UNIVERSIDADE FEDERAL DO ESTADO DO RIO DE JANEIRO
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
ESCOLA DE INFORMÁTICA APLICADA

Aprendizagem Profunda e Inteligência Artificial Verde: Caminhos para um Futuro mais Sustentável

Vívian Rique Gil Ferraro

Orientador
Pedro Nuno de Souza Moura
Coorientador
Daniel da Silva Costa

Rio de Janeiro, RJ – Brasil
Fevereiro, 2025

Catalogação informatizada pelo(a) autor(a)

R594 Rique Gil Ferraro, Vívian
Aprendizagem Profunda e Inteligência Artificial Verde: Caminhos para um Futuro mais Sustentável / Vívian Rique Gil Ferraro. -- Rio de Janeiro : UNIRIO, 2025.
84

Orientador: Pedro Nuno de Souza Moura.
Coorientador: Daniel da Silva Costa.
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Estado do Rio de Janeiro, Graduação em Sistemas de Informação, 2025.

1. Inteligência Artificial Verde. 2. Aprendizagem Profunda. 3. Sustentabilidade. I. Nuno de Souza Moura, Pedro, orient. II. da Silva Costa, Daniel, coorient. III. Título.

Aprendizagem Profunda e Inteligência Artificial Verde: Caminhos para um Futuro mais Sustentável

Vívian Rique Gil Ferraro

Projeto de Graduação apresentado à Escola de Informática Aplicada da Universidade Federal do Estado do Rio de Janeiro (UNIRIO) para obtenção do título de Bacharel em Sistemas de Informação.

Aprovado por:

Prof. Pedro Nuno de Souza Moura, D.Sc. (UNIRIO)

Daniel da Silva Costa, M.Sc. (UNIRIO)

Prof[a]. Geiza Maria Hamazaki da Silva, D.Sc. (UNIRIO)

Prof[a]. Laura de Oliveira Fernandes Moraes, D.Sc. (UNIRIO)

Rio de Janeiro, RJ – Brasil

Fevereiro, 2025

Agradecimentos

À minha mãe, Janei, que mesmo não estando mais aqui, continua sendo um exemplo na minha vida.

Ao meu amor, meu marido Leonardo, agradeço por ser meu melhor amigo, confidente e companheiro de vida. Seu amor, compreensão e incentivo foram essenciais. Sou grata por você acreditar em mim, mesmo quando eu duvidava de mim mesma.

Às minhas filhas, Gabriela e Giovanna, agradeço pelo amor incondicional e por me aguentarem nos momentos mais estressantes.

Aos meus amigos queridos, Andrea e Sapinho, Luciana e Moacyr, agradeço pela amizade verdadeira, pelos momentos de descontração e pelas palavras de incentivo. Mais do que amigos, vocês são a família que a vida me deu.

Ao meu orientador, Professor Pedro, agradeço pela orientação impecável, pelo conhecimento compartilhado com tanta generosidade e pela paciência. Agradeço por me guiar, pelas palavras ditas nos momentos certos e por me incentivar a dar o meu melhor.

Agradeço imensamente ao Daniel pela coorientação, com seu jeito sério e ao mesmo tempo tão gentil, sua ajuda foi fundamental na realização desse trabalho.

Aos meus amigos e colegas de curso, em especial Eduardo Santos e Eric Leal. Agradeço pelas trocas de conhecimento, pela paciência, companheirismo e por me ensinarem tanto. Foram muitas horas de estudo, trabalhos em grupo e noites em claro. Criamos laços que vão muito além das salas de aula.

Este trabalho é dedicado a todos vocês, que tornaram essa conquista possível.

Resumo

Nos últimos anos, avanços notáveis no desenvolvimento de modelos de Aprendizagem Profunda têm sido observados, e esses modelos estão cada vez mais presentes no cotidiano. Esses modelos podem ser aplicados em diversas áreas, desde a gestão de recursos naturais até a modelagem de fenômenos ambientais. No entanto, a demanda por grande capacidade computacional pode levar a um aumento exponencial no consumo de energia e, consequentemente, na pegada de carbono, contradizendo seu propósito ecológico.

Este trabalho visa apresentar evidências e promover uma discussão sobre o consumo energético associado ao desenvolvimento e uso desses modelos, com o objetivo de fomentar uma aplicação mais consciente e eficiente dessa tecnologia, integrando Computação e Ecologia.

O estudo busca avaliar e comparar a eficiência energética e o custo computacional de diferentes modelos de Aprendizagem Profunda. Para isso, foi realizada uma revisão da literatura para identificar as métricas mais relevantes, e experimentos foram realizados com modelos de destaque na área para medir seu desempenho segundo essas métricas, buscando modelos mais eficientes e sustentáveis.

As métricas adotadas para avaliar a eficiência energética e o custo computacional dos modelos de AP incluem: consumo de energia, pegada de carbono, tempo de treinamento, tempo de inferência, número de operações de ponto flutuante (FLOPs) e número de parâmetros. A seleção dessas métricas, baseada no levantamento bibliográfico, garante uma análise completa e relevante do impacto ambiental dos modelos de AP, abrangendo os principais aspectos relacionados ao consumo de energia e aos recursos computacionais.

Os resultados mostram que redes maiores, como AlexNet e ResNet34, apresentam o melhor desempenho, com a AlexNet atingindo uma acurácia de 0,9954, precisão de 0,99535, revocação de 0,99535 e medida-F1 de 0,99535, mas com custos energéticos e computacionais significativamente maiores, resultando em maior emissão de carbono (6,34910 g CO_2 eq e 6,46300 g CO_2 eq, respectivamente). Em contraste, redes menores e mais leves, como LeNet-5 e MobileNet, oferecem um bom equilíbrio entre desempenho e eficiência. A LeNet-5 destaca-se pela menor emissão de carbono (0,93286 g CO_2 eq), com desempenho apenas ligeiramente inferior (acurácia de 0,98680, precisão de 0,98691, revocação de 0,98680 e medida-F1 de 0,98682) aos modelos mais complexos. A MobileNet, embora leve, teve o menor desempenho (acurácia de 0,96900, precisão de 0,96886, revocação de 0,96859 e medida-F1 de 0,96869) e uma emissão de carbono de

1,46991 g CO_2 eq.

A escolha da arquitetura de rede neural deve considerar não apenas o desempenho, mas também o impacto ambiental, buscando um equilíbrio entre precisão e sustentabilidade. O teste de Pearson mostrou uma forte correlação positiva ($r = 0{,}95$, $p = 0{,}01$) entre o consumo de energia e a emissão de carbono, enquanto o teste t de Welch indicou que os modelos com menor custo computacional têm emissões de carbono significativamente menores ($t = -6{,}21$, $p = 0{,}01$). Vale destacar que a análise estatística revelou uma diferença significativa na emissão de carbono entre os modelos com alto custo computacional (AlexNet, ResNet34 e GoogLeNet) e os com baixo custo computacional (LeNet-5 e MobileNet). No entanto, não foi encontrada uma diferença estatisticamente significativa nos valores de medida-F1 entre esses grupos ($t = -1,67$, $p = 0,34$), sugerindo que, embora os modelos mais complexos tenham um maior impacto ambiental, eles não oferecem uma melhoria proporcional no desempenho para essa tarefa.

Os avanços na Aprendizagem Profunda e sua ampla adoção no meio acadêmico e na indústria são inegáveis. Apesar da demanda por alta capacidade computacional e do consequente impacto energético e de carbono, este estudo propõe a viabilidade de desenvolver modelos mais eficientes energeticamente sem comprometer o desempenho. Os resultados podem oferecer uma perspectiva mais sustentável para o futuro da tecnologia.

Palavras-chave: Inteligência Artificial Verde, Aprendizagem Profunda, Eficiência Energética, Modelos Computacionais, Sustentabilidade.

Abstract

In recent years, remarkable advancements in the development of Deep Learning models have been observed, and these models are increasingly present in everyday life. These models can be applied in various areas, from natural resource management to the modeling of environmental phenomena. However, the demand for high computational capacity can lead to an exponential increase in energy consumption and, consequently, in carbon footprint, contradicting their ecological purpose.

This work aims to present evidence and promote a discussion on the energy consumption associated with the development and use of these models, with the goal of fostering a more conscious and efficient application of this technology, integrating Computing and Ecology.

The study seeks to evaluate and compare the energy efficiency and computational cost of different Deep Learning models. To this end, a literature review was conducted to identify the most relevant metrics, and experiments were carried out with prominent models in the field to measure their performance according to these metrics, aiming to identify more efficient and sustainable models.

The metrics adopted to assess the energy efficiency and computational cost of Deep Learning models include: energy consumption, carbon footprint, training time, inference time, number of Floating Point Operations (FLOPs), and number of parameters. The selection of these metrics, based on the literature review, ensures a comprehensive and relevant analysis of the environmental impact of Deep Learning models, covering the main aspects related to energy consumption and computational resources.

The results show that larger networks, such as AlexNet and ResNet34, achieve the best performance, with AlexNet reaching an accuracy of 0.9954, precision of 0.99535, recall of 0.99535, and F1-score of 0.99535, but with significantly higher energy and computational costs, resulting in greater carbon emissions (6.34910 g CO_2 eq and 6.46300 g CO_2 eq, respectively). In contrast, smaller and lighter networks, such as LeNet-5 and MobileNet, offer a good balance between performance and efficiency. LeNet-5 stands out for having the lowest carbon emissions (0.93286 g CO_2 eq), with performance only slightly lower (accuracy of 0.98680, precision of 0.98691, recall of 0.98680, and F1-score of 0.98682) compared to more complex models. MobileNet, although lightweight, had the lowest performance (accuracy of 0.96900, precision of 0.96886, recall of 0.96859, and F1-score of 0.96869) and a carbon emission of 1.46991 g CO_2 eq.

The choice of neural network architecture should consider not only performance

but also environmental impact, seeking a balance between accuracy and sustainability. Pearson's test showed a strong positive correlation ($r = 0.95$, $p = 0.01$) between energy consumption and carbon emissions, while Welch's t-test indicated that models with lower computational costs have significantly lower carbon emissions ($t = -6.21$, $p = 0.01$). Importantly, the statistical analysis revealed a significant difference in carbon emissions between more computationally expensive models (AlexNet, ResNet34, and GoogLeNet) and less expensive ones (LeNet-5 and MobileNet). However, no statistically significant difference was found in the F1-score values between these groups ($t = -1.67$, $p = 0.34$), suggesting that while more complex models have a higher environmental impact, they do not necessarily offer a proportional improvement in performance for this task.

The advancements in Deep Learning and its widespread adoption in academia and industry are undeniable. Despite the demand for high computational capacity and the consequent energy and carbon impact, this study proposes the feasibility of developing more energy-efficient models without compromising performance. The results may offer a more sustainable perspective for the future of technology.

Keywords: Green Artificial Intelligence, Deep Learning, Energy Efficiency, Computational Models, Sustainability.

Sumário

Lista de Figuras

Lista de Tabelas

1 Introdução

1.1 Motivação

Atualmente, vive-se uma era da inovação tecnológica sem precedentes. A Aprendizagem Profunda (*Deep Learning* - AP), um campo da Inteligência Artificial (IA), está se tornando cada vez mais prevalente em nossas vidas diárias. Desde carros autônomos até assistentes virtuais e diagnósticos médicos, a AP está remodelando o mundo como o conhecemos [14].

Embora conceitos como redes neurais artificiais existam desde meados do século XX, a verdadeira revolução causada pela AP começou em 2012, com os resultados impressionantes da rede AlexNet na competição *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [16]. Criada por Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton da Universidade de Toronto, a AlexNet demonstrou um desempenho significativamente maior na classificação de imagens em comparação com métodos de Aprendizagem de Máquina (*Machine Learning* - AM) tradicionais [15], sendo um verdadeiro divisor de águas no campo da IA e da visão computacional. Este evento despertou grande interesse e investimento em AP, abrindo caminho para o desenvolvimento de arquiteturas mais avançadas e poderosas.

A ascensão da AP se deve, em grande parte, à disponibilidade de grandes volumes de dados *(Big Data)* e ao aumento da capacidade de processamento computacional, especialmente com o uso de GPUs (Unidades de Processamento Gráfico). Esses fatores permitiram o treinamento de modelos de AP em escalas nunca vistas antes, resultando em avanços significativos em várias áreas. No entanto, com esses avanços, surgiram novos desafios que se precisa enfrentar.

Uma das áreas de aplicação mais importantes atualmente, considerando-se os Objetivos de Desenvolvimento Sustentável[1] indicados pela Organização das Nações Unidas, é o meio ambiente. Os modelos de AP têm um enorme potencial para abordar questões ambientais, como a simulação de fenômenos naturais, a análise de dados climáticos e de imagens de satélite, além da previsão dos impactos das mudanças climáticas. Em todos esses casos, os modelos de AP, por serem mais robustos e com reconhecido desempenho [14], podem ajudar na obtenção de modelos mais fiéis sobre a dinâmica dos fenômenos ambientais e a utilização dos recursos naturais.

Modelos de AP são capazes de realizar tarefas complexas, mas tipicamente requerem um

[1] https://www.unicef.org/brazil/objetivos-de-desenvolvimento-sustentavel

grande consumo de recursos computacionais [29]. O avanço no desempenho dos modelos de AP desde 2012 tem um preço ambiental cada vez mais alto. O custo computacional do treinamento desses modelos cresceu em um fator alarmante de 300.000 vezes nesse período (Figura 1), uma trajetória insustentável, como alertam Lenherr *et al.* [19]. A essa problemática, soma-se o impacto ambiental do descarte dos equipamentos utilizados, intensificando a urgência por soluções mais sustentáveis na área.

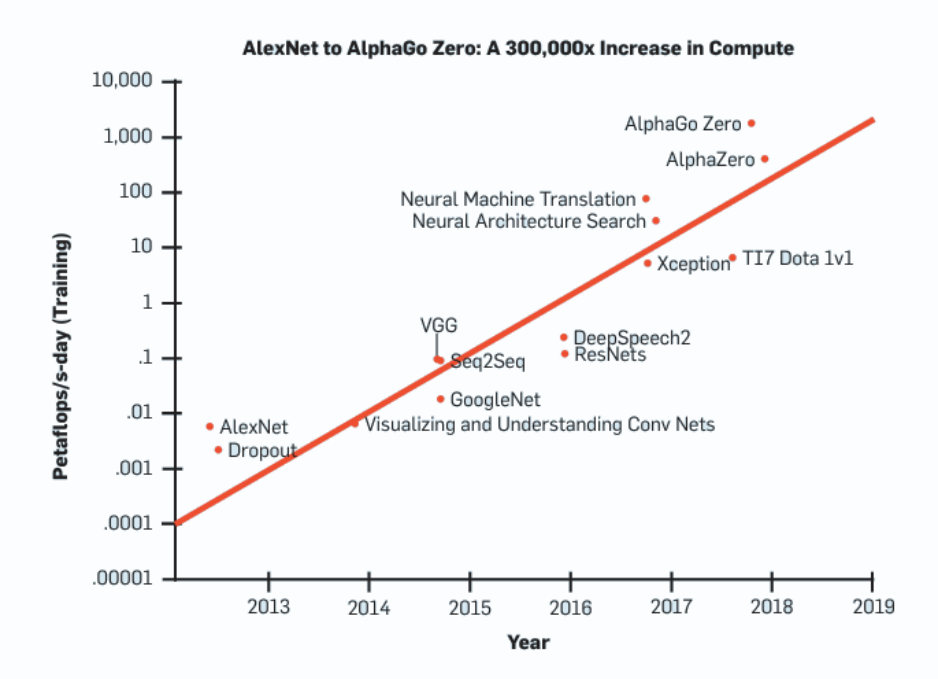

Figura 1: A quantidade de computação necessária para treinar modelos de AP aumentou 300.000 vezes entre 2012 e 2019 (extraída de [26]).

O impacto ambiental do desenvolvimento e uso de modelos de AP é uma questão importante que precisa ser considerada. As pesquisas e aplicações na área de IA estão em uma fase de bastante efervescência, e alguns trabalhos recentes apontam para a necessidade de adotar medidas para minimizar esse impacto [26, 29, 19, 5].

À medida que são feitos esforços para melhorar a qualidade de vida das pessoas por meio da tecnologia, também se deve considerar o impacto ambiental desses avanços. Afinal, a tecnologia utilizada diariamente na sociedade não está isenta de custos para o planeta.

A necessidade de abordar essa questão é mais urgente do que nunca. A humanidade está em um momento crucial na história, em que as decisões tomadas hoje definirão o futuro do planeta. A janela de oportunidade para mitigar os efeitos das mudanças climáticas está se fechando rapidamente, a falta de ação pode comprometer irreversivelmente a saúde do

ecossistema global e, consequentemente, a própria sobrevivência da espécie humana.[2]

Portanto, é essencial que se busquem maneiras de tornar o uso da AP mais sustentável, mitigando o consumo exagerado de energia, a emissão excessiva de carbono e a degradação do meio ambiente. Ao implementar essas medidas, a IA se alinha às necessidades climáticas do planeta e assegura a continuidade de seus avanços e benefícios para a sociedade.

Em meio aos avanços da IA, surgiu o conceito de Inteligência Artificial Verde (IA Verde), uma abordagem responsável que busca harmonizar o progresso tecnológico com a sustentabilidade ambiental. Conforme introduzido por Schwartz *et al.* [26], a IA Verde se opõe à IA Vermelha, que prioriza o aumento do desempenho sem considerar os custos ambientais, econômicos e sociais.

A IA Verde enfatiza a eficiência como um critério de avaliação essencial, promovendo práticas que reduzam o consumo de energia e as emissões de carbono. Isso inclui, por exemplo, o compartilhamento de modelos pré-treinados, que evita a redundância no uso de recursos, e a adoção de técnicas como a quantização e o aprendizado federado, recomendadas por Lenherr *et al.* [19].

A quantização otimiza o uso de recursos computacionais e energéticos ao reduzir a resolução dos parâmetros dos modelos, diminuindo seus tamanhos. Já o aprendizado federado permite o treinamento de modelos em dados descentralizados, preservando a privacidade das informações e reduzindo a necessidade de transferência massiva de dados, o que também contribui para a eficiência energética.

Ao valorizar a eficiência energética e a sustentabilidade, a IA Verde não apenas contribui para a mitigação do impacto ambiental, mas também incentiva a inovação em direção a soluções de IA mais acessíveis e inclusivas [26]. Dessa forma, a IA Verde representa um avanço consciente na forma de utilizar a tecnologia, promovendo também a preservação dos recursos naturais do planeta.

1.2 Objetivos

Este trabalho tem por objetivo comparar e avaliar a eficiência de modelos de AP, a fim de demonstrar a viabilidade de se obterem modelos de AP mais eficientes e com menor impacto ambiental.

Para orientar este estudo, foram utilizadas as principais métricas comumente adotadas na

[2]https://unfccc.int/climate-action

literatura para avaliar o desempenho de modelos de AP, como acurácia, precisão, revocação e medida-F1. Além disso, foram utilizadas métricas para medir a eficiência energética, como consumo de energia e pegada de carbono, e métricas referentes ao custo computacional, como tempo de execução, tempo de inferência, número de parâmetros e número de operações de ponto flutuante (FLOPs). Essas métricas foram selecionadas com base em trabalhos de referência sobre IA Verde.

A partir dessa análise, buscou-se estabelecer uma comparação experimental entre diversas arquiteturas de redes neurais utilizadas em visão computacional, ampliando a compreensão sobre como diferentes modelos se comportam em termos de eficiência e sustentabilidade. Assim, espera-se mostrar que é possível selecionar ou construir modelos que consumam menos energia nas etapas de treinamento e inferência, assim como tenham menor custo computacional, mas que ainda apresentem desempenho similar ao de modelos mais dispendiosos, contribuindo para práticas mais responsáveis no desenvolvimento de tecnologias de IA. Ao final, busca-se fortalecer a proposta da IA Verde, demonstrando que é possível avançar na área da AP de forma mais sustentável.

1.3 Organização do texto

O desenvolvimento deste trabalho está organizado em capítulos e, além desta introdução, seguirá o seguinte formato:

Capítulo 2: Apresenta os conceitos básicos de AP e IA Verde necessários para a compreensão deste trabalho;

Capítulo 3: Aborda a revisão de trabalhos relacionados;

Capítulo 4: Apresenta as arquiteturas utilizadas neste estudo;

Capítulo 5: Detalha o ambiente computacional utilizado, o conjunto de dados, as métricas e os experimentos computacionais realizados;

Capítulo 6: Analisa e discute os resultados obtidos;

Capítulo 7: Apresenta as conclusões finais, destaca as contribuições da pesquisa e sugere caminhos para aprofundamentos futuros.

