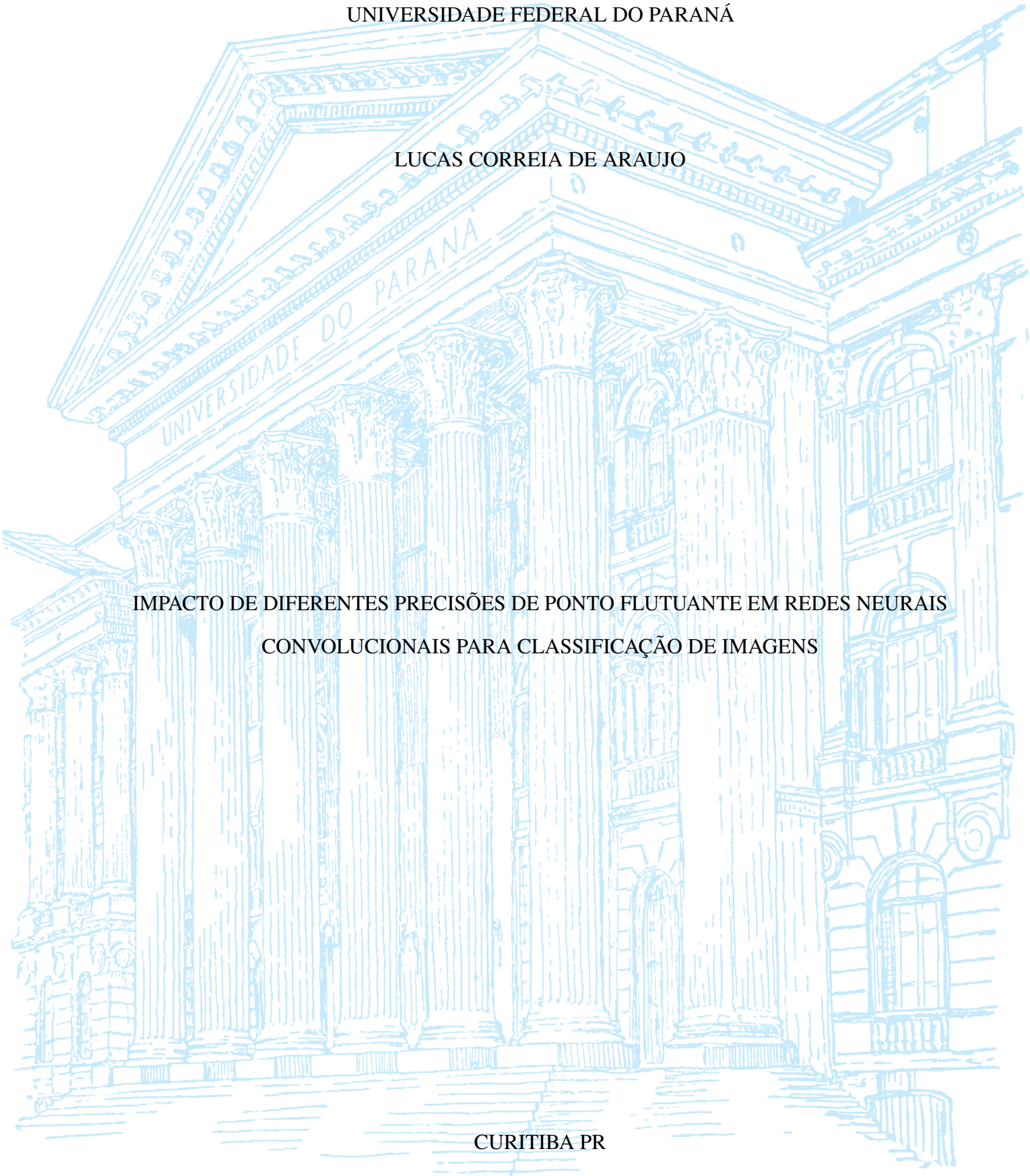
UNIVERSIDADE FEDERAL DO PARANÁ

LUCAS CORREIA DE ARAUJO



IMPACTO DE DIFERENTES PRECISÕES DE PONTO FLUTUANTE EM REDES NEURAIS CONVOLUCIONAIS PARA CLASSIFICAÇÃO DE IMAGENS

CURITIBA PR

2025

LUCAS CORREIA DE ARAUJO

IMPACTO DE DIFERENTES PRECISÕES DE PONTO FLUTUANTE EM REDES NEURAIS CONVOLUCIONAIS PARA CLASSIFICAÇÃO DE IMAGENS

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

Área de concentração: *Computação*.

Orientador: Paulo Ricardo Almeida Lisboa.

CURITIBA PR

2025

Universidade Federal do Paraná
Setor de Ciências Exatas
Curso de Ciência da Computação

Ata de Apresentação de Trabalho de Conclusão de Curso 2

Título do Trabalho: IMPACTO DE DIFERENTES PRECISÕES DE PONTO FLUTUANTE EM REDES NEURAIS CONVOLUCIONAIS PARA CLASSIFICAÇÃO DE IMAGENS

Autor(es):

GRR 20206150 Nome: LUCAS CORREIA DE ARAUJO

Apresentação: Data: 17/12/2025 Hora: 14h00 Local: Auditório DINF

Orientador: Paulo Ricardo Lisboa de Almeida

Membro 1: Marcelino Ulica Abel

Membro 2: Andre Vignatti

(nome) (assinatura)

AVALIAÇÃO – Produto escrito		ORIENTADOR	MEMBRO 1	MEMBRO 2	MÉDIA
Conteúdo	(00-40)				
Referência Bibliográfica	(00-10)				
Formato	(00-05)				
AVALIAÇÃO – Apresentação Oral					
Domínio do Assunto	(00-15)				
Desenvolvimento do Assunto	(00-05)				
Técnica de Apresentação	(00-03)				
Uso do Tempo	(00-02)				
AVALIAÇÃO – Desenvolvimento					
Nota do Orientador	(00-20)		**********	**********	
NOTA FINAL		**********	**********	**********	84

Os pesos indicados são sugestões.

Conforme decisão do colegiado do curso de Ciência da Computação, a entrega dos documentos comprobatório de trabalho de Conclusão de Curso 2 deve deve respeitar os seguintes procedimentos: o orientador deve abrir um processo no Sistema Eletrônico de Informações (SEI – UFPR); Selecionar o tipo: *Graduação: Trabalho Conclusão de Curso*; informar os interessados: nome do aluno e o nome do orientador; anexar esta ata escaneada e a versão final do PDF da monografia do aluno; Tramitar o processo para CCOMP (Coordenação de Ciência da Computação).

Dedico à minha esposa Vitorya Marques de Amorim de Araujo, mulher sábia e virtuosa, e à minha mãe Neusa Iracema Pereira dos Santos Araujo (in memoriam), pelo amor, educação e apoio incondicional em todos os momentos da minha vida.

AGRADECIMENTOS

Primeiramente, a Deus, pelo dom da vida e pelo privilégio de ter chegado até aqui. Pela Sua graça, tive força, discernimento e perseverança para superar desafios e seguir adiante mesmo nos momentos mais difíceis.

À minha esposa, Vitorya Amorim de Araujo, mulher da minha vida, por seu amor, paciência e apoio incondicional ao nesses últimos anos. Sua compreensão e incentivo foram fundamentais para que eu pudesse me dedicar aos estudos e à pesquisa. Como peça-chave nos momentos de bloqueio e incerteza, sua presença foi essencial para a conclusão deste trabalho.

À minha mãe, Neusa Iracema Pereira dos Santos Araujo *(in memoriam)*, que sempre me ensinou o valor da educação e do esforço para alcançar objetivos. Seu exemplo de dedicação e amor pela família permanece vivo em minha memória e continua a me guiar, inspirando-me a ser uma pessoa melhor a cada dia.

Ao meu pai, Josué Correia de Araujo, pela formação do meu caráter como homem e cidadão. Seus ensinamentos sobre ética, responsabilidade e perseverança foram fundamentais para minha trajetória acadêmica e pessoal. Seu exemplo de dedicação ao trabalho e à família possibilitou que eu tivesse a oportunidade de estudar e evoluir.

À minha madrasta, Juliana Ribeiro de Araujo, por acreditar em mim e me acolher com carinho de mãe. Sua presença em minha vida trouxe apoio e afeto indispensáveis durante os últimos anos da minha formação acadêmica e crescimento profissional.

Ao meu irmão, Rafael Correia de Araujo, pelos conselhos e pela torcida constante. Seu apoio fraterno foi um alicerce importante ao longo da minha jornada acadêmica. Estendo meus agradecimentos à minha cunhada Eliegie Cruzetta de Araujo e ao meu sobrinho Matheus Cruzetta de Araujo, pelo carinho, incentivo e bons momentos compartilhados em família.

Aos meus padrinhos, Jeferson dos Santos da Silva e Polyana Maryssa Leal dos Santos, pela amizade sincera e pelo suporte durante essa caminhada. Vocês são exemplos de integridade e bondade, e sua presença em minha vida é uma bênção.

Aos meus irmãos, filhos de minha madrasta, Lucas Ribeiro Gomes e Hanna Clarah, pelo companheirismo e amizade. Vocês são um presente importante em minha vida, e nossa convivência sempre foi fonte de alegria e apoio mútuo.

Ao meu orientador, Dr. Paulo Ricardo Lisboa de Almeida, pela paciência, disponibilidade e orientação ao longo de todo o processo de pesquisa e elaboração deste TCC. Seu profissionalismo e dedicação, aliados às aulas inspiradoras, motivaram-me a buscar conhecimento de forma mais profunda e a compreender melhor o papel do cientista da computação na sociedade.

À minha família da Igreja Assembleia de Deus Ministério Rocha Eterna, em especial ao pastor José de Oliveira e à missionária Lucimar de Oliveira, por todo o apoio espiritual, orações e palavras de encorajamento. A fé e a comunhão vivenciadas na igreja foram fundamentais para minha força interior e perseverança.

À Universidade Federal do Paraná (UFPR), como instituição de ensino, pela estrutura acadêmica oferecida e pelo ambiente propício à evolução intelectual, científica e profissional ao longo do curso.

Por fim, a todos os amigos e colegas que, direta ou indiretamente, contribuíram para minha trajetória acadêmica. Cada um teve um papel importante nessa caminhada, seja por meio de palavras de incentivo, momentos de descontração ou apoio prático.

RESUMO

Redes neurais convolucionais são amplamente empregadas em tarefas de classificação de imagens, porém o custo computacional associado ao uso de formatos de ponto flutuante de maior precisão pode ser elevado, especialmente em cenários com restrições de recursos. Este trabalho investiga o impacto do uso de diferentes precisões numéricas, a saber, precisão IEEE 754 simples e dupla (FP32 e FP64) e precisão reduzida de 16 bits no formato *bfloat16* (brain floating point). O objetivo é avaliar como essas escolhas afetam o tempo de treinamento, o comportamento de convergência e a acurácia de uma rede neural convolucional implementada em C++ com a biblioteca LibTorch, aplicada aos conjuntos de dados PKLot e Fruits360.

Inicialmente, são revisados conceitos fundamentais de redes neurais, redes neurais convolucionais e representação numérica em ponto flutuante, bem como trabalhos relacionados que exploram o uso de formatos de menor precisão em aprendizado profundo. Em seguida, descrevem-se a arquitetura da rede, o ambiente de desenvolvimento em C++ com LibTorch, os conjuntos de dados utilizados e o procedimento de coleta das métricas experimentais. Para cada formato numérico e configuração de tamanho de *batch*, foram registradas a acurácia de teste, o comportamento da função de perda, a época de convergência e o tempo total de treinamento.

Os resultados indicam que a migração de FP64 para FP32 reduz significativamente o tempo de treinamento sem comprometer a qualidade das classificações. Em ambos os conjuntos de dados, as acurácias finais obtidas com FP32 permaneceram muito próximas às de FP64, enquanto o tempo de treinamento em precisão simples foi substancialmente menor, com destaque para o conjunto Fruits360, no qual a diferença de tempo entre FP64 e FP32 foi significativamente superior ao esperado. O formato *bfloat16* apresentou desempenho comparável em termos de acurácia, porém não resultou em ganhos consistentes de tempo na implementação avaliada.

