uma nova imagem processada e existem várias estratégias que podem ser aplicadas para realizar as filtragens, dentre elas o Filtro Gaussiano e a Transformada Discreta de Fourier, duas técnicas que são discutidas a seguir.

3.4.2 Filtro Gaussiano

Filtros Gaussianos são amplamente utilizados durante o pré-processamento de imagens através da filtragem espacial. Tais filtros possuem a função de borrar e diminuir o ruído das imagens de entrada durante o pré-processamento, removendo pequenos detalhes não necessários para as imagens (GONZALEZ; WOODS, 2010).

Matematicamente, a função Gaussiana para uma dimensão pode ser definida pela Equação 3.6. Onde σ é o desvio padrão da distribuição. Essa equação é descrita pelos parâmetros de desvio padrão e média da curva de Gauss e, conhecidos esses valores, é possível determinar qualquer probabilidade de uma distribuição normal.

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \tag{3.6}$$

No contexto de imagens, a aplicação do Filtro Gaussiano é dada pela convolução da imagem com o Filtro Gaussiano e é necessário utilizar a Gaussiana de duas dimensões, que consiste no produto de duas funções Gaussianas e é dada pela Equação 3.7.

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.7}$$

Figura 7 – Imagem original antes da aplicação do Filtro Gaussiano. Fonte: Elaborada pelo autor.

A Figura 8 é um exemplo de uma imagem suavizada após a aplicação do Filtro Gaussiano. Na imagem modificada é possível notar uma atenuação dos detalhes relevantes da imagem original (Figura 7).

Figura 8 – Imagem suavizada após a aplicação do Filtro Gaussiano. Fonte: Elaborada pelo autor.

3.4.3 Transformada Discreta de Fourier

A Transformada Discreta de Fourier (*Discrete Fourier Transform*, DFT) é uma representação de Fourier para sequências de séries finitas que possuem amostras discretas no tempo. É uma técnica amplamente utilizada para a manipulação de imagens, que pode ser aplicada para o desenvolvimento de filtros que buscam o realce e restauração de áreas nas imagens (GONZALEZ; WOODS, 2010).

A DFT é utilizada para filtrar imagens no domínio da frequência o que consiste em aplicar a DFT em uma imagem para modificá-la, consequentemente obtendo um resultado no domínio da frequência, e depois aplicar a transformada inversa para obter o resultado processado.

A DFT de um sinal qualquer $x[n]$ é definida como na equação 3.8 e a sua inversa é definida como mostra a equação 3.9. As mudanças de domínios do sinal de entrada não causam a perda de informações do sinal.

$$X(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n} \tag{3.8}$$

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\Omega})e^{j\Omega n}d\Omega \tag{3.9}$$

Os valores $x[n]$ são considerados como os valores de um sinal em tempos igualmente espaçados $t = 0, 1, ..., N-1$. A saída $X(e^{j\Omega})$ é um número complexo que codifica a amplitude e a fase de uma onda senoidal com frequência Ω ciclos por unidade de tempo.

3.5 Sistemas embarcados

Sistemas embarcados são sistemas que processam informações e que são embutidos em um sistema maior. Segundo Marwedel (2006), são sistemas que devem ser autossuficientes e todos os dispositivos utilizados para o seu funcionamento devem integrá-lo de

modo a ser um sistema independente da ação de outro sistema para ser operado. Além de serem sistemas com funcionalidade única, possuem restrições rígidas quanto ao projeto em que estão sendo implementados e devem ser eficientes ao executar o propósito da sua tarefa, utilizando a menor quantidade possível de recursos disponíveis.

Outra característica importante de sistemas embarcados é o fato de eles serem reativos ao ambiente em que são implementados, ou seja, devem reagir a mudanças no ambiente e processarem essas informações em tempo real de modo a fornecerem respostas imediatas (ALMEIDA, 2019).

Neste trabalho, existe a necessidade de resposta em tempo real do sistema e também é preciso que o dispositivo desenvolvido consiga executar suas tarefas sem a necessidade de estar conectado a um computador *desktop* convencional, dessa forma, é utilizada uma placa pertencente à família Raspberry Pi para compor o sistema embarcado.

3.6 Ferramentas de Software

Para o desenvolvimento desse projeto, diversas ferramentas de software foram utilizadas. A breve descrição de tais ferramentas é apresentada a seguir.

3.6.1 A linguagem de programação Python e bibliotecas utilizadas

Python é uma linguagem de programação de alto nível, multiparadigma, orientada a objetos, funcional e de tipagem dinâmica, e se caracteriza por apresentar uma sintaxe simples e poderosa para o desenvolvimento de algoritmos. Seu fácil entendimento e a baixa complexidade de implementação para a programação a tornou uma das linguagens mais populares do mundo, sendo utilizada para diversos fins, inclusive por companhias como o Google e a NASA (HETLAND, 2005). Idealizada por Guido Van Rossum e desenvolvida pela Python Software Foundation em 1991, a linguagem de programação Python tem a intenção de ser uma linguagem que dá ênfase a legibilidade do código e sua sintaxe permite que os programadores expressem conceitos em poucas linhas de códigos.

A linguagem de programação Python possui diversas bibliotecas, muitas delas foram desenvolvidas por programadores que sentiram a necessidade de ampliar a capacidade da linguagem. Dentre essas bibliotecas, existem aquelas que facilitam a manipulação de vetores e matrizes e fizeram com que o Python se tornasse uma linguagem popular para o desenvolvimento de algoritmos de inteligência artificial e aprendizado de máquina.

Para este trabalho, foram utilizadas bibliotecas de aprendizado de máquina, aprendizado profundo, manipulação e visualização de dados, e bibliotecas para manipulação de vetores e matrizes. A Tabela 2 faz uma relação das bibliotecas utilizadas para o desenvolvimento desse projeto e a sua aplicabilidade.

Bibliotecas	**Aplicabilidade**
numpy	Vetores e funções matemáticas
matplotlib, sklearn	Visualização de dados
os, sys, time	Sistema
Tensorflow, keras, Tesorflow Lite	Aprendizado de máquina e Aprendizado profundo
opencv	Visão computacional
picamera, RPi.GPIO	Sistemas embarcados
tkinter	Interfaces gráficas
pickle	Serialização de objetos

Tabela 2 – Bibliotecas Python utilizadas e sua aplicabilidade. Fonte: Elaborada pelo autor.

3.6.2 TensorFlow e TensorFlow Lite

O TensorFlow [1] é uma biblioteca flexível e escalável que pode ser utilizada para desenvolver algoritmos em diversos contextos. É uma biblioteca que permite que o treinamento e desenvolvimento de algoritmos de aprendizado de máquina de forma eficiente, permitindo uma série de implementações e manipulações das aplicações. Possui uma série de interfaces de programação de aplicação (*Application Programming Interface*, API) em diversas linguagens de programação, sendo que a API para Python é a mais estável dentre elas.

O TensorFlow possui várias funções que facilitam a construção de modelos de aprendizado de máquina e que fazem operações matemáticas, como adições e multiplicações, em sua estrutura básica, os chamados tensores. Tais tensores são uma generalização de vetores e matrizes multidimensionais.

Neste trabalho duas APIs são utilizadas para o desenvolvimento do projeto, sendo elas a API para Python versão 2.9.2 para realizar o treinamento da CNN e para a criação do modelo utilizado no SBC, e o Tensorflow Lite, API utilizada para aplicações móveis e para dispositivos de borda, utilizado para que seja possível a execução do modelo no SBC.

3.6.3 Keras

O Keras[2] é uma biblioteca escrita em Python utilizada para a criação de modelos de redes neurais e possui uma série de rotinas variadas, dentre elas, rotinas para classificação de imagens, detecção de objetos em imagens, otimizadores e métricas para avaliação de modelos de aprendizado profundo. Pode ser utilizado sobre o TensorFlow e atualmente o incorpora, sendo um módulo do TensorFlow. O Keras é amplamente utilizado para o desenvolvimento deste trabalho.

[1] www.tensorflow.org
[2] https://keras.io/

3.6.4 OpenCV

O OpenCV[3] é uma biblioteca de código aberto e conta com mais de 2500 algoritmos otimizados que podem ser utilizados para implementar algoritmos para a detecção e reconhecimento de faces, detecção de objetos em imagens e vídeos, entre outras funções relacionadas à aplicações de visão computacional. O OpenCV possui diversas funções para a manipulação de imagens como a mudança de escala de cores, redimensionamento, filtros de suavização, detecção de bordas, entre outras.

Devido as suas funcionalidades, essa biblioteca foi utilizada para o pré-processamento das imagens do banco de imagens desenvolvido para este trabalho, visto que suas funções são extremamente úteis e atendem às necessidades do projeto.

3.6.5 Google Colaboratory

O Google Colaboratory[4] ou Colab é um serviço em nuvem gratuito criado pela Google Research que permite escrever e executar programas em Python. O Colab é um serviço de notebooks hospedados do Jupyter que não necessitam de configuração para serem utilizados e proporcionam acessos à GPU gratuitamente e ao compartilhamento de projetos entre usuários.

O Colab foi pensado como uma maneira de incentivar o estudo de aprendizado de máquina e inteligência artificial e possui no seu ambiente as principais ferramentas para este fim, como o TensorFlow, Numpy, Keras, Matplotlib entre outras bibliotecas amplamente utilizadas para o aprendizado de máquina. Devido a essas facilidades, o Colab foi escolhido para ser parte deste projeto, visto que a necessidade de recursos computacionais elevados são necessários para o desenvolvimento da CNN e, como o desenvolvimento deste trabalho envolveu um conjunto de pessoas, existe a necessidade do compartilhamento dos códigos entre os integrantes do grupo.

[3] https://opencv.org/
[4] https://colab.research.google.com/

4 Metodologia

Neste capítulo é apresentada a metodologia utilizada para o desenvolvimento deste trabalho e os passos para concluí-la. Também são especificadas as características da base de dados utilizada para a fase de treinamento e testes do modelo de rede neural convolucional utilizado, assim como as manipulações realizadas nas imagens de teste. E, por fim, são apresentadas as métricas utilizadas para avaliar os resultados obtidos.

4.1 Metodologia de pesquisa

Com o objetivo de desenvolver a pesquisa e atrelado a necessidade de entender as arquiteturas de CNNs, tem-se como problema de pesquisa o entendimento de tais arquiteturas, a definição da escolha ideal para a implementação em SBCs e qual a viabilidade de sistemas que as implementam ao classificar caracteres manuscritos. Define-se a seguinte questão que delineará o trabalho no decorrer da pesquisa: "Qual arquitetura de Redes Neurais Convolucionais que melhor se encaixa para a classificação de caracteres em imagens no contexto de implementação em Computadores de Placa Única?".

Esta pesquisa tem como objetivo compreender arquiteturas de CNNs implementadas em SBCs através de testes e análises de algoritmos que realizam a classificação de caracteres. Dessa forma, define-a como uma pesquisa exploratória, considerando que seu objetivo é proporcionar uma maior familiaridade como o tema em estudo. A pesquisa exploratória possui um planejamento flexível e que permite o desenvolvimento do tema em diversos ângulos e aspectos, envolvendo a análise de exemplos que estimulem a compreensão do pesquisador em relação à pesquisa (PRODANOV; FREITAS, 2013). Como método científico, optou-se por utilizar o método experimental para a realização dos procedimentos metodológicos, visto a necessidade de analisar, classificar e interpretar os resultados obtidos durante os testes realizados no decorrer da pesquisa. No caso desta pesquisa, o objetivo é classificar caracteres manuscritos utilizando sistemas embarcados.

