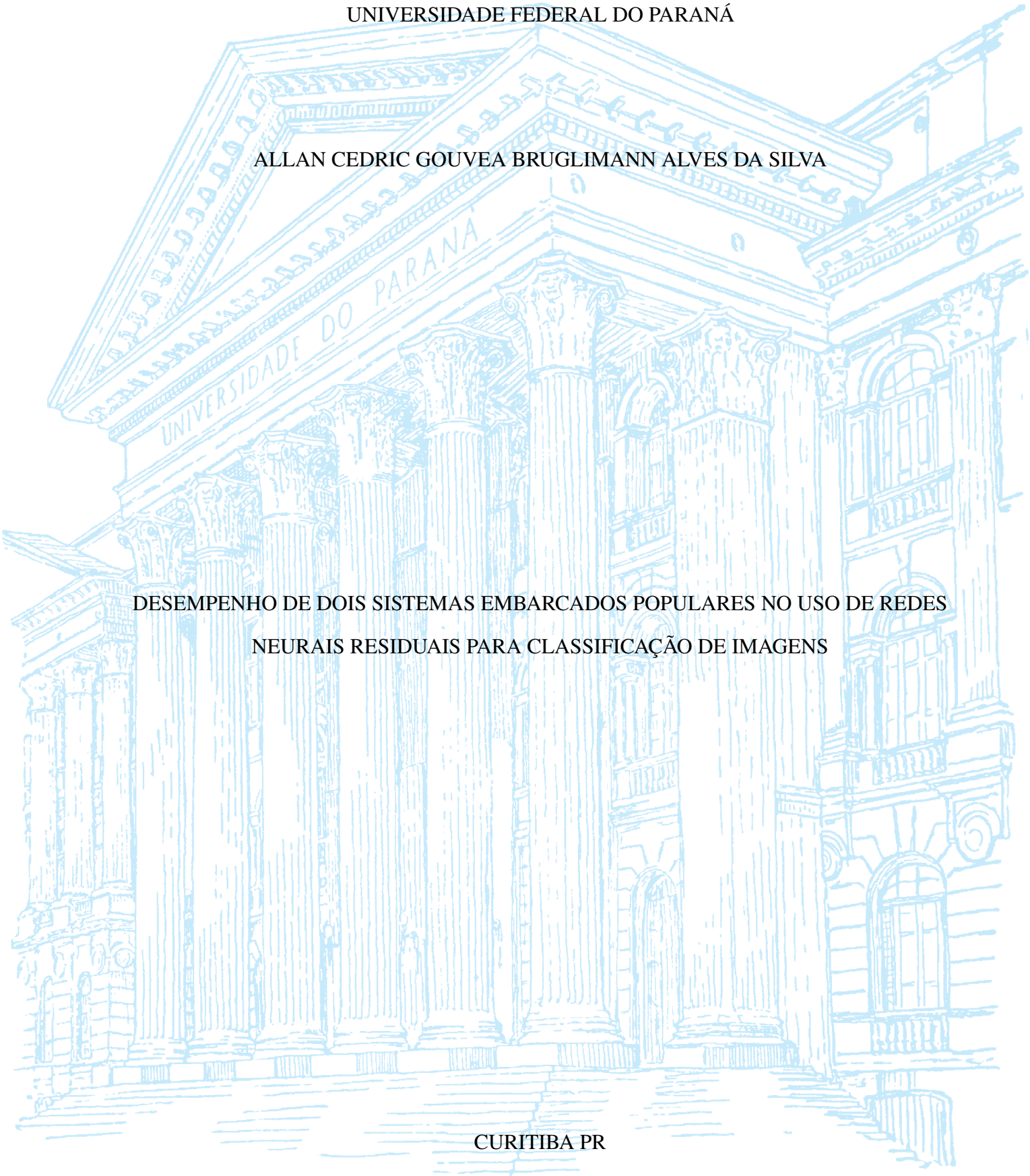
UNIVERSIDADE FEDERAL DO PARANÁ

ALLAN CEDRIC GOUVEA BRUGLIMANN ALVES DA SILVA



DESEMPENHO DE DOIS SISTEMAS EMBARCADOS POPULARES NO USO DE REDES NEURAIS RESIDUAIS PARA CLASSIFICAÇÃO DE IMAGENS

CURITIBA PR

2024

ALLAN CEDRIC GOUVEA BRUGLIMANN ALVES DA SILVA

DESEMPENHO DE DOIS SISTEMAS EMBARCADOS POPULARES NO USO DE REDES NEURAIS RESIDUAIS PARA CLASSIFICAÇÃO DE IMAGENS

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Eduardo Todt.

CURITIBA PR

2024

RESUMO

Atualmente, muitas redes neurais dependem geralmente de extenso poder computacional para serem treinadas e executadas, sendo normalmente confinadas em grandes servidores. Essa forma tradicional de lidar com redes neurais pode ter impactos significativos relacionados a latência de execução, privacidade de dados, consumo de energia e custo financeiro. Por essa razão, a área de *TinyML* busca levar a execução de redes neurais para dentro de sistemas embarcados que, de forma geral, têm menor custo financeiro e consomem menos energia. Entretanto, existem diversos desafios para embarcar redes neurais em sistemas com recursos limitados, sendo geralmente necessário aplicar técnicas de compressão de redes neurais para diminuir o tamanho dos modelos. A partir disso, este trabalho tem como objetivo avaliar o desempenho e eficiência energética de dois sistemas embarcados populares, ESP32 e Raspberry Pi 3B+, na tarefa de classificar imagens do *dataset* CIFAR-10 utilizando Redes Neurais Residuais (ResNets). Neste trabalho, a plataforma Raspberry Pi apresentou o melhor resultado de acurácia (ResNet-20: 0,8659), o melhor resultado de tempo de inferência (ResNet-8: 0,009 s), e o melhor resultado de consumo de energia por inferência (ResNet-8: 0,024 J). Enquanto o ESP32 teve o melhor resultado de potência gasta (ResNet-8: 0,38 W). O código deste trabalho e os resultados estão disponíveis em: `https://github.com/allan-cedric/tcc-resnet-cifar10`.

Palavras-chave: Visão Computacional. Redes Neurais Residuais. TinyML. Classificação de Imagens. Sistemas Embarcados. ESP32. Raspberry Pi.

ABSTRACT

Today, many neural networks generally rely on extensive computational power for training and execution, typically being confined to large servers. This traditional approach to handling neural networks can have significant impacts related to execution latency, data privacy, energy consumption, and financial costs. For this reason, the field of TinyML aims to bring the execution of neural networks to embedded systems, which are generally more cost-effective and energy-efficient. However, there are several challenges to deploying neural networks on resource-constrained systems, often requiring the application of neural network compression techniques to reduce model size. With this in mind, this work aims to evaluate the performance and energy efficiency of two popular embedded systems, ESP32 and Raspberry Pi 3B+, in the task of classifying images from the CIFAR-10 dataset using Residual Neural Networks (ResNets). In this work, the Raspberry Pi platform achieved the best accuracy result (ResNet-20: 0,8659), the best inference time result (ResNet-8: 0,009 s), and the best energy consumption per inference result (ResNet-8: 0,024 J). Meanwhile, the ESP32 achieved the best power consumption result (ResNet-8: 0,38 W). The code for this work and the results are available at: `https://github.com/allan-cedric/tcc-resnet-cifar10`.

Keywords: Computer Vision. Residual Neural Networks. TinyML. Image Classification. Embedded Systems. ESP32. Raspberry Pi.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

DINF	Departamento de Informática
UFPR	Universidade Federal do Paraná
IA	Inteligência Artificial
ML	*Machine Learning*
CNN	*Convolutional Neural Network*
YOLO	*You Only Look Once*
KiB	*Kibibyte*
MiB	*Mebibyte*
KB	*Kilobyte*
MB	*Megabyte*
ILSVRC	*ImageNet Large Scale Visual Recognition Challenge*
SBC	*Single Board Computer*
RAM	*Random Access Memory*
SRAM	*Static Random Access Memory*
SO	Sistema Operacional
SD	*Secure Digital*
GPU	*Graphics Processing Unit*
SIL	*Safety Integrity Level*
MHz	*Megahertz*
GHz	*Gigahertz*
ROM	*Read-Only Memory*

LISTA DE SÍMBOLOS

n	Número de blocos residuais de uma rede neural residual
x	Entrada de dados de um bloco residual em uma ResNet

SUMÁRIO

1 INTRODUÇÃO

Nos últimos anos, as redes neurais convolucionais profundas se tornaram extremamente populares para problemas de classificação de imagens, devido ao seu poder de extração de características. Entretanto, essas redes neurais demandam extenso poder computacional em termos de processamento e memória para serem treinadas e até mesmo executadas apropriadamente (Simonyan e Zisserman, 2015; Krizhevsky et al., 2012; He et al., 2015).

Por essa razão, foram desenvolvidos métodos de otimização como *pruning* e *quantization* (Han et al., 2016; Novac et al., 2021; Molchanov et al., 2017), a fim de reduzir o tamanho desses modelos neurais, sobretudo no que diz respeito a memória ocupada, tentando ainda garantir a mesma acurácia do modelo original (Iandola et al., 2016). A título de exemplo, temos a bem conhecida MobileNet que é uma arquitetura de rede neural convolucional eficiente para a classe de computadores de dispositivos móveis (Howard et al., 2017).

A partir disso, aumentou-se o interesse em expandir o uso de redes neurais para sistemas embarcados no geral, como os microcontroladores (Banbury et al., 2021c), os quais representam uma extensa e diversa classe de computadores com grandes limites de memória e processamento, gerando assim muitos desafios para se conseguir portar modelos neurais bem eficientes e ter resultados satisfatórios (Banbury et al., 2021c,a; Fedorov et al., 2019; Han et al., 2016). A memória e o processamento de microcontroladores estão na ordem de KB e MHz, respectivamente (Banbury et al., 2021c; Novac et al., 2021). Portanto, há uma clara disparidade de desempenho e recursos em relação a um computador convencional que possui geralmente memória e processamento na ordem de GB e GHz, respectivamente.

Apesar disso, esses computadores embarcados possuem baixíssimo consumo de energia na ordem de mW (miliwatt), baixíssimo custo e são bem compactos (Fedorov et al., 2019; Banbury et al., 2021c) comparados ao custo e tamanho de computadores dotados de poderosas GPUs, e até mesmo comparado a uma classe de computador pessoal popular, como um laptop ou desktop. Além disso, com os avanços das técnicas de otimização de redes neurais, já foi mostrado a possibilidade de portar redes neurais convolucionais em um microcontrolador (Fedorov et al., 2019). Na figura 1.1, pode ser visto a diferença de consumo de energia entre alguns sistemas embarcados e um sistema de GPU (6049GP-TRT) voltado para servidores.

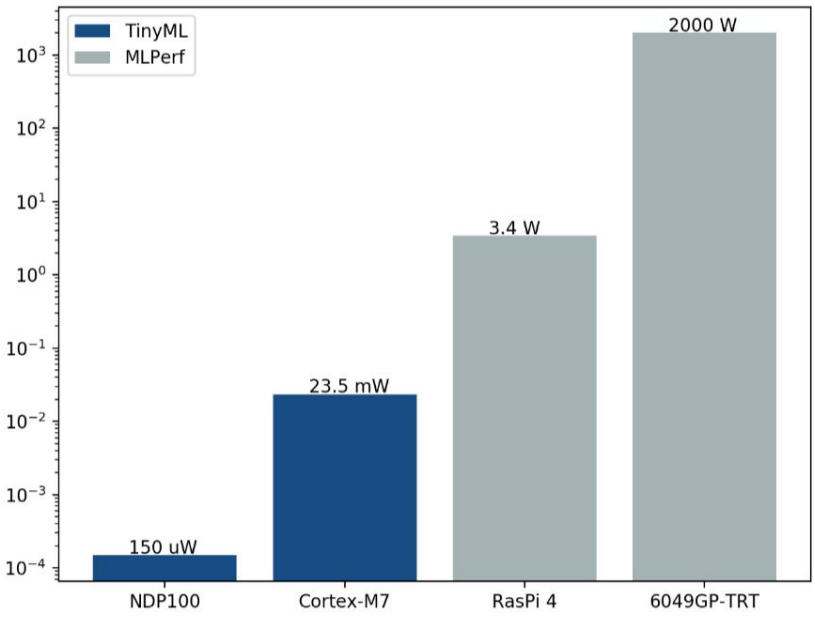


Figura 1.1: Comparação do consumo de energia entre sistemas de *TinyML* e aqueles suportados pelo *MLPerf* (Fonte: Banbury et al. (2021c))

Com a ascensão dos microcontroladores de 32-bits nos últimos anos, surgiu a viabilidade de rodar redes neurais extremamente compactas para tarefas de ML populares, como classificação de imagens, *Audio Wake Words*, *Visual Wake Words* e *Anomaly Detection* (Banbury et al., 2021c). A partir disso, a área de pesquisa conhecida como *TinyML* (tinyML Foundation, 2023) busca desenvolver ferramentas de software e hardware para embarcar modelos de redes neurais em computadores de baixo consumo e com memória e processamento limitados, como é o caso dos microcontroladores (tinyML Foundation, 2023; Banbury et al., 2021c) e de também computadores embarcados completos, como os da família Raspberry Pi. Além disso, essa área possui o potencial de tornar mais acessível o desenvolvimento de aplicações embarcadas de ML (Reddi et al., 2021) e expandir o universo de aplicações IoT (Banbury et al., 2021b; Saha et al., 2022), devido ao baixo custo do hardware necessário.

