Análise da Execução de Algoritmos de Aprendizado de Máquina em Dispositivos Embarcados

Lucas M. Alf, Renato B. Hoffmann, Caetano Müller, Dalvan Griebler

[1] Escola Politécnica, Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre – RS – Brasil

(lucas.alf,renato.hoffmann,caetano.muller)@edu.pucrs.br,
dalvan.griebler@pucrs.br

Resumo. *Os avanços na área de IoT motivam a utilização de algoritmos de aprendizado de máquina em dispositivos embarcados. Entretanto, esses algoritmos exigem uma quantidade considerável de recursos computacionais. O objetivo deste trabalho consistiu em analisar algoritmos de aprendizado de máquina em dispositivos embarcados utilizando paralelismo em CPU e GPU com o intuito de compreender quais características de hardware e software desempenham melhor em relação ao consumo energético, inferências por segundo e acurácia. Foram avaliados três modelos de Convolutional Neural Network, bem como algoritmos tradicionais e redes neurais de classificação e regressão. Os experimentos demonstraram que o PyTorch obteve o melhor desempenho nos modelos de CNN e nas redes neurais de classificação e regressão usando GPU, enquanto o Keras obteve um melhor desempenho ao utilizar somente CPU.*

1. Introdução

Dispositivos embarcados são projetados para desempenhar tarefas especificas sem interação direta ou continuada com o usuário. Devido a variedade de tecnologias no mercado, eles podem podem ser encontrados em diferentes formatos e tamanhos, porém, em geral, são dispositivos pequenos e com uma quantidade de recursos limitada [Lacamera 2018]. Mesmo assim, muitos são equipados com múltiplos núcleos de processamento e aceleradores sofisticados.

Dentre os formatos de dispositivos embarcados, pode-se destacar o *Single Board Computer* (SBC) que pode ser definido como um computador completo composto por um único PBC (*Printed Cirtuit Board*). Ao contrário de computadores tradicionais, o *hardware* base de um SBC não pode ser alterado, visto que todos os componentes (CPU, RAM, GPU e portas de interface) são integrados a própria placa de circuitos. Alguns exemplos populares de SBCs podem ser mencionados como a linha Asus Tinkerboard, NVIDIA Jetson e Raspberry Pi [Pajankar 2020].

A aplicação de algoritmos de aprendizado de máquina neste tipo de dispositivo embarcado tem sido impulsionada pelos avanços na área de IoT (*Internet of Things*), abrangendo diversos casos de uso. Alguns exemplos são a classificação e reconhecimento de eventos sonoros [Karunaratna and Maduranga 2021], aplicação de visão computacional para o monitoramento de plantações [Savvidis and Papakostas 2021], detecção de pedestres em imagens aéreas [de Oliveira and Wehrmeister 2018], e aplicações voltadas à área da saúde, como modelos para detecção de arritmia cardíaca, preparados para serem executados em sistemas embarcados com recursos limitados [Mohebbanaaz et al. 2022].

Tradicionalmente, a aplicação de técnicas de aprendizado de máquina exige uma quantidade elevada de recursos computacionais, mesmo que a etapa de treinamento possa ser realizada por um sistema externo, e somente a inferência seja executada no dispositivo embarcado, ainda existem necessidades especiais referentes a limitação de tamanho dos modelos. [Nikouei 2018]. Em particular, a execução de inferências envolve computações de multiplicação de matrizes e também capacidade de memória e armazenamento suficiente para os modelos e pesos. Esses fatores tornam a execução de algoritmos inteligentes em sistemas embarcados ou de pequeno porte uma tarefa desafiadora, uma vez que geralmente estes sistemas são baseados em processadores de baixo consumo energético e possuem recursos limitados de memória, armazenamento e bateria.

O objetivo deste trabalho foi analisar a viabilidade de diversos algoritmos de aprendizado de máquina em dispositivos com recursos limitados, utilizando paralelismo em CPU e GPU, com o intuito de compreender quais características de hardware e software fornecem melhor desempenho. Sendo assim, este trabalho traz como contribuição científica a análise e explicação dos resultados de desempenho e consumo energético de diversos algoritmos de aprendizado de máquina, sobre os dispositivos NVIDIA Jetson Nano e Odroid N2, fornecendo uma visão geral sobre a execução de algoritmos de rede neural e algoritmos tradicionais de classificação, regressão e agrupamento de dados, usando as bibliotecas scikit-learn, PyTorch e Keras.

Em relação a organização deste trabalho, a introdução a pesquisa foi apresentada na Seção 1. A Seção 2 fornece o embasamento teórico e conceitual considerado necessário para este trabalho. Na Seção 3, foi descrita a metodologia utilizada na pesquisa. Por fim, na Seção 4 foram apresentadas as conclusões do estudo.