Como técnica de coleta de dados é utilizada a análise de conteúdo, considerando que são utilizados algoritmos disponíveis na literatura para a classificação de imagens e tais algoritmos são apenas modificados para se adaptarem para o problema de pesquisa. É desenvolvido apenas o algoritmo de pré-processamento de imagens e para o desenvolvimento da base de dados para a testagem do classificador embarcado. Quanto a análise de dados, os resultados são analisados de forma quantitativa, através da análise descritiva de conteúdo dos dados obtidos, visto que os resultados podem ser quantificados.

A Figura 9 representa o clico da pesquisa utilizado neste trabalho e inicia-se

pela busca e seleção do algoritmo de CNN a ser utilizado no dispositivo. Seguido da implementação e testes do algoritmo de CNN ao classificar caracteres. A montagem do *hardware* se caracteriza em duas fases, sendo a primeira pela montagem e configuração do SBC para que capture imagens e a segunda fase se caracteriza pela implementação da CNN no *hardware* da placa.

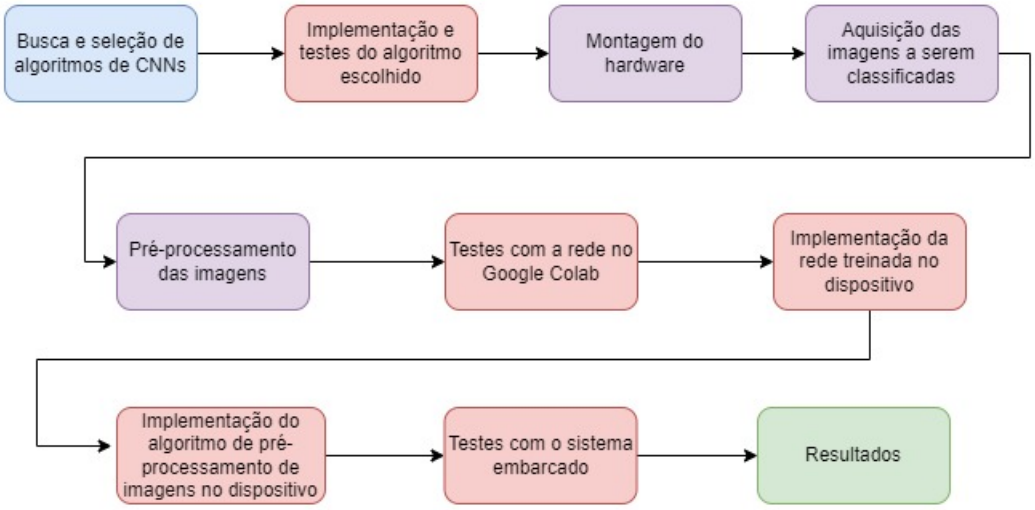

Figura 9 – Ciclo de pesquisa adotado. Formado pela busca, seleção e implementação de algoritmos de CNNs, montagem do *hardware*, aquisição e ajuste de imagens, testes no ambiente virtual do Google Colab e no ambiente do sistema embarcado e, finalmente, os resultados obtidos. Fonte: Elaborada pelo autor.

A aquisição das imagens se caracteriza por ser a fase em que as imagens da base de dados utilizadas para a classificação de caracteres são capturadas, essas imagens são pré-processadas na fase seguinte para que se adequem aos padrões definidos pela CNN. A base de dados é utilizada tanto no ambiente do Google Colab quanto no sistema embarcado, onde serão realizados testes para a obtenção de resultados da classificação nas dez classes definidas que correspondem aos dígitos de 0 a 9.

4.2 Base de dados

Para o desenvolvimento da pesquisa é utilizado um banco de imagens público chamado MNIST para realizar o treinamento da CNN utilizada no sistema embarcado final do projeto. O MNIST é comumente empregado para o estudo e treinamento de sistemas de processamento de imagens, possuindo 70000 imagens, desse total, 60000 imagens são para treino e 10000 para testes. A escolha para a utilização desse banco de imagens se dá devido a facilidade de lidar com suas imagens e a quantidade expressiva de dados que ele possui. Na Figura 10 é possível ver um exemplo de algumas imagens do MNIST.

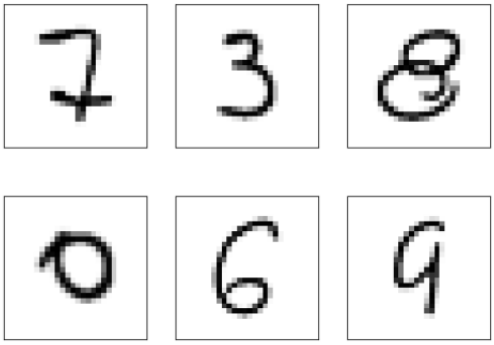

Figura 10 – Exemplo de imagens do MNIST usadas no treinamento da CNN. Fonte: Elaborada pelo autor.

Para os testes e validação da rede treinada, é utilizado uma base de dados produzida por um dos integrantes do grupo de pesquisa. Tal base de dados é formada por imagens de dígitos manuscritos que possuem pouca ou nenhuma variação de intensidade, brilho e contraste, pois foram capturadas em um ambiente controlado para assegurar a acurácia do modelo da CNN.

A Figura 11 representa um exemplo das imagens da base de dados própria após o pré-processamento das imagens. As mudanças realizadas são explicitadas posteriormente.

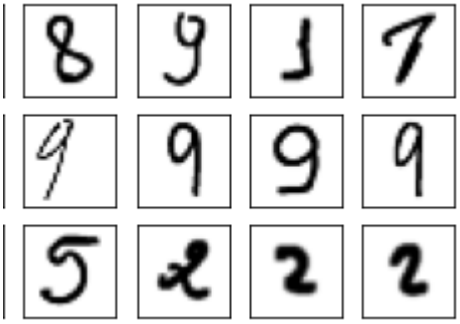

Figura 11 – Exemplo de imagens do banco de dados próprio desenvolvido. As imagens desse exemplo foram modificadas durante o pré-processamento (Seção 4.3) para que ficassem o mais similar possível às imagens do MNIST. Fonte: Elaborada pelo autor.

4.3 Algoritmo para o pré-processamento de imagens

Para que a rede neural seja capaz de classificar o dígito presente na imagem de forma correta é necessário que a imagem de entrada atenda aos requisitos da rede, ou seja, a dimensão da imagem, quantidade de canais e as características do conteúdo da imagem como, por exemplo, a forma como o dígito está disposto na imagem, precisam

ser semelhantes aos das imagens utilizadas no treinamento da rede neural para que seja possível uma correta classificação da rede.

Para que uma imagem qualquer que não pertence à base de dados utilizada durante o treinamento da rede atenda a tais requisitos, é preciso realizar um pré-processamento dessa imagem de forma a deixá-la o mais próximo possível dos padrões utilizados na base de dados de treino da rede neural.

Dessa forma, neste trabalho é desenvolvido, com a ajuda dos integrantes do grupo de pesquisa, um algoritmo para realizar o pré-processamento das imagens que fazem parte do banco de dados próprio, para que tais imagens estejam o mais semelhante possível das imagens do MNIST, base de dados utilizado para treino. São utilizadas as bibliotecas Numpy, Matplotlib, OpenCV, Os e Tensorflow para o desenvolvimento do algoritmo de pré-processamento.

O pré-processamento das imagens é realizado por uma função que recebe como entrada uma imagem da base de dados própria a ser processada. A imagem recebida por essa função é uma imagem "bruta", ou seja, contém ruído e outras informações que podem prejudicar a classificação da imagem pela rede. Cabe ao pré-processamento tratar a entrada, eliminando o ruído e o conteúdo indesejado (GONZALEZ; WOODS, 2010). Dessa forma, o algoritmo desenvolvido transforma a imagem de entrada de tal modo que, na saída, a imagem tenha as características requisitadas pela rede. A Figura 12 é a representação de uma imagem capturada pela câmera do sistema embarcado antes de ser pré-processada.

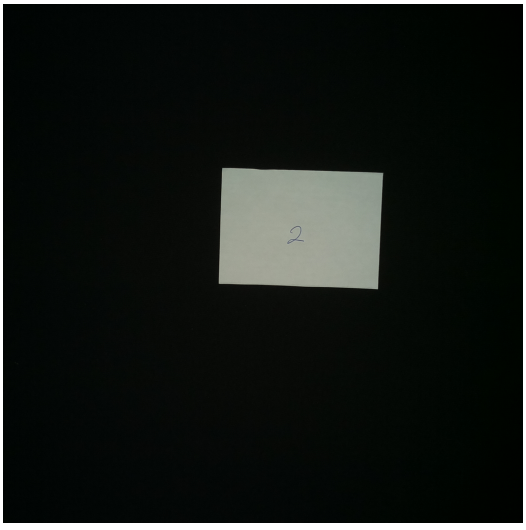

Figura 12 – Imagem capturada pela câmera Raspberry Rev 1.3. Fonte: Elaborada pelo autor.

O primeiro passo do pré-processamento é delimitar uma região de interesse. O dígito manuscrito geralmente se encontra no centro da imagem, então é realizado um recorte na entrada original de modo que a saída gerada seja igual a um quarto da área

da imagem original. A função que realiza o corte central recebe como entrada a imagem a ser processada e as dimensões desejadas na saída, criando-se uma nova imagem que será utilizada nos próximos passos do pré-processamento, conforme pode ser observado na Figura 13.

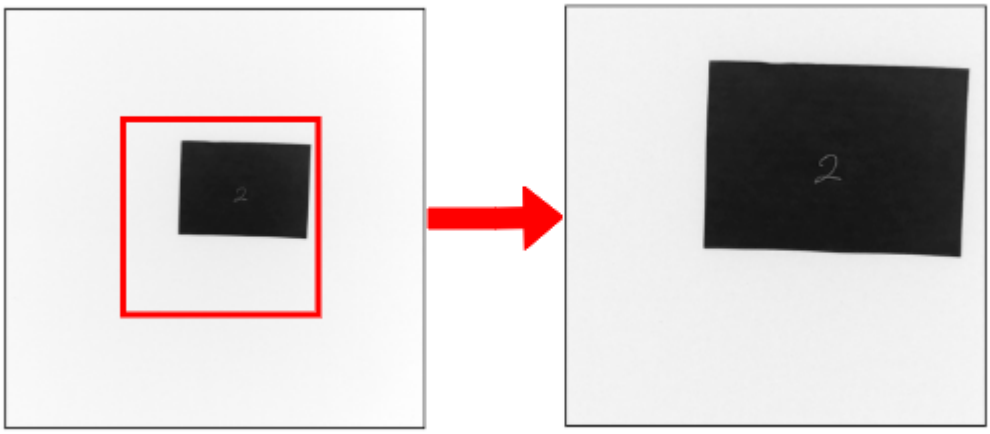

Figura 13 – Função corte central. Primeira parte do pré-processamento. Consistem em cortar a imagem de entrada de modo a destacar apenas a região de interesse que será enviada para as próximas etapas do pré-processamento. Fonte: Elaborada pelo autor.

O próximo passo é a aplicação do Filtro Gaussiano de modo a atenuar o ruído da imagem de entrada. O OpenCV possui uma função que implementa o Filtro Gaussiano, chamada de *cv.GaussianBlur()* que recebe como entrada a imagem a ser aplicada o filtro e a altura e largura do *kernel*, ou seja, a matriz de convolução a ser utilizada. Por comodidade, essa função é utilizada neste trabalho. A aplicação do Filtro Gaussiano pode ser visualizada na Figura 14.