Atualmente existem algumas formas de se desenvolver sistemas de *TinyML*, como *Hand coding*, geração de código e interpretadores de aprendizado de máquina (Banbury et al., 2021c). Uma das mais promissoras é através de interpretadores de aprendizado de máquina, pois traz consigo portabilidade para se desenvolver para diversas plataformas, um framework popular que fornece esse recurso é o *TensorFlow Lite For Microcontrollers* (Banbury et al., 2021c; David et al., 2021; Saha et al., 2022).

Entretanto, existe pouco material de pesquisa na área de *TinyML*, principalmente de datasets delimitados e específicos (Banbury et al., 2021c). Como exemplo, pode se citar o CIFAR-10 (Krizhevsky, 2009), um dataset de 60.000 imagens de tamanho 32x32px divididas em 10 classes distintas que é muito utilizado em pesquisas de *TinyML*, em razão da portabilidade das suas imagens para computadores com recursos limitados (Banbury et al., 2021c,a). Na figura 1.2, pode ser visto a diferença entre a quantidade de tipos de computadores e a quantidade de cursos sobre *TinyML*.

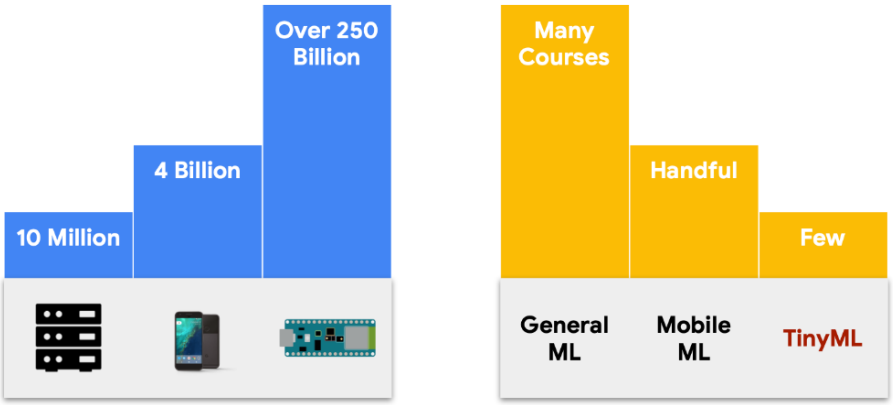


Figura 1.2: Número de tipos de computadores (servidores, dispositivos móveis e sistemas embarcados) e números de cursos de ML (ML tradicional, ML móvel e *TinyML*) (Fonte: Reddi et al. (2021))

1.1 MOTIVAÇÃO

No que tange a pesquisa de classificação de imagens com redes neurais, uma das arquiteturas que se destacam devido a sua simplicidade são as redes neurais residuais (He et al., 2015). Em particular, uma versão compacta de uma ResNet-8, embarcada em um microcontrolador de 32 bits, demonstrou um desempenho relevante no *dataset* CIFAR-10 com acurácia de 85% para um conjunto de 200 imagens de teste. Além disso, o tamanho desse modelo neural ocupou apenas 96 KB de memória (Banbury et al., 2021a).

Com isso, a arquitetura de rede neural residual aparenta ter o potencial de ser bem eficiente em termos de processamento e memória para realizar tarefas de classificação de imagens na grande maioria dos sistemas embarcados atuais. Com isso, há o interesse de se analisar e comparar o desempenho de alguns sistemas embarcados populares utilizando essas redes neurais.

Além disso, a realização desta pesquisa é uma oportunidade de explorar o contexto de aprendizagem de máquina com redes neurais para classificação de imagens em computadores com recursos limitados e de baixíssimo consumo e custo, sendo de interesse acadêmico e industrial.

1.2 OBJETIVOS

Essa pesquisa tem como objetivo geral analisar o desempenho de alguns sistemas embarcados populares utilizando redes neurais residuais compactas para classificação de imagens.

1.2.1 Objetivos Específicos

- Realizar uma revisão bibliográfica na área de classificação de imagens, arquiteturas de redes neurais convolucionais, técnicas de compressão de redes neurais, *TinyML* e hardwares embarcados.
- Embarcar as redes neurais residuais em distintos sistemas embarcados populares através de um *framework* de ML.
- Estudar e utilizar o *dataset* CIFAR-10 para testar redes neurais residuais, com foco nas redes embarcadas do tópico anterior.
- Analisar e comparar o desempenho dos hardwares embarcados escolhidos em termos de acurácia, tempo de execução e uso de energia dessas redes neurais residuais.

1.3 ESTRUTURA DO TRABALHO

O trabalho está estruturado da seguinte forma, o Capítulo 2 discorre sobre a fundamentação teórica para este estudo. O Capítulo 3 reúne e discute trabalhos relacionados a este, focando em estudos sobre benchmark em ML embarcado. No Capítulo 4, é descrito o projeto desenvolvido e sua metodologia, enquanto que no Capítulo 5 são apresentados os resultados obtidos. E por fim, no Capítulo 6 é realizada a conclusão final do trabalho e levantadas algumas questões para estudos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, será discorrida a base conceitual que fundamenta todo o trabalho realizado. O capítulo está dividido da seguinte forma: A seção 2.1 sobre a área de classificação de imagens, a seção 2.2 sobre as redes neurais convolucionais, a seção 2.3 sobre técnicas de compressão de redes neurais, a seção 2.4 sobre a área de pesquisa TinyML, a seção 2.5 irá cobrir algumas plataformas embarcadas populares para tarefas de IA, e por fim a seção 2.6 sumariza tudo o que foi visto no capítulo.

2.1 CLASSIFICAÇÃO DE IMAGENS

Classificação de imagens é uma área de pesquisa clássica de visão computacional, que possui como objetivo classificar um determinado objeto em uma cena, dentre um conjunto limitado de classes. Uma estrutura geral para um problema de classificação de imagens pode ser resumida em 4 fases: aquisição de imagens, pré-processamento de imagens, extração de características e classificação (Gonzalez e Woods, 2017).

A primera fase de aquisição de imagens é responsável pela obtenção de imagens digitais do problema em questão. Em seguida, na fase de pré-processamento, essas imagens podem ser tratadas de tal forma a melhorar o processamento delas para as próximas fases, por exemplo ajustando a resolução, brilho e contraste. Adiante na fase de extração de características, o objetivo é aplicar um método para realçar elementos de interesse na imagem, como a forma geométrica dos objetos, a fim de ser possível discriminar algum tipo de objeto. E por fim, com os resultados da última fase pode ser feita a classificação de(os) objeto(s) da imagem de entrada para uma classe possível definida pelo problema.

Em particular, na terceira e quarta fase podem ser utilizados métodos de descritores de características, como *Scale-Invariant Feature Transform* (SIFT) (Lowe, 2004) e *Speeded-Up Robust Features* (SURF) (Bay et al., 2006) aliados a classificadores como *Support Vector Machines* (SVM) (Scholkopf e Smola, 2001) para inferir a classe de um objeto em uma imagem. Entretanto, com o desenvolvimento e os avanços das redes neurais profundas, as redes neurais convolucionais ganharam destaque por seu alto desempenho na extração de características (Krizhevsky et al., 2012; Simonyan e Zisserman, 2015; He et al., 2015).

Na próxima seção, serão abordadas as redes neurais convolucionais e algumas de suas arquiteturas conhecidas.

2.2 REDES NEURAIS CONVOLUCIONAIS

As redes neurais convolucionais (CNNs) são abordagens modernas populares para aplicações de classificação de imagens, devido ao poder de extração de características de suas camadas convolucionais, as quais são capazes de capturar características espaciais e temporais de forma hierárquica, permitindo um aprendizado eficiente de representações complexas. Geralmente, no caso de classificação de imagens, essas redes são seguidas por um classificador, como por exemplo uma camada neural *fully-connected* que é responsável por fazer a inferência da classe. Para essas redes, existe um vasto cenário de aplicações como por exemplo na análise de imagens médicas (Razzak et al., 2017) e análise de textos manuscritos (Pashine et al., 2021).

Entretanto, essas redes neurais tendem a ser muito complexas, e com uma grande quantidade de parâmetros que são computacionalmente intensas para serem executadas em

ambientes com recursos de processamento e memória limitados (Molchanov et al., 2017; Han et al., 2016). Com isso, se motivou o estudo e desenvolvimento de redes neurais mais eficientes e que almejam um desempenho próximo das redes profundas.

Nesta seção serão apresentadas 3 arquiteturas de redes neurais convolucionais para classificação de imagens: ResNet, SqueezeNet e MobileNet.

2.2.1 ResNet

A ResNet (*Residual Neural Network)*, proposta em He et al. (2015), é uma arquitetura de rede neural inicialmente com o objetivo principal de resolver o problema da degradação de desempenho em redes muito profundas. A principal ideia da ResNet é o uso de blocos residuais, que permitem que um sinal de entrada *x* seja passado diretamente para camadas mais avançadas na rede através de conexões de atalho (*shortcut connections*). Este mecanismo permite a construção de redes muito profundas com muitas camadas, como a ResNet-50, ResNet-101 e ResNet-152.

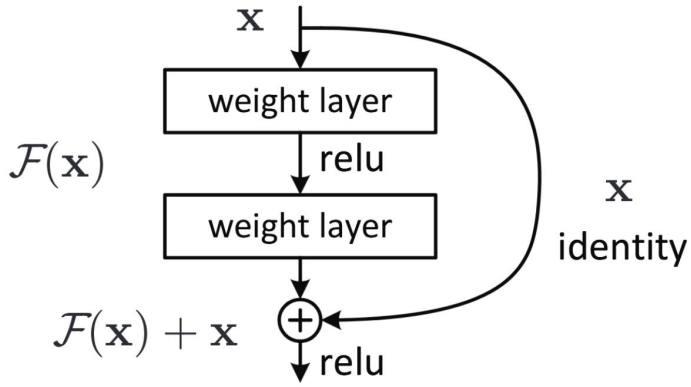


Figura 2.1: Arquitetura de um bloco residual (Fonte: He et al. (2015))

Essas redes demonstraram um grande desempenho em competições de reconhecimento de imagens, como a ILSVRC 2015 (He et al., 2015), e seu conceito se tournou base para outras arquiteturas modernas de CNNs. A partir disso, a capacidade de treinar redes mais profundas permitiu a extração de características complexas e uma melhora significativa na precisão de classificação de imagens.

Apesar da sua contribuição para redes profundas, existem versões de ResNet mais simples e menores como a ResNet-8 (Banbury et al., 2021a), sendo assim portáveis em sistemas embarcados, os quais possuem restrições de recursos computacionais.

2.2.2 SqueezeNet

A SqueezeNet, proposta por Iandola et al. (2016), busca minimizar o número de parâmetros em redes neurais profundas sem perder muita precisão. Essa arquitetura usa o que se chama de módulos *Fire*, que consistem em uma camada de compressão seguida por uma camada de expansão. A camada de compressão utiliza convoluções 1x1 para reduzir o número de canais de entrada, enquanto a camada de expansão usa uma combinação de convoluções 1x1 e 3x3 para recuperar a capacidade de representação.