2. Trabalhos Relacionados

O trabalho de [Süzen et al. 2020] realizou uma análise comparativa do desempenho entre diferentes dispositivos *Single Board Computer* para execução de *Convolutional Neural Network* (CNN). Para a execução do experimento, foi desenvolvido um modelo de rede neural baseado em 2D-CNN utilizando a biblioteca cuDNN[1]. Foram realizados testes nos dispositivos NVIDIA Jetson Nano, NVIDIA Jetson TX2 e Raspberry PI 4. Em seu experimento foram coletadas as métricas de acurácia (%), tempo de execução (segundos), uso de memória (GB), potência elétrica da CPU (W) e potência elétrica da GPU (W). Foi constatado que o dispositivo Jetson TX2 possui o maior consumo de energético devido ao seu *hardware* mais potente. Os dispositivos que possuem GPU (NVIDIA Jetson TX2 e NVIDIA Jetson Nano) apresentaram os menores tempos de execução. Erros de memória foram detectados tanto no NVIDIA Jetson Nano quanto no Raspberry PI quando o tamanho do *dataset* era maior que 20K. Foi percebido que o consumo energético tende a ser mais estável com *datasets* maiores no NVIDIA Jetson TX2.

A pesquisa conduzida por [Lee et al. 2021] teve como objetivo realizar uma análise comparativa do desempenho de diferentes modelos de detecção de objetos em vídeo sobre dispositivos embarcados. Para a execução do experimento foi desenvolvido o modelo FastAdapt, que consiste em uma variante do modelo ApproxDet, e comparado com o desempenho dos modelos FRCNN+, YOLO+, Faster R-CNN, YOLOv3, SELSA, MEGA

[1]https://developer.nvidia.com/cudnn

e REPP. Foram considerados diferentes níveis de restrição de recursos, como GPU e consumo de energia. Os dispositivos testados foram NVIDIA Jetson TX2, NVIDIA Jetson Xavier NX e NVIDIA Jetson AGX Xavier. Os resultados revelaram que o modelo proposto obteve a melhor relação entre acurácia e latência em todos os três dispositivos, independentemente do nível de restrição de recursos.

O trabalho de [Magalhães et al. 2023] realiza uma análise comparativa de diferentes plataformas heterogêneas para detecção de objetos em tempo real. Os dispositivos utilizados no experimento foram separados em 3 categorias, dispositivos com GPU: NVIDIA Jetson Nano e NVIDIA Jetson TX2; Dispositivos com TPU: Coral Dev Board; E dispositivos FPGA: AMD-Xilinx ZCU104 Development Board e AMD-Xilinx Kria KV206 Starter Kit. Foi constatado que os dispositivos com GPU apresentaram o pior desempenho, entre 3 e 5 FPS (*frames per second*), enquanto os dispositivos FPGA apresentaram entre 14 e 25 FPS. Porém no quesito de consumo de energia, os dispositivos com GPU e TPU apresentaram os melhores resultados, entre 1W e 5W, enquanto os dispositivos FPGA utilizaram entre 15W e 20W. Não houve diferenças relevantes de precisão do modelo entre os dispositivos.

O trabalho de [Baller et al. 2021] realiza uma analise comparativa de tempo por inferência, consumo de energia e acurácia dos modelos MobileNet V2 e MobileNet V1 (ambos também testados em formato TFLite e com quantização para 8 bit) sobre os dispositivos Nvidia Jetson Nano, Google Coral Dev Board, Asus Tinker Edge R, Raspberry Pi 4 e Arduino Nano 33 BLE. Para a execução dos experimentos foram utilizadas as ferramentas Tensorflow, TensorRT, Tensorflow Lite e RKNN-Toolkit. Os dispositivos com aceleradores foram testados tanto com o acelerador habilitado quanto desabilitado. Como resultado, o Google Coral Dev Board apresentou os melhores resultados para modelos que podem ser convertidos para o formato TFLite. Em relação a eficiência energética, o Arduino Nano 33 BLE apresentou o menor consumo de energia, porém também apresentou uma baixa taxa de inferências por segundo. Já ao desconsiderar o consumo de energia, o Nvidia Jetson Nano apresentou a melhor relação de tempo por inferência.

Em comparação aos trabalhos relacionados, este trabalho possui como objetivo realizar uma análise mais abrangente de diversos algoritmos tradicionais de aprendizado de máquina e algoritmos baseados em redes neurais, implementados sobre diferentes bibliotecas e diferentes dispositivos, avaliando as métricas de desempenho, consumo de energia e acurácia, com o intuito de compreender quais características de hardware e quais ferramentas desempenham melhor.

3. Análise de Desempenho

Nesta Seção, foi apresentado o método empregado para a realização dos experimentos, os dispositivos utilizados, a técnica adotada para coletar métricas de consumo energético e os algoritmos de aprendizado de máquina e aprendizado profundo aplicados.

3.1. Metodologia

Para a realização dos experimentos, os algoritmos de aprendizado de máquina foram organizados em duas categorias: algoritmos tradicionais e modelos de aprendizado profundo. Dentre os modelos de aprendizado profundo foram selecionados os modelos Mobilenet V2, EfficientNet B0 e DenseNet-121, que consistem em modelos de *Convolutional Neural Network* (CNN), utilizados para classificação e reconhecimento de objetos em imagens.

Estes modelos foram selecionados com base em dois critérios: primeiro, eles possuem um tamanho reduzido, tornando-os adequados para execução em dispositivos com recursos limitados; segundo, foram escolhidos porque possuem implementações em Python pré-treinadas disponibilizadas pelas bibliotecas PyTorch e Keras, este último executado sobre a plataforma TensorFlow, o que permite realizar um comparativo entre a implementação realizada entre as duas ferramentas.