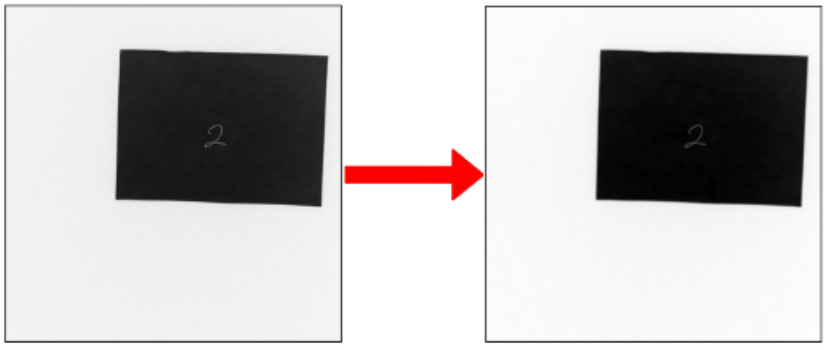

Figura 14 – Aplicação do Filtro Gaussiano afim de atenuar o ruído da imagem de entrada. Fonte: Elaborada pelo autor.

A imagem obtida após a aplicação do Filtro Gaussiano é então enviada para a função de limiarização da região de interesse. Tal processo consiste em atribuir um valor

para cada pixel do sinal de entrada, gerando uma nova imagem na saída. Essa nova imagem é baseada em um valor limiar que pode ser escolhido arbitrariamente. Se o valor do pixel for menor que o valor do limiar, o novo valor do pixel é 0, caso seja maior, o novo valor será o máximo possível, ou seja, 255 para imagens de 8 bits. Após esse processo, espera-se que os pixels pertencentes ao dígito estejam com valor igual a 0, totalmente diferentes dos pixels do fundo da imagem, que devem estar com valor igual a 255. Novamente, o OpenCV possui uma função que realiza a limiarização da imagem, o resultado da utilização dessa função pode ser visualizado na Figura 15.

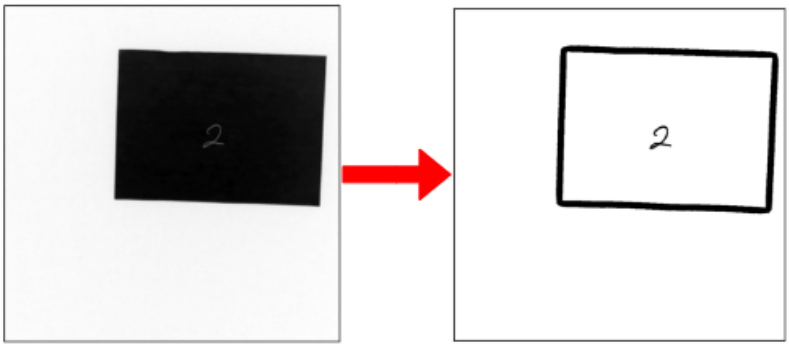

Figura 15 – Limiarização da imagem após a aplicação do Filtro Gaussiano. Fonte: Elaborada pelo autor.

O processo de limiarização é necessário para que seja possível encontrar o local exato onde o dígito está localizado na imagem, o objetivo é realizar mais uma etapa de recorte, dessa vez em torno do dígito. O recorte em torno do dígito é feito com base nos valores dos pixels da imagem. A função de recorte age sobre os pixels de valor 0, recortando um retângulo em torno da área dos pixels, conforme mostra a Figura 16.

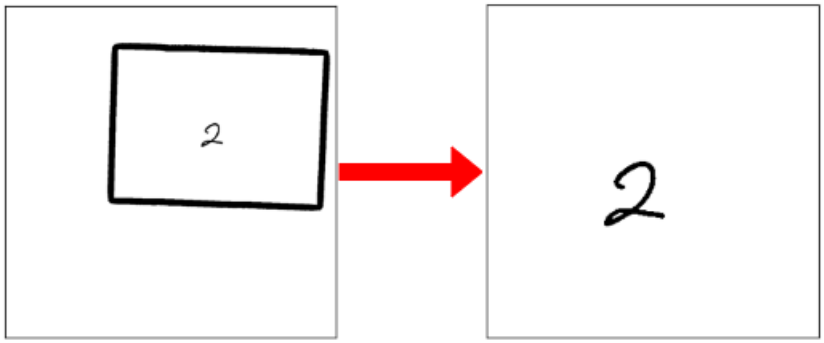

Figura 16 – Corte realizado ao redor do dígito para centralizá-lo na imagem. Essa etapa é feita identificando o área dos *pixels* do dígito. Fonte: Elaborada pelo autor.

Após essa fase, é aplicada a erosão e a dilatação de modo a reduzir os ruídos das imagens. Essas funções também são aplicadas utilizando bibliotecas do OpenCV. O

resultado dessa etapa pode ser visualizado na Figura 17.

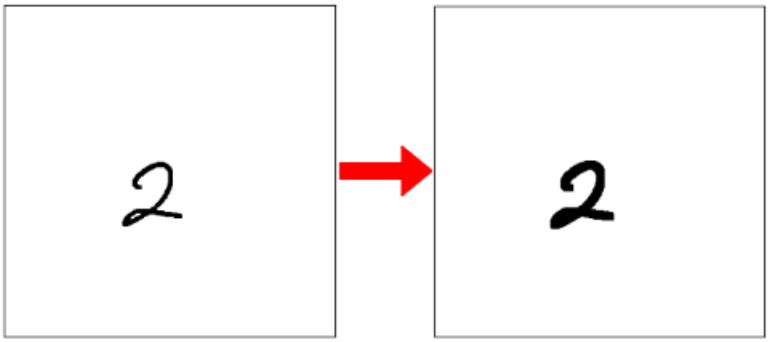

Figura 17 – Resultado da erosão e dilatação aplicadas à imagem de entrada. Esses processos são realizados para diminuir o ruído da imagem. Fonte: Elaborada pelo autor.

Também é realizado um novo corte na imagem resultante da fase anterior, de modo que o dígito fique mais destacado na imagem. Após o novo recorte o Filtro Gaussiano é mais uma vez aplicado, dessa vez para suavizar as bordas do dígito, conforme a Figura 18.

Figura 18 – Corte para destacar o dígito na imagem e aplicação do Filtro Gaussiano uma segunda vez para suavizar as bordas do dígito. Fonte: Elaborada pelo autor.

Por último, é realizado o redimensionamento da imagem, de modo a deixá-la com as mesmas dimensões que as imagens do MNIST possuem (28x28). Após a aplicação dessas funções, é obtida uma imagem semelhante às imagens de treinamento. As imagens de saída dessa função estão prontas para serem utilizadas para a classificação do algoritmo de CNN. O resultado final pode ser visto na Figura 19.

4.4 Algoritmo de visão computacional para a classificação de caracteres

Inicialmente, realizou-se a busca de algoritmos de CNNs implementados na linguagem de programação Python e que realizassem a classificação de dígitos manuscritos em imagens. A ideia inicial era verificar a estrutura desses algoritmos e escolher dentre eles o ideal para o projeto. A priori, é escolhido o algoritmo de classificação de imagens

Figura 19 – Imagem resultante da etapa do pré-processamento. As imagens pré-processadas são similares às imagens do MNIST que foram utilizadas na fase de treinamento com 28x28 de resolução. Fonte: Elaborada pelo autor.

do MNIST apresentado no minicurso "*Deep Learning: unidades de processamento densa, convolucional, recorrente, e estratégias de transferência de aprendizado*" que aconteceu na 4° Escola Avançada de Big Data Analysis em 2020. Porém, com o decorrer da pesquisa, é possível identificar que os melhores algoritmos que poderiam compor o protótipo final do trabalho são aqueles que possuem, como princípio, serem implementados em SBCs, ou seja, algoritmos com menos parâmetros e que possam ser comprimidos para um formato suportado pela placa escolhida para o projeto. Tal fato também é comprovado ao decorrer da Revisão Sistemática da Literatura como descrito na Seção 2.

Dessa forma, inicia-se a busca por CNNs que possuam inclinação para serem embarcadas. Após as buscas, foi decidido utilizar a *SqueezeNet* (IANDOLA et al., 2016) para fazer parte do sistema embarcado desenvolvido no projeto. As características da SqueezeNet se mostram adequadas ao projeto aqui desenvolvido, visto que ela é de fácil implementação e entendimento. Além de sua arquitetura suprir as necessidades da pesquisa.

4.4.1 *SqueezeNet*

A *SqueezeNet* é uma arquitetura de CNN proposta por Iandola *et al.* em seu estudo de 2016 intitulado *"Squeezenet: Alexnet-level accuracy with 50x fewer parameters and < 1mb model size"*. A arquitetura possui menos parâmetros do que modelos amplamente utilizados na literatura como a AlexNet (KRIZHEVSKY; SUTSKEVER; HINTON, 2017) e a LeNet-5 (LECUN et al., 1998), mas apresenta acurácia equivalente a tais modelos. A Figura 20 faz um comparativo relacionando o total de parâmetros de algumas arquiteturas de redes neurais mais conhecidas e utilizadas para a classificação de imagens e da *SqueezeNet*.

Para atingir a acurácia similar a estes modelos citados são implementadas três estratégias principais descritas a seguir:

- Atingir a diminuição da quantidade de parâmetros através da substituição de filtros convolucionais 3x3 por filtros convolucionais 1x1;

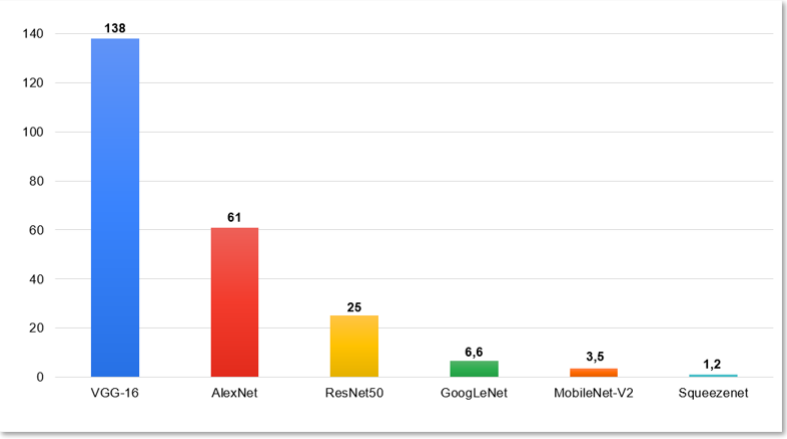

Figura 20 – Comparação do total de parâmetros (em milhões) da *SqueezeNet* e de outras arquiteturas de redes neurais amplamente utilizadas no contexto de classificação de imagens e objetos. Fonte: Elaborada pelo autor.

- Diminuir o número de canais da entrada para filtros convolucionais 3x3 utilizando *squeeze layers* de tal forma que a estratégia anterior seja válida e que o número de parâmetros seja mantido pela rede;
- Realizar a redução da amostragem do sinal apenas em camadas mais profundas da rede de modo que as camadas convolucionais tenham mapas de ativação maiores. Essa estratégia é utilizada para que a acurácia seja maximizada mesmo com uma quantidade limitada de parâmetros.

Outro ponto importante e o diferencial da *SqueezeNet* são os módulos denominados *Fire module*. Cada um dos *Fire modules* é composto por três hiperparâmetros dimensionais e possuem uma camada convolucional de compressão, que possui apenas filtros 1x1, e por uma camada de expansão que possui uma combinação de filtros 1x1 e 3x3. Os filtros que compõem a camada de compressão (*squeeze layer*) são configurados de tal forma que a sua quantidade não ultrapasse a soma da quantidade de filtros da camada de expansão (IANDOLA et al., 2016).