Conclui-se que, para os problemas de classificação considerados e no ambiente experimental adotado, o formato FP32 representa a melhor relação entre custo computacional e desempenho preditivo. Formatos de menor precisão, como *bfloat16*, mostram potencial, mas seus benefícios práticos dependem fortemente do suporte oferecido pelo hardware e pelas bibliotecas utilizadas.

Palavras-chave: Ponto flutuante. Redes neurais convolucionais. Precisão numérica. Custo computacional. PKLot. Fruits360.

ABSTRACT

Convolutional neural networks are widely used in image classification tasks; however, the computational cost associated with higher-precision floating-point formats can be significant, particularly in resource-constrained scenarios. This work investigates the impact of different numerical precisions, namely IEEE 754 single and double precision (FP32 and FP64) and the reduced 16-bit *bfloat16* (brain floating point) format. The objective is to evaluate how these choices affect training time, convergence behavior, and classification accuracy of a convolutional neural network implemented in C++ using the LibTorch library, applied to the PKLot and Fruits360 datasets.

First, fundamental concepts of neural networks, convolutional neural networks, and floating-point numerical representation are reviewed, as well as related work exploring the use of reduced-precision formats in deep learning. Subsequently, the adopted network architecture, the C++ development environment with LibTorch, the datasets, and the procedure for collecting experimental metrics are described. For each numerical format and *batch* size configuration, test accuracy, loss behavior, convergence epoch, and total training time were recorded.

The results indicate that migrating from FP64 to FP32 significantly reduces training time without compromising classification quality. For both datasets, the final accuracies obtained with FP32 remained very close to those of FP64, while training time in single precision was substantially lower. In particular, for the Fruits360 dataset, the time difference between FP64 and FP32 was considerably larger than initially expected. The *bfloat16* format achieved comparable accuracy but did not provide consistent training time improvements in the evaluated implementation.

It is concluded that, for the considered image classification problems and the adopted experimental environment, the FP32 format offers the best balance between computational cost and predictive performance. Lower-precision formats, such as *bfloat16*, remain promising, but their practical benefits strongly depend on the level of hardware and software support available.

Keywords: Floating point. Convolutional neural networks. Numeric precision. Computational cost. PKLot. Fruits360.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

UFPR	Universidade Federal do Paraná
PKLot	Parking Lot Image Database
CNN	Convolutional Neural Network
bfloat16	Brain Floating Point 16
FP16	Floating Point 16
FP32	Floating Point 32
FP64	Floating Point 64

SUMÁRIO

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

Redes neurais, em especial redes neurais convolucionais (do inglês, *Convolutional Neural Networks* ou CNNs), tornaram-se o núcleo de muitas aplicações de classificação de imagens, detecção de objetos e reconhecimento de padrões visuais (Goodfellow et al., 2016; LeCun et al., 2015; Bishop, 2006; Krizhevsky et al., 2012; Cireşan et al., 2012). Esses modelos exploram camadas sucessivas para extrair características cada vez mais abstratas das imagens de entrada e têm sido adotados em cenários que vão de grandes centros de dados a sistemas embarcados.

Um exemplo concreto é o uso de CNNs para classificar vagas de estacionamento como livres ou ocupadas em campi universitários. Trabalhos como o de Hochuli et al. (2022) utilizam imagens de estacionamentos da Universidade Federal do Paraná (UFPR) e da Pontifícia Universidade Católica do Paraná (PUCPR) para avaliar estratégias de anotação em sistemas de classificação de vagas, baseados no conjunto de dados PKLot (Almeida et al., 2015).

Essas redes exigem uma quantidade significativa de recursos computacionais, tanto para treinamento quanto para inferência, isto é, para realizar as predições que resolvem o problema proposto. Um dos principais fatores que influenciam esse custo é dependente do tipo de dados numéricos utilizados nas operações matemáticas que compõem o funcionamento das redes.

Em grande maioria, para realizar essas operações, utiliza-se a representação padrão de números reais em ponto flutuante, conforme definido pela norma IEEE 754 (IEEE, 2019). Essa norma especifica um protocolo para representar números reais em formatos binários, para ser utilizado em sistemas computacionais. Porém, dado a infinitude de números reais, a representação em ponto flutuante é necessariamente uma aproximação, o que pode introduzir erros numéricos nas operações realizadas.

Sendo assim, ao utilizar mais bits para representar os números, é possível reduzir esses erros, obtendo uma aproximação mais precisa dos valores reais. Melhor aproximação, por sua vez, pode levar a resultados mais precisos, porém com maior custo computacional. Por outro lado, o uso de menos bits reduz o custo computacional, mas pode aumentar os erros numéricos, podendo afetar a precisão dos resultados.

Portanto, o uso de diferentes formatos estabelecidos produz uma troca entre custo computacional e precisão numérica, que deve ser cuidadosamente avaliada, especialmente em aplicações de aprendizado profundo.

1.2 PROBLEMA DE PESQUISA

Diante desse contexto, esse trabalho busca responder à seguinte questão de pesquisa:

Como o uso de diferentes precisões de ponto flutuante afeta o desempenho, a acurácia e a convergência de redes neurais convolucionais aplicadas a problemas simples de classificação de imagens?

Para delimitar o escopo do estudo, a pesquisa se concentra nas precisões FP64 (precisão IEEE-754 dupla), FP32 (precisão IEEE-754 simples) e bfloat16 (precisão reduzida com 16 bits). A escolha dessas precisões se justifica pela sua relevância prática em aplicações de aprendizado profundo, (Micikevicius et al., 2018a), bem como pela disponibilidade de suporte em bibliotecas populares como a LibTorch (Paszke et al., 2019).

1.3 HIPÓTESES

As hipóteses consideradas neste trabalho são as seguintes:

- O uso de formatos numéricos de menor precisão no treinamento e na inferência de redes neurais convolucionais tende a reduzir o tempo de execução, podendo, entretanto, introduzir perdas na acurácia e alterações na taxa de convergência.
- A acurácia dos modelos treinados com formatos de menor precisão será inferior àquela obtida com formatos de maior precisão, como FP64, devido ao aumento dos erros numéricos.

1.4 OBJETIVOS

Objetivo geral: avaliar o impacto do uso de diferentes precisões de ponto flutuante (FP64, FP32 e bfloat16) no desempenho e na acurácia de redes neurais convolucionais aplicadas a problemas de classificação de imagens.

Objetivos específicos:

- Implementar uma rede neural convolucional simples, em C++ utilizando a biblioteca LibTorch, adequada a problemas simples de classificação de imagens.
- Adaptar a implementação para permitir o uso de diferentes tipos de ponto flutuante, incluindo FP64, FP32 e *bfloat16*.
- Treinar e inferir a rede para cada tipo de precisão, aplicando-a a conjuntos de dados de classificação de imagens.
- Coletar tempo de treinamento, o valor médio da perda de validação e de teste, acurácia em teste e número de épocas até convergência para cada configuração.
- Comparar e analisar as métricas obtidas com cada formato numérico, destacando as diferenças observadas nos termos de desempenho estabelecidos.

1.5 JUSTIFICATIVA

O uso de diferentes precisões de ponto flutuante em redes neurais convolucionais já é abordado na literatura, especialmente em arquiteturas conhecidas e cenários de alto desempenho. Ainda assim, o efeito prático da precisão numérica sobre tempo de treinamento, convergência e acurácia permanece um tema relevante para análise experimental.

Neste trabalho, a motivação é de caráter exploratório, buscando observar como a escolha da precisão influencia o comportamento de uma rede convolucional em um experimento controlado. O objetivo não é contestar resultados existentes, mas compreender de forma direta a relação entre representação numérica, custo computacional e desempenho do modelo.

A análise proposta contribui para uma visão mais concreta dos impactos associados ao uso de diferentes precisões, servindo como referência experimental complementar. Esses resultados auxiliam na tomada de decisão sobre configurações numéricas em aplicações práticas de classificação de imagens.

1.6 ORGANIZAÇÃO DO DOCUMENTO

Esta monografia está organizada em seis capítulos principais, contando com esta introdução, onde o Capítulo 2 apresenta a revisão bibliográfica, reunindo os conceitos fundamentais utilizados ao longo do trabalho. São descritos os princípios das redes neurais, com ênfase nas convolucionais, a representação numérica em ponto flutuante, bem como trabalhos relacionados que investigam o uso de diferentes precisões em aprendizado profundo.

O Capítulo 3 apresenta o estado da arte, discutindo pesquisas recentes que são relevantes ao escopo deste trabalho. Incluem-se estudos sobre redes neurais convolucionais, formatos numéricos alternativos e suas aplicações em aprendizado profundo, destacando algumas das contribuições e limitações desses trabalhos.

O Capítulo 4 descreve a metodologia do trabalho. São apresentados a descrição conceitual do estudo, o ambiente de desenvolvimento, os procedimentos de implementação em C++ com LibTorch, a configuração dos experimentos, os conjuntos de dados utilizados e a metodologia empregada para coleta de métricas de desempenho e acurácia.

O Capítulo 5 apresenta e discute os resultados obtidos nos experimentos, incluindo acurácia final em teste, métricas de validação, épocas de convergência e tempos de treinamento para diferentes formatos numéricos e tamanhos de batch. Nesse capítulo são comparadas as configurações de precisão de ponto flutuante aplicadas aos conjuntos PKLot e Fruits360, destacando seus efeitos sobre o desempenho global, a qualidade das classificações e as principais limitações observadas.

Por fim, o Capítulo 6 apresenta a conclusão do trabalho. São sintetizados os principais resultados, discutidas as contribuições do estudo e indicadas possíveis direções para trabalhos futuros relacionados ao uso de diferentes formatos numéricos em redes neurais convolucionais.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 INTRODUÇÃO

Este capítulo apresenta os fundamentos teóricos necessários para a compreensão dos conceitos abordados neste trabalho. São discutidos os princípios da representação numérica em ponto flutuante, as redes neurais e suas aplicações em tarefas de classificação de imagens. Também são apresentadas as ideias centrais das redes neurais convolucionais e a descrição dos conjuntos de dados utilizados nos experimentos.

2.2 REPRESENTAÇÃO NUMÉRICA EM PONTO FLUTUANTE

A representação numérica em ponto flutuante é um método utilizado para representar números reais em computadores (Iskandar, 2000). Neste método, um número real é representado por um par de números inteiros: um para a mantissa e outro para o expoente. A representação em ponto flutuante é dada por:

$$x = m \times b^e \tag{2.1}$$

onde x é o número real, m é a mantissa, b é a base do sistema numérico e e é o expoente. A base b é geralmente 2 ou 10, e a mantissa m é um número real normalizado no intervalo $[1, b)$. A precisão do ponto flutuante é limitada pela quantidade de bits disponíveis para a mantissa e o expoente.

Essa maneira de estruturar um número real foi padronizada pela norma IEEE 754, que define formatos com base em precisões, como meia precisão (FP16), precisão simples (FP32), precisão dupla (FP64) e precisão quádrupla (FP128), sendo representadas em 16, 32, 64 e 128 bits, respectivamente. Dado a natureza finita da representação em ponto flutuante, operações aritméticas podem introduzir erros de arredondamento e perda de precisão. Esses erros podem se acumular em cálculos complexos, afetando a estabilidade numérica de algoritmos e modelos computacionais, como redes neurais.

Portanto, a precisão escolhida para a representação em ponto flutuante pode ter um impacto significativo (Higham, 2002) na acurácia e recursos computacionais necessários para a execução de tais modelos (Wang e Kanwar, 2019).

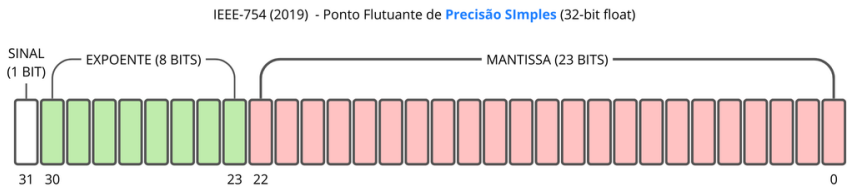


Figura 2.1: Representação de ponto flutuante pelo padrão IEEE 754 com precisão simples.

Fonte: Elaborado pelo autor com base em (IEEE, 2019), 2025.

A Figura 2.1 mostra a representação dos campos de um número em ponto flutuante pelo padrão IEEE 754, com precisão simples, utilizando 32 bits. O bit mais significativo é o sinal, seguido por 8 bits para o expoente e 23 bits para a mantissa. O expoente é representado em excesso de base 127, e a mantissa é normalizada com um bit implícito.

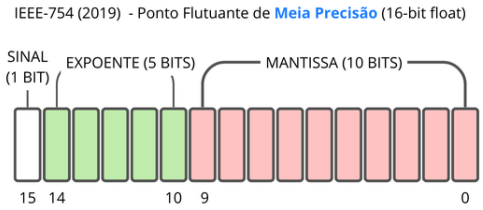


Figura 2.2: Representação de ponto flutuante pelo padrão IEEE 754 com precisão reduzida.

Fonte: Elaborado pelo autor com base em (IEEE, 2019), 2025.

Além da precisão simples, existe a precisão reduzida, conhecida como *half*, representada na Figura 2.2. Nesse formato, o número utiliza 16 bits, com 5 bits para o expoente e 10 bits para a mantissa. Embora seja menos precisa do que a precisão simples, trabalhos em aprendizado de máquina indicam que formatos de 16 bits podem reduzir o uso de memória e acelerar o processamento, em especial em dispositivos com recursos limitados (Yun et al., 2025).

No outro extremo, a precisão dupla, com 64 bits, apresenta maior precisão e maior alcance de representação. Porém, exige mais recursos computacionais, podendo ser problemático para dispositivos mais simples ou para aplicações que demandam alta eficiência. A Figura 2.3 mostra a representação de um número em ponto flutuante com precisão dupla.

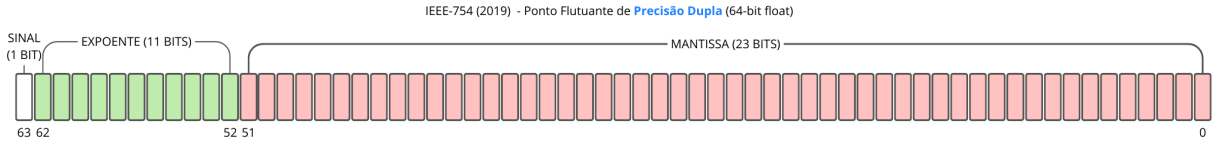


Figura 2.3: Representação de ponto flutuante pelo padrão IEEE 754 com precisão dupla.

Fonte: Elaborado pelo autor com base em (IEEE, 2019), 2025.

Além dessas precisões, existem variações do padrão IEEE 754 que modificam a quantidade de bits para a representação, mas mantendo a mesma estrutura já estabelecida. Como exemplo relevante para este trabalho, tem-se a representação proposta pelo grupo de pesquisa *Google Brain*, denominada *bfloat16* (Wang e Kanwar, 2019).

Esse formato estabelece o uso de 16 bits para representar números em ponto flutuante, sendo 8 bits para o expoente e 7 bits para a mantissa. A Figura 2.4 mostra a representação de um número em ponto flutuante com precisão *bfloat16*.

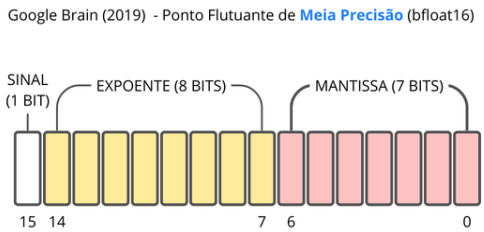


Figura 2.4: Representação de ponto flutuante pelo padrão *bfloat16* com precisão reduzida.

Fonte: Elaborado pelo autor com base em (Wang e Kanwar, 2019), 2025.

A precisão *bfloat16* é uma alternativa interessante para aplicações de inteligência artificial, como redes neurais, pois, pelo fato de manter o mesmo tamanho do expoente da precisão simples, preserva a faixa de representação, facilitando a conversão entre os formatos (Wang e Kanwar, 2019). Isso permite que modelos treinados em precisão simples possam ser

convertidos para *bfloat16* com menor risco de perda de informação, em comparação com outros formatos de 16 bits.

2.3 REDES NEURAIS

Redes neurais artificiais são sistemas computacionais inspirados no funcionamento do cérebro humano, desenvolvidos para resolver problemas de aprendizado de máquina complexos (Rumelhart et al., 1986; Nielsen, 2015). Elas se destacam por sua capacidade de generalizar padrões a partir de dados, sendo amplamente utilizadas em tarefas como reconhecimento de imagens, processamento de linguagem natural e detecção de fraudes.

2.3.1 Perceptron

O perceptron é um dos modelos mais simples e pioneiros de redes neurais, criado por Frank Rosenblatt em 1958 (Rumelhart et al., 1986). Ele é composto por um único neurônio artificial, capaz de realizar classificações lineares. O modelo pode ser matematicamente representado pela equação:

$$y = \phi\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{2.2}$$

onde y é a saída do perceptron, ϕ é a função de ativação (geralmente a função degrau), w_i são os pesos associados às entradas x_i, e b é o termo de ajuste.

O treinamento do perceptron, quando considerado o treinamento por aprendizado supervisionado, é realizado ajustando os pesos de acordo com a diferença entre a saída prevista e o valor esperado. Esse processo, conhecido como *Regra de Aprendizado do Perceptron*, possibilita que a rede aprenda a classificar padrões lineares.

Apesar de inovador, o perceptron apresenta limitações significativas, como a incapacidade de resolver problemas não linearmente separáveis, exemplificados pelo problema XOR. Essas limitações impulsionaram o desenvolvimento de arquiteturas mais avançadas, em particular o perceptron multicamadas, discutido na próxima subseção.

2.3.2 Perceptron Multicamadas (MLP)

O perceptron multicamadas, do inglês *Multilayer Perceptron* (MLP), é uma extensão do modelo básico, que incorpora uma ou mais camadas ocultas entre as camadas de entrada e saída. Esse modelo é capaz de aproximar funções não lineares e resolver problemas mais complexos.

Usando o problema do XOR como exemplo, um possível MLP para resolvê-lo pode ser representado pela arquitetura com uma camada oculta com dois neurônios e uma camada de saída com um neurônio. A entrada dupla, representando os dois bits de entrada do XOR, é processada pela camada oculta, cuja saída é então combinada na camada de saída para produzir o resultado final.

Matematicamente, esse possível MLP pode ser descrito pelas seguintes equações:

$$h_1 = \phi\left(w_{11}x_1 + w_{12}x_2 + b_1\right), \tag{2.3}$$

$$h_2 = \phi\left(w_{21}x_1 + w_{22}x_2 + b_2\right), \tag{2.4}$$

$$y = \phi \left(v_1 h_1 + v_2 h_2 + b_3 \right), \tag{2.5}$$

onde h_1 e h_2 são as saídas dos neurônios da camada oculta, representados pelas Equações 2.3 e 2.4, y é a saída final do MLP, representada pela Equação 2.5, w_{ij} são os pesos das conexões entre a camada de entrada e a camada oculta, v_i são os pesos das conexões entre a camada oculta e a camada de saída, b_i são os termos de ajuste, e ϕ é a função de ativação (geralmente a função sigmoide).

A Figura 2.5 ilustra essa arquitetura, com os mesmos elementos descritos nas equações.

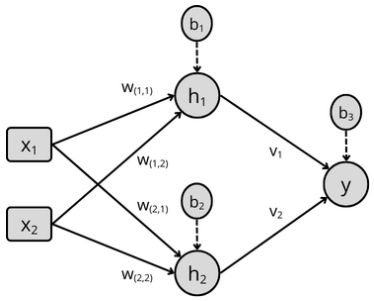


Figura 2.5: Arquitetura de um MLP para resolver o problema XOR.

Fonte: Elaborado pelo autor, 2025.

O MLP é comumente treinado por meio do algoritmo de retropropagação, que utiliza o algoritmo de gradiente descendente para minimizar o erro entre a saída prevista e a saída esperada (Rumelhart et al., 1986; Nielsen, 2015). Esse processo envolve o cálculo do gradiente do erro em relação aos pesos da rede, permitindo ajustes iterativos que melhoram a performance do modelo ao longo do tempo.

Dada a sua capacidade de modelar relações não lineares, o MLP é amplamente utilizado em diversas aplicações de aprendizado de máquina, incluindo classificação, regressão e reconhecimento de padrões (Nie et al., 2021; Chang, 2022; Tolstikhin et al., 2021). No entanto, o MLP também apresenta limitações, como a necessidade de grandes quantidades de dados para treinamento eficaz e a suscetibilidade ao sobreajuste, especialmente em redes profundas. Essas limitações motivaram o desenvolvimento de arquiteturas mais especializadas, como as redes neurais convolucionais, discutidas na próxima seção.

2.4 REDES NEURAIS CONVOLUCIONAIS (CNN)

As Redes Neurais Convolucionais (CNN) são uma classe especial de redes neurais que têm sido amplamente utilizadas em tarefas de reconhecimento e classificação de imagens, sendo particularmente eficientes para trabalhar com dados que possuem estrutura espacial, como imagens e vídeos (LeCun et al., 1998; Goodfellow et al., 2016).

A arquitetura de uma CNN é composta por uma sequência de camadas que realizam operações específicas para extrair e processar características dos dados de entrada. Em geral, essas camadas podem ser classificadas em camadas convolucionais, camadas de agrupamento e camadas totalmente conectadas, além de camadas de ativação não linear que intercalam esses blocos e permitem que a rede aprenda funções de decisão mais complexas (Goodfellow et al., 2016).

Em arquiteturas modernas também é comum empregar camadas adicionais de normalização e *dropout*, que auxiliam na estabilização do treinamento e na redução de overfitting, mas

esses recursos não serão foco deste trabalho. A combinação desses elementos define o fluxo de informação desde a imagem de entrada até o vetor de saídas associado às classes do problema.

A camada convolucional aplica filtros sobre os dados de entrada, gerando mapas de ativação que destacam padrões específicos. Matematicamente, a operação de convolução pode ser representada como:

$$z_{ij} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} w_{mn} \cdot x_{(i+m)(j+n)} + b, \tag{2.6}$$

onde z_{ij} é o valor do mapa de ativação na posição (i, j), w_{mn} são os pesos do filtro, $x_{(i+m)(j+n)}$ são os valores da entrada, e b é o termo de ajuste (Goodfellow et al., 2016). A aplicação de múltiplos filtros permite que a rede aprenda diferentes padrões, como bordas, texturas e formas. Nesse contexto, a convolução tende a ser mais eficiente no uso de memória e processamento, pois reutiliza filtros locais e reduz a quantidade de pesos em relação a uma camada densa tradicional, diminuindo o custo computacional sem perda proporcional de capacidade expressiva (LeCun et al., 1998).

Após a convolução, as camadas de agrupamento (*pooling*) são utilizadas para reduzir a dimensionalidade dos mapas de ativação, agregando informações essenciais e descartando redundâncias. Os métodos mais comuns de agrupamento são o máximo (*max pooling*) e a média (*average pooling*) (Goodfellow et al., 2016). Por exemplo, o agrupamento máximo seleciona o maior valor em uma região, o que ajuda a capturar características mais dominantes. Essa operação é representada por:

$$z_{ij} = \max_{(m,n)\in R} x_{(i+m)(j+n)}, \tag{2.7}$$

onde R é a região de agrupamento (Krizhevsky et al., 2012).

Em redes neurais convolucionais modernas, as camadas convolucionais são tipicamente seguidas por funções de ativação não lineares. Uma escolha comum é a ReLU (*Rectified Linear Unit*), definida por $f(x) = \max(0, x)$, que zera valores negativos e mantém valores positivos. Essa função introduz não linearidade de forma simples e eficiente, evitando a saturação observada em funções como a sigmoide e facilitando a propagação do gradiente em arquiteturas profundas (Goodfellow et al., 2016).

A inicialização dos pesos influencia diretamente a estabilidade do treinamento. No contexto de redes com ReLU, é frequente o uso da inicialização proposta em (He et al., 2015), conhecida como inicialização de Kaiming, que ajusta a variância dos pesos de acordo com o número de conexões de entrada de cada neurônio. Essa estratégia procura manter a distribuição dos ativamentos aproximadamente estável ao longo das camadas, reduzindo o risco de explosão ou desaparecimento de gradientes durante o processo de otimização.

Após blocos convolucionais e de ativação, é comum empregar camadas de *max pooling* com janelas reduzidas, como 2×2, e passo 2. Essas camadas realizam uma redução amostral dos mapas de ativação, selecionando o maior valor em cada região de interesse. O efeito é a redução controlada das dimensões espaciais, preservando as ativações mais fortes e tornando o modelo menos sensível a pequenas variações de posição nas entradas.

À medida que as camadas convolucionais e de agrupamento são aplicadas, os mapas de ativação usualmente têm redução das dimensões espaciais e aumento do número de canais. Antes da etapa de classificação, aplica-se o achatamento (*flatten*), concatenando as ativações de cada canal em um único vetor unidimensional que resume a representação extraída pela base convolucional.

Esse vetor é então alimentado em uma ou mais camadas totalmente conectadas (*fully connected*), que realizam a classificação final. Essas camadas funcionam de maneira semelhante ao perceptron multicamadas, onde cada neurônio está conectado a todos os neurônios da camada anterior. A última camada totalmente conectada geralmente utiliza uma função de ativação apropriada para o problema.

Como exemplo clássico, a arquitetura AlexNet (Krizhevsky et al., 2012), proposta por Alex Krizhevsky e colaboradores em 2012, revolucionou o campo do reconhecimento de imagens ao vencer a competição ImageNet em 2012 com uma margem significativa. A AlexNet é composta por cinco camadas convolucionais seguida por três camadas totalmente conectadas, utilizando ReLU como função de ativação e *max pooling* para redução espacial. Essa arquitetura demonstrou a eficácia das CNNs em tarefas complexas de visão computacional, estabelecendo um novo padrão para o desenvolvimento de modelos subsequentes.

2.5 PRECISÃO NUMÉRICA E EFICIÊNCIA EM REDES NEURAIS

Diversos estudos recentes investigam o uso de formatos numéricos reduzidos e configurações de precisão mista em redes neurais, com o objetivo de reduzir o consumo de memória, o tempo de treinamento e a energia gasta, mantendo níveis de acurácia considerados adequados (Micikevicius et al., 2018b; Ni et al., 2019). Essas abordagens exploram, por exemplo, o uso combinado de formatos de 16 e 32 bits em diferentes partes do modelo, além de técnicas de quantização pós-treinamento.

Além da redução de memória, escolhas cuidadosas de formato numérico podem trazer ganhos concretos de eficiência no uso de recursos em aceleradores modernos e GPUs (Wang et al., 2018, 2019). Esses ganhos são particularmente relevantes quando se considera a implantação de modelos em dispositivos com recursos limitados, como sistemas embarcados e dispositivos de borda.

Em ambientes embarcados, restrições de energia, memória e capacidade de processamento impõem limites práticos ao tamanho dos modelos e às configurações numéricas adotadas. Nesses cenários, formatos reduzidos, como *half* e *bfloat16*, podem viabilizar o uso de redes convolucionais em aplicações de visão computacional, desde que a perda de precisão seja controlada.

Este trabalho insere-se neste contexto ao comparar, em um cenário controlado, o efeito de diferentes formatos de ponto flutuante sobre desempenho, acurácia e consumo de recursos em redes convolucionais avaliadas sobre os conjuntos PKLot e Fruits360.

Os avanços recentes em aprendizado profundo também têm sido acompanhados por um expressivo aumento no tamanho de modelos e no custo computacional associado ao seu treinamento e inferência (Hoffmann et al., 2022). Esse crescimento reforça a importância de configurações numéricas e arquiteturais mais compactas, que reduzam o uso de memória e o tempo de processamento, sem comprometer de forma relevante a acurácia.

3 ESTADO DA ARTE

3.1 INTRODUÇÃO

Este capítulo apresenta os principais trabalhos relacionados ao uso de formatos numéricos reduzidos em redes neurais profundas, com ênfase em configurações de precisão mista e no impacto de formatos como FP32, FP16 e *bfloat16* sobre desempenho, acurácia e convergência. Em seguida, são discutidas aplicações dessas técnicas em visão computacional, aproximando o estado da arte do problema investigado nesta monografia.

3.2 REDES CONVOLUCIONAIS CLÁSSICAS

As primeiras grandes conquistas em classificação de imagens com redes convolucionais ocorreram com arquiteturas profundas treinadas em FP32 em GPUs. Em (Krizhevsky et al., 2012) é proposta a AlexNet, que alcança 15,3% de erro Top-5 no ImageNet, contra 26,2% do melhor método anterior, estabelecendo o uso de CNNs profundas como padrão em visão computacional. Trabalhos posteriores, como (He et al., 2016; Cireşan et al., 2012), mostram que o aumento da profundidade e o uso de conexões residuais permitem modelos ainda mais precisos, ao custo de redes maiores e mais caras.

Seguindo essa linha, arquiteturas eficientes como MobileNetV2 e MobileNetV3 buscam reduzir a complexidade mantendo boa acurácia em ImageNet. Em (Sandler et al., 2018) são introduzidos blocos com *depthwise separable convolutions* e gargalos lineares, que diminuem de forma expressiva o número de operações de ponto flutuante para uma dada acurácia. Em (Howard et al., 2019), emprega-se busca automatizada de arquitetura combinada a otimizações de blocos convolucionais para obter modelos que superam variantes anteriores em relação Top-1/latência, usando menos operações.

Esses resultados consolidam a visão de que CNNs profundas em FP32 se tornaram referência em tarefas de visão computacional, enquanto variantes móveis como MobileNetV3 demonstram que é possível reduzir o custo computacional mantendo acurácia competitiva. Em ambos os casos, o desempenho é fortemente influenciado pelo número de operações em ponto flutuante, o que motiva investigações sobre o impacto direto do formato numérico utilizado no treinamento e na inferência.

3.3 TREINAMENTO EM PRECISÃO REDUZIDA E QUANTIZAÇÃO

A redução da precisão numérica em redes profundas tem sido estudada tanto na forma de quantização quanto por meio de treinamento em precisão mista. Em (Micikevicius et al., 2018a) é proposta uma estratégia de treinamento que combina pesos e ativações em FP16 com acumulação em FP32, explorando o suporte nativo de GPUs modernas a operações de 16 bits. Os autores relatam reduções expressivas no consumo de memória e no tempo de treinamento em arquiteturas como ResNet e Inception, com diferenças de acurácia inferiores a um ponto percentual em relação ao treinamento integral em FP32.

Resultados complementares aparecem em (Jia et al., 2018), onde se demonstra o treinamento de modelos para ImageNet em poucos minutos, combinando paralelismo em larga

escala com técnicas de mixed precision. Nesses cenários, utiliza-se também *loss scaling* para mitigar *underflow* de gradientes em FP16, evidenciando que o ganho computacional vem acompanhado de desafios de estabilidade numérica que precisam ser tratados de forma sistemática.

Além de mixed precision, a quantização pós-treinamento e durante o treinamento tem sido explorada como forma de reduzir ainda mais o custo. Em (Schaefer et al., 2023) é proposta uma abordagem de quantização pós-treinamento com precisão mista, guiada por sensibilidade, que ajusta o número de bits por camada conforme o impacto estimado na acurácia. Estudos como (Gernigon et al., 2023) investigam ainda formatos de baixa precisão para inferência em redes convolucionais, discutindo o compromisso entre erro numérico e ganho de desempenho.

De forma geral, esses trabalhos indicam que CNNs podem ser treinadas e implantadas com precisões inferiores a FP32, desde que se adotem estratégias adequadas de estabilização numérica e seleção de formato por componente do modelo. Essa linha de pesquisa fornece o pano de fundo para a análise, neste trabalho, do impacto de formatos como FP64, FP32 e *bfloat16* em uma CNN de porte moderado.

3.4 PONTO FLUTUANTE EM REDES NEURAIS

O comportamento numérico de algoritmos de aprendizado profundo depende diretamente das propriedades da aritmética de ponto flutuante utilizada. A norma IEEE 754 define os formatos binários amplamente adotados em hardware geral, incluindo meia precisão (FP16), precisão simples (FP32) e precisão dupla (FP64) (IEEE, 2019). Conforme discutido em (Higham, 2002), a escolha de precisão afeta o acúmulo de erros de arredondamento e a estabilidade de métodos iterativos, aspectos que se tornam críticos em redes profundas com grande número de operações encadeadas.

Estudos recentes investigam especificamente o uso de formatos reduzidos em treinamento de redes neurais. Em (Kalamkar et al., 2019) é apresentada uma análise sistemática do formato *bfloat16* em diversos modelos, mostrando que, ao preservar a faixa dinâmica de FP32 e reduzir apenas a mantissa, obtêm-se redes com acurácia próxima à de FP32 em tarefas de visão e linguagem, com ganhos relevantes de desempenho. Também em (Micikevicius et al., 2018a), reforça-se que FP16 pode exigir cuidados adicionais, devido à faixa dinâmica mais estreita, o que aumenta a incidência de *overflow* e *underflow* em gradientes.

Por outro lado, (Gernigon et al., 2023) investiga combinações de formatos de baixa precisão em redes convolucionais, avaliando empiricamente o impacto de diferentes configurações sobre erro numérico e custo computacional. Esses estudos apontam que não há uma única precisão ótima para todos os componentes da rede, motivando abordagens em que se calibram formatos por camada ou por tipo de operação, a fim de equilibrar consumo de recursos e qualidade de predição.

O conjunto dessas evidências empíricas sustenta a hipótese central deste trabalho: varia-se a precisão numérica de uma CNN relativamente simples e avaliam-se, de forma controlada, seus efeitos sobre tempo de treinamento, acurácia e convergência em problemas de classificação de imagens estruturalmente semelhantes aos adotados em estudos clássicos de visão computacional.

4 METODOLOGIA

4.1 INTRODUÇÃO

A metodologia adotada neste trabalho foi concebida para avaliar, de forma controlada, como diferentes precisões de ponto flutuante afetam o treinamento e a inferência de redes neurais convolucionais em tarefas de classificação de imagens. Este capítulo apresenta, em ordem, o ambiente de hardware e ferramentas utilizados, os conjuntos de dados empregados, o processo de pré-processamento aplicado às imagens, a arquitetura convolucional adotada, o procedimento de treinamento e validação e, por fim, o protocolo experimental que organiza as combinações de formatos numéricos e conjuntos de dados.

4.2 AMBIENTE DE HARDWARE E FERRAMENTAS

Os experimentos foram executados em uma máquina local com sistema operacional Linux e uma GPU dedicada NVIDIA GeForce RTX 2070 SUPER com 8 GB de memória. O projeto principal foi desenvolvido em C++ com a interface LibTorch da biblioteca PyTorch (Paszke et al., 2019), compilado com CMake e padrão C++17, utilizando a pilha CUDA da NVIDIA para acelerar operações matriciais e convolucionais (NVIDIA Corporation, 2025).

O pré-processamento dos conjuntos de dados foi implementado em Python com a mesma biblioteca PyTorch (versão 2.9.1+cu126), visando simplificar os scripts de preparação e manter compatibilidade com o ambiente de GPU.

4.3 CONJUNTOS DE DADOS UTILIZADOS

Os experimentos deste trabalho utilizam dois conjuntos de dados de classificação de imagens amplamente empregados na literatura: PKLot e Fruits360. Ambos permitem avaliar o comportamento de redes neurais convolucionais em cenários de baixa complexidade em termos de estrutura de classes, mas com variações relevantes de iluminação, perspectiva e textura.

A Tabela 4.1 resume as principais características de cada conjunto de dados, incluindo o número de classes, imagens e resolução.

Tabela 4.1: Características dos conjuntos de dados utilizados nos experimentos.

Conjunto de Dados	**Número de Classes**	**Número de Imagens**
PKLot	2	695.899
Fruits360	225	156.788

Esses conjuntos foram escolhidos por sua diversidade em termos de número de classes e complexidade visual, permitindo realizar o experimento com um problema de classificação binária (PKLot) e outro de classificação multiclasse (Fruits360). A seguir, são descritas as principais características de cada conjunto.

4.3.1 PKLot

O PKLot é um conjunto de dados voltado para a classificação de vagas de estacionamento como livres ou ocupadas (Almeida et al., 2015). O conjunto foi construído a partir de imagens

capturadas em estacionamentos da Universidade Federal do Paraná e PUC-PR, em diferentes condições de iluminação e clima (dias ensolarados, nublados e chuvosos).

Esse *dataset* contém centenas de milhares de recortes de vagas rotuladas, obtidos a partir de múltiplas vistas de câmeras fixas. Cada recorte é associado a uma classe binária, indicando se a vaga está ocupada ou livre, o que o torna adequado para experimentos de classificação binária com redes neurais convolucionais.

Como um conjunto de dados realista, o PKLot apresenta variações significativas de iluminação, ângulo de visão e obstruções parciais, o que desafia a capacidade de generalização dos modelos treinados. Por essa razão, o conjunto também é amplamente utilizado como referência em estudos de visão computacional, como em (Hochuli et al., 2022).

4.3.2 Fruits360

O Fruits360 é um conjunto de dados composto por imagens de frutas e vegetais, organizado em múltiplas classes correspondentes a diferentes tipos de frutas. As imagens são tipicamente capturadas em fundos controlados, com variações moderadas de posição e orientação, o que resulta em um cenário de classificação multiclasse de complexidade intermediária.

Cada amostra do Fruits360 contém uma única fruta centralizada em uma imagem colorida, permitindo explorar a capacidade das redes convolucionais em capturar padrões de cor e textura em condições relativamente controladas. O conjunto é amplamente utilizado em exemplos didáticos e em estudos experimentais de classificação de imagens com redes neurais (Dandekar et al., 2021).

O Fruits360 oferece um cenário complementar ao PKLot, permitindo avaliar o impacto dos formatos de ponto flutuante em uma tarefa de classificação multiclasse, com um número significativamente maior de classes (225) em comparação ao PKLot (2). Para esse experimento, foi adotada a versão com imagens na resolução de 100×100 pixels (Oltean, 2017).

4.4 PRÉ-PROCESSAMENTO DOS CONJUNTOS DE DADOS

Para preparar os conjuntos de dados, foi realizado um processo de pré-processamento que inclui a divisão em subconjuntos de treino, validação e teste, além da aplicação de transformações nas imagens. Ainda que não tenha sido realizada a comparação entre diferentes arquiteturas ou ajustes de hiperparâmetros, essa divisão foi mantida para preservar a avaliação em dados não utilizados diretamente na otimização do modelo.

O conjunto de validação é empregado para acompanhar o comportamento de convergência da rede ao longo do treinamento. Esse acompanhamento permite identificar indícios de sobreajuste e eventuais instabilidades numéricas, sendo utilizados, por exemplo, para avaliar o critério de parada antecipada. O conjunto de teste, por sua vez, é reservado para a avaliação final do modelo, permitindo uma estimativa imparcial da acurácia da rede em dados não vistos durante o treinamento. Cada conjunto é particionado em três subconjuntos disjuntos, na proporção de 60% para treino, 30% para validação e 10% para teste.

No PKLot, a divisão segue o critério proposto em (Hochuli et al., 2022), baseado em dias de captura. Logo, os primeiros 60% dos dias compõem o conjunto de treino, os 30% seguintes formam o conjunto de validação e os 10% finais são reservados para teste. Esse procedimento evita que diferentes vistas do mesmo dia apareçam simultaneamente em treino e teste. Aliás, com o passar dos dias, as condições de iluminação e clima variam, o que contribui para uma avaliação mais robusta do modelo.

No Fruits360, o conjunto já apresenta uma divisão pré-definida entre treino e teste (75% e 25%). A partir dessa partição, ajusta-se a proporção desejada para este trabalho, destinando

60% das imagens de treino ao novo conjunto de treinamento e utilizando os 15 pontos percentuais restantes, em conjunto com o conjunto de teste original, para compor os subconjuntos de validação e teste.

Embora o tamanho bruto das imagens do conjunto PKLot sugira que os dados pudessem caber integralmente na memória principal, essa estimativa não considera o custo real da representação em tensores durante o treinamento. Os subconjuntos de treinamento, validação e teste armazenados como tensores ocuparam aproximadamente 5 GB, 2 GB e 1 GB, respectivamente. Além disso, operações internas do framework durante a criação dos carregadores de dados resultaram em picos adicionais de uso de memória. Esses picos levaram a estouros de memória, inviabilizando o carregamento integral do conjunto.

Outro fator relevante foi a normalização dos lotes de imagens, realizada sob demanda durante o treinamento. As imagens foram originalmente armazenadas como *uint8* e convertidas para *float32* a cada lote, o que quadruplica temporariamente o consumo de memória. Para contornar essa limitação, o conjunto PKLot foi particionado em blocos menores, com tamanho máximo de aproximadamente 2 GB. Cada bloco foi processado sequencialmente, permitindo a execução do treinamento sem exceder a capacidade de memória disponível.

Com o intuito de reduzir o espaço ocupado em disco, as imagens são armazenadas em formato inteiro de 8 bits por canal (*uint8*), sem qualquer normalização prévia nos arquivos. A conversão para ponto flutuante e a normalização são realizadas apenas no momento do carregamento, durante o treinamento e a inferência. Essa abordagem é aplicada a ambos os conjuntos de dados e mantém a escolha do formato de ponto flutuante sob controle do programa principal.

Para cada conjunto, são calculadas previamente a média e o desvio padrão por canal, a partir das imagens já redimensionadas [1]. Esses valores são então utilizados em tempo de execução para normalizar as amostras, após a conversão para o formato *float32*. A operação de normalização adotada segue a forma:

$$\text{normalização}(x) = \frac{\frac{x}{255} - \mu}{\sigma}, \tag{4.1}$$

em que x representa o valor inteiro original em *uint8*, e μ e σ denotam, respectivamente, o vetor de médias e o vetor de desvios padrão estimados para cada canal do conjunto de dados em questão. Os valores específicos de μ e σ utilizados para o PKLot e para o Fruits360 são apresentados na Tabela 4.2.

Tabela 4.2: Valores de média (μ) e desvio padrão (σ) por canal utilizados na normalização dos conjuntos de dados.

Conjunto de Dados	μ (R, G, B)	σ (R, G, B)
PKLot	(0.527, 0.523, 0.499)	(0.240, 0.237, 0.237)
Fruits360	(0.657, 0.567, 0.504)	(0.319, 0.366, 0.396)

Após a conclusão do preprocessamento, os tamanhos finais dos conjuntos de dados são apresentados na Tabela 4.3. Esses valores refletem a divisão adotada e o número total de imagens disponíveis em cada conjunto após a aplicação das transformações descritas.

[1] Muitos frameworks de aprendizado profundo, como PyTorch utilizam o padrão ImageNet para normalização, mas isso é recomendado para quando é utilizado um modelo pré-treinado nessa base. Como o presente trabalho utiliza uma arquitetura simplificada sem pré-treinamento, optou-se por calcular média e desvio padrão específicos para cada conjunto.

Tabela 4.3: Tamanhos dos conjuntos de dados após o preprocessamento.

Conjunto de Dados	Treino	Validação	Teste
PKLot	429.131	189.586	77.134
Fruits360	94.094	47.054	15.640

Neste trabalho, não foi realizada uma análise específica do balanceamento das classes nos conjuntos de dados utilizados. Assim, a avaliação do desempenho concentrou-se na acurácia e no comportamento de convergência da rede ao longo do treinamento. Métricas mais sensíveis à distribuição das classes não foram consideradas nesta etapa.

Essa escolha está alinhada ao caráter exploratório do estudo, cujo foco principal é a análise do impacto da precisão numérica no treinamento de redes neurais convolucionais. A verificação do balanceamento das classes e a inclusão de métricas adicionais, como precisão, revocação e F1-score, são tratadas como limitações do trabalho. Esses aspectos são discutidos como possibilidades para investigações futuras.

4.5 MODELO CONVOLUCIONAL E CARREGAMENTO DE DADOS

A tarefa considerada é a classificação de imagens, para a qual redes neurais convolucionais são amplamente empregadas em cenários acadêmicos e aplicados. A arquitetura adotada neste trabalho foi inspirada em padrões consagrados da literatura, como a AlexNet, no sentido de empregar uma sequência de camadas convolucionais para extração hierárquica de características, seguida por camadas totalmente conectadas para a decisão de classificação. Entretanto, optou-se por uma configuração substancialmente mais simples, com foco em reduzir a complexidade do modelo.

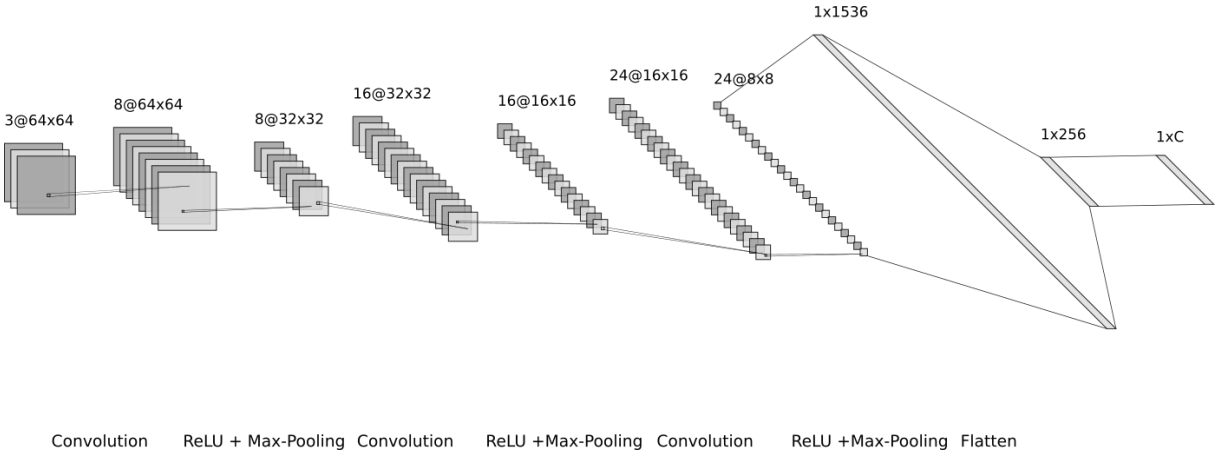
O objetivo dessa escolha arquitetural não foi alcançar desempenho de estado da arte, mas construir um modelo de porte moderado, com número limitado de parâmetros e custo computacional controlado. Essa decisão permite analisar de forma mais direta o impacto da precisão numérica, evitando que efeitos associados a arquiteturas profundas ou altamente otimizadas mascarem os resultados experimentais. Assim, buscou-se um compromisso entre capacidade de representação e simplicidade estrutural.

A rede consiste em três camadas convolucionais, cada uma seguida por uma camada de *max pooling*, e duas camadas totalmente conectadas (*fully connected*). As camadas convolucionais utilizam filtros de tamanho 3×3, com preenchimento (*padding*) de 1 e passo (*stride*) de 1, preservando as dimensões espaciais das imagens. As camadas de *max pooling* reduzem as dimensões espaciais pela metade, enquanto as camadas totalmente conectadas processam as representações extraídas para produzir a saída final.

Uma representação esquemática da arquitetura é apresentada na Figura 4.1, na qual se observam as transformações aplicadas às imagens ao longo das camadas da rede. A imagem de entrada é representada como um tensor de dimensão $3 \times 64 \times 64$, correspondente aos canais RGB e às dimensões espaciais. A saída final da rede é um vetor de dimensão C, onde C é o número de classes do conjunto de dados considerado.

Como apenas a camada totalmente conectada final depende diretamente do número de classes, o total de parâmetros treináveis varia em pequena quantidade entre os conjuntos avaliados. Para o PKLot, com duas classes, o modelo possui 398 858 parâmetros treináveis; para o Fruits360, com 225 classes, o total é de 456 169 parâmetros treináveis, conforme apresentado

Figura 4.1: Arquitetura da rede neural convolucional utilizada nos experimentos.



Fonte: Elaborado pelo autor com uso da ferramenta (LeNail, 2019).

na Tabela 4.4. Esses valores caracterizam o modelo como uma rede de porte moderado, adequada para experimentos controlados.

Tabela 4.4: Parâmetros treináveis do modelo para cada conjunto de dados.

Conjunto de Dados	Número de classes	Parâmetros treináveis
PKLot	2	398 858
Fruits360	225	456 169

O número de parâmetros foi obtido a partir das dimensões de cada camada. Nas camadas convolucionais, considera-se o produto $C_{out} \times C_{in} \times k \times k$, enquanto nas camadas lineares utiliza-se o produto entre o número de entradas e saídas. As dimensões espaciais foram propagadas analiticamente ao longo das operações da rede. O detalhamento completo desses cálculos é apresentado no Apêndice A.

Os pesos da rede são inicializados segundo a estratégia de Kaiming Uniforme (He et al., 2015), adequada para redes com funções de ativação ReLU. O treinamento utiliza o otimizador Adam e a função de perda de entropia cruzada, escolhas amplamente consolidadas para problemas de classificação (Kingma e Ba, 2017; Goodfellow et al., 2016).

Para aceleração do treinamento e da inferência, utilizam-se os carregadores de dados (*data loaders*) fornecidos pela LibTorch. Esses componentes são responsáveis por gerenciar o acesso eficiente aos dados durante o processo de treinamento e avaliação. A arquitetura da rede e a estratégia de carregamento de dados permanecem idênticas em todas as configurações numéricas avaliadas, garantindo uma comparação justa entre os diferentes formatos de ponto flutuante.

4.6 PROCEDIMENTO DE TREINAMENTO E VALIDAÇÃO

O treinamento é limitado a, no máximo, trinta épocas. Em cada época, o modelo é atualizado com o conjunto de treino e, em seguida, avaliado com o conjunto de validação. Ao final de cada época, são computados o valor médio da perda no conjunto de validação, o tempo total gasto no treinamento, o valor médio da função de perda e a acurácia no conjunto de teste.

Para controlar o sobreajuste e evitar o uso desnecessário de recursos computacionais, aplica-se um critério de parada antecipada baseado exclusivamente no comportamento da perda de validação. O método adotado segue a formulação clássica apresentada por Prechelt (Prechelt, 1998), segundo a qual o treinamento deve ser interrompido quando a perda no conjunto de validação deixa de apresentar melhora após um certo número de épocas consecutivas, chamado de *patience*.

Essa abordagem baseia-se na observação de que, durante o treinamento, a perda no conjunto de validação tende a diminuir inicialmente, refletindo a capacidade do modelo de aprender padrões relevantes. Após determinado ponto, a perda de validação pode começar a aumentar, indicando que o modelo está se ajustando demais aos dados de treinamento e perdendo capacidade de generalização para novas amostras. Trata-se de um regulador eficaz (Goodfellow et al., 2016), amplamente adotado em implementações modernas de bibliotecas de aprendizado profundo (Chollet et al., 2015).

Para o experimento, o valor de *patience* é fixado em cinco épocas. O treinamento é interrompido caso a perda de validação não apresente melhora durante esse intervalo, preservando como resultado final o modelo correspondente à menor perda registrada. Para redes pequenas, como a utilizada neste trabalho, esse valor de *patience* é suficiente para evitar o sobreajuste sem comprometer a eficiência do processo de treinamento.

Por fim, ao finalizar o processo de treinamento — seja pelo limite máximo de épocas, seja pela ativação do critério de parada antecipada — é realizada uma avaliação final utilizando o *data loader* de teste. Para avaliar o desempenho do modelo treinado, são calculadas as seguintes métricas durante todo o experimento:

- Tempo total de treinamento (em segundos)
- Número de épocas até a convergência
- Valor médio da perda no conjunto de validação
- Valor médio da perda no conjunto de teste
- Acurácia no conjunto de teste

4.7 PROTOCOLO EXPERIMENTAL

O protocolo experimental foi definido de forma a garantir que a única variável independente entre as condições avaliadas seja o formato de ponto flutuante utilizado nas operações numéricas. Mantêm-se fixos o hardware, as arquiteturas de rede e de carregamento de dados, o processo de pré-processamento, os hiperparâmetros de treinamento e as partições de treino, validação e teste.

Os experimentos consideram três formatos de ponto flutuante amplamente utilizados em aprendizado de máquina: precisão dupla IEEE 754 (FP64), precisão simples IEEE 754 (FP32) e o formato reduzido *bfloat16*. Esses formatos representam, respectivamente, um limite superior de precisão, a configuração operacional padrão e uma alternativa compacta com menor espaço de mantissa. A arquitetura da rede permanece idêntica em todas as execuções, variando apenas o tipo numérico interno utilizado nos tensores e parâmetros.

Para cada combinação de conjunto de dados (PKLot e Fruits360) e formato numérico (FP64, FP32 e *bfloat16*), o programa principal é executado cinco vezes de forma independente. As execuções diferem apenas nas inicializações aleatórias dos pesos da rede e na ordem de apresentação das amostras em cada época. Emprega-se uma semente fixa para o gerador de

números aleatórios, iniciando em 42 e sendo incrementada em uma unidade a cada nova execução, o que facilita a reprodutibilidade dos resultados.

Todas as execuções são realizadas no mesmo hardware descrito na Seção 4.3, sem carga concorrente significativa, assegurando condições equivalentes de medição de tempo. Utiliza-se um tamanho de lote (*batch size*) igual a 128 em todos os cenários, e o modelo é treinado e avaliado conforme o procedimento descrito na Seção 4.6.

Nas análises apresentadas no Capítulo 5, são consideradas as médias das métricas obtidas ao longo das cinco execuções para cada combinação de conjunto de dados e formato numérico. Essa estratégia reduz o impacto da variabilidade inerente ao processo de otimização estocástica e permite comparar, de forma robusta, o comportamento de cada precisão numérica no treinamento e na inferência.

5 VALIDAÇÃO

5.1 INTRODUÇÃO

Nesse capítulo, são apresentados os resultados dos experimentos definidos no capítulo anterior. Para cada combinação de conjunto de dados e formato numérico, os valores reportados correspondem à média de cinco execuções independentes, diferenciadas apenas pelas sementes aleatórias utilizadas na inicialização dos pesos e na ordem de apresentação das amostras.

Os resultados são organizados em três partes: PKLot, Fruits360 e uma seção de comparação e análise integrada. Em cada conjunto, os efeitos da precisão numérica são discutidos à luz da variabilidade estocástica do treinamento e do custo computacional associado.

5.2 RESULTADOS PARA O CONJUNTO PKLOT

5.2.1 Acurácia Final em Teste

Tabela 5.1: Test accuracy médio por formato numérico (PKLot)

Formato	Test accuracy
bfloat16	0.9895
float32	0.9935
float64	0.9937

A acurácia final em teste para PKLot está na Tabela 5.1. Os três formatos obtiveram valores acima de 98,9%, com FP64 em torno de 99,37%, FP32 cerca de 99,35% e *bfloat16* aproximadamente 98,95%. As diferenças observadas, inferiores a 0,5 ponto percentual, são pequenas e podem se confundir com a variabilidade estocástica inerente ao treinamento, uma vez que os valores apresentados correspondem a médias de múltiplas execuções independentes.

5.2.2 Época de Convergência

Tabela 5.2: Época média de convergência (early stopping) por formato numérico (PKLot)

Formato	Época média de convergência
bfloat16	15.40
float32	16.20
float64	13.20

A Tabela 5.2 apresenta a época média até a parada antecipada. FP64 converge em cerca de 13,2 épocas, *bfloat16* em torno de 15,4 e FP32 próximo de 16,2. Há leve vantagem de FP64 em um problema binário simples. Nesse cenário, todos convergem de forma estável e bem antes do limite de trinta épocas.

5.2.3 Tempo de Treinamento

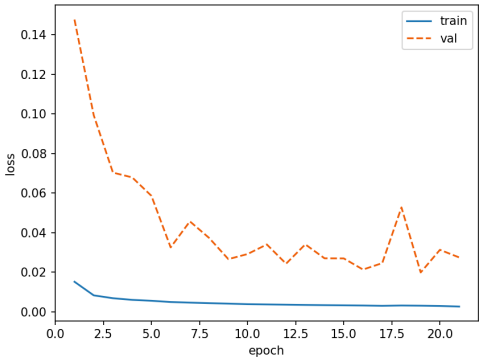
A Tabela 5.3 detalha o tempo de treinamento. FP32 foi o mais eficiente: 664,8 s (aproximadamente 11 min). FP64 levou 2057,6 s (aproximadamente 34 min) e *bfloat16* 2394,3 s

Tabela 5.3: Tempo médio total de treinamento (ms) por formato numérico (PKLot)

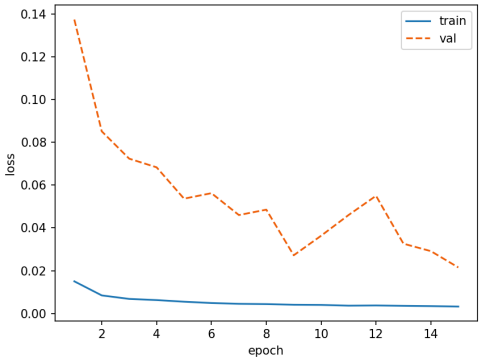
Formato	Tempo total de treinamento (ms)
bfloat16	2394322
float32	664841
float64	2057627

(aproximadamente 40 min). Isso mostra custo temporal elevado ao aumentar precisão e que *bfloat16* não trouxe aceleração nesta configuração.

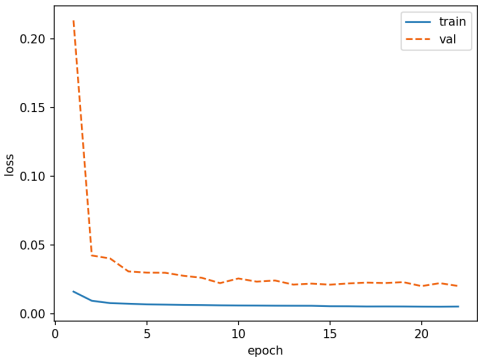
5.2.4 Curvas de Treinamento



(a) Curva de perda para PKLot com FP32



(b) Curva de perda para PKLot com FP64



(c) Curva de perda para PKLot com bfloat16

Figura 5.1: Curvas de perda durante o treinamento no conjunto PKLot para diferentes formatos numéricos.

A Figura 5.1 mostra as curvas de perda para o conjunto PKLot. Todos os formatos reduzem rapidamente a perda de treino e atingem estabilização em valores baixos. Na validação, o formato *bfloat16* apresenta trajetória mais suave e valores próximos de 0,02, enquanto FP32 e FP64 exibem oscilações ligeiramente maiores e picos tardios, ainda assim dentro de uma faixa reduzida.

Considerando que o PKLot representa um problema binário de baixa complexidade, essas diferenças visuais são limitadas. O comportamento global das curvas indica que as variações observadas decorrem principalmente da dinâmica de otimização e da inicialização dos pesos, não de efeitos sistemáticos associados ao formato numérico.

5.2.5 Síntese Parcial para PKLot

Tabela 5.4: Resumo por formato numérico (PKLot)

Tipo	Acurácia Teste (média)	Perda Teste (média)	Epochs (média)	Tempo (s)
bfloat16	0.98947	0.03251	15.40	2394.3
float32	0.99346	0.02649	16.20	664.8
float64	0.99366	0.02361	13.20	2057.6

A Tabela 5.4 sintetiza os resultados. FP64 tem a maior acurácia e menor perda; FP32 apresenta valores muito próximos; *bfloat16* fica ligeiramente abaixo. Assim, o FP64 converge em menos épocas, mas o FP32 é o mais eficiente em tempo. Desta forma, no PKLot, FP32 oferece melhor compromisso entre custo e desempenho.

5.3 RESULTADOS PARA O CONJUNTO FRUITS360

5.3.1 Acurácia Final em Teste

Tabela 5.5: Test accuracy médio por formato numérico (Fruits360)

Formato	Test accuracy
bfloat16	0.9750
float32	0.9759
float64	0.9768

A acurácia final em teste para Fruits360 está na Tabela 5.5. FP64 obteve cerca de 97,7%; FP32 e *bfloat16* ficaram por volta de 97,5%. Mesmo com maior complexidade e parâmetros (Tabela 4.4), a redução de precisão não compromete substancialmente a acurácia. Esse padrão repete a tendência observada no PKLot, em que as diferenças entre formatos também são pequenas.

5.3.2 Época de Convergência

Tabela 5.6: Época média de convergência (early stopping) por formato numérico (Fruits360)

Formato	Época média de convergência
bfloat16	13.80
float32	15.20
float64	16.25

A Tabela 5.6 apresenta a convergência. *bfloat16* para em cerca de 13,8 épocas; FP32 em torno de 15,2; FP64 próximo de 16,3. No cenário multiclasse, *bfloat16* aciona a parada com menos épocas. Todos ficam muito abaixo do limite de trinta épocas, sem instabilidade relevante.

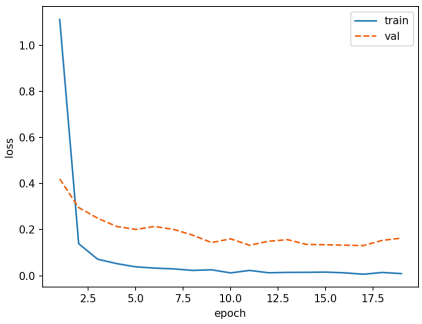
5.3.3 Tempo de Treinamento

A Tabela 5.7 detalha o tempo. FP32 foi o mais rápido: 73,7 s (aproximadamente 1,2 min). FP64 levou 583 s (aproximadamente 9,7 min) e *bfloat16* 363,5 s (aproximadamente 6,1 min). Assim, hardware e biblioteca favorecem FP32; *bfloat16* não acompanha o mesmo nível de otimização.

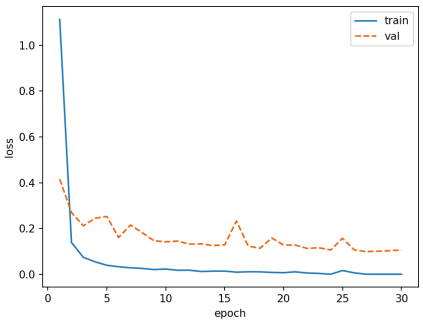
Tabela 5.7: Tempo médio total de treinamento (ms) por formato numérico (Fruits360)

Formato	Tempo total de treinamento (ms)
bfloat16	363532
float32	73716
float64	582986

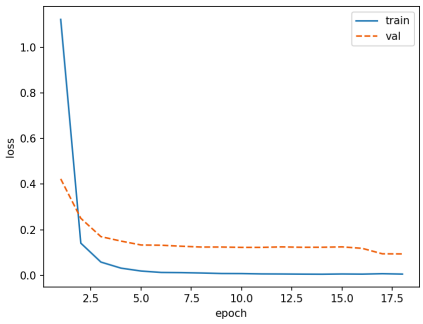
5.3.4 Curvas de Treinamento



(a) Curva de perda para Fruits360 com FP32



(b) Curva de perda para Fruits360 com FP64



(c) Curva de perda para Fruits360 com bfloat16

Figura 5.2: Curvas de perda durante o treinamento no conjunto Fruits360 para diferentes formatos numéricos.

A Figura 5.2 apresenta as curvas de perda para o conjunto Fruits360. Em todos os formatos, a perda de treino decresce de forma consistente e se estabiliza em valores baixos. As curvas de validação de FP32 e *bfloat16* permanecem muito próximas ao longo das épocas, enquanto FP64 apresenta variações um pouco mais acentuadas, ainda que sem indícios de instabilidade.

Por se tratar de um problema multiclasse, com maior número de classes e parâmetros, o Fruits360 tende a apresentar maior sensibilidade ao processo de treinamento. Nesse contexto, pequenas diferenças nas trajetórias de validação podem ser atribuídas a flutuações naturais do processo de aprendizado e às características do otimizador, sem impacto relevante no desempenho final em teste.

5.3.5 Síntese Parcial para Fruits360

A Tabela 5.8 resume os resultados. FP64 tem maior acurácia e menor perda; FP32 mostra valores próximos; e *bfloat16* fica ligeiramente abaixo. Nesse conjunto, *bfloat16* converge

Tabela 5.8: Resumo por formato numérico (Fruits360)

Tipo	Acurácia Teste (média)	Perda Teste (média)	Epochs (média)	Tempo (s)
bfloat16	0.97500	0.12696	13.80	363.5
float32	0.97591	0.13308	15.20	73.7
float64	0.97684	0.12038	16.25	583.0

em menos épocas, mas FP32 é o mais eficiente em tempo. Por esse motivo, FP32 oferece o melhor compromisso custo–desempenho no Fruits360.

5.4 COMPARAÇÃO E ANÁLISE INTEGRADA

Considerando a acurácia final em teste, os resultados mostram que os três formatos numéricos avaliados apresentam desempenho muito semelhante em ambos os conjuntos de dados. No PKLot, todas as configurações superam 98,9% de acurácia média, enquanto no Fruits360 os valores permanecem em torno de 97,5% a 97,7%. As diferenças observadas, inferiores a 0,2 ponto percentual, são marginais e podem estar dentro da variabilidade estocástica do treinamento, dado que os resultados correspondem à média de múltiplas execuções com diferentes sementes.

Sob a ótica da perda média em teste, observa-se comportamento análogo. Embora a precisão dupla (FP64) apresente, em média, valores ligeiramente menores de perda, as diferenças em relação ao FP32 são pequenas e não se refletem em ganhos relevantes de acurácia. Isso sugere que, para os problemas considerados, o treinamento é bem condicionado e tolerante à redução da precisão numérica.

A análise da convergência indica que o número de épocas até a parada antecipada varia conforme o conjunto de dados e o formato numérico. No PKLot, FP64 tende a convergir em menos épocas, enquanto no Fruits360 o *bfloat16* aciona a parada antecipada mais cedo. No entanto, essas diferenças ocorrem dentro de uma faixa estreita, e todos os formatos convergem de forma estável e bem antes do limite máximo de épocas, reforçando a robustez do treinamento frente à precisão numérica.

Em contraste, o custo computacional apresenta variações substanciais entre os formatos. Em ambos os conjuntos, FP32 é consistentemente o mais rápido, enquanto FP64 impõe um custo significativamente maior, chegando a diferenças de até sete vezes no Fruits360. O formato *bfloat16* apresenta desempenho intermediário no Fruits360 e desempenho inferior no PKLot, evidenciando que seus benefícios dependem fortemente do suporte e das otimizações disponíveis no ambiente de execução.

Do ponto de vista do trade-off entre desempenho e custo, os resultados indicam que a transição de FP64 para FP32 oferece ganho computacional substancial sem prejuízo prático da acurácia, caracterizando um ganho absoluto claro. Já a transição de FP32 para *bfloat16* não produz benefícios consistentes neste cenário, com ganhos modestos ou inexistentes que tendem a se confundir com a variabilidade estocástica do treinamento.

Em síntese, a análise integrada mostra que o desempenho do modelo é robusto à escolha da precisão numérica, enquanto o custo computacional é fortemente afetado por essa decisão. Nesse contexto, FP32 se apresenta como a opção mais equilibrada, ao passo que FP64 implica custo elevado sem ganhos proporcionais e o *bfloat16* permanece dependente de condições específicas de otimização para se tornar vantajoso.

6 CONCLUSÃO

Neste trabalho, investigou-se de forma experimental o impacto de diferentes formatos de ponto flutuante no treinamento e na inferência de uma rede neural convolucional aplicada a problemas de classificação de imagens. Os experimentos foram conduzidos sobre os conjuntos PKLot e Fruits360, com o objetivo de analisar como a escolha da precisão numérica afeta o tempo de treinamento, o comportamento de convergência e a acurácia final, mantendo controladas as demais variáveis do processo.

A hipótese central postulava que a redução da precisão numérica poderia diminuir o custo computacional do treinamento, com possíveis efeitos sobre a dinâmica de convergência e a qualidade das classificações. Os resultados indicaram que essa hipótese foi parcialmente confirmada. A substituição da precisão dupla (FP64) pela precisão simples (FP32) resultou em reduções consistentes no tempo de treinamento, sem degradação relevante da acurácia final. Em ambos os conjuntos de dados, os valores de acurácia obtidos com FP32 permaneceram muito próximos aos de FP64, e o comportamento de convergência mostrou-se semelhante.

Para o formato *bfloat16*, os resultados apresentaram maior dependência do contexto experimental. No conjunto PKLot, observou-se uma leve redução de acurácia em relação a FP32 e FP64, ainda que os valores tenham permanecido elevados. No Fruits360, o desempenho foi próximo aos demais formatos, com convergência em menos épocas, porém sem ganhos consistentes em tempo de treinamento. Assim, nas condições avaliadas, o uso de *bfloat16* não apresentou vantagens práticas claras em relação ao FP32.

Um resultado que merece destaque ocorreu no conjunto Fruits360, no qual o tempo de treinamento em FP64 foi significativamente superior ao de FP32, atingindo uma diferença de aproximadamente sete vezes. Esse comportamento foi mais acentuado do que o inicialmente esperado, considerando que a precisão dupla apenas duplica o número de bits em relação à precisão simples. Essa discrepância sugere a influência de fatores adicionais, como padrões de acesso à memória, uso de caches e decisões internas de implementação do framework, que não foram isolados neste estudo e constituem um ponto relevante para investigações futuras.

De forma geral, as diferenças observadas entre os formatos numéricos em termos de acurácia final e convergência mostraram-se pequenas quando comparadas à variabilidade natural do processo de treinamento. Mesmo considerando múltiplas execuções independentes com diferentes sementes, essas variações não se traduziram em ganhos consistentes de desempenho final. Esse comportamento indica que os modelos avaliados são bem condicionados e tolerantes à redução de precisão numérica nas condições experimentais consideradas.

Do ponto de vista prático, os resultados indicam que a precisão FP32 representa uma escolha equilibrada para problemas de classificação de imagens semelhantes aos analisados, oferecendo um bom compromisso entre tempo de treinamento, estabilidade de convergência e acurácia final. Formatos de maior precisão, como FP64, apresentaram custo computacional elevado sem ganhos proporcionais, enquanto formatos de menor precisão mostraram benefícios fortemente dependentes do ambiente de execução e do suporte do framework.

Em síntese, os resultados mostram que a escolha do formato de ponto flutuante deve ser orientada por evidência empírica e pelas características do problema e do ambiente computacional. A análise apresentada contribui para uma compreensão mais concreta do impacto prático da precisão numérica em redes neurais convolucionais, ao mesmo tempo em que fundamenta decisões técnicas mais conscientes no treinamento desses modelos.

6.1 LIMITAÇÕES E TRABALHOS FUTUROS

Uma limitação relevante deste trabalho está relacionada à gestão de memória durante o carregamento dos conjuntos de dados. Embora o Fruits360 tenha permitido o carregamento integral de seus subconjuntos na memória principal, o mesmo não foi possível para o PKLot devido ao volume total de dados e ao comportamento do carregador durante o treinamento. A necessidade de particionar o PKLot em blocos menores evidencia que estimativas baseadas apenas no tamanho bruto das imagens não refletem o consumo real de memória em execução.

Outra limitação refere-se ao conjunto de métricas adotadas para a avaliação dos resultados. A análise concentrou-se principalmente em valores médios obtidos a partir de múltiplas execuções independentes, considerando tempo de treinamento, acurácia final e comportamento das curvas de convergência. No entanto, não foram reportadas métricas estatísticas complementares, como desvio padrão, intervalo de variação ou análises por semente individual.

A ausência dessas métricas limita uma avaliação mais rigorosa da estabilidade numérica e da consistência dos resultados. Testes estatísticos formais, bem como análises por execução individual, poderiam reforçar a interpretação de que as diferenças observadas entre os formatos numéricos não são estatisticamente relevantes, mesmo quando pequenas variações são observadas.

Da mesma forma, embora o trabalho discuta qualitativamente o trade-off entre desempenho e custo computacional, não foi realizada uma análise quantitativa explícita de ganho absoluto ou relativo entre os formatos, como razões diretas de aceleração ou economia de tempo por unidade de acurácia. A formalização dessas métricas poderia tornar a comparação ainda mais objetiva em cenários de decisão prática.

Também não foi realizada uma análise detalhada do balanceamento das classes nos conjuntos utilizados. A avaliação priorizou a acurácia global, que é adequada para uma análise inicial, mas pode não capturar completamente o desempenho do modelo em cenários com distribuições desbalanceadas. Métricas como precisão, revocação e F1-score constituem extensões diretas e relevantes deste estudo.

A quantização não foi abordada por não constituir o foco principal da investigação. Técnicas como INT8 introduzem decisões adicionais de projeto e etapas de calibração que aumentam a complexidade experimental, podendo dificultar a análise isolada do impacto da precisão de ponto flutuante.

Por fim, não foi realizada uma análise da distribuição numérica dos parâmetros aprendidos pela rede, como faixa de valores, dispersão e concentração dos pesos. Esse tipo de análise pode fornecer indícios mais claros sobre a tolerância do modelo a representações de menor precisão, sendo apontado como um caminho natural para trabalhos futuros.

REFERÊNCIAS

Almeida, P., Oliveira, L. S., Jr., E. S., Jr., A. B. e Koerich, A. (2015). Pklot – a robust dataset for parking lot classification. *Expert Systems with Applications*, 42(11):4937–4949.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Chang, P. (2022). Multi-layer perceptron neural network for improving detection performance of malicious phishing urls without affecting other attack types classification.

Chollet, F. et al. (2015). Keras. https://keras.io.

Cireşan, D., Meier, U. e Schmidhuber, J. (2012). Multi-column deep neural networks for image classification.

Dandekar, M., Punn, N. S., Sonbhadra, S. K., Agarwal, S. e Kiran, R. U. (2021). Fruit classification using deep feature maps in the presence of deceptive similar classes. Em *2021 International Joint Conference on Neural Networks (IJCNN)*, página 1–6. IEEE.

Gernigon, C., Filip, S.-I., Sentieys, O., Coggiola, C. e Bruno, M. (2023). Low-precision floating-point for efficient on-board deep neural network processing.

Goodfellow, I., Bengio, Y. e Courville, A. (2016). *Deep Learning*. MIT Press.

He, K., Zhang, X., Ren, S. e Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.

He, K., Zhang, X., Ren, S. e Sun, J. (2016). Deep residual learning for image recognition. Em *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, páginas 770–778.

Higham, N. J. (2002). *Accuracy and Stability of Numerical Algorithms*. SIAM, Philadelphia, PA, 2nd edition.

Hochuli, A. G., Jr., A. S. B., de Almeida, P. R. L., Alves, W. B. S. e Cagni, F. M. C. (2022). Evaluation of different annotation strategies for deployment of parking spaces classification systems.

Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de Las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Rae, J. W., Vinyals, O. e Sifre, L. (2022). Training compute-optimal large language models.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V. e Adam, H. (2019). Searching for mobilenetv3. Em *Proceedings of the IEEE International Conference on Computer Vision*.

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

Iskandar, J. (2000). *Normas da ABNT Comentadas para Trabalhos Científicos*. Editora Champagnat (PUCPR).

Jia, X., Song, S., He, W., Wang, Y., Rong, H., Zhou, F., Xie, L., Guo, Z., Yang, Y., Yu, L., Chen, T., Hu, G., Shi, S. e Chu, X. (2018). Highly scalable deep learning training system with mixed-precision: Training imagenet in four minutes.

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., Yang, J., Park, J., Heinecke, A., Georganas, E., Srinivasan, S., Kundu, A., Smelyanskiy, M., Kaul, B. e Dubey, P. (2019). A study of bfloat16 for deep learning training.

Kingma, D. P. e Ba, J. (2017). Adam: A method for stochastic optimization.

Krizhevsky, A., Sutskever, I. e Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Em *Advances in Neural Information Processing Systems*, volume 25.

LeCun, Y., Bengio, Y. e Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436–444.

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

LeNail, A. (2019). Nn-svg: Publication-ready neural network architecture schematics. *Journal of Open Source Software*, 4(33):747.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G. e Wu, H. (2018a). Mixed precision training.

Micikevicius, P., Narang, S., Alben, J. et al. (2018b). Mixed precision training. *International Conference on Learning Representations*.

Ni, G., He, X. e Wu, Y. (2019). Mixed-precision neural networks: A survey. *arXiv preprint arXiv:1909.03082*.

Nie, Y., Li, L., Gan, Z., Wang, S., Zhu, C., Zeng, M., Liu, Z., Bansal, M. e Wang, L. (2021). Mlp architectures for vision-and-language modeling: An empirical study.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

NVIDIA Corporation (2025). CUDA C++ Programming Guide. urlhttps://docs.nvidia.com/cuda/cuda-c-programming-guide. Version 13.0.

Oltean, M. (2017). Fruits-360 dataset. https://www.kaggle.com/datasets/moltean/fruits. Branch used: 100x100.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. e Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library.

Prechelt, L. (1998). Early stopping—but when? *Neural Networks: Tricks of the Trade*, páginas 55–69.

Rumelhart, D. E., Hinton, G. E. e Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A. e Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. Em *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, páginas 4510–4520.

Schaefer, C. J., Guo, E., Stanton, C., Zhang, X., Jablin, T., Lambert-Shirzad, N., Li, J., Chou, C., Joshi, S. e Wang, Y. E. (2023). Mixed precision post training quantization of neural networks with sensitivity guided search.

Tolstikhin, I., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., Lucic, M. e Dosovitskiy, A. (2021). Mlp-mixer: An all-mlp architecture for vision.

Wang, S. e Kanwar, P. (2019). BFloat16: The secret to high performance on Cloud TPUs. https://cloud.google.com/blog/products/ai-machine-learning/bfloat16-the-secret-to-high-performance-on-cloud-tpus. Google Cloud Blog.

Wang, S., Xiong, C., Yu, J., Lin, Z. e Ling, S. (2018). Training vs inference: Memory consumption by neural networks. *NeurIPS Workshop on Efficient Methods for Deep Reinforcement Learning*.

Wang, X., Xu, Y. e Yang, Y. (2019). Energy measurement and optimization for deep learning on gpus. Em *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*.

Yun, J., Choi, S., Rameau, F., Kang, B. e Fu, Z. (2025). Revisiting 16-bit neural network training: A practical approach for resource-limited learning.

APÊNDICE A – CÁLCULO DAS DIMENSÕES E PARÂMETROS DA REDE CONVOLUCIONAL

Este apêndice apresenta o detalhamento completo da propagação de dimensões e do cálculo do número de parâmetros da arquitetura convolucional utilizada nos experimentos. O objetivo é permitir a verificação analítica dos valores reportados no corpo do trabalho, evitando sobrecarga descritiva nos capítulos principais.

As dimensões espaciais foram obtidas a partir das expressões clássicas de convolução e *pooling*, enquanto o número de parâmetros foi calculado diretamente a partir das dimensões de entrada e saída de cada camada.

A.1 PROPAGAÇÃO DE DIMENSÕES E PARÂMETROS

A seguir é apresentado o registro estruturado da arquitetura, incluindo dimensões de entrada e saída, hiperparâmetros e contagem de parâmetros por camada. As operações seguem a ordem exata de execução da rede.

```
Conv:
    Entrada:      3 × 64 × 64
    Kernel:       3 × 3
    Stride:       1
    Padding:      1
    Operation:    floor((64 + 2*1 - 3) / 1) + 1 = 64
    Canais out:   8
    Saída:        8 × 64 × 64
    Params:       8 × 3 × 3 × 3 = 216

ReLU:
    Entrada:      8 × 64 × 64
    Saída:        8 × 64 × 64

MaxPool:
    Entrada:      8 × 64 × 64
    Kernel:       2 × 2
    Stride:       2
    Padding:      0
    Operation:    floor((64 + 2*0 - 2) / 1) + 1 = 32
    Saída:        8 × 32 × 32

Conv:
    Entrada:      8 × 32 × 32
    Kernel:       3 × 3
    Stride:       1
    Padding:      1
    Operation:    floor((32 + 2*1 - 3) / 1) + 1 = 32
    Canais out:   16
```

```
    Saída:        16 × 32 × 32
    Params:       16 × 8 × 3 × 3 = 1152

ReLU:
    Entrada:      16 × 32 × 32
    Saída:        16 × 32 × 32

MaxPool:
    Entrada:      16 × 32 × 32
    Kernel:       2 × 2
    Stride:       2
    Padding:      0
    Operation:    floor((32 + 2*0 - 2) / 1) + 1 = 16
    Saída:        16 × 16 × 16

Conv:
    Entrada:      16 × 16 × 16
    Kernel:       3 × 3
    Stride:       1
    Padding:      1
    Operation:    floor((16 + 2*1 - 3) / 1) + 1 = 16
    Canais out:   24
    Saída:        24 × 16 × 16
    Params:       24 × 16 × 3 × 3 = 3456

ReLU:
    Entrada:      24 × 16 × 16
    Saída:        24 × 16 × 16

MaxPool:
    Entrada:      24 × 16 × 16
    Kernel:       2 × 2
    Stride:       2
    Padding:      0
    Operation:    floor((16 + 2*0 - 2) / 1) + 1 = 8
    Saída:        24 × 8 × 8

Flatten:
    24 × 8 × 8 = 1536

Linear:
    Entrada:     1536
    Saída:       256
    Params:      1536 × 256 = 393216

Linear:
    Entrada:     256
    Saída:
```

```
        Fruits360:  225
        PKLot:      2
    Params:
        Fruits360:  256 × 225 = 57600
        PKLot:      256 × 2   = 512

TotalParams:
    Conv:         216 + 1152 + 3456 = 4824
    Base:         4824 + 393216 = 398040
    Fruits360:    398040 + 57600 = 455640
    PKLot:        398040 + 512 = 398552
```

A.2 OBSERVAÇÕES

Nota-se que a diferença no número total de parâmetros entre os experimentos decorre exclusivamente da dimensão da camada de saída, que varia de acordo com o número de classes de cada conjunto de dados. As camadas convolucionais e a primeira camada totalmente conectada permanecem idênticas em todos os cenários avaliados.