Já referente aos algoritmos de aprendizado de máquina tradicionais, foi utilizada a biblioteca *scikit-learn* para implementar os algoritmos de classificação: *Gaussian Naive Bayes*, *K-Nearest Neighbors*, *Logistic Regression*, *Random Forest* e *Support Vector Classification*; algoritmos de regressão: *Linear Regression*, *Stochastic Gradient Descent*, e *Support Vector Regression*; e o algoritmo *K-means* para agrupamento de dados.

Para fins de comparação com os algoritmos tradicionais de aprendizado de máquina, também foram criadas redes neurais de classificação e regressão utilizando as bibliotecas PyTorch e scikit-learn. As redes foram criadas usando a função de ativação ReLU com duas camadas ocultas, sendo que a primeira camada possui 24 nodos, a segunda 16 nodos e 10 nodos para a camada de saída usando a função *Softmax*. O número de nodos da camada de saída representa exatamente o número de possíveis classes do resultado final. Esta configuração se mostrou eficiente o suficiente tanto em questão de consumo de recursos computacionais quanto em tempo de execução, apresentando os melhores resultados de treino e teste.

Os modelos de aprendizado profundo selecionados foram avaliados utilizando o *dataset "Felidae | Cat species recognition"*[2], o qual é composto por um banco de imagens aberto contendo 243 fotos de felinos, organizados em 5 espécies distintas. Já os algoritmos tradicionais de aprendizado de máquina foram executados sobre o *dataset "Red Wine Quality"*[3], que consiste em conjunto de 1599 registros de 12 diferentes características de vinhos. A escolha destes *datasets* foi realizada com base em seu tamanho compacto, o que possibilita a realização do experimento em dispositivos embarcados em um tempo razoável, visto que estes dispositivos possuem um poder computacional limitado.

O experimento foi executado sobre os dispositivos NVIDIA Jetson Nano e Odroid N2. As métricas coletadas foram inferências por segundo, consumo de energia e acurácia. As métricas de consumo de energia foram coletadas por meio de um dispositivo externo voltímetro/amperímetro USB UM25C[4], que fica situado entre a tomada e a entrada de energia do dispositivo. Foi utilizando do gestor de pacotes Conda para criar ambientes virtuais e manter a mesma versão do Python 3.6 em todos os dispositivos. Foi desenvolvido um *script* para automatizar a execução 5 rodadas de cada modelo, e mais uma rodada inicial de aquecimento, ao final os arquivos gerados pelas 5 rodadas de execução são combinados e a média dos resultados é calculada.

3.2. Características dos Algoritmos

Na área de algoritmos de aprendizado de máquina tradicionais, a biblioteca *scikit-learn* tem sido amplamente utilizada para implementar algoritmos de classificação, regressão e agrupamento de dados. Para todos algoritmos abaixo as complexidades apresentadas são

[2]https://www.kaggle.com/datasets/juliencalenge/felidae-tiger-lion-cheetah-leopard-puma

[3]https://www.kaggle.com/datasets/uciml/red-wine-quality-cortez-et-al-2009

[4]https://joy-it.net/en/products/JT-UM25C

apenas relacionados à complexidade de predição/inferência ou seja não é levado em consideração a complexidade do treinamento dos modelos. Em relação aos hiperparâmetros, em todos os algoritmos são utilizados os padrões da biblioteca *scikit-learn*.

1. ***Gaussian Naive Bayes***: utiliza de multiplicação entre elementos de vetores do conjunto de dados, tornando-o uma escolha eficiente em tempo e consumo de recursos computacionais em diversos problemas de classificação com complexidade de $O(m \cdot k)$ em que m é a dimensão do dado e k é o número de *datapoints*.
2. ***K-Nearest Neighbors* (KNN)**: realiza cálculos de distância entre o dado inferido e todos os dados de treinamento para determinar a classe ou rótulo mais próximo. Embora seja um algoritmo simples e intuitivo, pode ser computacionalmente exigente, especialmente em conjuntos de dados grandes. KNN possui complexidade de $O(k \cdot n)$ onde n é o número de *datapoints*.
3. **Random Forest**: emprega várias árvores de decisão para chegar a um resultado. A etapa mais exigente é o processo de *Bagging*, que envolve a criação de *subsets* do conjunto de treinamento e a geração de múltiplas árvores de decisão. O resultado final é obtido por meio de uma votação entre as saídas das árvores. Este algoritmo apresenta uma complexidade maior de ser calculada mas pode ser representada por $O(T \cdot k \cdot m)$ em que T é o número de árvores, k é o numero de *datapoints* e m é a dimensão do dado.
4. ***Support Vector Classification* (SVC)**: realiza principalmente multiplicações entre vetores e matrizes para construir o hiperplano que melhor separa as classes no espaço de características. A complexidade deste algoritmo é de $O(k \cdot n \cdot m)$, onde k é o número de *datapoints*, m é a dimensão do dado e n é o número de *datapoints* do *dataset* de treino.
5. ***Logistic Regression***: diferente da regressão linear, este algoritmo é utilizado para problemas de classificação binária, onde se busca encontrar a probabilidade de uma instância pertencer a uma classe específica. O algoritmo possui complexidade de $O(k \cdot m)$, onde k é número de dados a serem classificados e m representa a dimensão dos dados.
6. ***Linear Regression***: consiste em um método de regressão clássico que busca ajustar uma linha reta sobre dados de treinamento para realizar previsões. O algoritmo possui complexidade de $O(k \cdot m)$, onde k é número de dados a serem classificados e m representa a dimensão dos dados.
7. ***Stochastic Gradient Descent* (SGD)**: realiza múltiplas operações matemáticas e produtos escalares para calcular uma convergência. O SGD não está necessariamente em nenhuma família de algoritmos de aprendizado de máquina, trata-se apenas de uma técnica de otimização, ou seja, é usado para treinar modelos. O algoritmo possui complexidade de $O(k \cdot m)$, na qual k consiste no número de dados a serem classificados e m representa a dimensão dos dados.
8. ***Support Vector Regression* (SVR)**: é similar ao SVC, este algoritmo utiliza multiplicação entre vetores e matrizes para realizar regressão e encontrar uma função de aproximação que melhor se ajuste aos dados. A complexidade deste algoritmo é de $O(k \cdot n \cdot m)$, k é o numero de *datapoints*, m é a dimensão do dado e n é o número de *datapoints* do *dataset* de treino.
9. ***K-means***: é utilizado para agrupar dados com base na similaridade entre as instâncias. Assim como o KNN, esse algoritmo também envolve o cálculo de distâncias