4.4.1.1 Arquitetura

A arquitetura original da SqueezeNet conta com uma camada convolucional (conv1), seguida de 8 *Fire modules* (fire2-9), finalizando com uma camada convolucional (conv10), gradualmente aumentando o número de filtros a cada *Fire module* do início ao final da rede. O Caffe *framework* é utilizado no trabalho de Iandola *et. al* (2016), esse *framework* não realiza a convolução de camadas que possuem filtros de múltiplas resoluções, dessa forma é implementado uma camada de expansão e duas camadas de convoluções, uma com filtros 1x1 e outra com filtros 3x3. As saídas de tais camadas processadas são concatenadas no canal de dimensão da imagem. Realizar a operação dessa forma é numericamente equivalente a

implementar uma camada que possui os filtros 1x1 e 3x3 (filtros multidimensionais). A Figura 21 ilustra a arquitetura original da SqueezeNet.

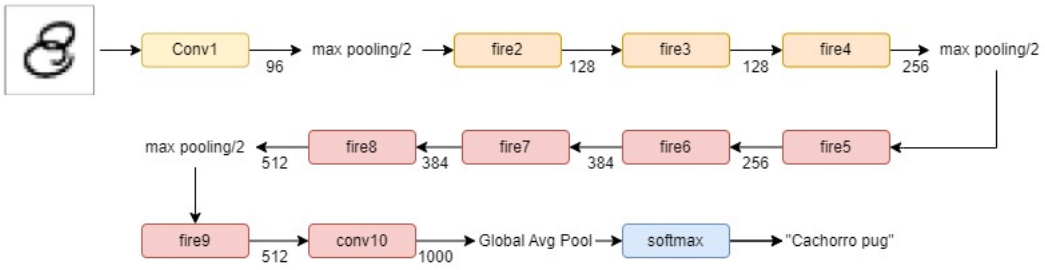

Figura 21 – Arquitetura original da SqueezeNet. Fonte: Adapatada de Iandola *et.al* (2016).

A arquitetura original da SqueezeNet tem como entrada imagens de 224x224 *pixels*, sendo diferente das imagens do MNIST que possuem imagens de tamanho 28x28 como especificado na Subseção 4.2. Dessa forma, é necessário adaptar a arquitetura da *SqueezeNet* para que ela receba as imagens de entrada com o tamanho das imagens que pretende-se utilizar.

Ao realizar as buscas, pôde-se encontrar um algoritmo que possuía a arquitetura da *SqueezeNet* adaptado para ser utilizado para treino e classificação de imagens do CIFAR-10, banco de imagens amplamente utilizado para a classificação de imagens utilizando CNNs. Essa rede recebe como dados de entrada imagens com dimensões 32x32 e com 3 canais de cores (RGB). Para este algoritmo ser utilizado no projeto que aqui é apresentado, é necessário alterar a entrada para que ela receba imagens com as dimensões do MNIST. Os canais de cores são alterados para receber apenas imagens em escala de cinza (1 canal).

A arquitetura utilizada para realizar os experimentos desse trabalho pode ser visualizada na Figura 22. A camada de entrada recebe imagens de resoluções 28x28 e a camada de saída possui a distribuição de probabilidade da função *softmax* dentre as 10 classes que a rede pode classificar.

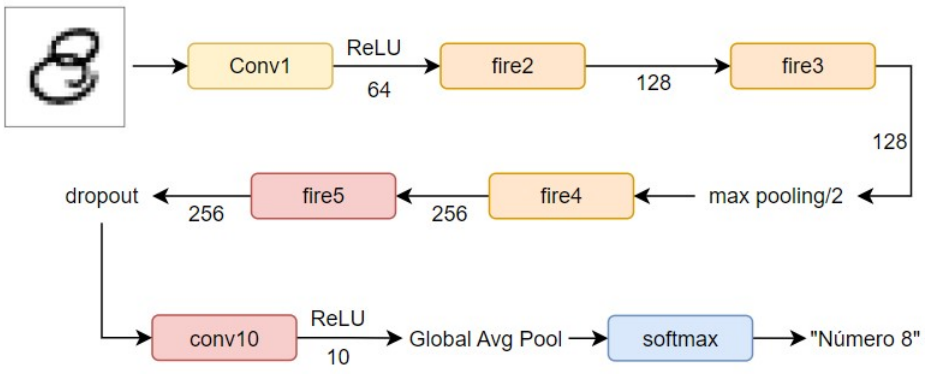

Figura 22 – Arquitetura da SqueezeNet adaptada para o treino utilizando imagens de dimensão 28x28. Fonte: Elaborada pelo autor.

As camadas ocultas são formadas por 4 *Fire modules* (fire2-5), uma camada de subamostragem que utiliza o *max pooling* entre o segundo e o terceiro *Fire modules*. Duas camadas convolucionais, uma camada de *Dropout* para a regularização e uma camada que utiliza o *Global Avarage Pooling*. São utilizadas como funções de ativação a Unidade Linear Retificada (*Rectified Linear Unit*, ReLU) e a *Softmax*.

4.4.2 Treinamento

A próxima etapa do projeto é realizar o treinamento da CNN. É utilizada a função *fit_generator()* do Keras que tem como parâmetros de entrada vetores do tipo *numpy* e os *labels* de cada elemento desse vetor. O modelo utiliza o BP como algoritmo de aprendizagem e o RMSprop como otimizador.

A função *fit_generator()* permite que seja utilizado uma base de dados gerada através das imagens da base de dados de treino, gerando-as através de *data augmentation*, esse deve ser o primeiro parâmetro de entrada da função. Também é possível definir a quantidade de épocas de treino até a parada, o total de etapas a serem consideradas antes de uma época ser declarada como concluída, os dados de validação a serem utilizados ao final de cada época, entre outros parâmetros que são relacionados ao treinamento da rede.

O treinamento da *SqueezeNet* é realizado em 45 épocas e cada uma delas corresponde a uma execução do algoritmo BP através dos dados. A quantidade de épocas necessárias para o treinamento é avaliada em relação a curva de aprendizado que relaciona os dados utilizados para o treino e os dados utilizados para a validação do modelo. Os pesos sinápticos da rede são atualizados a uma taxa de aprendizagem igual a 0,0001 e com o lote (ou *batch*) igual a 32 amostras por atualização do gradiente, esses dois dados são os hiperparâmetros definidos para o treinamento e são valores fixos.

Como discutido anteriormente, o MNIST é utilizado como base de dados de treino e validação da rede neural, sendo utilizadas 60000 imagens para o treinamento e 10 *labels* que correspondem aos dígitos de 0 a 9.

O treinamento é realizado no Google Colab devido a facilidade em compartilhar os detalhes das etapas do processo de implementação com todas as pessoas envolvidas no desenvolvimento do projeto, além da disponibilidade de recursos computacionais como discutido na Seção 3.6. Dessa forma, apenas a parte *forward* da rede foi executada no Raspberry Pi, sendo toda a parte anterior a esta executada em computadores convencionais, mais especificamente no ambiente virtual do Google Colab.

Após o treinamento, o modelo é salvo em dois formatos: um no formato *.h5* para ser utilizado na classificação no Google Colab; e outro no formato *.pb* que posteriormente será utilizado para conversão para o formato *.tflite*, extensão utilizada pelo Tensorflow Lite. É necessário salvar o modelo em *.pb*, pois a função de conversão do Keras só converte

arquivos desse tipo para o formato *.tflite*.

Os testes de classificação da rede são inicialmente realizados no ambiente do Google Colab. As imagens utilizadas nessa etapa são pertencentes à base de dados desenvolvida para este trabalho. Para realizar a classificação dessas imagens, primeiramente elas precisam passar pelo pré-processamento para se adequarem às imagens de entrada da rede e as suas características, como discutido na Seção 4.3.

As imagens da base de dados desenvolvida estão armazenadas no Google Drive em sua forma bruta, isto é, ainda não passaram pelo pré-processamento, sendo este apenas no momento da classificação. Tal abordagem é utilizada para que sejam evitados possíveis erros de classificação advindos do armazenamento das imagens pré-processadas no Google Drive, considerando-se que a forma como essas imagens são armazenadas possam afetar o pré-processamento.

Para classificar a base de dados, é necessário carregar o modelo treinado utilizando a função *load.model()* que tem como parâmetro o endereço em que o modelo está armazenado. Após o carregamento do modelo é realizado o pré-processamento das imagens. E, então, é realizada a classificação das imagens, utilizando a função *model.predict()* que tem como argumento uma imagem a ser classificada. Para classificar todas as imagens da base de dados, elas são armazenadas em uma lista e o algoritmo de classificação a itera, classificando uma imagem por vez.

4.5 Computador de Placa Única para compor o sistema embarcado

4.5.1 Especificações do *hardware*

A Raspberry Pi 3B+ é um modelo de SBC fabricado pela Raspberry Pi Foundation com o intuito de incorporar sistemas embarcados. É uma placa de baixo custo, porte pequeno e fácil utilização, sendo alguns dos motivos para a escolha desta placa. A possibilidade de poder conectar mídias de vídeo também é fator motivador para a escolha, levando em conta que este trabalho desenvolve uma aplicação com imagens.

A placa possui conexão *Ethernet*, *wireless*, *Bluetooth* e via USB. Um processador de 64-bit quad-core de 1.4GHz e uma memória SDRAM de 1GB. Também possui disponibilidade de acesso via HDMI para saída de vídeo, além de suportar um micro SD para a transferência de arquivo e carregamento do sistema operacional. As características da placa são suficientes para a implementação de sistemas que realizam a captura e classificação de imagens se forem sistemas planejados para atuarem nessas condições.

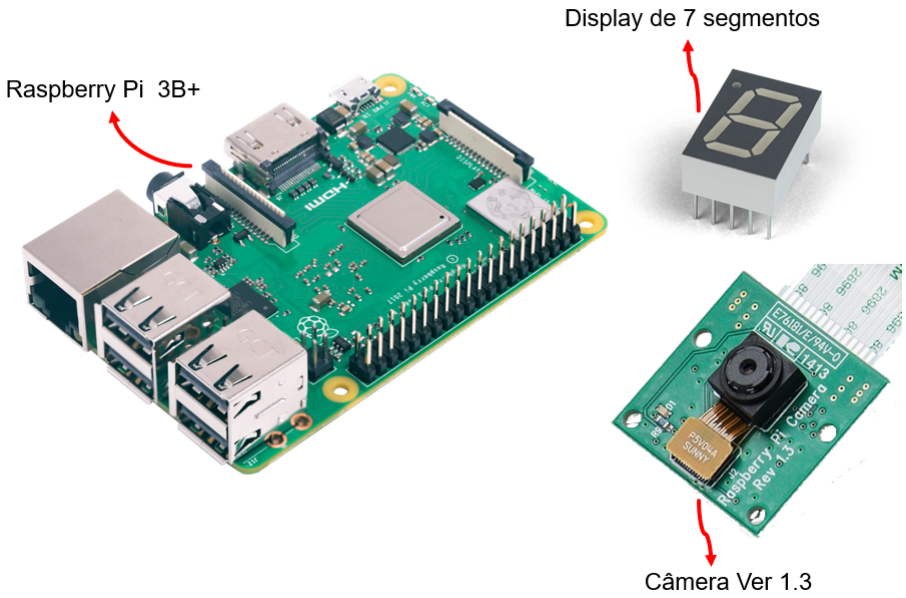

Figura 23 – Componentes de *hardware* utilizados para compor o sistema embarcado. Sendo a placa Raspberry Pi 3B+ responsável por executar os algoritmos desenvolvidos, a câmera Rev 1.3 responsável por realizar a aquisição das imagens para formar o banco de imagens de testes e para capturar as imagens a serem classificadas durante o uso do sistema embarcado final. E o display de 7 segmentos é responsável por exibir os resultados da classificação. Fonte: Imagens da internet.

