Uma Arquitetura de Rede Neural com Auxílio da Nuvem para Dispositivos Computacionalmente Limitados

Caio Gevegir Miguel Medeiros[1], **Pedro Cruz**[1], **Rodrigo de Souza Couto**[1] *

[1]Universidade Federal do Rio de Janeiro - GTA/PEE-COPPE/DEL-Poli

{cgevegir,cruz,rodrigo}@gta.ufrj.br

Abstract. *Deep Neural Networks (DNNs) may be unfeasible on constrained devices due to their computational complexity. This work proposes an optimized neural network model that runs on these devices at the cost of lower accuracy, but that is supported by a DNN hosted on the cloud if local inferences do not reach a defined confidence threshold. Experiments using handwritten numerical digits show that the proposed model has low memory usage and lower training and inference times compared to a known DNN. In addition, the strategy of inferring data locally before querying to the cloud reduced the average inference time to at least a quarter of its original time, holding an accuracy of 96%.*

Resumo. *A execução de redes neurais profundas (Deep Neural Networks – DNNs) pode ser inviável em dispositivos de baixo poder computacional. Este trabalho propõe um modelo de rede neural otimizado para esses dispositivos, ao custo de uma menor acurácia, mas auxiliado por uma DNN na nuvem caso a inferência não atinja um determinado valor de confiança. Experimentos com dígitos escritos à mão mostram que o modelo ocupa baixa quantidade de memória, além de possuir tempo de treinamento e de inferência menores quando comparado a uma DNN conhecida. Adicionalmente, a estratégia de realizar a inferência local antes de consultar a nuvem reduziu o tempo médio de inferência em pelo menos quatro vezes, com uma acurácia de 96%.*

1. Introdução

A robustez das redes neurais profundas (*Deep Neural Networks* – DNNs) atuais veio ao preço de um alto custo de processamento e uso de memória [Bochie et al., 2021]. Portanto, torna-se inviável implementar alguns modelos de DNN diretamente em dispositivos de baixo poder computacional, como computadores pessoais, *smartphones* e sistemas embarcados [Matsubara et al., 2022].

Uma alternativa para utilizar redes neurais profundas em dispositivos de poder computacional mais baixo é terceirizar sua execução. Isto é, executar as redes neurais profundas em máquinas virtuais ou contêineres hospedados na nuvem ou na borda, que possuam as especificações adequadas para o seu processamento. Assim, os dispositivos de menor poder computacional podem enviar os dados para a nuvem, que realiza a inferência

*O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) - Código de Financiamento 001. O trabalho também foi financiado pelo CNPq, FAPERJ (E-26/010.002174/2019 e E-26/201.300/2021), PR2/UFRJ, e pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP), auxílio no. 2015/24494-8.

desejada, e, por fim, retorna a classificação dos dados para o usuário final. Essa alternativa, por outro lado, também possui seus problemas; o tempo de resposta, por exemplo, se torna sujeito a perturbações decorrentes da rede entre o dispositivo e a nuvem, como atrasos e falhas na conexão. Além disso, quanto mais poderosa a máquina virtual e maior a sua demanda, mais caro será esse serviço.

O *offloading* adaptativo [Pacheco et al., 2021a, Pacheco et al., 2021b] é uma solução que busca dividir o processamento de uma rede neural entre o dispositivo e nuvem. O dispositivo executa localmente uma parte da DNN e recorre à nuvem somente se o resultado não for considerado confiável. Porém, o dispositivo ainda deve ser capaz de executar uma parte do modelo DNN, o que pode não ser possível para dispositivos muito restritos. Também é possível implementar redes neurais em dispositivos de baixo poder computacional utilizando tipos de redes neurais que otimizam o espaço em memória e o processamento, como redes neurais sem peso (*Weighless Neural Networks* – WNNs) [Aleksander et al., 2009, Aleksander et al., 1984] e redes neurais binarizadas (*Binarized Neural Networks* – BNNs) [Hubara et al., 2016, Rastegari et al., 2016]. As WNNs já são aplicadas diretamente em FPGAs (*Field Programmable Gate Arrays*) [Torres et al., 2020], [Susskind et al., 2022] e há estudos voltados para aplicação de BNNs em sistemas embarcados [McDanel et al., 2017]. Uma limitação das WNNs e BNNs é que a simplicidade desses modelos se reflete em suas acurácias, tipicamente mais baixas do que a de modelos mais robustos.

Neste trabalho, utiliza-se uma abordagem híbrida entre a execução em um dispositivo de capacidade limitada e a execução na nuvem. Primeiro, desenvolve-se um modelo de classificação baseado em WNNs. O modelo é simples o suficiente para ser executado em dispositivos limitados, mas possui acurácia inferior aos modelos mais complexos, que precisam ser executados na nuvem. Então, este trabalho utiliza uma métrica para decidir a confiabilidade de cada classificação no dispositivo. As classificações consideradas confiáveis são utilizadas, enquanto as não confiáveis são enviadas para classificação pelo modelo mais robusto da nuvem, que possui acurácia maior que 98%. Os resultados mostram que é possível reduzir em pelo menos quatro vezes o tempo médio de resposta para o problema de classificação de números escritos à mão quando comparado ao tempo de resposta puramente da nuvem, com uma acurácia média total de 96%.

Este artigo está organizado da seguinte forma. A Seção 2 aborda os trabalhos relacionados. A Seção 3 apresenta o sistema proposto. A Seção 4 descreve a metodologia, enquanto a Seção 5 apresenta os resultados. Por fim, a Seção 6 conclui o trabalho.