entre os dados para agrupá-los de acordo com a proximidade. Dado que K-means é um algoritmo de *clustering* não existe a parte de treinamento e predição separada e todo o algoritmo tem a complexidade $O(i \cdot k \cdot n \cdot m)$, em que i representa o número de iterações para convergência, n é o número de *datapoints* e m é a dimensão dos dados.

3.3. Análise com Redes Neurais Convolucionais (CNNs)

As Figuras 1 e 2 apresentam um comparativo de inferências por segundos dos modelos Mobilenet V2, EfficientNet B0 e DenseNet 121, implementados respectivamente sobre as bibliotecas Keras e PyTorch, sobre os dispositivos Odroid N2 e NVIDIA Jetson Nano. Para a realização do comparativo, o dispositivo NVIDIA Jetson Nano foi testado tanto com o suporte a GPU habilitado, quanto desabilitado, permitindo analisar o impacto da GPU sobre os resultados.

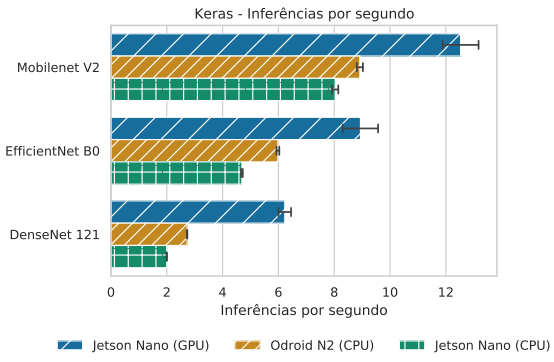

Figura 1. Inferências por Segundo CNN - Implementação Keras

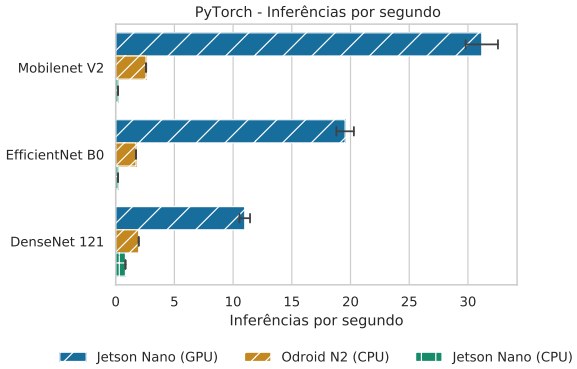

Figura 2. Inferências por Segundo CNN - Implementação PyTorch

Como visível nas Figuras 1 e 2, a implementação em PyTorch do modelo MobileNet V2 apresentou o melhor desempenho, atingindo um valor aproximado de 31 inferências por segundo no dispositivo NVIDIA Jetson Nano com suporte a GPU habilitado. Em geral, é perceptível que os modelos implementados pelo PyTorch possuem um melhor aproveitamento de GPU, enquanto os modelos implementados pelo Keras apresentaram os melhores resultados nos dispositivos que possuem apenas suporte a CPU. Para confirmar essa observação, foi realizado uma pequena análise preliminar com uma GPU NVIDIA RTX 3060, onde o modelo MobileNet V2 atingiu 151 inferências por segundo (487.10% superior ao NVIDA Jetson Nano) no PyTorch, enquanto a implementação do mesmo modelo fornecida pelo Keras atingiu apenas 74 inferências por segundo.

Embora ambas as bibliotecas sejam implementas em C++ e utilizem do NVIDIA cuDNN[5] para aceleração em GPU, existem algumas diferenças fundamentais que podem contribuir para a diferença de desempenho presente nas Figuras 1 e 2. Enquanto o PyTorch fornece um melhor controle em relação a memória, de forma que os dados precisam ser explicitamente movidos entre CPU e GPU antes da execução do modelo, o Keras funciona apenas como uma interface que facilita a utilização dos modelos sobre o TensorFlow, que por sua vez abstrai a movimentação dos dados entre CPU e GPU, e escolhe

[5]https://developer.nvidia.com/cudnn

de forma dinâmica quais dados devem ir para a GPU, e quando serão movidos. Sendo assim, para múltiplas inferências em sequência, a versão PyTorch permite explicitamente realizar menos cópias de dados entre CPU e GPU.