4.5.2 Configuração do Rasbperry Pi 3B+

A configuração do SBC para compor o sistema embarcado ocorre em duas etapas. A primeira é a configuração inicial em que é instalado um Sistema Operacional (SO) chamado Raspberry Pi OS[1], um SO baseado no Debian e criado especialmente para compor sistemas embarcados que possuam uma placa da família Raspberry. A instalação da mídia do SO é feita através do cartão micro SD que faz parte da placa e é acoplado nela. Após a instalação, é possível realizar a configuração do ambiente para realizar a captura das imagens que compõem a base de dados.

É utilizada a câmera Raspbery Rev 1.3 para compor o *hardware*, a câmera possui 5MP de resolução. Ela está disposta em um suporte de forma que ela consiga capturar todas as imagens com a mesma distância, mesma iluminação possível, ou seja, em um ambiente controlado para que exista a menor quantidade possível de diferenças de uma imagem para outra (Figura 24). Utilizando a biblioteca *PiCamera* é possível desenvolver uma função que realiza a captura de imagens. Os algoritmos que estão na placa são desenvolvidos em Python.

Um suporte com luz de LED é utilizado para iluminar o ambiente em que as imagens são capturadas, de forma a auxiliar no controle de iluminação e fazer com que

[1] https://www.raspberrypi.com/software/

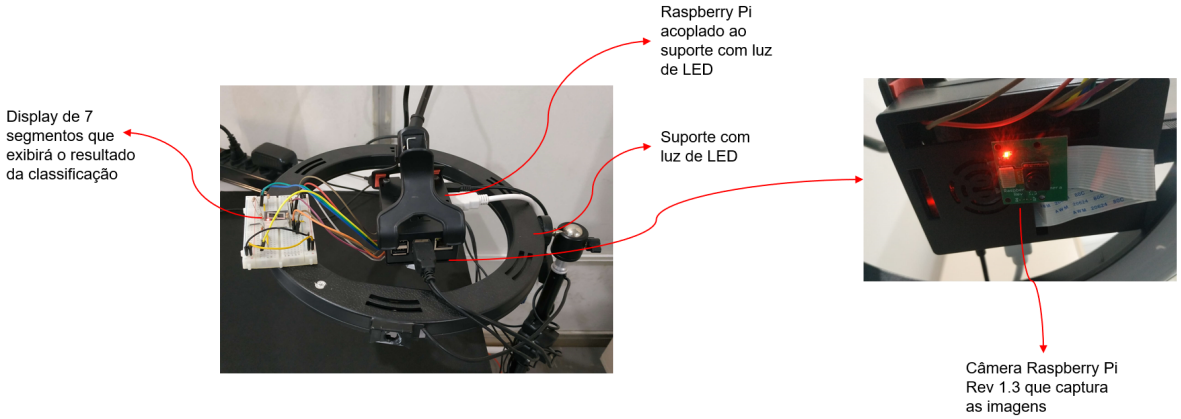

Figura 24 – Dispositivo para a identificação, reconhecimento e classificação de imagens. O dispositivo também é utilizado para a captura de exemplos a serem utilizados na fase de teste. É usado um suporte com luz para iluminar as imagens a serem capturadas. O sistema possui um *display* de 7 segmentos para exibir o resultado do classificador. Fonte: Elaborada pelo autor.

as imagens possuam pouca ou nenhuma diferença de intensidade entre elas. O fundo do ambiente de captura é na cor preta para reforçar o controle do ambiente. Para garantir que exista o mínimo de diferenças entre as imagens capturadas, foi criado um surpote fixo para que a câmera sempre tivesse a mesma distância em todas as vezes que fosse utilizada.

As imagens capturadas são armazenadas no micro SD que está acoplado à placa, sendo possível exportá-las para outros dispositivos de mídia posteriormente. As imagens em seu formato bruto são armazenadas no micro SD para que o pré-processamento e classificação possa ocorrer. A parte final do sistema embarcado requer que as imagens sejam pré-processadas após a captura, para serem classificadas e mostradas no *display* de 7 segmentos em tempo real, porém também é possível classificar imagens previamente capturadas, como por exemplo, classificar as imagens que compõem a base de dados própria e que são utilizadas para os testes de viabilidade e eficiência do sistema.

4.5.3 Aquisição das imagens para compor o banco de imagens

Para a realização dos testes e validação do sistema embarcado, foi desenvolvido um banco de imagens próprio utilizando o aparato. Para que as imagens com dígitos manuscritos fossem capturadas corretamente, de forma a ter o mínimo de diferença entre elas, foi preciso utilizar uma caneta preta para escrever um número de 0 a 9 no centro de um pedaço de papel branco de no máximo 10 centímetros de altura e 10 centímetros de largura. Em seguida foi necessário que o papel fosse posicionado abaixo da câmera, de forma que ele ficasse centralizado abaixo do aparato. Tais condições foram utilizadas para capturar todas as imagens que compõem o banco de imagens. A Figura 25 demonstra um exemplo de imagem que possui um dígito manuscrito sendo capturada pelo sistema de aquisição de imagens. É possível notar que o papel está centralizado de forma que a

câmera o capture corretamente como foi descrito acima.

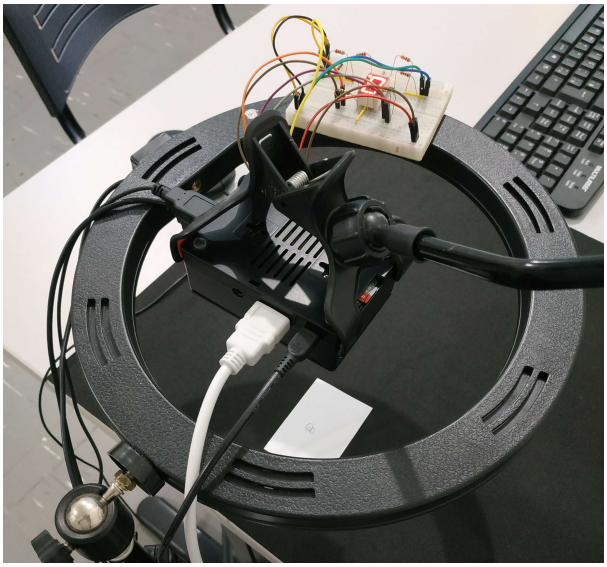

Figura 25 – Aparato para a aquisição de imagens. Fonte: Elaborada pelo autor.

As imagens que compõem o banco de imagens próprio estão no formato de cores RGB, ou seja, imagens coloridas, e foram capturadas através da câmera Raspberry Pi Rev 1.3 e possuem uma resolução de 1944 pixels de altura e 1944 pixels de largura.

A resolução das imagens é diminuída para 28x28 (altura x largura) através do pré-processamento de imagens para se adequar à entrada da CNN utilizada no sistema embarcado. As imagens da base de dados própria também tem sua escala de cor alterada para a escala de cinza, considerando que as imagens de treinamento estão nessa escala. As alterações na resolução das imagens do banco de imagens e na escala de cor são realizadas de modo que as imagens não percam as suas características principais.

Tabela 3 – Quantidade de imagens por classe da base de dados produzida para a testagem da CNN selecionada para ser utilizada no protótipo do projeto. Fonte: Elaborada pelo autor.

Classes	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**TOTAL**
Quantidade de imagens	14	17	17	11	15	22	18	17	17	18	166

A base de dados desenvolvida é formada por 166 imagens, divididas em 10 classes que representam os números de 0 a 9. A Tabela 3 apresenta a quantidade de imagens por classes presentes na base de dados.

4.5.4 Configuração do SBC para o reconhecimento, identificação e classificação de imagens

Para poder classificar as imagens capturadas, é necessário preparar o ambiente da placa para receber o modelo treinado. Como discutido anteriormente, o modelo gerado no