2. Trabalhos Relacionados

Há diversos trabalhos que abordam o uso de redes neurais em dispositivos limitados. As Redes Neurais Quantizadas (*Quantized Neural Networks* - QNN) [Hubara et al., 2017] são soluções para reduzir o tamanho das DNNs. Geralmente, os pesos das DNNs tradicionais são armazenados em pontos flutuantes de 32 bits. DNNs conhecidas da literatura, como AlexNet, ResNet e Inception possuem milhões de parâmetros treináveis e bilhões de operações envolvendo pontos flutuantes (FLOP), o que demanda alto poder computacional e capacidade de armazenamento em suas implementações. No processo de quantização, os pesos são comprimidos em representações que requerem menor espaço na memória, como pontos flutuantes menos

precisos utilizando uma quantidade reduzida de bits [Hubara et al., 2017] ou números inteiros para reduzir o custo computacional de FLOPs [Jacob et al., 2018]. O modelo para dispositivos limitados do presente trabalho também realiza simplificações de forma a representar cada peso com apenas 1 byte. Entretanto, utiliza-se apenas uma camada oculta, reduzindo o custo computacional comparando-se com as QNNs.

As Redes Neurais Binarizadas (*Binarized Neural Networks* - BNN) [Hubara et al., 2016] são exemplos extremos da quantização de redes neurais pois utilizam apenas dois valores de peso (-1 e $+1$) e álgebra booleana em suas funções de ativação. Essas redes neurais reduzem drasticamente o uso de memória e o consumo de energia [Hubara et al., 2016], sendo o XNOR-Net [Rastegari et al., 2016] um modelo de destaque. Esses modelos conseguem trabalhar com conjuntos de dados de imagens mais sofisticadas, como o ImageNet[1], mas ainda utilizam uma quantidade relevante de camadas de convolução. Essas camadas de convolução resultam em uma alta complexidade computacional. O modelo proposto neste trabalho não possui camadas de convolução ou de qualquer outra aritmética mais complexa, possibilitando sua execução em dispositivos computacionalmente limitados.

WNNs são modelos de redes neurais baseados em células RAM (*Random Access Memory*), que são responsáveis por armazenar os padrões observados no conjunto de dados a serem inferidos. Essas células RAM estão agrupadas em discriminadores de classes para contabilizar e calcular a probabilidade de uma amostra do conjunto de dados pertencer a cada classe possível. Por fim, a classe com maior probabilidade é a que será atribuída para aquela amostra [Aleksander et al., 2009]. Um dos modelos a ser destacado é o WiSARD (*Wilkes, Stonham and Aleksander Recognition Device*), desenvolvido comercialmente para o reconhecimento de padrões em imagens [Aleksander et al., 1984]. Essas redes possuem a vantagem de serem treinadas e realizarem inferências utilizando aritmética simples, como operações matemáticas básicas e álgebra booleana. As redes neurais profundas tradicionais, por outro lado, utilizam operações mais custosas, como trigonometria, multiplicação matricial, convolução e recursão [Bochie et al., 2021].

Como as WNNs são dependentes da observação de padrões, o tamanho do modelo pode crescer indefinidamente para calcular a observação de cada combinação de padrão [Aleksander et al., 2009, Aleksander et al., 1984]. Este trabalho propõe um modelo com um tamanho mínimo de células RAM, capaz de caber na memória de dispositivos embarcados, mas sem perda significativa de sua capacidade de generalização. As possíveis perdas são compensadas por um modelo mais robusto, executado na nuvem.

Há alternativas para otimizar o tamanho e o desempenho do modelo WiSARD, como o uso de *hashes* e filtros de Bloom, possibilitando até mesmo implementá-los em FPGAs [Bloom, 1970, Santiago et al., 2020, Susskind et al., 2022]. Porém, essas alternativas utilizam estruturas de dados que podem não ser viáveis em dispositivos mais limitados. Por isso, o modelo apresentado no presente trabalho é reduzido sem a necessidade de estruturas de dados complexas.

O *offloading* adaptativo é um exemplo de uso da nuvem ou da borda para dispositivos limitados [Pacheco et al., 2021a, Pacheco et al., 2021b], no qual a DNN pode ser particionada em diversos pontos. Assim, camadas iniciais da DNN são executadas no próprio

[1]https: //image-net.org/index.php

dispositivo, enquanto a borda ou a nuvem executa as demais [Teerapittayanon et al., 2017, Xue et al., 2021, Chen et al., 2021]. Além do particionamento da rede, o *offloading* adaptativo utiliza estratégias de DNNs com saídas antecipadas [Teerapittayanon et al., 2016]. Essas DNNs possuem saída intermediárias entre algumas de suas camadas, que calculam a confiança de uma classificação. Caso esse valor ultrapasse um limiar definido, isto é, a classificação tenha um nível de confiança aceitável, finaliza-se o processo de inferência. Isso evita o processamento de todas as camadas posteriores e economiza tempo e energia. Caso contrário, as demais camadas são processadas. Esse processo continua até a chegada de um ponto de particionamento. Nesse caso, a saída da camada de particionamento é enviada para a nuvem ou para a borda, continuando o processo de inferência. Este trabalho utiliza a mesma ideia de delegar à nuvem, ou à borda, o processamento que o dispositivo não é capaz de fazer. Porém, para lidar com capacidades de processamento ainda menores, utilizam-se modelos diferentes na nuvem e no dispositivo. Assim, o dispositivo emprega o modelo proposto, enquanto a nuvem executa uma DNN tradicional.

3. Solução Proposta

A Figura 1[2] ilustra o cenário considerado, no qual um dispositivo gera amostras que devem ser classificadas por um dispositivo de baixo custo. Assim, propõe-se que amostras para classificação sejam entregues ao dispositivo, que executa um modelo simples de inferência. O nível de confiança na classificação é avaliado e, se a inferência é considerada confiável, a inferência é concluída. Caso contrário, a amostra é enviada para um modelo executado na nuvem, que possui acurácia alta, e então conclui a inferência.