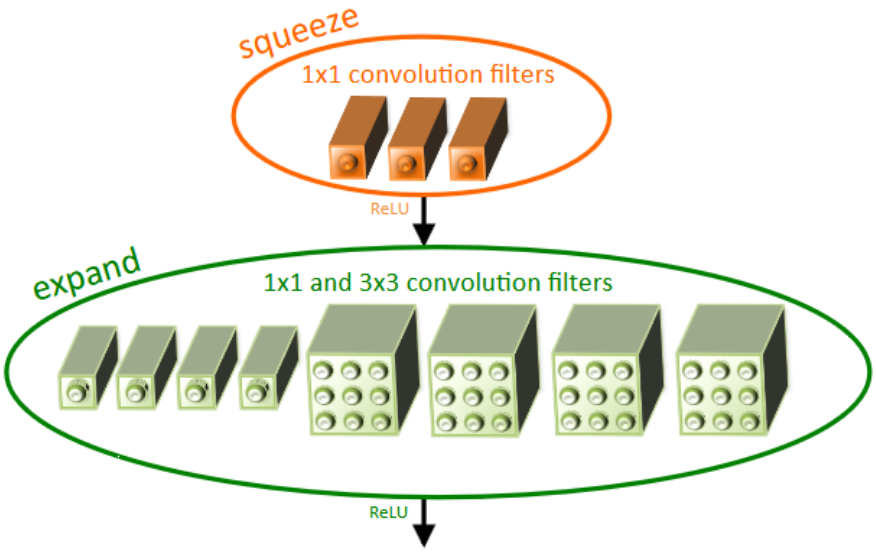


Figura 2.2: Arquitetura de um módulo *Fire* (Fonte: Iandola et al. (2016))

Essa arquitetura reduz bastante o número de parâmetros, sendo assim eficiente em termos de memória, o que é particularmente útil para aplicações em dispositivos com recursos limitados.

2.2.3 MobileNet

A MobileNet, proposta em Howard et al. (2017), foi projetada com o objetivo de otimizar o uso de CNNs em dispositivos móveis e embarcados, onde os recursos computacionais e de energia são limitados. Para atingir esse objetivo, a MobileNet introduz o conceito de convoluções separáveis em profundidade (*depthwise separable convolutions*), que decompõem a operação de convolução padrão em duas etapas mais simples e eficientes: convoluções em profundidade e convoluções ponto a ponto.

Com isso, essa abordagem reduz bastante o número de parâmetros e operações computacionais, mantendo um bom desempenho na tarefa de classificação de imagens. A MobileNet também permite ajustes de *trade-off* entre latência e precisão através de hiperparâmetros que controlam a largura e a resolução da rede, tornando-a altamente flexível para diferentes aplicações (Howard et al., 2017).

2.3 COMPRESSÃO DE REDES NEURAIS

A compressão de redes neurais é uma área de pesquisa focada em reduzir o tamanho e a complexidade de modelos neurais, tornando-os mais eficientes em termos de armazenamento e processamento, com o objetivo de minimizar a perda de desempenho. Esta seção irá abordar sobre duas técnicas comuns utilizadas para compressão de redes neurais: *pruning* e quantização.

2.3.1 *Pruning*

O *pruning*, ou "poda", é uma técnica que visa remover conexões ou neurônios redundantes em uma rede neural, resultando em um modelo mais eficiente e que tenta garantir o mínimo possível de redução da acurácia. Esta técnica se baseia na observação de que algumas das

conexões em uma rede neural treinada tem pouca importância no desempenho da rede (Han et al., 2015; Molchanov et al., 2017).

Em Han et al. (2015) é proposto um método de *pruning* em um pipeline de três etapas: treinamento de conectividade da rede para aprender quais conexões são importantes, remoção das conexões que possuem peso abaixo de um limiar pré-definido, e por fim é feito um treinamento sobre essa nova rede resultante para aprender novos pesos. Em Molchanov et al. (2017) é proposto algo semelhante, mas com foco na utilização de uma política de *pruning* diferente que é baseada na expansão de Taylor.

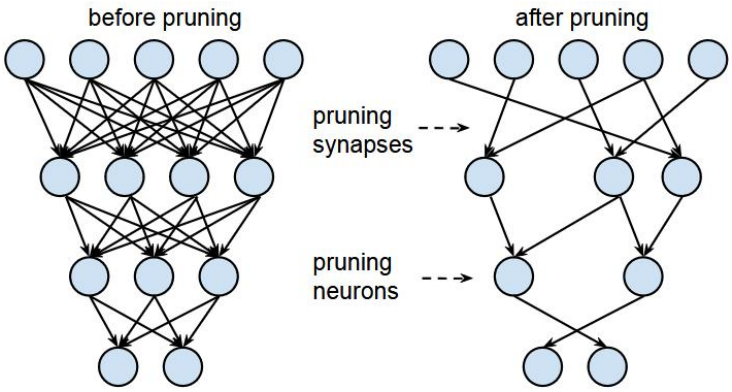


Figura 2.3: Exemplo de uma rede neural após aplicação do *pruning* (Fonte: Han et al. (2015))

A partir disso, essa técnica pode resultar em modelos significativamente menores e mais rápidos, tornando-os mais adequados para implantação em dispositivos com recursos limitados. Além disso, essa técnica pode ser combinada com outras estratégias de compressão, como a quantização, para alcançar ainda maiores reduções no tamanho do modelo (Han et al., 2016).

2.3.2 Quantização

A quantização é outra técnica de compressão que possui como objetivo reduzir a precisão dos pesos e ativações de uma rede neural, representando-os com um menor número de *bits*. Por exemplo, ao invés de utilizar a representação em ponto flutuante de 32 *bits*, a quantização pode reduzir os pesos para representações em ponto flutuante ou inteiras de 16 *bits*, 8 *bits* ou até menos, dependendo do nível de compressão desejado.

Além disso, existem estratégias de quantização que procuram realizar a conversão da representação em ponto flutante para uma representação em ponto fixo (Novac et al., 2021), na qual um número real possa ser descrito em duas partes inteiras de tamanho fixo, uma representando a parte inteira e uma outra a parte fracionária, ao invés de seguir o padrão IEEE754 de ponto flutuante com sinal, expoente e mantissa (iee, 2019). Essa representação em ponto fixo é interessante de ser aplicada em ambientes com recursos limitados como microcontroladores, pois podem ser utilizadas apenas operações de inteiros que são mais rápidas e eficientes do que operações de ponto flutuante (Jacob et al., 2017; Novac et al., 2021). Além disso, em projetos de sistemas embarcados com certificação SIL 3, muitas vezes opta-se pelo uso de ponto fixo em vez de ponto flutuante, devido à sua maior segurança, previsibilidade e facilidade de teste.

Portanto, a quantização pode levar a grandes economias em termos de armazenamento e eficiência computacional, permitindo que modelos complexos sejam executados em hardware

com capacidades limitadas, como dispositivos móveis e embarcados (Novac et al., 2021). No entanto, é importante considerar o *trade-off* entre a redução de precisão e a perda de desempenho para garantir que a quantização seja eficaz para a aplicação específica.

2.4 TINY ML

TinyML é uma subárea de ML que se tornou popular nesses últimos anos, pois ela busca trazer o potencial das aplicações de ML para o mundo dos sistemas embarcados de baixo custo e consumo (David et al., 2021; Reddi et al., 2021; Banbury et al., 2021c). Esta é uma área considerada promissora, pois busca eliminar as dificuldades do ML tradicional, como alto custo do hardware comumente utilizado e a disponibilidade de dados, além de tornar mais acessível o desenvolvimento e ensino de ML (Reddi et al., 2021).

Além disso, a implantação de ML em sistemas embarcados de baixo custo possibilita a execução e interpretação local, trazendo grandes vantagens em termos de privacidade e latência (Banbury et al., 2021b), gerando assim uma independência de uma infraestrutura de rede para ser aplicada (Saha et al., 2022), em contraste com ML tradicional que geralmente se confina em ambientes de nuvem e que são acessados remotamente para realizar alguma tarefa (Reddi et al., 2021).

Algumas das principais aplicações de TinyML que podem ser mencionadas são *Image Recognition*, *Speech Recognition*, *Anomaly Detection* e *Gesture Tracking*. Em particular, essas aplicações são bem comuns e utilizadas em benchmarks de TinyML (Banbury et al., 2021c,a; Saha et al., 2022), os quais tem como objetivo mensurar o desempenho de diversas plataformas embarcadas e alanvancar a pesquisa na área.

Entretanto, implementar ML em dispositivos embarcados de baixa potência apresentam diversos desafios, incluindo restrições de memória, capacidade computacional limitada e restrições energéticas (Banbury et al., 2021c; Saha et al., 2022). Para superar esses desafios, várias técnicas são empregadas, em particular para modelos neurais podem ser aplicadas técnicas de compressão como *pruning* e quantização, conforme discutido na seção 2.3, a fim de reduzir o tamanho e a complexidade dos modelos.

Além disso, a heterogeneidade dos hardwares embarcados é muito grande, pois existem diversos tipos de arquitetura e pode ser uma tarefa complexa adaptar o mesmo modelo neural compilado de um sistema para outro (Banbury et al., 2021c; Saha et al., 2022). Por essa razão que frameworks de ML como o *TensorFlow Lite* (TFL) e *TensorFlow Lite For Microcontrollers* (TFLM) (David et al., 2021) facilitam a implantação de modelos neurais para diversos tipos de sistemas embarcados, ao fornecer ferramentas e bibliotecas otimizadas para esse propósito. Para o *TensorFlow Lite For Microcontrollers*, umas das estratégias adotadas é o uso de um interpretador de ML dentro do sistema embarcado, sendo responsável por gerenciar os recursos e execução do modelo, permitindo assim uma portabilidade melhor entre plataformas.

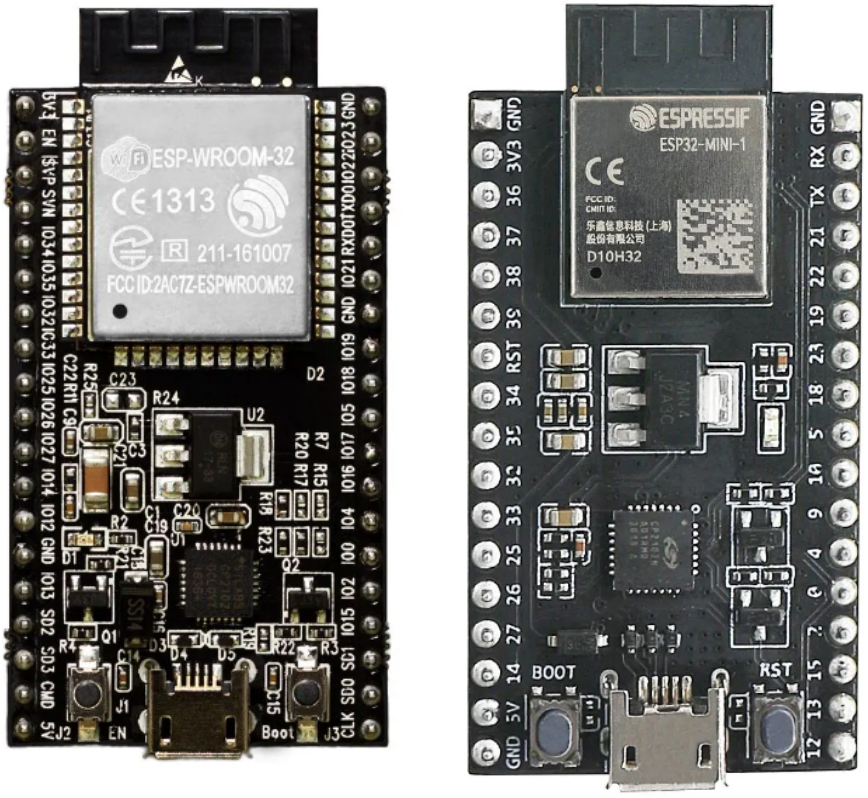
2.5 HARDWARES EMBARCADOS

Nesta seção serão apresentadas algumas plataformas embarcadas populares que podem ser utilizadas para executar modelos neurais para classificação de imagens. As plataformas em questão são: ESP32, Raspberry Pi e NVIDIA Jetson.