Google Colab é convertido para uma versão que melhor se adeque ao ambiente de SBCs, como é o caso do Raspberry Pi. Então, utiliza-se as bibliotecas do *.tflite* que realizam a conversão de um modelo no formato *pb* para o formato *.tflite*, como mostrado na Figura 26. É importante mencionar que o *tflite* só aceita conversões de modelos no formato *pb*.

```
#converte o modelo para o formato .tflite
converter = tf.lite.TFLiteConverter.from_saved_model(path)

#converter.experimental_new_converter = True
tflite_model = converter.convert()

#salva o modelo em memória.
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

Figura 26 – Algoritmo para a conversão de um modelo de Rede Neural em formato .pb salvo em disco para o formato .tflite. A conversão utiliza funções da biblioteca Tflite e salva o novo modelo no disco. Fonte: Elaborada pelo autor.

Para utilizar o modelo convertido é preciso instalar as bibliotecas de visão computacional utilizadas anteriormente como o Tensorflow, numpy, OpenCV e etc. na placa. Dessa forma, é possível que o ambiente da placa implemente o modelo treinado da rede.

Os algoritmos de captura de imagens, pré-processamento e classificação estão implementados na placa, ou seja, não é necessário a placa ter acesso a mídias externas para buscar os algoritmos utilizados para a classificação das imagens.

Para um usuário utilizar o sistema embarcado afim de classificar um dígito manuscrito ele deve executar o *script* em Python chamado *Classificador.py* que se encontra armazenado na placa. Conforme demonstrado na Figura 27, a execução desse *script* gera uma tela com a opção "tirar foto" e, ao selecionar essa opção, o algoritmo é capaz de capturar a imagem que está posicionada em frente a câmera do sistema embarcado. Após a captura é realizado o pré-processamento da imagem em tempo real. O resultado da classificação é exibido no *display* de 7 segmentos.

A Figura 28 apresenta o fluxograma para a classificação do sistema embarcado. Primeiramente, a imagem a ser classificada é capturada pela câmera do dispositivo. Então, ela é pré-processada de forma a se assemelhar às imagens utilizadas durante o treinamento para uma melhor acurácia da rede. Após o pré-processamento, a imagem é classificada dentre as classes disponíveis que correspondem dos dígitos de 0 a 9 e esse resultado é mostrado no *display* de 7 segmentos. Todas essas etapas são realizadas em tempo real, porém, também foram realizados experimentos de forma análoga à classificação do ambiente do Google Colab, ou seja, a base de dados desenvolvida e armazenada em disco também foi totalmente classificada com o sistema embarcado.

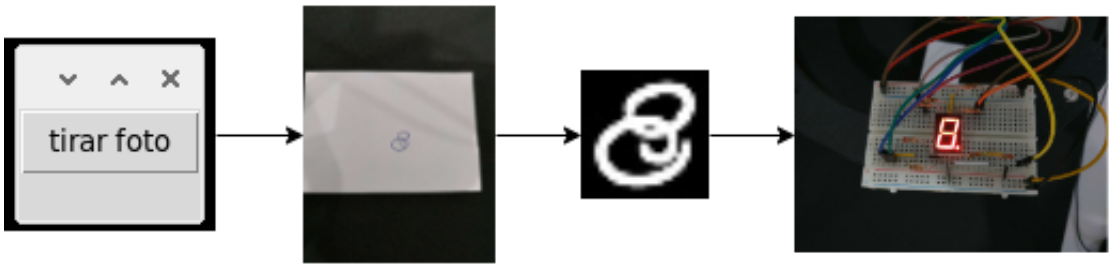

Figura 27 – Classificação utilizando o sistema embarcado. A primeira imagem mostra a execução do *script Classificador.py*, em seguida pode-se observar a imagem capturada pela câmera e a versão pré-processada dessa imagem. Por último, é mostrado o resultado do classificador. Nesse exemplo, a imagem de entrada contém o número '8' e o resultado da classificação acontece corretamente, classificando a entrada na classe que ela pertence. Fonte: Elaborada pelo autor.

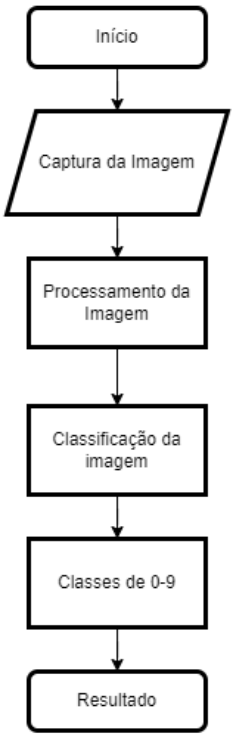

Figura 28 – Fluxo utilizado para a classificação de imagens com o sistema embarcado. Iniciando pela captura da imagem, seguida do seu processamento. Após é realizada a classificação da imagem entre as classes de 0 a 9. E por fim, é mostrado o resultado do classificador. Fonte: Elaborada pelo autor.

4.5.5 Avaliação dos experimentos

A avaliação dos experimentos é realizada utilizando uma matriz de confusão de forma a quantificar estatisticamente os resultados obtidos durante os experimentos. Uma matriz de confusão indica os erros e acertos do modelo em relação a quantidade de entradas que ele classificou corretamente que são denominados de Verdadeiros Positivos (VP), quantas foram classificadas como não pertencentes à classe e, de fato, não pertenciam

a ela são chamados de Verdadeiros Negativos (VN). Os resultados classificados de forma errada são divididos entre Falsos Negativos (FN) e Falsos Positivos (FP), onde FN são os exemplos classificados como não pertencentes à classe, porém pertencentes a ela e o FP são os exemplos classificados como pertencentes à classe quando, na verdade, não pertenciam.

A Tabela 4 representa a relação entre as classificações da entrada de um classificador binário, as dividindo entre a classe estimada (ou classe predita) e a classe correta (ou real). Para um classificador ser considerado ideal os valores de FP e FN devem ser iguais a zero, o que corresponde ao modelo classificar corretamente 100% dos exemplos de entrada.

		Estimada	
		Classe 1	Classe 2
Real	Classe 1	Verdadeiro Positivo (VP)	Verdadeiro Negativo (VN)
	Classe 2	Falso Positivo (FP)	Falso Negativo (FN)

Tabela 4 – Matriz de confusão para duas classes. A matriz de confusão faz uma relação entre os resultados do classificador. Fonte: Elaborada pelo autor.

Para os casos de três ou mais classes, as linhas e colunas da matriz de confusão são aumentadas de forma a abranger todas as classes. E a avaliação dos resultados se dá ao cruzar os dados entre linhas e colunas. No caso da classificação de dígitos para este trabalho, a matriz de confusão possui 10 linhas e 10 colunas.

Para uma maior exploração dos resultados obtidos, são consideradas outras métricas avaliativas que medem o desempenho geral e por classe do classificador. A acurácia é um indicativo geral do desempenho do modelo e é amplamente utilizada para avaliar classificadores. Tal métrica considera a quantidade total de classificações corretas que foram obtidas. Utilizando os valores de VP, VN, FP e FN é possível obter a acurácia de qualquer classificador através da Equação 4.1. Apesar de o Keras disponibilizar a acurácia do modelo e avaliá-lo através da função *model.evaluate()*, tais resultados não são considerados suficientes para avaliar o modelo, sendo necessário utilizar avaliações mais precisas para este caso.

$$A = \frac{VP + VN}{VP + VN + FP + FN} \tag{4.1}$$

Também é utilizada a métrica chamada precisão que avalia dentre as classificações que o modelo considerou como verdadeiras e verifica quantas dessas classificações estão de fato corretas. A Precisão pode ser obtida através da Equação 4.2, sendo ela a quantidade de VP divida pela soma de VP e FP. A sensibilidade (*recall*) do sistema também é uma métrica utilizada neste trabalho e diz respeito às classificações consideradas como VP em relação a soma de VP e FN, a sensibilidade é obtida pela Equação 4.3. Outra métrica utilizada é o f1-*score* que é a média harmônica entre a precisão e sensibilidade do classificador e pode ser calculada através da Equação 4.4.

$$P = \frac{VP}{VP + FP} \tag{4.2}$$

$$R = \frac{VP}{VP + FN} \tag{4.3}$$

$$F = 2 \cdot \frac{P \cdot R}{P + R} \tag{4.4}$$

5 Resultados e discussões

5.1 Treinamento da SqueezeNet

Como foi dito na Seção 4 a Squeezenet foi treinada no ambiente virutal do Google Colab. A Figura 29 indica o comportamento da acurácia durante as épocas de aprendizado da rede, que consiste em quão bem o modelo consegue classificar os dados de entrada, e do custo, que quantifica a medida do erro da rede em relação às respostas corretas que eram esperadas. A curva azul indica a acurácia (a) e o custo (b) obtidos com os dados de treinamento. A curva laranja indica a acurácia e custo com base no conjunto de validação. Ao avaliar os gráficos, é perceptível que para a maioria das épocas, a acurácia do conjunto de validação é maior do que a do conjunto de treino, o que não costuma ser comum durante o treinamento de modelos de redes neurais. Porém entende-se que, pelo fato de os dados do MNIST serem dados bem estruturados e suas imagens possuírem pouco ou nenhum ruído, pode-se concluir que a similaridade entre os dados de treino e validação influencia diretamente na curva de aprendizado e, consequentemente, o modelo se adapta aos seus dados.

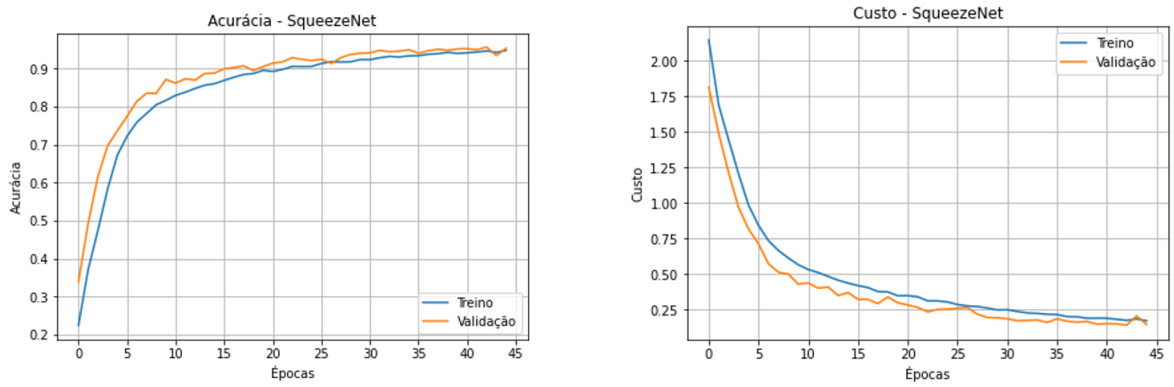

Figura 29 – Curva da acurácia e da perda da Rede Neural Convolucional *SqueezeNet* durante as 45 épocas de treinamento. As épocas correspondem as execuções do algoritmo BP através dos dados de treino. A acurácia (a) indica o desempenho do modelo. E o custo (b) indica o quanto o modelo é capaz de classificar erroneamente os dados. Fonte: Elaborada pelo autor.

O modelo obtém mais de 90% de acurácia a partir da época 23 e continua crescendo até o final do treinamento, finalizando a última época com 94,8% de acurácia para o conjunto de treinamento e 95,34% para o conjunto de validação. Os piores resultados de acurácia se mantém nas primeiras épocas sendo que a pior para o conjunto de treinamento e para o conjunto de validação se encontram na primeira época (22,36% e 33,8% respectivamente) o que costuma ser comum. Existem variações em relação a acurácia que se expressam durante todo o treinamento, entretanto, considerando que a validação do modelo se mantém

acima do esperado, considera-se que o treinamento da rede foi realizado de forma eficaz e trouxeram resultados satisfatórios para o total de 45 épocas em que foi configurado.

Avaliando gráfico de custo, nota-se que ele diminui conforme as épocas vão aumentando e, de forma semelhante a acurácia, o custo do conjunto de validação se mantém menor do que o do conjunto de treino para a maioria das épocas, o que era esperado considerando-se a curva de acurácia do modelo.

Neste trabalho é utilizado o método *model.evaluate()* para a realização de avaliação preliminar do modelo de treinamento. Essa função recebe como parâmetros os dados de testes e seus respectivos *labels* e calcula a acurácia do modelo em teste, realizando a avaliação em lotes. Para este modelo é obtida uma acurácia de 95,2%.

5.1.1 Pré-processamento das imagens de entrada