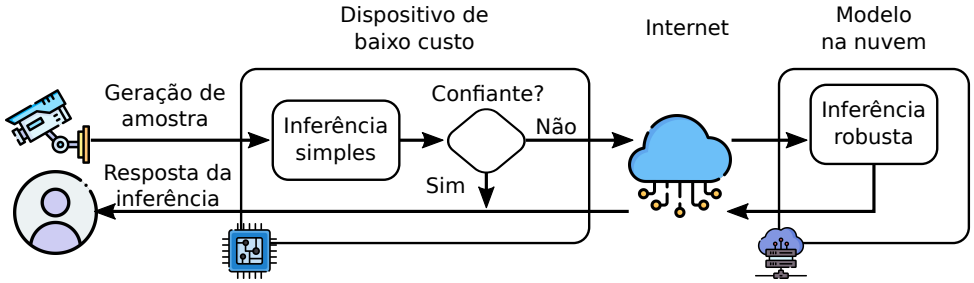

Figura 1. Cenário estudado de inferência em dispositivos de baixo custo.

Este trabalho considera dispositivos comerciais limitados com preço na faixa de US$ 5,00 a US$ 20,00 e memórias na faixa de 32kB a 512kB, como o Arduino[3] e o ESP-32[4]. Tais dispositivos são viáveis para operações mais simples e possuem baixo custo. Visando gerar um modelo capaz de ser executado por esses dispositivos, arquitetou-se um modelo capaz de reunir as vantagens econômicas de consumo de memória e energia das WNNs e das BNNs. Adicionalmente, os dispositivos limitados devem executar somente a fase de inferência. A fase de treinamento do modelo é realizada por um dispositivo menos limitado ou por uma máquina virtual, executada na nuvem.

Tendo em vista as limitações desse tipo de modelo, é proposta uma arquitetura com *offloading* das inferências para um modelo mais acurado, executado na nuvem. Para isso, seguem-se estes princípios na construção do modelo que executa no dispositivo:

[2]Esta figura possui ícones de Flaticon.com
[3]https://docs.arduino.cc/ hardware/uno-rev3
[4]https://www.espressif.com/en/ products/modules/esp32

- O treinamento deve ser realizado em um único lote do conjunto de dados;
- O espaço ocupado na memória por cada peso não deve ultrapassar 1 byte;
- A rede deve ter apenas uma única camada oculta, ou seja, uma única etapa de processamento entre entrada e saída de dados;
- As operações lógicas e aritméticas, assim como as estruturas de dados implementadas, devem ser simples o suficiente para o programa ocupar pouco espaço de armazenamento.

Esses princípios foram escolhidos visando reduzir o espaço ocupado na memória pelo modelo, assim como sua complexidade computacional, a fim de reduzir o tempo de resposta.

3.1. Matriz de observação

O modelo proposto toma como base os discriminadores de células RAM utilizados em redes neurais sem peso. Esses componentes observam e detectam padrões em diferentes seções de uma amostra a ser inferida. Para cada padrão observado em uma determinada seção, o discriminador daquela seção retorna a classe mais provável. Ao final, a classe mais contabilizada por todos os discriminadores é a que será inferida para aquela amostra [Aleksander et al., 2009].

Na solução proposta, o modelo implementa pesos como células RAM de tamanho unitário, ou seja, que observam a menor unidade indivisível da amostra a ser inferida. Por exemplo, um pixel em uma imagem. Para simplificar a descrição da solução proposta, em vez de trabalhar com os termos "células RAM" e "discriminadores", opta-se por defini-las como "matriz de observação", com as linhas representando os discriminadores de cada classe C e as colunas sendo os índices das unidades indivisíveis da amostra I.

A Equação 1 mostra a definição da matriz de observação M em função do tamanho do dado $|I|$ em bytes e a quantidade de classes $|C|$. Além disso, cada elemento $m_{c,i}$ da matriz precisa respeitar os limites absolutos dos valores dos pesos para que possam ser armazenados em um único byte. Assim, o tamanho em bytes do modelo é $S = |\mathcal{C}| \times |\mathcal{I}|$, crescendo linearmente com o número de classes e o tamanho dos dados.

$$\mathcal{M} \in \mathbb{N}^{|\mathcal{C}|\times|\mathcal{I}|}, 0 \leq m_{c,i} \leq 255 \tag{1}$$

3.2. Treinamento

O modelo WiSARD treina seus parâmetros, os discriminadores de células RAM, armazenando o valor lógico 1 em seus endereços quando um padrão de uma classe é observado em uma determinada posição [Aleksander et al., 1984]. Essa abordagem otimiza bastante o espaço ocupado na memória, pois pode resumir o tamanho dos parâmetros a um único *bit*. Entretanto, isso pode abstrair demais as características do conjunto de treinamento do conjunto de dados, levando a muitas perdas de informação. Por isso, no modelo proposto, os pesos são treinados como acumuladores, para dar mais ênfase à frequência de observação [Grieco et al., 2010]. O Algoritmo 1 define, em poucas linhas, o processo de treinamento do modelo. Inicia-se com a matriz de observação zerada e, para cada unidade do dado, os pesos correspondentes às classes e àquela posição no dado são incrementados com seu valor. O treinamento termina quando todo o conjunto de dados for processado. Assim, o Algoritmo 1 retorna a matriz de observação M treinada contendo os pesos para cada classe.