2.5.1 ESP32

Essa família de chips foi inicialmente lançada em 2016 pela empresa Espressif Systems, uma multinacional chinesa de semicondutores, e correspondem a um conjunto de microcontrola-

dores de 32 *bits* de baixo consumo de energia e baixo custo. Tornando-se populares ao longo dos últimos anos devido à sua robustez, versatilidade e confiabilidade em diversos tipos de aplicação, como por exemplo automação residencial e industrial, agricultura inteligente e reconhecimento de imagens (esp, 2024).



(a) ESP32-DevKitC (b) ESP32-DevKitM-1

Figura 2.4: Exemplo de placas de desenvolvimento embarcadas com o chip ESP32 (Fonte 2.4(a): Adaptado de Electronics (2024a); Fonte 2.4(b): Adaptado de Electronics (2024b))

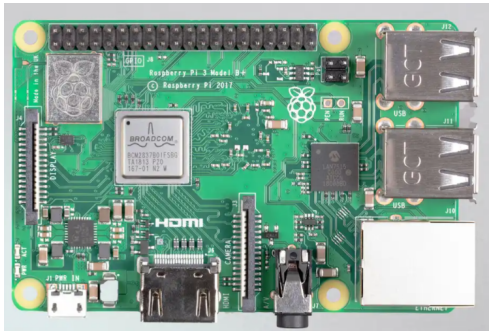
Como sucessor direto do chip ESP8266, lançado pela mesma empresa em 2014, os chips ESP32 são compostos pelo microprocessador Xtensa 32-bit LX6 microprocessor (single ou dual-core) que opera a uma frequência de até 240 MHz (esp, 2024). Eles dotam também de uma memória ROM de 448 KB (para *booting* e funcionalidades internas) e uma mémoria SRAM de 520 KB (para dados e instruções), e suporte de módulos de memória flash externos até uma capacidade total de 16 MB (esp, 2024). Além disso, esses chips possuem interfaces para módulos de Wi-Fi e Bluetooth, expandindo assim o universo de aplicações que podem ser usados (esp, 2024). Ademais, visando um consumo de energia eficiente, esses chips dispõem de 5 modos de energia projetados para diversos cenários: *Active*, *Modem-sleep*, *Light-sleep*, *Deep-sleep*, *Hibernation* (esp, 2024).

Geralmente, eles são embarcados e comercializados em placas de desenvolvimento, por exemplos nas placas ESP32-DevKitC e ESP32-DevKitM-1. Permitindo assim um processo rápido de desenvolvimento de aplicações, pois oferecem uma interface de hardware mais abstraída e intuitiva do que utilizar diretamente os chips (Systems, 2024).

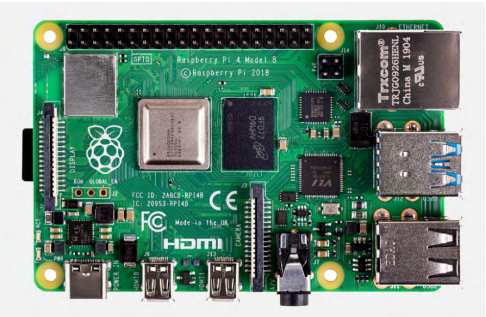
2.5.2 Raspberry Pi

Desde 2012, a empresa Raspberry Pi Ltd projeta computadores embarcados e de propósito geral com foco primário na fomentação do ensino e pesquisa de computação com tecnologias acessíveis. Entretanto, a plataforma Raspberry Pi se tornou popular ao longo de todos esses anos, principalmente devido a sua portabilidade, baixo custo e alto desempenho. Sendo assim utilizada em várias outras áreas, como automação residencial e industrial, aplicações de robótica, ML, etc.

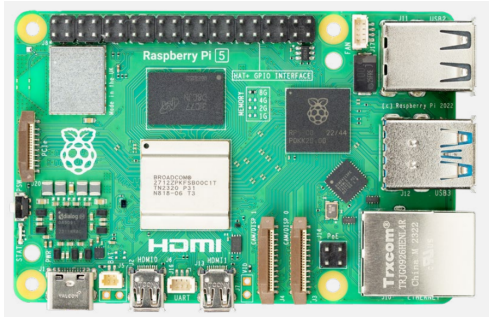
Alguns dos computadores embarcados, também conhecidos como SBCs (*Single Board Computers*), lançados são o Raspberry Pi 1, 2, 3, 4 e recentemente o 5 (Ltd, 2024b). E que apesarem de serem mais custosos que plataformas de microcontroladores, esses computadores tem mais recursos de processamento e memória, tendo a capacidade de executar sistemas completos, como um sistema operacional com diversas aplicações. Até hoje, esses SBCs utilizam chips Broadcom, desenvolvidos pela empresa de semicondutores Broadcom Inc., que na sua grande maioria embarcam processadores de 64-bits que operam a frequências na faixa de GHz, como por exemplo os utilizados no Raspberry Pi 3B+, 4B e 5, respectivamente os processadores Arm Cortex-A53 (1,4 GHz), Arm Cortex-A76 (1,5 GHz) e Arm Cortex-A76 (2,4 GHz) (Ltd, 2024a; pi4, 2023; pi5, 2023). Em termos de memória, esses SBCs são geralmente fabricados com quantidades razoáveis de RAM na ordem de GBs, como é o caso do Raspberry Pi 3B+ que possui 1 GB de RAM e o Raspberry Pi 5 que possui um modelo de 8 GB (Ltd, 2024a; pi4, 2023; pi5, 2023).



(a) Raspberry Pi 3 Model B+



(b) Raspberry Pi 4 Model B



(c) Raspberry Pi 5

Figura 2.5: Alguns SBCs da família Raspberry Pi (Fonte 2.5(a): Adaptado de Ltd (2024a); Fonte 2.5(b): Adaptado de pi4 (2023); Fonte 2.5(c): Adaptado de pi5 (2023))

Entretanto em 2021, a família Raspberry Pi se expandiu para o universos de plataformas microcontroladas, com o lançamento do Raspberry Pi Pico, que é uma placa embarcada com o chip RP2040, um microcontrolador dotado do processador Dual ARM Cortex-M0+ (32-bit) que opera a uma frequência de até 133 MHz e que possui sistemas avançados de controle de energia,

com foco no baixo consumo energético e alto desempenho. Além disso, esse chip apresenta uma SRAM embutida de 264 KB para dados e instruções (piP, 2023; rp2, 2024).

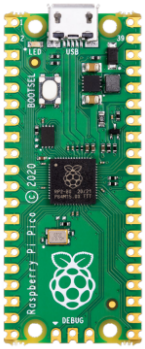


Figura 2.6: Raspberry Pi Pico (Fonte: Adaptado de piP (2023))

A chegada do Raspberry Pi Pico é um indicativo de como as plataformas embarcadas microcontroladas estão ganhando força nos últimos tempos, devido a sua baixa latência na execução de tarefas, baixo consumo de energia e baixo custo.

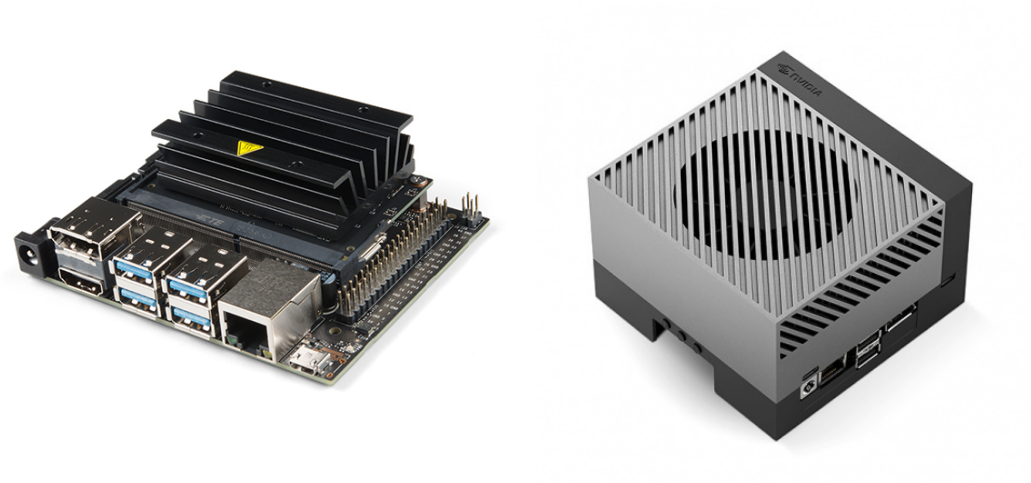
2.5.3 NVIDIA Jetson

Fundada em 1993, a NVIDIA é uma empresa de referência se falarmos no desenvolvimento de tecnologias de hardware de computação gráfica e de alto desempenho, permeando vários segmentos do mercado de tecnologia como jogos eletrônicos, robótica aplicada, e principalmente hoje em dia o setor de inteligência artificial (Corporation, 2023). Popularmente conhecida por suas poderosas séries de GPUs, como as linhas GTX e RTX, a NVIDIA tem um impacto significativo no desenvolvimento de tecnologia de hardware no mundo atual.

Entretanto, em 2014 a NVIDIA lançou um computador embarcado de alto desempenho, a NVIDIA Jetson TK1 Developer Kit, orientado principalmente a aplicações de visão computacional em áreas como robótica e medicina por exemplo (Corporation, 2014). Marcando assim o início da família NVIDIA Jetson, a qual prosseguiu ao longo dos anos com a série TX1 (Corporation, 2015), TX2 (Corporation, 2017), AGX Xavier (Gopalakrishna, 2018), NANO (Corporation, 2019a), Xavier NX (Corporation, 2019b), e mais recentemente em 2022 com a linha Jetson Orin (Shivashankar, 2022).

Um dos modelos populares dessa família é a NVIDIA Jetson NANO, que possui uma versão em uma placa embarcada, a NVIDIA Jetson NANO Developer Kit, e utiliza um processador Quad-core ARM Cortex-A57 64-bit que opera até 1,43 GHz, uma GPU de 128 núcleos baseada na arquitetura NVIDIA Maxwell que opera até 921 MHz e uma RAM de 4 GB (Corporation, 2024a). Entretanto, existem modelos mais potentes como a NVIDIA Jetson AGX Orin 64 GB, que possui um processador 12-core ARM Cortex-A78AE 64-bit que opera até 2,2 GHz, uma GPU de 2048 núcleos baseada na arquitetura NVIDIA Ampere que opera até 1,3 GHz, uma RAM de 64 GB, e também possui um DLA (*Deep Learning Accelerator*) que um é hardware especializado para acelerar cargas de trabalho relacionadas a Deep Learning (Corporation, 2024b). Em contrapartida este último é um modelo muito mais caro que a NVIDIA Jetson NANO.

Atualmente, a NVIDIA como referência em tecnologias de hardware para IA, possui um acervo de plataformas embarcadas de ponta no mercado, que podem ser utilizadas para cargas de trabalho de ML. Entretanto, uma ressalva é que a grande maioria dessas plataformas tem um custo elevado, não sendo assim tão acessível quanto as plataformas anteriores.



(a) NVIDIA Jetson Nano Developer Kit V3 (b) NVIDIA Jetson AGX Orin 64GB Developer Kit

Figura 2.7: Exemplo de placas de desenvolvimento embarcadas da família NVIDIA Jetson (Fonte 2.7(a): Adaptado de Electronics (2024d); Fonte 2.7(b): Adaptado de Electronics (2024c))

2.6 CONCLUSÃO

Este capítulo teve o objetivo de fornecer um panorama geral sobre classificação de imagens, arquiteturas de redes neurais convolucionais, métodos de compressão de redes neurais, e em apresentar a área de TinyML, mostrando o seu potencial na área de ML, além de listar alguns hardwares embarcados candidatos a serem utilizados com aplicações de ML. No próximo capítulo, serão abordados alguns trabalhos relacionados a este em questão.

3 TRABALHOS RELACIONADOS

Neste capítulo serão apresentados alguns estudos relacionados ao presente trabalho, mais especificamente trabalhos sobre benchmarks em sistemas de ML embarcado, que buscam oferecer uma forma de análise de desempenho de plataformas embarcadas em atividades populares de ML, como por exemplo classificação de imagens.