O pré-processamento das imagens é utilizado para eliminar ou minimizar quaisquer aspectos que sejam considerados desnecessários encontrados nas imagens e que possam prejudicar a classificação das imagens de entrada do sistema embarcado, ou seja, o intuito do pré-processamento de imagens é eliminar ou minimizar o ruído que elas possam possuir.

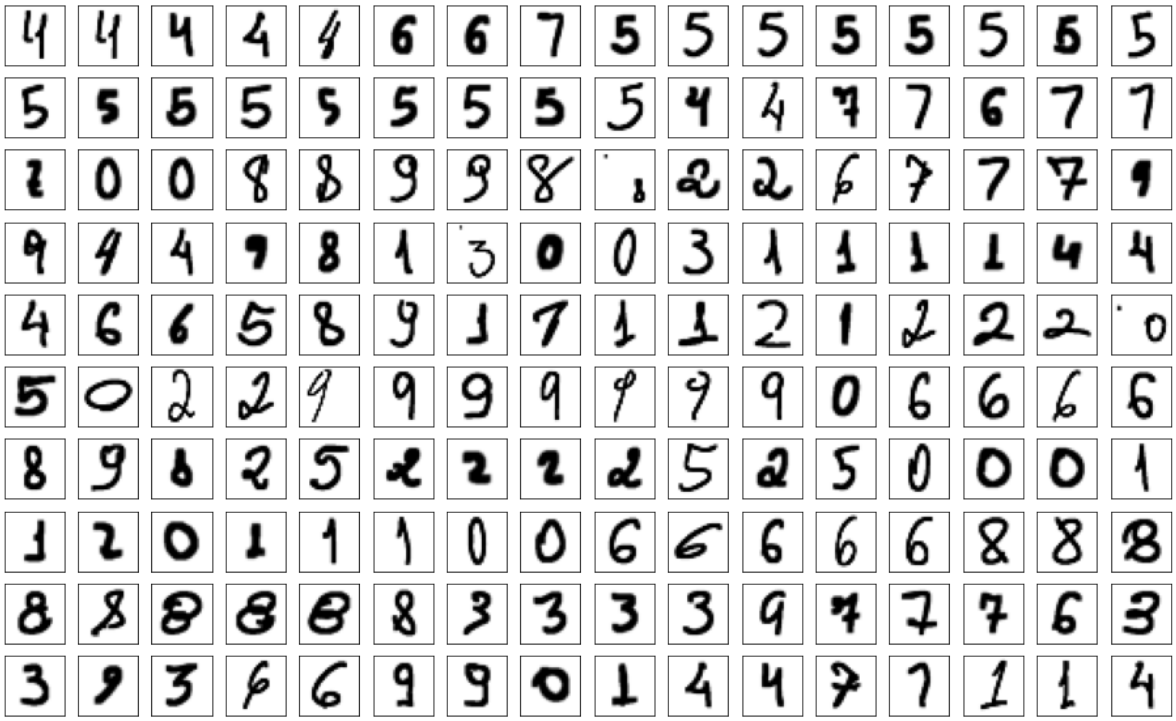

Figura 30 – 160 imagens da base de dados desenvolvida para realizar os testes do sistema embarcado. As imagens foram pré-processadas utilizando o algoritmo desenvolvido neste projeto. Fonte: Elaborada pelo autor.

Duante o desenvolvimento do projeto são aplicadas duas técnicas de remoção de ruído das imagens: o Filtro Gaussiano e a Transformada Discreta de Fourier. Dentre as duas técnicas, o Filtro Gaussiano demonstrou melhores resultados para as imagens

pertencentes à base de dados desenvolvida durante os testes, se mostrando mais eficiente para a minimização do ruído. Dessa forma, utiliza-se dessa técnica durante o processo de pré-processamento das imagens.

A Figura 30 é uma representação das 160 primeiras imagens da base de dados desenvolvida após o pré-processamento.

Durante o processo de pré-processamento das imagens, nota-se que o ruído não é totalmente eliminado em algumas imagens. Tais imagens possuem "borrões" que os papéis utilizados possuíam antes de serem utilizados e que se assemelham à riscos dos objetos utilizados para a escrita ou foram inseridos erroneamente pelo usuário durante a escrita do dígito. A Figura 31 representa um exemplo desse erro no pré-processamento. Na imagem, o dígito 8 foi escrito por um usuário e havia um pequeno risco acima da folha de papel, mesmo ele sendo quase imperceptível quando observa-se a foto tirada pelo dispositivo. Para este caso, o algoritmo de pré-processamento não conseguiu perceber a diferença entre a informação desnecessária e o número manuscrito, considerando o ruído parte do dígito e não eliminando-o.

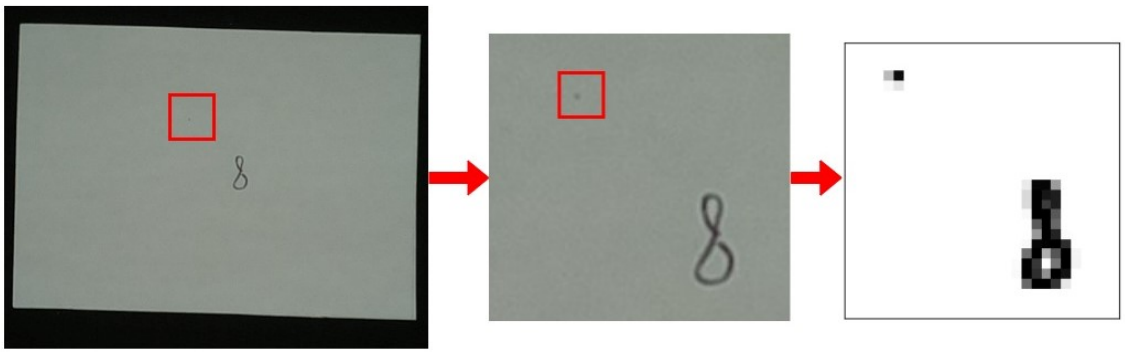

Figura 31 – Erro durante o pré-processamento de um exemplo do dígito 8 manuscrito por um usuário. O algoritmo não percebe a diferença entre o ruído da imagem (destacado pelo quadrado vermelho) e considera-o como parte do dígito. Ao final tem-se uma imagem distorcida.

Entretanto, para a maior parte dos resultados (cerca de 98,2% do total de 166 imagens) os resultados foram satisfatórios, ou seja, eliminam e suavizam os ruídos da maioria das imagens de entrada.

5.1.2 Classificador

Para realizar os testes com o sistema embarcado ao classificar imagens utiliza-se a base de dados desenvolvida conforme descrito na Seção 4.2. Os exemplos estão armazenados na placa após terem sido capturados pela câmera do sistema embarcado, de modo que não há necessidade do sistema buscar os exemplos em outros dispositivos. Os resultados da classificação são avaliados conforme descrito no Capítulo 4. Todos os 166 exemplos

do banco de imagens desenvolvido são utilizados para o teste do sistema e a Tabela 5 apresenta os resultados obtidos ao avaliar o classificador.

Tabela 5 – Resultado da avaliação do Classificador obtidos através das métricas definidas anteriormente. Fonte: Elaborada pelo autor.

Métrica	**Resultado(%)**
Acurácia	76%
Precisão	80%
Sensibilidade	76%
f1-score	75%

As métricas utilizadas mostram que o sistema obteve uma acurácia inferior ao medido utilizando a função *model.evaluate()* do Keras. Era esperado que o resultado das avaliações fossem inferiores à obtida no Google Colab, visto que as imagens utilizadas para os testes do sistema são imagens não pertencentes ao conjunto de treino, diferentemente do testado com o *model.evaluate()* em que são as 10000 imagens do MNIST para avaliar o modelo. Entretanto, existe a possibilidade de o classificador utilizado no sistema embarcado ter sofrido com o *overfitting*, ou seja, ao invés da rede aprender a classificar quaisquer imagens que contém dígitos, a rede se adequou aos dados do MNIST e, consequentemente, possui desempenho inferior ao classificar dados não pertencentes ao conjunto de treinamento.

Dentre as métricas utilizadas, a Precisão foi a métrica que o classificador obteve o melhor desempenho, significando que, dentre todas as classes classificadas como positivas, 80% foram rotuladas corretamente.

A Figura 32 apresenta a matriz de confusão do classificador. A análise da matriz de confusão pode trazer uma ideia da forma que a rede se comporta ao classificar as imagens e, consequentemente, pode-se inferir sobre o desempenho da rede. As colunas da matriz de confusão representam a quantidade de imagens classificadas pela rede em cada classe e as linhas indicam a classe verdadeira (ou real). É importante ressaltar que para este caso, as classes não possuem a mesma quantidade de exemplos, por isso deve-se analisar os dados de forma a considerar os pesos de cada classe.

É possível perceber que existe uma certa dificuldade na classificação de imagens que contém o digito 6, do total de 18 imagens, somente 4 foram classificadas corretamente. Analisando as imagens de entrada (Figura 30), é compreensível a rede ter classificado erroneamente esses exemplos como 2 ou 9 devido à similaridade que esses dígitos possuem. O mesmo pode ser percebido para a classe 9, o sistema classifica erroneamente metade dos exemplos disponíveis, os classificando como a classe 3 e 7. Em contrapartida, a rede possui 100% de sucesso ao classificar as imagens da base de dados desenvolvida que continham o número 5.

As imagens classificadas no ambiente do Google Colab e no sistema embarcado

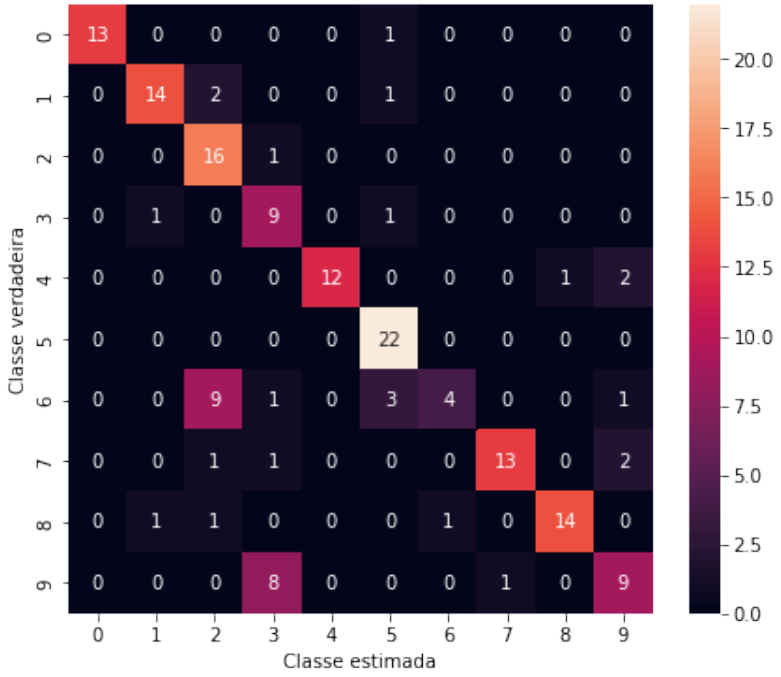

Figura 32 – Matriz de confusão gerada através dos resultados do classificador. Os valores representam as classificações, sendo as corretas representadas pela interseção da classe verdadeira e classe estimada de mesmo *label*. Fonte: Elaborada pelo autor.

possuem resultados idênticos, o que é esperado visto que, apesar do ambiente de aplicação ser diferente, o modelo da rede é igual. O que constata que não há perda de dados ou de qualidade do modelo ao convertê-lo para diversos formatos.

Entretanto, para o objetivo do projeto proposto, os resultados são razoáveis, porém abaixo de exemplos de classificação de caracteres encontrados na literatura. O algoritmo desenvolvido para pré-processar as imagens mostrou-se eficiente ao minimizar os ruídos encontrados nas imagens, porém em exemplos que possuíam ruídos muito expressivos o algoritmo não conseguiu eliminar a maior parte e, consequentemente, isso afetou o desempenho do classificador em alguns momentos da classificação. Os resultados obtidos com o sistema embarcado em tempo real se mostram eficientes, não sendo detectados erros de memória durante o processamento das imagens tampouco erros relacionados ao *software* do sistema.

6 Conclusão

Neste projeto de pesquisa foram abordados e explorados temas relacionados no aprendizado de máquina e ao processamento digital de sinais. Principalmente ao que diz respeito a redes neurais convolucionais, processamento de imagens digitais e classificação de imagens. Foi proposto o desenvolvimento de um dispositivo embarcado que realizasse a simples tarefa de classificar dígitos manuscritos, utilizando conhecimentos de visão computacional para realizá-lo. A classificação de caracteres, apesar de ser simples, auxilia àqueles que desejam compreender as arquiteturas de redes neurais que realizam esse objetivo.

De forma a aprofundar os conhecimentos em redes neurais convolucionais, foram utilizados algoritmos implementados por terceiros, alterando minimamente a arquitetura desses algoritmos para se adequarem às necessidades do projeto. Também foram abordados os fundamentos do processamento digital de imagens, com ênfase na redução e suavização de ruídos utilizando filtros espaciais, detecção de bordas e limiarização. Para essas técnicas serem utilizadas foi desenvolvido um algoritmo para realizar tais processos e os resultados obtidos foram satisfatórios para o objetivo proposto.

O sistema embarcado proposto obteve acurácia de 76%. Esperava-se que o classificador fosse mais eficiente, levando em conta os resultados obtidos durante o treinamento e comparando com exemplos dispostos na literatura para o mesmo problema. Entretanto, para o cenário aqui apresentado, os resultados são relevantes e demonstram que algoritmos robustos como CNNs podem ser utilizados em sistemas com poucos recursos computacionais.

Os testes realizados ainda englobam uma quantidade pequena de exemplos e vale ressaltar que esse cenário não pode ser generalizado para todos os sistemas embarcados com o mesmo propósito. Para este cenário em específico o dispositivo possuiu esse comportamento e o mesmo não pode ser dito caso as variáveis sejam alteradas.

Espera-se que os resultados obtidos neste trabalho sirvam como exemplos para projetos futuros de sistemas embarcados que envolvam redes neurais convolucionais e das diversas aplicações que envolvam o processamento digital de imagens.

Para trabalhos futuros, pretende-se avaliar a viabilidade de alterar as características da rede neural convolucional utilizada e realizar novos testes com o algoritmo de pré-processamento de imagens apresentado para um conjunto de imagens com mais exemplos. Além de utilizar novas estratégias para o pré-processamento de imagens, de forma a melhorar o desempenho deste algoritmo. Também é considerado utilizar técnicas como o *Transfer Learning* para analisar a influência dessa técnica no desempenho final da rede.

Com o intuito de aprofundar os conhecimentos em aprendizado de máquina, pretende-se utilizar o sistema embarcado aqui apresentado para outros conjuntos de dados como por exemplo, para a classificação de caracteres alfanuméricos, de objetos ou até mesmo para a classificação de faces, e avaliar o desempenho do sistema embarcado para tais cenários.