Algoritmo 1 Treinamento do Modelo

$\mathcal{M} \leftarrow [0]_{|\mathcal{C}|\times|\mathcal{I}|}$
para cada item $\mathcal{I}$ no conjunto de dados e sua classe c **faça**
    para $i \leftarrow 0, |\mathcal{I}|$ **faça**
        $\mathcal{M}[c][i] \leftarrow \mathcal{M}[c][i] + \mathcal{I}[i]$
retorne $\mathcal{M}$

3.3. Pós-treinamento

Ao fim do treinamento, tem-se uma matriz cujos elementos são o acúmulo de observações de valores em cada índice dos dados de entrada e suas classes. Porém, se o conjunto de treinamento do conjunto de dados for muito grande, isso resultará em altos valores absolutos, o que levará o modelo a um *overtraining* ou até mesmo ultrapassar o valor limite estipulado de 255, necessário para possibilitar o armazenamento dos pesos em 1 byte. Por isso, uma etapa importante para viabilizar o modelo nessas limitações é reduzir esses valores.

Um solução para a saturação dos valores é realizar um escalonamento linear para o intervalo 0-255. Entretanto, para reduzir ainda mais o impacto de valores excessivamente altos, opta-se por realizar um escalonamento logarítmico. Assim, valores muito altos são proporcionalmente mais atenuados, aumentando a capacidade de generalização do modelo. A Equação 2 sintetiza a operação de atenuação. Todos os elementos da matriz de observação são atualizados para o logaritmo dos seus valores originais. Tal operação é possível pois a fase de treinamento é realizada em um dispositivo sem limitações.

$$m'_{c,i} = \begin{cases} 0, & \text{se } m_{c,i} = 0 \\ ceil[\log_{10}(m_{c,i})], & \text{caso contrário} \end{cases}, \forall c, i \tag{2}$$

3.4. Inferência

Uma estratégia para economizar tempo de execução ao processar os dados de entrada para inferência é o uso de um limiar de ativação. Isto é, uma condição que cada parâmetro do dado precisa atingir para que um passo da instrução de inferência seja realizado. Assim, informações consideradas irrelevantes podem ser descartadas durante o processamento. Essa condição pode ser global, ou seja, a mesma para todos os parâmetros, ou particular de cada parâmetro ou conjunto deles. Neste trabalho, conforme será detalhado na Seção 4, é utilizado um limiar global.

O processo de inferência é similar ao de treinamento. Primeiramente, declara-se um vetor de resposta unidimensional R com $|C|$ elementos, um para cada classe. Em seguida, para cada parâmetro do dado, caso este atinja o limiar de ativação, os elementos r_c do vetor de resposta são incrementados com o valor de $m_{c,i}$ da matriz de observação. Ao final do processamento, o índice do vetor de resposta que contiver o maior valor total será a classe inferida para aquele dado. O Algoritmo 2 mostra, também em poucas linhas, como se dá o processo de inferência.

3.5. Limiar de confiança

O modelo proposto decide se irá realizar *offloading* para a nuvem a partir de um valor de confiança na inferência realizada. Para isso, seguindo a definição de confiança

Algoritmo 2 Inferência de Dados

Precondições: $\mathcal{M}, \mathcal{I}$

$\mathcal{R} \leftarrow [0]_{|\mathcal{C}|}$
para $i \leftarrow 0, |\mathcal{I}|$ **faça**
    se $\mathcal{I}[i]$ atinge o seu limiar de ativação **então**
        para $c \leftarrow 0, |\mathcal{C}|$ **faça**
            $\mathcal{R}[c] \leftarrow \mathcal{R}[c] + \mathcal{M}[c][i]$
retorne $argmax(\mathcal{R})$

dada por [Aleksander et al., 1984], calcula-se a diferença relativa entre o maior e o segundo maior valor calculados no vetor de resposta, como mostrado na Equação 3. Um valor alto de confiança significa que a resposta do modelo tem alta probabilidade de ser correta. Uma confiança baixa indica que os dois maiores valores retornados são próximos e, assim, a resposta do modelo tem baixa probabilidade de estar correta.

$$T = 1 - \frac{2^{nd}\max(\mathcal{R})}{\max(\mathcal{R})} \tag{3}$$

A partir do valor de confiança na inferência, é possível estabelecer um limiar de confiança, que define se a inferência realizada pelo modelo é confiável o suficiente. Caso contrário, deve haver uma nova inferência em um modelo mais robusto na nuvem por meio de *offloading*. A escolha de um limiar de confiança deve ser cuidadosa pois, quanto maior seu valor, maior poderá ser a acurácia do modelo. Porém, isso aumenta o número de inferências descartadas no dispositivo. Consequentemente, mais dados são enviados de dados à nuvem, aumentando o tempo de resposta.

4. Metodologia Experimental

A proposta é avaliada utilizando um estudo de caso de identificação de dígitos escritos à mão. Esta seção detalha a metodologia experimental para avaliação da proposta, descrevendo o conjunto de dados, a configuração dos dispositivos, os procedimentos experimentais e outros ajustes realizados.

4.1. Conjunto de dados

O conjunto de dados utilizado no experimento é o MNIST [LeCun et al., 1995], que consiste em imagens monocromáticas 28x28 com dígitos numéricos manuscritos de 0 a 9. O MNIST é um conjunto de dados utilizado na literatura para medir o desempenho de métodos para dispositivos de baixo poder computacional [Rajapakse et al., 2022]. O objetivo da metodologia é realizar a inferência de qual algarismo está escrito na imagem. Os dados estão distribuídos em 50.000 imagens no conjunto de treinamento, 10.000 no conjunto de validação e 10.000 no conjunto de teste.