Também é perceptível que o dispositivo NVIDIA Jetson Nano com suporte a GPU desabilitado apresentou os piores resultados, sendo inferior ao Odroid N2. Levando apenas em consideração a CPU, o Odroid tem mais capacidade computacional. Isso se deve ao Odroid N2 possuir um processador baseado na arquitetura big.LITTLE com quatro núcleos baseados no ARM Cortex-A73 @ 1,8Ghz e dois baseados no ARM Cortex-A53 @ 1,9 Ghz, enquanto o NVIDIA Jetson Nano possui apenas um processador de quatro núcleos baseado no ARM Cortex-A57 @ 1,43 Ghz.

Em relação à acurácia, os modelos implementados em PyTorch apresentaram os melhores resultados, com o modelo DenseNet 121 chegando a 90,46% de acurácia, o modelo MobileNet V2 apresentou o segundo melhor resultado, com 88,85%, e o modelo EfficientNet B0 apresentou a pior acurácia dentre os modelos, chegando a 81,48%. Em contraste o modelo que apresentou a melhor acurácia também apresentou o pior desempenho. Dessa forma, o modelo MobileNet V2 demonstrou a melhor relação entre desempenho e acurácia. Entretanto, a acurácia dos modelos não é o foco deste trabalho, mas sim a avaliação do seu desempenho.

A Tabela 1 apresenta as métricas de acurácia do modelo, potência máxima atingida pelos dispositivos em Watss (W), e o consumo de energia ao longo da execução do experimento em microwatt/hora (uWh). Para facilitar a leitura da tabela, os valores de maior potência atingida e o maior consumo de energia foram destacados em vermelho, e os valores de menor potência e menor consumo foram destacados em azul.

Tabela 1. Consumo de energia dos modelos de CNN

Dispositivo	Implementação	Modelo	Potência máxima (W)	Consumo de energia (uWh)	Acurácia (%)
Jetson Nano (CPU)	PyTorch	DenseNet 121	6,52	513,40	90,46
		EfficientNet B0	5,64	1834,60	81,48
		Mobilenet V2	5,66	1853,60	88,85
	Keras	DenseNet 121	5,86	198,80	86,81
		EfficientNet B0	5,06	84,80	75,47
		Mobilenet V2	4,88	58,20	76,82
Jetson Nano (GPU)	PyTorch	DenseNet 121	7,04	91,00	90,46
		EfficientNet B0	5,38	52,40	81,48
		Mobilenet V2	4,94	43,20	88,85
	Keras	DenseNet 121	7,20	84,40	86,81
		EfficientNet B0	5,82	58,20	75,47
		Mobilenet V2	5,20	46,40	76,82
Odroid N2 (CPU)	PyTorch	DenseNet 121	6,47	238,40	90,46
		EfficientNet B0	6,08	248,60	81,48
		Mobilenet V2	6,02	172,60	88,85
	Keras	DenseNet 121	6,07	146,60	86,81
		EfficientNet B0	5,08	65,00	75,47
		Mobilenet V2	4,51	45,40	76,82

Em geral, o dispositivo NVIDIA Jetson Nano com o suporte a GPU habilitado apresentou a maior potência, atingindo um pico de 7,20 W no modelo DenseNet 121 fornecido pelo Keras, ao mesmo tempo que apresentou o menor consumo de energia no modelo MobileNet V2 fornecido pelo PyTorch. Entretanto ao desabilitar o suporte a GPU, o dispositivo NVIDIA Jetson Nano apresentou o maior consumo de energia, chegando a 1853,60 uWh de consumo no modelo MobileNet V2, tendo em vista que este modelo al-

cançou somente 0,20 inferências por segundo, demorando mais de 20 minutos para completar a execução do *dataset*. Desta forma dentre os modelos selecionados, o MobileNet V2 fornecido pelo PyTorch obteve a melhor relação entre inferências por segundo, acurácia e consumo de energia, apresentando os melhores resultados no dispositivo NVIDIA Jetson Nano com suporte a GPU habilitado.

3.4. Análise com Algoritmos Tradicionais

Nesta Seção, foram apresentados os resultados referentes aos algoritmos tradicionais de aprendizado de máquina, organizados em classificação, regressão e agrupamento de dados, na forma de métricas como inferências por segundo, acurácia e consumo energético.

3.4.1. Algoritmos de Classificação

As Figuras 3 e 4 apresentam respectivamente um comparativo de inferências por segundo, e um comparativo de acurácia dos modelos de classificação *Gaussian Naive Bayes*, *K-Nearest Neighbors* (KNN), *Logistic Regression*, *Random Forest* e *Support Vector Classification* (SVC) sobre os dispositivos NVIDIA Jetson Nano e Odroid N2. Todos os algoritmos foram implementados usando a biblioteca scikit-learn, que até o momento da escrita deste artigo somente suporta paralelismo a nível de CPU.