2 Conceitos Fundamentais

Este capítulo visa a introduzir os conceitos fundamentais para a compreensão deste trabalho, especialmente aqueles relacionados à Aprendizagem Profunda e à Inteligência Artificial Verde. Aqueles que se sentirem confortáveis com tais conceitos podem pular este capítulo sem prejuízo para o entendimento do trabalho.

2.1 Aprendizagem Profunda

A Aprendizagem de Máquina (*Machine Learning* - AM) é um campo da inteligência artificial que se dedica ao estudo e ao desenvolvimento de modelos capazes de aprender a partir de dados. Mais especificamente, Tom Mitchell fornece a seguinte definição para a AM [22]: "Diz-se que um programa de computador 'aprende' por meio da experiência *(E)* em relação a uma tarefa *(T)* e uma medida de desempenho *(P)*, se seu desempenho em tarefas em *T*, conforme medido por *P*, melhora com a experiência *E*.".

A AP, também conhecida como *Deep Learning*, é um subcampo específico dentro da AM que foca no uso de redes neurais artificiais com várias camadas. Essas redes são chamadas de redes neurais profundas devido à sua profundidade, ou seja, ao grande número de camadas que possuem [16].

Enquanto a AM tradicional exige a extração prévia das características (*features*) mais relevantes dos dados para que o modelo possa aprender e realizar inferência, na AP as redes neurais profundas são capazes de aprender a extrair automaticamente as características mais importantes dos dados durante o próprio processo de treinamento [16]. Isso é possível graças à grande capacidade de aprendizado automático de representações (*representation learning*) dos dados que as redes neurais apresentam. Como resultado, os modelos de AP podem ser aplicados a uma ampla variedade de dados ou tarefas sem a necessidade de provimento ou implementação de extratores de características para cada tipo de dado ou tarefa.

2.2 Redes Neurais

O conceito de redes neurais artificiais remonta à década de 1950, quando o neurobiólogo Frank Rosenblatt desenvolveu o Perceptron, um algoritmo inspirado nos neurônios biológicos.

O perceptron foi a primeira formulação de um neurônio artificial, o modelo original foi concebido para receber entradas binárias (0 ou 1), calcular uma soma ponderada dessas entradas e gerar uma saída binária.

$$z = wx + b \tag{1}$$

Na equação 1, x representa as entradas, w os pesos associados a cada entrada, e b o viés *(bias)*, um termo constante. O resultado z é a soma ponderada das entradas com o ajuste do viés, e serve como a base para a saída do perceptron.

Uma rede neural artificial é um modelo computacional composto por unidades interconectadas de neurônios artificiais. Cada neurônio artificial recebe um conjunto de sinais de entrada, processa esses sinais e gera uma saída, similar ao que acontece nos neurônios biológicos. Os pesos associados a cada entrada, em cada neurônio, são ajustáveis por meio de um processo de treinamento a fim de realizar tarefas específicas.

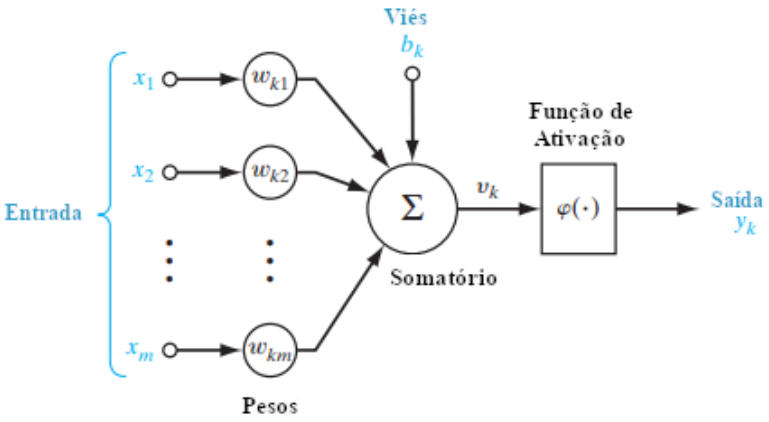

Figura 2: Estrutura de um neurônio artificial (adaptada de [10]).

A Figura 2 ilustra a estrutura básica de um neurônio artificial (k). O neurônio recebe diversos sinais de entrada $(x_1, x_2, \ldots, x_m)$, que podem representar características de um exemplo (amostra) ou as saídas de outros neurônios. Cada entrada é então multiplicada por um peso $(w_1, w_2, \ldots, w_m)$, que representa a importância relativa dessas entradas na decisão do neurônio. Os resultados das multiplicações são somados a um viés (b_k), que é um valor constante. Esse resultado passa por uma função de ativação $\phi(.)$ e produz uma saída (y_k) do neurônio. Essa saída (y_k) é então passada para os neurônios da camada seguinte na rede neural, ou, no caso da última camada, representa a saída final do modelo.

As redes neurais são capazes de aprender representações hierárquicas dos dados, de modo que cada camada da rede aprende a representar os dados de uma maneira mais abstrata e complexa do que a camada anterior [16].

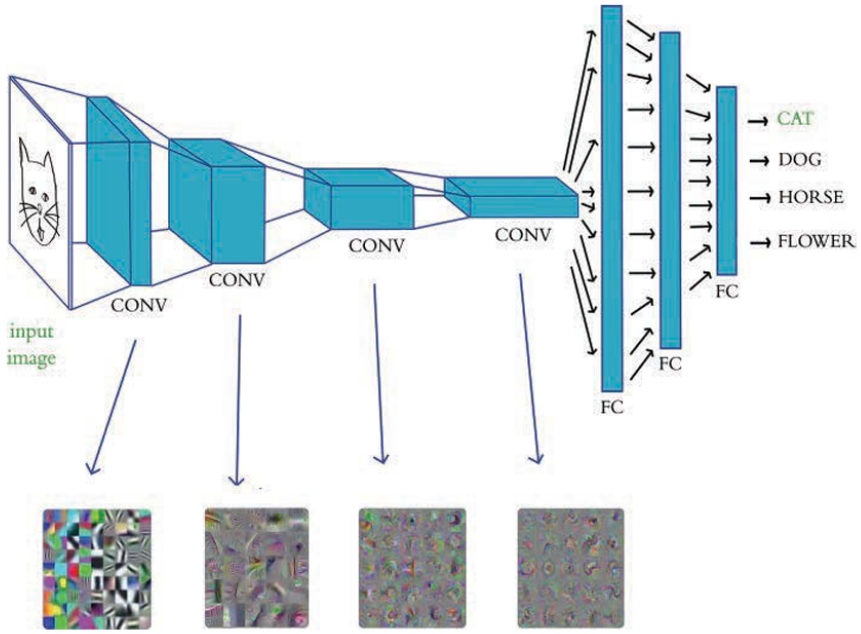

Figura 3: Arquitetura hierárquica do AlexNet para a tarefa de classificação de imagens (extraída de [16]).

A Figura 3 ilustra o processamento de uma imagem de gato pela AlexNet. A rede extrai características visuais em diferentes níveis de complexidade: desde características simples, como bordas, na primeira camada, até conceitos abstratos progressivamente mais complexos em camadas mais profundas. À medida que a informação flui para as camadas mais profundas, a rede aprende a reconhecer características mais complexas, como texturas, partes do corpo do gato (orelhas, olhos, etc.) e, finalmente, o conceito abstrato de "gato" como um todo, levando à sua correta classificação.

2.2.1 Estrutura de Redes Neurais

Por meio de um único neurônio artificial, só é possível abordar problemas linearmente separáveis, isto é, cujos exemplos podem ser separados por uma reta em duas dimensões ou por um hiperplano em muitas dimensões [14]. A estratégia comumente adotada é então de se disporem os neurônios em diversas camadas, de modo que neurônios de camadas subsequentes estejam conectados uns aos outros. Assim sendo, uma rede neural típica contém três tipos principais de camadas:

- Camada de Entrada: representa os dados brutos que serão processados pela rede;
- Camadas Ocultas: são as camadas intermediárias da rede, que realizam a maioria dos cálculos. Podem existir várias camadas ocultas, cada uma composta por diversos neurônios;
- Camada de Saída: responsável por produzir o resultado final da rede, como, por exemplo, uma classificação ou uma previsão.

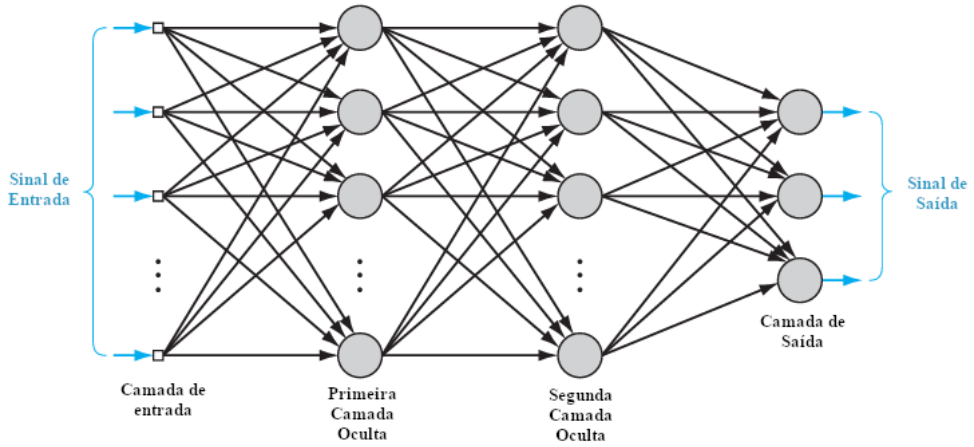

Figura 4: Estrutura básica de uma rede neural (adaptada de [10]).

Na Figura 4, está representada uma rede composta por quatro camadas. A primeira camada, de entrada, que representa os dados introduzidos na rede, duas camadas ocultas totalmente conectadas e a camada de saída.

Uma camada totalmente conectada, também conhecida como camada densa, é um tipo de camada oculta comum em arquiteturas de AP. Nessas camadas, cada neurônio recebe informações de todos os neurônios da camada anterior, estabelecendo conexões completas entre as camadas. Isso permite que a camada densa recombine as informações de forma não-linear, aumentando a capacidade da rede de aprender e representar características complexas a partir dos dados de entrada [16].

2.2.2 Funcionamento básico

As redes neurais artificiais são compostas por camadas de neurônios interconectadas. Em redes *feedforward* (com propagação para a frente), como a rede ilustrada na Figura 4, as camadas são organizadas sequencialmente, com o fluxo de informação seguindo uma única direção, da camada de entrada para a camada de saída. As conexões entre os neurônios

geralmente são totalmente conectadas (ou densas) e cada conexão possui um peso específico. O processo de funcionamento de uma rede neural *feedforward*, durante o treinamento, consiste em:

- Inicialização: Inicialmente, os pesos e vieses (parâmetros da rede) são inicializados aleatoriamente, geralmente seguindo uma distribuição de probabilidade específica (como a distribuição uniforme ou normal).

- Propagação para a frente (*Forward Propagation*): Os dados de entrada são introduzidos na camada de entrada. Os neurônios da camada seguinte recebem a soma ponderada das entradas com o ajuste do viés, provenientes dos neurônios da camada anterior. Essa soma é então passada por uma função de ativação (como, por exemplo, a sigmoide, ReLU ou tangente hiperbólica), que introduz não-linearidade na rede. Esse processo se repete camada a camada até que a informação atinja a camada de saída, produzindo a predição da rede.

- Retropropagação (*Backpropagation*) e Aprendizado: A saída da rede é comparada com o valor correto esperado (*ground truth*) através de uma função de perda (também chamada de função de custo), como o erro quadrático médio ou a entropia cruzada, que serão detalhadas na seção 2.2.8 - Funções de Perda. A função de perda quantifica a discrepância entre a predição da rede e o valor real. O algoritmo de retropropagação calcula o gradiente da função de perda em relação a cada peso e viés da rede, propagando esse gradiente da camada de saída para as camadas anteriores. Com base nesses gradientes, a rede ajusta os pesos e vieses, buscando minimizar a função de perda e, consequentemente, melhorar a exatidão das predições. Este processo de ajuste iterativo dos pesos e vieses é o que chamamos de aprendizado da rede neural.

2.2.3 Treinamento da Rede Neural

O processo de treinamento de uma rede neural consiste em iterativamente ajustar os pesos e vieses da rede para minimizar a função de perda. Este processo envolve a repetição dos passos de propagação para frente, cálculo da perda e retropropagação sobre o conjunto de dados de treinamento. Cada repetição completa desse processo sobre o conjunto de dados de treinamento define uma época (*epoch*).

Para otimizar o processo de treinamento, o conjunto de dados é frequentemente dividido em lotes (*batches*) menores. O tamanho do lote (*batch size*) determina a quantidade de

exemplos processados simultaneamente antes da atualização dos pesos e vieses.

Durante uma época, cada exemplo de treinamento é apresentado à rede, dentro de um lote, e os pesos e vieses da rede são ajustados com base no erro de predição. O treinamento usualmente compreende múltiplas épocas, possibilitando que a rede aprenda padrões complexos presentes nos dados.

A camada de saída é a responsável por fornecer a resposta final da rede. Essa resposta pode ser a indicação de uma classe mais provável em uma tarefa de classificação (por exemplo, "é um gato" ou "é um cachorro") ou um valor numérico mais provável em uma tarefa de regressão (como a temperatura do dia seguinte).

As funções de ativação são componentes essenciais das redes neurais artificiais. As funções de ativação de interesse são aquelas que introduzem não-linearidade no modelo, permitindo que ele aborde problemas não-linearmente separáveis, assim como aprenda e represente padrões mais complexos do que os que podem ser modelados por modelos lineares [14]. Cada função de ativação possui características específicas. Algumas das principais funções de ativação serão apresentadas a seguir.

2.2.4 Função Sigmoide

A função Sigmoide mapeia qualquer argumento para o intervalo $[0,1]$. Esta função de ativação foi uma das primeiras funções criadas para introduzir não-linearidade em redes neurais [16] e a sua definição é expressa na Equação 2:

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{2}$$

em que z representa o resultado da soma ponderada entre entradas e pesos do neurônio mais o viés ($wx+b$) e que é mapeado para um valor no intervalo $[0,1]$. Aqui, $\sigma(z)$ é a saída da função Sigmoide e e é a base do logaritmo natural.

A Figura 5 apresenta o gráfico da função Sigmoide, em que é possível ver que o seu conjunto imagem corresponde ao intervalo $[0,1]$. Essa função consiste em uma versão suave da função de ativação do perceptron, cuja saída é 0 ou 1.

No entanto, a Sigmoide apresenta o problema do desaparecimento do gradiente (*vanishing gradient problem*), pois as derivadas desta função podem se tornar extremamente pequenas, o que pode levar à saturação dos neurônios e à interrupção do aprendizado da rede. A saturação de um neurônio acontece quando os seus valores de entrada ou de saída ficam muito grandes

ou muito pequenos, de maneira que a função fica saturada, situando-se no seu valor máximo ou mínimo [16]. Nessa situação, os ajustes feitos nos pesos de tal neurônio tendem a ser inócuos, pois não alteram o valor de saída da função de ativação. Tal limitação levou ao desenvolvimento de outras funções de ativação, que serão descritas nas subseções seguintes.

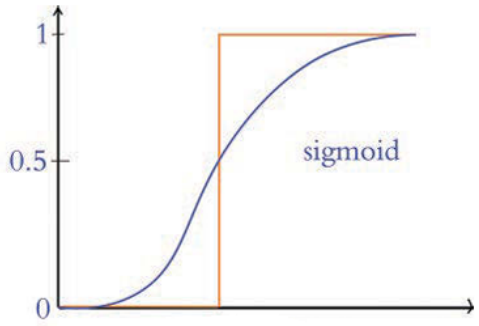

Figura 5: Gráfico da função de ativação Sigmoide (linha azul), que demonstra uma transição suave entre 0 e 1, em contraste com a transição abrupta da função de ativação do Perceptron (linha laranja) (extraída de [16]).

2.2.5 Função Tanh

A função *tanh* (Tangente Hiperbólica) é semelhante à Sigmoide, mas mapeia o seu argumento para o intervalo $[-1,1]$, em vez de para o intervalo $[0,1]$. A definição da função *tanh* segue na Equação 3:

$$\sigma(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{3}$$

em que z representa a soma ponderada entre entradas e pesos do neurônio, acrescida do viés ($wx+b$) e e é a base do logaritmo natural. A saída da função *tanh*, $\sigma(z)$, varia no intervalo $[-1,1]$ e tende a ser centrada em torno de 0, permitindo que as saídas dos neurônios sejam tanto positivas quanto negativas, o que torna a saturação dos neurônios e o desaparecimento do gradiente menos provável, permitindo que a rede aprenda com mais eficiência [16].

A Figura 6 mostra o gráfico da função de ativação *tanh*. Para valores de argumento muito negativos, a função *tanh* se aproxima de -1, enquanto para valores muito positivos, aproxima-se de 1. A função passa pela origem para $z = 0$. Para valores de entrada extremos, a *tanh* satura em -1 ou 1, resultando em uma derivada próxima de zero. Isso torna a rede suscetível ao problema do desaparecimento do gradiente, embora geralmente em menor grau

quando comparada à função de ativação Sigmoide [1].

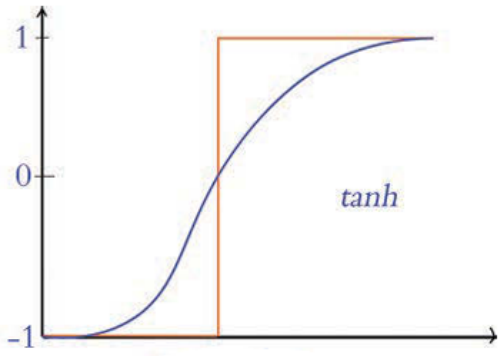

Figura 6: Gráfico da função de ativação *tanh* (linha azul), que demonstra uma transição suave entre -1 e 1, em contraste com a transição brusca da função de ativação do Perceptron (linha laranja) (extraída de [16]).

2.2.6 Função ReLU

A função ReLU (*Rectified Linear Unit*) é atualmente a função de ativação mais popular em redes neurais profundas [16]. Essa função mapeia valores de argumento negativos para zero e mantém os valores positivos inalterados. Segundo [15], "CNNs profundas com neurônios usando a função de ativação ReLU treinam várias vezes mais rápido do que suas equivalentes com neurônios usando a função *tanh*". Isso ocorre porque a função é computacionalmente mais simples e fácil de calcular, agilizando o processamento da etapa de treinamento.

A função ReLU está definida na Equação 4, em que z representa o valor da soma ponderada $(wx+b)$, e a saída da função $\sigma(z)$ corresponde à entrada se a entrada for positiva; caso contrário, corresponde a zero.

$$\sigma(z) = \max(0, z) \tag{4}$$

A representação gráfica da ReLU, conforme a Figura 7 mostra, é uma linha reta com declive 1 para valores positivos de z e uma linha horizontal em $y = 0$ para valores negativos ou iguais a zero. É, portanto, a composição de duas funções lineares. A função é significativamente mais rápida de calcular em relação a funções como a Sigmoide e a *tanh*, que envolvem exponenciais em sua definição, pelo fato envolver apenas uma comparação e, possivelmente, uma atribuição. Além disso, a derivada da ReLU permanece constante e igual a 1 e não se aproxima de zero para entradas positivas. Essa diferença permite que

o gradiente se propague mais efetivamente através das camadas, mitigando o problema do desaparecimento do gradiente [16].

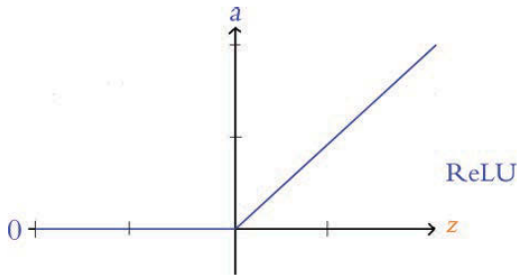

Figura 7: Gráfico da função de ativação ReLU (extraída de [16]).

Entretanto para entradas negativas, a saída do neurônio será sempre zero, sua derivada e o gradiente também serão zero, por essa razão os pesos desse neurônio nunca serão atualizados durante o treinamento (neurônios "mortos"). Além disso, sua saída não variará entre diferentes opções de entradas e, portanto, não desempenhará um papel na discriminação entre diferentes instâncias [1].