Referências

AJIT, A.; ACHARYA, K.; SAMANTA, A. A review of convolutional neural networks. In: *2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)*. Vellore, India: IEEE, 2020. p. 1–5. Citado 2 vezes nas páginas 33 e 35.

ALI, A. S.; DUMAN, B.; BETÜL Şen. Benchmark Analysis of Jetson TX2, Jetson Nano and Raspberry PI using Deep-CNN. In: *2020 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*. Ankara, Turkey: IEEE, 2020. p. 1–5. Citado na página 20.

ALMEIDA, C. C. de. *Identificação e Classificação de Imagens usando Rede Neural Convolucional e Machine Learning: Implementação em Sistema Embarcado*. Tese (Doutorado) — Universidade Estadual de Campinas, Campinas, 2019. Citado 3 vezes nas páginas 33, 37 e 40.

ASHIQUZZAMAN, A.; TUSHAR, A. K. Handwritten arabic numeral recognition using deep learning neural networks. In: *2017 IEEE International Conference on Imaging, Vision Pattern Recognition (icIVPR)*. Dhaka, Bangladesh: IEEE, 2017. p. 1–4. Citado na página 18.

BENGIO, Y. Learning deep architectures for ai. *Foundations*, v. 2, p. 1–55, 01 2009. Citado na página 33.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. 1. ed. [S.l.]: Springer New York, NY, 2006. ISBN 0-201-39829-X. Citado 2 vezes nas páginas 31 e 36.

BOSCHI, A. et al. A Cost-Effective Person-Following System for Assistive Unmanned Vehicles with Deep Learning at the Edge. *Machines*, v. 8, n. 3, p. 49, 2020. Citado 2 vezes nas páginas 25 e 27.

CALABRESE, B. et al. Solar-Powered Deep Learning-Based Recognition System of Daily Used Objects and Human Faces for Assistance of the Visually Impaired. *Energies*, v. 13, n. 22, p. 6104, 2020. Citado 3 vezes nas páginas 18, 25 e 27.

CHO, C.; CHOI, W.; KIM, T. Leveraging Uncertainties in Softmax Decision-Making Models for Low-Power IoT Devices. *Sensors*, v. 20, n. 16, p. 4603, 2020. Citado na página 26.

CURTIN, B. H.; MATTHEWS, S. J. Deep Learning for Inexpensive Image Classification of Wildlife on the Raspberry Pi. In: *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*. New York, NY, USA: IEEE, 2019. p. 0082–0087. Citado 3 vezes nas páginas 24, 26 e 27.

DAYAL, A. et al. Design and Implementation of Deep Learning Based Contactless Authentication System Using Hand Gestures. *Electronics*, v. 10, n. 2, p. 182, 2021. Citado 2 vezes nas páginas 25 e 27.

DENIZ, O. et al. Eyes of Things. *Sensors*, v. 17, n. 5, p. 1173, 2017. Citado 2 vezes nas páginas 17 e 20.

GE, D.-Y. et al. Design of high accuracy detector for mnist handwritten digit recognition based on convolutional neural network. In: . Xiangtan, China: IEEE, 2019. p. 658–662. Citado na página 17.

GONZALEZ, R. C.; WOODS, R. C. *Machine Learning*: Processamento digital de imagens. 3. ed. São Paulo: Pearson Prentice Hall, 2010. Citado 5 vezes nas páginas 17, 37, 38, 39 e 46.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, Massachusetts: MIT Press, 2016. <http://www.deeplearningbook.org>. Citado na página 32.

HAYKIN, S. *Redes Neurais: Princípios e Prática*. Bookman Editora, 2001. ISBN 9788577800865. Disponível em: <https://books.google.com.br/books?id=bhMwDwAAQBAJ>. Citado 6 vezes nas páginas 10, 28, 29, 30, 31 e 32.

HETLAND, M. L. *Beginning Python: From Novice to Professional*. 1. ed. Berkeley, CA: Apress, 2005. Citado na página 40.

HOSSAIN, S.; LEE, D. jin. Deep Learning-Based Real-Time Multiple-Object Detection and Tracking from Aerial Imagery via a Flying Robot with GPU-Based Embedded Devices. *Sensors*, v. 19, n. 15, p. 3371, 2019. Citado 2 vezes nas páginas 24 e 27.

HSIAO, T.-Y. et al. Filter-based deep-compression with global average pooling for convolutional networks. *Journal of Systems Architecture*, Elsevier B.V., Amesterdã, Países Baixos, v. 95, p. 9–18, 2019. ISSN 1383-7621. Disponível em: <https://www.sciencedirect.com/science/article/pii/S1383762118302340>. Citado na página 35.

IANDOLA, F. N. et al. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. Disponível em: <http://arxiv.org/abs/1602.07360>. Citado 6 vezes nas páginas 11, 36, 37, 50, 51 e 52.

KHAN, T. An Intelligent Microwave Oven with Thermal Imaging and Temperature Recommendation Using Deep Learning. *Applied System Innovation*, v. 3, n. 1, p. 13, 2020. Citado na página 25.

KIM, W.; JUNG, W.-S.; CHOI, H. K. Lightweight Driver Monitoring System Based on Multi-Task Mobilenets. *Sensors*, v. 19, n. 14, p. 3200, 2019. Citado 3 vezes nas páginas 17, 25 e 27.

KITCHENHAM, B.; CHARTERS, S. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. 2007. Citado 3 vezes nas páginas 20, 21 e 22.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 60, n. 6, p. 84–90, may 2017. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/3065386>. Citado na página 50.

LECUN, Y. et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, Piscataway, Nova Jersey, EUA, v. 86, n. 11, p. 2278–2324, 1998. Citado 2 vezes nas páginas 33 e 50.

LECUN, Y. et al. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Society Magazine*, Institute of Electrical and Electronics Engineers Inc., v. 27, n. 11, p. 41–46, nov. 1989. ISSN 0163-6804. Citado na página 18.

LECUN, Y.; KAVUKCUOGLU, K.; FARABET, C. Convolutional networks and applications in vision. In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. Paris, France: IEEE, 2010. p. 253–256. Citado 2 vezes nas páginas 33 e 34.

LEROUX, S. et al. The cascading neural network: building the Internet of Smart Things. *Knowledge and Information Systems (KAIS)*, v. 54, n. 3, p. 791–814, 2017. Citado 4 vezes nas páginas 20, 25, 26 e 27.

LINDNER, T. et al. Face recognition system based on a single-board computer. In: *2020 International Conference Mechatronic Systems and Materials (MSM)*. Bialystok, Poland: IEEE, 2020. p. 1–6. Citado 3 vezes nas páginas 24, 26 e 27.

MARWEDEL, P. *Embedded System Design*. 1. ed. Netherlands: Springer, 2006. Citado na página 39.

MELLO, R. F.; PONTI, M. A. *Machine Learning*: A practical approach on the statistical learning theory. São Paulo: Springer, Cham, 2018. Citado na página 30.

MILIOTO, A.; LOTTES, P.; STACHNISS, C. Real-Time Blob-Wise Sugar Beets vs Weed Classification for Monitoring Fields Using Convolutional Neural Networks. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, IV-2/W3, n. 1, p. 41–48, 2017. Citado 3 vezes nas páginas 18, 25 e 27.

NIKODEM, M. et al. Multi-Camera Vehicle Tracking Using Edge Computing and Low-Power Communication. *Sensors*, v. 20, n. 11, p. 3334, 2020. Citado 4 vezes nas páginas 18, 25, 26 e 27.

OLIVEIRA, D. C. D.; WEHRMEISTER, M. A. Using Deep Learning and Low-Cost RGB and Thermal Cameras to Detect Pedestrians in Aerial Images Captured by Multirotor UAV. *Sensors*, v. 18, n. 7, p. 2244, 2018. Citado 3 vezes nas páginas 24, 26 e 27.

OLIVEIRA, F. M. C. de; BORIN, E. Partitioning Convolutional Neural Networks to Maximize the Inference Rate on Constrained IoT Devices. *Future Internet*, v. 11, n. 10, p. 209, 2019. Citado na página 25.

OSÓRIO, F.; BITTENCOURT, J. R. Sistemas inteligentes baseados em redes neurais artificiais aplicados ao processamento de imagens. In: *I WORKSHOP DE INTELIGÊNCIA ARTIFICIAL UNISC*. Santa Cruz do Sul: UNISC, 2000. Citado 4 vezes nas páginas 17, 28, 32 e 33.

PEINE, A. et al. A Deep Learning Approach for Managing Medical Consumable Materials in Intensive Care Units via Convolutional Neural Networks: Technical Proof-of-Concept Study. *JMIR Med Inform*, v. 7, n. 4, p. e14806, 2019. Citado na página 25.

PONTI, M. A.; COSTA, G. B. P. d. Como funciona o deep learning. In: *Tópicos em Gerenciamento de Dados e Informações: Minicursos do SBBD 2017*. 1. ed. Uberlândia - Minas Gerais: SBC, 2017. v. 1, cap. 3, p. 63–93. Citado 6 vezes nas páginas 17, 31, 32, 33, 34 e 36.

PONTI, M. A. et al. Everything you wanted to know about deep learning for computer vision but were afraid to ask. In: *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*. Niteroi, Brazil: IEEE, 2017. p. 17–41. Citado 3 vezes nas páginas 10, 31 e 34.

PRODANOV, C. C.; FREITAS, E. C. d. *Metodologia do trabalho científico*: métodos e técnicas de pesquisa e do trabalho acadêmico. 2. ed. Novo Hamburgo: Editora Feevale, 2013. Citado na página 43.

ROSENBLATT, F. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, v. 65 6, p. 386–408, 1958. Citado na página 29.

SERMANET, P.; CHINTALA, S.; LECUN, Y. Convolutional neural networks applied to house numbers digit classification. *CoRR*, abs/1204.3968, 2012. Disponível em: <http://arxiv.org/abs/1204.3968>. Citado na página 18.

SHIDDIEQY, H. A.; HARIADI, F. I.; ADIONO, T. Implementation of deep-learning based image classification on single board computer. In: *2017 International Symposium on Electronics and Smart Devices (ISESD)*. Yogyakarta, Indonesia: IEEE, 2017. p. 133–137. Citado na página 24.

SORDILLO, S. et al. Customizable Vector Acceleration in Extreme-Edge Computing: A RISC-V Software/Hardware Architecture Study on VGG-16 Implementation. *Electronics*, v. 10, n. 4, p. 518, 2021. Citado 2 vezes nas páginas 25 e 27.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, JMLR, Inc., US, v. 15, n. 56, p. 1929–1958, 2014. Disponível em: <http://jmlr.org/papers/v15/srivastava14a.html>. Citado 2 vezes nas páginas 33 e 36.

TABIK, S. et al. A snapshot of image pre-processing for convolutional neural networks: Case study of mnist. *International Journal of Computational Intelligence Systems*, v. 10, p. 555, 01 2017. Citado 2 vezes nas páginas 31 e 33.

VOULODIMOS, A. et al. Deep learning for computer vision: A brief review. *Computational Intelligence and Neuroscience*, v. 2018, p. 1–13, 02 2018. Citado na página 31.

WEI, B. et al. A Deep-Learning-Driven Light-Weight Phishing Detection Sensor. *Sensors*, v. 19, n. 19, p. 4258, 2019. Citado na página 17.