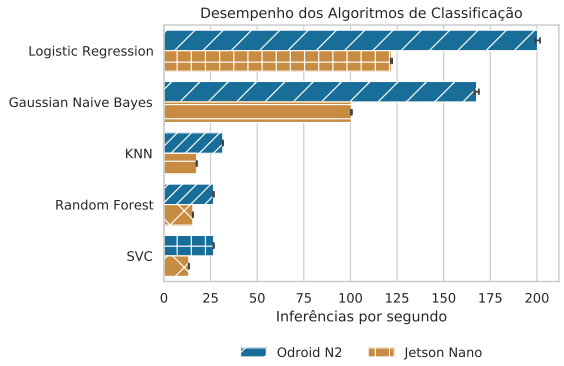

Figura 3. Inferências por Segundo Algoritmos de Classificação

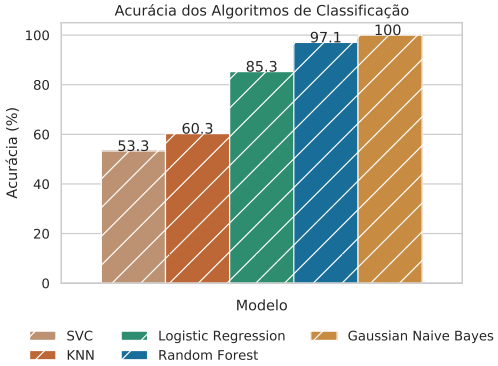

Figura 4. Acurácia dos algoritmos de Classificação

De forma proporcional, o desempenho dos algoritmos foi semelhante em ambos os dispositivos, com o Odroid N2 apresentando os melhores resultados. Isso se deve ao Odroid N2 possuir uma CPU mais robusta que NVIDIA Jetson Nano. Além disto, foi utilizado o sbc-bench[6] para verificar a largura de banda de memória dos dispositivos, onde foi observado que o Odroid N2 possui uma largura de banda de 3884 MB/s, enquanto o NVIDIA Jetson Nano possui 3728 MB/s, portanto as diferenças de desempenho são primariamente pela capacidade da CPU. Dentre os algoritmos o *Logistic Regression* apresentou a maior quantidade de inferências por segundo, porém ao considerar a acurácia, o algoritmo *Gaussian Naive Bayes* obteve a melhor relação entre desempenho e acurácia.

[6]https://github.com/ThomasKaiser/sbc-bench

Para realizar um comparativo entre os algoritmos tradicionais de aprendizado de máquina e modelos baseados em redes neurais, foram criados dois modelos de classificação utilizando o PyTorch e scikit-learn. Para realizar uma comparação mais justa entre as bibliotecas, o suporte a GPU do PyTorch foi desabilitado, sendo assim ambas as bibliotecas possuem apenas paralelismo a nível de CPU. Como resultado o modelo implementado em PyTorch apresentou um desempenho superior ao scikit-learn, conseguindo atingir aproximadamente 3500 inferências por segundo no dispositivo Odroid N2. Em relação a acurácia o modelo implementado em PyTorch também conseguiu o melhor resultado, atingindo 100% de acurácia, enquanto o modelo implementado em scikit-learn atingiu 87,5%. Em relação aos algoritmos tradicionais, o modelo de rede neural desenvolvido em PyTorch teve resultados muito superiores tanto em desempenho quanto em acurácia. Já a rede neural desenvolvida em scikit-learn teve resultados inferiores em ambas métricas.

Este resultado pode ser explicado devido ao scikit-learn ser uma biblioteca voltada a facilidade de uso, que utiliza como base bibliotecas como o NumPy e SciPy, e permite implementar algoritmos de classificação, regressão e agrupamento de dados em poucas linhas de código. Para a criação de redes neurais o scikit-learn fornece duas abstrações, sendo o *MLPClassifier* e *MLPRegressor* que consistem em implementações simples do *Multi Layer Perceptron*. Enquanto o PyTorch consiste em uma biblioteca muito mais robusta, geralmente utilizada para o desenvolvimento de modelos de *Deep Learning*, que possui diversas optimizações tanto para o treinamento quanto para a execução dos modelos, como por exemplo a utilização de grafos computacionais dinâmicos e o *Autograd* que realiza o cálculo de forma automática do melhor gradiente para optimização do modelo.

3.4.2. Algoritmos de Regressão

A Figura 5 apresenta um comparativo de inferências por segundo dos algoritmos de regressão *Linear Regression*, *Stochastic Gradient Descent* (SGD) e *Support Vector Regression* (SVR). Já em comparação a Figura 6 apresenta um comparativo de inferências por segundo dos modelos de rede neural desenvolvidos em PyTorch e scikit-learn. Da mesma forma que nos modelos de classificação, o suporte a GPU foi desabilitado na biblioteca PyTorch, assim ambas as bibliotecas utilizam apenas de paralelismo a nível de CPU.