Para mitigar esse problema, podemos usar a função Leaky ReLU (*Leaky Rectified Linear Unit*). Diferente da ReLU convencional, que define a saída como zero para entradas negativas, a Leaky ReLU permite que pequenas frações dessas entradas negativas sejam passadas adiante. A fórmula da Leaky ReLU é:

$$f(x) = \begin{cases} x, & \text{se } x \geq 0 \\ \alpha x, & \text{se } x < 0 \end{cases} \tag{5}$$

na qual, α é um pequeno valor positivo, menor do que 1. Representada graficamente na figura 8, a função Leaky ReLU minimiza o problema dos neurônios "mortos", pois, mesmo para entradas negativas, garante uma pequena saída diferente de zero [1]. Essa característica assegura que os gradientes não se anulem completamente, permitindo que os pesos dos neurônios continuem a ser atualizados durante o treinamento, permitindo à rede aprender melhor.

2.2.7 Função Softmax

A função Softmax converte um vetor de valores em uma distribuição de probabilidade, em que todo componente está no intervalo $[0,1]$ e a soma desses componentes é igual a 1. Essa

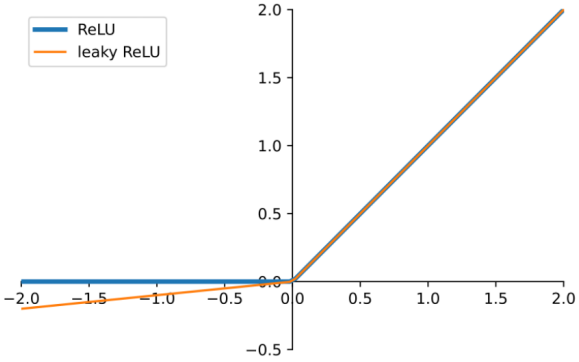

Figura 8: Gráfico da função de ativação Leaky ReLU (linha laranja), que permite uma pequena inclinação para valores negativos, em contraste com a função ReLU (linha azul), que zera todos os valores negativos (extraída de [13]).

função é utilizada frequentemente em redes neurais, especialmente na camada de saída para tarefas de classificação multiclasses, isto é, quando há mais de duas classes.

A Equação 6 mostra a sua definição, em que cada componente z_i do vetor z é transformada para um valor no intervalo $[0,1]$. A soma dos exponenciais ($\sum_{j=1}^{n} e^{z_j}$) no denominador realiza uma ponderação e garante que as saídas somem 1.

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{6}$$

2.2.8 Funções de Perda

As funções de perda, também conhecidas como funções de custo, são componentes cruciais no treinamento de modelos de AP. Elas calculam a diferença entre as previsões do modelo e os valores reais esperados, isto é, quantificam o quanto as previsões fornecidas pelo modelo estão se distanciando das saídas esperadas, que é o erro.

Alguns tipos comuns de funções de perda são:

Custo Quadrático ou Erro Quadrático Médio (*Mean Squared Error*): Essa função calcula a média dos quadrados das diferenças entre as previsões do modelo e das saídas reais esperadas.

A Equação 7 expressa a função de Custo Quadrático. Nessa, n é o número de exemplos de treinamento, $\hat{y}_i$ corresponde à previsão do modelo para o i-ésimo exemplo e y_i, ao valor observado (real) para o i-ésimo exemplo.

$$C = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{7}$$

Segundo os autores Krohn *et al.* [16], embora o custo quadrático sirva como uma introdução direta às funções de perda, ele tem uma falha vital, já que pode levar a uma saturação dos neurônios de uma rede neural. Isso ocorre porque a função de Custo Quadrático tende a gerar gradientes muito pequenos para erros elevados ou muito grandes para erros pequenos durante a retropropagação.

Algumas estratégias, como normalização de lote (*Batch Normalization*) e o ajuste dos pesos, podem ser utilizadas para mitigar a saturação dos neurônios que compõem a rede.

Custo de Entropia Cruzada (*Cross-Entropy*): A Entropia Cruzada é comumente usada em problemas de classificação, especialmente em classificação binária ou multiclasse (*n*-ária). Ela mede a diferença entre duas distribuições de probabilidade: a distribuição observada (real) e a distribuição prevista pelo modelo. De acordo com [16], usar o Custo de Entropia Cruzada ao invés do Custo Quadrático minimiza o impacto de neurônios saturados e também afeta positivamente a velocidade de treinamento da rede.

A Equação 8 apresenta a definição da função de Custo de Entropia Cruzada, em que n é o número de exemplos de treinamento, y_i é a distribuição de probabilidade observada (real) para o i-ésimo exemplo e $\hat{y}_i$ é a distribuição de probabilidade prevista pelo modelo para o i-ésimo exemplo. Como tal custo penaliza estimativas incorretas de forma logarítmica, erros maiores são penalizados mais severamente.

$$C = -\frac{1}{n} \sum_{i=1}^{n} [y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i)] \tag{8}$$

As funções de perda servem como guias para o processo de treinamento e dos modelos de AP. Esse processo de aprendizagem ocorre através da retropropagação, que distribui o custo de volta pelas camadas, ajustando os pesos de cada neurônio para minimizar esse erro e tornar a rede cada vez mais precisa em suas previsões.

2.2.9 Otimização

Para minimizar o erro produzido por um modelo, representado pela função de custo, são utilizados algoritmos otimizadores como o gradiente descendente estocástico que ajusta os pesos durante a retropropagação e permite um aprendizado mais eficiente. No que segue, são fornecidas algumas definições:

Taxa de Aprendizado: É um hiperparâmetro utilizado para controlar como uma rede neural ajusta seus pesos durante o treinamento. Normalmente, durante as diversas rodadas de treinamento, é possível testar diferentes valores e observar o comportamento da função de perda. Pode-se ainda utilizar um mecanismo adaptativo para a taxa de aprendizado que reduz o seu valor quando o valor da função de custo para de diminuir.

Gradiente Descendente: É o algoritmo de otimização que busca encontrar o ponto de mínimo de uma função a partir do cálculo do gradiente, que é dado pela derivada da função de custo em relação aos pesos e vieses. Como o gradiente indica a direção de maior crescimento da função de custo, caminha-se no sentido contrário do apontado por ele, buscando minimizar o custo.

Gradiente Descendente Estocástico: Nesse tipo de algoritmo, o gradiente descendente é calculado sobre lotes (*batches*) dos dados de treinamento. Em vez de utilizar todo o conjunto de dados, tal como no gradiente descendente original, o que exigiria uma quantidade muito grande de memória e poderia se tornar inviável, a solução encontrada foi dividir todo o conjunto de dados em pequenos subconjuntos, os lotes, que são obtidos a partir de amostragens aleatórias. Cada rodada de treinamento é então realizada em um dado lote. O otimizador **Adam**, utilizado neste trabalho, é um tipo de algoritmo que utiliza o gradiente descendente estocástico.

2.2.10 Sobreajuste e Regularização

Durante o treinamento de um modelo, o custo (erro) calculado nos dados de treinamento geralmente diminui com o tempo (épocas), pois é o que está sendo objeto de minimização pelo algoritmo de otimização. Idealmente, o custo nos dados de validação (dados não usados no treinamento, mas usados para testar a generalização do modelo) também deve diminuir. No entanto, quando um modelo é treinado por várias épocas, pode acontecer de o custo no

conjunto de validação começar a aumentar, enquanto o custo no conjunto de treino continuar a diminuir. Esse fenômeno é conhecido como sobreajuste ou *overfitting*. [16]

O sobreajuste ocorre quando um modelo se ajusta tão bem aos dados de treinamento que memoriza ruídos e detalhes específicos, prejudicando sua capacidade de generalizar para novos dados. Quando se dispõe de poucos dados de treinamento e se adota uma rede neural profunda complexa, com um número grande de parâmetros, esse é um problema comum.

Algumas técnicas, denominadas de técnicas de regularização, podem ser utilizadas para reduzir o sobreajuste:

Regularização L1/L2: Conhecidas como Regressão LASSO (L1) e Regressão de Cume ou *Ridge* (L2), essas técnicas penalizam os modelos por usar um número excessivo de parâmetros, adicionando-os à função de custo do modelo. Parâmetros grandes contribuem mais para a função de custo, então, a menos que os parâmetros reduzam significativamente a diferença entre a saída estimada do modelo ($\hat{y}$) e a verdadeira (y), o custo irá aumentar [16].

***Dropout*:** Essa técnica desenvolvida por Geoffrey Hinton e seus colegas da Universidade de Toronto no artigo [15], simula a remoção aleatória de neurônios durante rodadas de treinamento. Ao descartar neurônios aleatoriamente, o *dropout* dificulta a criação de caminhos de propagação muito específicos através da rede, evitando que o modelo se torne excessivamente dependente de características específicas dos dados de treinamento para gerar previsões [16].

***Early Stopping*:** É uma técnica que monitora o desempenho do modelo para o conjunto de validação e interrompe o treinamento quando, após um número pré-definido de épocas, a performance nesse conjunto não apresenta melhora significativa ou começa a piorar, evitando que o modelo memorize os dados de treino [10].

***Data Augmentation*:** Além de regularizar os parâmetros do modelo, aumentar o tamanho do conjunto de dados de treinamento pode melhorar a capacidade de generalização do modelo para dados não vistos. Se novos dados de qualidade puderem ser coletados, esta é a melhor alternativa, porém, caso a coleta de novos dados seja difícil, é possível gerar novos dados a partir dos existentes por meio de técnicas de *data augmentation*, como, por exemplo, distorção, desfoque, deslocamento, adição de ruído e rotação da imagem [16].

Normalização em Lote (*Batch Normalization*): Durante o treinamento de uma rede neural, a distribuição dos parâmetros em uma camada oculta muda gradualmente, já que a rede ajusta esses parâmetros a fim de minimizar o custo. No entanto, essa mudança na distribuição dos pesos de uma camada pode deslocar as entradas da camada seguinte para longe de uma distribuição ideal (como, por exemplo, a distribuição normal). A normalização em lote (*batch normalization*) resolve esse problema normalizando as ativações dos neurônios da camada anterior. Ela centraliza a distribuição dos valores com média 0 e desvio padrão 1. Isso evita que valores extremos na camada anterior causem problemas como gradientes que tendem a valores excessivamente grandes (explodindo) ou que tendem a zero (desaparecendo) na camada seguinte [16].

2.2.11 Redes Neurais Convolucionais

"Uma Rede Neural Convolucional (*Convolutional Neural Network* - CNN) é uma rede neural artificial que apresenta uma ou mais camadas convolucionais" [16].

As CNNs revolucionaram o campo da visão computacional [16], por sua capacidade de analisar e interpretar imagens. As CNNs são inspiradas na maneira como o cérebro humano processa informações visuais, detectando padrões e combinando essas características com o objetivo de reconhecer formas mais complexas.

Um avanço significativo no campo das CNNs veio com o desenvolvimento da rede LeNet-5 por Yann LeCun e Yoshua Bengio [18], que utilizou o algoritmo de retropropagação para treinar a primeira CNN. A LeNet-5 constituiu um marco na visão computacional, capaz de reconhecer dígitos manuscritos com muita exatidão.

No trabalho [15], os pesquisadores Alex Krizhevsky, Ilya Sutskever e Geoffrey Hinton apresentaram a rede AlexNet, que revolucionou a classificação de imagens ao vencer a competição *ImageNet Large Scale Visual Recognition Challenge* (ILSVRC) com uma margem significativa em relação aos demais competidores e atingindo um marco expressivo nesse *benchmark* [16]. Dessa maneira, esse evento marcou a ascensão da AP como a abordagem dominante na área de AM.

As CNNs utilizam camadas convolucionais, conjuntos de filtros (ou *kernels*), que são pequenas matrizes contendo pesos, que deslizam sobre diferentes posições da imagem, para extrair características espaciais dos dados de entrada.

A Figura 9, apresenta o esquema de uma rede convolucional, na qual imagens de entrada de 28×28 pixels com um único canal (escala de cinza) são submetidas a filtros convolucionais.

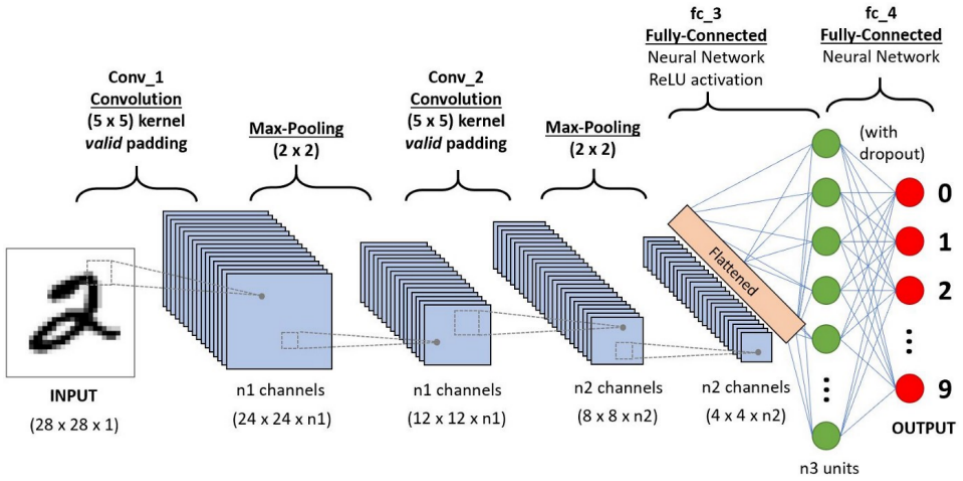

Figura 9: Esquema de Rede Neural Convolucional para reconhecimento de dígitos manuscritos (extraída de [25]).

Cada filtro percorre a imagem, calculando o produto escalar entre seus pesos e os valores dos pixels da região coberta, produzindo um mapa que destaca os padrões específicos detectados pelo filtro.

Uma função de ativação, como a ReLU, descrita na Seção 2.2.6, é aplicada aos mapas, gerando os chamados mapas de ativação, para introduzir não-linearidade no modelo.

As camadas convolucionais são comumente sucedidas por camadas de *pooling* (subamostragem), cuja função é reduzir a dimensionalidade espacial dos mapas de ativação. Essa redução tem como consequência a diminuição do número de parâmetros do modelo e da sua complexidade computacional [16]. O *Max Pooling*, que realiza a seleção do valor máximo dentro de uma janela de filtro com dimensão definida, é a operação de *pooling* mais frequente.

A figura 10 ilustra a aplicação de uma operação de *Max Pooling* em uma matriz de entrada que é dividida em regiões de 2×2. O valor máximo de cada região é selecionado para formar a matriz de saída. O resultado dessa subamostragem é mostrado na matriz inferior, que contém os valores máximos de cada região: 9, 7, 8, 6.

Os mapas de ativação resultantes de sucessivas camadas convolucionais são transformados em um vetor unidimensional, processo conhecido como *flattening*. Esse processo é essencial para converter a estrutura multidimensional dos mapas de características (geralmente em formato de matriz ou tensor) em um formato linear que possa ser utilizado no final da rede para realizar a classificação.

Muitas CNNs apresentam camadas totalmente conectadas ao final da rede. As informações extraídas pelas camadas convolucionais são integradas por estas camadas, sendo transformadas

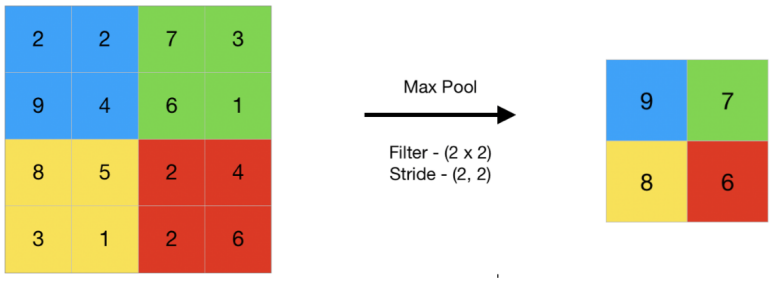

Figura 10: Operação de *Max Pooling* com filtro 2×2, resultando na seleção dos valores máximos de cada região (extraída de [21]).

em uma representação utilizável para a tarefa final, como classificação ou regressão.

Para problemas de classificação, a camada de saída frequentemente utiliza uma função de ativação do tipo Softmax, que transforma as saídas dos neurônios em probabilidades normalizadas, onde cada valor representa a probabilidade da entrada pertencer a uma determinada classe. Isso permite que a rede faça previsões probabilísticas, indicando a confiança da rede em cada classe.

2.3 Inteligência Artificial Verde

O conceito de Inteligência Artificial (IA) Verde, proposto por [26], refere-se a práticas de pesquisa em IA que são mais amigáveis ao meio ambiente e mais inclusivas. A ideia central é equilibrar a busca por resultados de alto desempenho com a eficiência energética, econômica e ambiental.

A pesquisa em IA tem se concentrado fortemente em obter resultados de "estado da arte" muitas vezes ignorando os custos econômicos, ambientais e sociais associados. A IA Verde propõe que a eficiência seja um critério de avaliação para artigos de IA, juntamente com a precisão e outras métricas de desempenho. Isso significa que, além de buscar o melhor desempenho possível, os pesquisadores também devem considerar quanto de energia e recursos estão sendo utilizados para alcançar esses resultados.

A prática de IA Verde envolve medir e divulgar o consumo energético associado ao desenvolvimento, treinamento e execução de modelos de IA. Essa medição permite aos pesquisadores compreender o impacto ambiental de seus trabalhos e comparar a eficiência de diferentes abordagens. Por exemplo, ao treinar um modelo de Aprendizagem Profunda, é importante saber quanta energia é consumida pelos servidores e *data centers*.

Além disso, a IA Verde incentiva a adoção de técnicas que reduzam o consumo de energia,

como, por exemplo, a otimização de algoritmos e o uso de hardware mais eficiente. Isso pode incluir a adoção de técnicas de compressão de modelos, que reduzem o número de parâmetros e operações necessárias para fazer previsões, ou a utilização de processadores especializados, como, por exemplo, as unidades de processamento gráfico (*Graphic Processing Units - GPU)* e as unidades de processamento tensorial (*Tensor Processing Units - TPU)*, que são mais eficientes para determinadas tarefas de IA.

Ao integrar a eficiência energética, a redução de emissões de carbono e o uso consciente de recursos em todos os projetos de IA, essa abordagem busca promover a transparência, permitindo a implementação de medidas para otimizar o desempenho energético e minimizar o impacto ambiental. Além disso, a IA Verde pretende estabelecer um marco para futuras pesquisas e desenvolvimentos, incentivando a criação de modelos de IA mais sustentáveis e eficientes.

Em última análise, a IA Verde visa a criar um campo de pesquisa mais sustentável e responsável, em que os benefícios da IA possam ser alcançados sem comprometer a saúde do planeta. Isso requer uma mudança de paradigma, em que a eficiência e a sustentabilidade se tornem tão importantes quanto o desempenho e a inovação.

3 Trabalhos Relacionados

Este capítulo apresenta trabalhos que lidam com o tema de IA Verde, seja pela apresentação de definições à área, seja por trazer uma perspectiva experimental.

A preocupação com o consumo energético por modelos de AP é uma questão bastante recente e que passou a atrair a atenção da academia e da indústria à medida que tais modelos foram crescendo em números de parâmetros, aumentando cada vez mais o seu desempenho, mas também a sua demanda energética.

Nesse sentido, o trabalho de Schwartz *et al.* [26] introduziu o conceito de IA verde, uma abordagem de pesquisa em inteligência artificial que valoriza a eficiência computacional e promove a redução do uso de recursos. Em contraste, a IA Vermelha busca melhorar a acurácia usando grandes quantidades de poder computacional, sem levar em conta os custos ambientais, econômicos e sociais. Uma proposta dos autores é usar o número total de operações de ponto flutuante como uma medida de eficiência, incentivando os pesquisadores a reportarem essa métrica. Eles defendem que a IA Verde é mais inclusiva e sustentável do que a IA Vermelha, e apontam várias direções de pesquisa para avançar a eficiência na IA.

Em Anthony *et al.* [29], foi proposto o Carbontracker, uma ferramenta que monitora o consumo de energia e estima as emissões de carbono do treinamento de modelos de AP. A ferramenta visa a conscientizar os pesquisadores sobre o impacto ambiental dos seus trabalhos, de modo a incentivá-los a adotar práticas mais sustentáveis. O artigo também avaliou a ferramenta em diferentes arquiteturas de redes neurais e conjuntos de dados para segmentação de imagens médicas e fornece recomendações para diminuir as emissões de carbono. Os autores propuseram e utilizaram uma variedade de métricas para avaliar a eficiência, sob a perspectiva energética, de modelos de Aprendizagem Profunda.