4.2. Dispositivos

A Figura 2 ilustra como os dispositivos utilizados estão conectados entre si, enquanto a Tabela 1 resume suas especificações. A comunicação entre o computador pessoal e o dispositivo de baixo custo é realizada pela porta serial. O acesso à Internet pelo dispositivo de baixo custo é feito por meio de WiFi e um provedor residencial. Os dispositivos são listados e descritos a seguir.

- **Computador pessoal:** Máquina com sistema operacional Windows 10, processador Ryzen 7 5700G 3,8 GHz e 16 GB de memória RAM. Esta máquina é responsável por treinar o modelo e enviar as imagens para o dispositivo de baixo custo por meio de comunicação serial. Note que o computador pessoal é utilizado apenas para fins experimentais. Em um cenário real, o próprio dispositivo pode capturar a imagem, como em uma câmera de segurança.
- **Dispositivo de baixo custo:** ESP32-S3 com processador Dual-core Xtensa LX7 CPU 240 MHz, 512 kB de memória SRAM e conexão WiFi 2,4 GHz nativa. Este dispositivo é o núcleo deste experimento, responsável por receber o modelo proposto e realizar as inferências das imagens recebidas por comunicação serial do computador pessoal. O dispositivo também decide quais imagens são enviadas para *offloading*.
- **Nuvem:** Instância de máquina virtual AWS *t2.medium* hospedada na região *us-east-2* com sistema operacional Linux Ubuntu 22.04, 4 GB de memória RAM e processador Intel Xeon escalável até 3,3 GHz [5]. Essa máquina recebe os dados das imagens provenientes do *offloading* do dispositivo de baixo custo. A nuvem utiliza uma DNN convolucional LeNet-5 [LeCun et al., 1995] pré-treinada para realizar a inferência. A resposta da inferência é enviada para o dispositivo de baixo custo.

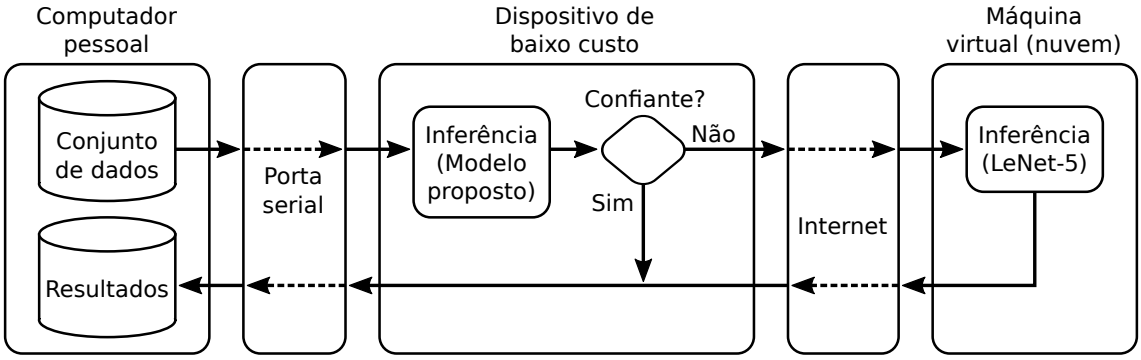

Figura 2. Configuração do experimento.

Equipamento	Memória RAM/SRAM	Processador
Computador Pessoal	16 GB	Ryzen 7 5700G 3.8 GHz
ESP32-S3	512 kB	Xtensa LX7 CPU 240 MHz
AWS *t2.medium*	4 GB	Intel Xeon 3.3 GHz

Tabela 1. Especificações dos equipamentos utilizados no experimento.

4.3. Descrição dos experimentos

Os experimentos deste trabalho, cujos resultados são discutidos na Seção 5, seguem três etapas:

- Validação e comparação;
- Ajuste do limiar de confiança;
- Teste dispositivo-nuvem.

[5]https: //aws.amazon.com/pt/ec2/instance-types/

As duas primeiras etapas são executadas em um computador pessoal. Na etapa de validação e comparação, treina-se tanto o modelo proposto quanto uma rede neural convolucional utilizando o conjunto de treinamento. Esses modelos são comparados em relação ao tamanho e desempenho utilizando o conjunto de validação. Na etapa de ajuste do limiar de confiança, utilizam-se as inferências do conjunto de validação para ajustar o limiar de confiança de forma que as inferências consideradas confiáveis tenham uma acurácia de 95%. Por fim, na etapa de teste dispositivo-nuvem, o modelo é exportado do computador pessoal e carregado no dispositivo de baixo custo com o limiar de confiança ajustado. O modelo da rede neural convolucional é carregado na nuvem e configurado para receber as inferências resultantes do *offloading*.

4.4. Ajustes no treinamento e inferência

O conjunto de dados é formado por imagens monocromáticas, ou seja, as intensidades de tons de cinza são definidas por valores numéricos no intervalo de 0 a 255. Assim, para operar com um limiar de ativação simples, esse limiar possui valor zero. Tal decisão torna o processo de inferência muito mais prático em linguagens de alto-nível pois essa se torna uma única multiplicação matricial e uma aplicação do operador $sign$, como descrito na Equação 4.

$$c = argmax[sign(\mathcal{I}) \times \mathcal{M}^T] \tag{4}$$

O operador $sign$ retorna -1 se o valor for negativo, +1 se for positivo ou 0 se for nulo. Como os dados do MNIST só possuem valores positivos e nulos, o $sign$ se torna o operador adequado para aplicar o limiar de ativação, somar as contribuições relevantes no vetor de resposta e obter a classe inferida.