3.1 MLPERF TINY

Um dos trabalhos sobre benchmarks de TinyML, incluem o de Banbury et al. (2021a) que propõe um benchmark *open source* para sistemas de TinyML, o MLPerf Tiny, o qual engloba 4 tipos de aplicações: *Keyword Spotting*, *Visual Wake Words*, *Image Classification* e *Anomaly Detection*.

Para esse benchmark, as métricas de desempenho consideradas são de acurácia, latência e consumo de energia dos modelos de ML. A escolha para essas métricas e aplicações consideraram necessidades do meio indústrial e acadêmico, provendo assim uma forma de analisar e comparar o desempenho de diversas plataformas embarcadas para atividades populares de ML.

Mais especialmente para a aplicação de classificação de imagens, é utilizado o modelo neural ResNet-8, uma versão compacta baseada na arquitetura neural ResNet, para classificar um conjunto aleatório de 200 imagens do dataset CIFAR-10.

Além disso, em Banbury et al. (2021c) é discorrido os desafios na construção de um benchmark de TinyML representativo e justo na medição de sistemas de TinyML, alguns deles são a medição do consumo de energia, memória limitada, heterogeneidade de hardwares embarcados e suporte de ferramentas de software voltadas para TinyML.

3.2 DEEPEDGEBENCH

Com interesse em estudar o desempenho de modelos neurais profundos em sistemas com recursos limitados, o benchmark *DeepEdgeBench* proposto em Baller et al. (2021), busca oferecer um método de avaliação de sistemas embarcados em termos de consumo de energia, tempo de inferência e acurácia.

Neste estudo, é comparado e avaliado o desempenho de modelos comprimidos e não comprimidos de MobileNets na classificação de imagens do dataset ImageNet ILSVRC2012 em diversas plataformas embarcadas: Asus Tinker Edge R, Raspberry Pi 4, Google Coral Dev Board, Nvidia Jetson Nano e Arduino Nano 33 BLE.

Além disso, esse estudo endereça uma questão de eficiência energética de um sistema embarcado para uma aplicação de ML de classificação de imagens com base no estilo da aplicação, o qual pode ser de computação contínua ou esporádica. O primeiro se preocupa com o consumo geral de energia para inferência contínua de um conjunto de imagens, ao passo que a segunda leva em consideração uma inferência esporádica em que o consumo de energia é impactado quando o sistema está ocioso.

3.3 YOLOBENCH

No contexto de detecção de objetos, o benchmark YOLOBench proposto em Lazarevich et al. (2023), oferece uma abordagem de comparação justa entre diferentes versões de modelos

baseados em YOLO (*You Only Look Once*), a fim de demonstrar o impacto em aplicações embarcadas. Além disso, com esse benchmark é visado endereçar os desafios no desenvolvimento de modelos eficientes de detecção de objetos baseados em YOLO.

3.4 CONCLUSÃO

Em relação aos trabalhos mencionados anteriormente, o MLPerf Tiny (Banbury et al., 2021a) apenas utiliza a ResNet-8 e realiza a classificação de apenas 200 imagens do conjunto de teste do CIFAR-10, neste caso seria interessante avaliar mais modelos de ResNet e classificar o conjunto de teste inteiro do CIFAR-10 que correspondem a 10.000 imagens, a fim de melhorar o entendimento do desempenho das ResNets em sistemas embarcados e gerar um resultado mais preciso em relação ao *dataset* utilizado.

No que tange o *DeepEdgeBench* (Baller et al., 2021) e YOLOBench (Lazarevich et al., 2023), apenas algumas das plataformas utilizadas são bem acessíveis, como é o caso do Raspberry Pi 4 e Arduino Nano 33 BLE, sendo esta última plataforma apenas usada no *DeepEdgeBench*. Por essa razão, a inclusão de mais plataformas embarcadas acessíveis para pesquisa é interessante para ampliar a análise e entender os limites de uma aplicação, além de facilitar a reprodução de experimentos e testes.

No próximo capítulo, será discorrido sobre o projeto desenvolvido e metodologia deste trabalho, buscando endereçar as questões levantadas nesta seção.

4 PROJETO DESENVOLVIDO

Neste capítulo, será descrito as escolhas feitas para realização do trabalho, sobretudo no que diz respeito a metodologia dos experimentos: o *dataset* utilizado, as arquiteturas de redes residuais utilizadas, o treinamento dessas redes, os hardwares alvos de execução e as métricas de desempenho e eficiência.

4.1 DATASET

O *dataset* escolhido para pesquisa foi o CIFAR-10, o qual corresponde a um conjunto total de 60.000 imagens de tamanho 32 por 32 *pixels*, segmentado em 50.000 imagens para treinamento (5.000 imagens por classe) e 10.000 para teste (1.000 imagens por classe). Sendo que cada imagem desse *dataset* pode ser classificada dentre 10 classes possíveis: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship* e *truck* (Krizhevsky, 2009).

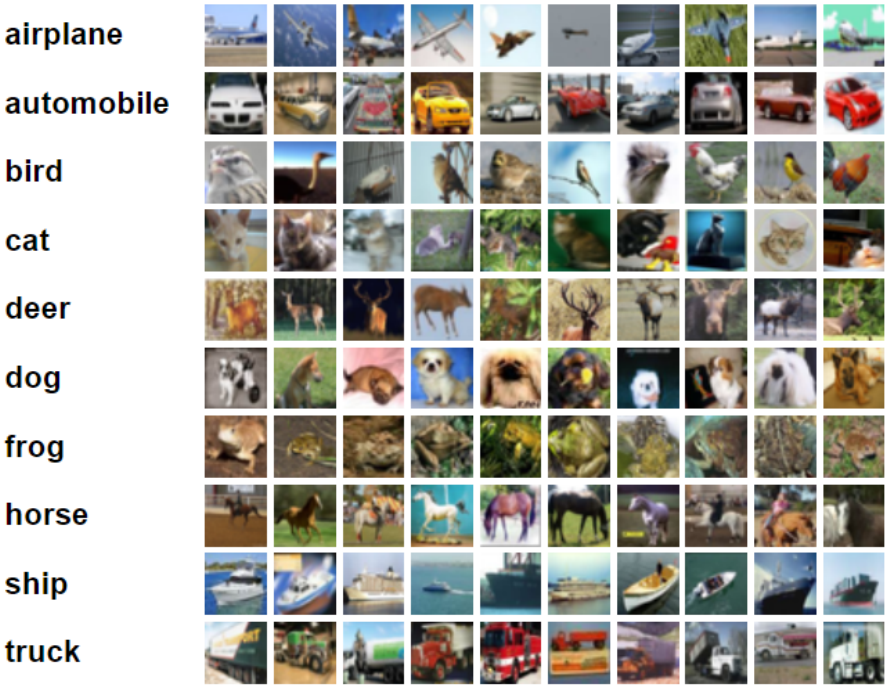


Figura 4.1: Amostra de 10 imagens para cada classe do *dataset* CIFAR-10 (Fonte: Adaptado de Krizhevsky (2024))

A principal razão para se ter escolhido esse *dataset* são suas imagens compactas, pois cada imagem pode ocupar 3 KiB [1] de espaço em memória, sendo uma quantidade acessível para a maioria dos sistemas embarcados. Além disso, esse *dataset* é uma referência clássica para pesquisa de classificação de imagens.

Para o trabalho em questão, foi feito apenas uma única etapa de pré-processamento das imagens que foi a normalização dos *pixels* das imagens para o intervalo $[0..1]$, a fim de facilitar o processamento nas redes neurais.

[1] Cada *pixel* da imagem tem 3 canais de cor (RGB) e cada um pode ser codificado em 1 byte, então $32 * 32 * 3 = 3072$ bytes.

4.2 REDES NEURAIS RESIDUAIS

A princípio foi utilizado a versão de arquitetura de redes neurais residuais apresentada em He et al. (2015) que é usada no *dataset* CIFAR-10. Nesta arquitetura residual básica, a quantidade de camadas da rede é escalada linearmente através da seguinte fórmula: $6n + 2$, sendo n o número de blocos residuais. A partir disso, foram desenvolvidos 4 modelos residuais: ResNet-8 ($n = 1$), ResNet-14 ($n = 2$), ResNet-20 ($n = 3$) e ResNet-26 ($n = 4$).

Layer	Output size	ResNet-8	ResNet-14	ResNet-20	ResNet-26
conv1	$32 \times 32 \times 16$	3x3 kernel size, 16 filters, stride 1			
conv2.x	$32 \times 32 \times 16$	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 16 \\ 3 \times 3, 16 \end{bmatrix} \times 4$
conv3.x	$16 \times 16 \times 32$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix} \times 4$
conv4.x	$8 \times 8 \times 64$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 1$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 4$
	1×10	AveragePooling $\rightarrow$ 10-d fully-connected $\rightarrow$ softmax			

Tabela 4.1: Arquitetura simplificada das quatro redes residuais utilizadas. Os blocos residuais estão representados entre colchetes junto com a sua quantidade ao lado. As alterações de dimensões das *feature maps* ocorrem no primeiro bloco residual de conv3.x e conv4.x.

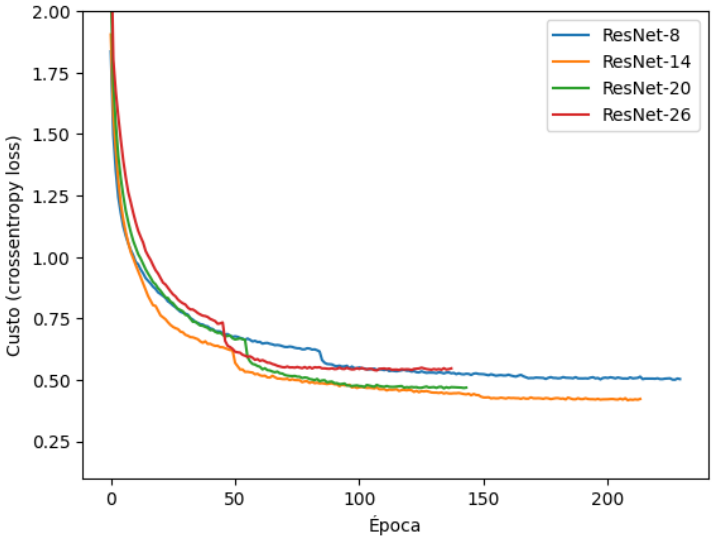
4.2.1 Detalhes de implementação

As redes neurais foram implementadas através da plataforma *TensorFlow*, os *kernels* das camadas convolucionais são inicializados com valores aleatórios que obedecem uma distribuição normal, ademais utilizam regularização L2 com valor 0,0001. Além disso, foi colocado *batch normalization* após cada camada convolucional e antes da função de ativação como feito em He et al. (2015). As conexões *shortcut* para cada bloco residual foram implementadas como uma camada convolucional que ajusta a dimensão da entrada do bloco, como sugerido em He et al. (2015). Para fins de treinamento, foi aplicado a técnica de *Dropout* com valor 0,2 após cada execução da função de ativação.

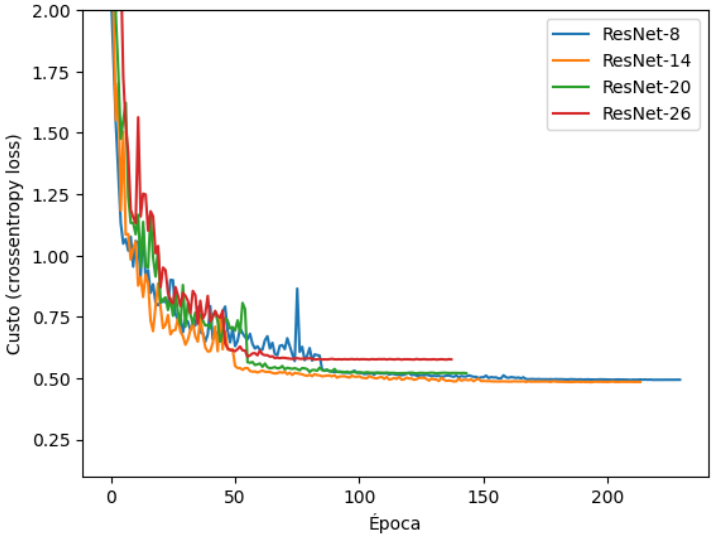
4.2.2 Treinamento

Para o treinamento das redes neurais, foram utilizados *batch size* de 128, número máximo de 300 épocas, *learning rate* inicial de 0,001 que decresce exponencialmente em potências de base 10 a cada 10 épocas seguidas quando não há melhorias na rede, e técnica de *Early Stopping* para 20 épocas seguidas sem melhorias na rede, começando a contar a partir de 100 épocas treinadas. Além disso, a evolução do treinamento era considerada pelo desempenho de um conjunto de validação, o qual corresponde a 10% do conjunto de treinamento e separado antes da execução de uma época.