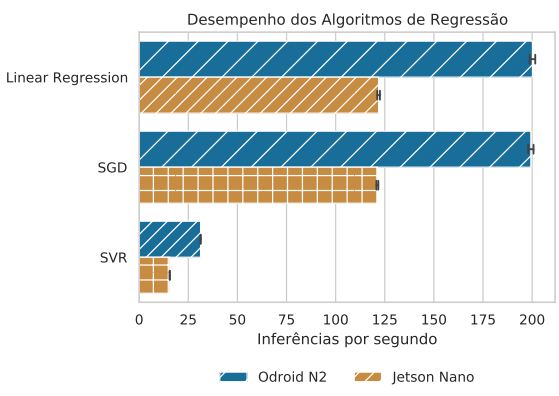

Figura 5. Inferências por Segundo Algoritmos de Regressão

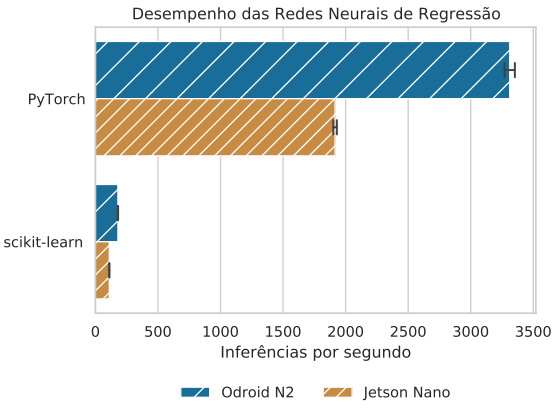

Figura 6. Inferências por Segundo Rede Neural de Regressão

Dentre os algoritmos tradicionais de regressão, *Linear Regression* e *Stochastic Gradient Descent* apresentaram o melhor desempenho, atingindo aproximadamente 200 inferências por segundo. Enquanto dentre os modelos de rede neural, a implementação realizada em PyTorch apresentou o melhor desempenho, chegando a atingir aproximadamente 3300 inferências por segundo no dispositivo Odroid N2.

3.4.3. Algoritmos de Agrupamento de dados

Para o experimento de agrupamento de dados foi utilizado do algoritmo K-means fornecido pela biblioteca scikit-learn. Somente a métrica de inferências por segundo foi analisada, visto que não é possível calcular a acurácia de algoritmos de agrupamento de dados sobre *datasets* que não possuem *labels* bem definidas.

Como resultado o dispositivo Odroid N2 apresentou o melhor desempenho, atingindo aproximadamente 190 inferências por segundo, enquanto o NVIDIA Jetson Nano alcançou um valor aproximado de 120 inferências por segundo. Em relação ao consumo de energia, o Odroid N2 apresentou o melhor resultado consumindo apenas 0,80 uWh, enquanto o NVIDIA Jetson NANO consumiu 1,80 uWh. Uma relação mais detalhada do consumo de energia entre algoritmos tradicionais pode ser encontrada na Seção 3.4.4.

3.4.4. Consumo energético dos algoritmos

A Tabela 2 apresenta as métricas de potências máxima em Watts (W) e o consumo de energia dos dispositivos em microwatt/hora (uWh) ao longo da execução dos algoritmos tradicionais de classificação, regressão e agrupamento de dados, bem com os modelos de rede neural de classificação e regressão desenvolvidos em PyTorch e scikit-learn.

Tabela 2. Consumo de energia dos algoritmos tradicionais

Dispositivo	Caso de uso	Implementação	Modelo	Potência máxima (W)	Consumo de energia (uWh)
Jetson Nano	Classificação	Scikit-learn	Gaussian Naive Bayes	3,02	2,20
		Scikit-learn	K-Nearest Neighbors	3,01	13,80
		Scikit-learn	Logistic Regression	3,03	1,60
		Scikit-learn	Random Forest	3,09	16,00
		Scikit-learn	Support Vector Classification	3,22	19,80
		Scikit-learn	Rede Neural (Classificação)	3,05	1,80
		PyTorch	Rede Neural (Classificação)	1,83	0,20
	Regressão	Scikit-learn	Linear Regression	3,06	1,60
		Scikit-learn	Stochastic Gradient Descent	3,13	1,20
		Scikit-learn	Support Vector Regression	3,26	16,60
		Scikit-learn	Rede Neural (Regressão)	3,09	2,00
		PyTorch	Rede Neural (Regressão)	1,81	0,20
	Agrupamento	Scikit-learn	K-means	3,09	1,80
Odroid N2	Classificação	Scikit-learn	Gaussian Naive Bayes	3,10	0,80
		Scikit-learn	K-Nearest Neighbors	3,04	7,60
		Scikit-learn	Logistic Regression	3,11	0,80
		Scikit-learn	Random Forest	3,12	8,60
		Scikit-learn	Support Vector Classification	3,33	9,80
		Scikit-learn	Rede Neural (Classificação)	3,12	0,60
		PyTorch	Rede Neural (Classificação)	2,16	0,10
	Regressão	Scikit-learn	Linear Regression	3,08	1,00
		Scikit-learn	Stochastic Gradient Descent	2,83	0,60
		Scikit-learn	Support Vector Regression	3,32	8,00
		Scikit-learn	Rede Neural (Regressão)	3,14	1,20
		PyTorch	Rede Neural (Regressão)	2,48	0,10
	Agrupamento	Scikit-learn	K-means	3,16	0,80

Em geral ambos os dispositivos atingiram potências entre 1,81 e 3,33 Watss. Já em relação ao consumo de energia, o Odroid N2 apresentou os melhores resultados, com um consumo máximo de 9,80 uWh no algoritmo *Support Vector Classification*, enquanto o dispositivo NVIDIA Jetson Nano chegou a atingir um consumo máximo de 19,80 uWh no mesmo algoritmo, tendo em vista que este obteve somente 13,36 inferências por segundo, demorando aproximadamente 22 segundos para completar a execução do *dataset*.