O estudo de Lacoste *et al.* [17], investigou o impacto ambiental do treinamento de redes neurais, mostrando que diversos fatores influenciam a emissão de carbono, como localização do servidor, fonte de energia, duração do treinamento e hardware utilizado. Para auxiliar na compreensão desse impacto, os autores apresentaram a Calculadora de Emissões de Aprendizado de Máquina, ferramenta que estima as emissões de acordo com o tipo de energia e infraestrutura de computação utilizados. Os autores destacam que, embora a calculadora seja apenas uma aproximação, ela serve como ponto de partida para estimar o impacto das escolhas no treinamento de modelos. Os autores esperam que o estudo incentive discussões

e ações para reduzir o impacto ambiental da pesquisa em IA, promovendo um futuro mais sustentável para essa área.

Em [6], Payal Dhar discute o impacto da utilização da IA no meio ambiente. De um lado, a IA tem o potencial de contribuir para a luta contra a crise climática, com projetos de infraestrutura sustentável e previsões meteorológicas mais precisas. Por outro lado, o custo ambiental de operar e treinar modelos de IA é significativo, com um único grande modelo de linguagem podendo emitir tanto carbono quanto 125 voos transcontinentais, de modo que o autor adverte sobre a necessidade de transparência nas emissões de carbono dos modelos de IA, citando como exemplo a ferramenta para o cálculo de emissões de carbono desenvolvida por [17].

Nesse artigo, o autor sugere que a construção de centros de dados em áreas com energia renovável e incentivos fiscais para práticas sustentáveis podem ajudar a mitigar o impacto ambiental da IA, e destaca o papel dos governos como ponto fundamental na criação de regulamentações que promovam a transparência e a sustentabilidade no desenvolvimento da IA. O artigo conclui que uma abordagem colaborativa entre políticas de tecnologia e clima, transparência e consciência do consumidor é essencial para avançar em direção a uma IA mais sustentável e responsável.

Por sua vez, o trabalho de Douwes *et al.* [7] abordou o problema do consumo de energia dos modelos geradores de áudio baseados em AP. Os autores propõem uma nova medida baseada na otimização de Pareto, que considera tanto a qualidade do som sintetizado quanto o consumo de energia do modelo. Eles aplicam essa medida em vários modelos existentes, como SampleRNN, SING, WaveGAN, GANSynth e FloWaveNet, e mostraram como ela pode alterar significativamente os resultados obtidos. Além disso, também realizaram experimentos com um modelo recentemente proposto chamado WaveFlow, e mediram o consumo de energia real de treinamento e inferência para cinco configurações alternativas. Eles usaram um espaço multiobjetivo para representar as soluções ótimas de Pareto e analisaram o compromisso entre qualidade e eficiência energética.

Já o trabalho de Desislavov *et al.* [5] analisou o consumo de energia em modelos de visão computacional e processamento de linguagem natural (PLN), com um foco especial em seus custos de inferência e não apenas no treinamento. Os autores argumentaram que, para um sistema em produção, o custo de inferência ultrapassa o de treino, por causa do fator multiplicativo oriundo dos inúmeros usos em produção. A motivação por trás deste estudo foi entender se o progresso exponencial em alguns paradigmas de IA, como a AP, resulta também

em um aumento exponencial no consumo de energia. Esta é uma questão crucial para se avaliar o impacto ambiental e social da IA, bem como os limites do seu desempenho futuro.

O estudo envolveu a coleta de dados sobre o desempenho de modelos da literatura, o número de parâmetros, o número de operações de ponto flutuante (FLOPs) associadas às inferências, de modo que os autores estimaram o consumo de energia por inferência usando dados de hardware de placas gráficas (GPUs) da NVIDIA. Os resultados mostraram que, embora o consumo de energia dos modelos de ponta ainda cresça exponencialmente para um aumento sustentado no desempenho, o ritmo é mais lento do que o esperado, graças às melhorias algorítmicas e à especialização do hardware. No entanto, os autores terminaram o artigo alertando para o efeito multiplicativo associado à penetração cada vez maior da IA na sociedade, que pode elevar o consumo de energia global.

O trabalho de Lenherr *et al.* [19] discutiu a necessidade de métricas universais para avaliar a sustentabilidade da IA em dispositivos de borda (*edge computing*). Os autores propuseram novas métricas de eficiência de reconhecimento e treinamento para balancear desempenho, complexidade e consumo de energia.

Além disso, o artigo comparou a eficiência de diferentes dispositivos e modelos de AP, mostrando que os aceleradores de borda têm um progresso significativo e podem alcançar precisões próximas aos sistemas baseados em nuvem, mas com um consumo de energia muito menor. O artigo também comparou a eficiência dos modelos de AP em relação às abordagens centralizada e federada, e analisou o ciclo de vida da AP, incluindo o número de vezes que os modelos são usados.

Por fim, os autores abordaram a importância de medir e padronizar a eficiência dos modelos de IA para promover o desenvolvimento de uma IA verde e sustentável.

No trabalho de Strubell *et al.* [27], foram analisados o custo financeiro e ambiental do treinamento de modelos de AP para PLN. Para tal, os autores estimaram o consumo de energia, as emissões de carbono e o custo da computação em nuvem referentes ao treinamento de vários modelos populares de PLN, como Transformer, ELMo, BERT e GPT-2. Também foi realizado um estudo de caso do desenvolvimento de um modelo de ponta para análise sintática e semântica, mostrando que a pesquisa e o ajuste de hiperparâmetros exigem uma quantidade enorme de recursos computacionais. Com base nessas descobertas, eles propuseram recomendações para reduzir os custos e melhorar a equidade na pesquisa e na prática de PLN, tais como: relatar o tempo de treinamento e a sensibilidade aos hiperparâmetros, prover acesso equitativo aos recursos computacionais e priorizar o desenvolvimento de modelos e hardware

eficientes.

Nesse sentido, o trabalho de Li *et al.* [20] propôs uma metodologia para a estimativa da pegada hídrica (*water footprint*) de um modelo de IA por meio do qual calcularam que o treinamento do modelo de linguagem GPT-3, com 175 bilhões de parâmetros, consumiu cerca de 700.000 litros de água limpa. Além disso, o trabalho também mostrou que, a cada 10 a 50 respostas fornecidas pelo *chatbot*, há o consumo de uma quantidade equivalente a uma garrafa de 500 mililitros de água. Ademais, houve até mesmo a publicação de matérias jornalísticas alertando e debatendo esta questão [24, 30, 3].

O trabalho de Breder *et al.* [4] explora o paradoxo da IA: embora possa impulsionar a sustentabilidade ao otimizar processos e auxiliar na predição de eventos climáticos, seu desenvolvimento e operação demandam recursos econômicos, tecnológicos e energéticos significativos, gerando preocupações ambientais. Os autores enfatizaram a necessidade de mensurar e divulgar o consumo energético e as emissões de carbono associadas aos modelos de IA, defendendo uma abordagem mais ecológica e transparente. Eles investigaram como a IA pode simultaneamente mitigar as mudanças climáticas e contribuir para a degradação ambiental devido ao seu alto consumo de energia, exemplificando com o impacto de grandes modelos de linguagem como o GPT-3.

O artigo também apresentou ferramentas como Green-Algorithms, MLCO2, CodeCarbon e Eco2AI, que ajudam a estimar a pegada de carbono e o consumo de energia dos treinos, buscando maior conscientização entre desenvolvedores. Além disso, o estudo destacou que a localização dos centros de dados e a fonte de energia utilizada podem influenciar significativamente as emissões de carbono, com locais que utilizam fontes renováveis apresentando menores emissões. Por fim, os autores concluíram que a transparência e a adoção de práticas eficientes são cruciais para mitigar os impactos negativos da IA e promover um desenvolvimento sustentável.

4 Arquiteturas Escolhidas

Neste capítulo, são apresentadas as redes neurais convolucionais escolhidas para serem utilizadas nos experimentos computacionais conduzidos neste trabalho. Essas redes foram selecionadas por serem amplamente adotadas na literatura, em *benchmarks* conduzidos na área, assim como por possuírem diferentes níveis de complexidade. Tais redes serão descritas a seguir.

4.1 LeNet-5

Proposta por Lecun *et al.* [18], a LeNet-5 foi um modelo pioneiro de CNN para a tarefa de reconhecimento de dígitos manuscritos. Apesar de sua simplicidade, a LeNet-5 é capaz de alcançar ótimo desempenho no reconhecimento de dígitos e em outras tarefas de visão computacional.

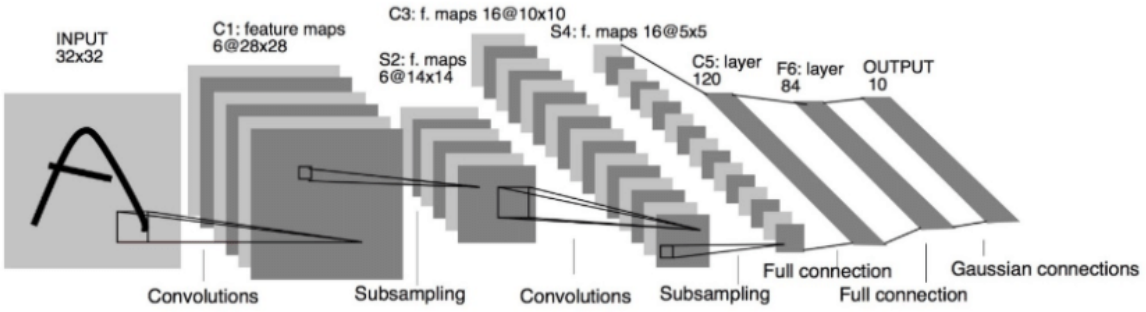

Figura 11: Arquitetura da LeNet-5 (extraída de [18]).

A Figura 11 apresenta a arquitetura da LeNet-5, concebida por seus autores no artigo [18], destacando suas camadas convolucionais, de subamostragem *(pooling)* e totalmente conectadas.

A LeNet-5 foi a primeira CNN proposta e utiliza a técnica de retropropagação para ajustar os pesos dos filtros durante o treinamento. A combinação de camadas convolucionais e de subamostragem permite que a rede aprenda características hierárquicas das imagens, capturando desde bordas simples até padrões mais complexos.

4.2 AlexNet

A utilização da rede AlexNet foi um verdadeiro divisor de águas no campo da IA e da visão computacional. Desenvolvida em 2012 por Krizhevsky *et al.* no trabalho [15], a AlexNet venceu a competição ILSVRC *(ImageNet Large Scale Visual Recognition Challenge)* desse ano, alcançando uma taxa de erro *top*-5 de 15,3%. Dessa forma, essa rede neural foi a responsável por popularizar e impulsionar o uso da AP em tarefas de reconhecimento de imagens e dar origem à atual revolução da IA.

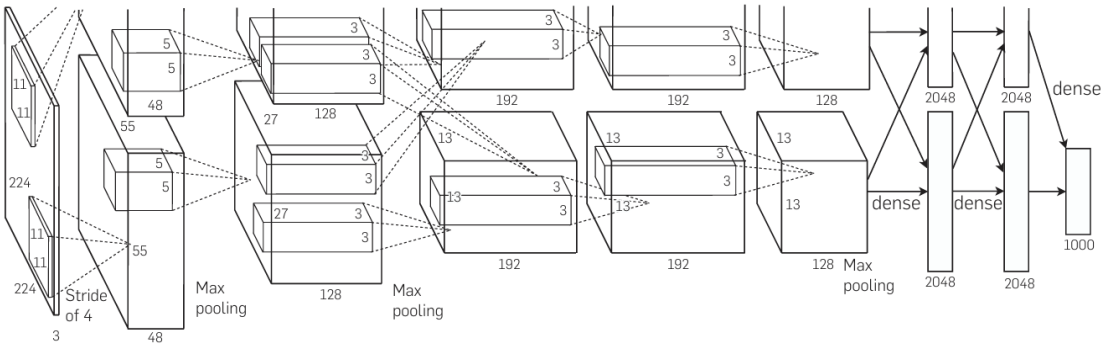

Figura 12: Arquitetura da AlexNet (extraída de [15].)

A arquitetura geral da AlexNet, ilustrada na Figura 12, consiste em oito camadas: cinco camadas convolucionais e três camadas totalmente conectadas, conforme proposto por Krizhevsky *et al.* [15].

A AlexNet inovou ao usar um número maior de camadas para aprender características complexas das imagens, acelerando o treinamento com a função de ativação ReLU. Ela implementou normalização com *pooling* sobreposto, *data augmentation* e *dropout* para melhorar o desempenho e reduzir o *overfitting*, conceito explicado na Seção 2.2.10. Além disso, um dos principais diferenciais foi a realização do treinamento em GPUs, algo inovador para a época. Isso permitiu reduzir drasticamente a taxa de erro em comparação com outros modelos. Esses fatores surpreenderam a comunidade científica e mostraram o grande potencial das redes neurais profundas quando treinadas com grandes conjuntos de dados e maior poder computacional.

4.3 ResNet

A ResNet é uma arquitetura de redes neurais convolucionais, introduzida em 2015 por He *et al.*, no trabalho [11]. Desenvolvida para mitigar algumas dificuldades de treinamento de redes neurais profundas, a ResNet introduziu o conceito de "conexões residuais" *(residual connections)*, também conhecidas como "atalhos" *(skip connections)*.

A parte principal desta rede é composta de múltiplos estágios e blocos, cada um contendo diversos tipos de camadas, formatos de saída e diferentes quantidades de parâmetros. A Figura 13 ilustra a estrutura de um bloco residual típico, que é composto por duas ou três camadas convolucionais. A ideia central da arquitetura ResNet é implementada por meio da adição de uma conexão de "atalho" *(skip connection)*, visualizada em azul na figura, que contorna algumas camadas. Essas conexões permitem a propagação eficiente do gradiente durante o treinamento, possibilitando a construção de redes mais profundas sem o problema do desaparecimento do gradiente.

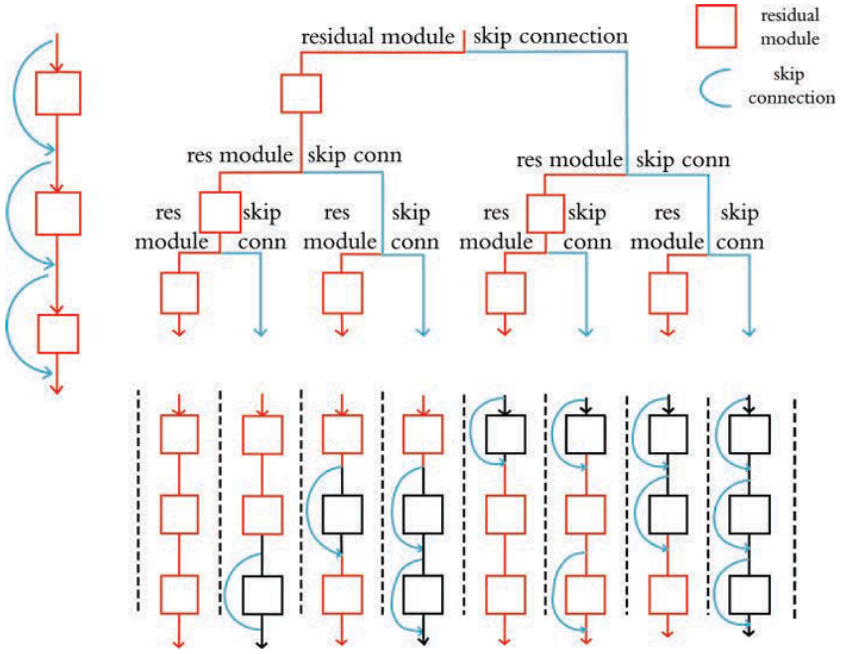

Figura 13: Blocos Residuais da ResNet (extraída de [16].)

A ResNet, com sua abordagem revolucionária resultou em avanços significativos no campo da visão computacional, estabelecendo novos recordes de desempenho em tarefas como classificação de imagens e detecção de objetos. A ResNet venceu a competição ILSVRC de 2015 com uma taxa de erro *top*-5 de 3,57% no conjunto de testes do ImageNet. Por conta dos resultados obtidos, a ResNet se tornou uma referência para problemas de visão

computacional.

4.4 GoogLeNet

A GoogLeNet (ou *Inception*) é uma CNN que ganhou destaque por seu desempenho na competição ILSVRC de 2014. Ela introduziu o conceito de "módulos de *Inception*", que permitem à rede ser mais profunda e larga, mas de forma computacionalmente eficiente.

Segundo o trabalho de Szegedy *et al.* [28], os módulos *Inception* combinam várias camadas de convolução com filtros de diferentes tamanhos (1×1, 3×3, 5×5), simultaneamente, para criar uma estrutura densa e interligada dentro da rede neural. Cada uma dessas convoluções captura diferentes níveis de detalhes e características da imagem.

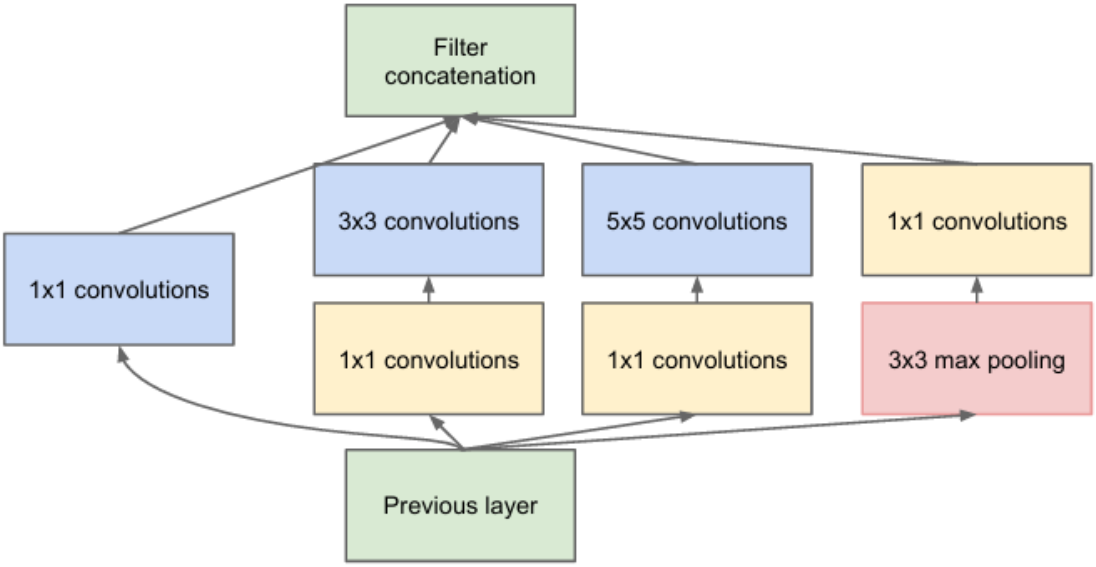

Figura 14: Módulo Inception com redução de dimensionalidade (extraída de [28]),

A Figura 14 ilustra o uso de convoluções 1×1 nos módulos Inception para reduzir o número de canais antes das convoluções 3×3 e 5×5, que são mais custosas. Isso diminui a complexidade computacional e o número de parâmetros, permitindo que a rede seja mais profunda e larga sem aumentar drasticamente o custo computacional.

4.5 MobileNet

A MobileNet é uma família de CNNs desenvolvidas pelo Google, projetadas especificamente para aplicações em ambientes com restrições de recursos, como dispositivos móveis, Internet

das Coisas (*Internet of Things* - IoT) e IA embarcada.

No artigo [12], os autores descrevem uma arquitetura de rede eficiente e um conjunto de dois hiperparâmetros (tamanho e latência), que permitem que um desenvolvedor escolha especificamente uma pequena rede que corresponda às restrições de recursos para sua aplicação.

As MobileNets são baseadas em uma arquitetura que usa convoluções separáveis em profundidade (*Depthwise Separable*) para construir redes neurais profundas leves. Os blocos *Depthwise Separable* são a parte central da arquitetura da MobileNet. Cada bloco é composto por duas camadas convolucionais:

Convolução em Profundidade (*Depthwise Convolution*): Nesta etapa é aplicado um único filtro para cada canal de entrada separadamente. Ou seja, se a entrada tem três canais, são aplicados três filtros.

Convolução Pontual (*Pointwise Convolution*): Após a convolução em profundidade, cada conjunto de mapas de características resultantes é combinado usando convoluções 1×1. A convolução pontual combina linearmente os mapas de características gerados pela convolução em profundidade.

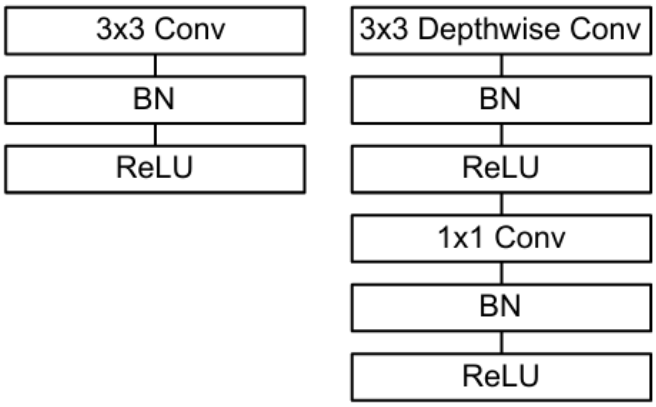

Figura 15: Esquerda: Convolução padrão. Direita: Convolução separável em profundidade. BN (*Batch Normalization*) (extraída de [28]).

A Figura 15 ilustra a diferença entre uma convolução padrão, à esquerda, que aplica um filtro 3×3 a todos os canais de entrada simultaneamente, e uma convolução separável em profundidade, à direita, que aplica um único filtro 3×3 para cada canal de entrada separadamente, combinando as saídas desses canais em uma convolução 1×1. A Normalização em Lote (BN) é aplicada após a convolução, seguida pela função de ativação ReLU.

A utilização dessas redes neurais leves possibilita um excelente equilíbrio entre desempenho e eficiência, viabilizando a execução de modelos robustos mesmo em hardware com recursos limitados.

5 Experimentos Computacionais

Neste capítulo, detalham-se os experimentos conduzidos para avaliar comparativamente arquiteturas de redes neurais amplamente utilizadas aplicadas à visão computacional, tanto sob a perspectiva de desempenho quanto sob a perspectiva de eficiência energética e custo computacional.

5.1 Metodologia

5.1.1 Ambiente Computacional

O ambiente computacional utilizado nos experimentos foi o seguinte:

Processador: Intel® Core™, modelo i5-11400, de 11ª geração e 2,60 GHz de frequência.

Memória: 32 GB de memória RAM DDR4.

Processamento Gráfico: Placa NVIDIA GeForce RTX 3060.

Sistema Operacional: Ubuntu 22.04 LTS.

A implementação dos modelos foi feita na linguagem de programação Python na versão 3.11, usando-se a biblioteca PyTorch[3] na versão 2.2. Os códigos desenvolvidos nesta pesquisa, assim como o resultado dos experimentos, estão disponíveis publicamente em um repositório criado no GitHub[4].

5.1.2 Conjunto de Dados

O conjunto de dados escolhido para os experimentos foi o *MNIST (Modified National Institute of Standards and Technology)*, proposto por LeCun *et al.* no trabalho [18]. Este conjunto de dados consiste em uma coleção de dígitos manuscritos coletados originalmente pelo *National Institute of Standards and Technology (NIST)*, a partir de formulários preenchidos por trabalhadores do censo dos Estados Unidos e estudantes do ensino médio. A Figura 16 ilustra alguns exemplos pertencentes ao conjunto MNIST. Essa versão do MNIST, especificamente,

[3] https://pytorch.org/blog/pytorch2-2/
[4] https://github.com/viferraro/TCC

foi criada a partir de um subconjunto dos conjuntos de dados NIST originais, com o objetivo de facilitar o treinamento e teste de algoritmos de AM.

O conjunto de dados MNIST é composto por um total de 70.000 imagens, divididas em 60.000 para treinamento e 10.000 para teste. Cada imagem representa um dígito de 0 a 9, em tons de cinza e com resolução de 28×28 pixels.

Para este trabalho, o conjunto de treinamento original do MNIST foi dividido em dois subconjuntos: treino e validação. A divisão foi realizada utilizando a função `random_split` do PyTorch, alocando 80% dos dados (48.000 imagens) para treinamento e os 20% restantes (12.000 imagens) para validação. Essa divisão permitiu o monitoramento do desempenho do modelo durante o treinamento e a seleção de hiperparâmetros que melhor generalizam para dados não vistos.

No final, a distribuição dos dados ficou assim:

- Treino: 48.000 imagens;
- Validação: 12.000 imagens;
- Teste: 10.000 imagens (conjunto original de teste do MNIST, não alterado).

Figura 16: Exemplos normalizados do conjunto de dados MNIST (extraída de [18]).

Ainda que conjuntos de dados maiores e mais complexos tenham surgido desde então, como o conjunto de dados ImageNet, destacado no artigo [15], o conjunto de dados MNIST ainda permanece bastante útil e amplamente utilizado porque os dígitos manuscritos são

diversos e detalhados o suficiente para serem desafiadores, mas ainda assim permitem que algoritmos de AM os classifiquem com alto desempenho [16].

5.1.3 Arquiteturas das Redes Utilizadas

As seções seguintes apresentam os detalhes de implementação das CNNs utilizadas nos experimentos, cujas descrições foram apresentadas no Capítulo 4. Assim sendo, serão abordados os componentes específicos das redes e suas configurações.

5.1.4 LeNet-5

A arquitetura da rede utilizada neste trabalho, baseada na LeNet-5 e representada na Figura 17, é uma adaptação do modelo original. A camada de entrada recebe imagens em escala de cinza com dimensões 28x28x1 (altura, largura, canais). O primeiro bloco convolucional consiste em uma camada convolucional (`Conv2d`) com *kernel* 3×3, seguida por ativação ReLU. O segundo bloco convolucional contém uma camada convolucional (`Conv2d`) com *kernel* 3×3, uma ativação ReLU e *max pooling*. Após isso, há uma camada de *dropout* de 25%. Os mapas de características resultantes são então convertidos em um vetor unidimensional (*flatten*) e processados por duas camadas totalmente conectadas de neurônios, com ativação ReLU e Softmax, respectivamente), com *dropout* (50%) entre elas. O modelo possui um total de 609.354 parâmetros, todos treináveis.

5.1.5 AlexNet

A Figura 18 detalha a arquitetura da rede convolucional utilizada neste trabalho, que consiste em um modelo adaptado da AlexNet [15] para o conjunto MNIST.

A rede utilizada recebe imagens em escala de cinza e aplica uma sequência de camadas convolucionais (`Conv2d`), com ativação ReLU e *kernel* 3×3, intercaladas com camadas de *max pooling* para redução dimensional. Após essas etapas, aplica-se *pooling* sobreposto (*adaptive average pooling*) para reduzir ainda mais a dimensionalidade dos mapas de características obtidos. O resultado é então convertido em um vetor unidimensional *(flatten)* e processado por três camadas totalmente conectadas, com *dropout* aplicado nas duas primeiras. A última camada utiliza a função de ativação Softmax para a classificação final. O modelo possui 23.271.114 parâmetros treináveis, representando uma diferença considerável em relação ao número de parâmetros da LeNet-5 (609.354 parâmetros).

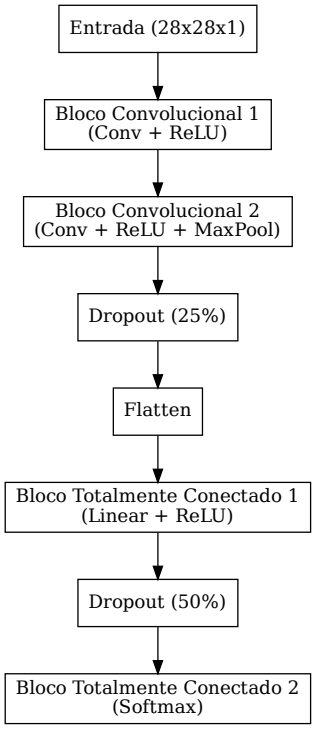

Figura 17: Detalhes da arquitetura da rede baseada na LeNet-5.

5.1.6 ResNet

Utiliza-se neste estudo a rede ResNet-34, uma das variantes da ResNet que possui 34 camadas. Um resumo de sua arquitetura é apresentado na Figura 19.

A rede é organizada em estágios, cada um contendo um certo número de blocos de construção:

Camada Inicial: Consiste em uma camada convolucional (`Conv2d`) com 64 filtros, seguida por normalização em lote (BN), ativação ReLU e *max pooling* (`MaxPool2d`). Esta etapa inicial reduz as dimensões espaciais da entrada.

Blocos Residuais (Bloco 1, Bloco 2, Bloco 3 e Bloco 4): São a parte central da ResNet. Cada bloco é composto por um conjunto de camadas convolucionais, normalização em lote e ativação ReLU, organizadas em uma estrutura que permite a criação de conexões residuais (ou "atalhos"). Esses atalhos somam a saída das convoluções à entrada do bloco, facilitando o treinamento de redes profundas. A notação '(xn)' indica a repetição do bloco n vezes.

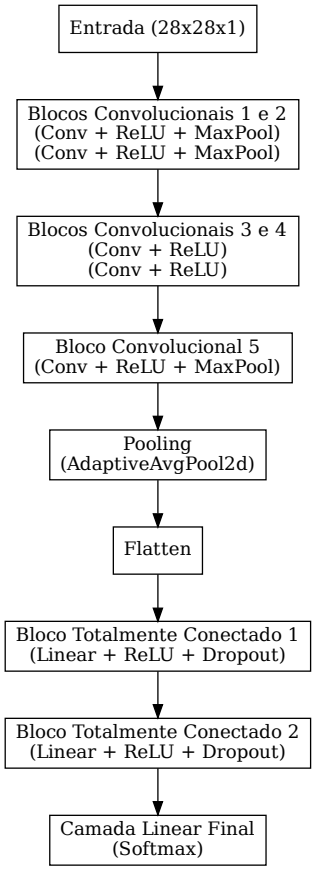

Figura 18: Detalhes da arquitetura da rede baseada na AlexNet.

BasicBlock: É a unidade básica dos blocos residuais na ResNet-34. Ele consiste em duas camadas convolucionais 3×3 com o mesmo número de filtros.

Projeção (proj): Em alguns blocos (Bloco 2 e Bloco 4), quando o número de canais aumenta, uma camada convolucional 1×1 é usada no atalho para ajustar as dimensões e permitir a soma. Essa camada é chamada de "projeção" e adiciona parâmetros ao bloco.

Subamostragem (*pooling*) e Classificação: Após os blocos residuais, uma camada de *pooling* médio adaptativo (`AdaptiveAvgPool2d`) é aplicado para reduzir as dimensões espaciais para 1×1. Finalmente, uma camada totalmente conectada com dez saídas, seguida por uma função de ativação Softmax, é usada para a classificação final. O modelo possui um total de 21.292.042 parâmetros treináveis.

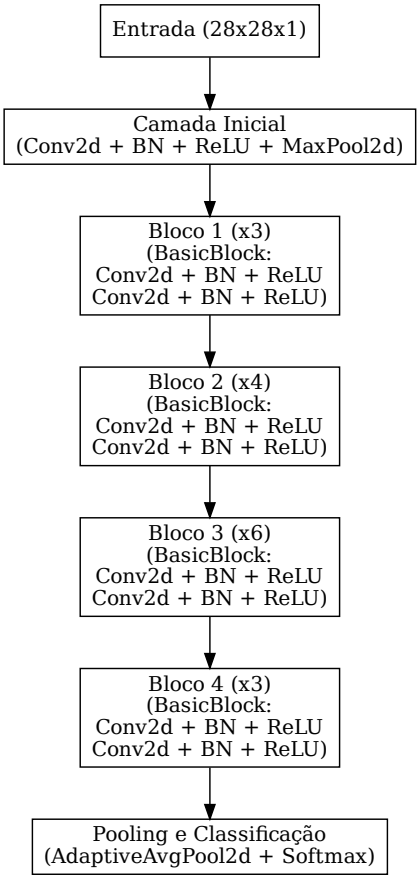

Figura 19: Detalhes da arquitetura da rede baseada na ResNet34.

5.1.7 GoogLeNet

Para este estudo, utilizamos uma versão da GoogleNet, representada na Figura 20. Sua arquitetura caracteriza-se por diversos blocos (ou "estágios"), cada um composto de diferentes tipos de camadas que contribuem para a eficácia no processamento e na análise das imagens.

Camada Inicial: Consiste em uma camada convolucional (`Conv2d`) com 64 filtros, seguida por uma camada de *max pooling* (`MaxPool2d`).

Bloco 1: É formado por duas camadas convolucionais e uma camada de *max pooling*.

Bloco 2 e Bloco 3 (com ramificações): Estes blocos incluem múltiplas camadas de convoluções, além de uma camada de *max pooling*. A soma dos parâmetros das camadas convolucionais envolvidas é considerada, enquanto a camada de *max pooling* não adiciona novos parâmetros.

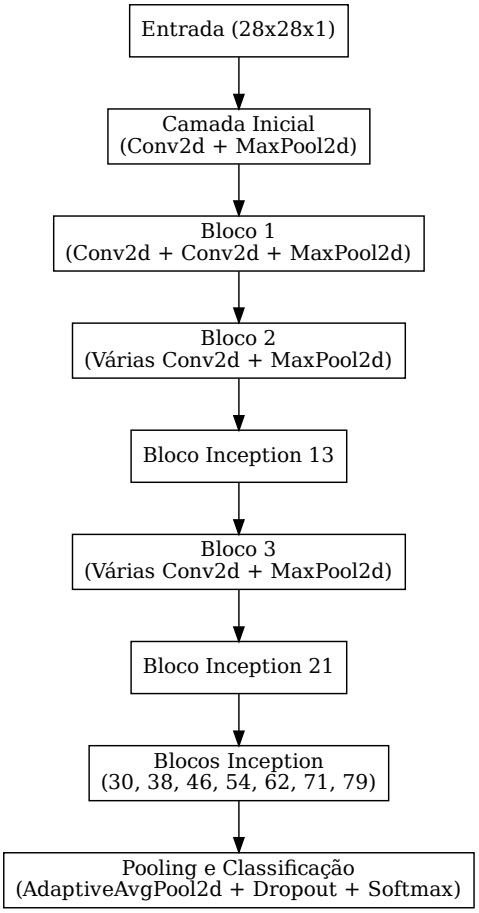

Figura 20: Detalhes da arquitetura da rede baseada na GoogLeNet.

Bloco Inception 13 e Bloco Inception 21: Esses blocos Inception utilizam múltiplas ramificações dentro da camada Inception. Da mesma forma, a soma dos parâmetros das camadas convolucionais envolvidas é considerada.

Blocos Inception 30, 38, 46, 54, 62, 71 e 79: Cada um desses blocos tem uma saída de diferentes tamanhos, e contribuem com a soma dos parâmetros das camadas convolucionais envolvidas.

Subamostragem *(Pooling)* e Classificação: Após os blocos Inception, a rede aplica uma camada de *pooling* médio adaptativo (`AdaptiveAvgPool2d`) para reduzir as dimensões espaciais para 1×1. Em seguida, utiliza uma camada de *dropout* para regularização, prevenindo o *overfitting*. Finalmente, uma camada totalmente conectada com dez saídas, seguida por uma função de ativação Softmax, é empregada para a classificação final. A arquitetura do modelo totaliza 5.977.530 parâmetros treináveis.

5.1.8 MobileNet

Neste trabalho utilizamos uma versão básica e adaptada da MobileNet. A Figura 21 ilustra a arquitetura utilizada.

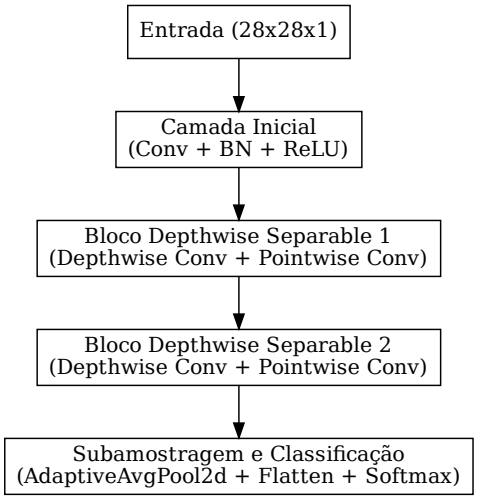

Figura 21: Detalhes da arquitetura da rede baseada na MobileNet.

Camada Inicial: Consiste em uma camada convolucional (`Conv2d`) com 1 único filtro (3×3), seguida por uma camada de normalização em lote (`BatchNorm2d`) e função de ativação ReLu.

Bloco *Depthwise Separable* 1 : Este bloco, cuja estrutura foi detalhada na Seção 4.5, recebe a saída da camada inicial (32 canais) e aplica uma convolução *depthwise* com 32 grupos, seguida por uma convolução *pointwise* que expande para 64 canais.

Bloco *Depthwise Separable* 2 : Este bloco recebe a saída do Bloco 1 (64 canais) e aplica uma convolução *depthwise* com 64 grupos, seguida por uma convolução *pointwise* que expande para 128 canais.

Subamostragem (*Pooling*) e Classificação: Após os blocos *depthwise separable*, um *pooling* médio adaptativo (`AdaptiveAvgPool2d`) com saída 1×1 é aplicado para reduzir as dimensões espaciais para um vetor de características de 128 elementos. Em seguida, uma camada *flatten* transforma este vetor de características em um vetor unidimensional. Finalmente, uma camada totalmente conectada com dez saídas, seguida por uma função

de ativação Softmax, é usada para a classificação final. O modelo possui um total de 24.618 parâmetros treináveis.

5.1.9 Implementação e Hiperparâmetros utilizados

Os hiperparâmetros são valores escolhidos *a priori* pelo projetista da rede e que impactam no seu treinamento e respectivo desempenho. A escolha dos hiperparâmetros é fundamental para o treinamento eficaz de redes neurais. Esta seção descreve os hiperparâmetros utilizados em todos os experimentos realizados, fornecendo uma base para a compreensão e reprodutibilidade dos resultados.

Para o treinamento dos modelos de AP, utilizaram-se o conjunto de dados de treinamento e de validação. O processo de treinamento envolve a alimentação iterativa do modelo com esses dados, ajustando seus parâmetros internos para minimizar a diferença entre as previsões do modelo e os valores reais, tal como descrito na Seção 2.2.3. Esse processo é influenciado por hiperparâmetros como o número de épocas e o tamanho do lote. Além disso, empregou-se o otimizador Adam, um algoritmo de otimização que corresponde ao estado da arte para ajuste dos pesos da rede, e a função de custo correspondeu à entropia cruzada (*Cross-Entropy*), adequada para problemas de classificação.

Tamanho do Lote (*Batch Size*): 32.

O *batch size*, que define o número de exemplos considerados em cada rodada de treinamento, foi definido como 32. A escolha deste valor prioriza a estabilidade da convergência e o menor consumo de memória da GPU, mesmo com um possível aumento no tempo de treinamento.

Número Máximo de Épocas (*Epochs*): 50.

Cinquenta épocas foram definidas como o limite máximo de treinamento, com a utilização da técnica *Early Stopping* (Seção 2.2.10) para evitar *overfitting*.

Especificamente, utilizamos a função `EarlyStopping` com os seguintes parâmetros: `patience` $= 5$ e `min_delta` $= 0.001$.

O valor do parâmetro `patience` significa que o treinamento será interrompido se não houver melhoria no desempenho do modelo no conjunto de validação por cinco épocas consecutivas. Já o valor do parâmetro `min_delta` define o limiar mínimo de melhoria na função de custo para que uma época seja considerada uma melhoria. Ou seja, apenas

melhorias superiores a $0,001$ serão consideradas relevantes para evitar a parada precoce do treinamento.

Otimizador: `Adam(lr=0.001, weight_decay=`1.0×10^{-4}`)`.

O otimizador Adam (*Adaptive Moment Estimation*) é um algoritmo de otimização que ajusta os pesos de uma rede neural durante o treinamento (Seção 2.2.9).

Parâmetros utilizados no otimizador:

- Taxa de Aprendizagem (*Learning Rate* - `lr`): Controla o tamanho do passo que o otimizador dá na direção do mínimo da função de custo durante o treinamento. O valor de $0,0001$ é uma taxa relativamente baixa e ajuda a garantir uma convergência mais estável e evitar oscilações durante o treinamento, apesar de levar mais tempo para chegar a um mínimo local.
- Decaimento de Pesos (`weight_decay`) : Adiciona uma penalidade à função de custo, fazendo com que pesos com valores altos decaiam em direção a zero durante o treinamento para prevenir o *overfitting*.

Função de Perda: Custo de Entropia Cruzada (Seção 2.2.8).

Controlador da Taxa de Aprendizagem *scheduler*: `ReduceLROnPlateau(patience=3, factor=0.1)`.

O `ReduceLROnPlateau` é um controlador oferecido pela biblioteca PyTorch, ele ajusta dinamicamente a taxa de aprendizagem durante o treinamento. Ele monitora uma métrica (no caso, a perda no conjunto de validação) e, quando essa métrica para de melhorar por um certo número de épocas (definido pelo parâmetro `patience`), ele reduz a taxa de aprendizagem por um fator definido pelo parâmetro `factor`.

5.1.10 Métricas para Avaliação do Desempenho

Esta seção dedica-se à apresentação das métricas utilizadas para avaliar o desempenho das redes neurais nos experimentos computacionais conduzidos. A adoção dessas métricas, comuns na literatura, possibilita a interpretação detalhada do comportamento das redes, com foco em aspectos como exatidão, desempenho e capacidade de generalização.

As métricas comumente adotadas para avaliação de um modelo que aborda um problema de classificação correspondem a: acurácia, precisão e revocação [1]. A base para o cálculo

dessas métricas é a matriz de confusão, uma tabela que oferece uma visão detalhada do desempenho do modelo, apresentando a distribuição de verdadeiros positivos, verdadeiros negativos, falsos positivos e falsos negativos, permitindo uma análise mais profunda dos tipos de erros cometidos. Essas métricas serão detalhadas a seguir.

Matriz de Confusão

A matriz de confusão é uma ferramenta útil para avaliar o desempenho de um algoritmo de classificação. Ela permite visualizar o desempenho do algoritmo, facilitando a compreensão em termos de previsões corretas e incorretas feitas.

		actual *y*	
		1	0
predicted *y*	1	True positive	False positive
	0	False negative	True negative

Figura 22: Exemplo de matriz de confusão (extraída de [16]).

A Figura 22 ilustra um exemplo de matriz de confusão para um problema classificação binária que é organizada em 4 setores principais. A diagonal principal representa os exemplos corretamente preditos pelo modelo, à medida que a diagonal secundária representa os exemplos incorretamente preditos pelo modelo.

Verdadeiros Positivos (*True Positive*) - VP: Representam o número de amostras que foram corretamente classificadas como pertencentes à classe positiva.

Falsos Positivos (*False Positive*) - FP: São o número de amostras que foram incorretamente classificadas como pertencentes à classe positiva, quando na verdade pertencem à classe negativa.

Verdadeiros Negativos (*True Negative*) - VN: Representam o número de amostras que foram corretamente classificadas como pertencentes à classe negativa.

Falsos Negativos (*False Negative*) - FN: São o número de amostras que foram incorretamente classificadas como pertencentes à classe negativa, quando na verdade pertencem à classe positiva.

Acurácia

A acurácia *(accuracy)* é uma métrica fundamental em diversas áreas do conhecimento, especialmente nas ciências exatas e aplicadas, em que a medição exata de fenômenos é essencial para garantir a confiabilidade dos resultados obtidos.

"Para avaliar a performance de um algoritmo de Aprendizagem de Máquina, é necessário projetar uma medida quantitativa de seu desempenho." [9]. Modelos de AP lidam com grandes quantidades de dados complexos e a acurácia é fundamental para avaliar o desempenho de um modelo, indicando o quão bem o modelo aprende e generaliza esse aprendizado para dados não vistos.

$$\text{Acurácia} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{FP} + \text{VN} + \text{FN}} \tag{9}$$

A acurácia representa a exatidão dos resultados de uma medição em relação ao valor verdadeiro ou esperado. Em termos matemáticos, conforme representado na Equação 9, a acurácia pode ser definida como a razão entre as predições corretas em relação ao total de predições feitas por um modelo [23].

Em cenários com classes desbalanceadas, a acurácia pode ser enganosa, mascarando um desempenho ruim em classes minoritárias. Uma forma de resolver esse problema é complementar a acurácia com outras métricas, como precisão, revocação e medida-F1 [9].

Precisão

A precisão (*precision*) é uma métrica utilizada para avaliar a qualidade das predições positivas de um modelo de classificação, essa métrica mede o quanto um modelo prediz corretamente exemplos de uma determinada classe.

$$\text{Precisão} = \frac{\text{VP}}{\text{VP} + \text{FP}} \tag{10}$$

A Precisão é calculada dividindo-se a quantidade de predições feitas corretamente para uma classe pela quantidade de exemplos que o modelo classificou como sendo dessa

classe, conforme a Equação 10, ou seja, a precisão indica a proporção de exemplos classificados como positivos que são, de fato, positivos. Uma alta precisão significa que a maioria das previsões positivas do modelo são corretas, o que é fundamental em cenários em que falsos positivos podem resultar em custos elevados ou consequências indesejadas, como em sistemas de diagnósticos médicos. "Para qualquer aplicação, precisamos nos perguntar quais podem ser as consequências desses erros no mundo real." [23].

Revocação

A revocação *(recall)* mede a capacidade do modelo de identificar corretamente todos os exemplos reais de uma determinada classe presentes no conjunto de dados. Matematicamente, é calculada dividindo-se a quantidade de predições feitas corretamente para uma classe pela quantidade de exemplos existentes dessa classe, como representado na Equação 11.

$$\text{Revocação} = \frac{\text{VP}}{\text{VP + FN}} \tag{11}$$

Medida-F1

A medida-F1 (ou *F1-score*) é uma métrica que busca o equilíbrio entre precisão e revocação, correspondendo à média harmônica das duas. O seu cálculo é exibido na Equação 12.

$$\text{Medida-F1} = 2 \times \frac{\text{Precisão} \times \text{Revocação}}{\text{Precisão} + \text{Revocação}} \tag{12}$$

A média aritmética (simplesmente somar os valores e dividir por dois) pode ser enganosa quando há um desequilíbrio entre precisão e revocação. A média harmônica penaliza mais os valores baixos, o que a torna mais adequada para avaliar o desempenho em problemas de classificação. Ela é particularmente útil em problemas com classes desbalanceadas ou quando os custos de falsos positivos e falsos negativos são diferentes. Ela busca um equilíbrio entre a capacidade do modelo de fazer previsões

positivas corretas (precisão) e sua capacidade de encontrar todos os casos positivos reais (revocação).

5.1.11 Métricas para Eficiência e Custo Computacional

Diante da crescente preocupação com o impacto ambiental de modelos de AP, torna-se imprescindível uma análise que vá além das métricas de desempenho tradicionais. O presente estudo adota um conjunto abrangente de métricas, selecionadas a partir de um levantamento bibliográfico de trabalhos de referência em IA Verde.

As métricas de eficiência energética e custo computacional adotadas nesta pesquisa abrangem: consumo de energia, pegada de carbono, FLOPs, número de parâmetros, tempo de treinamento e tempo de inferência. A seleção destas métricas, embasada no levantamento bibliográfico em IA Verde, visa garantir uma avaliação completa e relevante do impacto ambiental dos modelos de AP, de acordo com as práticas e recomendações mais recentes na área. Essas métricas serão abordadas a seguir.

Consumo de Energia da GPU

Esta métrica mede diretamente o consumo elétrico dos componentes de processamento gráfico durante a execução das tarefas. A medição foi realizada utilizando a Biblioteca de Gerenciamento NVIDIA[5] (*NVIDIA Management Library* - NVML) que fornece dados precisos e em tempo real de diversos atributos de componentes NVIDIA, incluindo o consumo de energia. A unidade de medida adotada pela NVML é o miliwatt (mW), o que permite um monitoramento detalhado e preciso da energia consumida. Tais funcionalidades contribuem para uma gestão energética mais eficiente e sustentável das GPUs, possibilitando um uso mais responsável da energia.

Pegada de Carbono

Esta métrica estima o impacto ambiental do consumo de energia da GPU, convertendo-o em equivalente de dióxido de carbono (CO_2) liberado na atmosfera. Ela quantifica as emissões de gases de efeito estufa associadas à produção e ao uso da energia elétrica

[5] https://developer.nvidia.com/nvidia-management-library-nvml

consumida pela GPU. A pegada de carbono oferece uma perspectiva mais ampla do impacto ambiental, permitindo a comparação entre diferentes modelos e configurações de hardware.

A pegada de carbono foi mensurada com o auxílio do Carbontracker[6], uma ferramenta desenvolvida por Wolff *et al.* [29] para o monitoramento do consumo energético e estimativa das emissões de carbono em modelos de AP. O Carbontracker fornece as emissões estimadas em gramas de CO_2 equivalentes, permitindo a conversão para unidades mais compreensíveis, como a distância equivalente percorrida por um veículo. Essa contextualização auxilia na interpretação do impacto ambiental dos modelos avaliados. A Figura 23 ilustra a saída impressa da ferramenta para um modelo baseado na ResNet34, em que é possível ver o tempo tomado no treinamento, a quantidade de energia gasta e as estimativas de emissões de carbono produzidas.

A ferramenta oferece ainda opções de configuração para monitorar diferentes componentes do sistema (e.g., GPU e CPU), ajustar a frequência das medições e definir o comportamento em caso de erros.

```
2024-11-22 10:53:42 - CarbonTracker: The following components were found: GPU with device(s) NVIDIA
GeForce RTX 3060. CPU with device(s) cpu:0.
2025-01-22 11:21:38 - CarbonTracker: Average carbon intensity during training was 98.35 gCO2/kWh at
detected location: Rio de Janeiro, Rio de Janeiro, BR.
2024-11-22 11:21:38 - CarbonTracker:
Actual consumption for 50 epoch(s):
        Time:   0:29:06
        Energy: 0.065715488050 kWh
        CO2eq:  6.463002590422 g
        This is equivalent to:
        0.060120954330 km travelled by car
2024-11-22 11:21:38 - CarbonTracker: Finished monitoring.
```

Figura 23: Exemplo de saída do Carbontracker para um modelo baseado na ResNet34.

Número Total de Parâmetros

Esta métrica reflete a complexidade e a capacidade de aprendizado do modelo. Cada parâmetro representa um valor ajustável que o modelo otimiza durante o treinamento para minimizar o erro e melhorar suas previsões. Um número maior de parâmetros geralmente indica que o modelo possui uma maior capacidade de representar padrões

[6]https://github.com/lfwa/carbontracker

complexos nos dados, mas também pode resultar em maior consumo de recursos computacionais, tanto durante o treinamento quanto na inferência. É importante considerar o número de parâmetros em relação ao desempenho do modelo e aos recursos disponíveis.

Número de FLOPs

Esta métrica quantifica o número de operações de ponto flutuante realizadas por um modelo durante a propagação para frente. FLOPs correspondem a uma medida de desempenho computacional, indicando a capacidade do hardware em realizar cálculos numéricos. Um maior número de FLOPs geralmente indica maior capacidade de processamento e, consequentemente, maior consumo de energia. No contexto deste trabalho, o número de FLOPs é utilizado para analisar a complexidade computacional dos modelos e sua relação com o consumo de energia e tempo de execução. Para calcular o número de FLOPs, utilizamos a biblioteca thop[7].

Tempo de Treino

A eficiência do treinamento de redes neurais é um aspecto crucial para o desenvolvimento de modelos escaláveis e viáveis. Normalmente, a maior parte do gasto de tempo e de custo energético de um modelo de AP está na fase de treinamento e, por isso, o tempo de treino de tais modelos desempenha um papel fundamental na viabilidade de sua implantação.

Para avaliar o tempo de treinamento, cada rede foi treinada dez vezes consecutivas, com cada treinamento consistindo em 50 épocas. O tempo total de treinamento para cada execução foi registrado em segundos. Essa abordagem de múltiplas execuções visa a contemplar o caráter estocástico do processo de treinamento de redes neurais, como a inicialização de pesos, a separação em lotes e a ordem de apresentação desses. O tempo de treinamento apresentado para cada modelo representa a média dos tempos das suas dez execuções.

[7] https://pypi.org/project/thop/

Tempo de Inferência

Embora o treinamento de um modelo de AP demande um grande esforço computacional e tempo de treino significativo, ele é realizado um número limitado de vezes. A inferência, por outro lado, é tipicamente executada inúmeras vezes, principalmente em aplicações de uso massivo, como apontam Desislavov *et al.* [5]. Nesse sentido, o tempo de inferência é um fator que deve ser considerado para a viabilidade desses modelos.

Para avaliar o tempo de inferência, foram realizadas dez inferências consecutivas em um modelo selecionado para cada arquitetura de rede. O tempo de inferência apresentado para cada modelo representa a média dos tempos das dez inferências realizadas. O tempo médio de inferência dos modelos foi registrado em segundos.

5.2 Resultados

5.2.1 Treinamento

Nesta seção, os resultados da fase de treinamento para as redes neurais adotadas são apresentados em detalhes. Durante o treinamento, a cada época, a acurácia e a função de custo (Seção 2.2.8) são monitoradas tanto no conjunto de treinamento quanto no conjunto de validação.

Para garantir uma avaliação robusta e identificar a melhor configuração, dez execuções do treinamento da rede com inicializações aleatórias diferentes foram realizadas. Para cada rede neural, a seleção do melhor modelo baseia-se na melhor acurácia alcançada no conjunto de validação, com o objetivo de identificar o modelo com melhor capacidade de generalização para dados não vistos durante o treinamento, preparando-o para a etapa de inferência no conjunto de teste, cujos resultados serão apresentados na Seção 5.2.3.

Modelos baseados na arquitetura LeNet-5

A Tabela 1 apresenta as principais métricas de treinamento para os dez modelos que foram treinados com a arquitetura de rede baseada na LeNet-5. A linha destacada indica o modelo 6 como o melhor com base na maior acurácia alcançada no conjunto de validação, ainda que a diferença no valor da acurácia para os demais modelos tenha sido marginal.

Tabela 1: Desempenho dos modelos baseados na LeNet-5 na fase treino.

Modelo	**Perda Treino**	**Acurácia Treino**	**Perda Validação**	**Acurácia Validação**
Modelo 1	1,469018	0,993083	1,477195	0,985167
Modelo 2	1,468757	0,993448	1,476323	0,985917
Modelo 3	1,467569	0,994451	1,475254	0,986944
Modelo 4	1,466853	0,995057	1,474785	0,987313
Modelo 5	1,466903	0,995021	1,474891	0,987283
Modelo 6	1,467106	0,994861	1,474844	0,987389
Modelo 7	1,467092	0,994908	1,474927	0,987238
Modelo 8	1,467164	0,994849	1,474950	0,987219
Modelo 9	1,467074	0,994938	1,474897	0,987343
Modelo 10	1,467116	0,994900	1,475038	0,987167
Média	1,467466	0,994191	1,475710	0,987048
Desvio Padrão	0,000769	0,000721	0,000801	0,000830

Modelos baseados na arquitetura AlexNet

Os resultados do treinamento dos modelos baseados na arquitetura AlexNet são apresentados na Tabela 2. Dentre esses, a despeito da diferença marginal, o modelo 1 obteve a maior acurácia no conjunto de validação.

Tabela 2: Desempenho dos modelos baseados na AlexNet na fase de treino.

Modelo	**Perda Treino**	**Acurácia Treino**	**Perda Validação**	**Acurácia Validação**
Modelo 1	0,000728	0,999875	0,023939	0,994333
Modelo 2	0,001011	0,999802	0,025768	0,994042
Modelo 3	0,000951	0,999833	0,025522	0,993917
Modelo 4	0,001005	0,999833	0,025820	0,993958
Modelo 5	0,001133	0,999808	0,026235	0,993883
Modelo 6	0,002560	0,999378	0,027721	0,993528
Modelo 7	0,002446	0,999408	0,027757	0,993488
Modelo 8	0,002299	0,999451	0,027688	0,993469
Modelo 9	0,002174	0,999491	0,027741	0,993491
Modelo 10	0,002137	0,999513	0,027433	0,993542
Média	0,001544	0,999540	0,026362	0,993952
Desvio Padrão	0,000715	0,000191	0,001239	0,000255

Modelos baseados na arquitetura ResNet34

Os resultados principais do treinamento dos modelos baseados na arquitetura ResNet34 podem ser consultados na Tabela 3. O modelo 3, indicado por uma linha destacada, foi selecionado como o de melhor desempenho, com valores muito próximos aos demais (diferença na terceira ou quarta casa decimal).

Tabela 3: Desempenho dos modelos baseados na ResNet34 na fase de treino.

Modelo	**Perda Treino**	**Acurácia Treino**	**Perda Validação**	**Acurácia Validação**
Modelo 1	0,014789	0,996	0,03486	0,9915
Modelo 2	0,009511	0,997385	0,032725	0,991458
Modelo 3	0,009445	0,997389	0,033507	0,991528
Modelo 4	0,009348	0,997417	0,034911	0,991417
Modelo 5	0,009213	0,997429	0,037769	0,9908
Modelo 6	0,009885	0,997208	0,039717	0,990319
Modelo 7	0,010326	0,997054	0,039408	0,990167
Modelo 8	0,009982	0,997169	0,038568	0,990427
Modelo 9	0,010148	0,997102	0,03812	0,990556
Modelo 10	0,010193	0,997075	0,037871	0,990542
Média	0,010404	0,997203	0,035844	0,991122
Desvio Padrão	0,001738	0,000388	0,002542	0,000518

Modelos baseados na arquitetura GoogLeNet

Para a arquitetura GoogLeNet, os resultados de treinamento dos modelos são apresentados na Tabela 4. O modelo 1, com o melhor desempenho está destacado na tabela.

Modelos baseados na arquitetura MobileNet

Os resultados do processo de treinamento com a arquitetura MobileNet estão compilados na Tabela 5. O modelo 1, que obteve maior acurácia no conjunto de validação, foi o modelo selecionado.

Tabela 4: Desempenho dos modelos baseados na GoogLeNet na fase treino.

Modelo	**Perda Treino**	**Acurácia Treino**	**Perda Validação**	**Acurácia Validação**
Modelo 1	0,000149	1	0,055514	0,990583
Modelo 2	0,000153	1	0,053456	0,990375
Modelo 3	0,000144	1	0,053894	0,990306
Modelo 4	0,000143	1	0,053972	0,989958
Modelo 5	0,000152	0,999996	0,053572	0,989967
Modelo 6	0,000171	0,999997	0,05471	0,989736
Modelo 7	0,000171	0,999997	0,055237	0,989679
Modelo 8	0,00017	0,999997	0,055564	0,989667
Modelo 9	0,000164	0,999998	0,056047	0,989694
Modelo 10	0,000162	0,999998	0,055685	0,989717
Média	0,000158	0,999998	0,054965	0,989898
Desvio Padrão	0,000011	0,000001	0,000904	0,000315

Tabela 5: Desempenho dos modelos baseados na MobileNet na fase de treino.

Modelo	**Perda Treino**	**Acurácia Treino**	**Perda Validação**	**Acurácia Validação**
Modelo 1	0,10087	0,96896	0,11649	0,96883
Modelo 2	0,10074	0,96917	0,11722	0,96863
Modelo 3	0,10668	0,96742	0,12453	0,96569
Modelo 4	0,10838	0,96703	0,12608	0,96508
Modelo 5	0,10629	0,96769	0,12479	0,96565
Modelo 6	0,10629	0,96768	0,12495	0,96550
Modelo 7	0,10751	0,96744	0,12586	0,96514
Modelo 8	0,11136	0,96634	0,12948	0,96420
Modelo 9	0,11106	0,96645	0,12933	0,96415
Modelo 10	0,10994	0,96675	0,12830	0,96447
Média	0,10668	0,96790	0,12430	0,96577
Desvio Padrão	0,00386	0,00098	0,00425	0,00157

5.2.2 Eficiência e Custo Computacional

A Tabela 6 apresenta a emissão de carbono e o consumo de energia medidos para cada modelo durante o treinamento das redes neurais selecionadas neste trabalho. Para cada modelo, foram realizadas dez execuções de treinamento, e os valores de consumo de energia apresentados correspondem à média aritmética dessas execuções. A estimativa da emissão de carbono foi calculada uma única vez, antes do início do treinamento, considerando o número máximo de épocas definido para cada modelo.

O consumo de energia foi medido durante cada execução, e os valores foram convertidos

de miliwatts para watts para facilitar a visualização. Já a emissão de carbono foi estimada em gramas de carbono equivalente(g CO_2 eq).

O modelo baseado na rede LeNet-5, destacado na tabela, apresentou os menores valores de emissão de carbono e consumo de energia.

Tabela 6: Resultados de eficiência energética das redes adotadas.

Modelos	**Emissão de carbono (g CO_2 eq)**	**Consumo de Energia (watts)**
Lenet-5	0,93286	49,60409
AlexNet	6,34910	95,36337
ResNet34	6,46300	87,37495
GoogLeNet	4,37320	75,87055
MobileNet	1,46991	66,01626
Média	3,91761	74,84584
Desvio Padrão	2,58074	17,11975

Tabela 7: Resultados do custo computacional das redes adotadas.

Modelos	**Número de Parâmetros**	**FLOPs**	**Tempo de Treino (s)**	**Tempo de Inferência (s)**
Lenet-5	609.354	11.401.984	132,05211	1,12889
AlexNet	23.271.114	147.910.912	546,07588	1,69993
ResNet34	21.292.042	69.921.536	667,21660	2,13568
GoogLeNet	5.977.530	36.235.520	626,06714	1,85027
MobileNet	24.618	8.054.656	424,10751	1,25443
Média	10.235.131	54.704.922	479,10385	1,61384
Desvio Padrão	10.318.545	56.078.348	213,04845	0,40415

A Tabela 7 apresenta uma comparação do custo computacional entre as diferentes arquiteturas de redes neurais. São apresentados o número total de parâmetros, o número de FLOPs e os tempos médios de treinamento e inferência para cada modelo, permitindo a análise da relação entre a arquitetura e seu custo computacional. Para facilitar a identificação dos modelos com menor custo computacional, as células que contêm os menores valores para cada métrica foram destacadas com um fundo sombreado.

5.2.3 Desempenho na Inferência

Esta seção apresenta os resultados de desempenho na fase de inferência, obtidos a partir da aplicação do modelo selecionado (melhor modelo com base na acurácia no conjunto de

validação), para cada rede, ao conjunto de teste do MNIST. Este conjunto de dados contém 10.000 imagens em escala de cinza de dígitos de 0 a 9. A Tabela 8 mostra o desempenho destes modelos neste conjunto de dados, avaliando sua capacidade de generalização através das métricas de acurácia, precisão, revocação e medida-F1.

Tabela 8: Desempenho dos modelos na fase de inferência.

Melhor Modelo	**Acurácia**	**Precisão**	**Revocação**	**Medida-F1**
Lenet-5	0,98680	0,98691	0,98680	0,98682
AlexNet	0,99540	0,99535	0,99535	0,99535
ResNet34	0,99140	0,99130	0,99126	0,99126
GoogLeNet	0,99240	0,99237	0,99232	0,99234
MobileNet	0,96900	0,96886	0,96859	0,96869
Média	0,98780	0,98776	0,98766	0,98769
Desvio Padrão	0,00966	0,00969	0,00978	0,00973

O modelo que obteve o melhor desempenho durante a inferência foi aquele baseado na arquitetura AlexNet, conforme destacado na Tabela 8 pela linha sombreada em cinza.

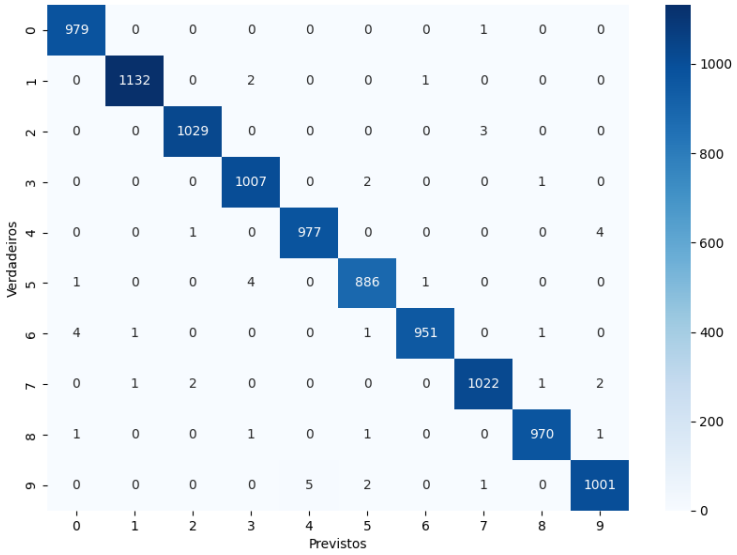

Figura 24: Matriz de confusão do modelo AlexNet

Os cinco modelos adotados apresentaram um bom desempenho na tarefa de classificação proposta. A Figura 24 apresenta a matriz de confusão do modelo baseado na rede AlexNet, que obteve o melhor desempenho dentre os cinco. A Figura 25 exibe a matriz de confusão do modelo baseado na rede MobileNet, que teve o pior desempenho. Ambas as matrizes

mostram uma alta taxa de acerto e uma baixa taxa de erros de classificação, confirmando a eficácia dos modelos na discriminação precisa das classes.

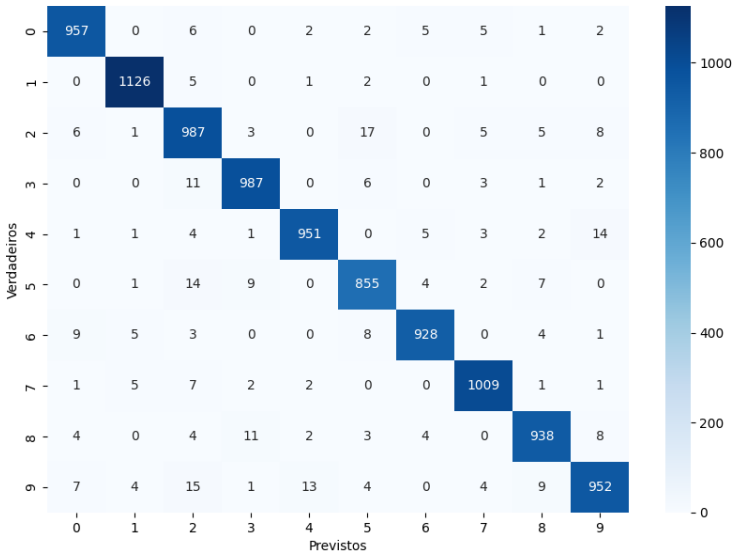

Figura 25: Matriz de confusão do modelo MobileNet

5.3 Testes Estatísticos

5.3.1 Teste de Pearson

A fim de se demonstrar que há, de fato, uma relação linear entre o consumo de energia e a emissão de carbono aferidos no treinamento dos modelos adotados foi realizado um teste estatístico utilizando o coeficiente de correlação de Pearson. Este teste estatístico mede a intensidade e a direção da relação linear entre duas variáveis, variando de -1 (correlação negativa perfeita) a +1 (correlação positiva perfeita), com 0 indicando ausência de correlação linear. A hipótese nula (H_0) foi de que não há correlação linear entre o consumo de energia e a emissão de carbono. Já a hipótese alternativa (H_1) foi de que há correlação linear entre o consumo de energia e a emissão de carbono.

A implementação computacional do teste foi realizada utilizando a biblioteca `scipy.stats` da linguagem Python. Especificamente, a função `pearsonr` foi empregada, recebendo como entrada os pares de dados correspondentes às variáveis em análise, isto é, consumo de energia e emissão de carbono, para cada modelo avaliado. A função `pearsonr`

retorna tanto o coeficiente de correlação de Pearson (r-valor) quanto o respectivo p-valor.

Os resultados da análise de correlação de Pearson indicaram uma forte correlação positiva entre o consumo de energia e a emissão de carbono ($r = 0,95$, $p = 0,01$). O p-valor de $0,01$ é menor que o nível de significância adotado ($\alpha = 0,05$), o que permite rejeitar a hipótese nula e concluir que há uma correlação linear estatisticamente significativa entre o consumo de energia e a emissão de carbono. A forte correlação positiva observada indica que modelos com maior consumo de energia tendem a apresentar maiores emissões de carbono.

5.3.2 Teste t de Welch

Para investigar as diferenças entre modelos com diferentes demandas computacionais, foi feita uma separação em dois grupos:

- Grupo 1: modelos baseados em redes com baixo custo computacional (LeNet-5 e MobileNet);
- Grupo 2: modelos baseados em redes com alto custo computacional (AlexNet, ResNet34 e GoogLeNet).

O objetivo foi verificar se há diferenças significativas em relação à emissão de carbono e à medida-F1 entre esses dois grupos. Devido ao pequeno número de modelos disponíveis em cada grupo (2 com baixo custo computacional e 3 com alto custo computacional), optou-se pelo teste t de Welch, também conhecido como teste t para amostras independentes com variâncias desiguais. O teste t de Welch é uma adaptação do teste t de Student que não assume a igualdade das variâncias das populações.

A hipótese nula (H_0) foi de que não há diferença significativa entre as médias dos grupos (baixo e alto custo computacional) para a métrica em análise (emissão de carbono ou medida-F1). Já a hipótese alternativa (H_1) foi de que há uma diferença significativa entre as médias dos grupos para a métrica em análise.

A implementação computacional do teste foi realizada utilizando a biblioteca `scipy.stats` da linguagem Python. Especificamente, a função `ttest_ind` foi empregada, recebendo como entrada os dados correspondentes à variável em análise (emissão de carbono ou medida-F1) para cada grupo de modelos (baixo e alto custo computacional). A função `ttest_ind` retorna tanto a estatística t (ou t-valor) quanto o respectivo p-valor.

Os resultados do Teste t de Welch revelaram uma diferença estatisticamente significativa entre os grupos no que diz respeito à emissão de carbono ($t = -6,21$, $p = 0,01$). O p-valor

menor que $0,05$ indica que a diferença observada é estatisticamente significativa, sugerindo que os modelos com baixo custo computacional apresentam, em média, emissões de carbono significativamente menores do que os modelos com alto custo computacional.

Em relação à medida-F1, o teste não encontrou uma diferença estatisticamente significativa entre os grupos ($t = -1,67$, $p = 0,34$), indicando que não há evidências suficientes para concluir que o custo computacional influencia o desempenho dos modelos nessa métrica.

6 Análise e Discussão dos Resultados

Neste capítulo, são apresentadas a análise e a discussão dos resultados obtidos ao longo desta pesquisa. Os resultados coletados, que se referem ao desempenho, impacto ambiental e custo computacional dos modelos de AP avaliados, foram examinados em conformidade com os objetivos propostos neste trabalho.

6.1 Treinamento dos Modelos

Os resultados de treinamento dos modelos demonstram um bom desempenho geral em todas as arquiteturas avaliadas, com alta acurácia e baixa perda nos conjuntos de treinamento e validação. O conjunto de treinamento, composto por dados já conhecidos pela rede, foi utilizado para ajustar os pesos e parâmetros dos modelos, permitindo que aprendessem a identificar padrões e características relevantes para a tarefa em questão. Já o conjunto de validação, composto por dados não vistos durante o treinamento, foi utilizado para avaliar a capacidade de generalização dos modelos.

Ao selecionar o modelo com a maior acurácia no conjunto de validação, busca-se garantir que o modelo escolhido seja capaz de identificar padrões e generalizar para dados não vistos, garantindo bom desempenho na etapa de inferência no conjunto de teste.

6.2 Eficiência e Custo Computacional dos Modelos Avaliados

Com base nos dados apresentados na Tabela 6, uma análise da eficiência energética dos modelos avaliados revela uma relação significativa entre seus custos computacionais e o impacto ambiental.

O desvio padrão para emissão de carbono (2,58074 g CO_2 eq) é relativamente alto em comparação com a média (3,91761 g CO_2 eq). Isso indica que há uma grande variação na emissão de carbono entre os modelos, com alguns modelos (como AlexNet e ResNet34) apresentando emissões muito acima da média, enquanto outros (como Lenet-5 e MobileNet) apresentam emissões bem abaixo.

O desvio padrão para consumo de energia (17,11975 w) também é considerável em relação à média (74,84584 w), indicando uma variabilidade significativa no consumo de energia entre

os modelos.

O modelo baseado na rede MobileNet se destaca pela leveza em termos de número de parâmetros (24.618) e FLOPs (8.054.656), resultando em um consumo de energia relativamente baixo (66,01626 w) e um tempo de treinamento e de inferência também relativamente baixos (424,10751 s e 1,25443 s, respectivamente). Esses resultados indicam que a MobileNet exige menos recursos computacionais para ser treinada e executada em relação aos modelos mais complexos, o que a torna uma boa opção para aplicações com restrições de recursos, como dispositivos móveis ou sistemas embarcados. Sua emissão de carbono (valor médio de $1,46991$ g) é menor que a maioria dos valores obtidos pelos demais modelos testados, mas não é a menor entre esses.

O modelo baseado na rede LeNet-5 demonstra uma eficiência excelente, especialmente em relação ao número de parâmetros (609.354) e FLOPs (11.401.984). Apresenta a menor emissão de carbono entre todos os modelos (0,93286 g) e um baixo consumo de energia (49,60409 w), além da menor média no tempo de treinamento e inferência entre todos os modelos, 132,05211 s e 1,12889, respectivamente. Esses dados confirmam a característica de leveza da LeNet-5, embora sua capacidade de aprendizado seja ligeiramente menor em relação aos modelos mais complexos, conforme visto na análise das métricas de desempenho.

O modelo baseado na rede GoogLeNet se posiciona em uma posição intermediária em termos de eficiência. Apresenta um número de parâmetros (5.977.530) e FLOPs (36.235.520) consideravelmente menores que as redes AlexNet e ResNet34, resultando em uma emissão de carbono (4,37320 g) e consumo de energia (75,87055 w) também intermediários. No entanto, seu tempo de treinamento e de inferência (626,06714 s e 1,85027 s, respectivamente) ainda são relativamente altos, embora menores que o da ResNet34, demonstrando que a complexidade de sua arquitetura Inception (Seção 4.4) impõe um custo computacional em termos de tempo de treinamento.

O desvio padrão para número de parâmetros (10.318.545) é alto em comparação com a média (10.235.131), refletindo a grande diferença no número de parâmetros entre modelos como MobileNet (24.618 parâmetros) e AlexNet (23.271.114 parâmetros).

O desvio padrão para FLOPs (56.078.348) também é alto em relação à média (54.704.922), revelando a diferença na carga computacional necessária para processar os dados em cada modelo.

Os desvios padrões para tempo de treino (213,04845 s) e tempo de inferência (0,40415 s) são mais baixos em relação às suas respectivas médias, indicando uma menor variabilidade

nesses aspectos.

Em contraste, os modelos baseados nas redes AlexNet e ResNet34 apresentam os maiores custos computacionais e impactos ambientais. A AlexNet possui o maior número de FLOPs (147.910.912) e um alto número de parâmetros (23.271.114), resultando em alta emissão de carbono (6,34910 g), alto consumo de energia (95,36337 w) e um tempo de treinamento e de inferência consideráveis (546,07588 s e 1,69993 s, respectivamente). A ResNet34, embora tenha um número de parâmetros ligeiramente menor que a AlexNet (21.292.042), apresenta um número de FLOPs (69.921.536) significativamente menor. Apesar disso, a ResNet34 apresenta emissão de carbono (6,46300 g) e consumo de energia (87,37495 w) comparáveis aos da AlexNet, além de apresentar os maiores tempos de treinamento e de inferência entre todos os modelos (667,21660 e 2, 13568 s, respectivamente). Esses resultados evidenciam que esses modelos, apesar de apresentarem excelente desempenho nas métricas de avaliação, exigem uma quantidade maior de recursos computacionais e possuem menor eficiência energética.

6.3 Desempenho dos Modelos Avaliados

Como visto na Tabela 8, o modelo baseado na rede AlexNet se destaca como o de melhor desempenho, atingindo valores excelentes em todas as métricas: acurácia de 0,9954, precisão de 0,99535, revocação de 0,99535 e medida-F1 de 0,99535. Esses resultados indicam que a AlexNet classifica as instâncias com muita eficácia e consegue identificar quase todos os exemplos corretamente.

Logo em seguida, os modelos baseados nas redes GoogLeNet e ResNet34 apresentam desempenhos muito próximos, com uma diferença marginal para a AlexNet, e também notáveis. A GoogLeNet alcança uma acurácia de 0,9924, precisão de 0,99237, revocação de 0,99232 e medida-F1 de 0,99234. A ResNet34, por sua vez, apresenta acurácia de 0,9914, precisão de 0,9913, revocação de 0,99126 e medida-F1 de 0,99126. A proximidade entre os valores desses dois modelos sugere que ambos são altamente eficazes na tarefa, com uma ligeira vantagem (na terceira casa decimal) para a GoogLeNet em todas as métricas avaliadas. Esses resultados estão de acordo com aqueles disponíveis na literatura, em que modelos mais complexos costumam oferecer melhor desempenho em tarefas de visão computacional.

O modelo baseado na rede LeNet-5, embora apresente um desempenho ligeiramente inferior em comparação com os modelos das redes AlexNet, GoogLeNet e ResNet34, ainda

demonstra resultados muito bons. Sua acurácia de 0,9868, precisão de 0,98691, revocação de 0,9868 e medida-F1 de 0,98682 indicam que ele também é capaz de classificar os exemplos com alta confiabilidade. No entanto, a pequena diferença em relação aos modelos de melhor desempenho sugere que ele pode ter uma capacidade ligeiramente menor de generalização e ser mais suscetível a erros de classificação.

Por fim, o modelo baseado na rede MobileNet apresenta o desempenho mais baixo entre os modelos avaliados, com acurácia de 0,969, precisão de 0,96886, revocação de 0,96859 e medida-F1 de 0,96869. Embora esses valores ainda sejam considerados bons, a diferença em relação aos demais modelos é mais perceptível. Isso indica que a MobileNet pode ser mais um pouco mais propensa a erros de classificação do que as outras redes, possivelmente devido à sua arquitetura mais leve, projetada para ambientes com recursos limitados, como dispositivos móveis.

Os dados coletados na inferência mostram também que a média dos valores de acurácia, precisão, revocação e medida-F1 para todos os modelos é de 0,98780, 0,98776, 0,98766 e 0,98769, respectivamente. Isso significa que, em média, todos os modelos testados apresentam um desempenho elevado em termos de acurácia, precisão, revocação e medida-F1.

Os desvios padrão de acurácia (0,00966), precisão (0,00969), revocação (0,00978) e medida-F1 (0,00973) são pequenos em comparação com as médias, sugerindo que os modelos oferecem desempenho consistente e confiável, com margem de erro mínima.

É importante ressaltar que esses resultados de desempenho são específicos para o conjunto de dados MNIST, que é relativamente simples para os modelos envolvidos. Em tarefas de classificação mais complexas, com conjuntos de dados maiores e mais desafiadores, as diferenças de desempenho entre os modelos podem ser mais significativas.

6.4 Relação entre Desempenho e Eficiência dos Modelos

A relação entre o desempenho dos modelos e a eficiência energética também foi investigada. O desempenho foi avaliado utilizando a medida-F1, uma métrica que oferece um balanço entre a precisão e a revocação. Embora o conjunto de dados MNIST seja conhecido por ser balanceado, a medida-F1 oferece uma visão mais completa do desempenho, considerando tanto a capacidade do modelo de evitar falsos positivos quanto falsos negativos.

O gráfico da Figura 26 apresenta a emissão de carbono (barras azuis claras) e a medida-F1 (linha vermelha com marcadores) para as cinco redes escolhidas: LeNet-5, AlexNet,

ResNet34, GoogLeNet e MobileNet. Observa-se que as redes AlexNet, ResNet34 e GoogLeNet, que estão associadas às maiores emissões de carbono, também apresentam um alto desempenho,conforme medido pela medida-F1. Por sua vez, a rede LeNet-5 está associada à menor emissão de carbono, mas com um desempenho ligeiramente inferior às anteriores. A MobileNet apresenta o pior desempenho entre os modelos avaliados e uma emissão de carbono baixa, mas não a menor. A análise visual sugere a ausência de uma clara relação linear entre as duas métricas.

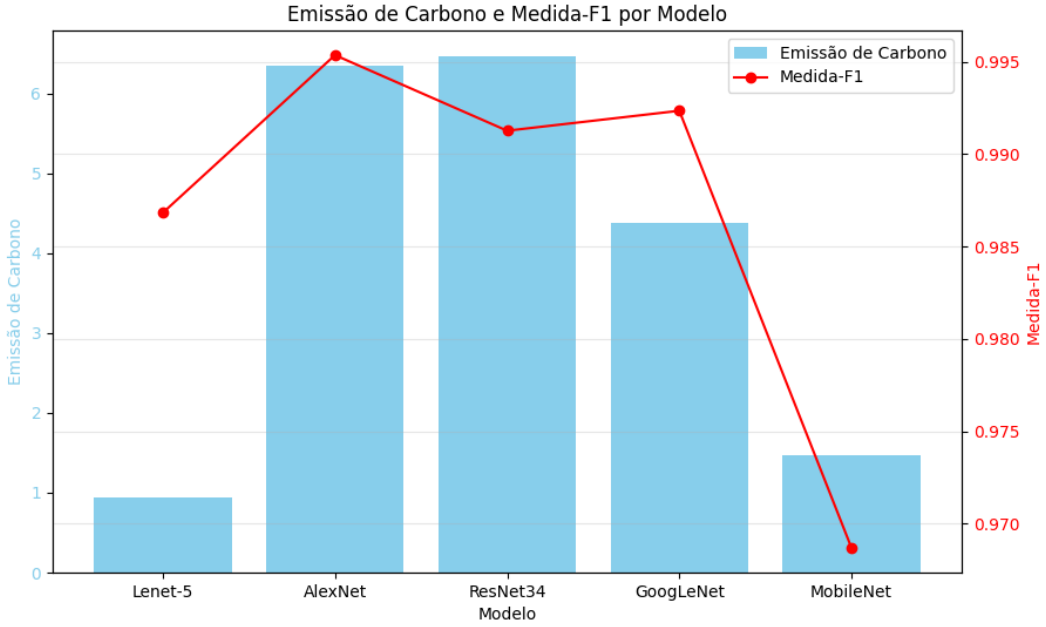

Figura 26: Emissão de carbono (g) e medida-F1 por modelo.

Para investigar a relação entre o consumo de energia e a emissão de carbono, foi realizado o Teste de Pearson (Seção 5.3.1). Os resultados demonstraram uma forte correlação positiva entre essas duas variáveis. Este resultado era esperado, uma vez que a emissão de carbono está diretamente relacionada ao consumo de energia durante o treinamento dos modelos. Como as duas métricas são fortemente correlacionadas, a emissão de carbono foi usada como medida para representar a eficiência dos modelos e simplificar a relação de custo-benefício nas análise subsequentes.

A Figura 27 apresenta a distribuição da emissão de carbono e da medida-F1 para os grupos de modelos com menor (grupo 1, em azul) e maior custo computacional (grupo 2, em vermelho). Observa-se uma clara distinção entre os grupos em relação à emissão de carbono, com o grupo de maior custo computacional apresentando medianas significativamente mais

altas. A mediana da emissão de carbono para o grupo de menor custo computacional está em torno de 1, enquanto a mediana para o grupo de maior custo computacional se situa acima de 6. A variabilidade dos dados, representada pelo tamanho das caixas, é consideravelmente maior para o grupo de maior custo computacional (grupo 2, em vermelho), indicando uma maior dispersão dos valores dentro desse grupo. O grupo de menor custo computacional (grupo 1, em azul) apresenta uma variabilidade muito menor, com os valores concentrados em uma faixa estreita.

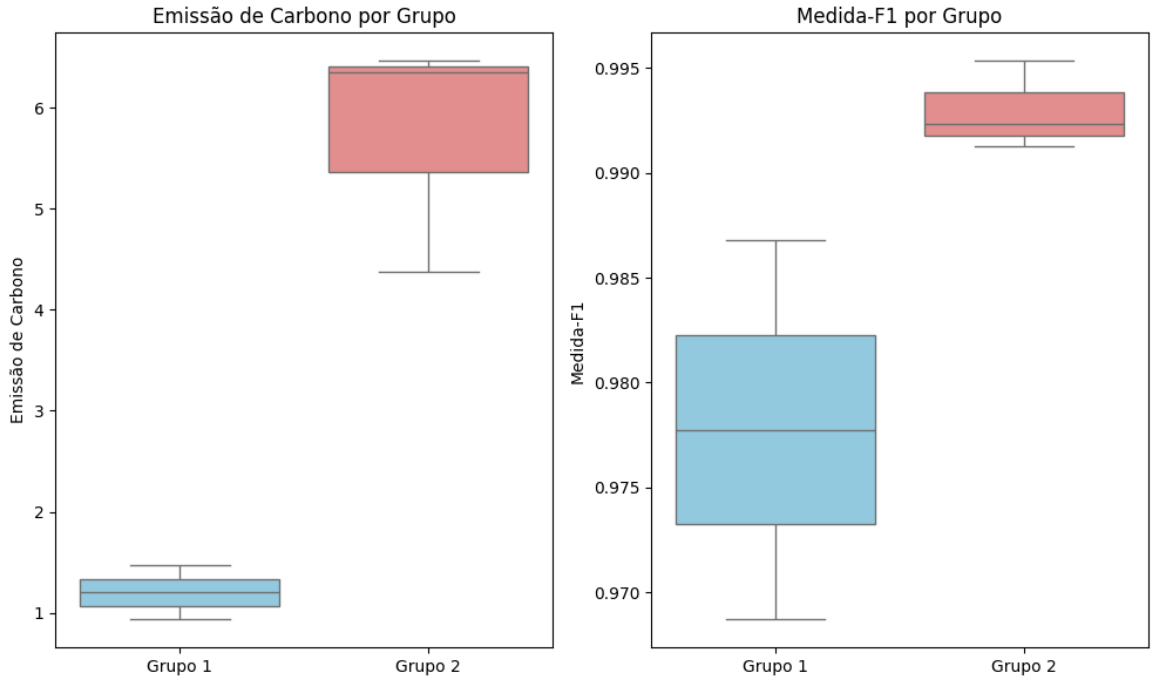

Figura 27: Distribuição da emissão de carbono (à esquerda) e da medida-F1 (à direita), por grupo de custo computacional: grupo 1, em azul, e grupo 2, em vermelho.

Em relação à medida-F1, a diferença entre as medianas dos grupos é menos pronunciada em comparação com a métrica de emissão de carbono. O grupo de maior custo computacional (grupo 2, em vermelho) apresenta uma mediana ligeiramente superior, próxima de 0,995, enquanto o grupo de menor custo computacional (grupo 1, em azul) tem uma mediana em torno de 0,980. A variabilidade dos dados também é diferente entre os grupos. O grupo de menor custo computacional (grupo 1, em azul) apresenta uma variabilidade maior, com uma caixa mais alta e limites mais distantes, indicando uma maior dispersão dos valores de medida-F1. O grupo de maior custo computacional (grupo 2, em vermelho) apresenta uma variabilidade menor, com os valores mais concentrados. No entanto, é importante notar que todos os valores de medida-F1 variam dentro de um intervalo muito pequeno, entre

aproximadamente 0,97 e 0,995, indicando que o desempenho dos modelos, em termos de Medida-F1, é relativamente consistente, independentemente do grupo.

Para verificar se as diferenças observadas entre os grupos são estatisticamente significativas, aplicou-se o Teste t de Welch (Seção 5.3.2), adequado para amostras independentes com variâncias possivelmente desiguais e especialmente relevante considerando o pequeno tamanho amostral de cada grupo. Os resultados do teste t de Welch indicaram uma diferença estatisticamente significativa na emissão de carbono ($t = -6{,}205$, $p = 0{,}013$), corroborando a observação visual de que os modelos com menor custo computacional apresentam emissões significativamente menores. Em relação à medida-F1, o teste t de Welch não encontrou diferenças estatisticamente significativas entre os grupos ($t = -1{,}665$, $p = 0{,}338$), apesar da diferença nas medianas visualizadas nos *box plots*.

Em conjunto, a análise visual dos gráficos e o teste t de Welch sugerem uma possível relação entre desempenho e eficiência. A forte correlação entre consumo de energia e emissão de carbono, demonstrada pelo teste de Pearson (Seção 5.3.1), justifica o uso da emissão de carbono como um indicador da eficiência energética. Os modelos com maior custo computacional tendem a apresentar um desempenho consideravelmente melhor, conforme observado nos *box plots* da medida-F1, mas incorrem em um custo ambiental significativamente maior, evidenciado tanto pela análise visual dos *box plots* da emissão de carbono quanto pelo resultado estatisticamente significativo do teste t de Welch para essa métrica.

Entre as redes de menor custo computacional, o modelo baseado na LeNet-5 demonstrou um desempenho apenas ligeiramente inferior aos modelos mais complexos, atingindo uma medida-F1 de 0,98682, enquanto apresentou a menor emissão de carbono entre todos os modelos avaliados. Essa combinação de bom desempenho com baixo custo computacional torna a LeNet-5 uma opção particularmente interessante quando se busca um bom equilíbrio entre desempenho e eficiência. Dessa forma, considerando o contexto deste estudo e os resultados obtidos com o conjunto de dados MNIST, a LeNet-5 se destaca como uma rede com melhor custo-benefício. Ela consegue atingir um valor apenas ligeiramente inferior para as métricas de desempenho quando comparado aos modelos mais complexos, enquanto apresenta um impacto ambiental drasticamente menor, indo ao encontro do que é preconizado pela IA Verde, que busca o desenvolvimento e a aplicação de IA de forma responsável, minimizando seu impacto ambiental.

7 Conclusão

7.1 Considerações Finais

O principal objetivo deste trabalho foi comparar e avaliar a eficiência de modelos de AP, demonstrando a viabilidade de se obterem modelos mais eficientes e com menor impacto ambiental, em consonância com os princípios da IA Verde. Para isso, diversas arquiteturas de CNNs foram analisadas, considerando métricas de desempenho, eficiência energética e custo computacional.

A análise revelou uma importante relação entre desempenho e eficiência. A emissão de carbono, utilizada como indicador do custo computacional devido à sua forte correlação com o consumo de energia, apresentou uma clara distinção entre os grupos de modelos com menor e maior custo computacional, confirmada pelo teste *t* de Welch. Em contraste, as diferenças de desempenho, conforme caputado pela medida-F1, entre tais grupos foram menos pronunciadas, com o teste *t* de Welch não encontrando diferenças estatisticamente significativas.

O modelo baseado na rede LeNet-5, pertencente ao grupo de menor custo computacional, destacou-se por apresentar um desempenho apenas ligeiramente inferior aos modelos mais complexos e, ao mesmo tempo, a menor emissão de carbono entre todos os modelos avaliados. Este achado demonstrou a viabilidade de se alcançar um bom equilíbrio entre desempenho e eficiência, sugerindo que, para certas aplicações, tal como o reconhecimento de dígitos manuscritos, a busca por um desempenho máximo pode não justificar o aumento substancial no custo ambiental. A rede LeNet-5, portanto, exemplificou a possibilidade de se desenvolverem soluções mais eficientes e sustentáveis, alinhadas aos princípios da IA Verde.

Este estudo também reforça a necessidade de maior conscientização sobre o impacto ambiental da IA na comunidade acadêmica e também na indústria, assim como a importância do emprego mais frequente de métricas de eficência energética. Como sugerido por Wolff Anthony *et al.* [2], “Propomos que a pegada total de energia e carbono do desenvolvimento e treinamento do modelo seja relatada juntamente com a precisão e métricas semelhantes para promover a computação responsável em AP”. Alinhado com essa recomendação, e como demonstrado neste trabalho, o registro sistemático de métricas de eficiência energética e custo computacional, como consumo de energia, emissão de carbono, tempo de treinamento, tempo

de inferência e número de FLOPs, em estudos futuros não só contribui para a transparência e a reprodutibilidade da pesquisa, mas também ajuda a quantificar e reduzir a pegada de carbono da área. Ao tornar visíveis os custos ambientais envolvidos no treinamento e uso de modelos complexos, incentiva-se o desenvolvimento de soluções mais eficientes e sustentáveis, alinhadas com os princípios da IA Verde.

A crescente utilização de aplicações baseadas em AP, incluindo modelos de linguagem como os utilizados no ChatGPT e Gemini, tem gerado uma preocupação crescente com o impacto ambiental da área. O desenvolvimento, treinamento e utilização desses modelos exigem uma quantidade enorme de recursos computacionais, resultando em um consumo energético significativo e, consequentemente, em uma grande pegada de carbono. Nesse contexto, qualquer ação que contribua para a redução desse impacto é de extrema importância. Como proposto por Lenherr *et al.* [19], estratégias como a otimização de algoritmos, o uso de plataformas de borda (*Edge Computing*) para inferência, a utilização de aprendizado federado e o desenvolvimento de hardwares mais eficientes representam caminhos promissores para mitigar o impacto ambiental da AP. Dessa forma, a adoção de práticas de registro e divulgação de métricas de eficiência energética torna-se um passo essencial para uma pesquisa em IA mais responsável e consciente do seu impacto sobre o planeta.

Além disso, a IA Verde preconiza democratizar o acesso à IA, tornando os modelos mais baratos e menos dispendiosos. A redução dos custos computacionais e energéticos associados ao treinamento e à inferência de modelos ajuda a torná-los mais acessíveis a uma ampla gama de usuários, desde pequenas empresas até pesquisadores de instituições com recursos limitados [27]. Essa democratização permite que um número maior de indivíduos e organizações tenha acesso à tecnologia de ponta e possa se beneficiar da IA para inovar e encontrar soluções para desafios complexos.

Esta pesquisa teve como fruto o artigo “Aprendizagem Profunda e Inteligência Artificial Verde: Caminhos para um Futuro mais Sustentável” [8], apresentado no XV Workshop de Computação Aplicada à Gestão do Meio Ambiente e Recursos Naturais (WCAMA).

A busca por uma IA mais eficiente e sustentável deve ser vista como uma necessidade, e não apenas uma tendência. A trajetória da IA Verde depende da capacidade do ser humano de inovar de forma responsável, buscando soluções que otimizem o desempenho sem comprometer a saúde do planeta.

7.2 Trabalhos Futuros

Apesar da relevância dos achados deste trabalho, reconhecem-se algumas limitações que abrem caminhos para investigações futuras. A principal limitação reside no tamanho da amostra de modelos avaliados (apenas 5 redes) e no foco em uma única tarefa de classificação de imagens utilizando o conjunto de dados MNIST. Embora o MNIST seja um *benchmark* comum para avaliação inicial de modelos, ele representa um cenário simplificado que pode não refletir a complexidade de aplicações do mundo real.

Portanto, diversas direções para trabalhos futuros se apresentam:

- **Ampliação do conjunto de dados e tarefas:** Avaliar os modelos em conjuntos de dados mais complexos e representativos, como CIFAR-10, CIFAR-100, ImageNet ou outros conjuntos de dados específicos para aplicações práticas (como por exemplo, imagens médicas). Investigar o desempenho e a eficiência energética em diferentes tarefas de aprendizagem de máquina, como detecção de objetos ou processamento de linguagem natural, permitirá uma análise mais abrangente da relação entre desempenho, custo computacional e impacto ambiental.

- **Diversificação das arquiteturas de modelos:** Expandir a análise para incluir uma gama mais ampla de arquiteturas de redes neurais, incluindo modelos basedos na arquitetura Transfomer, como os *Large Language Models* [27]. Avaliar diferentes técnicas de otimização de modelos, tais como como poda de pesos, quantização e destilação, com o objetivo de investigar como essas técnicas afetam a eficiência energética e o desempenho.

- **Análise do impacto de diferentes hardwares:** Investigar o impacto do hardware utilizado no treinamento e inferência dos modelos. Comparar o consumo de energia e a emissão de carbono em diferentes plataformas de hardware, como CPUs, GPUs e TPUs, ou mesmo em dispositivos de borda (*edge devices*), ajudaria a entender como a escolha do hardware influencia a sustentabilidade da AP.

- **Aprofundamento em métricas de eficiência energética:** Investigar de forma mais profunda as métricas de eficiência energética é uma direção promissora para trabalhos futuros. Isso inclui realizar um levantamento mais abrangente das diferentes métricas

propostas na literatura ou que venham a ser desenvolvidas e aplicá-las a diferentes arquiteturas de redes neurais.

Por fim, espera-se que este trabalho inspire outros pesquisadores a considerar a sustentabilidade como um critério fundamental na pesquisa em AP, impulsionando o desenvolvimento de uma IA mais consciente e responsável com o meio ambiente. Acredita-se que a busca por modelos mais eficientes e sustentáveis é um passo fundamental para garantir o futuro da IA e a saúde do planeta.

Referências

[1] Charu C. Aggarwal. *Neural Networks and Deep Learning: A Textbook*. Springer Publishing Company, Incorporated, 2nd edition, 2023.

[2] Lasse F. Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. ICML Workshop on Challenges in Deploying and monitoring Machine Learning Systems, July 2020. arXiv:2007.03051.

[3] Vitor Hugo Batista. Desenvolvedores apostam em códigos verdes para diminuir impacto ambiental de aplicativos. `https://tinyurl.com/3xvmaufv`, 2024. Acessado em: 15-Jan-2025.

[4] Gabriel Breder, Douglas Brum, Lucas Dirk, and Mariza Ferro. O paradoxo da ia para sustentabilidade e a sustentabilidade da ia. In *Anais do V Workshop sobre as Implicações da Computação na Sociedade*, pages 105–116, Porto Alegre, RS, Brasil, 2024. SBC.

[5] Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Trends in ai inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38:100857, 2023.

[6] Payal Dhar. The carbon impact of artificial intelligence. *Nature Machine Intelligence*, 2(8):423–425, 2020.

[7] Constance Douwes, Philippe Esling, and Jean-Pierre Briot. Energy consumption of deep generative audio models. *arXiv preprint arXiv:2107.02621*, 2021.

[8] Vívian Ferraro, Gabriel Gullo, Daniel Costa, and Pedro Moura. Aprendizagem profunda e inteligência artificial verde: Caminhos para um futuro mais sustentável. In *Anais do XV Workshop de Computação Aplicada à Gestão do Meio Ambiente e Recursos Naturais*, pages 159–168, Porto Alegre, RS, Brasil, 2024. SBC.

[9] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[10] S.S. Haykin. *Neural Networks and Learning Machines*. Pearson International Edition. Pearson, 2009.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv.org*, 2015.

[12] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Adam Hartwig. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv.org*, 2017.

[13] Arnulf Jentzen, Benno Kuckuck, and Philippe von Wurstemberger. Mathematical introduction to deep learning: Methods, implementations, and theory, 10 2023.

[14] John D Kelleher. *Deep Learning*. The MIT Press Essential Knowledge series, 1st edition, 2019.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[16] J. Krohn and A. Beyleveld, G. ang Bassens. *Deep Learning Illustrated. A Visual, Interactive Guide to Artificial Intelligence*. Addison Wesley Data & Analytics Series, 2020.

[17] Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv.org*, 2019.

[18] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[19] Nicola Lenherr, René Pawlitzek, and Bruno Michel. New universal sustainability metrics to assess edge intelligence. *Sustainable Computing: Informatics and Systems*, 31:100580, 2021.

[20] Pengfei Li, Jianyi Yang, Mohammad A. Islam, and Shaolei Ren. Making ai less "thirsty": Uncovering and addressing the secret water footprint of ai models. *arXiv preprint arXiv:2304.03271*, 2023.

[21] Medium. Complete guide about cnns - part 1. `https://medium.com/analytics-vidhya/complete-guide-about-cnns-part-1-65bdfee3cae3`, 2023. Acessado em: 29 jan. 2025.

[22] T.M. Mitchell. *Machine Learning*. McGraw-Hill international editions - computer science series. McGraw-Hill Education, 1997.

[23] Andreas C. Müller and Sarah Guido. *Introduction to Machine Learning with Python*. O'Reilly Media, 1st edition, 2016.

[24] O Globo. Treino do chatgpt consumiu 700 mil litros de água, equivalente a encher uma torre de resfriamento de um reator nuclear. `https://tinyurl.com/treinoChatGPTConsumoAgua`, 2023. Acessado em: 15-Ago-2024.

[25] Sumit Saha. A comprehensive guide to convolutional neural networks—the eli5 way. `https://tinyurl.com/32yju4y9`, 2018. Acessado em: 20-Nov-2024.

[26] Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. Green ai. *Communications of the ACM*, 63(12):54–63, 2020.

[27] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

[28] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[29] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. Carbontracker: Tracking and predicting the carbon footprint of training deep learning models. *arXiv.org*, 2020.

[30] Época Negócios. Chatgpt usa 17 mil vezes mais energia que uma família média dos eua diariamente, aponta relatório. `https://tinyurl.com/chatGPTGastoEnergiaDiariamente`, 2024. Acessado em: 15-Ago-2024.