O modelo é executado sob as mesmas condições de outras redes neurais profundas para avaliar seu desempenho. Por isso, o algoritmo de treinamento do modelo foi desenvolvido na linguagem *Python* utilizando a biblioteca numérica *NumPy*, também utilizada no desenvolvimento de outras redes neurais profundas. Adicionalmente, são realizadas inferências com a mesma linguagem e biblioteca conforme a Equação 4 utilizando o conjunto de validação. Ao final do treinamento, os pesos da matriz de observação são exportados para o código-fonte que será carregado no dispositivo de baixo custo.

5. Resultados

Esta seção discute os resultados obtidos nos experimentos descritos na Seção 4. Assim, apresentam-se análises de tempo de treinamento e inferência, estimativa de espaço ocupado na memória, acurácia de validação e teste, além da distribuição das amostras inferidas no dispositivo de baixo custo e na nuvem.

5.1. Avaliação do modelo

A Tabela 2 mostra comparações do modelo proposto com a rede neural convolucional LeNet-5. Ambos os modelos são treinados e avaliados com os mesmos conjuntos de treinamento e validação. A tabela compara, respectivamente, o tamanho ocupado em memória pelos parâmetros dos modelos, o tempo total necessário para completarem seus treinamentos, o tempo médio para realizarem a inferência de uma imagem durante a etapa de validação e, por fim, a acurácia do conjunto de validação.

	Proposta	LeNet-5
Tamanho (bytes)	7.840	240.000
Tempo Total de Treinamento (s)	0,0990	25,3686
Tempo Médio de Inferência (s)	0,00004	0,0256
Acurácia (%)	75,03	98,49

Tabela 2. Comparações do modelo proposto com a rede neural LeNet-5.

A Tabela 2 mostra que a LeNet-5 ocupa um espaço aproximado de 240 kB na memória, visto que possui cerca de 60.000 parâmetros treináveis e armazenados em variáveis do tipo *float* com tamanho 4 bytes. A solução proposta, com 7.840 parâmetros treinados armazenados em apenas 1 byte cada, ocupa 7.840 bytes. Isso mostra uma redução significativa de 87% de tamanho, que a possibilita ser armazenada em microcontroladores, como o ATmega328P, utilizado na plataforma Arduino UNO.

Os valores de tempo da Tabela 2 mostram que a solução proposta é consideravelmente mais rápida tanto no treinamento quanto na inferência. O treinamento do modelo proposto foi concluído em cerca de 0,099 segundos, sendo 256 vezes mais rápido do que o LeNet-5. Para a inferência, a diferença foi ainda mais significativa, sendo 640 vezes mais rápido ao processar os mesmos dados.

Apesar dos ganhos de tamanho e tempo, o compromisso disso é a redução da acurácia em aproximadamente 24 pontos percentuais, como mostrado na Tabela 2. A Figura 3 complementa a avaliação com a matriz de confusão da validação do modelo, mostrando que o modelo tem dificuldades para acertar o dígito cinco, enquanto uma quantidade significativa de inferências são feitas erroneamente como valor oito.

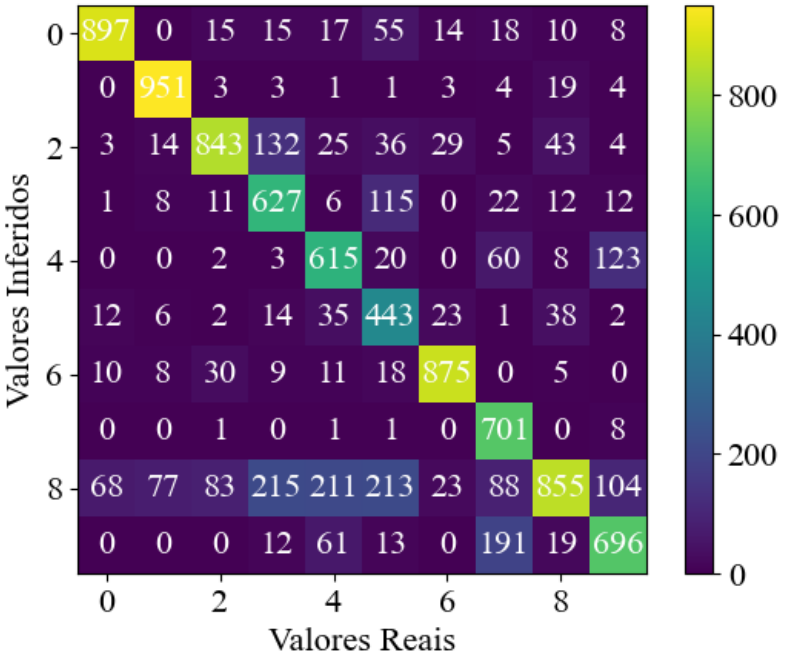

Figura 3. Matriz de confusão do modelo proposto para o conjunto MNIST.

5.2. Experimento dispositivo-nuvem

Após os ajustes de índice de confiança e seu limiar (Seções 3.5 e 4.3), realizam-se os experimentos com o conjunto de testes. A Figura 4 mostra a distribuição acumulada dos valores de confiança obtidos pelo modelo no conjunto de testes. Além disso, apresenta-se o limiar de confiança adotado, mostrando o efeito da escolha desse valor na quantidade de amostras enviadas para a nuvem. Dado o limiar escolhido, 57,9% das amostras possuem confiança menores que esse valor e são então enviadas para a nuvem. A Tabela 3 complementa esses resultados mostrando a porcentagem de inferências em cada local. Além disso, a tabela apresenta a acurácia em cada local do sistema e a total do sistema, considerando as amostras dos dois locais.

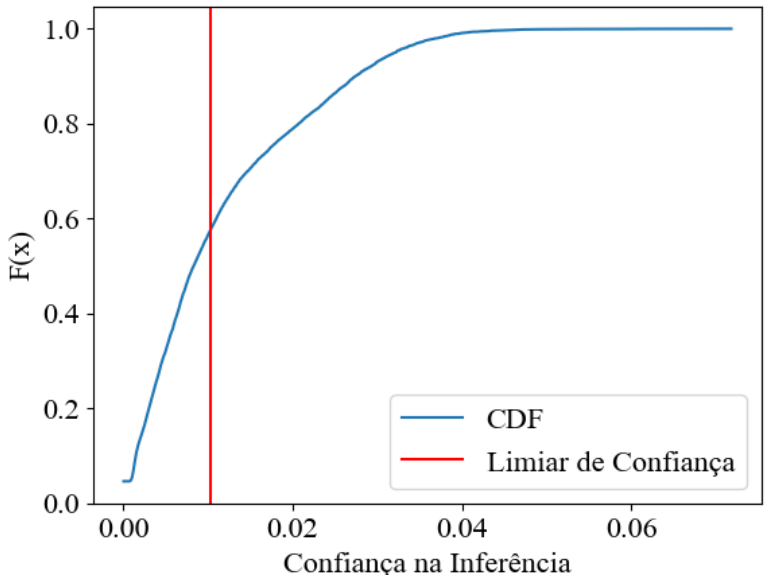

Figura 4. Distribuição acumulada da confiança na inferência e o limiar adotado.

Local	Porcentagem de Inferências (%)	Acurácia (%)
Dispositivo	42,1	94,4
Nuvem	57,9	98,3
Total	100,0	96,67

Tabela 3. Comparação entre quantidade e acurácia de inferências realizadas no dispositivo e na nuvem.

A partir da Tabela 3, é possível observar que uma quantidade relevante das imagens não precisou ser enviada para a nuvem. Isto é, mais de 40% das inferências foram confiáveis o suficiente para serem realizadas pelo modelo proposto e atingiram uma acurácia próxima de 94% no dispositivo. A acurácia total, considerando todos os locais de inferência, é de 96,5%. Isso mostra uma queda considerada pequena quando comparado ao desempenho do caso no qual todas as imagens são enviadas diretamente para a nuvem.

A Figura 5 mostra um diagrama de caixa (*boxplot*) do tempo de inferência das imagens do conjunto de testes. Nessa figura, separam-se em caixas distintas os resultados inferidos no dispositivo e na nuvem. É importante ressaltar que o tempo de inferência inclui o tempo de transmissão da comunicação serial entre o dispositivo e o computador

pessoal (Figura 2). Esse tempo é necessário para enviar as imagens e receber a resposta do dispositivo de baixo custo. Entretanto, além de ser pequeno, esse tempo não afeta a comparação entre os locais, visto que ambos usam a comunicação serial.

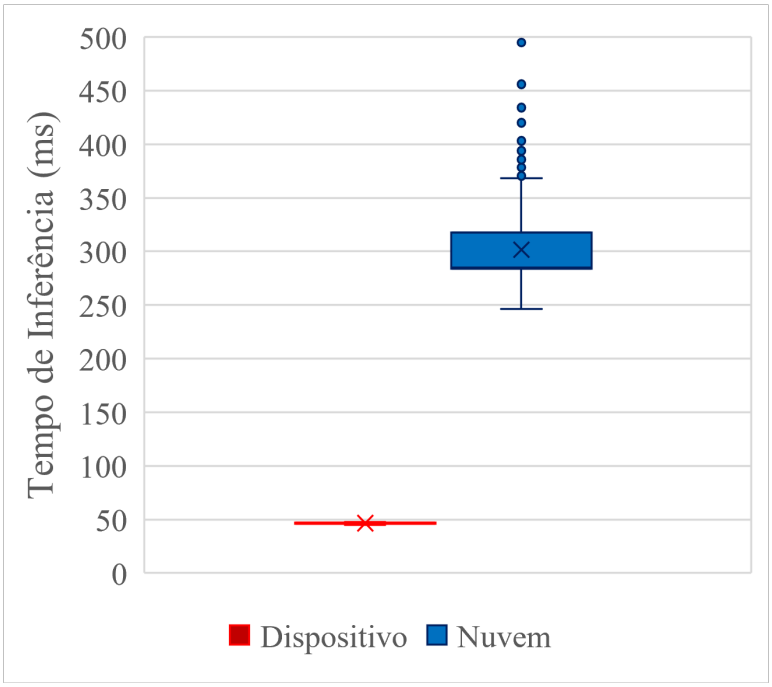

Figura 5. Diagrama de caixa do tempo de inferência em cada local.

A partir dos resultados da Figura 5, é possível notar que o tempo de inferência no dispositivo possui baixa variação, correspondendo a aproximadamente 50 ms. As inferências na nuvem, por sua vez, possuem alta variação e valores atípicos (*outliers*), dadas as condições da comunicação com o servidor via Internet. É importante notar que o tempo de inferência na nuvem já considera o tempo desperdiçado na primeira tentativa de realizar a inferência no dispositivo, de cerca de 50 ms. Mesmo descontando esse tempo do limiar inferior do diagrama (isto é, considerando uma inferência na nuvem de 200 ms, descontando os 50 ms), é possível notar que o modelo proposto reduz em pelo menos quatro vezes o tempo de inferência em comparação a uma solução que usa apenas a nuvem. Portanto, com apenas uma redução de aproximadamente dois pontos percentuais na acurácia (Tabela 3), é possível reduzir significativamente o tempo de inferência.

6. Conclusão

As redes neurais estão cada vez mais profundas para atingirem uma acurácia cada vez maior. Esse aumento da profundidade limita sua aplicação em dispositivos limitados. Este trabalho propôs um modelo voltado para esses dispositivos, reduzindo o armazenamento de memória e complexidade de processamento necessários na inferência, ao custo de uma penalidade na acurácia. Para reduzir essa penalidade, desenvolveu-se um esquema de *offloading*. Esse esquema consiste em calcular uma métrica de confiança para direcionar inferências consideradas incertas pelo modelo para outro mais acurado, que executa em uma máquina virtual na nuvem.

A solução proposta se mostrou rápida e compacta o suficiente para operar em dispositivos de *hardware* limitado como primeira linha de atuação em sistemas IoT, diminuindo a dependência da nuvem. Mesmo com suas limitações, o modelo se mostrou competente na inferência de dados simples, como dígitos numéricos monocromáticos.

Em trabalhos futuros, pretende-se explorar experimentos com novos conjuntos de dados, tanto de imagem quanto de outros formatos, como texto e áudio, como diferentes quantidades de classes. Além disso, pretende-se realizar experimentos em dispositivos com capacidade ainda menor do que o ESP32, testando os limites de operação do modelo. Por fim, é possível analisar estratégias de compressão, como armazenar os valores da matriz da observação em *nibbles*.

Referências

Aleksander, I., De Gregorio, M., França, F. M. G., Lima, P. M. V. e Morton, H. (2009). A brief introduction to weightless neural systems. Em *European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning (ESANN)*, p. 299–305. Citeseer.

Aleksander, I., Thomas, W. e Bowden, P. (1984). Wisard· a radical step forward in image recognition. *Sensor review*, 4(3):120–124.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13:422–426.

Bochie, K., Gilbert, M. S., Gantert, L., Barbosa, M. S. M., Medeiros, D. S. V. e Campista, M. E. M. (2021). A survey on deep learning for challenged networks: Applications and trends. *Journal of Network and Computer Applications*, 194:103213.

Chen, X., Zhang, J., Lin, B., Chen, Z., Wolter, K. e Min, G. (2021). Energy-efficient offloading for dnn-based smart iot systems in cloud-edge environments. *IEEE Transactions on Parallel and Distributed Systems*, 33(3):683–697.

Grieco, B. P., Lima, P. M., De Gregorio, M. e França, F. M. (2010). Producing pattern examples from "mental" images. *Neurocomputing*, 73(7):1057–1064.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R. e Bengio, Y. (2016). Binarized neural networks. *Advances in Neural Information Processing Systems*, 29.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R. e Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H. e Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. Em *Proceedings of the IEEE Conference on Computer Vision and Patter Recognition*, p. 2704–2713.

LeCun, Y., Jackel, L. D., Bottou, L., Cortes, C., Denker, J. S., Drucker, H., Guyon, I., Muller, U. A., Sackinger, E., Simard, P. et al. (1995). Learning algorithms for classification: A comparison on handwritten digit recognition. *Neural Networks: The Statistical Mechanics Perspective*, 261(276):2.

Matsubara, Y., Levorato, M. e Restuccia, F. (2022). Split computing and early exiting for deep learning applications: Survey and research challenges. *ACM Computing Surveys*, 55(5):1–30.

McDanel, B., Teerapittayanon, S. e Kung, H. T. (2017). Embedded binarized neural networks. *arXiv preprint arXiv:1709.02260*.

Pacheco, R., Couto, R. e Simeone, O. (2021a). Calibration-aided edge inference offloading via adaptive model partitioning of deep neural networks. Em *IEEE International Conference on Communications (ICC)*, p. 1–6.

Pacheco, R., Oliveira, F. R. e Couto, R. (2021b). Early-exit deep neural networks for distorted images: providing an efficient edge offloading. Em *IEEE Global Communications Conference (GLOBECOM)*, p. 1–6.

Rajapakse, V., Karunanayake, I. e Ahmed, N. (2022). Intelligence at the extreme edge: a survey on reformable tinyml. *ACM Computing Surveys*.

Rastegari, M., Ordonez, V., Redmon, J. e Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279.

Santiago, L., Verona, L., Rangel, F., Firmino, F., Menasché, D. S., Caarls, W., Breternitz Jr, M., Kundu, S., Lima, P. M. e França, F. M. (2020). Weightless neural networks as memory segmented bloom filters. *Neurocomputing*, 416:292–304.

Susskind, Z., Arora, A., Miranda, I. D. D. S., Villon, L. A. Q., Katopodis, R. F., de Araújo, L. S., Dutra, D. L. C., Lima, P. M. V., Franca, F. M. G., Breternitz Jr, M. et al. (2022). Weightless neural networks for efficient edge inference. *arXiv preprint arXiv:2203.01479*.

Teerapittayanon, S., McDanel, B. e Kung, H. (2017). Distributed deep neural networks over the cloud, the edge and end devices. Em *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, p. 328–339.

Teerapittayanon, S., McDanel, B. e Kung, H.-T. (2016). Branchynet: Fast inference via early exiting from deep neural networks. Em *IEEE International Conference on Pattern Recognition (ICPR)*, p. 2464–2469.

Torres, V. A., Jaimes, B. R., Ribeiro, E. S., Braga, M. T., Shiguemori, E. H., Velho, H. F., Torres, L. C. e Braga, A. P. (2020). Combined weightless neural network FPGA architecture for deforestation surveillance and visual navigation of uavs. *Engineering Applications of Artificial Intelligence*, 87:103227.

Xue, M., Wu, H., Peng, G. e Wolter, K. (2021). DDPQN: An efficient dnn offloading strategy in local-edge-cloud collaborative environments. *IEEE Transactions on Services Computing*, 15(2):640–655.