3.5. Resumo dos Resultados

Os resultados mostraram que dentre os modelos de CNN, o MobileNet V2 apresentou o melhor desempenho, alcançando aproximadamente 30 inferências por segundo no NVIDIA Jetson Nano. Em geral, é perceptível que os modelos implementados pelo PyTorch possuem um melhor aproveitamento de GPU, enquanto os modelos implementados pelo Keras apresentaram os melhores resultados nos dispositivos que possuem apenas suporte a CPU. Além do mais, o Odroid N2 desempenhou melhor nos experimentos que apenas utilizavam CPU, comprovando que a utilização dos núcleos *little* da arquitetura BIG.little é vantajosa para desempenho e eficiência energética.

Como esperado, as cargas de trabalho das redes neurais obtiveram melhores resultados na GPU, visto que repetidas multiplicações de matrizes são facilmente mapeadas para este acelerador. Inclusive, a menor quantidade de registradores de alta precisão da GPU não é um problema pois a acurácia foi equivalente à CPU. A potência máxima da GPU foi relativamente maior do que na CPU, porém o consumo de energia foi menor.

Considerando o consumo de energia, o modelo MobileNet V2 apresentou a melhor relação entre desempenho, acurácia e consumo de energia, obtendo os melhores resultados no dispositivo NVIDIA Jetson Nano com suporte a GPU habilitado. Em relação aos algoritmos de rede neural para classificação e regressão, o modelo implementado em PyTorch obteve um desempenho superior ao scikit-learn, tanto em desempenho quanto em acurácia.

4. Conclusões

Este trabalho teve como objetivo analisar a execução de diversos algoritmos de aprendizado de máquina em dispositivos com recursos limitados, utilizando paralelismo em CPU e GPU. Foram utilizados três modelos de rede neural convolucional (CNN), implementados pelas bibliotecas PyTorch e Keras, sendo MobileNet V2, EfficientNet B0 e DenseNet-121. Além disso, foram utilizados algoritmos tradicionais de aprendizado de máquina implementados com a biblioteca scikit-learn, incluindo classificação, regressão e agrupamento de dados. Para fins de comparação, também foram desenvolvidas redes neurais de classificação e regressão utilizando as bibliotecas scikit-learn e PyTorch. Os experimentos demonstram que o modelo de CNN MobileNet V2 obteve a melhor relação entre desempenho, acurácia e consumo de energia. Em relação aos algoritmos tradicionais de classificação, *Gaussian Naive Bayes* obteve a melhor relação entre desempenho e acurácia, enquanto na regressão, *Linear Regression* e *Stochastic Gradient Descent* obtiveram o melhor desempenho. Futuramente, este artigo pode ser expandido ao incluir a análise de outros dispositivos de *Single-Board Computer* com diferentes características de *hardware* e outros algoritmos de aprendizado de máquina.

Referências

Baller, S., Jindal, A., Chadha, M., and Gerndt, M. (2021). Deepedgebench: Benchmarking deep neural networks on edge devices. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*, pages 20–30, Los Alamitos, CA, USA. IEEE Computer Society.

de Oliveira, D. C. and Wehrmeister, M. A. (2018). Using deep learning and low-cost rgb and thermal cameras to detect pedestrians in aerial images captured by multirotor uav. *Sensors Journal (Switzerland)*, 18(7).

Karunaratna, S. and Maduranga, P. (2021). Artificial intelligence on single board computers: An experiment on sound event classification. In *2021 5th SLAAI International Conference on Artificial Intelligence (SLAAI-ICAI)*, pages 1–5.

Lacamera, D. (2018). *Embedded Systems Architecture: Explore architectural concepts, pragmatic design patterns, and best practices to produce robust systems*. Packt Publishing.

Lee, J., Wang, P., Xu, R., Dasari, V., Weston, N., Li, Y., Bagchi, S., and Chaterji, S. (2021). Benchmarking video object detection systems on embedded devices under resource contention. In *Proceedings of the 5th International Workshop on Embedded and Mobile Deep Learning*, EMDL'21, page 19–24, New York, NY, USA. Association for Computing Machinery.

Magalhães, S. C., dos Santos, F. N., Machado, P., Moreira, A. P., and Dias, J. (2023). Benchmarking edge computing devices for grape bunches and trunks detection using accelerated object detection single shot multibox deep learning models. *Engineering Applications of Artificial Intelligence*, 117:105604.

Mohebbanaaz, Padma Sai, Y., and et al. (2022). Cognitive assistant deepnet model for detection of cardiac arrhythmia. *Biomedical Signal Processing and Control*, 71.

Nikouei, S. Y. e. a. (2018). Smart surveillance as an edge network service: From harr-cascade, svm to a lightweight cnn. In *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*, pages 256–265.

Pajankar, A. (2020). *Raspberry Pi Computer Vision Programming: Design and implement computer vision applications with Raspberry Pi, OpenCV, and Python 3, 2nd Edition*. Packt Publishing.

Savvidis, P. and Papakostas, G. A. (2021). Remote crop sensing with iot and ai on the edge. page 48 – 54.

Süzen, A. A., Duman, B., and Şen, B. (2020). 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)*, pages 1–5.