Por fim, o ambiente de execução do treinamento foi feito pelas máquinas do *Google Colab* utilizando uma GPU NVIDIA L4 através de uma inscrição paga *Colab Pro*.



(a) Conjunto de treinamento (45.000 imagens)



(b) Conjunto de validação (5.000 imagens)

Figura 4.2: Históricos de custo (*crossentropy loss*) ao longo do treinamento das ResNets.

4.2.3 Modelos resultantes

Após o treinamento dos modelos neurais definidos na seção 4.2, foram deduzidos para cada um sua versão comprimida, utilizando a técnica padrão de compressão do *TensorFlow Lite* de *post-training quantization*. Totalizando assim 8 modelos de redes neurais residuais.

Modelo	**Tamanho (MiB)**		**Proporção**
	Original	Comprimido	
ResNet-8	1,0337	0,0913	11,322
ResNet-14	2,242	0,1942	11,5448
ResNet-20	3,453	0,2971	11,6223
ResNet-26	4,663	0,4001	11,6546

Tabela 4.2: Tamanho dos modelos de ResNets gerados.

O objetivo da técnica de *post-training quantization* é reduzir o tamanho do modelo com o mínimo de perda de acurácia possível. Foi optado o método por *full integer quantization*, o qual quantiza todos os parâmetros do modelo para inteiros, isso é importante para a execução em sistemas embarcados, pois operações com inteiros são mais eficientes do que com números de ponto flutuante.

4.3 HARDWARES EMBARCADOS

Foram utilizados a placa de desenvolvimento embarcada ESP32-DevKitC [2] e um Raspberry Pi 3B+. A plataforma do ESP32 foi adquirida externamente, ao passo que a plataforma Raspberry foi emprestada pelo professor orientador. Esses dois hardwares foram escolhidos devido ao seu baixo custo e popularidade no mundo de tecnologia embarcada.

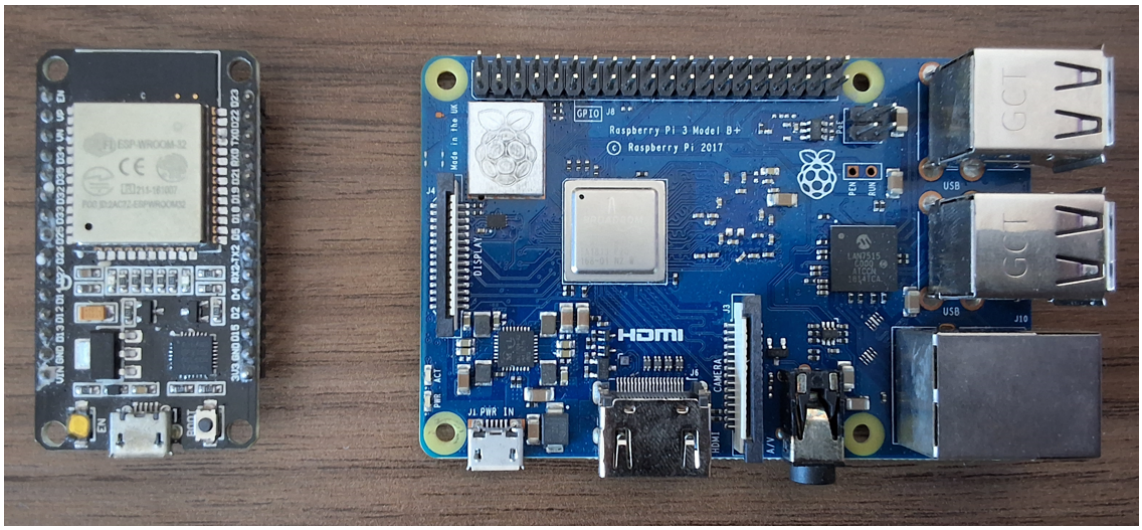


Figura 4.3: Placas ESP32-DevKitC (esquerda) e Raspberry Pi 3B+ (direita)

	ESP32-DevKitC	**Raspberry Pi 3B+**
Processador	Xtensa 32-bit LX6 @ 240 MHz	Arm Cortex-A53 64-bit @ 1.4 GHz
RAM	520 KB	1 GB
Armazenamento	4 MB (Flash)	64 GB (Cartão SD) [3]
SO	-	Ubuntu Server 24.04 LTS

Tabela 4.3: Especificação dos hardwares utilizados.

Por questões de compatibilidade e suporte com o ambiente do TensorFlow, foi escolhido o sistema operacional Ubuntu Server 24.04 LTS para o Raspberry Pi 3B+. Além disso, para a plataforma ESP32 foi necessário embarcar o ambiente TensoFlow Lite for Microcontrollers, e por essa razão que só foram consideradas as versões comprimidas das ResNets nesta plataforma, devido ao tamanho e incompatibilidade das versões originais.

[2] Contém o chip ESP32-D0WDQ6, que é um chip popular da família do ESP32.
[3] Adquirido externamente.

4.4 MÉTRICAS

Nesta seção, serão descritas as métricas utilizadas para análise de desempenho e eficiência das plataformas embarcardas na execução dos modelos neurais definidos na seção 4.2 para inferir o conjunto de teste padrão do *dataset* CIFAR-10. As métricas escolhidas foram acurácia e tempo de execução médio dos modelos, e a potência média gasta pelo sistema embarcado para cada modelo. A partir dessas métricas pode-se validar o desempenho dos modelos (acurácia e tempo de execução) e a eficiência (potência gasta) dos mesmos. Além disso, vale destacar que para extração dessas métricas foi apenas realizada uma única execução para cada modelo neural, isso deve-se principalmente a limitação de tempo imposta pela plataforma ESP32.

4.4.1 Acurácia

Foram calculados a acurácia média e o F1-Score médio dos modelos neurais utilizados para classificar todas as 10.000 imagens, com o objetivo de avaliar o desempenho desses modelos na tarefa de classificação de imagens proposta. É esperado que tenha alguma redução de acurácia no uso das redes neurais comprimidas, devido a simplificação gerada pela técnica de quantização.

4.4.2 Tempo de execução

O cálculo do tempo de execução dos modelos foi feito em nível de software em ambas as plataformas. Foram utilizados contadores de tempo padrões dos sistemas utilizados para se calcular o início e fim da inferência do modelo neural. No ESP32 foi medido em microssegundos e posteriormente convertido em segundos, ao passo que no Raspberry Pi 3B+ foi medido diretamente em segundos.

4.4.3 Potência do sistema

A extração de potência de cada sistema embarcado foi feito a partir de um hardware externo que monitora os dados de energia entre o sistema e uma fonte de energia. A partir desse hardware de medição de energia, foi possível extrair a potência do sistema ao longo do tempo na inferência de todas as imagens do conjunto de teste. Com isso, é possível calcular a potência média do sistema em *Watts* (W), e em conjunto com o tempo de execução médio pode ser calculado também o consumo de energia em *Joules* (J).

(a) UM34C - Dispositivo de medição de energia via USB

(b) Medição de energia com o ESP32

Figura 4.4: Dispositivo de medição 4.4(a) e um exemplo de uso 4.4(b)

Com esse medidor, é possível se conectar via Bluetooth para extrair os dados de energia mensurados, e assim armazená-los num arquivo formatado.

4.5 CONCLUSÃO

No geral, a ideia é fazer uma análise de desempenho e eficiência de 2 sistemas embarcados populares, ESP32 e Raspberry Pi 3B+, para classificação de imagens do *dataset* CIFAR-10 utilizando ResNets. O desempenho será considerado em termos de acurácia e tempo de execução médios dos modelos, e a eficiência na potência média gasta pelos sistemas embarcados.

A partir disso, é esperado que na plataforma ESP32, devido ao seu processamento e memória limitados, se tenha um tempo de execução maior do que no Raspberry Pi 3B+. Entretanto, é esperado que sua potência gasta seja menor do que no Raspberry Pi 3B+, sendo

assim mais eficiente. Em relação a acurácia, espera-se que cada versão comprimida da ResNet tenha um valor menor que à sua versão original não comprimida, pois o que impactaria essa métrica seria apenas o nível de compressão da rede e não a limitação do hardware. Da mesma forma, é esperado que a acurácia, para uma mesma versão de modelo neural, seja similiar independente da plataforma.

No próximo capítulo, serão mostrados os resultados obtidos para cada métrica considerada em ambas as plataformas embarcadas.

5 RESULTADOS

Neste capítulo serão apresentados e analisados os resultados obtidos de acurácia, tempo de execução e potência gasta dos modelos de ResNets em dois sistemas embarcados, ESP32 e Raspberry Pi 3B+, como definido no capítulo anterior.

5.1 ACURÁCIA

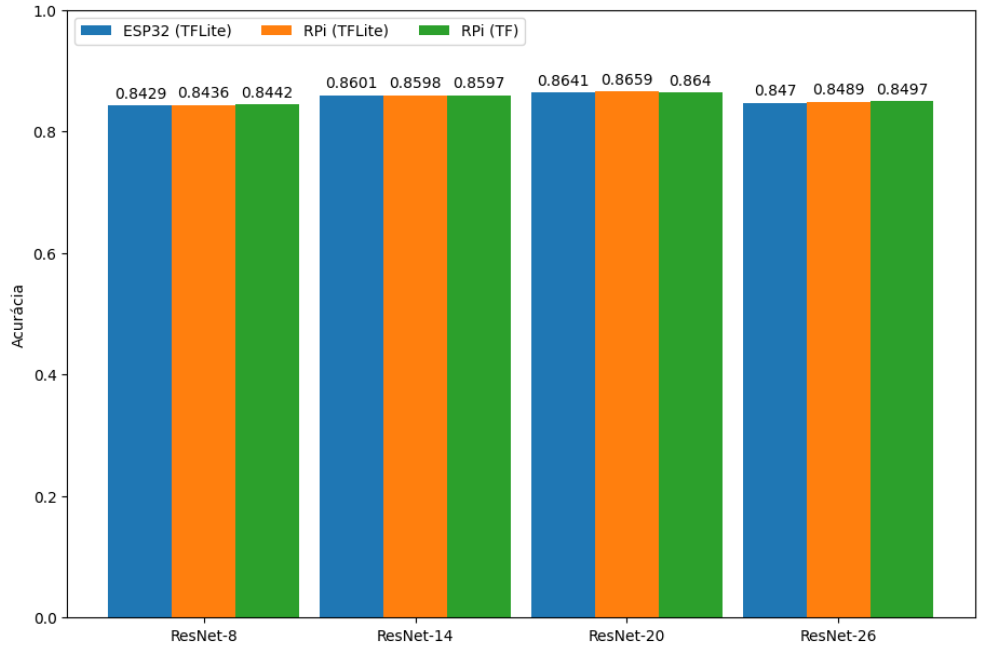


Figura 5.1: Acurácia média para cada modelo neural comprimido (TFLite) e não comprimido (TF) na plataforma Raspberry Pi 3B+ (RPi) e do modelo comprimido no ESP32.

De forma geral, a acurácia média dos modelos foi estritamente parecida. Entretanto, os modelos comprimidos no ESP32 surpreenderam, pois apesar de ser um hardware mais limitado, conseguiu se manter numa boa margem em comparação ao Raspberry Pi, mesmo se comparado com a versão não comprimida que foi executada apenas no Raspberry Pi. A melhor acurácia foi de 0,8659 com a plataforma Raspberry Pi com a ResNet-20 comprimida.

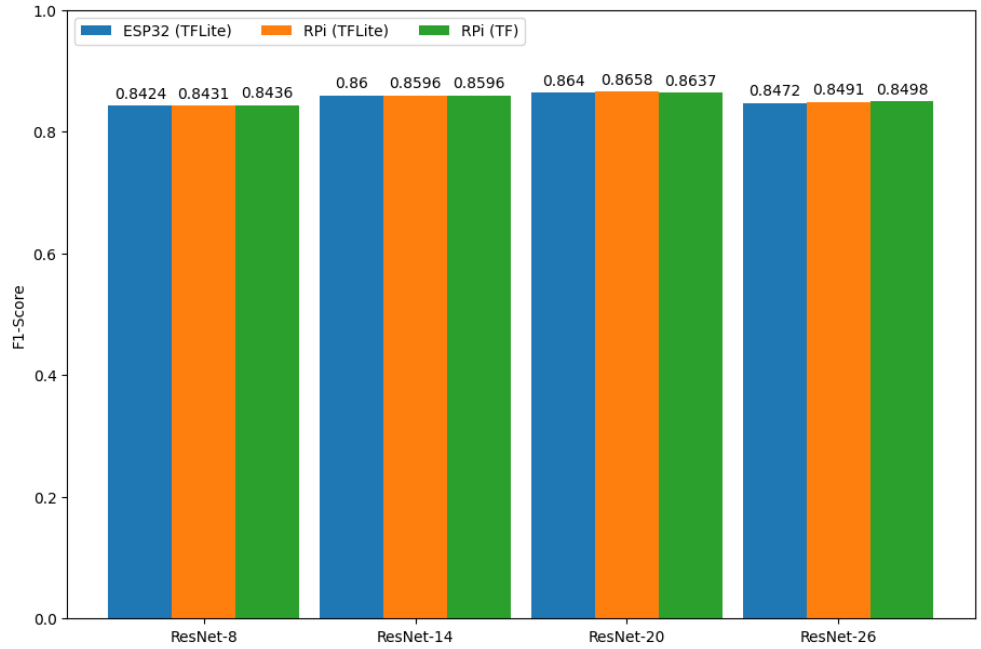


Figura 5.2: F1-Score médio para cada modelo neural comprimido (TFLite) e não comprimido (TF) na plataforma Raspberry Pi 3B+ (RPi) e do modelo comprimido no ESP32.

No que tange as medidas F1-Score, houve uma estabilidade e valores bem próximos em relação as respectivas acurácias mostradas em 5.1. A melhor medida F1-Score foi de 0,8658 na plataforma Raspberry Pi com a ResNet-20 comprimida.

Portanto, considerando os resultados anteriores de acurácia e F1-Score, se evidencia o bom trabalho do método de quantização pós-treinamento para reduzir a acurácia o mínimo possível e um desempenho equilibrado na classificação de cada uma das 10 classes do *dataset*.

5.2 TEMPO DE EXECUÇÃO

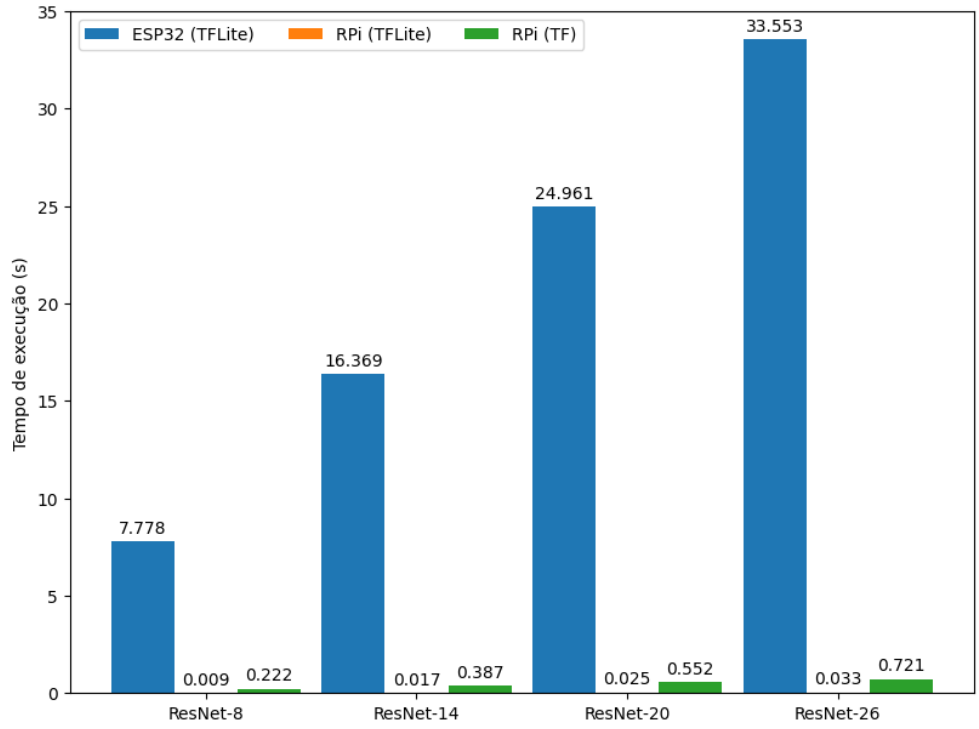


Figura 5.3: Tempo de execução médio por inferência para cada modelo neural comprimido (TFLite) e não comprimido (TF) na plataforma Raspberry Pi 3B+ (RPi) e do modelo comprimido no ESP32.

Comparando as plataformas ESP32 e Raspberry Pi, houve uma grande disparidade no tempo de execução médio. Ao passo que, analisando apenas a plataforma Raspberry Pi, o tempo de execução teve uma diferença considerável entre o modelo comprimido e não comprimido. O melhor tempo de execução foi de 0,009 segundos na plataforma Raspberry Pi com a ResNet-8 comprimida.

Portanto, considerando o resultado anterior de tempo de execução, a plataforma ESP32 teve um comportamento esperado devido aos recursos de processamento e memória limitados, sendo que a plataforma Raspberry Pi se sobressai neste quesito, proporcionando assim valores de tempo menores. Além disso, para a plataforma Raspberry Pi, o modelo comprimido teve um tempo de execução menor do que o não comprimido, sendo um resultado esperado devido a simplificação de processamento do modelo comprimido proporcionado pelo método de quantização.

5.3 POTÊNCIA

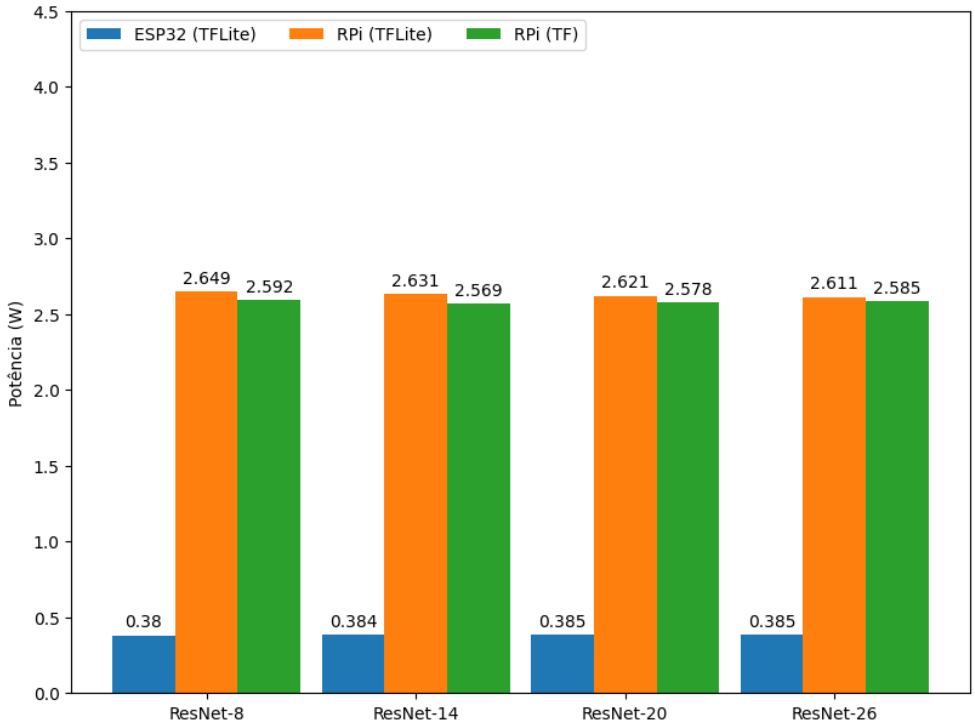


Figura 5.4: Potência média gasta para cada modelo neural comprimido (TFLite) e não comprimido (TF) na plataforma Raspberry Pi 3B+ (RPi) e do modelo comprimido no ESP32.

A primeira vista, a plataforma ESP32 teve uma potência gasta muito menor comparada a plataforma Raspberry Pi. Além disso, para a plataforma Raspberry Pi, houve uma potência gasta maior para os modelos comprimidos. A melhor potência média gasta foi de 0,38 W na plataforma ESP32 com a ResNet-8 comprimida.

Portanto, considerando os resultados anteriores de potência gasta, a plataforma ESP32 teve um resultado esperado devido aos recursos de processamento e memória limitados e também por ser um computador projetado com o objetivo de ter um baixo consumo de energia. Ao passo que, a plataforma Raspberry Pi teve um resultado mais elevado que também já era esperado, devido a uma maior capacidade de processamento e memória. Além disso, uma das razões para o Raspberry Pi ter uma potência gasta maior ao executar modelos comprimidos está relacionada à necessidade de alocar estruturas adicionais, como o interpretador de modelos do TensorFlow Lite, exigido para a execução desses modelos.

Entretanto, vale ressaltar que o consumo de energia total para classificar as 10.000 imagens do conjunto de teste é muito maior para o ESP32 do que para o Raspberry Pi, pois o tempo de execução médio para cada classificação é bem maior no ESP32.

Plataforma	**Consumo médio de energia (J)**			
	ResNet-8	ResNet-14	ResNet-20	ResNet-26
ESP32 (TFLite)	2,956	6,286	9,61	12,918
RPi (TFLite)	0,024	0,045	0,065	0,086
RPi	0,575	0,994	1,423	1,864

Tabela 5.1: Consumo médio de energia por inferência para cada plataforma.

5.4 CONCLUSÃO

De forma geral, considerando todos os resultados apresentados neste capítulo, a plataforma ESP32 teve resultados de tempo de execução e potência gasta esperados devido as suas limitações de hardware. Além disso, a plataforma ESP32 teve um resultado esperado de acurácia comparado a outra plataforma, provando assim que foi o nível de compressão que impactou o nível de acurácia do modelo neural, e não a limitação do hardware. Ao passo que, para a plataforma Raspberry Pi 3B+, todos os resultados apresentados já eram esperados, pecando apenas na parte de potência gasta comparado ao ESP32. Além disso, o consumo de energia por inferência no Raspberry Pi é melhor do que no ESP32 devido ao seu tempo de execução muito menor. Entretanto, este último fato não invalida o propósito do ESP32 como um dispositivo de baixo consumo, pois quando se compara neste caso o consumo de energia por um certo intervalo de tempo, o ESP32 vai ter um consumo de energia menor do que o Raspberry Pi, devido a sua potência gasta ser menor.

Em relação aos modelos de ResNets, a acurácia geral dos modelos comprimidos foi satisfatória, pois não se distanciou muito do que foi obtido com os modelos originais. O método de quantização pós-treinamento do TensorFlow se mostrou eficiente em comprimir o tamanho dos modelos sem comprometer a acurácia.

No próximo capítulo, será concluído este trabalho considerando o que foi elaborado até aqui e levantado algumas questões para trabalhos futuros.

Plataforma	**ResNet-8**				
	Acurácia	F1-Score	Tempo (s)	Potência (W)	Consumo (J)
ESP32 (TFLite)	0,8429	0,8424	7,778	0,38	2,956
RPi (TFLite)	0,8436	0,8431	0,009	2,649	0,024
RPi	0,8442	0,8436	0,222	2,592	0,575

Tabela 5.2: Resumo dos resultados médios para a ResNet-8 para ambas as plataformas.

Plataforma	**ResNet-14**				
	Acurácia	F1-Score	Tempo (s)	Potência (W)	Consumo (J)
ESP32 (TFLite)	0,8601	0,86	16,369	0,384	6,286
RPi (TFLite)	0,8598	0,8596	0,017	2,631	0,045
RPi	0,8597	0,8596	0,387	2,569	0,994

Tabela 5.3: Resumo dos resultados médios para a ResNet-14 para ambas as plataformas.

Plataforma	**ResNet-20**				
	Acurácia	F1-Score	Tempo (s)	Potência (W)	Consumo (J)
ESP32 (TFLite)	0,8641	0,864	24,961	0,385	9,61
RPi (TFLite)	0,8659	0,8658	0,025	2,621	0,065
RPi	0,864	0,8637	0,552	2,578	1,423

Tabela 5.4: Resumo dos resultados médios para a ResNet-20 para ambas as plataformas.

Plataforma	**ResNet-26**				
	Acurácia	F1-Score	Tempo (s)	Potência (W)	Consumo (J)
ESP32 (TFLite)	0,847	0,8472	33,553	0,385	12,918
RPi (TFLite)	0,8489	0,8491	0,033	2,611	0,086
RPi	0,8497	0,8498	0,721	2,585	1,864

Tabela 5.5: Resumo dos resultados médios para a ResNet-26 para ambas as plataformas.

6 CONCLUSÃO E TRABALHOS FUTUROS

Em suma, o trabalho se propôs a realizar um experimento de desempenho e eficiência entre duas plataformas embarcadas populares para realização de uma atividade clássica e prática de ML: classificação de imagens. Ambas as plataformas possuem suas vantagens e desvantagens, enquanto o ESP32 é um hardware de mais baixo custo e baixa potência, o Raspberry Pi 3B+ é mais caro e tem uma potência gasta maior, mas possui uma maior capacidade computacional que proporciona tempos de execução bem menores e um menor consumo de energia por inferência. No geral, ambas as plataformas mostraram seus pontos positivos e negativos para poderem ser utilizadas em aplicações reais que podem possuir critérios importantes de escolha, como custo financeiro, desempenho e eficiência energética.

Em particular para problemas reais de ML, existe um variado espectro de possibilidades, principalmente no que diz respeito a carga de processamento do modelo neural que pode ser esporádica ou contínua, como visto em Baller et al. (2021). Em cenários de computação esporádica, em que as imagens são classificadas de tempos em tempos, o ESP32 pode ser viável, pois se o tempo de execução estiver condizente com a janela de tempo do problema, o ESP32 se torna uma solução muito barata com baixo consumo de energia. Ao passo que o Raspberry Pi 3B+ se sairia melhor em cenários de computação sequencial, em que várias imagens são enviadas em tempo real com critérios rigorosos de tempo de execução para serem processadas rapidamente.

No geral, os principais desafios deste trabalho foram embarcar os modelos neurais no ESP32, pois não existe muita documentação a respeito, e também o tempo gasto para mensurar os dados nessa plataforma devido a sua baixa capacidade de processamento. Ao passo que no Raspberry Pi 3B+ já existia bastante documentação, e o tempo gasto foi muito menor.

Para concluir, esse trabalho visou contribuir para a área de ML em sistemas embarcados, a qual abre um horizonte de novas possibilidades que podem ajudar a democratizar e reduzir custos no desenvolvimento de tecnologias com inteligência artificial. Considerando o presente trabalho, poderia ser avançados estudos futuros incluindo mais plataformas embarcadas, uma vez que ao longo dos anos os sistemas embarcados estão evoluindo rapidamente com novos recursos e arquiteturas com mais desempenho e eficiência. Por outro lado, seria interessante também incluir outros modelos neurais para realizar, por exemplo, uma comparação de qual modelo seria melhor em uma determinada plataforma ou de forma geral.

REFERÊNCIAS

(2019). Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, páginas 1–84.

(2023). *Raspberry Pi 4 Model B Product Brief*. Raspberry Pi Ltd.

(2023). *Raspberry Pi 5 Product Brief*. Raspberry Pi Ltd.

(2023). *Raspberry Pi Pico Product Brief*. Raspberry Pi Ltd.

(2024). *ESP32 Series Datasheet*. Espressif Systems. Version 4.5.

(2024). *RP2040 Datasheet A microcontroller by Raspberry Pi*. Raspberry Pi Ltd. Version 2.3.

Baller, S. P., Jindal, A., Chadha, M. e Gerndt, M. (2021). Deepedgebench: Benchmarking deep neural networks on edge devices.

Banbury, C., Reddi, V. J., Torelli, P., Holleman, J., Jeffries, N., Kiraly, C., Montino, P., Kanter, D., Ahmed, S., Pau, D., Thakker, U., Torrini, A., Warden, P., Cordaro, J., Guglielmo, G. D., Duarte, J., Gibellini, S., Parekh, V., Tran, H., Tran, N., Wenxu, N. e Xuesong, X. (2021a). Mlperf tiny benchmark.

Banbury, C., Zhou, C., Fedorov, I., Navarro, R. M., Thakker, U., Gope, D., Reddi, V. J., Mattina, M. e Whatmough, P. N. (2021b). Micronets: Neural network architectures for deploying tinyml applications on commodity microcontrollers.

Banbury, C. R., Reddi, V. J., Lam, M., Fu, W., Fazel, A., Holleman, J., Huang, X., Hurtado, R., Kanter, D., Lokhmotov, A., Patterson, D., Pau, D., sun Seo, J., Sieracki, J., Thakker, U., Verhelst, M. e Yadav, P. (2021c). Benchmarking tinyml systems: Challenges and direction.

Bay, H., Tuytelaars, T. e Gool, L. V. (2006). Surf: Speeded up robust features. Em *European Conference on Computer Vision*.

Corporation, N. (2014). Nvidia unveils first mobile supercomputer for embedded systems. `https://nvidianews.nvidia.com/news/nvidia-unveils-first-mobile-supercomputer-for-embedded-systems`. Acessado em 30/11/2024.

Corporation, N. (2015). Tiny nvidia supercomputer to bring artificial intelligence to new generation of autonomous robots and drones. `https://nvidianews.nvidia.com/news/tiny-nvidia-supercomputer-to-bring-artificial-intelligence-to-new-generation-of-autonomous-robots-and-drones`. Acessado em 30/11/2024.

Corporation, N. (2017). Nvidia jetson tx2 enables ai at the edge. `https://nvidianews.nvidia.com/news/nvidia-jetson-tx2-enables-ai-at-the-edge`. Acessado em 30/11/2024.

Corporation, N. (2019a). Nvidia announces jetson nano: $99 tiny, yet mighty nvidia cuda-x ai computer that runs all ai models. `https://nvidianews.nvidia.com/news/nvidia-announces-jetson-nano-99-tiny-yet-mighty-nvidia-cuda-x-ai-computer-that-runs-all-ai-models`. Acessado em 30/11/2024.

Corporation, N. (2019b). Nvidia announces jetson xavier nx, world's smallest supercomputer for ai at the edge. https://nvidianews.nvidia.com/news/nvidia-announces-jetson-xavier-nx-worlds-smallest-supercomputer-for-ai-at-the-edge. Acessado em 30/11/2024.

Corporation, N. (2023). Nvidia in brief. https://media.iprsoftware.com/219/files/202311/corporate-nvidia-in-brief-pdf-december-3056300-r2-2.pdf. Acessado em 30/11/2024.

Corporation, N. (2024a). Nvidia jetson nano. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/product-development/. Acessado em 30/11/2024.

Corporation, N. (2024b). Nvidia jetson orin. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/. Acessado em 30/11/2024.

David, R., Duke, J., Jain, A., Reddi, V. J., Jeffries, N., Li, J., Kreeger, N., Nappier, I., Natraj, M., Regev, S., Rhodes, R., Wang, T. e Warden, P. (2021). Tensorflow lite micro: Embedded machine learning on tinyml systems.

Electronics, M. (2024a). Esp32-devkitc. https://www.mouser.com/ProductDetail/Espressif-Systems/ESP32-DevKitC?qs=chTDxNqvsyn3pn4VyZwnyQ%3D%3D. Acessado em 24/04/2024.

Electronics, M. (2024b). Esp32-devkitm-1. https://www.mouser.com/ProductDetail/Espressif-Systems/ESP32-DEVKITM-1?qs=/ha2pyFaduifTIKK1pCXCokFgM3A2OdCZmwfsbWhrm2CgxD3Q2kHAQ%3D%3D. Acessado em 24/04/2024.

Electronics, S. (2024c). Nvidia jetson agx orin 64gb developer kit. https://www.sparkfun.com/products/22099. Acessado em 30/11/2024.

Electronics, S. (2024d). Nvidia jetson nano developer kit (v3). https://www.sparkfun.com/products/16271. Acessado em 30/11/2024.

Fedorov, I., Adams, R. P., Mattina, M. e Whatmough, P. N. (2019). Sparse: Sparse architecture search for cnns on resource-constrained microcontrollers.

Gonzalez, R. e Woods, R. (2017). *Digital Image Processing Global Edition*. Pearson Deutschland.

Gopalakrishna, M. (2018). Now available: Nvidia jetson agx xavier module for next-gen autonomous machines. https://blogs.nvidia.com/blog/nvidia-jetson-agx-xavier-module-now-available/. Acessado em 30/11/2024.

Han, S., Mao, H. e Dally, W. J. (2016). Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding.

Han, S., Pool, J., Tran, J. e Dally, W. J. (2015). Learning both weights and connections for efficient neural networks.

He, K., Zhang, X., Ren, S. e Sun, J. (2015). Deep residual learning for image recognition.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. e Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J. e Keutzer, K. (2016). Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H. e Kalenichenko, D. (2017). Quantization and training of neural networks for efficient integer-arithmetic-only inference.

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images. Dissertação de Mestrado, University of Toronto, Toronto - Ontario.

Krizhevsky, A. (2024). CIFAR-10 and CIFAR-100 datasets. `https://www.cs.toronto.edu/~kriz/cifar.html`. Acessado em 13/06/2024.

Krizhevsky, A., Sutskever, I. e Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Em Pereira, F., Burges, C., Bottou, L. e Weinberger, K., editores, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

Lazarevich, I., Grimaldi, M., Kumar, R., Mitra, S., Khan, S. e Sah, S. (2023). Yolobench: Benchmarking efficient object detectors on embedded systems.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110.

Ltd, R. P. (2024a). Raspberry pi 3 model b+. `https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/`. Acessado em 26/04/2024.

Ltd, R. P. (2024b). Raspberry pi products. `https://www.raspberrypi.com/products/`. Acessado em 09/04/2024.

Molchanov, P., Tyree, S., Karras, T., Aila, T. e Kautz, J. (2017). Pruning convolutional neural networks for resource efficient inference.

Novac, P.-E., Hacene, G. B., Pegatoquet, A., Miramond, B. e Gripon, V. (2021). Quantization and deployment of deep neural networks on microcontrollers. *Sensors*, 21(9):2984.

Pashine, S., Dixit, R. e Kushwah, R. (2021). Handwritten digit recognition using machine and deep learning algorithms. *CoRR*, abs/2106.12614.

Razzak, M. I., Naz, S. e Zaib, A. (2017). Deep learning for medical image processing: Overview, challenges and future. *CoRR*, abs/1704.06825.

Reddi, V. J., Plancher, B., Kennedy, S., Moroney, L., Warden, P., Agarwal, A., Banbury, C., Banzi, M., Bennett, M., Brown, B., Chitlangia, S., Ghosal, R., Grafman, S., Jaeger, R., Krishnan, S., Lam, M., Leiker, D., Mann, C., Mazumder, M., Pajak, D., Ramaprasad, D., Smith, J. E., Stewart, M. e Tingley, D. (2021). Widening access to applied machine learning with tinyml.

Saha, S. S., Sandha, S. S. e Srivastava, M. (2022). Machine learning for microcontroller-class hardware: A review. *IEEE Sensors Journal*, 22(22):21362–21390.

Scholkopf, B. e Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA.

Shivashankar, T. B. (2022). Nvidia jetson agx orin 32gb production modules now available; partner ecosystem appliances and servers arrive. `https://blogs.nvidia.com/blog/nvidia-jetson-agx-orin-32gb-production-modules/`. Acessado em 30/11/2024.

Simonyan, K. e Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition.

Systems, E. (2024). Esp32 series devkits. `https://espressif.com/en/products/devkits?id=ESP32`. Acessado em 08/04/2024.

tinyML Foundation (2023). About tinyml foundation. `https://www.tinyml.org/about`. Acessado em 05/07/2023.