
# PARTICIONAMENTO DE REDES NEURAIS PROFUNDAS COM SAÍDAS ANTECIPADAS

Roberto Gonçalves Pacheco

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientador: Rodrigo de Souza Couto

Rio de Janeiro
Setembro de 2020

PARTICIONAMENTO DE REDES NEURAIS PROFUNDAS COM SAÍDAS ANTECIPADAS

Roberto Gonçalves Pacheco

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Orientador: Rodrigo de Souza Couto

Aprovada por: Prof. Rodrigo de Souza Couto
Prof. Luís Henrique Maciel Kosmalski Costa
Prof. Cristiano Bonato Both

RIO DE JANEIRO, RJ – BRASIL
SETEMBRO DE 2020

Pacheco, Roberto Gonçalves

Particionamento de redes neurais profundas com saídas antecipadas/Roberto Gonçalves Pacheco. – Rio de Janeiro: UFRJ/COPPE, 2020.

XVII, 90 p.: il.; $29,7$cm.

Orientador: Rodrigo de Souza Couto

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2020.

Referências Bibliográficas: p. 84 – 90.

1. Redes Neurais Profundas. 2. Computação em Borda. 3. Particionamento de Redes Neurais Profundas. I. Couto, Rodrigo de Souza. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

*À minha mãe.*

# Agradecimentos

Primeiramente, agradeço à minha mãe, Leila dos Santos Gonçalves, meu maior exemplo de força e superação, pelo seu amor, confiança e por toda luta para que eu tenha todas as oportunidades que não lhe foram concedidas. Obrigado pelo investimento em minha educação, que mesmo em momento difíceis, sempre foi a sua prioridade. Espero te dar muito orgulho e felicidade nessa vida, porque desde a entrada na graduação, até a conclusão do mestrado, tudo só foi possível graças a você. "*In memorian*" ao meu pai, Roberto da Costa Pacheco, a vida nos separou cedo demais. A Nilton Rafael Machado pela presença em minha vida e por estar sendo um pai. À minha namorada, Nívea Ázara, por todo amor, carinho, apoio e por esse sorriso que me ensinou que a felicidade é real e alcançável. A todos os amigos que estiveram comigo nessa jornada. Em especial a Kevin Wetter, por todos os anos de amizade.

Agradeço aos amigos do Grupo de Teleinformática e Automação que tanto contribuíram com o ambiente de trabalho durante o desenvolvimento desta dissertação. Agradeço imensamente a todos os professores pela contribuição para a minha formação. Especialmente ao orientador, Rodrigo de Souza Couto, por compartilhar seus conhecimentos e embarcar nessa jornada desde a graduação. Obrigado pela colaboração durante a iniciação científica até o fim do mestrado e pelos futuros trabalhos. Uma menção especial aos Professores Luís Henrique Maciel Kosmalski Costa e Osvaldo Simeone pelas valiosas discussões e contribuições a esta dissertação. Eu também gostaria de agradecer aos professores Miguel Elias Mitre Campista, Otto Carlos Duarte e Pedro Braconnot Velloso. Aos membros da banca, por aceitarem analisar e discutir este trabalho. Agradeço também à secretaria do Programa de Engenharia Elétrica. À FAPERJ e CAPES pelo financiamento do projeto. Por fim, agradeço a todos aqueles que, de alguma forma, contribuíram para que este projeto se concretizasse.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

# PARTICIONAMENTO DE REDES NEURAIS PROFUNDAS COM SAÍDAS ANTECIPADAS

Roberto Gonçalves Pacheco

Setembro/2020

Orientador: Rodrigo de Souza Couto

Programa: Engenharia Elétrica

O processo de inferência em redes neurais profundas (*Deep Neural Networks* - DNNs) demanda uma alta capacidade computacional. Essa capacidade pode não estar disponível nos dispositivos finais, sendo necessário o uso de uma infraestrutura de computação em nuvem. No entanto, enviar dados brutos à nuvem pode aumentar o tempo de inferência, devido ao tempo de comunicação. Para reduzir esse tempo, as primeiras camadas neurais da DNN podem ser executadas em uma infraestrutura de computação na borda e o restante na nuvem. Dependendo de quais camadas são processadas na borda, isso pode reduzir a quantidade de dados enviada, mas pode também aumentar o tempo de processamento. Como o tempo de inferência é composto pelo tempo de comunicação e de processamento, é necessário lidar com esse compromisso. Problemas de particionamento de DNNs buscam resolver esse compromisso, escolhendo o conjunto de camadas a ser executado na borda para minimizar o tempo de inferência. Esta dissertação aborda o particionamento de DNNs com saídas antecipadas. Nesse tipo de DNN, o processo de inferência pode ser concluído nas camadas intermediárias, dependendo do nível de incerteza da classificação de uma amostra de entrada. Assim, além das condições da rede e do *hardware* da nuvem e da borda, características dos dados de entrada também podem influenciar a decisão de particionamento. Para considerar tais características, esta dissertação modela o problema de particionamento como um problema de caminho mais curto em um grafo e, portanto, resolvido em tempo polinomial. O modelo é usado como base para propor o sistema POPEX (*Partitioning OPtimization for deep neural networks with Early eXits*). Além disso, esta dissertação avalia como o modelo da rede neural e os dados de entrada alteram o particionamento ótimo em DNNs com saídas antecipadas. Em relação ao primeiro, considera-se o processo de calibração de DNNs, enquanto o segundo refere-se ao impacto da distorção da imagem no particionamento.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

PARTITIONING OF DEEP NEURAL NETWORKS WITH EARLY EXITS

Roberto Gonçalves Pacheco

September/2020

Advisor: Rodrigo de Souza Couto

Department: Electrical Engineering

Deep Neural Networks (DNNs) requires high computation power. This power may not be available on end devices, requiring the use of a cloud computing infrastructure. However, sending raw data to the cloud can increase the inference time, due to the communication time. To reduce this time, the first layers of DNN can be executed in a edge device and the remaining layers in the cloud. Depending on which layers are processed at the edge, this can reduce the amount of data sent, but can also increase processing time. As the inference time is composed of the communication and processing time, it is necessary to deal with this trade-off. Partitioning problems try to solve this trade-off, choosing a set of layers to be executed in the edge device to minimize the inference time. This dissertation addresses DNN partitioning with early exits. In this kind of DNN, the inference can be finished in the middle layers, depending on the level of uncertainty of the classification of an input sample. Therefore, besides of network conditions and cloud and edge hardware, input data characteristics can also influence the partitioning decision. To consider these characteristics, this disseration models the partitioning problem as a shortest path problem in a graph and, thus, can be solved in polynomial time. This model is used as the basis for proposing the POPEX (*Partitioning OPtimization for deep neural networks with Early eXits*) system. Moreover, this dissertation evaluates as the DNN model and input data can affect the DNN partitioning with early exits. Regarding the first, this considers the process of DNN calibration, while the second refers to image distortion in the partitioning.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Abreviações

**BLE** *Bluetooth Low Energy* - *Bluetooth* de Baixa Energia

**CNN** *Convolutional Neural Network* - Redes Neurais Convolucionais

**Conv** *camada Convolucional*

**COPPE** Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

**CPU** *Central Process Unit* - Unidade de Processamento Central

**DADS** *Dynamic Adaptive DNN Surgery*

**DAG** *Directed Acyclic Graph*

**DNN** *Deep Neural Network* - Redes Neural Profunda

**ECE** *Expected Calibration Error* - Valor Esperado do Erro de Calibração

**ERB** *Estações Rádio Base*

**FC** *camada Fully-Connected*

**FPGA** *Field Programming Gate Array*

**GPU** *Graphics Processing Unit* - Unidade de Processamento Gráfico

**HTTP** *HyperText Transfer Protocol*

**MLPs** *MultiLayer Perceptrons*

**NLL** *Negative Log-Likelihood*

**POPEX** *Partitioning OPtimization for deep neural networks with Early eXits*

**RTT** *Round Trip Time*

**VRAM** *Video Random Access Memory*

# Lista de Símbolos

$\alpha_i$ Tamanho, em *bytes*, do conjunto de saída da $i$-ésima camada neural da BranchyNet

$\boldsymbol{\tau}$ Vetor que contém as configurações de entropia de cada ramo lateral

$\boldsymbol{B}$ Histórico da taxa de envio

$\boldsymbol{p}(\boldsymbol{x})$ Vetor de probabilidade referente à amostra de entrada $\boldsymbol{x}$

$\boldsymbol{p}^{(i)}(\boldsymbol{x})$ Vetor de probabilidade do $i$-ésimo ramo lateral referente à amostra de entrada $\boldsymbol{x}$

$\boldsymbol{t_c}$ Vetor do tempo de processamento na nuvem

$\boldsymbol{t_e}$ Vetor do tempo de processamento na borda

$\boldsymbol{t}_{\text{net}}$ Vetor que contém os tempo de comunicação de cada camada da BranchyNet

$\boldsymbol{X}$ Vetor que contém as variáveis aleatórias $X_k$

$\boldsymbol{x}$ Amostra de entrada $\boldsymbol{x}$

$\emptyset$ Conjunto Vazio

$\eta$ Função entropia usando o vetor de probabilidade do i-ésimo ramo lateral

$\gamma$ Fator de processamento entre borda e nuvem

$\hat{p}$ Confiança de classificação

$\hat{q}$ Confiança de classificação bem calibrada

$\hat{y}$ Classe inferida

$\hat{y}^{(i)}$ Classe inferida pelo $i$-ésimo ramo lateral

$\mathbb{1}$ Função indicadora

$\mathbb{E}[T_{inf}(k)]$ Valor esperado do tempo de inferência com BranchyNet rom ramo lateral na posição $k$

$\mathbb{E}[T_{inf}]$ Valor Esperado do tempo de inferência

$\mathbb{N}_*$ Conjunto dos naturais positivos

$\mathbb{R}_{>}$ Conjunto dos reais positivos

$\mathcal{B}$ Conjunto dos ramos laterais da BranchyNet

$\mathcal{B}^*$ Conjunto que contém os ramos laterais capazes de maximizar a acurácia da BranchyNet

$\mathcal{C}$ Conjunto de classes predefinidas

$\mathcal{E}$ Conjunto que contém as arestas do grafo $\mathcal{G}$

$\mathcal{G}'^{*}_{\text{BDNN}}$ Grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ da BranchyNet com os ramos $\mathcal{B}^*$ que maximizam a acurácia

$\mathcal{G}'_{BDNN}$ Grafo de particionamento baseado na arquitetura da BranchyNet

$\mathcal{G}$ Grafo baseado na arquitetura da BranchyNet

$\mathcal{G}_{\text{BDNN}}$ Grafo da BranchyNet

$\mathcal{N}(v_i)$ Conjunto que contém os vértices vizinhos ao vértice $v_i$

$\mathcal{P}[\eta(\boldsymbol{p}^{(i)}) < \tau_i]$ Probabilidade de classificar uma amostra no $i$-ésimo ramo lateral

$\mathcal{T}$ Conjunto que contém as configurações de entropia

$\mathcal{V}'$ Conjunto que contém os vértices do grafo $\mathcal{G}$ pertencentes ao ramo principal

$\mathcal{V}$ Conjunto que contém os vértices do grafo $\mathcal{G}$

$\mathcal{V}_A$ Conjunto de vértices auxiliares

$\mathcal{V}_c$ Conjunto que contém os vértices do grafo $\mathcal{G}$ processados na nuvem

$\mathcal{V}_e$ Conjunto que contém os vértices do grafo $\mathcal{G}$ processados no dispositivo em borda

$\mathcal{X}$ Conjunto das amostras de entrada

$\mathcal{Y}$ Conjunto de classes predefinidas

$\omega_{(v_i,v_j)}$ Peso associado a aresta $(v_i, v_j)$ no grafo $\mathcal{G}_{\text{BDNN}}$ da BranchyNet

$\omega^{\text{BDNN}}_{(v_i,v_j)}$ Pesos associados às arestas em uma BranchyNet

$\pi^*$ Caminho mais curto entre os vértices *input* e *output*

$\sigma$ Função *softmax*

$\tau_i$ Limiar de entropia do $i$-ésimo do ramo lateral

$\theta$ Vetor de parâmetros do modelo da BranchyNet

$\varsigma$ Estratégia de particionamento

$\widehat{acc}$ Acurácia alvo

$Acc(B_m)$ Acurácia do *bin* $B_m$

$acc_i$ Acurácia do $i$-ésimo ramo lateral

$B(t_0)$ Taxa de envio no instante $t_0$

$b_i$ $i$-ésimo ramo lateral da BranchyNet

$Conf(B_m)$ Confiança do *bin* $B_m$

$d(v_i)$ Grau do vértice $v_i$

$e_{ij}$ Aresta entre os vértices $v_i$ e $v_j$

$f_\theta(\boldsymbol{x})$ Modelo da DNN com parâmetros $\theta$

$L(\hat{y}, y)$ Função de erro final da BranchyNet

$L_i(\hat{y}, y)$ Função de erro do $i$-ésimo ramo lateral

$p_k$ Probabilidade de classificar amostras no $k$-ésimo ramo lateral

$T$ Parâmetro do método de calibração *Temperature Scaling*

$t_{\text{input}}^{\text{net}}$ Tempo de comunicação para enviar a imagem bruta da borda à nuvem

$t_{\text{max}}$ Tempo de inferência máximo predefinido pela aplicação

$T_c$ Atraso total de processamento na nuvem

$T_e$ Atraso total de processamento na borda

$T_{inf}^{\text{DNN}}$ Tempo de inferência da DNN

$t_i^{\text{net}}$ Tempo de comunicação para enviar os dados da $i$-ésima da borda à nuvem

$t_i^e$ Tempo de processamento no dispositivo em borda da $i$-ésima camada neural

$t_i^e$ Tempo de processamento no nuvem da $i$-ésima camada neural

$t_s^{\text{net}}$ Tempo de comunicação para enviar os dados de saída da camada de particionamento da borda à nuvem

$v_i$ $i$-ésima camada neural do ramo principal

$v_i^{*c}$ Vértices auxiliar referente à $i$-ésima camada neural processada na nuvem

$v_i^{*e}$ Vértices auxiliar referente à $i$-ésima camada neural processada na borda

$v_s$ Camada de particionamento

$X_k$ Variável Aleatória de Bernoulli que indica se uma amostra é classificada no $k$-ésimo ramo lateral

# Capítulo 1

# Introdução

As Redes Neurais Profundas (*Deep Neural Networks* - DNNs) são técnicas de Inteligência Artificial, empregadas em diversos campos de pesquisa, especialmente em visão computacional [1–3], processamento de linguagem natural [4] e reconhecimento de padrões [5]. Como exemplo de aplicações de visão computacional, é possível citar assistência cognitiva e veículos inteligentes. No primeiro tipo de aplicação, os usuários utilizam um dispositivo vestível (p.ex., óculos ou relógios inteligentes) que os guiam em suas tarefas, como preparação de receitas culinárias, podendo até atuar como guia de pessoas cegas. Nos veículos inteligentes, câmeras e outros sensores auxiliares monitoram o ambiente ao seu redor. Assim, as DNNs executam reconhecimento de objetos e análise dos dados capturados, visando evitar acidentes e auxiliar o motorista na condução do veículo [6–8].

A arquitetura de uma DNN é composta de uma camada de entrada, uma sequência de camadas intermediárias e, por fim, uma camada de saída. Uma vez treinada para classificação de imagens, a DNN executa o algoritmo *feed-forward* para rotular a imagem em uma classe predefinida. Por exemplo, uma DNN treinada para reconhecer determinados objetos tem uma classe para cada possível objeto. No algoritmo *feed-forward*, cada camada recebe a saída da camada anterior, a processa e, em seguida, propaga seus dados de saída para a próxima camada. A DNN executa esse algoritmo desde a camada de entrada, percorrendo as camadas intermediárias até alcançar a camada de saída. Para problemas de classificação, a camada de saída calcula a probabilidade de a imagem pertencer a cada classe predefinida [9].

As DNNs podem ser implementadas nos próprios dispositivos finais, como *smartphones* e dispositivos vestíveis. Entretanto, tanto o treinamento quanto o processo de inferência das DNNs podem demandar alto poder computacional. Consequentemente, sua execução em dispositivos computacionalmente limitados pode introduzir um tempo de processamento proibitivo para determinadas aplicações. Nesses casos, a DNN pode ser executada na infraestrutura de computação em nuvem, que possui recursos computacionais responsáveis por acelerar o processamento, como as GPUs (*Graphics Processing Unit*) [10, 11]. Nessa abordagem, o dispositivo final envia os dados brutos para a nuvem

que, por sua vez, executa a inferência. Nota-se que essa abordagem introduz um tempo de comunicação entre o dispositivo final e a nuvem, que aumenta o tempo de inferência.

A introdução do tempo de comunicação representa um grave obstáculo, já que muitas aplicações nas quais DNNs são empregadas podem exigir alta responsividade [10]. Em outras palavras, essas aplicações demandam que a inferência da DNN seja executada o mais rápido possível para que, em seguida, ações sejam realizadas. Por exemplo, em aplicações de assistência cognitiva, o requisito de latência máxima pode ser de 100 ms [12], enquanto para veículos autônomos é de 300 ms [13]. Assim, para alcançar alta responsividade e viabilizar tais aplicações, é necessário reduzir os atrasos de processamento e comunicação. Por um lado, executar a DNN no dispositivo final aumenta o tempo de processamento, que está relacionado ao *hardware* utilizado. Por outro lado, a execução na nuvem aumenta o tempo de comunicação, dado o tempo necessário para enviar dados, por meio da Internet, do dispositivo final ao servidor em nuvem.

A computação na borda é uma alternativa para lidar com o tempo de comunicação imposto pela computação em nuvem [10]. Esse paradigma consiste em instalar recursos computacionais próximos aos dispositivos finais, reduzindo o tempo de comunicação. Os dispositivos de borda podem ser instalados em locais intermediários, como estações rádio base (ERBs) de celular e pontos de acesso Wi-Fi. Outro exemplo é executar parte da computação no próprio dispositivo final e outra parte na nuvem ou na infraestrutura intermediária [14]. De qualquer forma, a borda pode possuir capacidade computacional significativamente inferior em comparação ao servidor em nuvem, introduzindo tempo de processamento. Portanto, a computação em borda reduz o tempo de comunicação, mas ainda pode aumentar o tempo de processamento comparado à nuvem. Logo, em relação à responsividade, permanece o compromisso entre o tempo de comunicação e processamento.

Na literatura há diversas propostas para reduzir os atrasos relacionados à inferência de DNN. Para reduzir o tempo de processamento, há a possibilidade de classificar uma amostra de entrada já nas camadas intermediárias. Essas redes neurais são chamadas de DNNs com saída antecipada (*DNNs with early exits*), como é o caso das BranchyNets [15]. Nessas redes, ramos laterais são adicionados entre as camadas intermediárias. Esses ramos geram um vetor de probabilidade, da mesma forma que a camada de saída da DNN. A partir desse vetor, calcula-se a confiança de sua classificação. Caso a confiança atinja um determinado limiar, o processo de inferência se encerra. Caso contrário, prossegue para as próximas camadas. Entretanto, dependendo da configuração do limiar de confiança, classificar as amostras de entrada nas camadas intermediárias pode resultar em declínio da acurácia de classificação. Portanto, há um compromisso entre acurácia de classificação e tempo de inferência.

Uma outra estratégia de redução do tempo de inferência é utilizar o particionamento de DNNs, que consiste em processar as primeiras camadas neurais da DNN no dispositivo em

borda e as demais na nuvem. O particionamento de DNNs utiliza problemas de otimização para determinar a camada de particionamento, que divide a DNN em duas partes. Assim, as camadas anteriores à de particionamento são processadas na borda, e as camadas posteriores são processadas na nuvem. Esses problemas de otimização escolhem a camada de particionamento com um determinado objetivo, como reduzir o tempo de inferência, ou até mesmo economizar a energia do dispositivo em borda. A proposta de particionamento baseia-se no fato de que o tempo de comunicação para enviar dados de saída das camadas intermediárias é significativamente menor do que enviar a imagem bruta, reduzindo o tempo de comunicação em comparação à solução apenas na computação em nuvem [16]. Por um lado, o particionamento de DNN reduz o tempo de comunicação entre borda e nuvem. Por outro, ainda adiciona tempo de processamento, em razão da limitação computacional dos dispositivos em borda. Para contornar isso, e reduzir não só com o tempo de comunicação, mas também o de processamento, esta dissertação combina o particionamento de DNNs com a abordagem de DNNs com saída antecipadas. Assim, propõe-se uma estratégia de particionamento de BranchyNets. Para tal, formaliza-se um problema de otimização, cujo objetivo é encontrar um particionamento ótimo que minimize o tempo de inferência da BranchyNet. Como as propostas já existentes de particionamento de DNNs, o problema deste trabalho depende da taxa de envio e da capacidade computacional da nuvem e da borda. Entretanto, por considerar uma BranchyNet, o problema também depende dos limiares de confiança e de aspectos inerentes aos dados processados, como a probabilidade da entrada ser classificada em uma camada intermediária. Para formular o problema de otimização, este trabalho modela matematicamente o tempo de inferência da BranchyNet. Em seguida, para minimizar esse tempo, mostra-se a equivalência entre o particionamento de BranchyNets e o problema de encontrar o caminho mais curto em grafos. Isso permite que o problema proposto obtenha uma solução ótima global em tempo polinomial.

A partir do problema formulado, esta dissertação propõe o sistema POPEX (*Partitioning OPtimization for deep neural networks with Early eXits*). Em linhas gerais, esse sistema usa o problema de otimização para minimizar o tempo de inferência, mas considerando também a acurácia de classificação. Para tal, o sistema POPEX executa o problema de otimização para diversos limiares de confiança da BranchyNet. Assim, o sistema encontra uma estratégia de particionamento ótima que minimiza o tempo de inferência associada a cada limiar. Em seguida, seleciona-se a estratégia de particionamento ótima que maximiza a acurácia, desde que cumpra um requisito de tempo de inferência máximo predefinido pelo usuário. Uma vez que a estratégia de particionamento é selecionada, o dispositivo em borda processa até a camada de particionamento definida pela estratégia de particionamento. Em seguida, o sistema envia os dados de saída dessa camada e a informação da estratégia de particionamento escolhida à nuvem que, por sua vez, processa as demais camadas da DNN.

Além de propor o POPEX, esta dissertação avalia como a calibração de DNNs impacta o particionamento de DNNs com saídas antecipadas. Uma DNN com a confiança descalibrada superestima sua capacidade de classificar amostras e, como mostrado em [17], pode ter uma redução da sua acurácia. No caso das BranchyNets, esse problema pode ser ainda mais grave, já que a confiança da classificação é utilizada na decisão de classificação nos ramos. Para resolver esses problemas, é possível usar um método de calibração, como o *Temperature Scaling* [17]. Esta dissertação mostra que a calibração diminui o número de amostras classificadas antecipadamente, o que aumenta o tempo de inferência, mas melhora a acurácia da classificação [17].

Por fim, esta dissertação avalia o particionamento em um cenário real considerando um problema de classificação de imagens, no qual as distorções na imagem podem impactar a escolha do particionamento de BranchyNets e, consequentemente, o tempo de inferência. Por exemplo, quando DNNs são utilizadas para identificar objetos em imagens, a presença de distorções pode influenciar a quantidade mínima de camadas necessárias para a inferência. Uma imagem de alta qualidade pode atingir o nível desejado de confiança nas primeiras camadas da BranchyNet, enquanto imagens de baixa qualidade podem exigir o processamento de todas as camadas. Este trabalho mostra que imagens sem nenhuma distorção são classificadas em até 468,55 ms a menos do que imagens com alto nível de distorção, o que corresponde a uma redução de 35,33% do tempo de inferência. Além disso, esta dissertação mostra que, em uma BranchyNet, quanto mais próximos da camada de entrada os ramos laterais estão posicionados, mais a presença de distorções impacta a acurácia da inferência.

Esta dissertação está organizada da seguinte maneira. O Capítulo 2 apresenta os conceitos básicos de DNNs e sua evolução até a abordagem de BranchyNets. O Capítulo 3 apresenta uma revisão dos trabalhos relacionados sobre técnicas para acelerar a inferência em DNNs. O Capítulo 4 descreve a visão geral do sistema proposto e detalha cada cada componente do sistema. Os experimentos de avaliação do POPEX são mostrados no Capítulo 5. Em seguida, o Capítulo 6 analisa o problema de calibração em DNNs com saídas antecipadas e seu respectivo impacto nas decisões de particionamento. O Capítulo 7 avalia o impacto da distorção na imagem em relação à acurácia e no tempo de inferência. Além disso, apresenta experimentos para avaliar o efeito da distorção da imagem no particionamento, considerando um cenário real de processamento na borda. Por fim, o Capítulo 8 apresenta conclusões e sugere direções para trabalhos futuros.

# Capítulo 2

# Das Redes Neurais Profundas às BranchyNets

Este capítulo descreve os fundamentos das Redes Neurais Profundas (DNNs), com foco em Redes Neurais Convolucionais (CNNs - *Convolutional Neural Networks*). Em seguida, apresentam-se as DNNs com saídas antecipadas, focando o exemplo das BranchyNets.

## 2.1 Redes Neurais Profundas

O estudo das DNNs é uma subárea de aprendizado de máquina. Tais métodos visam permitir que o computador aprenda um determinado fenômeno, sem que seja necessariamente programado com um modelo específico. Assim, ao contrário de programas especialistas que modelam uma função matemática para descrever o comportamento de um dado fenômeno, os métodos de aprendizado de máquina visam aprender o comportamento do fenômeno a partir dos dados de entrada. Uma vantagem desses métodos é a capacidade de serem utilizados para descrever fenômenos distintos, ao invés de modelar uma função matemática para cada fenômeno específico [1].

As DNNs são modelos computacionais inspirados no sistema neural humano, que adquirem conhecimento através do aprendizado baseado em dados. O neurônio computacional recebe um dado de entrada $x_i$. Essas entradas são ponderadas pelos pesos sinápticos $\omega_i$ que armazenam as informações aprendidas. Em seguida, o neurônio aplica uma função não-linear na soma ponderada dos dados de entrada, gerando uma saída não-linear. Em resumo, o neurônio artificial gera uma saída não-linear a partir da soma ponderada a partir dos dados recebidos. Dessa forma, pesos diferentes geram saídas diferentes para a mesma entrada. Assim, o processo de aprendizado, também chamado de treinamento, consiste em ajustar esses pesos $\omega_i$, de modo que as saídas geradas descrevam um determinado fenômeno a partir dos dados de entrada fornecidos.

Nesta dissertação as DNNs são utilizadas em problemas de classificação. Assim, dada uma amostra de entrada, a saída da DNN é um vetor que contém a probabilidade de a amostra pertencer a cada uma das classes predefinidas. Em seguida, a DNN rotula a amostra de entrada com a classe que possui a maior probabilidade. Por exemplo, seja a amostra de entrada imagens de gatos ou cachorros, a DNN deve classificar se o animal presente na imagem é um gato ou um cachorro. Há diversas formas de treinar uma rede neural. A abordagem mais comum é por meio do aprendizado supervisionado, no qual a rede neural tem acesso tanto às amostras de entrada quanto ao seu rótulo correto. O principal objetivo do treinamento da DNN é determinar os pesos sinápticos que permitam rotular corretamente o maior número de amostras. Para isso, apresenta-se à DNN um conjunto de amostras de entrada e seus respectivos rótulos corretos. Em seguida, calcula-se uma função de perda, que consiste na diferença entre as predições fornecidas pela DNN e os rótulos corretos. Por fim, o objetivo é determinar os pesos sinápticos que minimizem essa função de perda. Para isso, utiliza-se o algoritmo *backpropagation* [18].

Inicialmente, os modelos de redes neurais eram constituídos de até três camadas neurais. Entretanto, ao longo dos anos, as arquiteturas de redes neurais vêm se tornando cada vez mais complexas e, consequentemente, também mais profundas (ou seja, com mais camadas). As redes neurais que possuem mais de três camadas neurais são denominadas *Deep Neural Networks* (DNNs). Atualmente, por exemplo, a arquitetura AlexNet (2012) é composta de 8 camadas, enquanto a ResNet (2016) pode possuir de 18, 34 a 152 camadas neurais. Entretanto, o aumento do número de camadas introduz atraso de processamento. Isso pode ser proibitivo para aplicações que necessitem de alta responsividade.

## 2.2 Redes Neurais Convolucionais

Uma Rede Neural Convolucional (CNN) é um tipo específico de DNN que possui camadas que realizam operações de convolução. As CNNs são utilizadas, principalmente, na área de visão computacional, em tarefas relacionadas à classificação de imagens [19, 20], detecção de objetos [21, 22] e segmentação de imagens [23, 24]. Dado o seu desempenho em tarefas relacionadas à classificação de imagens, as CNNs são utilizadas em aplicações como veículos autônomos [6–8].

As CNNs possuem uma estrutura hierárquica, composta de uma sequência alternada de camadas convolucionais (*Conv*), funções de ativação e camadas de subamostragem (*pooling*), terminando com camadas *fully-connected*. Utilizando operações de convolução, as camadas convolucionais são capazes de extrair atributos dos dados de entrada. Em seguida, a função de ativação (p.ex., *tangh* e *ReLU*) é responsável por inserir não-linearidade aos conjuntos de atributos extraídos pela camada convolucional anterior, o que possibilita aprender representações mais complexas dos dados. Então, as camadas de *pooling* reduzem a dimensionalidade dos dados. A partir do processamento desse conjunto de cama-

das, as CNNs são capazes de extrair automaticamente atributos dos dados, possibilitando descrever um determinado fenômeno sem necessidade de interação humana.

A Figura 2.1 apresenta a estrutura básica de uma arquitetura de CNNs composta apenas de duas camadas de convolução (Conv.) e de *pooling*, seguidas por três camadas *fully-connected* (FC). Na Figura 2.1, a CNN recebe uma imagem de 32x32 *pixels*. A primeira camada convolucional, que consiste em seis filtros de 32x32 *pixels* cada, produz um mapa de atributos, representando o conjunto de atributos extraídos por essa camada convolucional. Geralmente, após uma sequência de camada convolucional e *pooling*, há uma função de ativação, com intuito de conferir não-linearidade aos dados extraídos. A função de ativação após as camadas *pooling* não são representadas na Figura 2.1 por questão de concisão. Por fim, as camadas *fully-connected* utilizam os atributos extraídos pelas camadas anteriores para classificar as amostras de entrada.

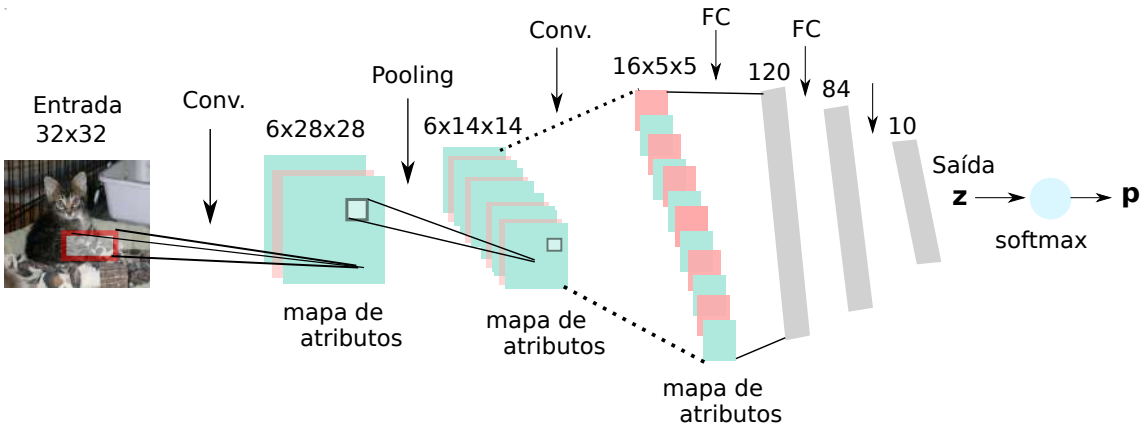



Figura 2.1: Arquitetura básica de uma Rede Neural Convolucional (CNN).

A estrutural sequencial e hierárquica das CNNs possibilita que cada camada convolucional não só extraia atributos, mas também os combine com os atributos da camada convolucional anterior. Essa combinação pode ser não-linear, permitindo representações mais complexas dos dados. Assim, os atributos das primeiras camadas convolucionais retratam representações mais simples dos dados, enquanto os atributos extraídos pelas últimas são mais complexos. Por exemplo, em um problema de classificação de imagens, a CNN recebe a imagem de um gato e, assim, a primeira camada convolucional consegue extrair atributos relacionados à presença de arestas. Assim, essa primeira camada convolucional consegue captar o contorno do animal. A segunda camada convolucional extrai atributos mais complexos e combina com os atributos extraídos na primeira camada, permitindo detectar características específicas da imagem, como, nesse exemplo, as orelhas pontiagudas do gato.

A seguir, detalha-se o funcionamento de cada uma das camadas neurais e a composição de cada uma das arquiteturas de DNNs utilizadas neste trabalho.

### 2.2.1 Camadas Convolucionais

Para realizar a operação de convolução, os filtros dividem a imagem em regiões menores conforme a sua dimensão. Em cada uma dessas sub-regiões, aplica-se uma convolução, extraindo um atributo local característico daquela sub-região. Em seguida, realiza-se o mesmo procedimento para a próxima região, até que percorra a região total da imagem, resultando, assim, no conjunto de atributos.

A convolução trata-se de uma operação linear, na qual os valores dos *pixels* de uma determinada sub-região da imagem são multiplicados por um conjunto de pesos pertencentes aos filtros da camada convolucional. A Figura 2.2 ilustra a operação de convolução entre uma imagem 4x4, cujos valores correspondem à intensidade dos *pixels*, e um filtro 3x3. Nessa figura, a imagem é representada como uma matriz $M$, de modo que cada elemento da matriz $m_{i,j}$ corresponde ao valor do pixel na $i$-ésima linha e $j$-ésima coluna. Para calcular a convolução na sub-região destacada na figura, calcula-se o produto entre os valores do pixel da imagem $m_{ij}$ em uma dada posição e o valor presente no filtro na mesma posição. Então, soma-se tais produtos da região em destaque. Em seguida, desliza-se o filtro por toda a dimensão da imagem, repetindo o mesmo procedimento descrito.

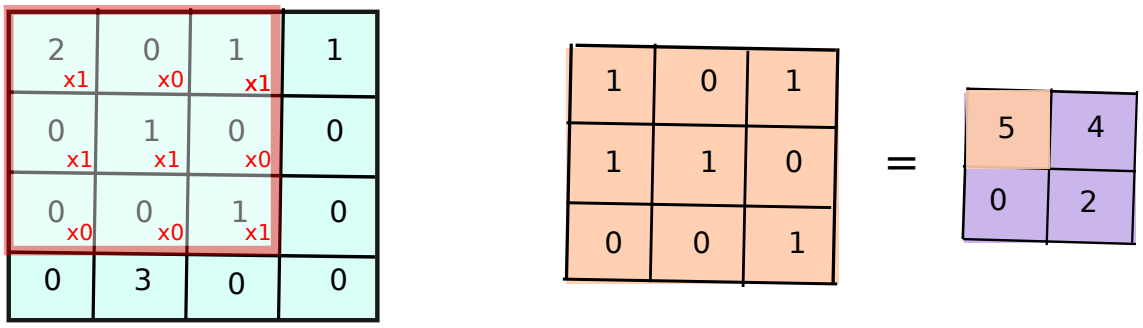



Figura 2.2: Funcionamento de uma camada convolucional.

Em uma CNN, os valores presentes nos filtros são obtidos pelo processo de aprendizado, para que extraiam um conjunto de atributos que permita detectar padrões e classificar as amostras de entrada. Portanto, esses filtros possuem diferentes valores para detectarem diferentes características nos dados de entrada, produzindo diferentes conjuntos de atributos.

### 2.2.2 Camadas de *pooling*

Na estrutura de uma CNN, as camadas de *pooling* são inseridas, geralmente, logo após uma camada convolucional. Essas camadas recebem o mapa de atributos extraído pela camada convolucional anterior. A função das camadas de *pooling* é filtrar as informações irrelevantes do conjunto de atributos recebido, mantendo apenas as informações signifi-

cativas para a detecção de padrões. Esse procedimento confere robustez ao conjunto de atributos e reduz a sensibilidade a ruídos [25].

As camadas de *pooling* substituem os valores de uma determinada região do mapa de atributos, extraído pela camada convolucional anterior, por alguma estatística. Essa substituição resume as informações contidas naquela região. A principal estatística utilizada na camada *pooling* é o operador *max*, cuja função é extrair o valor máximo de uma determinada região. Outras operações também utilizadas pelas camadas de *pooling* são a média, média ponderada e normal L2 [26].

A Figura 2.3 apresenta o funcionamento de uma camada de *pooling* com o operador *max*, denominada *max-pooling*. Nessa figura, a camada *max-pooling*, de tamanho 2x2, recebe um mapa de atributos de tamanho 4x4, e extrai o máximo valor de cada região de tamanho 2x2. É possível observar que o resultado produzido pela camada *max-pooling* reduz a dimensionalidade do mapa de atributos anterior, reduzindo o número de parâmetros a serem aprendidos pela camada convolucional posterior.

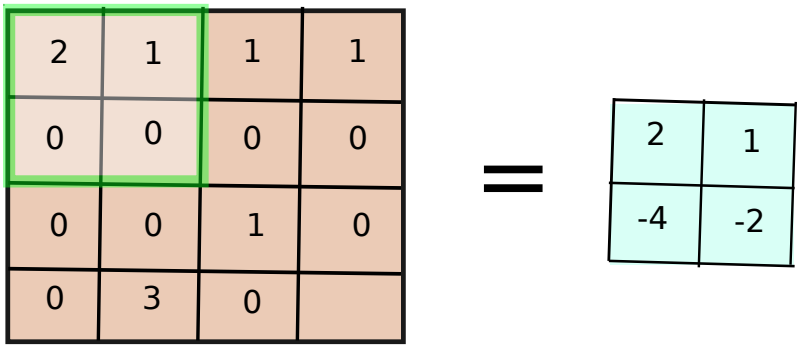



Figura 2.3: Representação da camada *max-pooling*.

### 2.2.3 Camadas Totalmente Conectadas

As arquiteturas das CNNs possuem uma ou mais camadas totalmente conectadas (*fully-connected*). Tanto nas camadas convolucionais quanto nas de *pooling*, os neurônios sempre estão relacionados a uma determinada região. Já as camadas *fully-connected* caracterizam-se por ter todos os neurônios interligados. Essa característica possibilita analisar globalmente os atributos extraídos pelas camadas anteriores, combinando-os de uma forma não-linear.

Na última camada *fully-connected*, denominada camada de saída, cada neurônio representa uma classe predefinida, utilizada para classificar a amostra de entrada. Portanto, o número de neurônios dessa camada corresponde à quantidade de classes do modelo. Dada uma amostra de entrada $\boldsymbol{x}$, o modelo da DNN gera um vetor de saída $\boldsymbol{z}$. Esse vetor é utilizado para obter um vetor de probabilidade $\boldsymbol{p}$, contendo a probabilidade de a amostra pertencer a cada uma das classes predefinidas. Em um problema de classificação com

$K > 2$ classes, o vetor de probabilidade $\boldsymbol{p}$ é obtido usando a função *softmax* $\sigma$ dada por

$$\boldsymbol{p}(\boldsymbol{x}) = \sigma(\boldsymbol{z}) = \frac{\exp(\boldsymbol{z_i})}{\sum_{c \in \mathcal{C}} \exp(z_c)}, \tag{2.1}$$

onde $\mathcal{C}$ é o conjunto das classes predefinidas e $\boldsymbol{z} \in \mathbb{R}^{|\mathcal{C}|}$ é o vetor de saída gerado pela camada de saída da DNN. A partir de $\boldsymbol{p}(\boldsymbol{x})$, obtém-se a classe inferida $\hat{y}$ pela DNN fazendo

$$\hat{y} = arg\, max_{\{1,\ldots,|\mathcal{C}|\}}(\boldsymbol{p}(\boldsymbol{x})). \tag{2.2}$$

### 2.2.4 Arquiteturas de Redes Neurais Convolucionais

A arquitetura das redes neurais define os tipos de camadas, a dimensão de cada um dos filtros, o número de filtros e as conexões entre camadas e neurônios [18]. As arquiteturas das CNNs são compostas, basicamente, por três tipos de camadas neurais: convolucionais, *pooling* e *fully-connected*, descritas anteriormente. Ao longo dos anos, diversas arquiteturas de CNNs foram propostas com objetivo de melhorar o desempenho em diversas aplicações. A seguir, apresenta-se as duas arquiteturas de CNNs utilizadas nesta dissertação.

#### AlexNet

A arquitetura da AlexNet [19] foi proposta com objetivo de classificar imagens e foi a primeira CNN a vencer o desafio ImageNet de 2012. A arquitetura básica da AlexNet é composta por oito camadas neurais, sendo cinco camadas convolucionais e *pooling*, e três camadas *fully-connected*. A Figura 2.4 ilustra de forma simplificada a arquitetura da AlexNet.

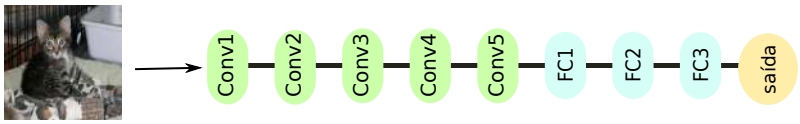



Figura 2.4: Representação da arquitetura da CNN AlexNet.

#### SqueezeNet

A arquitetura SqueezeNet [27] é proposta com objetivo de obter um desempenho próximo à AlexNet, mas com um número menor de parâmetros, cerca de 50 vezes menos do que a AlexNet. Essa redução do número de parâmetros possibilita a sua implementação em dispositivos embarcados, de baixo custo e FPGAs. Além disso, o menor número de parâmetros acelera o processo de treinamento [27].

A arquitetura da SqueezeNet é composta por dez camadas, sendo duas camadas convolucionais intercaladas por uma sequência de oito blocos *fire*, como apresentado na Figura 2.5. Os blocos *fire* são constituídos de três camadas convolucionais com filtros 1x1, ilustrado na Figura 2.6. Esses blocos são responsáveis por reduzir os parâmetros das CNNs, mas sem perdas significativas de acurácia.

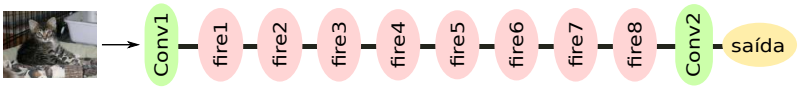



Figura 2.5: Representação da arquitetura da CNN SqueezeNet.

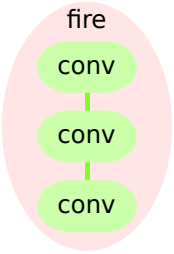


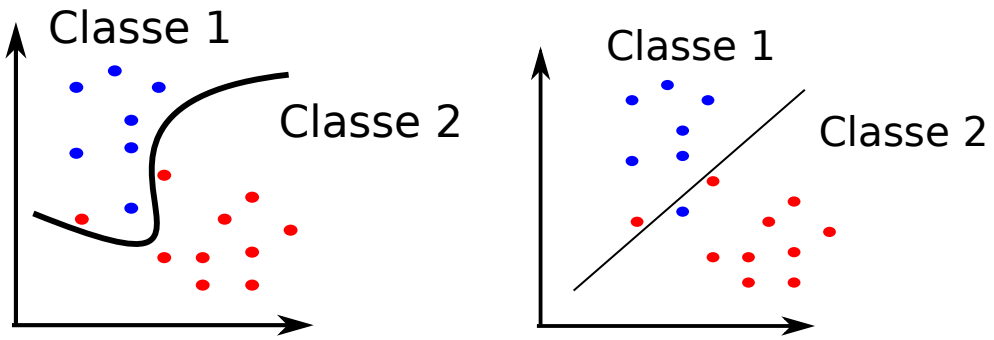

Figura 2.6: Bloco *fire* presente da SqueezeNet.

## 2.3 Redes Neurais com Saídas Antecipadas

As DNNs com saídas antecipadas são arquiteturas de redes neurais que possuem ramos laterais inseridos nas camadas intermediárias [28]. Esses ramos permitem classificar amostras sem que haja a necessidade de processar todas as camadas da DNN. Essas DNNs utilizam uma métrica de incerteza, como a entropia, para decidir se um determinado ramo lateral é capaz de classificar uma amostra. Caso essa classificação seja possível, o tempo de inferência é reduzido. Isso se torna ainda mais importante caso a DNN seja executada em um dispositivo de baixo custo.

Esta dissertação aborda a BranchyNet, um exemplo de DNN com saídas antecipadas. A proposta de BranchyNet baseia-se na ideia que os atributos extraídos nas primeiras camadas são capazes de classificar corretamente uma grande parcela de amostras de um conjunto de dados. Enquanto outras propostas, como Edgent [29], utilizam o conceito de DNNs com saídas antecipadas para ajustar o comprimento adequado da DNN para classificar todo o conjunto de dados, a BranchyNet utiliza uma métrica de confiança para identificar as amostras que podem ser classificadas nos ramos laterais. Em outras palavras, as BranchyNets consideram que somente uma pequena parcela de amostras do conjunto de dados é difícil de classificar e exige ser processada por todas as camadas da DNN. Essa ideia é ilustrada na Figura 2.7, na qual compara-se uma classificação binária, isto é, em duas classes, de um conjunto de dados realizado por uma abordagem tradicional e pelo

primeiro ramo de uma BranchyNet. A Figura 2.7(a) ilustra a abordagem de uma DNN tradicional, na qual é construída uma superfície de decisão complexa e não linear que é capaz de classificar todas as amostras do conjunto de dados em duas classes. Entretanto, nota-se que grande parte das amostras são distantes da superfície de decisão, o que significa que podem ser classificadas por estruturas mais simples, como o primeiro ramo de uma BranchyNet. Essa situação é mostrada na Figura 2.7(b). Entretanto, nota-se nessa figura que ainda há pontos classificados erroneamente pelo primeiro ponto. Por isso, utiliza-se a entropia para identificá-los e, então, processá-los também pelas próximas camadas.




(a) DNN Tradicional (b) Ramo Lateral da BranchyNet.

Figura 2.7: Comparação ilustrativa entre as superfícies de decisão de DNNs tradicionais, sem saídas antecipadas, e as DNNs com saídas antecipadas como BranchyNet.

A Figura 2.8 ilustra a estrutura geral de uma BranchyNet com $M$ ramos laterais. Nessa figura, as camadas $v_1, \cdots, v_N$ representam as camadas neurais do ramo principal, enquanto $b_1, \cdots, b_M$ representam as camadas do ramo lateral. Diferentemente das DNNs tradicionais que possuem somente uma saída, cada ramo lateral da BranchyNet é capaz de classificar as amostras, desde que a confiança de sua classificação atinja um determinado nível. Assim, cada ramo lateral possui uma camada *fully-connected* responsável por gerar um vetor de probabilidades, contendo a probabilidade de a amostra pertencer a cada uma das classes predefinidas. Assim, dada uma amostra $\boldsymbol{x}_i$, cada ramo lateral $i$ é capaz de gerar um vetor de probabilidade $\boldsymbol{p}^{(i)}(\boldsymbol{x})$. Assim, a classe inferida $\hat{y}^{(i)}$ por cada ramo $i$ é obtida usando a Equação 2.2.

A Figura 2.9 apresenta as arquiteturas de BranchyNets utilizadas neste trabalho. A Figura 2.9(a) mostra a arquitetura AlexNet, que possui dois ramos laterais inseridos nas duas primeiras camadas convolucionais do ramo principal. Já a Figura 2.9(b) apresenta a

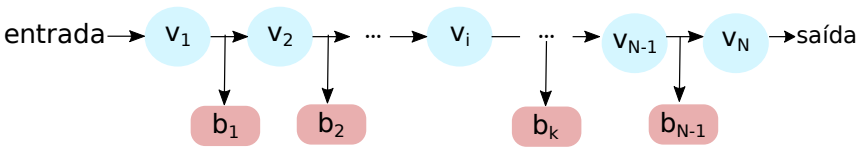
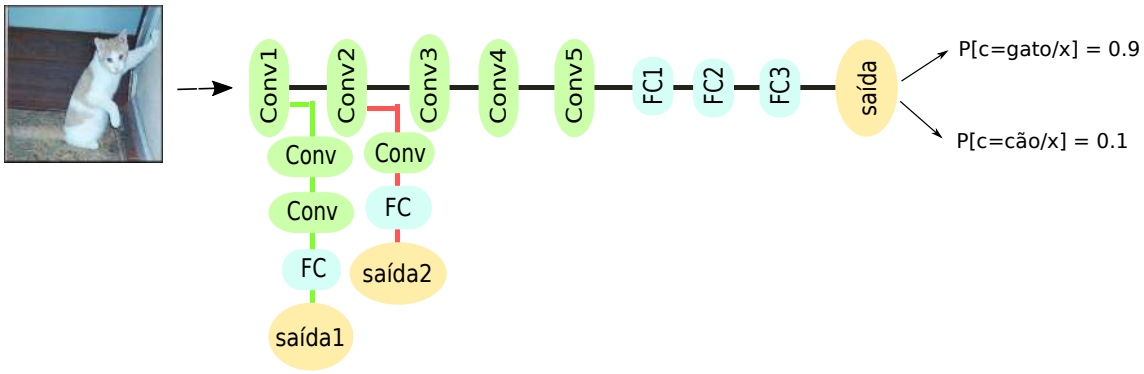
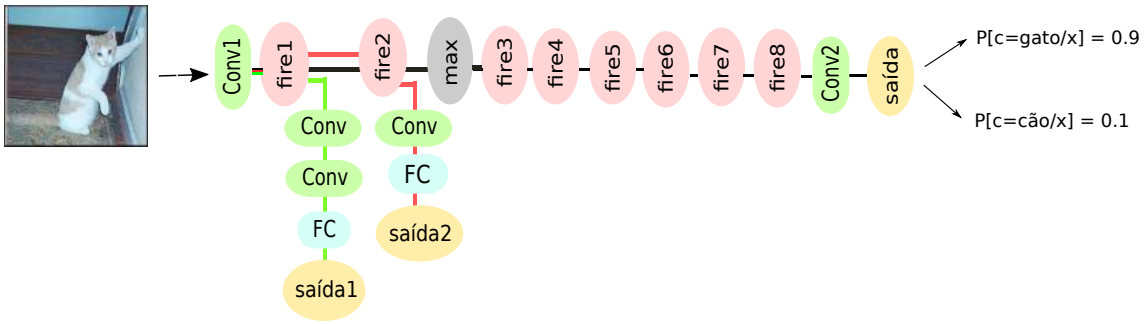



Figura 2.8: Ilustração da estrutura geral da DNNs com saídas antecipadas, como a BranchyNet.

arquitetura da B-SqueezeNet, que é uma BranchyNet criada a partir da SqueezeNet descrita na Seção 2.2.4. Nessa rede, dois ramos laterais são inseridos após os dois primeiros blocos *fire*.




(a) Uma B-AlexNet com dois ramos laterais inseridos no ramo principal.




(b) Uma B-SqueezeNet com dois ramos laterais inseridos no ramo principal.

Figura 2.9: Arquitetura das BranchyNets utilizadas neste trabalho.

Para treinar uma BranchyNet, define-se uma única função objetivo para toda BranchyNet, a partir da combinação das funções de erro de cada ramo lateral:

$$L(\hat{y}_i, t) = \sum_{i=1}^{M+1} \alpha_i L_i(\hat{y}_i, t), \tag{2.3}$$

onde $M + 1$ é o número de ramos da BranchyNet, considerando também o ramo principal, $t$ e $\hat{y}^{(i)}$ são, respectivamente, a classe verdadeira da amostra analisada e a classe inferida pelo $i$-ésimo ramo [30]. As constantes $\alpha_i$ da Equação 2.3 são hiper-parâmetros da BranchyNet relacionados com a importância de cada ramo lateral. Neste trabalho, todas as BranchyNets utilizadas possuem todos os ramos com $\alpha_i = 1$, o que indica que não há prioridade entre os ramos.

Uma vez treinada, A BranchyNet pode utilizar seus ramos laterais para classificar determinadas amostras antecipadamente. Assim, dada uma amostra de entrada $\boldsymbol{x}$, essa é processada pelas camadas até que um dos ramos laterais seja alcançado. Em um dado ramo $b_i$, sua camada *fully-connected* gera um vetor de probabilidades $\boldsymbol{p}^{(i)}(\boldsymbol{x})$, a partir do qual é calculado o nível de confiança da classificação usando a entropia $\eta(.)$ como a

métrica de incerteza, dada por

$$\eta(\boldsymbol{p}^{(i)}) = \frac{1}{|\mathcal{C}|} \sum_{c \in \mathcal{C}} \boldsymbol{p}_c^{(i)} \log(\boldsymbol{p}_c^{(i)}), \tag{2.4}$$

onde $\boldsymbol{p}^{(i)}$ é o vetor de probabilidade do ramo $b_i$ composto das probabilidades de a amostra pertencer a cada classe $c \in \mathcal{C}$, sendo $\mathcal{C}$ o conjunto de todas as classes predefinidas. Em seguida, é verificado se o valor da entropia $\eta(\boldsymbol{p}^{(i)})$ é inferior a um determinado limiar $\tau_i \in [0, 1]$ no ramo lateral $b_i$. Se sim, a classe inferida $\hat{y}$ será a classe do vetor $\hat{\boldsymbol{y}}^{(\boldsymbol{i})}$ referente ao ramo $b_i$ com a maior probabilidade, conforme a Equação 2.2. Dessa forma, essa amostra não é mais processada por nenhuma camada posterior, reduzindo o número de camadas processadas, o que diminui o tempo de processamento. Caso contrário, a amostra continua sendo processada pelas próximas camadas neurais intermediárias do ramo principal até alcançar o próximo ramo lateral. Se nenhum dos ramos laterais realizar a classificação, então a inferência termina quando a amostra alcança a camada de saída do ramo principal.

Na Figura 2.9(a) supondo que, por exemplo, o limiar de entropia do primeiro ramo seja $\tau_1 = 0,9$, então a grande maioria das amostras são classificadas no $i$-ésimo ramo lateral. Isso reduz o tempo de inferência, mas provoca queda da acurácia. Por outro lado, sendo o limiar $\tau_1 = 0,1$, menos amostras são classificadas no primeiro ramo lateral, o que aumenta o tempo de inferência e a acurácia de classificação. Portanto, há um compromisso entre acurácia e tempo de inferência em relação a diferentes configurações do limiar de entropia.

Se, por um lado, os ramos laterais podem acelerar a inferência classificando determinadas amostras antecipadamente nos ramos laterais, por outro lado, esses mesmos ramos podem introduzir atraso de processamento, quando a maioria das amostras do conjunto de dados não podem ser classificadas antecipadamente, pois não são confiáveis o suficiente. Isso ocorre pois, conforme o procedimento de inferência apresentado anteriormente, é sempre necessário processar o ramo lateral antes de processar as próximas camadas do ramo principal. Dessa forma, caso a maioria das amostras não seja classificada no ramo, o processamento do ramo é desnecessário, introduzindo somente atraso à inferência. Assim, para evitar processar as próximas camadas da DNN no dispositivo de borda, uma alternativa é transferir a tarefa de processar as próximas camadas na nuvem, particionando a BranchyNet. O particionamento da BranchyNet determina em qual camada o particionamento deve ocorrer de forma a minimizar o tempo de inferência. A decisão sobre em qual camada o particionamento deve ocorrer considera as condições da rede, como a taxa de envio para enviar os dados à nuvem, os recursos de processamento da nuvem e da borda e a probabilidade de classificar as amostras nos ramos laterais.

# Capítulo 3

# Trabalhos Relacionados

Diversas aplicações no campo de visão computacional demandam alta responsividade. Para isso, os trabalhos na literatura investigam como acelerar o processo de inferência. Tais propostas podem seguir três abordagens: aceleração por *hardware*, modificação estrutural da DNN ou utilização de uma infraestrutura computacional externa [29]. A primeira abordagem propõe equipar o dispositivo responsável por executar a DNN, seja um dispositivo em borda, seja um servidor em nuvem, com *hardware* que acelere o processo de inferência. Por exemplo, é possível empregar GPUs e FPGAs (*Field Programming Gate Arrays*) [31–33] para acelerar a inferência. A segunda modifica a estrutura da DNN com o objetivo de reduzir o número de camadas percorridas. Por fim, a terceira visa utilizar uma infraestrutura externa ao dispositivo final, para executar parcialmente ou completamente o processo de inferência. Este trabalho concentra-se na segunda e na terceira abordagens. Assim, as seções a seguir descrevem os trabalhos relacionados a cada uma.

## 3.1 Modificação Estrutural da DNN

Essa abordagem visa modificar estruturalmente a arquitetura da DNN, de forma que não haja perda significativa de acurácia, e ainda economize recursos computacionais de processamento. Dentro dessa abordagem, diversos trabalhos propõem técnicas de compressão do modelo da DNN removendo neurônios e camadas que contribuem pouco para a acurácia final do modelo. O objetivo é reduzir o consumo de energia e recursos computacionais, já que são realizadas menos operações durante a inferência [34–36].

Outra proposta de modificação da estrutura de DNNs é a inserção de saídas antecipadas (*DNNs with early exits*). Essa proposta consiste em inserir ramos laterais ao longo da estrutura da DNN, com objetivo de classificar, já nas camadas intermediárias, determinadas amostras de entrada. Conforme essa proposta, os ramos laterais geram um vetor de probabilidade composto da probabilidade de pertencer a cada classe predefinida, tal qual é gerado na camada de saída de uma DNN tradicional. A partir desse vetor de probabilidade,

calcula-se a confiança da classificação. Caso a confiança da classificação de um determinado ramo lateral atinja um limiar predefinido, o processo de inferência é finalizado nesse ramo e a amostra é classificada. Caso contrário, a amostra é processada pelas próximas camadas. Teerapittayanon *et al.* [15] propõe a BranchyNet, um exemplo de *Deep Neural Network with Early Exits*. Esse trabalho mostra que a inserção de ramos laterais permite reduzir o tempo de inferência, a partir da redução do tempo de processamento, pois uma grande parcela das amostras são classificadas nesses ramos laterais. Entretanto, a inserção de ramos laterais em todas as camadas pode resultar em um atraso de processamento, já que é necessário processar um ramo lateral e calcular a sua confiança a cada camada do ramo principal. Para contornar isso, Pandas *et al.* [37] propõe uma estratégia de posicionamento dos ramos laterais ao longo da arquitetura da DNN. Em outras palavras, a proposta determina se é vantajoso inserir um ramo lateral em cada uma das camadas do ramo principal. Para isso, calcula-se a eficiência de cada ramo, definindo eficiência como o produto entre o número de amostras classificadas corretamente e o tempo de processamento adicional causado pela inserção de cada ramo lateral. Por fim, caso a eficiência seja maior do que um dado limiar, então o ramo lateral pode ser inserido. Caso contrário, o ramo não é inserido na respectiva posição. DynExit [38] aplica o conceito de DNNs com ramos laterais em ResNets e a implementa em um *hardware* específico. Outros trabalhos propõem problemas de otimização para selecionar o tamanho da DNN com objetivo de reduzir o tempo de inferência [39], consumo de energia [40] ou até aumentar a taxa com que as amostras são classificadas em uma aplicação como, por exemplo, na análise de vídeo [41]. Kaya *et al.* [42] definem um problema recorrente em DNNs, chamado *overthinking*. Esse problema ocorre quando a DNN erra a classificação de uma determinada amostra na sua camada de saída, mas é capaz de classificar corretamente a mesma amostra nos ramos laterais posicionados nas camadas intermediárias. Esse problema causa perda de acurácia da DNN, dado que poderia ter classificado corretamente uma camada intermediária antes da camada de saída. Isso também resulta em um atraso de processamento desnecessário para processar todas as camadas neurais da DNN. O referido trabalho avalia as DNNs com saídas antecipadas como uma alternativa para reduzir esse problema.

Os trabalhos citados de DNNs com saídas antecipadas visam reduzir o tempo de inferência, em razão da redução do número de camadas neurais processadas durante a classificação de uma amostra. Assim, essas propostas reduzem, consequentemente, o tempo de processamento no dispositivo em borda. Contudo, caso uma amostra não seja classificada antecipadamente nos ramos laterais, o tempo de processamento, devido à capacidade de processamento dos dispositivos em borda, pode impedir a implementação de aplicações sensíveis ao atraso. Uma alternativa para contornar esse problema é utilizar uma infraestrutura computacional externa, como a nuvem. Esta dissertação, diferentemente desses trabalhos, combina tanto a estratégia de modificação estrutural da DNN, por meio do conceito de DNNs com saídas antecipadas (p.ex., BranchyNets), como também por meio da

utilização de uma estrutura externa, a qual é detalhada a seguir.

## 3.2 Utilização de uma Infraestrutura Computacional externa

A abordagem de utilizar uma infraestrutura externa [29] consiste em transferir a tarefa de processar as camadas neurais da DNN a uma infraestrutura computacional externa. Essa infraestrutura, seja um dispositivo de borda ou um servidor em nuvem, deve estar equipada com recursos computacionais para acelerar o tempo de inferência. Essa abordagem pode ser realizada por processamento exclusivamente na nuvem, exclusivamente na borda ou com particionamento.

O processamento exclusivamente na nuvem refere-se à estratégia de processar todas as camadas da DNN apenas na nuvem, a qual é equipada com recursos computacionais necessários para acelerar a inferência, como GPUs [10, 11]. Dessa forma, o dispositivo na borda da Internet captura os dados de entrada (isto é, imagens, vídeo ou áudio) e os envia à nuvem, que então executa a inferência da DNN. A principal desvantagem desse cenário é transferir uma grande quantidade de dados da borda da rede à nuvem. Assim, esse cenário introduz um tempo de comunicação para enviar os dados do dispositivo final à nuvem. Consequentemente, o tempo de inferência depende das condições da infraestrutura de rede, o que também pode inviabilizar aplicações sensíveis ao atraso.

O processamento exclusivamente na borda consiste em processar todas as camadas da DNN no dispositivo em borda, que pode ser instalado, por exemplo, em pontos de acesso Wi-Fi e ERBs [43, 44]. Apesar de a borda ser próxima ao dispositivo final e reduzir o tempo de comunicação, as limitações de recursos computacionais do dispositivo em borda introduzem atraso de processamento em comparação à nuvem. Isso impede o funcionamento de aplicações sensíveis ao atraso, como detecção de pedestres [45, 46] e detecção dos limites da estrada [47] em aplicações de carros inteligentes. Por um lado, esse cenário permite alocar poder de processamento na borda da rede, reduzindo o atraso de comunicação. Por outro lado, esses dispositivos também podem causar atrasos de processamento, já que possuem poder de processamento significativamente inferior em comparação com um servidor em nuvem [16]. Portanto, há um compromisso evidente entre o tempo de comunicação e o tempo de processamento nos cenários de processamento exclusivamente na borda e na nuvem. Vigil [48] é um exemplo de um sistema de visão computacional dentro do cenário de processamento exclusivamente na borda. Vigil consiste em uma rede de câmeras de segurança equipadas de módulo de comunicação sem fio que capturam imagens do ambiente e as enviam aos dispositivos em borda. Em seguida, esses dispositivos em borda selecionam *frames* que permitam contar o número de clientes em uma fila ou até mesmo identificar pessoas. Nessa proposta, o processamento exclusivamente na borda

reduz o consumo de banda, além de prover mais privacidade, em comparação com o cenário de processamento exclusivamente na nuvem. Além disso, os assistentes de voz, como a Siri [1] e a Alexa [2], são exemplos de dispositivos que realizam processamento de linguagem natural, tanto exclusivamente na nuvem quando somente na borda. Esses dispositivos processam localmente a tarefa de detecção de palavras específicas, como "Alexa" ou "*Hey Siri*", para iniciar o funcionamento dos assistentes de voz [49]. Uma vez que essas palavras são identificadas, a gravação de voz para as demais buscas é enviada à nuvem. Essa infraestrutura, por sua vez, executa a aplicação de processamento de linguagem natural para identificar a mensagem de voz, interpretá-la e responder a requisição.

O processamento com particionamento permite dividir a estrutura da DNN em duas partes. Nesse cenário, a primeira parte da DNN é processada localmente, enquanto a outra é executada remotamente no servidor em nuvem. Dessa forma, a decisão de como dividir as camadas da rede neural lida com o compromisso entre o tempo de processamento, tanto o do dispositivo final quanto o da borda, e o de comunicação decorrentes da infraestrutura de rede. Diversos estudos já se dedicaram a investigar o problema de transferência de processamento de um dispositivo móvel a um servidor em nuvem [50, 51]. Entretanto, essas propostas tratam de transferência de tarefas de processamento de forma genérica. Portanto, não contemplam as especificidades de DNNs, já que o seu particionamento depende da sua estrutura em camadas. Recentemente, os trabalhos começaram a investigar, especificamente, o particionamento de DNNs. Neurosurgeon [11] objetiva particionar uma DNN entre um dispositivo em borda e um servidor em nuvem. Para isso, esse trabalho modela a DNN como um grafo, no qual os vértices são as camadas e as arestas representam a comunicação entre as camadas. No primeiro momento, NeuroSurgeon constrói modelos preditivos para estimar o tempo de processamento no dispositivo em borda e no servidor em nuvem. Em seguida, esses modelos preditivos são combinados com a taxa de envio da infraestrutura de rede sem-fio, para dinamicamente selecionar a melhor partição que minimiza o tempo de inferência. Contudo, NeuroSurgeon limita-se a grafo com topologia linear. Para contornar essa limitação, DADS (*Dynamic Adaptive DNN Surgery*) [16] encontra a partição ótima para DNNs, cuja topologia seja tanto grafos lineares como grafos acíclicos direcionados (DAGs). Para isso, DADS trata o problema de particionamento em grafo como um problema de corte mínimo. No entanto, as propostas de NeuroSurgeon e DADS não abordam o tema de particionamento de DNNs com saídas antecipadas. Dessa forma, o tempo de inferência desses trabalhos não modela a possibilidade de a amostra ser classificada já nos ramos laterais. Ainda dentro da abordagem de particionamento de DNNs entre uma infraestrutura de borda e a nuvem, Ko *et al.* [52] propõem um particionamento fixo, de modo que as camadas convolucionais sejam posicionadas no dispositivo em borda, enquanto as camadas *fully-connected* sejam posicionadas na nuvem. Assim, o

[1] `https://www.apple.com/br/siri/`

[2] `https://developer.amazon.com/pt-BR/alexa`

dispositivo de borda é responsável por extrair os atributos dos dados e a nuvem é responsável por classificá-los. O trabalho propõe essa estratégia de particionamento para lidar com o compromisso existente entre o consumo energético do dispositivo em borda e a taxa de transferência de dados da borda à nuvem. Esse trabalho mostra que, por um lado, particionar a DNN em camadas mais profundas resulta em um maior consumo de energia por parte do dispositivo em borda. Por outro lado, particionar a DNN em camadas mais próximas da camada de entrada resulta em sobrecarga da transmissão de dados da borda à nuvem. Além disso, o trabalho mostra que, em geral, os dados de saída das camadas mais próximas da camada de saída são mais esparsos, ou seja, os dados contem maior número de zeros, o que permite maior compressão dos dados. Então, em seguida, o trabalho propõe comprimir os dados de saída das camadas antes de transmiti-los à nuvem, para reduzir o tempo de comunicação entre a borda e a nuvem. Por fim, Ko *et al.* utilizam a compressão JPEG com diferentes valores de fator de qualidade da compressão para avaliar a proposta.

Diferentemente dos trabalhos citados anteriormente, DeepWear particiona a DNN entre um dispositivo vestível (p.ex., como *smartwatch* e *smartglass*) e o dispositivo móvel, como um *smartphone*, conectado a ele. O dispositivo móvel, dessa forma, é visto como a infraestrutura de borda. Logo, DeepWear divide a DNN em duas partes, de modo que a primeira é processada no dispositivo vestível, enquanto a segunda é processada no dispositivo móvel. A comunicação entre o dispositivo vestível e o *smartphone* pode ser realizada por uma conexão local sem fio de baixo custo e baixo consumo energético, como *Bluetooth* ou *Bluetooth Low Energy* (BLE). Assim, o DeepWear não demanda uma conexão de Internet, como os trabalhos citados anteriormente. Dentre todos os particionamentos possíveis, essa proposta executa uma busca por força bruta para encontrar aquele que minimiza o tempo e inferência ou o consumo de energia.

Nenhum dos trabalhos citados anteriormente aborda o problema de particionamento em DNNs com saídas antecipadas e suas especifidades, como a probabilidade de classificar determinadas amostras nos ramos laterais localizados nas camadas intermediárias. Por outro lado, esta dissertação aborda o problema de particionamento em DNNs com saídas antecipadas (p.ex., BranchyNet). Para isso, este trabalho modela o tempo de inferência, considerando a probabilidade de classificar amostras nos ramos laterais. Em seguida, apresenta um método de particionamento dessas DNNs, visando minimizar o tempo de inferência. O método de particionamento é baseado na modelagem da DNN como um grafo, que converte o problema de particionamento em grafo em um problema de caminho mais curto. Isso possibilita determinar a partição ótima com complexidade computacional em tempo polinomial.

Recentemente, a literatura já começa a abordar, especificamente, o problema de particionamento de DNNs com saídas antecipadas. Teerapittayanon *et al.* [53] propõem uma DDNN (*Distributed Deep Neural Network*) que consiste em uma arquitetura hierárquica, constituída de um nível de dispositivos finais geograficamente distribuídos, outro nível de

dispositivos de borda e uma infraestrutura de computação em nuvem. Esse trabalho propõe distribuir a estrutura de uma BranchyNet ao longo dos níveis da DDNN, de modo que cada nível dessa arquitetura seja capaz de classificar a amostra de entrada. Por exemplo, um dispositivo em borda processa as camadas da BranchyNet até o primeiro ramo lateral. Caso o nível de confiança da classificação não seja atingido, o dispositivo em borda a envia para o próximo nível da infraestrutura da DNN, o qual pode ser um outro nível de dispositivo em borda ou mesmo a própria nuvem. Porém, nessa proposta, o particionamento é estático, pois a divisão das camadas da BranchyNet, nos níveis da arquitetura DDNN, é predefinida. Esta dissertação, por outro lado, propõe um método aplicável em cenários dinâmicos, que pode ser executado a cada mudança na taxa de envio entre a borda e a nuvem e na probabilidade de classificação das amostras em cada um dos ramos laterais.

Li *et al.* propõem o Edgent [29], que é um *framework* que particiona uma DNN entre uma infraestrutura de computação em borda e nuvem. O Edgent determina um particionamento com objetivo de maximizar a acurácia, enquanto cumpre um critério de latência predefinido. Essa proposta utiliza o conceito de DNNs com ramos laterais para escolher a profundidade da DNN que permite encontrar uma partição que cumpra o critério de latência. E, para maximizar a acurácia, Edgent realiza uma busca exaustiva em todos os ramos laterais, iniciando a partir do ramo mais profundo (isto é, mais próximo da camada de saída). A ordem dessa busca exaustiva entre os ramos laterais baseia-se na premissa de que a acurácia do ramo lateral aumenta à medida que o ramo está mais próximo da camada de saída. Em um dado ramo lateral, que possui uma determinada acurácia, o *framework* Edgent determina o particionamento que minimiza o tempo de inferência, usando uma busca exaustiva dentre todas as possíveis formas de particionar a DNN para aquele dado ramo lateral. Caso o tempo de inferência da partição escolhida seja inferior ao critério de latência predefinido, o Edgent ajusta a profundidade da DNN até o ramo lateral em análise e executa o particionamento determinado pela busca anterior. Caso contrário, o Edgent repete o mesmo procedimento em relação ao ramo lateral anterior ao analisado, seguindo a ordem de ramos laterais estabelecida. Assim, uma vez que esse *framework* determina a profundidade necessária para cumprir o critério de latência, as amostras não podem ser classificadas em quaisquer outros ramos laterais, mesmo que os ramos laterais anteriores atinjam altos níveis de confiança em suas classificações. Consequentemente, o Edgent não calcula o nível de confiança da classificação de cada ramo, usando uma métrica de incerteza, e nem há a probabilidade de classificação nos ramos laterais que não seja a determinada pelo próprio *framework*. Portanto, nota-se uma diferença entre a utilização dos ramos laterais no Edgent e a proposta de DNNs com saídas antecipadas apresentada anteriormente.

Diferentemente do *framework* Edgent, esta dissertação propõe o sistema POPEX, que particiona a DNN entre a borda e a nuvem, utilizando o conceito de DNNs com saídas antecipadas. Mais especificamente utiliza-se uma DNN do tipo BranchyNet, proposto por

Teerapittayanon *et al.* [15], como descrito anteriormente. Portanto, o sistema POPEX utiliza o nível de confiança da classificação de cada ramo, baseado em uma métrica de incerteza, como a entropia, para determinar se uma amostra pode ser classificada em um determinado ramo lateral. Além disso, esta dissertação modela o valor esperado do tempo de inferência, considerando a probabilidade de classificar amostras em cada um dos ramos laterais, em função do limiar de entropia definido. Modela-se a estrutura da BranchyNet como um grafo direcionado acíclico. Assim como Edgent, o sistema POPEX determina o particionamento visando maximizar a acurácia, enquanto cumpre também um critério de latência predefinido. Para determinar a camada de particionamento ótima, o sistema POPEX, primeiramente, realiza uma busca exaustiva entre cada ramo lateral da BranchyNet. Assim, a partir do ramo lateral mais próximo da camada de entrada da BranchyNet, o sistema POPEX remove todos os ramos que possuem acurácia inferior ao primeiro ramo. Uma vez que esses ramos são removidos, o sistema POPEX encontra o particionamento que minimiza o valor esperado do tempo de inferência, utilizando um problema de caminhos mais curtos. Em seguida, o sistema POPEX repete o mesmo procedimento para o segundo ramo lateral, removendo todos aqueles ramos que possuem acurácia inferior ao segundo ramo.

Após repetir a busca exaustiva para todos os ramos laterais, o sistema POPEX possui uma sequência de estratégias de particionamento, da qual se escolhe aquela com a máxima acurácia. Dessa forma, o sistema consegue percorrer todas as combinações de ramos laterais existentes, reduzindo o espaço de busca, em comparação a uma busca exaustiva simples como a proposta pelo *framework* Edgent. No POPEX, a busca exaustiva é possível, devido à minimização do tempo de inferência em tempo polinomial, usando o algoritmo Dijkstra, já que o POPEX modela o problema de particionamento como um problema de caminho mais curto. Além disso, o sistema POPEX não utiliza a premissa de que a acurácia dos ramos laterais aumenta conforme a profundidade da posição do ramo lateral, tornando a solução mais genérica para diferentes DNNs com saídas antecipadas. Em seguida, o sistema POPEX executa o procedimento anterior para encontrar a estratégia de particionamento para cada configuração do limiar de entropia. Então, basta selecionar aquela com a maior acurácia, e que ainda cumpre o requisito de latência predefinido, o que é detalhado na Seção 4.3. A seguir, esta dissertação apresenta o sistema POPEX, que encontra o particionamento ótimo para DNNs com saídas antecipadas.

# Capítulo 4

# POPEX: *Partitioning OPtimization for deep neural networks with Early eXits*

Esta dissertação propõe o POPEX [1] (*Partitioning OPtimization for deep neural networks with Early eXits*). Esse sistema determina o particionamento ótimo em DNNs com saídas antecipadas, como as BranchyNets. Esta dissertação formula um problema de otimização, cujo objetivo é encontrar o particionamento ótimo. Para isso, o problema de otimização escolhe os particionamentos que gerem as melhores combinações entre acurácia e tempo de inferência. Em seguida, seleciona-se o particionamento que maximiza a acurácia baseado na latência máxima predefinida pela aplicação.

Uma vez que o POPEX determina o particionamento ótimo, o sistema divide a arquitetura das DNNs com saídas antecipadas em duas partes. A primeira parte é processada localmente, enquanto a outra é executada em uma infraestrutura externa. O sistema POPEX permite particionar DNNs com saídas antecipadas entre um dispositivo em borda e a nuvem, entre um dispositivo final e a borda, ou entre um dispositivo final e a nuvem. Este trabalho considera que o particionamento ocorre entre a borda e a nuvem.

Diversos fatores impactam a decisão do particionamento do POPEX. Como exemplo, é possível citar fatores relacionados à capacidade de processamento dos dispositivos da borda e da nuvem e fatores ligados às condições da infraestrutura de rede, como a taxa de envio de dados. Além disso, em DNNs com saídas antecipadas, a probabilidade de classificar as amostras nos ramos laterais também afeta a decisão de particionamento, impactando tanto o tempo de inferência quanto a acurácia.

As BranchyNets utilizam o limiar de entropia em cada ramo lateral para decidir se uma determinada amostra pode ser classificada antecipadamente nesse ramo. Esses limiares de entropia são hiper-parâmetros do modelo da BranchyNet, como visto no Capítulo 2.3. Nesse sentido, quando o limiar de entropia de um determinado ramo lateral é ajustado para valores altos, como 0,9, a maioria das amostras que chegam a esse ramo podem ser

[1] Disponível em `https://github.com/pachecobeto95/POPEX`, registrado no INPI sob o número BR512020001239-6

classificadas nele, aumentando a probabilidade de classificação das amostras nesse ramo lateral. Por outro lado, quando o limiar de um ramo é ajustado para valores baixos, como 0,1, somente amostras com alto nível de confiança podem ser classificadas, o que diminui a probabilidade de classificação nesse ramo. Portanto, a configuração do limiar de entropia em cada ramo lateral altera a probabilidade de classificação das amostras nos ramos laterais, alterando, consequentemente, a decisão de particionamento, o tempo de inferência e a acurácia.

Além da configuração dos limiares de entropia, fatores intrínsecos aos dados de entrada, como a distorção da imagem, podem modificar a decisão de particionamento. O Capítulo 7 aborda como a qualidade da imagem pode alterar essa decisão. Em resumo, a decisão de particionamento da BranchyNet depende não só do poder de processamento dos dispositivos e da taxa de envio entre o dispositivo local (p.ex., borda) e o remoto (p.ex., nuvem), mas também da configuração do limiar de entropia em cada ramo e de fatores intrínsecos às amostras de entrada, como a distorção da imagem. Assim, o sistema proposto deve ser capaz de extrair tais parâmetros, sejam estáticos, como aqueles relacionados aos recursos computacionais da borda e da nuvem, ou dinâmicos, como taxa de envio. Em seguida, o sistema deve utilizá-los como parâmetros de entrada em um problema de otimização que, por sua vez, determina o particionamento que equilibre o compromisso entre acurácia e tempo de inferência em BranchyNet. Em seguida, seleciona-se o particionamento ótimo que maximiza a acurácia, baseado no requisito de latência máxima necessário para a aplicação. De maneira resumida, as principais funcionalidades do sistema POPEX são:

- ***Adaptação ao requisito de latência*:** cada aplicação exige um tempo de inferência máximo. Por exemplo, aplicações de assistência cognitiva exigem até 100 ms [12], enquanto aplicações de prevenção de colisão necessitam de até 300 ms [13]. O sistema deve então se adaptar ao requisito de tempo de inferência, mesmo em aplicações que exijam alta responsividade;

- ***Extração de parâmetros*:** o sistema POPEX é capaz de extrair todos os parâmetros necessários que impactam no problema de particionamento, sejam parâmetros estáticos, que são relacionados aos atrasos de processamento na borda e na nuvem, sejam parâmetros dinâmicos, como a taxa de envio;

- ***Monitoramento das condições da infraestrutura de rede*:** o POPEX monitora periodicamente a taxa de envio da rede, para coletar a taxa de envio atual. Com base nessa informação, o POPEX calcula o tempo de comunicação para enviar os dados referentes à camada de particionamento da borda para a nuvem;

  Para isso, o problema de otimização escolhe os particionamentos que gerem as melhores combinações entre acurácia e tempo de inferência. Em seguida, seleciona-se

o particionamento que maximiza a acurácia baseado na latência máxima predefinida pela aplicação.

- ***Otimização multi-objetivo entre acurácia e tempo de inferência*:** o sistema encontra os particionamentos da BranchyNet que melhor combinem acurácia e tempo de inferência. Em seguida, seleciona-se o particionamento ótimo que maximiza a acurácia baseado no critério de latência predefinido pela aplicação. Assim, esse sistema não compromete a acurácia na redução do tempo de inferência;

- ***Implementação de uma interface de comunicação*:** o sistema POPEX implementa uma interface de comunicação entre o dispositivo local (borda) e a infraestrutura externa (nuvem). Assim, o dispositivo local é capaz de enviar dados de saída da camada de particionamento à nuvem, dentre outras informações.

A arquitetura do sistema POPEX, ilustrada na Figura 4.1, é dividida em três componentes: *Background*, *Update Box* e *Decision Maker*. No primeiro momento, a aplicação fornece duas informações ao sistema POPEX: o tempo de inferência máximo e a arquitetura da BranchyNet empregada. Esta é inserida no componente *Background*, enquanto o tempo é inserido no *Decision Maker*. O tempo de inferência máximo, indicado por $t_{\max}$ na Figura 4.1, refere-se ao tempo máximo permitido para classificar uma amostra de entrada. O componente *App*, apresentado na Figura 4.1, refere-se a aplicação que no contexto desta dissertação está implementada em um dispositivo em borda, mas também poderia ser implementada no dispositivo final.

A partir do recebimento da arquitetura da DNN, o sistema POPEX executa o componente *Background*, que é responsável pela tarefa de extração dos parâmetros estáticos necessários para que o problema de otimização, executado pelo componente *Decision Maker*, consiga determinar o particionamento ótimo. Esses parâmetros estáticos são, por exemplo, o tempo de processamento de cada camada no dispositivo em borda e na nuvem. A extração dos parâmetros estáticos é executada apenas uma vez, durante a inicialização do sistema, pelo componente *Background*. Assim, a etapa de *Background* não adiciona atraso à decisão de particionamento. Dada a arquitetura da BranchyNet empregada pela aplicação, o componente *Background* extrai o tempo de processamento para executar cada uma das camadas da BranchyNet no dispositivo de borda e nuvem, além do tamanho em *bytes* dos dados de saída de cada uma das camadas da BranchyNet. Por fim, o componente *Background* modela a arquitetura da BranchyNet, fornecida pela aplicação, em um grafo. Assim, modela-se o problema de particionamento de DNNs como um particionamento de grafos. Uma vez que os parâmetros estáticos são extraídos, o componente *Background* fornece-os ao componente *Decision Maker*, como ilustrado na Figura 4.1. O componente *Background* é apresentado detalhadamente no Capítulo 4.1.

Como visto na Figura 4.1, o componente *Decision Maker* é o elemento central do POPEX. A partir dos parâmetros extraídos pelo *Background*, o *Decision Maker* executa a

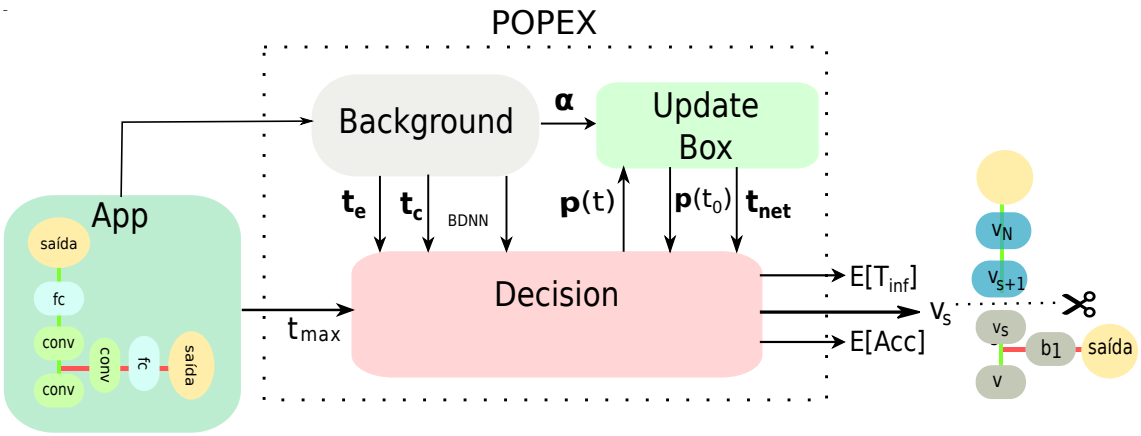



Figura 4.1: Visão geral do sistema POPEX.

tarefa de determinar o particionamento ótimo e enviar à nuvem essa decisão e os dados de saída da camada de particionamento. Para isso, o *Decision Maker* utiliza os parâmetros extraídos para executar o problema de otimização a fim de obter o particionamento ótimo. Em seguida, utiliza a interface de comunicação entre dispositivo de borda e nuvem para enviar tanto a decisão de particionamento quanto os dados de saída da camada de particionamento. O componente *Update Box* atualiza os parâmetros dinâmicos do sistema, como a taxa de envio e as probabilidades de uma amostra ser classificada em cada ramo. Dessa forma, o *Decision Maker* é responsável por tornar o POPEX um sistema realimentado, permitindo que o sistema se adapte às condições da infraestrutura de rede e da aplicação.

As próximas seções apresentam detalhadamente cada um dos componentes existentes no sistema POPEX. A Seção 4.1, a seguir, detalha os componentes *Background* e a Seção 4.2 apresenta o componente *Update Box*. Por fim, a Seção 4.3 detalha o *Decision Maker*.

## 4.1 *Background*

O componente *Background* recebe a arquitetura da DNN empregada e, após essa recepção, executa duas tarefas: extração dos parâmetros estáticos e construção do grafo baseado na arquitetura da DNN fornecida pela aplicação. A primeira tarefa permite obter os tempos de processamento na borda e na nuvem, além do tamanho em *bytes*, denotados, respectivamente, por $t_e$, $t_c$ e $\alpha$ na Figura 4.1. Em seguida, o *Background* constrói o grafo baseado na arquitetura da BranchyNet denotado por $\mathcal{G}$ na Figura 4.1. Essa tarefa permite converter o problema de particionamento em BranchyNet em um problema de particionamento de grafo. Uma vez que esses parâmetros são extraídos, o *Background* os fornece ao *Decision Maker*. As próximas seções abordam as tarefas de extração de parâmetros estáticos e construção do grafo baseado na arquitetura da DNN com saída antecipada.

### 4.1.1 Extração dos parâmetros estáticos

A tarefa de extração de parâmetros estáticos consiste em obter o tempo de processamento de cada camada neural da BranchyNet empregada, no dispositivo em borda ($\boldsymbol{t_e}$) e na nuvem ($\boldsymbol{t_c}$). Além disso, essa tarefa também obtém o tamanho dos dados de saída de cada camada neural ($\boldsymbol{\alpha}$), o qual será utilizado pelo componente para calcular o tempo de comunicação necessário para enviar os dados de saída de cada camada da borda à nuvem. Essa tarefa é executada pelo componente *Background* apenas uma vez, após a recepção da arquitetura da DNN empregada pela aplicação.

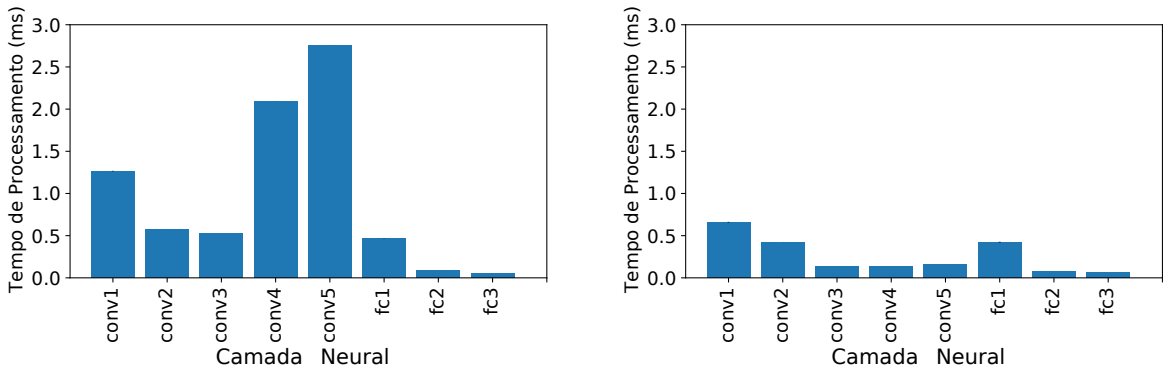
Na Figura 4.1, o termo $\boldsymbol{t_e}$ representa um vetor de tempo de processamento na nuvem, dado por $\boldsymbol{t_e} = [t_1^e, \ldots, t_N^e]^T$, sendo $t_i^e$ o tempo necessário para que o dispositivo em borda processe a $i$-ésima camada do ramo principal da BranchyNet. O termo $\boldsymbol{t_c} = [t_1^c, \ldots, t_N^c]^T$ é um vetor de tempo de processamento na borda, no qual $t_i^c$ é o tempo necessário para que o servidor em nuvem processe a $i$-ésima camada neural do ramo principal da BranchyNet. Os parâmetros $\boldsymbol{t_e}$ e $\boldsymbol{t_c}$ dependem diretamente do poder de processamento do dispositivo em borda e do servidor em nuvem em que as camadas da BranchyNet são executadas. Os tempos de processamento na borda $\boldsymbol{t_e}$ e na nuvem $\boldsymbol{t_c}$ podem ser obtidos pelo método de *benchmark* ou de construção de um modelo de predição.

O método *benchmark* consiste em medir o tempo de processamento de cada camada da BranchyNet, tanto na borda quanto na nuvem, executando o processo de inferência de uma amostra múltiplas vezes. Em seguida, calcula-se a média do tempo de processamento em cada uma das camadas. Por ser experimental, esse método retrata com exatidão o tempo de processamento das camadas no dispositivo em borda. Entretanto, necessita ser reexecutado sempre que se altera a arquitetura da BranchyNet ou o dispositivo de borda e nuvem.

As Figuras 4.2(a) e 4.2(b) mostram os tempos de processamento da AlexNet (Figura 2.9(a)) sendo executada no Google Colab[2] utilizando CPU e GPU, respectivamente. Google Colab é um serviço de computação em nuvem para aprendizado de máquina. Nesse caso, o Google Colab realiza o papel do servidor em nuvem. Primeiramente, essas figuras mostram que o tempo de processamento de cada camada é irregular, ou seja, segue um comportamento não-monótono conforme as camadas se aproximam da camada de saída. Quando compara-se o tempo de processamento nas Figuras 4.2(a) e 4.2(b), nota-se que a utilização de GPU permite reduzir significativamente o tempo de processamento de cada camada da DNN empregada.

O método de construir o modelo de predição pode ser utilizado como alternativa ao *benchmark*. Esse método consiste em treinar um modelo de regressão que consiga prever o tempo de inferência para diferentes tipos de camadas neurais (p.ex., convolucionais, *pooling*, *fully-connected*) e para as diversas configurações dessas camadas como, por exemplo,

[2] `https://colab.research.google.com`




(a) Google Colab usando CPU.

(b) Google Colab usando GPU.

Figura 4.2: Tempo de Processamento no servidor em nuvem de cada camada da AlexNet.

o número e o tamanho dos filtros utilizados em cada camada convolucional. Esse método não necessita ser reexecutado quando a arquitetura da BranchyNet é alterada, ao contrário do método *benchmark*. Contudo, é necessário ser reexecutado caso o dispositivo borda ou nuvem sejam alterados. Esse método adota como premissa que o modelo de regressão descreva corretamente o tempo de processamento real. Para construir o modelo preditivo da BranchyNet, a ideia é modelar o tempo de processamento de cada camada neural individualmente e, em seguida, somá-los para obter o tempo de processamento total da DNN. Entretanto, como mencionado anteriormente, há uma variedade de tipos de camadas neurais. Por exemplo, o Tensorflow [3] e o Pytorch [4] oferecem mais de 100 tipos de camadas neurais [14], o que dificulta a construção de modelos de regressão para todos os tipos de camadas existentes. Porém, uma pequena parcela de camadas neurais, dentre esses tipos diferentes, constitui a grande maioria das DNNs ou BranchyNets empregadas.

Finalmente, o módulo *Background* extrai a quantidade de dados de saída de cada camada da BranchyNet, denotado pelo vetor $\boldsymbol{\alpha}$ na Figura 4.1. Para isso, calcula-se o tamanho, em kB, dos dados de saída gerados por cada camada do ramo principal da BranchyNet. Esse parâmetro $\boldsymbol{\alpha}$ é então fornecido ao componente *Update Box* para que o tempo de comunicação seja calculado. O tempo de comunicação é o tempo necessário para enviar os dados de saída de uma determinada camada do dispositivo em borda à nuvem.

A Figura 4.3 apresenta os tamanhos dos dados de saída das camadas pertencentes ao ramo principal da AlexNet, ilustrada na Figura 2.4. Nessa figura, o termo *input* refere-se ao caso de enviar a imagem bruta da borda à nuvem. Esse caso corresponde ao cenário de processamento exclusivamente na nuvem, no qual o dispositivo de borda envia, por exemplo, uma imagem bruta diretamente à nuvem, sem que seja processada pelo dispositivo em borda. Nesse cenário, a nuvem é responsável por processar todas as camadas da DNN. É importante salientar que o dispositivo em borda armazena os tamanhos dos dados de saí-

[3] https://www.tensorflow.org/
[4] https://pytorch.org/

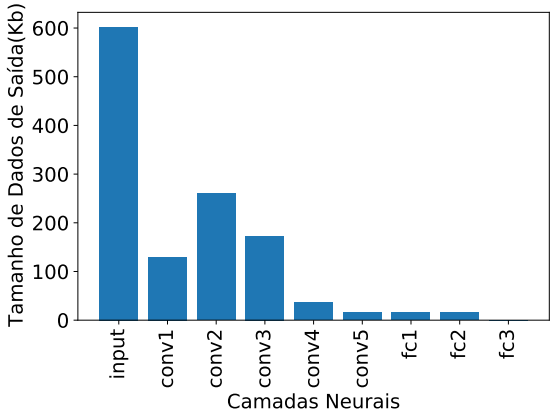



Figura 4.3: Tamanho de Saída, em kB, de cada camada da AlexNet

das das camadas do ramo principal, pois somente essas são candidatas a serem camadas de particionamento, o que será detalhado posteriormente. Além disso, nota-se, na Figura 4.3, que o tamanho dos dados também apresenta um comportamento não-monotônico. Isto é, camadas mais profundas não produzem necessariamente menos dados de saída do que as mais próximas da camada de entrada.

### 4.1.2 Construção do grafo de DNN com saídas antecipadas

A arquitetura da DNN pode ser modelada como grafo acíclico direcionado (DAG) [11, 16], no qual cada vértice representa uma ou um conjunto de camadas neurais da DNN e as arestas correspondem à comunicação entre as camadas. Seja a DNN modelada por um grafo $\mathcal{G}(\mathcal{V}, \mathcal{E})$, sendo $\mathcal{V} = \{v_1, \cdots, v_{|\mathcal{V}|}\}$ o conjunto que contém os vértices do grafo $\mathcal{G}$. Dessa forma, $v_1$ e $v_{|\mathcal{V}|}$ representam as camadas de entrada e saída da DNN, respectivamente.

O conjunto $\mathcal{E} = \{e_{ij} = (v_i, v_j) \mid v_i, v_j \in \mathcal{V}\}$ contém as arestas do grafo. Cada aresta $(v_i, v_j) \in \mathcal{E}$ existe se e somente se os dados de saída da camada $v_i$ são utilizados como dados de entrada pela camada $v_j$. Portanto, dada uma aresta $e_{ij} = (v_i, v_j)$, a camada $v_i$ é processada antes de $v_j$, o que significa que o fluxo de dados de saída parte de $v_i$ a $v_j$. Além disso, define-se uma função $\omega : \mathcal{E} \rightarrow \mathbb{R}_>$ que associa um peso real não-negativo $\omega_{(v_i,v_j)}$ a cada aresta $(v_i, v_j) \in \mathcal{E}$, o qual depende de característica da DNN e será detalhado posteriormente na Seção 4.6.

O principal objetivo de modelar uma DNN como um grafo é converter o problema de particionamento de DNN em um problema de particionamento de grafo, que é um problema já abordado na literatura [11, 16]. Esse problema é definido a partir de um grafo direcional e acíclico $\mathcal{G}(\mathcal{V}, \mathcal{E})$ com pesos reais não-negativos e um número $K \in \mathbb{N}^*$ de partições desse grafo $\mathcal{G}$. O problema de particionamento visa então dividir o grafo $\mathcal{G}$ em $K$ subconjuntos $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_K$, tal que:

1. $\bigcup_{i=1}^{K} \mathcal{V}_i = \mathcal{V}$,

2. $\mathcal{V}_i \cap \mathcal{V}_j = \emptyset, \quad \forall i \neq j \quad V_i, \mathcal{V}_j \in \mathcal{V}.$

Em uma DNN, o particionamento divide o grafo para que os subconjuntos $\mathcal{V}_i \in \mathcal{V}$ sejam processados em dispositivos diferentes. Por exemplo, o particionamento de DNN, geralmente, divide o grafo para que uma parte seja processada no dispositivo em borda e a outra na nuvem. Para isso, o particionamento de DNN divide o conjunto $\mathcal{V}$ do grafo $\mathcal{G}$ em dois subconjuntos ($K = 2$), $\mathcal{V}_e$ e $\mathcal{V}_c$. Os vértices $v_i \in \mathcal{V}_e$ representam as camadas da DNN processadas no dispositivo em borda, enquanto os vértices $v_i \in \mathcal{V}_c$ representam as camadas processadas na nuvem. Dessa forma, o particionamento de DNN está relacionado a determinar onde as camadas serão executadas.

Uma vez que a DNN é modelada como um grafo, o problema de particionamento de DNN consiste em um problema de otimização. O objetivo do problema é encontrar a partição que minimize uma função custo, relacionada aos pesos associados às arestas do grafo. Por exemplo, o particionamento pode minimizar o custo total da partição, correspondente à soma dos pesos das arestas que dividem o grafo. Uma vez que o particionamento é encontrado, é possível determinar a qual conjunto cada camada pertence.

Uma BranchyNet, por ser um caso particular de DNN, também pode ser modelada como um grafo. De acordo com a Seção 2.3, uma BranchyNet é caracterizada pela inserção dos ramos laterais $b_i \in \mathcal{B}$ entre as camadas intermediárias $v_i$ e $v_{i+1}$ do ramo principal, sendo $\mathcal{B}$ o conjunto de ramos laterais e $v_i, v_{i+1} \in \mathcal{V}'$, no qual $\mathcal{V}'$ é o conjunto composto pelos vértices do ramo principal. A Figura 4.4 ilustra a estrutura geral de uma BranchyNet com $N - 1$ ramos laterais. Nessa figura, cada vértice $v_1, \ldots, v_N$ representa camadas do ramo principal, e cada vértice $b_1, \ldots, b_{N-1}$ representa os ramos laterais inseridos nas camadas intermediárias. Dessa forma, o vértice $b_i$ dos ramos laterais corresponde a todas as camadas presentes no $i$-ésimo ramo lateral.

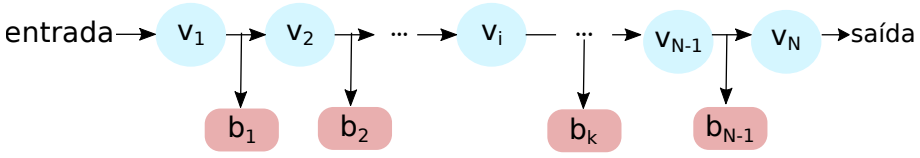



Figura 4.4: Ilustração de uma BranchyNet genérica.

A Figura 4.4 pressupõe que a estrutura do ramo principal é linear, ou seja, pode ser modelada como um grafo linear, como na AlexNet e na SqueezeNet. Assim, a estrutura geral da BranchyNet divide-se no ramo principal. Nos grafos em tripa, cada vértice $v_i \in \mathcal{V}'$ possui somente uma aresta de saída em direção ao vértice $v_{i+1}$. Nesse caso, o ramo principal da BranchyNet pode ser modelado como um grafo linear, denotado por $\mathcal{P}_{|\mathcal{V}'|}$. O grafo linear $\mathcal{P}_n = \mathcal{P}(n)$ é descrito por apenas um parâmetro $n$, que indica o número de vértices existentes no grafo. Assim, no grafo linear $\mathcal{P}_{|\mathcal{V}'|}$, o subíndice $|\mathcal{V}'|$ indica o número de vértices existentes no grafo e, consequentemente, o número de camadas do ramo principal da BranchyNet. Nesse tipo de grafo, o número de arestas é dado por $|\mathcal{V}'|-1$.

Para modelar o grafo da BranchyNet $\mathcal{G}_{\text{BDNN}}$, introduz-se, no grafo do ramo principal, os vértices $b_i \in \mathcal{B}$ correspondentes aos ramos laterais. A Figura 4.5 ilustra esse procedimento. Primeiramente, substituem-se as arestas $(v_i, v_{i+1})$ pelas arestas $(v_i, b_i)$ e $(b_i, v_{i+1})$. Em outras palavras, substitui-se a aresta do vértice $v_i$ ao seu vizinho $v_{i+1}$ por uma aresta de $v_i$ ao vértice do ramo lateral $b_i$ e, em seguida, adiciona-se uma aresta de $b_i$ a $v_{i+1}$. Dessa forma, a BranchyNet pode ser modelada pelo grafo linear denotado por $\mathcal{G}_{\text{BDNN}} = \mathcal{P}_{|\mathcal{V}' \cup \mathcal{B}|}$.

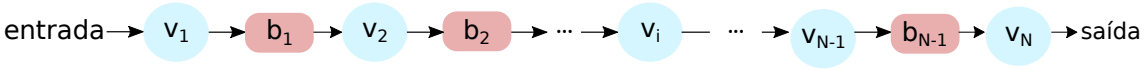



Figura 4.5: Ilustração de uma BranchyNet genérica com os ramos laterais já inseridos no ramo principal.

O grafo $\mathcal{G}_{\text{BDNN}}$ obtido é utilizado no problema de otimização executado na componente *Decision Maker*, com objetivo de encontrar a partição que equilibre o compromisso entre acurácia e tempo de inferência existente em DNNs com saídas antecipadas. Entretanto, a premissa que o ramo principal pode ser modelado como um grafo linear não é válida para diversas DNNs mais recentes (p.ex., GoogleNet [54] e ResNets [55]). Essas DNNs são modeladas como um DAG, dado por $\mathcal{G}_{\text{DNN}}$. Logicamente, o particionamento de DAGs é uma tarefa mais difícil do que particionar o caso particular de um grafo linear. Uma forma de contornar esse problema é linearizar a estrutura do grafo $\mathcal{G}_{\text{DNN}}$. Em outras palavras, é possível construir um grafo linear, denotado por $\mathcal{P}_{|\mathcal{V}'|}$, baseado no grafo $\mathcal{G}_{\text{DNN}}$. Para isso, é possível substituir trechos do grafo $\mathcal{G}_{\text{DNN}}$ por um único vértice representativo do respectivo trecho. Dessa forma, agrupa-se múltiplos vértices em um único vértice, de modo a linearizar o grafo $\mathcal{G}_{\text{DNN}}$ [14].

Uma vez que o componente *Background* constrói o grafo $\mathcal{G}_{\text{BDNN}}$ baseado na arquitetura da BranchyNet, esse componente fornece todos os parâmetros extraídos. Esses parâmetros são $\boldsymbol{t}_e$ e $\boldsymbol{t}_c$ para o *Decision Maker* e $\boldsymbol{\alpha}$ para o *Update Box*, detalhado a seguir.

## 4.2 *Update Box*

O componente *Update Box* é responsável pela extração dos parâmetros dinâmicos, como a taxa de envio entre a borda e a nuvem, além da atualização da probabilidade de classificação em cada ramo. Dessa forma, os parâmetros dinâmicos dependem tanto das amostras classificadas nos ramos laterais quanto da infraestrutura de rede entre a borda e a nuvem, enquanto os parâmetros estáticos extraídos pelo *Background* são próprios dos dispositivos nos quais a DNN é processada. Portanto, o componente *Update Box* executa a tarefa de extração dos parâmetros dinâmicos periodicamente.

O monitoramento da taxa de envio da infraestrutura de rede consiste em medir e armazenar a taxa de envio atual $B_0$ da infraestrutura de rede entre o dispositivo de borda e a

nuvem. Portanto, a partir da taxa de envio $B_0$ medida, o componente *Update Box* calcula o tempo de comunicação de cada camada $t_i^{\text{net}}$ da BranchyNet da seguinte forma

$$t_i^{\text{net}} = \frac{\alpha_i}{B_0}, \tag{4.1}$$

onde $\alpha_i$ é a quantidade de dados de saída da camada $v_i$, coletada na etapa de extração de parâmetros estáticos no componente *Background*, e $B_0$ é a taxa de envio medida no *Update Box*.

Conforme explicado na Seção 2.3, a probabilidade inicial $\boldsymbol{p}(t)$ de classificar as amostras em cada um dos ramos laterais é obtida durante a fase de validação do modelo da BranchyNet. Essa probabilidade inicial é fornecida ao componente *Decision Maker* que, por sua vez, utiliza-a para selecionar a estratégia de particionamento ótima, como será detalhado na Seção 4.3. Uma vez selecionada, o processo de inferência implementa-a para classificar as amostras recebidas pelo dispositivo em borda. Após um período de tempo, o componente *Decision Maker* informa ao *Update Box* o número de amostras de entrada classificada em cada ramo lateral. Dessa forma, o componente *Update Box* atualiza a probabilidade $\boldsymbol{p}(t)$ de classificar as amostras em cada um dos ramos laterais. Na versão atual do POPEX, assume-se que essa probabilidade $\boldsymbol{p}(t)$ ainda é constante. O próximo capítulo apresenta detalhadamente o componente *Decision Maker*.

## 4.3 *Decision Maker*

A partir dos parâmetros extraídos pelos componentes *Background* e *Update Box*, o componente *Decision Maker* é responsável por selecionar, dinamicamente, a camada de particionamento ótima. O seu objetivo é equilibrar o compromisso entre acurácia e tempo de inferência baseado no critério de latência predefinido pela aplicação. A seleção do particionamento ótimo determina quais as camadas da DNN com saídas antecipadas são executadas no dispositivo de borda e quais são executadas na nuvem. Para isso, o componente *Decision Maker* executa cinco tarefas: (1) estimação do tempo de inferência $\mathbb{E}[T_{inf}]$ ; (2) construção de um novo grafo $\mathcal{G}'_{\text{BDNN}}$ da BranchyNet; (3) otimização; (4) ajuste da configuração do limiar de entropia; (5) decisão.

A Figura 4.6 ilustra o componente *Decision Maker* e suas tarefas. É importante ressaltar que as três primeiras tarefas são executadas periodicamente, enquanto a quarta é executada a cada nova inferência.

No primeiro momento, o *Decision Maker* recebe os parâmetros de tempo de processamento na borda ($\boldsymbol{t}_e$) e na nuvem ($\boldsymbol{t}_c$), além do grafo associado à arquitetura da BranchyNet ($\mathcal{G}_{\text{BDNN}}$), vindos do componente *Background*. Além desses parâmetros, o *Decision Maker* recebe o tempo de comunicação ($\boldsymbol{t}_{\text{net}}$) e a probabilidade de classificação em cada ramo lateral $\boldsymbol{p}(t_0)$ vindos do componente *Update Box*.

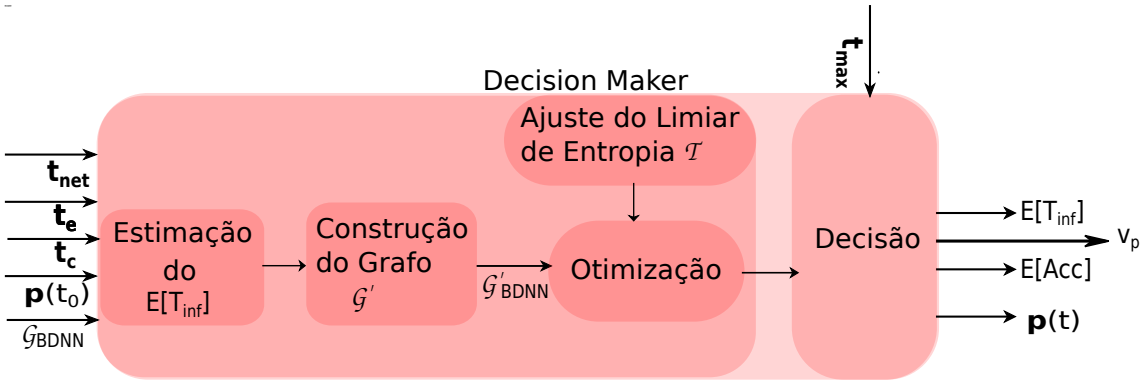



Figura 4.6: Ilustrações detalhadas das tarefas executadas pelo componente *Decision Maker*

Em seguida, o *Decision Maker* executa a etapa de estimação do tempo de inferência $\mathbb{E}[T_{inf}]$. Diferentemente das DNNs tradicionais, nas quais todas as camadas neurais são necessariamente processadas, seja na borda ou na nuvem, as DNNs com saídas antecipadas permitem que as amostras sejam classificadas antecipadamente nas suas camadas intermediárias. Assim, o tempo de inferência nas BranchyNets deve considerar a probabilidade de as amostras serem classificadas nos ramos laterais. Consequentemente, nesse caso, a amostra não é processada necessariamente por todas as camadas. Portanto, a estimação do tempo de inferência em uma DNN com saída antecipada deve considerar essas especificidades, como descrito detalhadamente na Seção 4.5.

Após a modelagem do tempo de inferência, o *Decision Maker* constrói o grafo de particionamento da BranchyNet, denotado por $\mathcal{G}'_{\text{BDNN}}$ na Figura 4.6. Esse novo grafo $\mathcal{G}'_{\text{BDNN}}$ possui pesos associados a cada aresta do grafo. Esses pesos correspondem ao tempo de processamento de cada camada ponderada pela probabilidade de serem processados. Essa etapa é descrita na Seção 4.7.

Após construir o grafo de particionamento da BranchyNet, o componente *Decision Maker* executa o processo de otimização. Esse processo determina um particionamento ótimo, visando equilibrar o compromisso entre tempo de inferência e acurácia, de modo a cumprir o requisito de latência predefinido pela aplicação. Para determinar esse particionamento ótimo, o problema de otimização divide-se em duas etapas. A primeira etapa visa maximizar a acurácia, enquanto a segunda visa minimizar o tempo de inferência.

Conforme descrito na Seção 2.3, cada ramo lateral da BranchyNet possui uma determinada acurácia. Assim, na primeira etapa, o componente *Decision Maker* escolhe a combinação dos ramos laterais que maximize a acurácia final da BranchyNet. Para isso, o componente *Decision Maker* realiza uma busca exaustiva entre os ramos laterais, a qual é detalhada na Seção 4.7.

A segunda etapa escolhe o particionamento que minimize o tempo de inferência. A tarefa de otimização executada pelo componente *Decision Maker* será detalhada adiante

na Seção 4.7. Contudo, conforme apresentado na Seção 2.3, a decisão do particionamento ótimo determinada pela tarefa de otimização depende também da configuração do hiperparâmetro chamado limiar de entropia em cada ramo lateral. Esse limiar afeta a decisão de particionamento, e, consequentemente, o tempo de inferência e a acurácia. Assim, o componente *Decision Maker* repete a tarefa de otimização para diversas configurações do limiar de entropia. Por fim, a tarefa de decisão escolhe o particionamento que equilibra o compromisso entre tempo de inferência e acurácia baseado no requisito de latência definido pela aplicação.

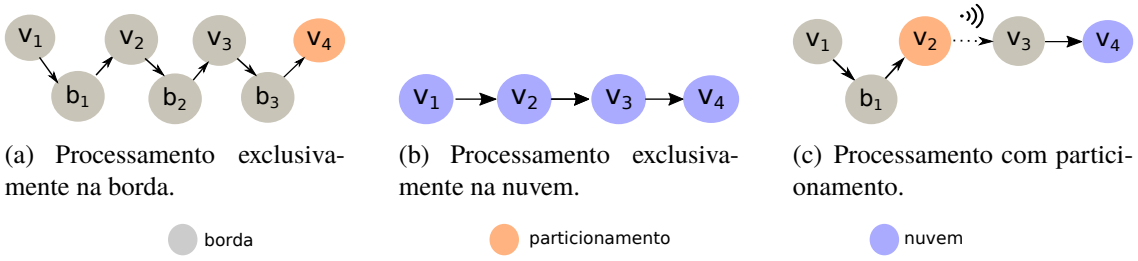
A próxima seção formaliza o problema de particionamento em BranchyNets. Em seguida, será detalhada cada tarefa executada pelo componente *Decision Maker*.

## 4.4 Particionamento de BranchyNets

O problema de particionamento de BranchyNets consiste em determinar uma camada capaz de dividi-la em duas partes, sendo uma processada no dispositivo local e a outra executada remotamente. Este trabalho considera o dispositivo local como sendo um dispositivo em borda e o dispositivo remoto como sendo um servidor em nuvem. Para isso, na prática, esse problema seleciona uma camada, denominada camada de particionamento, cujos dados de saída são enviados do dispositivo em borda à nuvem. Assim, as camadas anteriores à camada de particionamento são processadas no dispositivo em borda, enquanto as posteriores são processadas na nuvem.

Este trabalho aborda o particionamento de BranchyNets como um problema de particionamento de grafo, introduzido na Seção 4.1.2. O particionamento de BranchyNets separa o conjunto de vértices $\mathcal{V}$ do grafo $\mathcal{G}_{\text{BDNN}}$, fornecido pelo *Background*, em dois ($K = 2$) subconjuntos disjuntos $\mathcal{V}_e$ e $\mathcal{V}_c$. Os vértices $v_i \in \mathcal{V}_e$ representam as camadas da BranchyNet processadas no dispositivo de borda, enquanto os vértices $v_j \in \mathcal{V}_c$ correspondem às camadas processadas na nuvem. Como no particionamento de grafos, cada vértice pertence somente a um dos subconjuntos. Então, uma camada $v_i$ é processada somente no dispositivo de borda ou na nuvem, o que significa $\mathcal{V}_e \cap \mathcal{V}_c = \emptyset$. Conforme explicado na Seção 4.1.2, modela-se o ramo principal da BranchyNet como um grafo linear. Formalmente, a tarefa de particionamento consiste em encontrar uma camada de particionamento $v_s$, de modo que as camadas $v_1$ a $v_s$ são processadas na borda e as camadas restantes $v_{s+1}$ a $v_{|\mathcal{V}'|}$ são processadas na nuvem. A camada de particionamento $v_s$ é a última camada a ser processada no dispositivo em borda. Então, o dispositivo de borda envia os dados de saída da camada de particionamento $v_s$ à nuvem que, por sua vez, processa o restante (isto é, as camadas de $v_{s+1}$ a $v_{|\mathcal{V}'|}$). Para encontrar a camada de particionamento $v_s$, executa-se um problema de otimização, a fim de minimizar o custo associado aos pesos das arestas do grafo. Uma vez que a camada de particionamento é encontrada por meio de um problema de otimização, é possível determinar os conjuntos $\mathcal{V}_e$ e $\mathcal{V}_c$. Formalmente, em uma

BranchyNet, o conjunto das camadas processadas na borda é dado por $\mathcal{V}_e = \mathcal{V}' \cup \mathcal{B}$, sendo $\mathcal{V}' = \{v_1, \cdots, v_s\}$ e o conjunto de vértices correspondentes às camadas do ramo principal da BranchyNet e $\mathcal{B} = \{b_1, \cdots, b_{s-1}\}$ o conjunto dos ramos laterais. Consequentemente, o conjunto processado na nuvem é $\mathcal{V}_c = \mathcal{V} \backslash \mathcal{V}_e = \{v_{s+1}, \cdots, v_{|\mathcal{V}|}\}$. À exceção do Capítulo 6, é importante pontuar que neste trabalho não há ramo lateral sendo processado na nuvem. Portanto, não há vértice $b_i \in \mathcal{B}$ pertencente ao conjunto $\mathcal{V}_c$. Assim, todos os vértices $b_i$ posteriores a $v_s$ são descartados. Isso ocorre pois o tempo de processamento de cada camada neural na nuvem é significativamente inferior à borda. Consequentemente, não há necessidade de classificar antecipadamente, já que processar na nuvem não introduz atrasos de processamento significativos.




(a) Processamento exclusivamente na borda.

(b) Processamento exclusivamente na nuvem.

(c) Processamento com particionamento.

Figura 4.7: Possíveis cenários para o processamento da DNN.

A Figura 4.7 mostra os diferentes cenários de particionamento para uma BranchyNet composta de quatro camadas. Nessa figura, os vértices cinza representam as camadas processadas no dispositivo de borda, o vértice laranja corresponde à camada de particionamento $v_s$ e os vértices azuis referem-se às camadas processadas no servidor em nuvem. A Figura 4.7(a) ilustra o exemplo de processamento exclusivamente na borda, no qual a camada de particionamento escolhida é $v_4$. Assim, todas as camadas são processadas na borda. Dessa forma, tem-se o conjunto $\mathcal{V}_e = \{v_1, v_2, v_3, v_4, b_1, b_2, b_3\}$ e $\mathcal{V}_c = \emptyset$. Nesse caso, o dispositivo de borda não envia dados à nuvem. Por outro lado, a Figura 4.7(b) mostra o processamento exclusivamente na nuvem, sendo $\mathcal{V}_e = \emptyset$ e $\mathcal{V}_c = \{v_1, v_2, v_3, v_4\}$. Nesse caso, a quantidade de dados enviados à nuvem corresponde à quantidade de dados da imagem bruta. Por fim, a Figura 4.7(c) mostra um exemplo de processamento parcial, em que há particionamento na camada $v_2$, então $\mathcal{V}_e = \{v_1, b_1, v_2\}$ e $\mathcal{V}_c = \{v_3, v_4\}$. Nesse caso, os dados de saída da camada $v_2$ são enviados da borda à nuvem.

O particionamento divide a BranchyNet com um determinado objetivo, seja reduzir o tempo de inferência, maximizar a acurácia ou até mesmo minimizar o consumo de energia. Neste trabalho, o particionamento visa equilibrar o compromisso entre acurácia e tempo de inferência baseado no requisito de latência máxima fornecido pela aplicação. Para tal, a próxima seção modela a estimação do tempo de inferência em uma BranchyNet.

## 4.5 Estimação do Tempo de Inferência

Em aplicações de computação em borda usando DNNs, os dispositivos finais, como *smartphones* e vestíveis, enviam dados de entrada para os dispositivos em borda. Em uma DNN sem ramos laterais, a amostra de entrada é processada por todas as camadas localizadas na borda. Em seguida, o dispositivo de borda envia os dados de saída da camada de particionamento $v_s$ à nuvem, que é responsável por processar as camadas restantes. Consequentemente, nessas DNNs, o tempo de inferência depende tanto do tempo de processamento da borda e da nuvem como também do tempo de envio de dados entre borda e nuvem.

Os dispositivos de borda e servidores em nuvem diferem significativamente em relação às suas capacidades computacionais e, consequentemente, possuem atrasos diferentes de processamento. Portanto, o tempo de processamento para executar uma determinada camada depende do seu local de processamento. Seja $t_i^e$ e $t_i^c$ o tempo de processamento para processar a camada $v_i$ na borda e na nuvem, respectivamente. O atraso total de processamento na borda, quando a DNN não possui ramo lateral, é dado por

$$T_e = \sum_{i \,|\, v_i \in \mathcal{V}_e} t_i^e, \tag{4.2}$$

e o tempo de processamento para processar as camadas restantes na nuvem é dado por

$$T_c = \sum_{i | v_i \in \mathcal{V}_c} t_i^c. \tag{4.3}$$

O tempo de comunicação depende do tamanho dos dados de saída da camada de particionamento e da taxa de envio da infraestrutura de rede. Os tamanhos dos dados de saída gerados por cada camada da DNN apresentam um comportamento não monótono, como já mostrado na Figura 4.3. Assim, há camadas mais próximas da camada de entrada que geram mais dados de saída do que camadas mais profundas (isto é, próxima da camada de saída), resultando em um tempo de comunicação ainda maior. Para cada camada $v_i$, o tempo de comunicação $t_i^{\text{net}}$ é dado pela Equação 4.1. Como visto anteriormente, o dispositivo em borda envia os dados de saída da camada de particionamento $v_s$ à nuvem. Portanto, utiliza-se o tempo de comunicação $t_s^{\text{net}}$ para calcular o tempo de inferência, já que $v_s$ é a camada de particionamento.

Em uma DNN sem ramo lateral, o tempo de inferência é a soma do atraso total de processamento ($T_e$ e $T_c$) com o tempo de comunicação ($t_s^{\text{net}}$), conforme mostrado na Equação 4.4 [16].

$$T_{inf}^{DNN} = T_e + t_s^{\text{net}} + T_c \tag{4.4}$$

Se for considerada uma DNN com saídas antecipadas, tal qual uma BranchyNet, o

processo de inferência pode terminar em um dos ramos, caso a classificação atinja um determinado critério de confiança. Assim, o tempo de inferência em BranchyNet deve modelar a probabilidade de classificar as amostras antecipadamente nos ramos laterais. Para modelar o tempo de inferência para as DNNs com saídas antecipadas, divide-se a análise em um caso particular e um caso geral.

### Caso Particular

Primeiramente, considera-se o caso particular de uma BranchyNet com apenas um ramo inserido entre quaisquer camadas intermediárias $v_k$, sendo $1 \leq k \leq (|\mathcal{V}| - 1)$. Conforme descrito no algoritmo de inferência da BranchyNet na Seção 2.3, quando a amostra alcança o único ramo lateral $b_k$, essa pode ser classificada e terminar a inferência. Caso contrário, ela segue sendo processada pelas próximas camadas até atingir a saída. Para modelar essas duas possibilidades, define-se uma variável aleatória de Bernoulli $X_k$ que assume o valor 1, se a amostra é classificada no ramo lateral $b_k$ com probabilidade $\mathsf{P}[X_k = 1] = p_k$, sendo $p_k \in [0, 1]$. Caso contrário, faz-se $X_k = 0$ se a amostra for processada pelas camadas posteriores ao ramo lateral $b_k$, com probabilidade $\mathsf{P}[X_k = 0] = 1 - p_k$. Dessa forma, o tempo de inferência $\mathbb{E}[T_{inf}^{\text{BDNN}}(k)]$ no caso particular é dado por

$$\mathbb{E}[T_{inf}^{\text{BDNN}}(k)] = \sum_{\substack{i|v_i \in \mathcal{V}_e \\ i \leq k}} t_i^e + (1 - p_k) \left( \sum_{\substack{i|v_i \in \mathcal{V}_e \\ i > k}} t_i^e + t_s^{\text{net}} + T_c \right) \tag{4.5}$$

A Equação 4.5 mostra que o dispositivo de borda sempre processa as camadas anteriores ao único ramo lateral $b_k$. Contudo, os tempos de processamento e comunicação das camadas posteriores ao ramo $b_k$ são ponderados pela probabilidade $(1 - p_k)$ de não ser classificado no ramo lateral $b_k$. Em seguida, generaliza-se o problema para o caso geral, no qual a BranchyNet possui múltiplos ramos laterais.

### Caso Geral

Esta seção generaliza a análise anterior considerando a BranchyNet com $|\mathcal{V}| - 1$ ramos laterais, conforme ilustrado na Figura 2.8. Como no caso particular, define-se variáveis aleatórias de Bernoulli $X_k$ para cada ramo lateral $b_k \in \mathcal{B}$, resultando em uma sequência de variáveis aleatórias de Bernoulli $\boldsymbol{X} = (X_1, X_2, \cdots, X_{|\mathcal{V}|-1})$. Para modelar o caso geral, a ideia é que para uma amostra ser classificada no ramo lateral $b_k$, essa amostra não atende o critério de confiança em nenhum dos $k - 1$ ramos laterais anteriores. Assim, supondo que uma amostra seja classificada no ramo $b_k$, então a variável aleatória de Bernoulli $X_k$ assume o valor 1 pela primeira vez no ramo $b_k$, após $k - 1$ tentativas. Para modelar isso, define-se a variável aleatória $Y$, que representa o número dos ramos laterais que

já foram processados antes do atual ramo lateral $b_k$, e que não classificaram a amostra. Assim, a probabilidade de uma amostra ser classificada no ramo lateral $b_k$ corresponde à probabilidade da variável aleatória $\mathsf{P}[Y = k]$, dada por

$$p_Y(k) = \mathsf{P}[Y = k] = p_k \prod_{i=1}^{k-1} (1 - p_i), \tag{4.6}$$

onde $p_Y(k)$ denota a probabilidade de a amostra ser classificada no $k$-ésimo ramo lateral, após $k - 1$ tentativas e $p_i$ a probabilidade de classificar a amostra no ramo lateral $b_i$. Note que a probabilidade $p_Y(k)$ assemelha-se a uma distribuição geométrica.

Considerando um lote de amostras de entrada, é possível calcular o valor esperado do tempo de inferência. Para tal, o particionamento pode ocorrer até mesmo antes do primeiro ramo de entrada. Nesse caso, todas as camadas são processadas na nuvem, tal qual a Figura 4.7(b). Assim, o tempo de inferência é modelado pela Equação 4.4, como uma DNN sem ramo lateral, já que o modelo considera que a nuvem não possui uma BranchyNet. No entanto, se o particionamento ocorrer após um dos ramos laterais, a amostra pode ser classificada por esse dado ramo. Nesse caso, em uma BranchyNet com múltiplos ramos laterais, o valor esperado do tempo de inferência é

$$\mathbb{E}[T_{inf}^{\text{BDNN}}(k)] = \sum_{\substack{i|v_i \in \mathcal{V}_e \\ i \leq k}} t_i^e + (1 - p_Y(k)) \left( \sum_{\substack{i|v_i \in \mathcal{V}_e \\ i > k}} t_i^e + t_s^{\text{net}} + T_c \right). \tag{4.7}$$

A Equação 4.7 mostra que o dispositivo de borda sempre processa as camadas antes do ramo lateral $b_k$. No entanto, os tempos de processamento e comunicação dos ramos remanescentes são ponderados pela probabilidade de classificação da amostra de entrada no ramo lateral. No caso extremo, em que as amostras de entrada são sempre classificadas no ramo lateral, o que significa $p_Y(k) = 1, \forall 1 \leq k \leq |\mathcal{V}| - 1$, a Equação 4.7 não considera o tempo de comunicação nem o tempo de processamento para as camadas restantes. No outro extremo, se a inferência nunca termina no ramo lateral, isto é $p_Y(k) = 0$, a Equação 4.7 é igual à Equação 4.4. De acordo com a posição da camada de particionamento, o valor esperado do tempo de inferência pode ser modelado da seguinte forma:

$$\mathbb{E}[T_{inf}(k)] = \begin{cases} T_{inf}^{\text{DNN}}, & \forall v_k, v_s \mid s < k. \\ \mathbb{E}[T_{inf}^{\text{BDNN}}], & \text{caso contrário.} \end{cases} \tag{4.8}$$

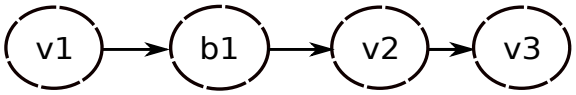
Na Equação 4.8, quando o particionamento na camada $v_s$ ocorre antes do ramo lateral inserido na camada $v_k$ (isto é, $s < k$), então o valor de inferência $\mathbb{E}[T_{inf}(k)]$ corresponde ao tempo de inferência de uma DNN tradicional, já que a nuvem não possui ramos laterais. Caso contrário, o tempo de inferência corresponde ao valor esperado em DNNs com saídas

antecipadas, dado pela Equação 4.7.

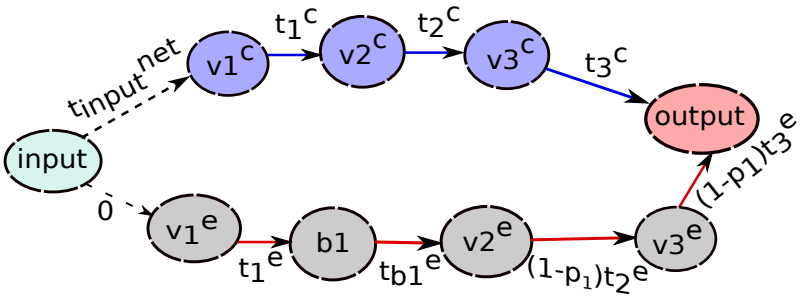
A próxima seção utiliza a estimação do tempo de inferência para construir o grafo de particionamento da BranchyNet $\mathcal{G}'_{\text{BDNN}}$, empregado na tarefa de otimização do particionamento da BranchyNet.

## 4.6 Construção do grafo de particionamento da BranchyNet

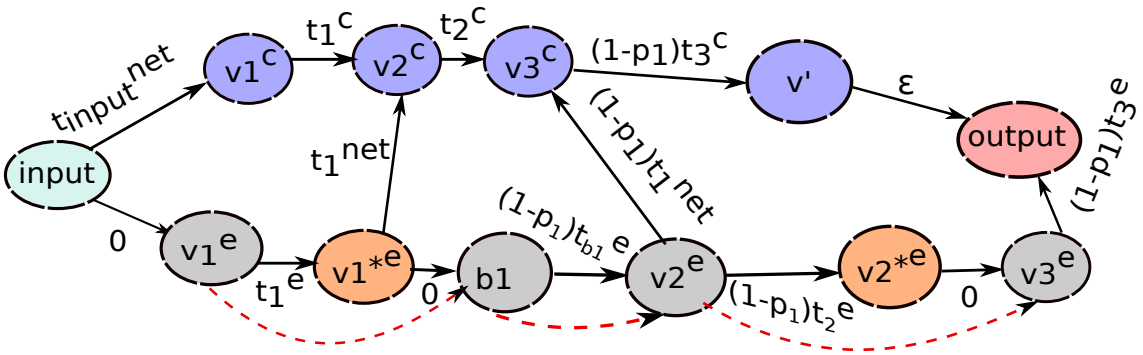
Esta seção propõe o modelo de um grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ baseado no grafo construído pelo componente *Background*. O grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ permite associar um dos atrasos da terna $(t_i^e, t_i^c, t_i^{\text{net}})$ em cada uma das arestas, como descrito na Seção 4.5. O atraso associado à aresta $(v_i, v_j)$ depende de onde o vértice $v_i$ é processado e da camada de particionamento $v_s$. Dessa forma, a partir da construção do grafo de particionamento, mostra-se que o problema de particionamento pode ser modelado como o problema de caminho mais curto.




(a) Grafo $\mathcal{G}_{\text{BDNN}}$ extraído pelo componente *Background*.




(b) Grafo utilizado para modelar os cenários de processamento exclusivamente na borda e na nuvem.




(c) Representação do grafo $\mathcal{G}'_{\text{BDNN}}$ baseado em uma BranchyNet de três camadas com somente um ramo lateral.

Figura 4.8: Construção do grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$

A Figura 4.8(a) ilustra o grafo $\mathcal{G}_{\text{BDNN}}$ construído pelo componente *Background*, baseado em uma BranchyNet constituída de um ramo principal com três camadas e um ramo lateral inserido após a primeira camada do ramo principal. Após receber o grafo $\mathcal{G}_{\text{BDNN}}$ do *Background*, o componente *Decision Maker* constrói o grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$. O grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ deve modelar os cenários de processamento exclusivamente na borda, exclusivamente na nuvem e com particionamento.

Com objetivo de construir o grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$, a primeira etapa consiste em modelar o cenário de processamento exclusivamente na borda e na nuvem. Para isso, conforme mostrado na Figura 4.8(b), criam-se dois grafos lineares disjuntos $\mathcal{P}^e_{|\mathcal{V}'\cup\mathcal{B}|}$ e $\mathcal{P}^c_{|\mathcal{V}_c|}$. Nessa figura, os vértices cinza correspondem aos vértices do grafo $\mathcal{P}^e_{|\mathcal{V}'\cup\mathcal{B}|}$ e representam as camadas neurais processadas na borda, enquanto os vértices azuis correspondem aos vértices do grafo $\mathcal{P}^c_{|\mathcal{V}_c|}$ e representam as camadas neurais processadas na nuvem. Para terminar de modelar o cenário de processamento exclusivamente na borda e na nuvem, introduz-se dois vértices virtuais chamados $input$ e $output$ e, depois, acrescentam-se as arestas (pontilhadas na Figura 4.8(b)) $(\text{input}, v_1^c)$ e $(\text{input}, v_1^e)$ no grafo $\mathcal{G}'_{\text{BDNN}}$. Em seguida, associam-se pesos a cada uma dessas arestas desses grafos. As arestas de $\mathcal{P}^e_{|\mathcal{V}'\cup\mathcal{B}|}$ e $\mathcal{P}^c_{|\mathcal{V}'|}$ representam o tempo de processamento para processar a camada $v_i$ na borda ($t_i^e$) e na nuvem ($t_i^c$), respectivamente. A aresta $(\text{input}, v_1^c)$ representa o caso de enviar uma imagem bruta diretamente à nuvem. Nesse caso, o peso $\omega_{(\text{input},v_1^c)}$ associado a essa aresta corresponde ao tempo de comunicação necessário para enviar a imagem bruta diretamente à nuvem denotado por $t^{\text{net}}_{\text{input}}$ na Figura 4.8(b). Por outro lado, na aresta $\omega_{(\text{input},v_1^e)}$, o peso $\omega_{(\text{input},v_1^e)} = 0$, pois não há envio de dados. A Figura 4.8(b) mostra que o caminho entre os nós $input$ e $output$, usando somente as arestas vermelhas e azuis, pode ser utilizado para calcular o tempo de inferência para os cenários de processamento exclusivamente na borda e na nuvem, respectivamente.

A Figura 4.8(c) mostra o grafo de modelagem do cenário de processamento com particionamento e o grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ final. Para modelar o cenário de processamento com particionamento, primeiramente, conforme a Figura 4.8(c), introduz-se um vértice auxiliar $v_i^{*e} \in \mathcal{V}_A$ (isto é, os vértices laranja da Figura 4.8(c)) para todo vértice $v_i^e$ pertencente ao ramo principal, sendo $\mathcal{V}_A$ o conjunto de vértices auxiliares. Então, adiciona-se a aresta $(v_i^e, v_i^{*e})$ e substitui-se $(v_i^e, v_i^{*e})$ (arestas pontilhadas vermelhas na Figura 4.8(c)) pela aresta $(v_i^{*e}, v_{i+1}^e)$. Por exemplo, na Figura 4.8(c), as arestas pontilhadas vermelhas são substituídas pelas arestas pretas $(v_1^e, v_1^{*e})$ e $(v_1^{*e}, b_1)$.

Por fim, para modelar o particionamento, adicionam-se arestas entre os vértices $v_s \in \mathcal{V}_e$ processados na borda e $v_{s+1} \in \mathcal{V}_c$ na nuvem. Na Figura 4.8(c), as arestas laranja representam a comunicação entre a borda e a nuvem. Por exemplo, a aresta $(v_1^{*e}, v_2^c)$ significa que os dados de saída da camada $v_1$ são enviados da borda à nuvem, ou seja, $v_1$ é a camada de particionamento. Por fim, associa-se os pesos referentes à comunicação entre borda e nuvem no grafo $\mathcal{G}'_{BDNN}$. O peso dessas arestas corresponde ao tempo de comunicação $t_s^{\text{net}}$

para enviar os dados de saída da camada de particionamento $v_s \in \mathcal{V}'$, localizada na borda, à camada $v_{s+1} \in \mathcal{V}_c$ localizada na nuvem. Dessa forma, todos os vértices anteriores à camada de particionamento são processados na borda, enquanto os vértices posteriores são processados na nuvem. Portanto, no exemplo anterior, o peso associado à aresta $(v_1^{*e}, v_2^c)$ é $\omega_{(v_1^{*e}, v_2^c)} = t_1^{\text{net}}$. Como os dados da camada de particionamento são enviados à nuvem, todas as camadas posteriores são processadas na nuvem, então não há retorno de dados à borda. Nas Figuras 4.8(b) e 4.8(c) , nota-se que os pesos associados às arestas posteriores ao ramo lateral $b_1$ são ponderados pelo termo $1 - p_1$, sendo $p_1$ a probabilidade da amostra ser classificada nesse ramo lateral. Dessa forma, quanto maior a probabilidade de uma amostra ser classificada no ramo lateral, menos significativos são os pesos das arestas após esse ramo lateral.

Na Figura 4.8(c), nota-se, quando a probabilidade $p_1 = 1$, que todas as arestas posteriores ao ramo $b_1$ são ponderadas pelo termo $(1 - p_1)$, consequentemente as arestas posteriores ao ramo $b_1$ possuem pesos iguais a zero. Nesse caso há uma ambiguidade, pois o caminho, que corresponde ao processamento exclusivamente na borda, possui o mesmo custo que o caminho com particionamento na camada $v_2^e$, indicando que dados são enviados à nuvem. Para impedir essa ambiguidade entre a escolha do menor caminho no caso $p_1 = 1$, adiciona-se o vértice virtual $v'$ como o sucessor de $v_3^c$ e predecessor do vértice $output$. Posteriormente, é adicionado um peso $\epsilon$ à aresta $(v', \text{output})$. O valor do peso $\epsilon$ deve ser muito pequeno, para que não interfira no resultado do problema de encontrar o menor caminho, sendo apenas um desempate entre caminhos com mesmo custo, quando $p_1 = 1$. Nota-se também que, para $p_1 = 0$, nenhuma amostra é classificada no ramo lateral $b_1$. Assim, o grafo $\mathcal{G}'_{\text{BDNN}}$ também modela uma DNN tradicional. De forma geral, em uma DNN tradicional sem os ramos laterais, os pesos das arestas do grafo de particionamento são associados da seguinte forma:

$$\omega_{(v_i,v_j)} = \begin{cases} t_i^e, & \text{se } v_i \in \mathcal{V}_e \\ t_i^c, & \text{se } v_i \in \mathcal{V}_c \\ t_i^{\text{net}}, & \text{se } (v_i \in \mathcal{V}_A, v_j \in \mathcal{V}_c) \text{ ou } (v_i = \text{input}, v_j \in \mathcal{V}_c) \\ \epsilon, & \text{se } v_i \in \mathcal{V}_c, v_j = \text{output} \\ 0, & \text{se } v_i \in \mathcal{V}_A, v_j \in \mathcal{V}_e \end{cases} \tag{4.9}$$

Até esse ponto, a Equação 4.9 associa pesos a uma DNN tradicional sem os ramos laterais. Entretanto, a BranchyNet possui ramos laterais que permitem classificar determinadas amostras antecipadamente. Portanto, os pesos $\omega_{(v_i,v_j)}^{\text{BDNN}}$ associados às arestas do grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ da BranchyNet devem ser ponderados pela probabilidade $p_Y(k)$ de a amostra ser classificada em cada um dos ramos laterais $b_k$. Então, os pesos associados às arestas em uma BranchyNet são dados por

$$\omega_{(v_i,v_j)}^{\text{BDNN}}(k) = p_Y(k)\omega_{(v_i,v_j)}, \tag{4.10}$$

onde $\omega^{\text{BDNN}}_{(v_i,v_j)}(k)$ é o peso da aresta $(v_i, v_j)$ do grafo $\mathcal{G}'_{\text{BDNN}}$ ponderados pela probabilidade $p_Y(k)$ de classificar a amostra no ramo $b_k$. No caso das camadas anteriores ao primeiro ramo lateral, a probabilidade $p_Y(k)$ assume valor igual a 1, já que não há ramos laterais para classificar amostras antecipadamente, então tais camadas são sempre processadas. Caso contrário, a probabilidade $p_Y(k)$ é dada conforme a Equação 4.6. Após construir o grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$, o componente *Decision Maker* executa a tarefa de otimização do particionamento da BranchyNet, descrita a seguir.

## 4.7 Otimização do Particionamento de BranchyNets

O particionamento em DNNs com saídas antecipadas, como as BranchyNets, depende não só dos tempos de processamento e de comunicação, como nas DNNs tradicionais, mas também da probabilidade de classificação das amostras nos ramos laterais. Essa probabilidade, por sua vez, depende da configuração do limiar de entropia em cada ramo lateral. Então, define-se o vetor de configuração dos limiares de entropia $\boldsymbol{\tau} = [\tau_1, \ldots, \tau_{|\mathcal{B}|}]^T \in \mathcal{T}$ sendo $\tau_i$ o limiar de entropia do ramo lateral $b_i$ e $\mathcal{T}$ o conjunto das todas configurações predefinidas.

O intuito desta tarefa é determinar um particionamento ótimo, com objetivo de equilibrar o compromisso entre acurácia e tempo de inferência baseado no requisito de latência máxima $t_{\text{max}}$ definido pela aplicação. Consequentemente, essa tarefa determina os conjuntos $\mathcal{V}_e$ e $\mathcal{V}_c$, definindo em qual dispositivo, borda ou nuvem, cada camada é processada. Esta tarefa divide-se em duas etapas. A primeira visa maximizar a acurácia da BranchyNet, enquanto a segunda minimiza o tempo de inferência.

A primeira etapa consiste em selecionar um subconjunto de ramos laterais $\mathcal{B}^* \in \mathcal{B}$ que maximize o valor esperado da acurácia $\mathbb{E}[Acc|\boldsymbol{\tau}]$, definido como

$$\mathbb{E}[Acc|\boldsymbol{\tau}] = \sum_{\substack{i|b_i \in \mathcal{B} \\ \tau_i \in \boldsymbol{\tau}}} \mathbb{P}[\eta(\boldsymbol{p}^{(i)}) < \tau_i] \cdot acc_i, \tag{4.11}$$

onde $\mathbb{P}[\eta(\boldsymbol{p}^{(i)}) < \tau_i]$ é a probabilidade de classificar uma amostra no ramo $b_i$ e a $acc_i$ é a acurácia do ramo $b_i$. O valor de acurácia em cada um dos ramos laterais é obtido durante o processo de treinamento e validação do modelo de BranchyNet empregado. Para selecionar o subconjunto $\mathcal{B}^*$, essa tarefa executa o Algoritmo 1, descrito a seguir.

O Algoritmo 1 executa uma busca exaustiva da seguinte forma. Primeiramente, esse algoritmo itera para todos os $|\mathcal{B}|$ ramos laterais. Assim, a complexidade dessa busca exaustiva é de $\mathcal{O}(|\mathcal{B}|)$. Em seguida, define-se uma acurácia alvo denotada por $\widehat{acc}$. Durante a iteração para todos os ramos laterais, a acurácia alvo assume o valor da acurácia do ramo atual (Linha 4). Dessa forma, a ideia da acurácia alvo $\widehat{acc}$ é que se houver ramos laterais nas camadas posteriores com uma acurácia inferior a $\widehat{acc}$, podem ser re-

**Algoritmo 1** Busca da Acurácia Máxima

**Entradas:**

$\mathcal{G}'_{\text{BDNN}}$: Grafo de particionamento associado à BranchyNet
$\boldsymbol{\tau} = [\tau_1, \cdots, \tau_{|\mathcal{B}|}]$: vetor configuração do limiar de entropia dos ramos
$\boldsymbol{Acc} = [acc_1, \cdots, acc_{|\mathcal{B}|}]$: vetor acurácia dos ramos laterais
$\boldsymbol{p} = \{p_1, \ldots, p_{|\mathcal{B}|}\}$: vetor que contém a probabilidade $p_i$ de classificação de amostras no $i$-ésimo ramo lateral

**Saída:**

$\mathcal{G}^*_{\text{BDNN}}$: Grafo de particionamento composto pelos conjunto de ramos laterais $\mathcal{B}^*$, que atingem a acurácia máxima

1: **procedimento** BUSCAACURACIA($\mathcal{G}'_{\text{BDNN}}, \boldsymbol{\tau}, \boldsymbol{Acc}, \boldsymbol{p}$)
2: $acc_{max} \leftarrow 0$;
3: **para** $i = 1, \cdots, |\mathcal{B}|$ **faça**
4: $\widehat{acc} \leftarrow acc_i$;
5: $\mathcal{B}' \leftarrow$ lista de índices j que satisfazem $acc_j \leq \widehat{acc}$;
6: $\mathbb{E}[Acc \,|\, \boldsymbol{\tau}] \leftarrow calcularValorEsperadoAcuracia(\boldsymbol{Acc}, \mathcal{B}')$;
7: **se** $\mathbb{E}[Acc \,|\, \boldsymbol{\tau}] > acc_{max}$ **então**
8: $acc_{max} \leftarrow \mathbb{E}[Acc \,|\, \boldsymbol{\tau}]$;
9: $\mathcal{B}^* \leftarrow \mathcal{B}'$;
10: $\mathcal{G}^*_{\text{BDNN}} \leftarrow podar(\mathcal{G}'_{\text{BDNN}}, \mathcal{B}^*)$;
11: **retorna** $[\mathcal{G}^*_{\text{BDNN}},\ acc_{max}]$;

movidos, já que só reduziriam o valor esperado da acurácia e também aumentariam o tempo de processamento. Portanto, em seguida, remove-se todos os ramos laterais $b_i \in \mathcal{B}$ (Linhas 5-6), tal que $acc_i < \widehat{acc}$. Utilizando os ramos que restaram no grafo, a função $calcularValorEsperadoAcuracia$ (Linha 7) calcula o valor esperado da acurácia $\mathbb{E}[Acc|\boldsymbol{\tau}]$ dada uma configuração de limiar conforme a Equação 4.11. Em seguida, o conjunto $\mathcal{B}^*$ é atualizado a cada iteração, com objetivo de armazenar os ramos laterais que resultam na acurácia máxima (Linhas 7-9). Por fim, a função $podar$ remove do grafo $\mathcal{G}'_{\text{BDNN}}$ os ramos laterais que não estão no conjunto $\mathcal{B}^*$, resultando no grafo $\mathcal{G}^*_{\text{BDNN}}$ (Linha 10). Portanto, o grafo $\mathcal{G}^*_{\text{BDNN}}$ consiste no grafo de particionamento com os ramos laterais do conjunto $\mathcal{B}^*$. Em seguida, executa-se a segunda etapa com objetivo de encontrar um particionamento para minimizar a inferência.

A partir do grafo $\mathcal{G}^*_{\text{BDNN}}$ obtido pelo Algoritmo 1, a segunda etapa determina o particionamento ótimo que minimiza o tempo de inferência, conforme a Equação 4.8. Para tanto, essa segunda etapa executa um problema de otimização cuja entrada é apenas o grafo $\mathcal{G}^*_{\text{BDNN}}$. Como definido na Seção 4.1.2, o ramo principal da BranchyNet é modelado como um grafo linear $\mathcal{P}_{|\mathcal{V}'|}$. Consequentemente, após encontrar a camada de particionamento $v_s$ que minimiza $\mathbb{E}[T_{inf}(k)]$, determinam-se as partições $\mathcal{V}_e$ e $\mathcal{V}_c$. A partir da observação da Figura 4.8(c), nota-se que os caminhos entre os nós *input* e *output* modelam o valor esperado do tempo de inferência $\mathbb{E}[T_{inf}(k)]$, segundo a Equação 4.8. Portanto, o objetivo é determinar o caminho com o mínimo custo que conecta os vértices virtuais *input* e *out-*

*put*. O custo do caminho é definido como a soma dos pesos associados às arestas do grafo $\mathcal{G}^*_{\text{BDNN}}$. Dessa forma, mostra-se a equivalência entre o problema de particionamento de BranchyNets e o problema de caminho mais curto [56]. Dados dois vértices, o problema de caminho mais curto é encontrar um caminho que conecte esses dois vértices, minimizando os custos das arestas. Neste trabalho, esses dois vértices são os nós *input* e *output*. O Algoritmo 2 executa a segunda etapa da tarefa de Otimização do Particionamento da BranchyNet. Assim, o Algoritmo 2 visa determinar o particionamento ótimo que minimiza o tempo de inferência, encontrando o caminho mais curto no grafo $\mathcal{G}^*_{\text{BDNN}}$ entre os vértices $input$ e $output$. Para isso, primeiramente, o Algoritmo 2 recebe o grafo $\mathcal{G}^*_{\text{BDNN}}$ (Linha 1) obtido pelo Algoritmo 1. Em seguida, esse algoritmo executa o algoritmo de Dijkstra (Linha 2) para encontrar o caminho mais curto, ou seja, o caminho com menor custo entre os vértices $input$ e $output$. O algoritmo de Dijkstra retorna o caminho mais curto $\pi^*$ e o custo total do caminho que corresponde ao $\mathbb{E}[T_{inf}]$. Em seguida, o Algoritmo 2 executa a função $obterParticionamento$ que encontra a camada de particionamento $v_s$ no caminho $\pi^*$, ou seja, a última camada do caminho $\pi^*$ processada na borda.

**Algoritmo 2** Minimizando o Tempo de Inferência

**Entradas:**

$\mathcal{G}^*_{\text{BDNN}}$: grafo de particionamento composto pelo conjunto de ramos laterais $\mathcal{B}^*$, que atingem a acurácia máxima

**Saída:**

$\mathbb{E}[T_{inf}]$: valor esperado do tempo de inferência
$v_s$: camada de particionamento

1: **procedimento** MINIMIZATEMPOINFERENCIA($\mathcal{G}^*_{\text{BDNN}}$)
2: $[\mathbb{E}[T_{inf}], \pi^*] \leftarrow algoritmoDijkstra(\mathcal{G}^*_{\text{BDNN}})$;
3: $v_s \leftarrow obterParticionamento(\mathcal{G}^*_{\text{BDNN}})$;
4: **retorna** $[\mathbb{E}[T_{inf}], v_s]$;

Na Figura 4.8(c), se o caminho mais curto $\pi^*$ contém as camadas do conjunto $\mathcal{V}_e$, isso significa que a estratégia de particionamento é processar as camadas exclusivamente na borda. Por outro lado, se o caminho mais curto $\pi^*$ contém os vértices $v_i \in \mathcal{V}_c$, então significa que o processamento ocorre apenas na nuvem. Caso contrário, se há vértices pertencentes tanto a $\mathcal{V}_e$ quanto a $\mathcal{V}_c$, então há particionamento. Nos três cenários anteriores, o custo total é dado pela Equação 4.8. Dessa forma, mostra-se que há equivalência entre os problemas de particionamento de BranchyNet e de caminho mais curto. Portanto, o tempo de inferência $\mathbb{E}[T_{inf}(k)]$ é definido como

$$\mathbb{E}[T_{inf}(k)] = \sum_{(v_i,v_j)\in\pi^*} \omega_{(v_i,v_j)}, \tag{4.12}$$

onde $\pi^*$ é o caminho mais curto encontrado entre os vértices *input* e *output*.

O problema do caminho mais curto é um problema bem conhecido na literatura e pode

ser resolvido em tempo polinomial. Este trabalho utiliza o algoritmo de Dijkstra para encontrar o caminho mais curto com a complexidade computacional de $\mathcal{O}(m+n\log(n))$, sendo $m$ e $n$ o número de arestas e vértices do grafo, respectivamente.

O grafo $\mathcal{G}_{\text{BDNN}}$ extraído pelo componente *Background* é modelado como um grafo linear $\mathcal{P}_{|\mathcal{V}'|+|\mathcal{B}|}$, sendo $|\mathcal{V}'|$ e $|\mathcal{B}|$ o número de vértices do ramo principal e dos ramos laterais, respectivamente. O grafo linear $\mathcal{P}_{|\mathcal{V}'|+|\mathcal{B}|}$ possui $|\mathcal{V}'| + |\mathcal{B}| - 1$ arestas. A partir do grafo $\mathcal{G}_{\text{BDNN}}$, o *Decision Maker* constrói o grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$, com objetivo de modelar os cenários de processamento exclusivamente na nuvem, na borda e com particionamento. Como mostrado na Figura 4.8(c), o grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ possui $2|\mathcal{V}'|+|\mathcal{B}|$ vértices com $3(|\mathcal{V}'|-1)+|\mathcal{B}|+4$ arestas. O número de ramos laterais é menor ou igual ao número de vértices do ramo principal. No pior cenário, sendo $|\mathcal{B}| = |\mathcal{V}'|$, o grafo possui $3|\mathcal{V}'|$ vértices e $4(|\mathcal{V}'| - 1) + 4$ arestas. Portanto, a complexidade computacional do Algoritmo 2, considerando a complexidade do algoritmo de Dijkstra, é de $\mathcal{O}(|\mathcal{V}'|log(|\mathcal{V}'|))$.

A tarefa de Otimização do Particionamento da BranchyNet determina o particionamento ótimo para uma dada configuração de limiar $\boldsymbol{\tau}$. Assim, cada configuração de limiar $\boldsymbol{\tau} \in \mathcal{T}$ está associada a um particionamento ótimo, o qual possui uma acurácia e um tempo de inferência. Dessa forma, dada uma configuração de limiar $\boldsymbol{\tau}$, a complexidade computacional do Algoritmo 1 é de $\mathcal{O}(|\mathcal{B}|^2)$, já que esse algoritmo realiza uma busca em lista a cada interação do laço com $|\mathcal{B}|$ iterações. Portanto, dada uma configuração de limiar $\boldsymbol{\tau}$, a complexidade computacional dessa tarefa, considerando o Algoritmo 1 e 2, é de $\mathcal{O}(|\mathcal{B}|^2 + |\mathcal{V}'|\log(|\mathcal{V}'|))$.

## 4.8 Ajuste da Configuração do Limiar de Entropia

Como visto na Seção 2.3, a configuração do limiar $\boldsymbol{\tau}$ impacta na decisão do particionamento e, consequentemente, afeta o valor esperado da acurácia $\mathbb{E}[Acc|\boldsymbol{\tau}]$ (Algoritmo 1) e do tempo de inferência $\mathbb{E}[T_{inf}(k)]$ (Algoritmo 2). Assim, cada configuração do limiar de entropia $\boldsymbol{\tau}$ está associada a um particionamento ótimo obtido por meio do problema de otimização apresentado na Seção 4.7.

Nesta seção, o objetivo é associar um particionamento ótimo a cada configuração do limiar $\boldsymbol{\tau}$ pertencente ao conjunto de vetores $\mathcal{T}$, gerando múltiplas estratégias de particionamento ótimo. Dessa forma, uma estratégia de particionamento significa uma configuração de limiar associada a um particionamento ótimo, obtido executando os Algoritmos 1 e 2. Para que a tarefa de decisão, detalhada na Seção 4.9, selecione a estratégia de particionamento final. Ou seja, a tarefa de decisão seleciona uma configuração de limiar associada a um particionamento ótimo que equilibre o compromisso entre acurácia e tempo de inferência, baseado no requisito de latência definido pela aplicação.

Com esse objetivo, o Algoritmo 3 percorre o conjunto $\mathcal{T}$ (Linha 2) e, para cada $\boldsymbol{\tau} \in \mathcal{T}$,

executa os Algoritmos 1 (Linha 3) e 2 (Linha 4) apresentados na Seção 4.7. Assim, dada uma configuração de limiar $\boldsymbol{\tau}$, o Algoritmo 3 executa o procedimento $buscaAcuracia$ (Algoritmo 1). Esse algoritmo retorna $\mathbb{E}[Acc|\boldsymbol{\tau}_j]$ e o grafo $\mathcal{G}^*_{BDNN}$ composto pelos laterais do conjunto $\mathcal{B}^*$ (Linha 3). Em seguida, o Algoritmo 3 executa o procedimento $minimizaTempoInferencia$ que obtém o valor esperado do tempo de inferência $\mathbb{E}[T_{inf}]$ e a camada de particionamento $v_s$ (Linha 4). Assim, é possível associar $\mathbb{E}[Acc|\boldsymbol{\tau}_j]$, $\mathbb{E}[T_{inf}]$ e $v_s$ a cada configuração de limiar $\boldsymbol{\tau}_j \in \mathcal{T}$. Para isso, define-se estratégia de particionamento $\varsigma_j$ como uma terna dada por $\varsigma_j = (\mathbb{E}[T_{inf}], \mathbb{E}[Acc|\boldsymbol{\tau}_j], v_s)$. Por fim, o Algoritmo 3 armazena cada uma das estratégias de particionamento no conjunto $\mathcal{S}$ (Linha 5). Até esse ponto, considerando os Algoritmos de 1 a 3, a complexidade computacional é de $\mathcal{O}(|\mathcal{T}| \cdot (|\mathcal{B}|^2 + |\mathcal{V}'| \log(|\mathcal{V}'|))$.

**Algoritmo 3** Algoritmo para geração de múltiplas estratégias de particionamento

**Entrada:**

$\mathcal{G}'_{\text{BDNN}}$: Grafo de particionamento associado à BranchyNet
$\mathcal{T}$: conjunto das configurações de vetor limiar de entropia predefinidas
$\boldsymbol{p}_j = [p_{1j}, \ldots, p_{|\mathcal{B}|j}]$: probabilidade de classificar amostras no ramos dado o limiar de entropia $\tau_j \in \mathcal{T}$
$\boldsymbol{Acc} = [acc_1, \cdots, acc_{|\mathcal{B}|}]$: conjunto de acurácias dos ramos laterais

**Saída:**

$\mathcal{S}$: conjunto de estratégia de particionamento

1: $\mathcal{S} = \emptyset$,
2: **para** $j = 1, \cdots, |\mathcal{T}|$ **faça**
3: $\quad \mathcal{G}^*_{\text{BDNN}} \leftarrow buscaAcuracia(\mathcal{G}_{\text{BDNN}}, \boldsymbol{\tau}_j, \boldsymbol{Acc}, \boldsymbol{p}_j)$;
4: $\quad [\mathbb{E}[T_{inf}], v_s] \leftarrow minimizaTempoInferencia(\mathcal{G}^*_{\text{BDNN}})$;
5: $\quad \mathcal{S}.append([\mathbb{E}[T_{inf}], \mathbb{E}[Acc|\tau_j], v_s])$;

**retorna** $\mathcal{S}$;

Uma vez executado, o Algoritmo 3 obtém todas as possíveis estratégias de particionamento para uma determinada taxa de envio $B$. Contudo, a taxa de envio é um parâmetro dinâmico. Uma taxa de envio diferente resulta em tempo de comunicação diferente, e, consequentemente, impacta nas estratégias de particionamento. Assim, considerando que há um histórico de taxa de envio $\boldsymbol{B}$, é possível associar um conjunto de estratégias de particionamento $\mathcal{S}$ a cada uma das taxas de envio pertencentes ao histórico $\boldsymbol{B}$, executando o Algoritmo 3. Esse procedimento resulta em uma matriz de estratégia de particionamento $\boldsymbol{S}$ com $|\boldsymbol{B}|$ linhas e $|\mathcal{T}|$ colunas. Nessa matriz, cada linha corresponde a uma taxa de envio diferente e cada coluna refere-se a uma configuração do limiar de entropia. Assim, a $i$-ésima linha da matriz $\boldsymbol{S}$ corresponde a uma taxa de envio $B_i$ e cada elemento dessa linha refere-se a uma configuração diferente de limiar de entropia. Portanto, cada elemento $(\boldsymbol{S})_{ij}$ dessa matriz corresponde à estratégia de particionamento $\varsigma_{ij} = (\mathbb{E}[T_{inf}], \mathbb{E}[Acc|\tau_i], v_s)$ com taxa de envio $B_i$ e configuração de limiar $\boldsymbol{\tau}_j$.

## 4.9 Decisão

A partir das múltiplas estratégias de particionamento fornecidas pela tarefa anterior, a tarefa de Decisão é responsável por escolher aquela que equilibra o compromisso entre acurácia e tempo de inferência baseado no requisito de latência máxima $t_{\text{max}}$ definido pela aplicação.

A tarefa de Decisão divide-se em três etapas: (1) selecionar a estratégia de particionamento ótima; (2) enviar a informação da camada de particionamento selecionada para a nuvem; (3) atualização da probabilidade de uma amostra ser classificada nos ramos laterais durante o processo de inferência.

Em relação à primeira etapa, o Algoritmo 4 descreve a seleção da estratégia de particionamento que equilibra o compromisso entre o valor esperado da acurácia $\mathbb{E}[Acc \mid \boldsymbol{\tau}]$ e o valor esperado do tempo de inferência $\mathbb{E}[T_{inf}]$, conforme o requisito de latência máxima $t_{\text{max}}$ fornecido pela aplicação. Para isso, dada uma taxa de envio $B_0$ obtida pelo *Update Box* monitorando a rede, o Algoritmo 4 busca uma taxa de envio $B_i \in \boldsymbol{B}$ que mais se aproxima desse valor $B$ (Linha 1).

Em seguida, seleciona-se a estratégia que maximiza a acurácia e que satisfaz o requisito de latência $t_{\text{max}}$ (Linha 2-3). Em caso de haver múltiplas estratégias de particionamento que possuam a mesma acurácia e cumpram o requisito de latência, escolhe-se aquela com menor tempo de inferência. Assim, a etapa de decisão determina a estratégia ótima, o que consequentemente determina, também, tanto o valor esperado da acurácia e do tempo de inferência quanto a camada de particionamento ótima. Considerando que a matriz de estratégias $\boldsymbol{S}$ já foi totalmente determinada durante a inicialização do POPEX, periodicamente, executa-se apenas a etapa de Decisão. Portanto, o componente *Decision Maker* divide-se em duas fases. Uma fase *offline* executada durante a inicialização do sistema POPEX, que determina a matriz de estratégias $\boldsymbol{S}$. E uma fase *online* executada periodicamente, que corresponde à tarefa de Decisão. Como visto anteriormente, a fase *offline*, executada durante a inicialização do sistema, possui complexidade computacional de $\mathcal{O}(|\boldsymbol{B}|\cdot|\mathcal{T}|\cdot(|\mathcal{B}|^2+|\mathcal{V}'|\log(|\mathcal{V}'|))$. Entretanto, Kaya *et al.* propõem posicionar os ramos laterais apenas em determinadas posições [42]. Por exemplo, esse trabalho posicionam os ramos apenas nas camadas próximas de 15%, 30%, 90% do custo total de inferência da BranchyNet, no qual o custo corresponde ao número de FLOPs (operações em ponto flutuante). Dessa forma, o número de ramos laterais é constante, não importa o tamanho ou a complexidade da DNN. Portanto, a complexidade computacional da fase *offline* é de $\mathcal{O}(|\boldsymbol{B}|\cdot|\mathcal{T}|(|\mathcal{V}'|\log(|\mathcal{V}'|))$. Por outro lado, a fase *online*, executada periodicamente, possui complexidade computacional de $\mathcal{O}(|\mathcal{T}|)$, conforme o Algoritmo 4.

Em relação à segunda tarefa, o sistema POPEX utiliza o *framework* Flask [5] para implementar um servidor web na nuvem, que lhe permite receber requisições HTTP do cliente

[5] `https://flask.palletsprojects.com/en/1.1.x/`

**Algoritmo 4** Seleção da estratégia de particionamento

**Entrada:** $\boldsymbol{S}$: matriz de estratégias de particionamento
$\boldsymbol{B}$: histórico de taxa de envio
$B$: taxa de envio atual extraída pelo *Background*
**Saída:** $(\boldsymbol{S})_{idx,j}$: estratégia de particionamento selecionada

1: $idx \leftarrow \arg\min_{i \in \{1,\cdots,|\boldsymbol{B}|\}} |B_i - B_0|$;
2: $\mathcal{R} \leftarrow$ lista de índices i que satisfaz $\boldsymbol{S}_{idx,*} \leq t_{\max}$;
3: $j \leftarrow \arg\max_{r \in \mathcal{R}} \boldsymbol{S}_{idx,r}$;
4: **retorna** $(\boldsymbol{S})_{idx,j}$;

implementado no dispositivo em borda. Isso possibilita enviar a informação da camada de particionamento $v_s$ e seus dados de saída por meio de uma requisição POST HTTP. Assim, sabendo a camada de particionamento, a nuvem pode processar todas as camadas posteriores à camada $v_s$, executando, na prática, o particionamento.

Durante a inicialização do sistema, o POPEX executa, primeiramente, as tarefas de extração dos parâmetros estáticos por meio do componente *Background*. Em seguida, também durante a inicialização do sistema, o POPEX executa as seguintes tarefas do componente *Decision Maker*: estimação do tempo de inferência, construção do grafo de particionamento da BranchyNet, otimização e ajuste da configuração do limiar de entropia. Periodicamente, o componente *Update Box* obtém a taxa de envio da infraestrutura de rede entre a borda e a nuvem. Durante a execução do sistema, quando o dispositivo de borda recebe uma imagem, esse executa apenas a etapa de decisão. Dessa forma, após receber uma nova imagem, o sistema POPEX executa o Algoritmo 4, usando as informações obtidas pelas tarefas executadas na inicialização do sistema.

# Capítulo 5

# Experimentos do POPEX

O sistema POPEX e sua respectiva documentação estão disponíveis em um repositório no GitHub [1]. Os experimentos apresentados tanto neste capítulo quanto nos demais capítulos desta dissertação avaliam especialmente o componente *Decision Maker*, pois o foco principal é a otimização proposta neste componente. Além disso, os experimentos consideram um cenário estático, com o objetivo de avaliar, em detalhes, o impacto de diversos fatores nas decisões de particionamento obtidas pelo problema de otimização proposto. Dessa forma, esses experimentos consideram que tanto a taxa de envio quanto a probabilidade de classificação das amostras nos ramos laterais são fixas. Assim, esses experimentos não consideram que a taxa de envio da infraestrutura de rede varie com o tempo, tampouco a probabilidade de classificação das amostras nos ramos laterais.

Neste capítulo, os experimentos apresentam uma análise de sensibilidade, para avaliar os impactos dos parâmetros de entrada descritos na Seção 4.1.1. Em seguida, analisa-se a escolha da camada de particionamento sob diversas capacidades de processamento do dispositivo em borda e do servidor em nuvem. Por fim, analisa-se o compromisso entre acurácia de classificação e o tempo de inferência para diferentes dispositivos de borda e taxas de envio.

## 5.1 Impacto da Probabilidade de Classificação no Tempo de Inferência

Esta seção avalia os impactos da probabilidade de classificação nos ramos laterais e das capacidades de processamento entre borda e nuvem no tempo de inferência obtido pelo problema de otimização, descrito na Seção 4.7. Para isso, implementa-se uma rede B-AlexNet, detalhada na Seção 2.3, com apenas um ramo lateral inserido após a primeira camada convolucional. A posição do ramo principal é escolhida para evitar processamento

[1] Disponível em `https://github.com/pachecobeto95/POPEX`, registrado no INPI sob o número BR512020001239-6

desnecessário da borda. O conjunto de treinamento é utilizado para treinar os parâmetros da DNN, o que a permite classificar corretamente as imagens, enquanto o conjunto de validação é utilizado para avaliar as escolhas dos hiperparâmetros utilizados. Por fim, o conjunto de testes é usado para avaliar o desempenho da DNN e a análise desta seção. Além disso, os experimentos apresentados nesta dissertação são escritos em Python 3 e utilizam o *framework* Pytorch [2] para treinar e avaliar as BranchyNets. O Pytorch é uma solução de código aberto desenvolvida pelo Facebook para facilitar o treinamento e implementação de DNNs. Os experimentos apresentados nesta seção utilizam os parâmetros estáticos tempo de processamento na borda ($\boldsymbol{t_e}$) e na nuvem ($\boldsymbol{t_c}$), considerando uma dada taxa de envio. Esses parâmetros estáticos independem do *dataset* utilizado, já que os tempos de processamento e o tamanho dos dados de saída de cada camada dependem apenas da arquitetura da DNN. O *dataset* influencia a probabilidade de classificação nos ramos laterais e a acurácia da rede neural. Entretanto, nos experimentos apresentados nesta seção, a probabilidade de classificação nos ramos laterais é um parâmetro conhecido para justamente realizar a análise de sensibilidade nesse experimento. Os experimentos relacionados à acurácia da BranchyNet são apresentados na Seção 5.3.

Para realizar a análise o primeiro passo é obter os parâmetros de entrada. O componente *Background* (Seção 4.1) é executado para obter o grafo $\mathcal{G}_{\text{BDNN}}$ baseado na arquitetura da BranchyNet e a quantidade de dados de saída $\alpha_i$ de cada camada da B-AlexNet. A partir dos parâmetros extraídos pelo *Background*, o componente *Decision Maker* constrói o grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ ilustrado na Figura 4.8(c). Conforme apresentado na Seção 4.6, o grafo de particionamento $\mathcal{G}'_{\text{BDNN}}$ permite modelar o problema de particionamento em um problema de caminho mais curto. Para avaliar o impacto dos parâmetros estáticos no tempo de inferência, os experimentos apresentados nesta seção executam o Algoritmo 2 executado no componente *Decision Maker*. Esse algoritmo permite obter o particionamento ótimo que minimiza o tempo de inferência. Em seguida, os experimentos desta seção executam esse algoritmo, variando a probabilidade de classificação no ramo lateral e o tempo de processamento na borda.

Esses experimentos consideram que o dispositivo de borda utiliza uma determinada tecnologia sem-fio para enviar dados à nuvem. Além disso, assume-se que o gargalo da comunicação é a rede de acesso da borda. Assim, no primeiro momento, o experimento utiliza as taxas de envio fixas, que foram derivadas das taxas de 3G, 4G e Wi-Fi em [16]. Os valores das taxas de envio são apresentados na Tabela 5.1.

Para obter o parâmetro $\boldsymbol{t_c}$ referente ao tempo de processamento da nuvem, mede-se o tempo necessário para processar cada camada da B-Alexnet, usando o *hardware* disponível no Google Colaboratory [3]. A plataforma Google Colaboratory disponibiliza recursos computacionais ociosos. Durante a execução desse experimento, a plataforma disponibili-

[2] `https://pytorch.org/`

[3] `https://colab.research.google.com`

| | Taxa de Envio (Mbps) |
|---|---|
| 3G | 1,1 |
| 4G | 5,5 |
| Wi-Fi | 18,8 |

Tabela 5.1: Taxa de envio média para diferentes tecnologias de comunicação sem fio

zou uma GPU NVIDIA Tesla K8 com 12 GB de memória VRAM e um processador Intel 2-core Xeon(R)@ 2.20GHz. Em relação ao tempo de processamento na borda, define-se o tempo de processamento de cada camada como sendo

$$t_i^e = \gamma \cdot t_i^c, \tag{5.1}$$

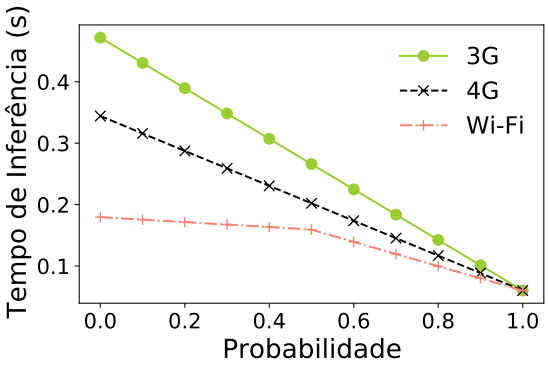
onde $\gamma \in \mathbb{N}_{>1}$ é um fator multiplicativo que indica a razão entre o processamento da borda e da nuvem para cada camada. Dessa forma, o $\gamma$ permite abranger diferentes tipos de *hardware* como dispositivos de borda. Por exemplo, os módulos Jetson TX2[4] desenvolvidos pela NVIDIA podem representar um dispositivo de borda com alto poder de processamento e, portanto, baixo $\gamma$. Por outro lado, o Raspberry Pi[5] representa um dispositivo em borda com recursos limitados, apresentando um alto fator $\gamma$.

A Figura 5.1 mostra como a probabilidade de classificação nos ramos laterais impacta o valor esperado do tempo de inferência, dado pela Equação 4.7, para três fatores de processamento: 10, 100 e 1000. Esses resultados são obtidos baseados na solução do problema de otimização, que visa minimizar o tempo de inferência usando o Algoritmo 2 para determinar o caminho mais curto e, consequentemente, o particionamento. A partir do resultado obtido pelo Algoritmo 2, variam-se as probabilidades de uma amostra ser classificada no ramo lateral.

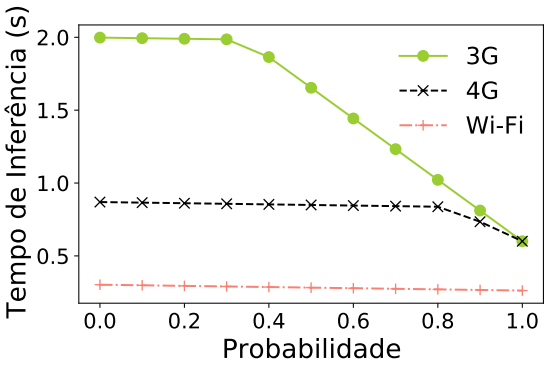
Considerando um determinado fator de processamento $\gamma$, as Figuras 5.1(a) e 5.1(b) mostram que tecnologias de comunicação sem fio com taxas de envio menores são mais afetadas pela probabilidade. Por exemplo, os resultados utilizando taxa de envio de 1,1 Mbps, na Figura 5.1(a), mostram que o tempo de inferência reduz em 87.27%, se o caso com a probabilidade igual a zero for comparado com o caso em que a probabilidade é 1. Por outro lado, essa diferença é de 82.98% e 70% utilizando redes com taxa de envio de 5,5 Mbps e 18,8 Mbps, respectivamente. Isso ocorre pois, quando a probabilidade de classificação nos ramos laterais é baixa, o problema de particionamento escolhe processar parte das camadas na nuvem. Consequentemente, o tempo de inferência correspondente a taxa de envio 1,1 Mbps é superior aos demais, já que possui menor taxa de envio. Por outro lado, a medida que a probabilidade de classificação aumenta, mais amostras são classificadas no ramo lateral, consequentemente mais amostras são classificadas localmente. Assim, essa Figura também mostra que, quando a probabilidade é igual a 1, o tempo de

[4] `https://developer.nvidia.com/embedded/jetson-tx2`

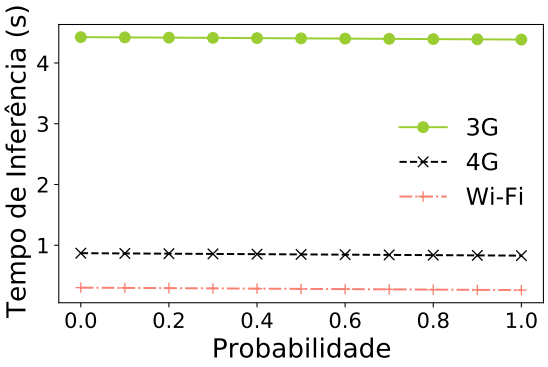
[5] `www.raspberrypi.org`




(a) Fator de Processamento 10.




(b) Fator de Processamento 100.




(c) Fator de Processamento 1000.

Figura 5.1: Tempo de inferência de acordo com a probabilidade de classificação de uma amostra para diferentes tecnologias sem-fio e fatores de processamento $\gamma$.

inferência é o mesmo independentemente da tecnologia de rede utilizada. Esse resultado é esperado pois, nesse caso, todas as amostras são classificadas no ramo lateral. Nesse caso, o tempo de inferência independe da taxa de envio da tecnologia de comunicação utilizada. É possível notar que, para os resultados das Figuras 5.1(a) e 5.1(b), o eixo y permanece constante para um dado intervalo de probabilidade. Esse comportamento é explicado pois, nesse intervalo de probabilidade, o problema de otimização escolhe processar todas as camadas exclusivamente na nuvem, já que grande parcela de amostras não são classificadas no ramo lateral. À medida que mais amostras são classificadas no ramo lateral, o tempo de inferência decresce e o problema de particionamento não escolhe apenas processar as camadas na nuvem, como será mostrado na Seção 5.2.

Em todos os gráficos da Figura 5.1 há um intervalo de probabilidade, no qual o tempo de inferência permanece constante. Isso significa que, nesse intervalo de probabilidade, o problema de otimização escolhe apenas a estratégia de particionamento na qual todas as camadas são processadas na nuvem. Consequentemente, como o processamento exclusivamente na nuvem não possui ramos laterais, o tempo de inferência não é afetado pela probabilidade, permanecendo constante. Após um determinado valor de probabilidade, que depende da tecnologia de rede e do fator de processamento, o problema de otimização

começa a escolher soluções de particionamento em que o dispositivo de borda participa. A partir desse valor, a probabilidade começa a afetar o tempo de inferência. Por exemplo, usando taxa de envio de 1,1 Mbps, o tempo de inferência na Figura 5.1(b) apenas começa a decair para probabilidades superiores a 0,3. Para taxas de envio de 5,5 Mbps, esse valor é 0,8. O valor mais alto para redes com 5,5 Mbps justifica-se pela sua maior taxa de envio, tornando mais vantajoso enviar a imagem bruta à nuvem por um intervalo maior de probabilidades.

Finalmente, a Figura 5.1(c) mostra uma situação extrema em que a probabilidade não afeta o tempo de inferência. Esse comportamento ocorre porque a borda tem baixo poder de processamento ($\gamma = 1000$) e, portanto, é sempre preferível enviar a imagem bruta para que seja processada na nuvem do que processar. Esse comportamento também ocorre, na Figura 5.1(b), usando redes com taxa de envio de 18,8 Mbps, porém, agora, devido a alta taxa de envio.

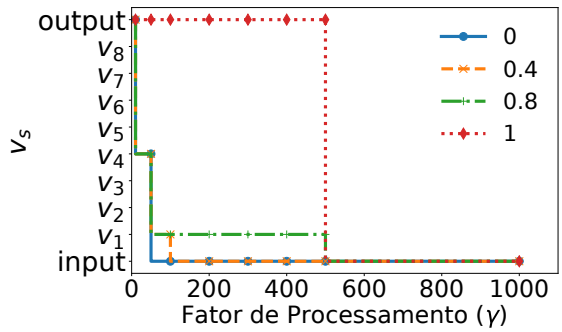
## 5.2 Análise da Camada de Particionamento

Usando o mesmo cenário descrito da Seção 5.1 e apresentado na Figura 5.1, varia-se o fator de processamento $\gamma$ e verifica-se qual é a camada de particionamento escolhida. Quando a camada de particionamento é *input*, isso significa que todas as camadas da DNN são processadas na nuvem. Por outro lado, camada de particionamento sendo *output* significa que todas as camadas da DNN são processadas na borda.

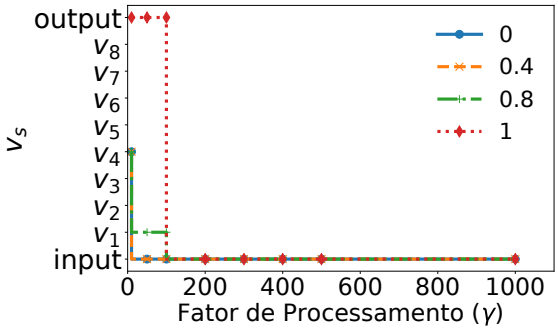
As Figuras 5.2(a) e 5.2(b) mostram a camada escolhida para diversos fatores de processamento usando taxas de envio de 1,1 e 5,5 Mbps, respectivamente. Os experimentos dessa seção não apresentam os resultados usando redes com 18,8 Mbps, pois nesse caso as camadas são processadas exclusivamente na borda ou exclusivamente na nuvem, devido a alta taxa de envio.

Nas Figuras 5.2(a) e 5.2(b), cada curva representa uma dada probabilidade de classificar a amostra no ramo lateral. A Figura 5.2(a) mostra que, à medida que o fator de processamento aumenta, a camada de particionamento escolhida tende a se aproximar da camada de entrada. Por exemplo, assumindo uma probabilidade $p = 0,8$, a Figura 5.2(a) mostra que, a partir de $\gamma = 500$, a camada de particionamento muda de $v_1$ a *input*. Isso significa que, a partir desse $\gamma$, o processamento é realizado exclusivamente na nuvem. Esse comportamento é esperado, já que a borda, com baixo poder de processamento, leva o problema a escolher processar as camadas na nuvem.

Quando compara-se a Figura 5.2(b) com a Figura 5.2(a), nota-se que, para a rede com taxa de envio de 5,5 Mbps, o problema começa a escolher a estratégia de processamento apenas na nuvem para um fator de processamento mais baixo (isto é, poder de processamento alto na borda). Isso ratifica a tendência observada na Figura 5.1, na qual, para altas taxas de envio, o problema possui uma maior tendência a escolher a estratégia de processa-




(a) Análise usando taxa de envio de 1,1 Mbps.




(b) Análise usando taxa de envio de 5,5 Mbps.

Figura 5.2: Camada de Particionamento para diferentes fatores de processamento.

mento exclusivamente na nuvem. Por fim, a Figura 5.2 também confirma o comportamento da Figura 5.1, em que a probabilidade influencia a escolha da camada de particionamento e, portanto, impacta no tempo de inferência.

Em linhas gerais, a Figura 5.1 analisa o quanto a probabilidade $p_Y(k)$ afeta o tempo de inferência com diferentes tecnologias de comunicação sem-fio e mostra que, de fato, as probabilidades de classificação cada vez maiores reduzem o tempo de inferência. Em seguida, na Figura 5.2, verifica-se que tanto o fator de processamento $\gamma$ quanto a probabilidade de classificação têm efeito na escolha da estratégia de particionamento. Em outras palavras, as análises anteriores justificam e validam a proposta de particionamento de BranchyNet como uma solução para reduzir o tempo de inferência e, consequentemente, uma alternativa para viabilizar aplicações sensíveis à latência.

A probabilidade de classificar uma amostra no ramo lateral é relacionada ao modelo da DNN (isto é, sua arquitetura e os hiperparâmetros com os quais é treinado) e a aspectos inerentes às amostras recebidas. Consequentemente, esses dois fatores alteram a decisão de particionamento. O Capítulo 6 desta dissertação mostra um estudo de caso referente ao modelo da DNN, enquanto o Capítulo 7 mostra um estudo de caso da influência da amostra processada. Mais especificamente, o Capítulo 7 analisa como a distorção da imagem afeta o particionamento. O modelo empregado no Capítulo 7 é utilizado nesta seção para mostrar como a probabilidade de classificação da amostra pode ser impactada pela distorção da imagem. Para tal, treina-se a rede B-AlexNet usando um conjunto de dados composto

de imagens de cães e gatos, em diferentes ambientes, sem nenhuma distorção [57]. Uma vez treinada, aplica-se um lote com 48 amostras com diferentes níveis de *blur* Gaussiano [57]. Esse experimento implementa filtros Gaussianos com dimensões 5, 15 e 65 para representar imagens com nível de distorção baixo, intermediário e alto, respectivamente. Essas amostras são utilizadas para que a B-AlexNet treinada identifique uma dada amostra como imagem de cão ou gato. A Figura 5.3 mostra a probabilidade de classificar uma amostra de entrada de acordo com o limiar de entropia do ramo lateral, para imagens com diversos níveis de distorção. Essa figura mostra que um nível de distorção mais elevado diminui a probabilidade de classificar a amostra no ramo lateral. Isso ocorre pois imagens com alto nível de distorção tendem a apresentar uma maior incerteza no processo de inferência, resultando em uma probabilidade menor de ser classificada no ramo lateral. O Capítulo 7 mostra, em mais detalhes, o impacto da distorção da imagem nas decisões de particionamento.

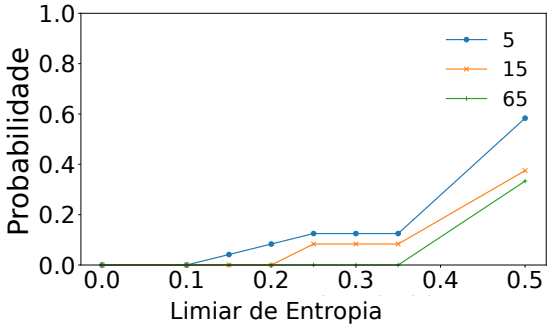



Figura 5.3: Probabilidade do ramo lateral classificar uma amostra sob diferentes níveis de distorção.

## 5.3 Avaliação da Etapa de Decisão

A etapa de Decisão, executada no componente *Decision Maker*, lida com o compromisso entre acurácia e tempo de inferência na escolha da estratégia de particionamento. Esta seção visa analisar esse compromisso. Para isso, implementa-se uma B-AlexNet com quatro ramos laterais, inseridos após cada camada convolucional.

A BranchyNet é treinada e avaliada para um problema de classificação de imagens em 10 classes, usando o *dataset* CIFAR-10 [58], que consiste de 60.000 imagens coloridas de 32x32 *pixels*. Nesse experimento, utilizam-se 40.000 imagens para treinamento da BranchyNet, 10.000 para a validação e 10.000 para teste. O conjunto de treinamento é utilizado para treinar os parâmetros da DNN, o que a permite classificar corretamente as imagens, enquanto o conjunto de validação é utilizado para avaliar as escolhas dos hiperparâmetros utilizados. Por fim, o conjunto de testes é usado para avaliar o desempenho da DNN e a análise desta seção. Os ramos laterais são treinados conforme a abordagem apresentada na

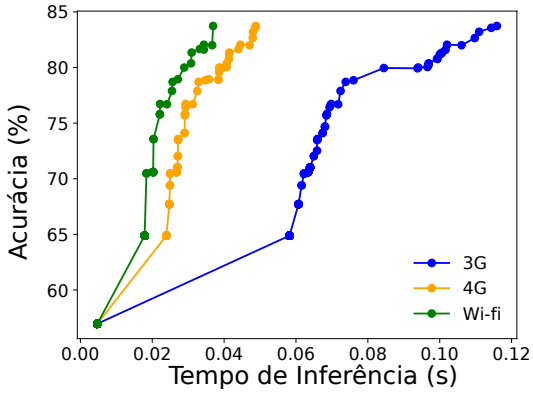
Seção 2.3. Essa abordagem define uma única função de perda, como descrito pela Equação 2.3, a qual é composta pela combinação linear da função de perda de cada ramo. As constantes $\alpha_i$, na Equação 2.3, são iguais a 1, o que significa que não há prioridade de um determinado ramo lateral sobre os demais.

Como nas análises anteriores, utiliza-se o fator de processamento $\gamma$ para representar dispositivos com diferentes capacidades de processamento na borda. Os fatores de processamento utilizados são 10, 50 e 100. O experimento consiste em escolher uma estratégia de particionamento, dada uma taxa de envio entre o dispositivo de borda e a nuvem. Em seguida, utiliza-se o Algoritmo 3 para associar a estratégia de particionamento ótima para cada configuração do limiar de entropia. Dessa forma, por meio do Algoritmo 3, associa-se a camada de particionamento, valor esperado da acurácia e do tempo de inferência a cada configuração dos limiares de entropia $\boldsymbol{\tau}$ da DNN. Os valores de limiar de entropia $\tau_i$ em cada ramo lateral são 0,01, 0,05, 0,1, 0,5, e 1,0.

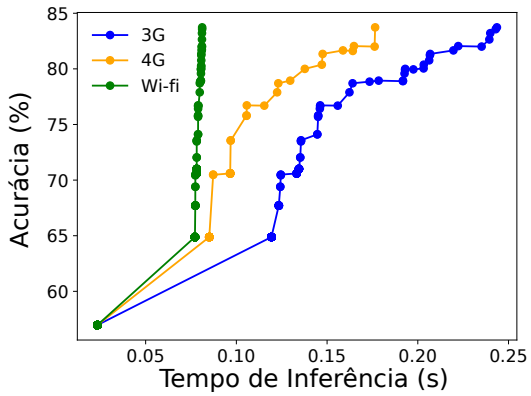
A Figura 5.4 mostra o compromisso entre a acurácia de classificação e o valor esperado do tempo de inferência para três fatores de processamento: 10, 50 e 100. Cada ponto nessa figura corresponde a um elemento da matriz de estratégias $\boldsymbol{S}$, definida na Seção 4.8. Os pontos dessa figura são obtidos executando o Algoritmo 3. Para um dado fator de processamento, cada curva corresponde a uma taxa de envio fixa de 1,1, 5,5 e 18,8 Mbps, cujas taxas de envio está relacionadas à taxa de envio média de diferentes tecnologias de comunicação sem-fio, como apresentada na Tabela 5.1.

A Figura 5.4 mostra que, para todos os cenários, a redução do tempo de inferência também causa uma redução da acurácia. Como já visto, dado que cada ponto das curvas está associado a uma configuração do limiar de entropia, isso significa que é possível utilizar esse parâmetro para ajustar o compromisso entre acurácia e tempo de inferência. Por exemplo, quando o limiar de entropia do primeiro ramo lateral é ajustado para valores altos (p.ex., 0,9), esse ramo é capaz de classificar uma grande parcela de amostras do conjunto de dados, inclusive as pouco confiáveis, reduzindo a acurácia e também o tempo de inferência. Por outro lado, quando o limiar de entropia do primeiro ramo é ajustado para valores baixos (p.ex., como 0,1), somente amostras de entrada altamente confiáveis podem ser classificadas nesse ramo, aumentando a acurácia e também o tempo de inferência. O tempo aumenta pois a maioria das amostra precisam ser processadas pelas próximas camadas. Portanto, os resultados da Figura 5.4 mostram um claro compromisso entre acurácia e tempo de inferência em DNNs com saídas antecipadas.

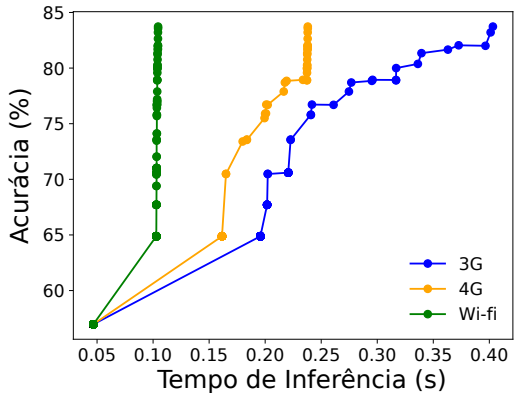
A escolha do particionamento para lidar com o compromisso entre acurácia e tempo de inferência depende do requisito de latência máxima definido pela aplicação. Por exemplo, considerando a taxa de envio de 1,1 Mbps e o fator de processamento de 10, a Figura 5.4(a) mostra que há uma configuração de limiar que permite atingir a mais alta acurácia, aproximadamente 83%, enquanto o valor esperado do tempo de inferência é de 0,12 s. Portanto, essa configuração do limiar de entropia classifica amostras de entrada com alta acurácia.




(a) Fator de Processamento de 10.




(b) Fator de Processamento de 50.




(c) Fator de Processamento de 100.

Figura 5.4: Múltiplas curvas de estratégias de particionamento para diversas taxas de envio.

No entanto, essa configuração não é viável para aplicações que exigem um tempo de inferência inferior a 0,12 s, como o caso de aplicações de assistência cognitiva, que demandam tempo de inferência menor do que 0,10 s [12]. Nesse caso, o sistema POPEX escolheria a configuração de limiar que cumprisse o requisito de latência de 0,10 s, reduzindo a acurácia para aproximadamente 80%. Nesse caso, não representa uma queda drástica de acurácia.

Por outro lado, considerando fator de processamento de 50 e taxa de envio de 5,5 Mbps, a Figura 5.4(b) mostra que o sistema POPEX escolheria uma configuração de limiar que resultasse em acurácia de 70% para cumprir o requisito de latência de 0,10 s da aplicação.

A Figura 5.4 mostra que cada fator de processamento apresenta um comportamento verticalmente assintótico para altas taxas de envio. Isso é ainda mais evidente para fatores de processamento altos, como os considerados na Figura 5.4(c). O comportamento verticalmente assintótico indica que o dado de entrada bruto é enviado diretamente à nuvem sem que nenhuma camada seja processada no dispositivo de borda. Esses resultados ocorrem porque conforme a taxa de envio aumenta e a capacidade de processamento dos dispositivos de borda diminui, isto é, seu fator de processamento aumenta, é mais rápido enviar dados brutos à nuvem do que processar alguma das camadas da BranchyNet localmente na borda. Já que a nuvem tem baixo tempo de processamento, os tempos de inferência serão sempre muito próximos para diversos limiares de entropia, explicando o comportamento vertical. Quando compara-se a Figura 5.4(a) com a Figura 5.4(c), é possível notar que o dispositivo em borda com baixa capacidade de processamento acelera a tendência da curva a apresentar um comportamento verticalmente assintótico.

Finalmente, avalia-se os resultados considerando um requisito de latência definido pela aplicação. Isso permite selecionar a estratégia de particionamento que maximiza a acurácia de classificação e também cumpre esse requisito de latência. Por exemplo, em aplicações de assistência cognitiva, como já mencionado, o requisito de latência é de 0,10 s. Portanto, na Figura 5.4(a), nota-se que, para a taxa de envio de 1,1 Mbps, a estratégia de particionamento escolhida deve ser aquela que atinge a acurácia de 75%. Nessa figura, para outras taxas de envio, todos os pontos nas curvas de estratégias satisfazem o requisito de latência, portanto, a estratégia de particionamento deve ser aquela com a maior acurácia.

Nos próximos capítulos, analisam-se fatores relacionados ao modelo da DNN e às amostras de entrada e seus impactos nas decisões de particionamento.

# Capítulo 6

# Impacto da Calibração no Particionamento de DNNs

Em diversas aplicações de visão computacional, as DNNs são responsáveis por decisões complexas e importantes, como, por exemplo, detectar situações perigosas em veículos inteligentes. Nesse tipo de aplicação, a DNN não deve ser somente acurada, mas também fornecer uma métrica de classificação de incerteza para quantificar a probabilidade de a classificação estar correta [59]. Na aplicação de veículos autônomos com computação na borda, por exemplo, a DNN deve ser capaz de detectar a presença de objetos e pedestres na estrada com alta confiança. Caso contrário, o veículo deve transferir a tarefa de detecção para sensores auxiliares instalados no veículo, como os baseados em tecnologia LIDAR [60], ou a sensores posicionados ao longo da estrada [61]. O próprio dispositivo de borda pode alterar suas técnicas de aprendizado de máquina (p.ex., *forest random*, *k-nearest*) [61] no caso de a classificação da DNN não ser confiável o suficiente. Portanto, a DNN deve prover uma medida de confiança bem calibrada de sua classificação para decidir se a classificação é confiável ou se deve utilizar os demais sensores, ou, até mesmo, executar outros modelos auxiliares no dispositivo em borda. No caso de DNNs com saídas antecipadas, os ramos laterais superestimam a confiança de sua classificação. Consequentemente, os ramos laterais podem decidir de forma errônea em classificar antecipadamente uma determinada amostra, como será mostrado na Seção 6.1.3.

As DNNs mais recentes têm apresentado um significativo aumento da acurácia quando comparadas às primeiras DNNs apresentadas há uma década. Contudo, Guo *et al.* [17] demonstram que há problema de calibração na confiança fornecida por essas DNNs. Em outras palavras, a probabilidade associada à classe inferida, nessas DNNs, não reflete verdadeiramente a probabilidade de a amostra de entrada ser de uma determinada classe. Guo *et al.* mostram que as DNNs modernas superestimam sua acurácia, o que representa um problema grave, especialmente em aplicações críticas. Então, há diversos métodos de calibração desenvolvidos com intuito de calibrar a DNN e, assim, fornecer uma melhor medida de confiança de sua classificação. Guo *et al.* concluem que o método de calibra-

ção *Temperature Scaling* é o mais efetivo para calibrar DNNs [17]. Entretanto, Guo *et al.* não abordam ou analisam o problema de calibração usando DNNs com saídas antecipadas e suas particularidades, como a configuração dos limiares de entropia.

O principal objetivo deste capítulo é analisar o problema de calibração em DNNs com saídas antecipadas e como isso afeta o seu particionamento. Para isso, esta dissertação propõe avaliar o erro de calibração em DNNs com saídas antecipadas, em função do posicionamento dos ramos laterais e do limiar de entropia de cada ramo. Essa métrica de calibração calcula a diferença média entre a acurácia da BranchyNet e sua confiança na classificação de amostras. A confiança na classificação de uma determinada amostra significa a probabilidade dessa amostra ser classificada corretamente. Dessa forma, um ramo lateral perfeitamente calibrado é aquele em que a confiança corresponda exatamente à acurácia. Por exemplo, supondo que a BranchyNet classifique um conjunto de 100 amostras e cada uma das amostras obtenha uma confiança de 0,8. Considerando uma BranchyNet perfeitamente calibrada, espera-se que 80% dessas amostras sejam classificadas corretamente. Os detalhes sobre a métrica ECE (*Expected Calibration Error*) são apresentados no Apêndice A.

Após medir o erro de calibração da BranchyNet com a métrica ECE, é necessário calibrá-la. O objetivo dos métodos de calibração é gerar uma nova confiança na classificação que se aproxime mais da acurácia obtida pela DNN. Então, este capítulo aplica o método de calibração *Temperature Scaling*, já que Guo *et al.* mostraram que esse método é mais efetivo para diversas arquiteturas de DNNs e aplicações de visão computacional [17]. Em seguida, avalia-se como o método *Temperature Scaling* pode impactar o número de amostras classificadas antecipadamente nos ramos laterais e, consequentemente, o tempo de inferência e a acurácia.

## 6.1 Análise da calibração de DNN

A análise apresentada nesta seção avalia o erro de calibração, usando a métrica ECE, em cada um dos ramos laterais, em função de sua posição na BranchyNet e seu limiar de entropia. Em seguida, avalia-se o impacto do método de calibração em DNNs com saídas antecipadas. Este trabalho utiliza a mesma configuração da Seção 5.3, mas agora a rede B-AlexNet possui quatro ramos inseridos nas camadas intermediárias, com intuito de avaliar os efeitos dos limiares de entropia de acordo com a posição dos ramos. Para cada um desses ramos laterais, escolhem-se valores de limiar de entropia $\tau_i$. Os valores do limiar de entropia utilizados nesses experimentos variam de 0,025 a 0,1 com passo 0,025, e depois de 0,1 a 1,0 com passo de 0,1. Cada análise é detalhada a seguir.

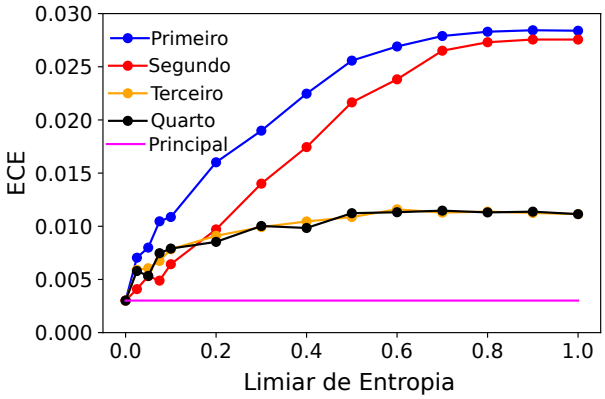



Figura 6.1: Os erros de calibração usando B-AlexNet para diferentes posições dos ramos laterais.

### 6.1.1 Calibração dos Ramos Laterais

A Figura 6.1 mostra o erro de calibração, usando a métrica ECE, para os ramos laterais da B-AlexNet, bem como para o seu ramo principal. Esses resultados são obtidos medindo o efeito do ECE em cada um dos ramos, variando seus limiares de entropia. Para analisar o $i$-ésimo ramo lateral, ajusta-se o limiar de entropia de todos os outros ramos laterais como zero e varia-se o limiar $\tau_i$ do respectivo ramo $i$. Nesse sentido, as amostras serão classificadas nos ramos em análise, isto é, no $i$-ésimo ramo, ou serão processadas por todas as próximas camadas do ramo principal. Então, para cada imagem do conjunto de validação, coleta-se a confiança de classificação $\hat{p}_i$ e a função indicadora $\mathbb{1}[\hat{y}_i = y_i]$. Essa função assume o valor igual a 1 quando a classe inferida $\hat{y}_i$ é igual ao rótulo verdadeiro $y_i$ e 0 caso contrário. Então, utiliza-se $\hat{p}_i$ e $\mathbb{1}[\hat{y}_i = y_i]$ coletados no $i$-ésimo ramo e no ramo principal para calcular o valor de ECE, como descrito no Apêndice A.

Em geral, a Figura 6.1 mostra que o valor do ECE apresenta um comportamento monótono em função dos limiares de entropia $\boldsymbol{\tau}$. A partir da comparação do ECE dos ramos laterais com o do principal, a figura mostra que inserir um ramo lateral aumenta o valor do ECE. Assim, esse resultado indica que a DNN com saídas antecipadas pode aumentar ainda mais o erro de calibração do que uma DNN tradicional com somente o ramo principal. Por exemplo, comparando os resultados do ramo principal com resultados do primeiro quando $\tau_1 = 1$, há uma diferença de quatro vezes no valor do ECE. Na Figura 6.1, é possível notar que o ECE aproxima-se da saturação em valores limiares elevados. Portanto, quando o limiar é fixado em valores altos, é mais provável que a classificação seja realizada nos ramos laterais, ao invés do principal, o que aumenta o ECE.

Finalmente, ao comparar o valor do ECE entre os quatro ramos laterais, considerando o limiar em cada ramo como 1, ou seja, com classificação sempre no ramo lateral, é possível notar que o primeiro ramo está pior calibrado que o segundo. O segundo, por sua vez, está pior calibrado que o terceiro. Por outro lado, o terceiro ramo lateral apresenta o mesmo

comportamento que o quarto, devido à arquitetura da AlexNet, na qual a terceira e quarta camada convolucional são iguais [62] [1]. Assim, isso mostra uma certa hierarquia, na qual os ramos mais profundos tendem a ser mais calibrados do que os ramos mais próximos da camada de entrada. Por exemplo, para $t_i = 1$, o valor do ECE do primeiro ramo lateral é 1,03 vezes maior do que o segundo ramo, mas é 2,5 vezes superior ao do terceiro ramo. Assim, a Figura 6.1 mostra que não só a inserção de um ramo lateral descalibra a confiança na classificação fornecida pela DNN, mas também a sua posição determina o nível do erro de calibração. As próximas seções concentram-se na análise no primeiro e segundo ramos, uma vez que são aqueles com pior calibração.

### 6.1.2 ECE após empregar um Método de Calibração

Esta seção avalia como um método de calibração pode reduzir o ECE em cada um dos dois ramos laterais. Para esse fim, aplica-se o método de calibração *Temperature Scaling* [63] nesses ramos laterais. Cada ramo é calibrado utilizando apenas a amostra que é classificada nele antes da calibração. Em seguida, analisa-se o comportamento do ECE em função de diferentes limiares de entropia. Utiliza-se a mesma B-AlexNet da análise anterior, mas apenas com o primeiro e segundo ramo lateral. O ECE é medido considerando todas as amostras, ou seja, aquelas classificadas nos ramos laterais e no ramo principal.

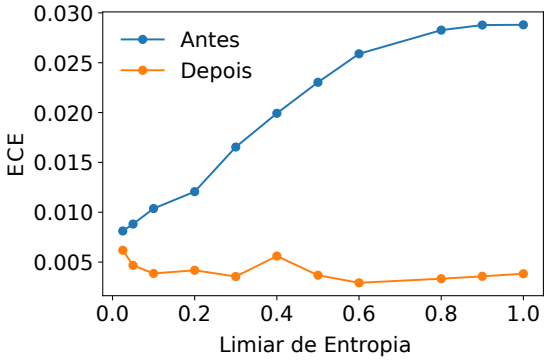


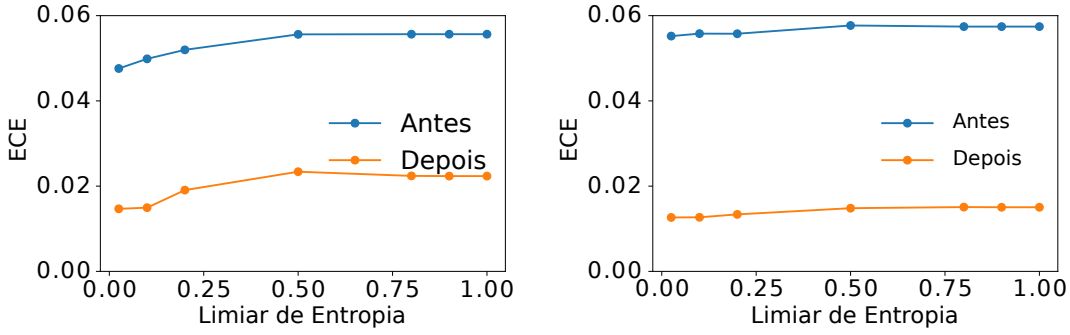

Figura 6.2: O efeito da calibração no ECE para o primeiro ramo lateral.

A Figura 6.2 mostra o ECE do primeiro ramo, antes e depois da calibração. Essa figura evidencia que o método *Temperature Scaling* pode reduzir o erro de calibração para qualquer configuração do limiar. Além disso, mostra que a redução do ECE é maior para valores elevados de limiares. A redução do ECE é maior, pois os ramos laterais pioram a calibração, conforme a Figura 6.1, e quanto maior o limiar de entropia, maior é a atuação desses ramos. Por exemplo, quando o limiar é $\tau_1 = 1$, a calibração pode reduzir o ECE em aproximadamente 6 vezes quando comparado a um ramo mal calibrado. Por outro lado,

[1] `https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py`

quando $\tau_1 = 0,025$, o ECE do ramo antes da calibração é 1,312 vez maior do que o ramo calibrado.

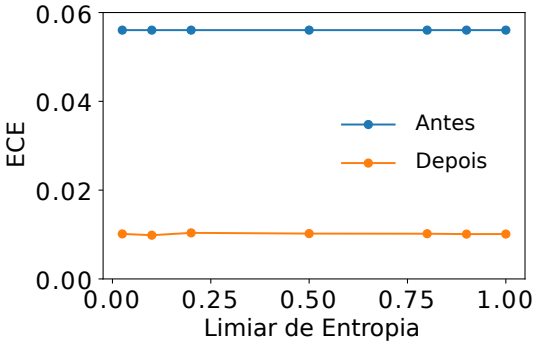
Em relação ao segundo ramo, a Figura 6.3 mostra o valor da métrica ECE de acordo com diferentes configurações do limiar $\tau_2$ no segundo ramo lateral. Cada gráfico dessa figura apresenta o resultado com uma configuração diferente do primeiro ramo, variando $\tau_1$ entre 0, 0,5, 0,7. A Figura 6.3 mostra que o efeito da calibração diminui conforme o valor do limiar de entropia do primeiro ramo. Nessas figuras, a medida que o valor $\tau_1$ aumenta, nota-se que o valor do ECE tende a uma constante.




(a) $\tau_1 = 0,0$.




(b) $\tau_1 = 0,5$.




(c) $\tau_1 = 0,7$.

Figura 6.3: O efeito do método de calibração no ECE para o segundo ramo lateral com diferentes configurações de $\tau_1$ no primeiro ramo.

### 6.1.3 Número de amostras classificadas nos ramos laterais

Esta seção avalia como o método de calibração empregado altera o número de amostras classificadas em cada ramo lateral. Usando o mesmo experimento descrito na Seção 6.1.1, obtém-se o número de amostras do conjunto de validação que são classificadas em cada ramo. A Figura 6.4 mostra a porcentagem das amostras classificadas no primeiro ramo lateral de acordo com a configuração do limiar, antes e depois da aplicação do método *Temperature Scaling*. De acordo com a figura, o método de calibração reduz a porcentagem de amostras classificadas no primeiro ramo, quando comparado com o caso em que

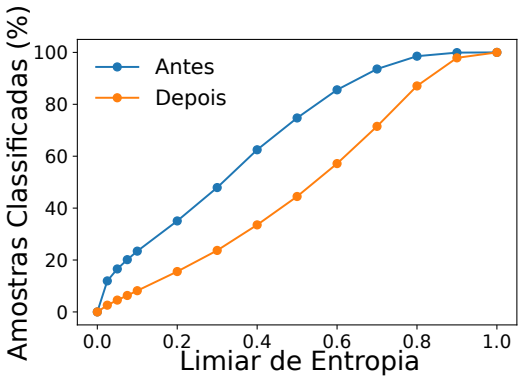


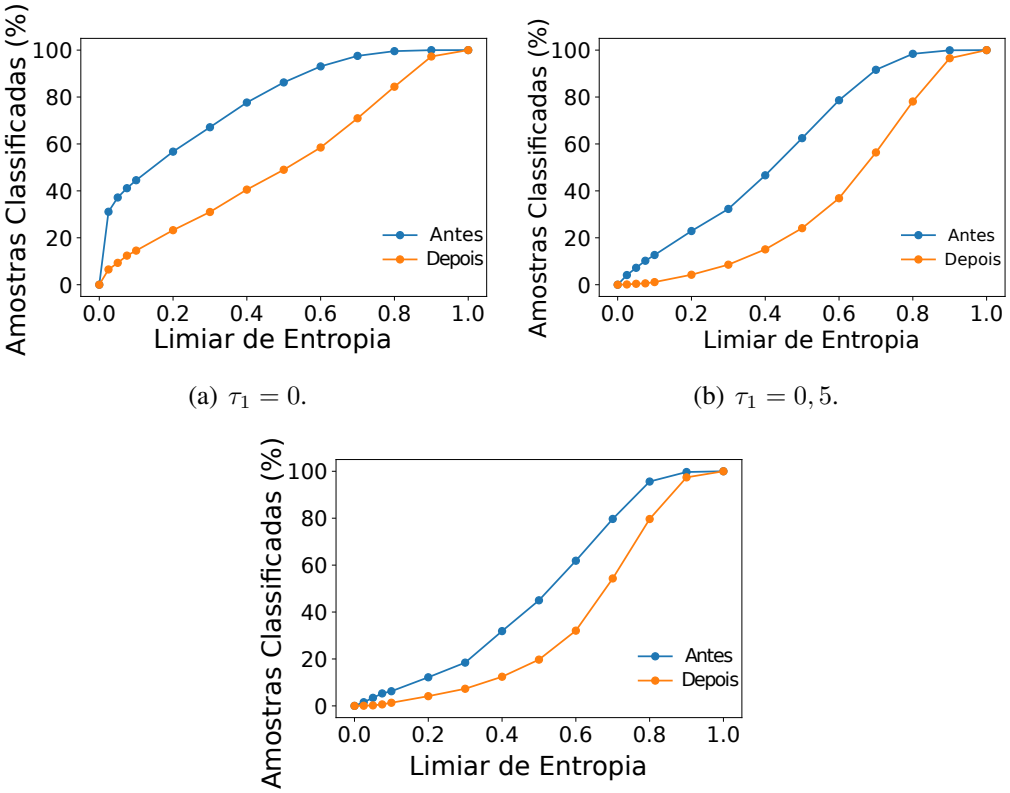

Figura 6.4: Porcentagem de amostras classificadas no primeiro ramo lateral.

não é aplicado nenhum método de calibração. Esse resultado é esperado, uma vez que o ramo lateral antes da calibração possui alta confiança quanto às suas previsões. Em outras palavras, o ramo superestima a sua capacidade de previsão, classificando mais amostras do que deveria. Para confirmar essa conclusão, mostra-se mais adiante, na Seção 6.1.4, que um ramo bem calibrado atinge uma acurácia substancialmente maior do que um ramo mal calibrado.

A Figura 6.5 mostra a porcentagem de amostras classificadas no segundo ramo lateral de acordo com a configuração do limiar. Nesta análise, considera-se apenas as amostras que não são classificadas no primeiro ramo lateral (ou seja, 100% significa que o segundo ramo classifica todas as amostras que não são classificadas pelo primeiro). Cada gráfico nessa figura corresponde a um valor de limiar diferente do primeiro ramo, variando entre 0, 0,5, e 0,7 que representam um valor baixo, intermediário e alto, respectivamente. Essa figura também mostra que um ramo bem calibrado classifica uma porcentagem menor de amostras do que um ramo mal calibrado, o que complementa os resultados apresentados na Figura 6.4. Além disso, é possível notar que o efeito da calibração depende do limiar do primeiro ramo. Isso ocorre pois, conforme mostrado na Figura 6.4, quanto maior o limiar $\tau_1$ do primeiro ramo lateral, menos amostras são disponíveis para serem classificadas no segundo ramo. Por exemplo, de acordo com a Figura 6.4, apenas 30% das amostras do conjunto de validação são disponíveis para serem classificadas no segundo ramo. Além disso, as amostras que não são classificadas no primeiro mesmo com $\tau_1 = 0,7$ tendem a ser amostras que também necessitam de elevado $\tau_2$ para serem classificadas antecipadamente.

### 6.1.4 Acurácia Antes e Depois da Calibração

Finalmente, esta seção avalia o impacto do método de calibração na acurácia. Para isso, a Figura 6.6 mostra a acurácia em função da configuração dos limiares usando uma B-AlexNet com somente o primeiro ramo lateral. Em geral, a Figura 6.6 mostra que uma DNN com um ramo bem calibrado atinge uma acurácia maior, quando comparado com o ramo lateral descalibrado. Portanto, no primeiro ramo lateral, uma etapa de calibração




(a) $\tau_1 = 0$.

(b) $\tau_1 = 0,5$.

(c) $\tau_1 = 0,7$.

Figura 6.5: Porcentagem de amostras classificadas no segundo ramo, para diferentes configurações $\tau_1$ do primeiro ramo lateral.

pode aumentar a acurácia das DNNs com saídas antecipadas. Esse comportamento é esperado, já que um ramo lateral descalibrado superestima sua confiança em relação à sua capacidade de inferir nos ramos laterais. Consequentemente, esses ramos classificam mais amostras, como já mostrado na Figura 6.4. Essa super confiança faz com que as DNNs classifiquem amostras pouco confiáveis no primeiro ramo lateral, reduzindo a acurácia da BranchyNet. Esse é um grave problema, especialmente em aplicações como carros inteligentes e diagnóstico médico, nas quais a perda de acurácia é inaceitável [64].

A Figura 6.7 mostra a acurácia em função dos limiares usando uma B-AlexNet com o primeiro e o segundo ramo, quando ajusta-se o limiar do primeiro ramo para 0, 0,5, e 0,7. Essa figura também mostra que uma etapa de calibração pode aumentar a acurácia em DNNs com saídas antecipadas, como já mostrado na Figura 6.6.

Utilizando a Figura 6.7, é possível também analisar o impacto do segundo ramo na acurácia de acordo com a configuração do primeiro ramo $\tau_1$. Nesse contexto, em geral, nota-se que a acurácia diminui conforme o valor do limiar $\tau_1$ do primeiro ramo aumenta. Isso é esperado, pois mais amostras são classificadas no primeiro ramo, que tende a ser menos acurado do que os outros ramos mais profundos. Em seguida, nota-se que, na Figura 6.7(c) ($\tau_1 = 0,7$), a acurácia apresenta um comportamento constante para um dado intervalo de limiares de 0 a 0,5. Isso ocorre porque quando $\tau_1 = 0,7$, a maioria das amos-

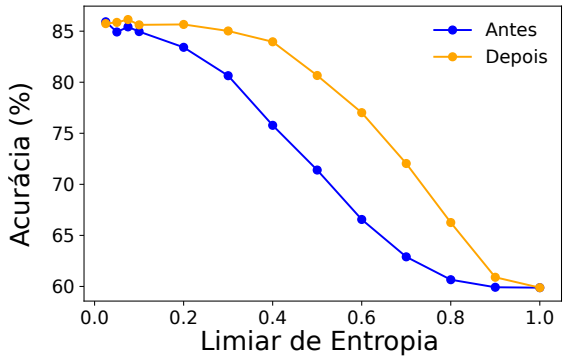
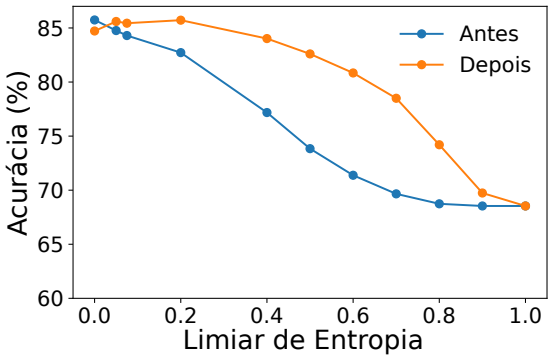
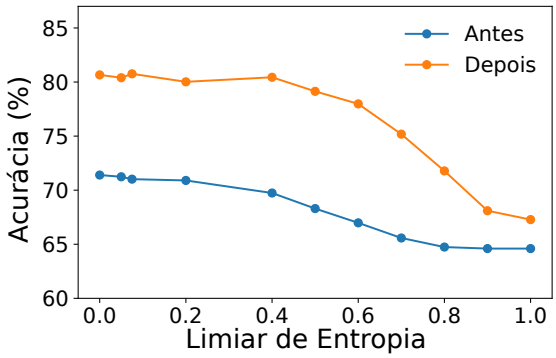



Figura 6.6: Acurácia conforme a configuração do limiar do primeiro ramo lateral.

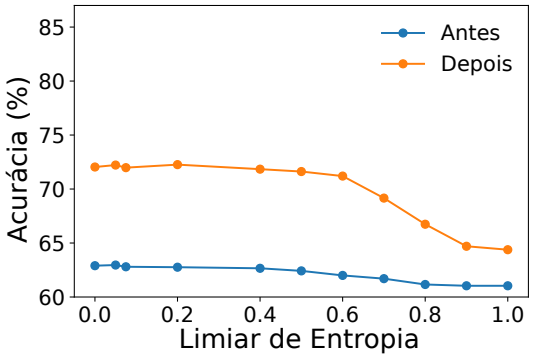
tras do conjunto de validação são classificadas no primeiro ramo (isto é, aproximadamente 80%), como visto na Figura 6.4. Consequentemente, a acurácia desse ramo contribui mais para a acurácia final da BranchyNet. Por fim, a figura mostra que a acurácia do ramo bem calibrado começa a decair quando $\tau_2 = 0,6$, que classifica 40% das amostras disponíveis, como visto na Figura 6.5(c).




(a) Segundo ramo com $\tau_1 = 0$.




(b) Segundo ramo com $\tau_1 = 0,5$.




(c) Segundo ramo com $\tau_1 = 0,7$.

Figura 6.7: Acurácia de classificação do segundo ramo para diferentes configurações de limiar $\tau_1$ do primeiro ramo.

## 6.2 Acurácia versus Tempo de Inferência

As DNNs com saídas antecipadas possuem um compromisso entre acurácia e tempo de inferência, conforme detalhado na Seção 5.3. Esta seção objetiva mostrar que o método de calibração influencia esse compromisso, impactando a decisão de particionamento. Para isso, esta seção utiliza a mesma B-AlexNet da seção anterior. Esse posicionamento do ramo lateral é escolhido para análise de um caso extremo, já que os ramos mais próximos da camada de entrada são os que possuem pior calibração, como mostra a Figura 6.1.

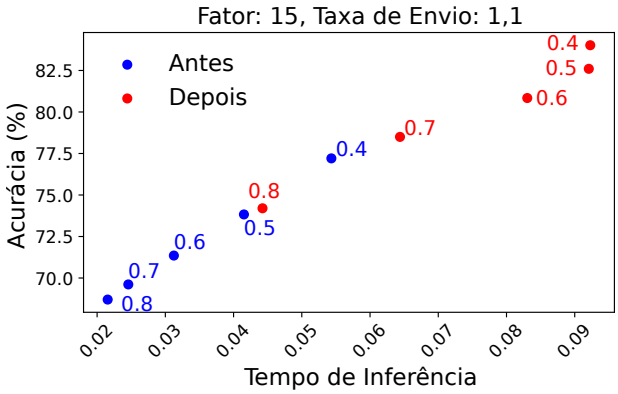
A análise consiste em executar o método de otimização descrito na Seção 4.7 para escolher a camada de particionamento dada uma taxa de envio entre dispositivo em borda e nuvem. Conforme mostrado na Figura 5.4, cada estratégia de particionamento escolhida está associada a uma configuração de limiar de entropia, a um tempo de inferência e acurácia.

Os valores dos limiares utilizados nessa análise variam de 0,4 a 0,8 com passo de 0,1. Esses valores são escolhidos para facilitar a visualização dos resultados. Para cada tecnologia sem-fio, os resultados das Figuras 6.8 e 6.9 apresentam o tempo de inferência e a acurácia associada com a estratégia de particionamento escolhida pela tarefa Otimização do Particionamento de BranchyNet, executada pelo componente *Decision Maker* do sistema POPEX e descrita na Seção 4.7.

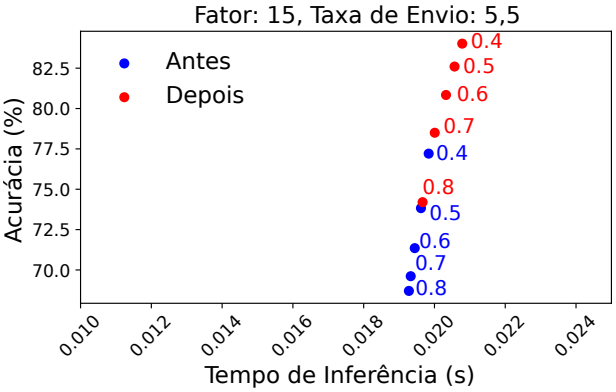
Esses resultados mostram o compromisso entre acurácia e tempo de inferência para o particionamento da B-AlexNet antes e depois da implementação do método de calibração *Temperature Scaling*. A Figura 6.8 apresenta os resultados considerando o fator de processamento $\gamma = 15$, enquanto a Figura 6.9 considera o fator de processamento $\gamma = 50$. Nessas figuras, cada ponto é associado a um valor de limiar de entropia, indicado próximo ao seu respectivo ponto.

As Figuras 6.8 e 6.9 mostram, como já observado na Seção 5.3, que a redução do tempo de inferência gera uma penalidade na acurácia. Em todos os resultados é possível notar que a BranchyNet calibrada possui, de fato, um tempo de inferência mais elevado, devido à menor quantidade de amostras classificadas no ramo lateral, mas isso é necessário para manter altos níveis de acurácia. Por exemplo, na Figura 6.9(a), para um limiar de 0,8, o ramo bem calibrado aumenta o tempo de inferência em 0,023 s, mas a acurácia aumenta em 8%. Comparando os resultados de 1,1 Mbps e 5,5 Mbps nas Figuras 6.8 e 6.9, é possível notar que a menor taxa de envio de 1,1 Mbps faz o impacto da descalibração ser maior.

A partir dos resultados obtidos neste capítulo, conclui-se que se deve calibrar a DNN com saídas antecipadas antes de aplicar um método de particionamento. Caso contrário, os métodos de particionamento podem artificialmente reduzir o tempo de inferência, desconsiderando os requisitos de acurácia da BranchyNet. Assim, mostra-se como o modelo da DNN pode impactar no particionamento. O próximo capítulo apresenta uma análise de como a distorção da imagem pode impactar o particionamento.

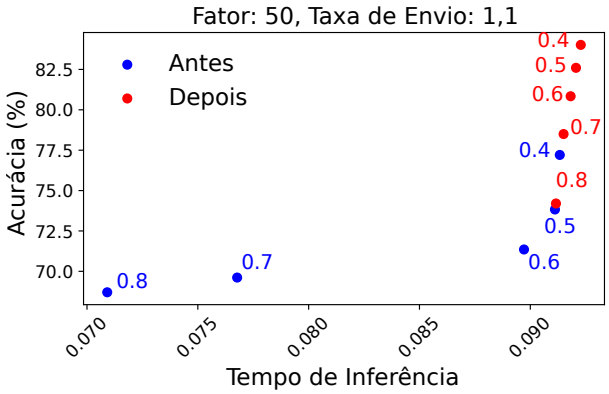



(a) Compromisso entre acurácia e tempo de inferência usando taxa de envio de 1,1 Mbps.

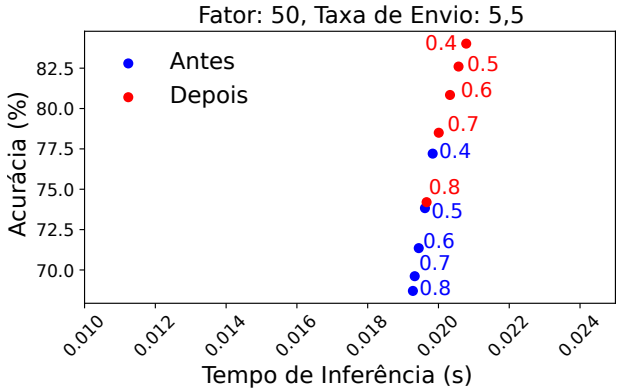



(b) Compromisso entre acurácia e tempo de inferência usando taxa de envio de 5,5 Mbps.

Figura 6.8: Compromisso entre acurácia e tempo de inferência $\gamma = 15$ usando taxas de envio de 1,1 Mbps e 5,5 Mbps.




(a) Compromisso entre acurácia e tempo de inferência usando taxa de envio de 1,1 Mbps.




(b) Compromisso entre acurácia e tempo de inferência usando taxa de envio de 5,5 Mbps.

Figura 6.9: Compromisso entre acurácia e tempo de inferência $\gamma = 50$ usando taxas de envio de 1,1 Mbps e 5,5 Mbps.

# Capítulo 7

# Impacto da Distorção da Imagem no Particionamento de DNNs

Esta dissertação mostra, no Capítulo 5, que a probabilidade de classificação nos ramos laterais altera a decisão de particionamento em DNNs com saídas antecipadas. Este capítulo demonstra que fatores intrínsecos às imagens de entrada impactam a probabilidade e, consequentemente, a decisão de particionamento. Por exemplo, quando DNNs são utilizadas para identificar objetos em imagens, o nível de distorção nessas imagens pode influenciar a quantidade mínima de camadas necessárias para a inferência.

Uma imagem com baixo nível de distorção pode atingir o nível desejado de incerteza nas primeiras camadas da BranchyNet, enquanto imagens com alto nível de distorção podem exigir o processamento de todas as camadas [65]. Em uma infraestrutura de computação na borda, isso pode impactar a decisão de enviar para a nuvem ou não. Assim, este capítulo tem o objetivo de mostrar o impacto da distorção da imagem no tempo de inferência de DNNs na borda.

## 7.1 Análise do Tempo de Inferência de Imagens sem Distorção

Os experimentos deste capítulo são baseados em DNNs treinadas para inferir se um animal em uma imagem é um cão ou um gato. Para o treinamento, utiliza-se um *dataset* composto de imagens de cães e gatos, em diferentes ambientes, sem nenhum tipo de distorção das imagens [66]. Para treinar as redes neurais, divide-se o *dataset* em um conjunto de treinamento com 20.000 amostras, além de um conjunto de validação e um outro de testes com 2.500 amostras cada. É importante ressaltar que o *dataset* é balanceado, ou seja, o número de imagens rotuladas como gatos é aproximadamente igual ao de cães. Os experimentos são avaliados nos três diferentes cenários da Figura 4.7, isto é, processamento exclusivamente na nuvem, exclusivamente na borda e baseado em particio-

namento. Nos cenários de processamento apenas na borda e baseado em particionamento, implementam-se redes neurais B-SqueezeNet. Quando o processamento ocorre somente na nuvem, utiliza-se a DNN AlexNet [19], que consiste em uma DNN sem ramos laterais que atinge uma acurácia maior do que a B-SqueezeNet, mas também com maior demanda computacional. A acurácia da DNN consiste na porcentagem de inferências realizadas corretamente dado um conjunto de amostras. Por exemplo, dadas diversas imagens de cães e gatos, a acurácia da DNN corresponde à porcentagem de imagens nas quais a inferência indicou corretamente qual é o animal da foto.

Os experimentos utilizam como dispositivo de borda o Raspberry Pi 3 Model B+, pois se trata de uma plataforma comumente utilizada para esse fim na literatura [16, 67]. O Raspberry Pi 3 Model B+ é equipado com um processador quad-core ARMv8 1.2 GHz com 1 GB de RAM. Em relação à nuvem, utiliza-se o Google Colaboratory. Portanto, este capítulo avalia o impacto da distorção da imagem no particionamento considerando um cenário real e não um fator de processamento $\gamma$, como nos experimentos anteriores. Assim, os experimentos dessa seção analisam o comportamento em um cenário real e não utilizam o problema de otimização das seções anteriores, justamente para um maior controle dos experimentos. O dispositivo de borda e a nuvem são utilizados para medir o tempo de processamento da DNN e realizar a inferência. O tempo de comunicação entre a borda e a nuvem é emulado a partir de experimentos com o iPerf3[1] e informações da literatura, conforme apresentado mais adiante. É válido ressaltar que todas as redes neurais utilizadas nos experimentos são previamente treinadas na nuvem. Em relação às ferramentas de *software*, os experimentos deste capítulo utilizam as mesmas ferramentas descritas na Seção 5.1, ou seja, são desenvolvidos em Python 3 com o *framework* Pytorch. Os diferentes cenários utilizados são detalhados a seguir, juntamente com análises preliminares. Em todo o trabalho, os resultados apresentados correspondem a médias de diversas amostras com o intervalo de confiança de 95%.

### 7.1.1 Processamento exclusivamente na nuvem ou na borda

No cenário de processamento exclusivamente na nuvem, todas as camadas da DNN são executadas em um servidor em nuvem, como ilustrado na Figura 4.7(b). Nesse cenário, o tempo de inferência é dado pela Equação 4.3. Para medir o tempo de comunicação entre a borda e a nuvem, seleciona-se uma imagem pertencente ao *dataset*, cujo tamanho é de 748,5 kB sem qualquer compressão. Em seguida, envia-se essa quantidade de dados de um PC no Rio de Janeiro a diversas localidades ao redor do mundo para simular o cenário de nuvens distribuídas geograficamente. Para tal, utiliza-se a ferramenta iPerf3, que oferece servidores iPerf gratuitos e geograficamente distribuídos. Além disso, utilizam-se as taxas de envio de 1,1 Mbps e 18,8 Mbps apresentadas na Tabela 5.1. Os valores dessas

[1] `https://iperf.cc/`

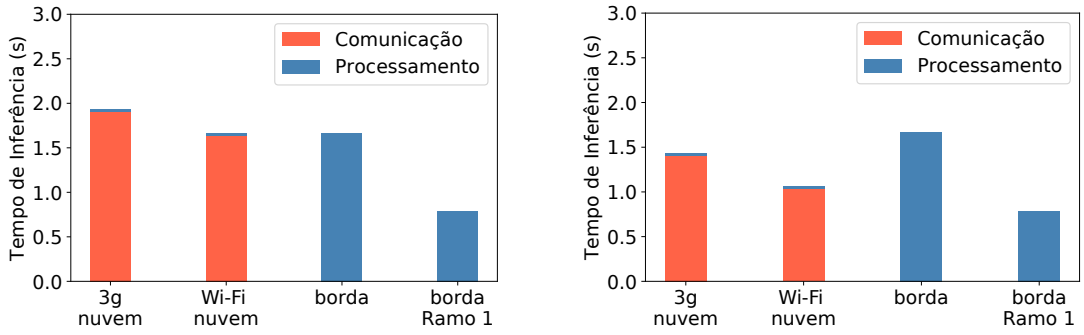
taxas são utilizados para limitar a taxa enviada pelo cliente iPerf (isto é, parâmetro "-b" da ferramenta). O tamanho da imagem é utilizado para limitar o tamanho do arquivo enviado (isto é, parâmetro "-n" da ferramenta). A Tabela 7.1 mostra os resultados de tempo de comunicação para cinco servidores iPerf espalhados pelo mundo. A partir desses resultados, os demais experimentos consideram os casos extremos: EUA e Rússia. É importante frisar que, neste trabalho, a taxa escolhida para 1,1 Mbps e 1,1 Mbps é inferior à banda disponível entre o PC do Rio de Janeiro e os servidores iPerf. Ou seja, o gargalo é a rede de acesso da borda.

| Servidor | País | Tempo de Envio (s) |
|---|---|---|
| iperf.he.net | EUA | 1,029 |
| iperf.worldstream.nl | Nova Zelândia | 1,024 |
| speedtest.wtnet.de | Alemanha | 1,259 |
| bouygues.iperf.fr | França | 1,263 |
| speedtest.hostkey.ru | Rússia | 1,435 |

Tabela 7.1: Tempo de envio de 748,5 kB para os servidores iPerf.

Em relação ao cenário de processamento exclusivamente na borda da Figura 4.7(a), o tempo de inferência corresponde apenas ao tempo de processamento das camadas na borda, sem necessidade de envio para a nuvem. Esse tempo é, então, influenciado apenas pela capacidade computacional do Raspberry Pi 3. Nesse cenário, implementa-se uma rede neural com arquitetura B-SqueezeNet, a qual é descrita na Seção 2.3 e ilustrada na Figura 2.9(b). Nesse caso, optou-se pela arquitetura SqueezeNet como ramo principal, devido a restrições de memória do dispositivo em borda. Por outro lado, essa arquitetura atinge um nível de acurácia próximo à rede neural AlexNet implementada na nuvem. Para comparar o tempo total de inferência nos cenários de processamento somente na nuvem ou na borda, as Figuras 7.1(a) e 7.1(b) apresentam os tempos de comunicação e processamento e, consequentemente, o tempo de inferência com diferentes taxas de envio, considerando nuvens localizadas na Rússia e EUA, respectivamente. No caso da borda, apresentam-se resultados nos quais todas as camadas são percorridas e o caso no qual a inferência termina no primeiro ramo.

De forma geral, a Figura 7.1 mostra que o tipo de rede e sua respectiva taxa de envio representam um fator decisivo para determinar a estratégia de processamento adotada [16]. Em relação ao tempo total de inferência, a Figura 7.1(a) mostra que utilizar apenas o dispositivo em borda não soluciona o problema de reduzir o tempo de inferência, pois o tempo de processamento na borda praticamente equivale ao tempo de envio de uma imagem bruta à nuvem, além de utilizar uma DNN com menor acurácia. A solução de inserir um ramo lateral consegue reduzir significativamente o tempo de inferência, contudo, reduz também a acurácia do modelo, como apresentado mais adiante na Figura 7.4. Assim, é necessário adotar o particionamento de DNNs para lidar com esse compromisso.




(a) Nuvem na Rússia.

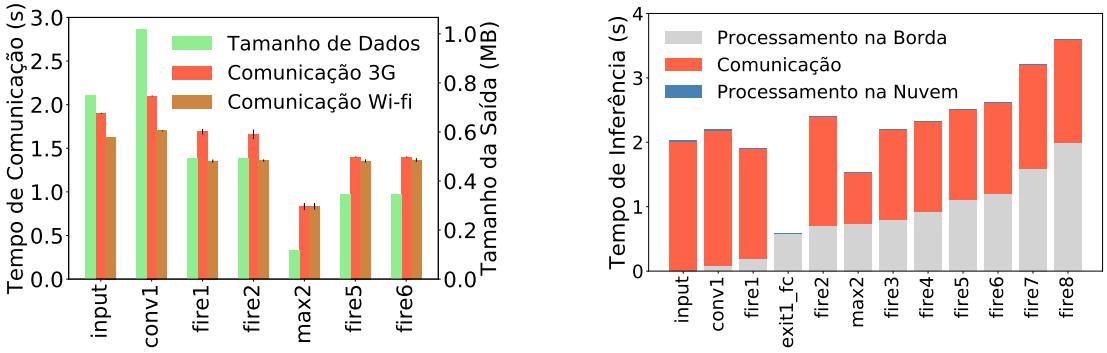
(b) Nuvem nos EUA.

Figura 7.1: Tempos de inferência na nuvem e na borda.

### 7.1.2 Processamento Baseado em Particionamento de DNN

No cenário de particionamento, a quantidade de dados enviada à nuvem corresponde aos dados de saída da camada de particionamento, a última camada processada na borda. A Figura 7.2(a) mostra o tempo de comunicação e tamanho dos dados enviados para alguns exemplos de particionamento. Na Figura 7.2(a), cada barra verde (isto é, a mais à esquerda) representa o tamanho dos dados enviados à nuvem caso uma dada camada seja escolhida para particionamento. Além das barras verdes, a figura também mostra barras relacionadas ao tempo de comunicação no envio de dados em cada camada.

Os experimentos consideram a nuvem na Rússia para enfatizar o pior caso de tempo de comunicação. O primeiro conjunto de barras, denominado *input*, corresponde a enviar a imagem bruta de 748,5 kB. Nota-se que a primeira camada convolucional (*conv1*) aumenta consideravelmente o tamanho de dados a serem enviados à nuvem [11]. Portanto, não é desejável particionar a B-SqueezeNet logo na primeira camada convolucional. Contudo, supondo um particionamento logo após a camada *max2*, os resultados mostram uma redução significativa na quantidade de dados enviados.




(a) Tempo de comunicação e tamanho da saída.

(b) Tempo de inferência para rede com taxa de envio de 1,1 Mbps.

Figura 7.2: Análise da nuvem da Rússia para diferentes camadas de particionamento.

No cenário de particionamento, o tempo de inferência corresponde à soma dos seguin-

tes fatores: tempo de processamento na borda das camadas da DNN até a de particionamento, tempo de comunicação e tempo de processamento das camadas restantes na nuvem. Neste experimento, utiliza-se o mesmo cenário descrito na Seção 7.1.1. Considera-se o uso de rede com taxa de 1,1 Mbps e nuvem na Rússia para uma análise de pior caso. