Compressão Consciente de Modelos de Deep Learning Aplicada a Ambientes Kubernetes

Mateus Arnaud S. S. Goldbarg*†, Vitor Y. F. Freitas*†,
Lucileide M. D. Silva*†‡, Sérgio N. Silva*†§ e Marcelo A. C. Fernandes*†‡¶
*InovAI Lab, nPITI/IMD, UFRN, Natal, RN, Brasil
†Leading Advanced Technologies Center of Excellence (LANCE), nPITI/IMD, UFRN, Natal 59078-970, RN, Brasil
‡Federal Institute of Education, Science and Technology of Rio Grande do Norte, Paraiso, Santa Cruz 59200-000, RN, Brasil
§Departamento de Engenharia Elétrica (DEE), UFCG, Campina Grande, PB, Basil
¶Departamento de Engenharia da Computação e Automação (DCA), UFRN, Natal, RN, Basil
Email: mateus.goldbarg@dca.ufrn.br, vitor.freitas.123@ufrn.edu.br,
lucileide.dantas@ifrn.edu.br, sergionatan@dee.ufcg.edu.br, mfernandes@dca.ufrn.br

***Abstract*—Modelos de *deep learning*, como redes neurais convolucionais (CNNs), têm se destacado em diversas tarefas de aprendizado supervisionado, mas frequentemente exigem elevada capacidade computacional para execução eficiente. No entanto, sua elevada complexidade computacional pode dificultar a implantação eficiente em ambientes distribuídos baseados em microsserviços, como clusters Kubernetes, especialmente quando há restrições de recursos. Diante desse cenário, foi conduzido um estudo experimental utilizando uma abordagem de compressão consciente de modelos de redes neurais profundas, baseada na aplicação iterativa de poda seguida de quantização ($P \rightarrow Q$), com o objetivo de reduzir o tamanho dos modelos, o consumo de memória e CPU, e o tempo de resposta durante a inferência. Para validar a proposta, foram utilizados modelos VGG-16 treinados com o conjunto de dados CIFAR-10, submetidos a diferentes configurações de compressão controladas por dois parâmetros: α, que define o grau de agressividade da poda, e b, que determina a quantidade de bits utilizados na quantização dos pesos. Os modelos comprimidos foram implantados como microsserviços em um cluster Kubernetes utilizando MicroK8s, e testados com geradores de carga via JMeter, sendo monitorados por Prometheus. Os resultados indicam que a técnica $P \rightarrow Q$ permite uma compressão de até 5,6 vezes no tamanho do modelo, mantendo níveis satisfatórios de acurácia, com perdas inferiores a 7%. Além disso, observou-se uma redução de até 1,8 vezes na latência de resposta, 1,71 vezes no uso de memória e 1,46 vezes no consumo de CPU. Esses achados demonstram o potencial e a viabilidade da abordagem proposta para a otimização de aplicações baseadas em DNNs em ambientes de microsserviços escaláveis.**

***Index Terms*—Compressão de modelos, deep learning, quantização, poda de redes neurais, Kubernetes, microsserviços, escalabilidade**

I. Introdução

Redes neurais profundas (DNNs) têm se mostrado eficazes na tarefa de classificação de imagens, sendo amplamente aplicadas em áreas como reconhecimento de objetos e diagnóstico médico. Entre as arquiteturas mais utilizadas está a Rede Neural Convolucional (CNN), capaz de extrair e processar características hierárquicas das imagens por meio de camadas convolucionais e de *pooling*. Embora apresentem alto desempenho, essas redes demandam um grande volume de operações numéricas, resultando em elevado consumo de energia e memória, o que pode se tornar um desafio em aplicações em tempo real [1].

Diante desse problema, acelerar a computação de Redes Neurais Profundas (*Deep Neural Networks* — DNNs) tem se tornado um foco importante no campo do aprendizado de máquina, com uma literatura em rápida expansão. As estratégias de aceleração da inferência podem ser agrupadas em duas categorias principais: abordagens baseadas em algoritmos e abordagens baseadas em sistemas. As técnicas baseadas em algoritmos incluem: processamento paralelo (paralelismo de dados e de modelo), compressão de modelos (remoção de pesos e quantização de valores de pesos e ativações), exploração de esparsidade (em nível de valor ou de bit) e redução do modelo por meio de aproximações [2], [3], [4], [5], [6], [7], [8], [9], [10].

A compressão convencional por poda é abordada por [2], [11] e [3], nos quais a remoção dos pesos é realizada apenas uma vez por época. Já a quantização convencional, discutida em [4] e [5], aplica diferentes quantizações por camada, também limitadas a uma única aplicação por época. Em contraste, este trabalho adota um novo esquema de compressão consciente de modelos, que minimiza simultaneamente as perdas associadas à poda e à quantização durante o treinamento, aplicando a compressão por poda seguida de quantização ($P \rightarrow Q$) a cada iteração. A compressão iterativa por $P \rightarrow Q$ é um algoritmo que combina ambas as técnicas — poda e quantização — de forma coordenada e contínua ao longo do treinamento do modelo. Essa abordagem foi inicialmente proposta para a tarefa de classificação de dados genômicos em [10], e posteriormente validada na classificação de modulações digitais em [12]. O principal desafio reside em reduzir o tamanho do modelo com perda mínima de acurácia. Enquanto a poda elimina parâmetros e reduz o número de operações matemáticas, a quantização diminui a precisão da representação binária dos pesos, reduzindo assim a complexidade do modelo. Essas técnicas de compressão para DNNs oferecem uma solução promissora para a otimização do

uso de recursos computacionais, especialmente em ambientes distribuídos baseados em microsserviços, como aqueles gerenciados por Kubernetes (K8s).

Microsserviços representam um estilo arquitetural que tem ganhado popularidade nos últimos anos, devido à sua abordagem ágil e escalável no desenvolvimento de aplicações. Nessa arquitetura, a aplicação é dividida em pequenos serviços independentes, cada um responsável por uma função específica e bem definida [13]. Esses serviços podem ser desenvolvidos, implantados e escalados de forma independente, o que facilita a manutenção e evolução da aplicação como um todo. Para viabilizar a orquestração, escalabilidade e gerenciamento desses microsserviços em ambientes distribuídos, plataformas como o Kubernetes têm se consolidado como soluções robustas [14]. O Kubernetes automatiza tarefas como balanceamento de carga, monitoramento, escalonamento horizontal e reinicialização de serviços em caso de falhas, garantindo maior resiliência e disponibilidade do sistema. No contexto da integração com inteligência artificial, um ou mais desses microsserviços podem ser responsáveis por realizar inferência utilizando modelos baseados em DNNs, beneficiando-se diretamente dos mecanismos de escalabilidade e isolamento fornecidos pelo Kubernetes [15]–[18].

Nesses ambientes, o uso eficiente de recursos computacionais é uma preocupação central. Cada serviço é executado em seu próprio contêiner, e a otimização do uso de recursos como CPU e memória é fundamental para garantir o bom desempenho da aplicação. Modelos de DNN não comprimidos, especialmente os mais complexos, podem demandar grande quantidade de memória, o que se torna problemático em sistemas que exigem baixa latência e alta disponibilidade. Nesse contexto, estratégias de compressão de modelos de aprendizado de máquina podem impactar positivamente tanto o desempenho quanto o consumo de recursos dos microsserviços. A aplicação de técnicas de compressão permite remover redundâncias, representar dados de forma mais compacta e empregar métodos de otimização que reduzem significativamente o tamanho do modelo. A redução no consumo de memória viabiliza a execução paralela de múltiplos serviços, economiza espaço em disco e diminui a sobrecarga no gerenciamento de recursos. Além disso, a compressão de modelos pode agilizar os processos de implantação e escalabilidade dos microsserviços. Modelos menores são transferidos e carregados com maior rapidez em diferentes ambientes, acelerando o tempo de inicialização e resposta. A escalabilidade horizontal, em particular, é uma estratégia essencial em arquiteturas modernas, pois permite lidar com aumentos de demanda por meio da replicação de instâncias distribuídas, ao invés da expansão vertical de recursos em um único servidor.

Assim, este trabalho tem como objetivo propor, implementar e avaliar uma abordagem de compressão consciente de modelos de *Deep Learning*, baseada na aplicação iterativa de poda seguida de quantização ($P \rightarrow Q$), em ambientes de microsserviços orquestrados com Kubernetes. A proposta visa reduzir o consumo de recursos computacionais, como memória, uso de CPU e tempo de resposta, sem comprometer significativamente a acurácia dos modelos. Para isso, diferentes níveis de compressão foram aplicados ao modelo VGG-16 treinado com o conjunto de dados CIFAR-10, sendo os impactos avaliados por meio de experimentos de inferência sob carga com métricas extraídas via Prometheus e JMeter. Os resultados obtidos demonstram os benefícios da compressão $P \rightarrow Q$ para a otimização de aplicações baseadas em DNNs em sistemas distribuídos escaláveis.

II. Compressão de Modelos por Poda seguido de Quantização ($P \rightarrow Q$) Iterativo

Conforme ilustrado na Figura 1, o processo inicia-se com a aplicação da poda, que identifica e remove os pesos menos relevantes da DNN, reduzindo a complexidade e o tamanho do modelo. Em seguida, realiza-se a quantização sobre os pesos remanescentes, diminuindo a precisão dos valores numéricos para representá-los com um número reduzido de bits, contribuindo para a compactação e eficiência computacional do modelo resultante.

A otimização utilizando a técnica de pod consiste na remoção dos pesos insignificantes do modelo com base em um determinado limiar. Diversas estratégias podem ser empregadas para definir esse limiar, como apresentado em [19], onde ele é determinado a partir do nível de esparsidade desejado para cada camada. Neste trabalho, durante o processo de poda, os pesos de cada k-ésima camada, $\mathbf{W}_k(n)$, com valores pequenos (abaixo de um limiar) são zerados, ou seja,

$$P\left(\mathbf{W}_k(n), \beta_k\right) = \begin{cases} \mathbf{W}_k(n) & \text{se } |\mathbf{W}_k(n)| \geq \beta_k \\ 0 & \text{se } |\mathbf{W}_k(n)| < \beta_k \end{cases} \quad (1)$$

onde $P(\cdot,\cdot)$ representa a função de poda e β_k é o limite de corte aplicado à k-ésima camada. Neste trabalho, considerou-se que os pesos de cada camada seguem uma distribuição Gaussiana, conforme apresentado em [20]. Dessa forma, o valor de β_k é definido como

$$\beta_k = \alpha \times \sigma_k \quad (2)$$

onde α define o grau de agressividade da poda e σ_k o desvio padrão dos pesos da k-ésima camada $\mathbf{W}_k(n)$. Após o processo de poda, os pesos podados de cada k-ésima camada, são quantizados.

A quantização produz inteiros de baixa precisão (largura de bits reduzida) para representar os pesos. Ou seja, os pesos de cada k-ésima camada, $\mathbf{W}_k(n)$, passam a ser representado por um dos M níveis discretos possíveis que podem ser codificados com b bits. Diversas técnicas de quantização são abordadas na literatura, como apresentado em [4], [5], [9], [8]. Um esquema amplamente utilizado é a quantização uniforme, com $M = 2^b - 1$. Nesse caso, a saída da quantização de cada k-ésima camada é expressa por

$$\mathbf{C}_k(n) = Q\left(P\left(\mathbf{W}_k(n), \beta\right), q_k\right) = \left\lfloor \frac{\mathbf{W}_k(n)}{q_k} \right\rfloor \times q_k, \quad (3)$$

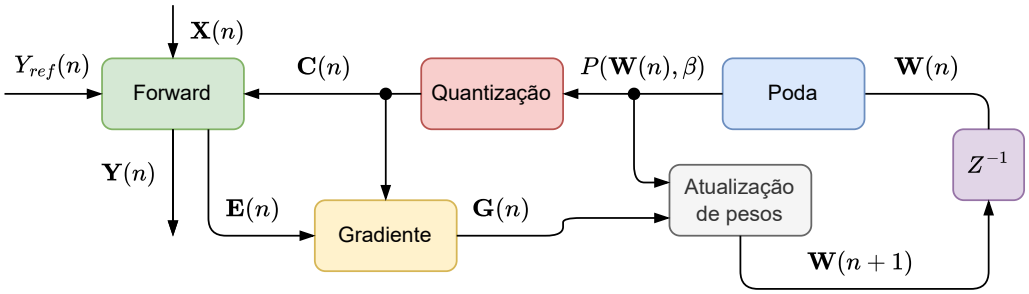


Fig. 1: Esquema de treinamento com compressão consciente por poda seguido de quantização ($P \rightarrow Q$) em cada iteração.

onde $Q(\cdot, \cdot)$ representa a função de quantização, $\lfloor \cdot \rfloor$ é a operação de piso (arredondamento para baixo), e q_k, o fator de quantização, é definido por

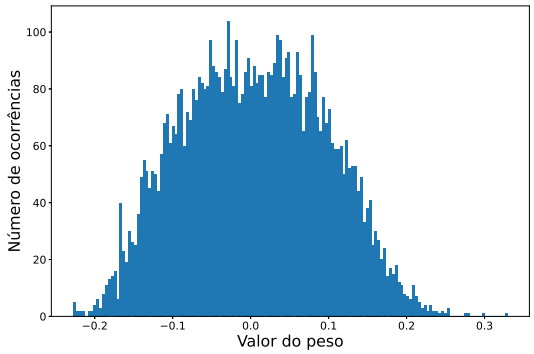
$$q_k = \frac{\max\{|\mathbf{W}_k(n)|\}}{2^{b-1} - 1}. \tag{4}$$

Essa abordagem permite a compressão otimizada de DNNs, resultando em modelos mais compactos e computacionalmente eficientes, com perdas mínimas de precisão. Figuras 2a e 2b mostram um exemplo dos pesos da k-ésima camada, $\mathbf{W}_k(n)$, de uma CNN antes e depois da aplicação da compressão consciente, respectivamente. O valor do limiar de corte utilizado foi $\beta_k = 0{,}5 \times \sigma_k$ (com $\alpha_k = 0{,}5$) e $b = 5$ bits.

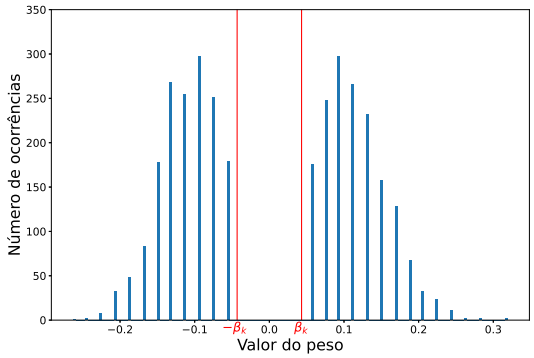
Após o processo de compressão, mesmo que os pesos sejam quantizados para uma menor quantidade de bits, eles ainda permanecem representados internamente no formato padrão de 32 bits. De forma semelhante, os pesos anulados por meio do poda continuam presentes no modelo com valor igual a zero.

Embora técnicas de quantização não linear também promovam a redução do modelo ao aproximarem pesos de zero e formarem regiões de insensibilidade, as chamadas zonas mortas [21], [22], a estratégia de P $\rightarrow$ Q iterativa, apresenta vantagens operacionais relevantes. A separação entre as etapas permite um controle mais refinado sobre a compressão, viabilizando o ajuste independente da taxa de esparsidade e da resolução numérica dos pesos. Isso facilita a adaptação do modelo a diferentes restrições de hardware e requisitos de desempenho. Além disso, a aplicação da poda prévia contribui para a regularização da distribuição dos pesos remanescentes, o que, por sua vez, favorece a quantização subsequente.

Na literatura, existem duas estratégias principais para tratar essa representação comprimida. A primeira consiste na utilização de representações esparsas, como o formato *Compressed Sparse Column* (CSC) [23], que armazena apenas os elementos diferentes de zero e seus respectivos índices, permitindo economia de memória e aceleração de operações quando suportado por hardware especializado [23]. A segunda estratégia, adotada neste trabalho, mantém a estrutura original do modelo (inclusive os pesos zerados) e aplica compressão no arquivo resultante por meio do algoritmo Deflate, utilizado no formato ZIP. O *Deflate* combina a codificação de Huffman com o algoritmo LZ77, identificando padrões e redundâncias nos dados para realizar compressão sem perdas [24]. Essa abordagem se mostra eficaz especialmente em modelos com alta esparsidade, pois a presença de muitos pesos zerados introduz padrões repetitivos e sequências recorrentes nos dados.



(a) Pesos antes da compressão.



(b) Pesos depois da poda seguida de quantização com $\alpha = 0{,}5$ e $b = 5$ bits

Fig. 2: Histograma dos valores de pesos associados a uma camada de uma rede CNN antes e depois da compressão consciente por poda seguido de quantização iterativa.

Para a quantização, os parâmetros do modelo foram transformados para uma representação de menor precisão, re-

duzindo o número de bits utilizados na codificação dos pesos. Essa transformação foi realizada com base em um fator de quantização q_k, específico para cada camada, conforme definido na Equação 4. A quantização foi implementada no formato uniforme, convertendo os valores reais dos pesos para inteiros com largura de b bits, resultando em uma codificação com 2^b níveis discretos. Essa representação permite reduzir significativamente o tamanho do modelo, mantendo a distribuição relativa dos pesos e preservando, em grande parte, a acurácia da rede neural.

III. METODOLOGIA

Para validar a abordagem proposta, foi implementado um ambiente de execução composto por um cluster Kubernetes (k8s) baseado no MicroK8s (versão 1.26), com microsserviços (ou pods) executando modelos de DNN comprimidos. Cada modelo é caracterizado pelos parâmetros (α, b), onde α define o grau de agressividade da poda (*pruning*) e b representa o número de bits utilizados na quantização dos pesos (ver Equações 2 e 4). Foram realizados experimentos com $b = 32$ e $b = 8$, variando-se α nos valores 0, 0,5 e 1,25. O cenário com $b = 32$ e $\alpha = 0$ representa o modelo sem nenhum tipo de compressão, enquanto o caso com $b = 8$ e $\alpha = 1{,}25$ corresponde ao modelo com o maior nível de compressão avaliado. A Figura 3 detalha ambiente de execução no qual foi criada uma máquina virtual com o sistema operacional Ubuntu 20.04, para rodar o Cluster K8s, equipada com 2 vCPUs e 4 GB de memória RAM, hospedada em um servidor com Ubuntu 22.04 e 64 GB de RAM e processador com 12 núcleos.

O microsserviço (ou pod) executa a inferência de uma DNN do tipo CNN baseada no modelo VGG-16 [25], cuja arquitetura está descrita na Tabela I. Nessa tabela, K_k representa o número de *kernels* da k-ésima camada convolucional, S_k corresponde ao tamanho desses *kernels*, e F_k indica o número de neurônios nas camadas densas da k-ésima camada.

as

Os modelos DNN associados aos microsserviços (pods) foram treinados utilizando o conjunto de dados CIFAR-10 [26], que contém 60.000 imagens coloridas com 3 canais (RGB) e resolução de 32×32 pixels, distribuídas em 10 classes, com 6.000 imagens por classe. O conjunto foi dividido em 50.000 imagens para treinamento e 10.000 para teste (inferência) [26]. Para o treinamento, os dados foram organizados em lotes de tamanho 64 e o otimizador utilizado foi o SGD. A taxa de aprendizado foi definida em $0{,}05$, com *momentum* de $0{,}9$. A função de perda adotada foi a entropia cruzada categórica (*categorical crossentropy*). O treinamento foi conduzido por 50 épocas. Para garantir a reprodutibilidade dos experimentos e melhor comparação entre os resultados, foi utilizada a mesma semente aleatória durante o processo de treinamento dos modelos.

Cada microsserviço (ou pod) foi desenvolvido utilizando o framework Flask (versão 2.3), com a linguagem Python 3.9, expondo uma interface HTTP para o recebimento das imagens e execução das inferências, conforme ilustrado na Figura 3. As imagens são transmitidas por meio de um gerador de carga baseado na ferramenta JMeter [27], sendo enviadas ao pod, que retorna o rótulo correspondente a cada imagem processada. O pod responsável pela DNN é monitorado por um outro pod interno ao cluster MicroK8s, no qual é executado o sistema de monitoramento Prometheus [28]. São monitorados três indicadores-chave de desempenho (KPIs — *Key Performance Indicators*), definidos neste trabalho como Δr, u_{CPU} e u_{Mem}. A variável Δr representa o tempo de resposta (em milissegundos), correspondente ao intervalo entre o envio da imagem e o recebimento do rótulo. A variável u_{CPU} mede o consumo de CPU, expresso em milicores (mCores), utilizado pelo microsserviço (ou pod). Por fim, a variável u_{Mem} indica o consumo de memória (em MiB) do pod responsável pela inferência (ver Figura 3).

TABLE I: Arquitetura VGG-16 adaptada para o dataset CIFAR-10.

k	Descrição	Dimensão	Ativação
1	Entrada	$32 \times 32 \times 3$	-
2	Conv2D($K_2 \times S_2$)	$K_2 = 64$, $S_2 = 3 \times 3$	ReLU
3	Conv2D($K_3 \times S_3$)	$K_3 = 64$, $S_3 = 3 \times 3$	ReLU
4	MaxPool2D(S_4)	$S_4 = 2 \times 2$	-
5	Conv2D($K_5 \times S_5$)	$K_5 = 128$, $S_5 = 3 \times 3$	ReLU
6	Conv2D($K_6 \times S_6$)	$K_6 = 128$, $S_6 = 3 \times 3$	ReLU
7	MaxPool2D(S_7)	$S_7 = 2 \times 2$	-
8	Conv2D($K_8 \times S_8$)	$K_8 = 256$, $S_8 = 3 \times 3$	ReLU
9	Conv2D($K_9 \times S_9$)	$K_9 = 256$, $S_9 = 3 \times 3$	ReLU
10	Conv2D($K_{10} \times S_{10}$)	$K_{10} = 256$, $S_{10} = 3 \times 3$	ReLU
11	MaxPool2D(S_{11})	$S_{11} = 2 \times 2$	-
12	Conv2D($K_{12} \times S_{12}$)	$K_{12} = 512$, $S_{12} = 3 \times 3$	ReLU
13	Conv2D($K_{13} \times S_{13}$)	$K_{13} = 512$, $S_{13} = 3 \times 3$	ReLU
14	Conv2D($K_{14} \times S_{14}$)	$K_{14} = 512$, $S_{14} = 3 \times 3$	ReLU
15	MaxPool2D(S_{15})	$S_{15} = 2 \times 2$	-
16	Conv2D($K_{16} \times S_{16}$)	$K_{16} = 512$, $S_{16} = 3 \times 3$	ReLU
17	Conv2D($K_{17} \times S_{17}$)	$K_{17} = 512$, $S_{17} = 3 \times 3$	ReLU
18	Conv2D($K_{18} \times S_{18}$)	$K_{18} = 512$, $S_{18} = 3 \times 3$	ReLU
19	MaxPool2D(S_{19})	$S_{19} = 2 \times 2$	-
20	Flatten	-	-
21	Dense(F_{21})	$F_{21} = 4096$	ReLU
22	Dense(F_{22})	$F_{22} = 4096$	ReLU
23	Dense(F_{23})	$F_{23} = 10$	Softmax

Antes da implantação dos modelos nos pods, foi necessário gerar os arquivos dos modelos comprimidos com base na estratégia de compressão por poda seguido de quantização ($P \rightarrow Q$) iterativo. Os modelos foram treinados utilizando o framework TensorFlow (versão 2.15.0), e em seguida foram gerados os arquivos finais compactados. Para a compressão via poda, adotou-se a estratégia de manter os pesos zerados no modelo e aplicar compressão do tipo *Deflate* para reduzir o tamanho final dos arquivos [24]. Já a quantização para 8 bits foi realizada convertendo os valores dos pesos para o formato inteiro com sinal de 8 bits, com valores variando entre -128 e 127, de acordo as Equações 3 e 4.

Foram definidos dois perfis de estresse para cada esquema de compressão (α, b). Esses perfis foram construídos utilizando o componente *Open Model Thread Group*, que especifica a quantidade de usuários virtuais gerados ao longo do tempo pelo JMeter, com o objetivo de simular a carga de tráfego na aplicação. O comportamento desses perfis está ilustrado na Figura 4, onde $T_p(t)$ representa o número de

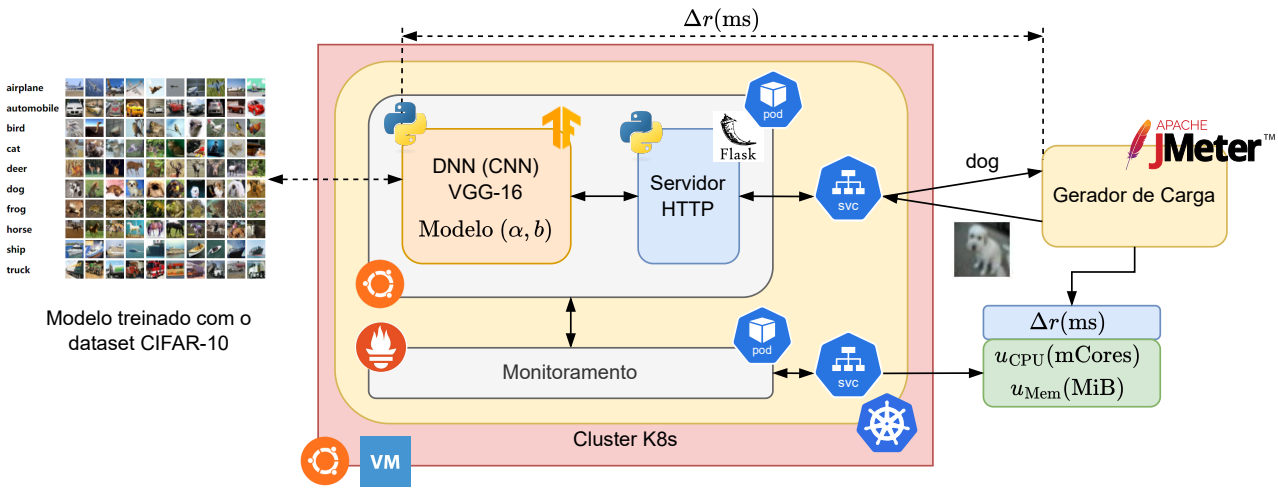


Fig. 3: Infraestrutura de execução dos microsserviços em cluster Kubernetes (k8s) com compressão consciente de modelos DNN.

usuários simultâneos ao longo do tempo, e N_p corresponde ao número máximo de usuários no p-ésimo perfil de estresse. Os valores de N_p foram fixados em 75 e 250 para os dois perfis utilizados.

IV. Resultados

A Tabela II apresenta os níveis de esparsidade, ou seja, a proporção de pesos irrelevantes nos modelos comprimidos para valores de $\alpha > 0$. Observa-se que o aumento do valor de α levou a uma maior esparsidade nos modelos, demonstrando a efetividade da técnica de poda, tanto isoladamente quanto combinada com quantização.

TABLE II: Esparsidades obtidas na classificação de imagens para diferentes tamanhos de bits e valores de $\alpha > 0$.

b \ α	0,50	1,25
32 bits	37,92%	74,76%
8 bits	36,94%	74,49%

A Tabela III apresenta os tamanhos finais dos arquivos após a aplicação das técnicas de compressão ($P \rightarrow Q$) Iterativo. Observa-se uma tendência clara de redução no tamanho dos modelos com o aumento de α e com a quantização para 8 bits, sendo possível uma redução de até 5,6 vezes no tamanho do modelo após a compressão para ($\alpha = 1{,}25$ e $b = 8$). Esses resultados corroboram diretamente os níveis de esparsidade observados, conforme a Tabela II. Tal correlação reforça a eficácia das estratégias aplicadas para tornar os modelos mais compactos, mantendo níveis adequados de desempenho.

A Tabela IV apresenta os valores de acurácia obtidos para os modelos comprimidos. O resultados revelam que os modelos quantizados mantêm comportamentos de acurácia bastante similares entre si, demonstrando a efetividade da compressão na representação dos parâmetros sem perdas significativas de desempenho. Ao comparar com os valores de acurácia na

TABLE III: Tamanho dos modelos comprimidos para diferentes valores de α e b bits, em megabytes (MB).

b \ α	0,00	0,50	1,25
32 bits	123,51 MB	87,11 MB	55,80 MB
8 bits	32,50 MB	32,12 MB	22,09 MB

Tabela IV, verifica-se que é possível eliminar até cerca de 75% dos parâmetros do modelo mantendo uma acurácia próxima ao modelo original, como evidenciado para $\alpha = 1{,}25$.

TABLE IV: Acurácias obtidas na classificação de imagens para diferentes tamanhos de b e α.

b \ α	0,00	0,50	1,25
32 bits	81,58%	79,39%	79,53%
8 bits	79,76%	80,77%	79,31%

A Figura 5 apresenta as matrizes de confusão do modelo original, sem compressão ($\alpha = 0{,}00$ e $b = 32$ bits), do modelo com poda seguida de quantização com $\alpha = 1{,}25$ e $b = 8$ bits.

Com base no ambiente de execução detalhando na Figura 3, foram executados os perfis de estress e foram coletadas os KPIs definidos na metodologia que são Δr, u_{CPU} e u_{Mem}. A Tabela V apresenta os dados referentes ao primeiro perfil de estresse, com $N_1 = 75$. O tempo de resposta, Δr, corresponde ao percentil 95% dos dados capturados, a fim de eliminar valores atípicos. As informações de u_{CPU} e u_{Mem} meio de requisições ao Prometheus, enquanto os dados de tempo de resposta Δr foram obtidos pelo próprio JMeter, também responsável por gerar a carga de estresse no sistema.

Na Tabela V, nota-se uma redução significativa no tempo de resposta, Δr (ms), dos microsserviços ao utilizar modelos quantizados, com desempenho aproximadamente $1,75$ vezes superior em relação aos modelos sem compressão por quantização. Por outro lado, a aplicação da técnica de poda

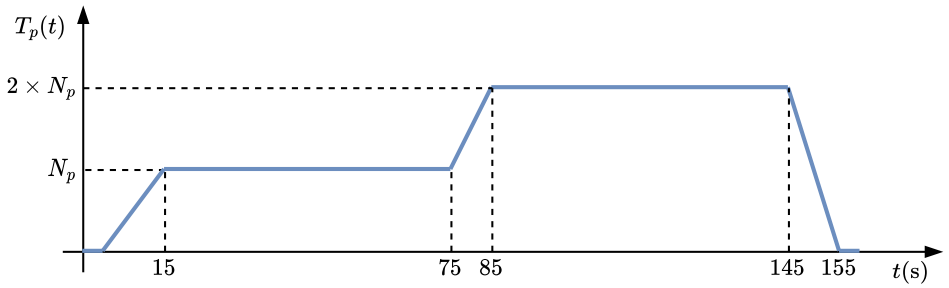


Fig. 4: Perfis de estresse definidos para avaliação de desempenho dos microsserviços, representando a evolução do número de usuários simultâneos ao longo do tempo.

TABLE V: Tempo de resposta, Δr (ms), consumo de memória, u_{Mem} (MiB), e consumo de CPU, u_{CPU} (mCore), de cada microsserviço para o primeiro perfil de estresse ($N_1 = 75$).

α	b	Δr (ms)	u_{Mem} (MiB)	u_{CPU} (mCore)
0,00	32	16,23 ± 2,988	701	135
0,50	32	15,97 ± 3,467	707	136
1,25	32	15,44 ± 2,597	703	132
0,00	8	9,39 ± 2,026	412	95
0,50	8	9,27 ± 3,011	414	94
1,25	8	9,44 ± 1,729	411	94

isoladamente não influenciou esses resultados. A Tabela V também apresenta uma redução no consumo de memória, u_{Mem}, e CPU, u_{CPU}, de, respectivamente, $\approx 1{,}70$ e $\approx 1{,}43$ vezes para os modelos quantizados, em comparação aos modelos não quantizados. Não foram observadas variações relevantes nos consumos quando aplicada apenas a compressão via poda.

A Tabela VI apresenta os resultados obtidos para o segundo perfil de estresse ($N_2 = 250$). Os resultados para o segundo perfil, semelhante ao primeiro, evidenciam uma redução no tempo de resposta, Δr (ms), no consumo de memória, u_{Mem}, e CPU, u_{CPU}, respectivamente, $1{,}80$, $1{,}71$ e $1{,}46$ vezes. Mais uma vez, diferentes valores de α na técnica de poda não resultaram em melhorias significativas nos parâmetros analisados.

TABLE VI: Tempo de resposta, Δr (ms), consumo de memória, u_{Mem} (MiB), e consumo de CPU, u_{CPU} (mCore), de cada microsserviço para o primeiro perfil de estresse ($N_2 = 250$).

α	b	Δr (ms)	u_{Mem} (MiB)	u_{CPU} (mCore)
0,00	32	16,23 ± 3,122	702	135
0,50	32	16,78 ± 3,214	706	136
1,25	32	15,34 ± 2,767	701	134
0,00	8	9,30 ± 2,666	412	94
0,50	8	9,25 ± 2,714	414	95
1,25	8	8,71 ± 2,991	411	94

V. Conclusão

Este trabalho apresentou uma abordagem de compressão consciente de modelos de *deep learning*, baseada na aplicação iterativa de poda seguida de quantização ($P \rightarrow Q$), visando a otimização de recursos computacionais em ambientes de microsserviços orquestrados com Kubernetes. A proposta foi avaliada utilizando o modelo VGG-16 treinado com o conjunto de dados CIFAR-10, considerando diferentes configurações de compressão parametrizadas por α e b. Os resultados experimentais demonstraram que a técnica $P \rightarrow Q$ é eficaz na redução do tamanho dos modelos, com compressão de até $5{,}6$ vezes, mantendo perdas de acurácia inferiores a 7%. Além disso, foram observados ganhos significativos de desempenho nos testes de inferência sob carga, com redução de até $1{,}8$ vezes na latência de resposta, $1{,}71$ vezes no uso de memória e $1{,}46$ vezes no consumo de CPU em comparação com o modelo não comprimido. Esses ganhos tornam o processo de escalonamento horizontal mais eficiente, contribuindo diretamente para a responsividade e a escalabilidade do sistema. Dessa forma, conclui-se que a compressão iterativa por poda e quantização é uma estratégia viável e vantajosa para a implantação de modelos DNN em arquiteturas distribuídas baseadas em microsserviços. Trabalhos futuros incluem a investigação do impacto da compressão em modelos de maior porte, a adoção de técnicas adaptativas baseadas em contexto e a aplicação da abordagem em tarefas além da classificação de imagens.

Agradecimentos

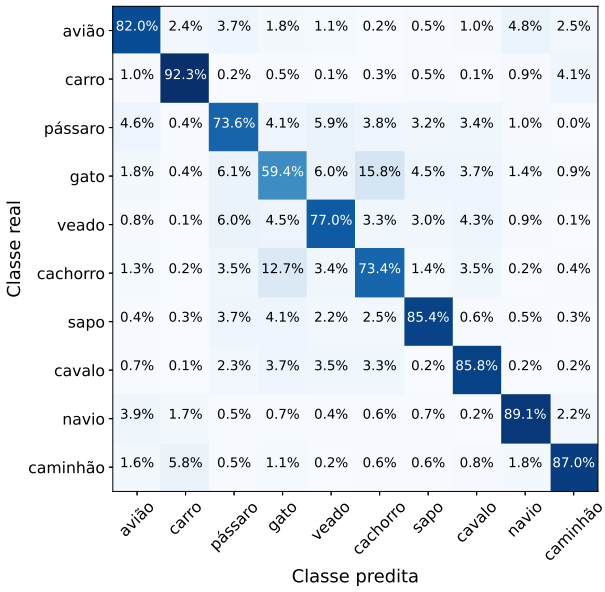
Os autores agradecem ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pelo suporte e financiamento.

References

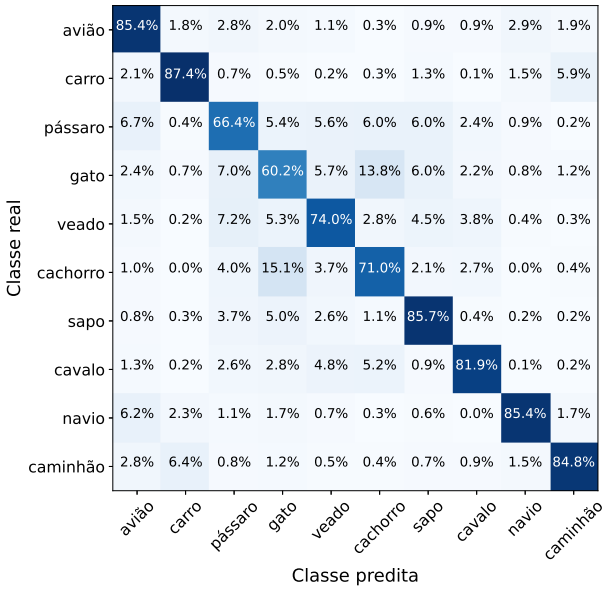
[1] (2015) Gpu-based deep learning inference: A performance and power analysis. nvidia. Acesso em: 10 jul. 2021. [Online]. Available: https://www.nvidia.com/content/tegra/embedded-systems/pdf/jetson_tx1_whitepaper.pdf

[2] D. Blalock, J. J. G. Ortiz, J. Frankle, and J. Guttag, "What is the state of neural network pruning?" arXiv preprint arXiv:2003.03033, 2020.

[3] Q. Huang, K. Zhou, S. You, and U. Neumann, "Learning to prune filters in convolutional neural networks," in 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2018, pp. 709–718.



(a) Modelo sem compressão.



(b) Poda seguida de quantização com $\alpha = 1{,}25$ e $b = 8$ bits

Fig. 5: Matrizes de confusão de modelo não comprimido e de Modelo comprimido na classificação do CIFAR-10

[4] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, "Haq: Hardware-aware automated quantization with mixed precision," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2019, pp. 8612–8620.

[5] A. S. Rakin, J. Yi, B. Gong, and D. Fan, "Defend deep neural networks against adversarial examples via fixed and dynamic quantized activation functions," arXiv preprint arXiv:1807.06714, 2018.

[6] S. Jung, C. Son, S. Lee, J. Son, J.-J. Han, Y. Kwak, S. J. Hwang, and C. Choi, "Learning to quantize deep networks by optimizing quantization intervals with task loss," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2019, pp. 4350–4359.

[7] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 42, no. 3, pp. 568–579, 2020.

[8] W. Zhe, J. Lin, V. Chandrasekhar, and B. Girod, "Optimizing the bit allocation for compression of weights and activations of deep neural networks," in 2019 IEEE International Conference on Image Processing (ICIP), 2019, pp. 3826–3830.

[9] H. Kung, B. McDanel, and S. Q. Zhang, "Term revealing: Furthering quantization at run time on quantized dnns," arXiv preprint arXiv:2007.06389, 2020.

[10] M. A. C. Fernandes and H. T. Kung, "A novel training strategy for deep learning model compression applied to viral classifications," in 2021 International Joint Conference on Neural Networks (IJCNN), 2021, pp. 1–9.

[11] K. Ramesh, A. Chavan, S. Pandit, and S. Sitaram, "A comparative study on the impact of model compression techniques on fairness in language models," in Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 15 762–15 782. [Online]. Available: https://aclanthology.org/2023.acl-long.878

[12] M. A. Goldbarg, M. Balza, S. N. Silva, L. M. D. Silva, and M. A. C. Fernandes, "A novel training strategy for deep learning model compression applied to automatic modulation classification," IEEE Open Journal of the Communications Society, vol. 6, pp. 477–492, 2025.

[13] I. Oumoussa and R. Saidi, "Evolution of microservices identification in monolith decomposition: A systematic review," IEEE Access, vol. 12, pp. 23 389–23 405, 2024.

[14] J. Noor, M. B. Faysal, M. S. Amin, B. Tabassum, T. R. Khan, and T. Rahman, "Kubernetes application performance benchmarking on heterogeneous cpu architecture: An experimental review," High-Confidence Computing, p. 100276, 2024.

[15] J. Park, U. Choi, S. Kum, J. Moon, and K. Lee, "Accelerator-aware kubernetes scheduler for dnn tasks on edge computing environment," in 2021 IEEE/ACM Symposium on Edge Computing (SEC). IEEE, 2021, pp. 438–440.

[16] D. R. Torres, C. Martín, B. Rubio, and M. Díaz, "An open source framework based on kafka-ml for distributed dnn inference over the cloud-to-things continuum," Journal of Systems Architecture, vol. 118, p. 102214, 2021.

[17] V. Dakić, M. Kovač, and J. Slovinac, "Evolving high-performance computing data centers with kubernetes, performance analysis, and dynamic workload placement based on machine learning scheduling," Electronics, vol. 13, no. 13, p. 2651, 2024.

[18] C.-D. Bak and S.-J. Han, "Efficient scaling on gpu for federated learning in kubernetes: A reinforcement learning approach," in 2024 International Technical Conference on Circuits/Systems, Computers, and Communications (ITC-CSCC). IEEE, 2024, pp. 1–6.

[19] M. Zhu and S. Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression." arXiv preprint arXiv:1710.01878, 2017. [Online]. Available: https://arxiv.org/pdf/1710.01878.pdf

[20] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," Journal of Machine Learning Research - Proceedings Track, vol. 9, pp. 249–256, 01 2010.

[21] P. Pietron, D. Szlachetko, and M. Śmieja, "Compression-aware design of deep autoencoders for multivariate time series anomaly detection," 2024.

[22] W. Kim, M. Choi, S. J. Kwon, and J. Shin, "Nonlinear quantization for vision transformers: Why and how?" 2024.

[23] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Numerical Recipes: The Art of Scientific Computing, 3rd ed. Cambridge University Press, 2007.

[24] P. Deutsch, "Rfc1951: Deflate compressed data format specification version 1.3," 1996.

[25] A. Z. Karen Simonyan, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[26] J. Brownlee. (2020) How to develop a cnn from scratch for cifar-10 photo classification. Machine Learning Mastery. Acesso em: 14 jul. 2021. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/

[27] Apache Software Foundation, Apache JMeter: User Manual, 2024, https://jmeter.apache.org.

[28] Prometheus Authors, Prometheus: Monitoring System & Time Series Database, 2024, https://prometheus.io.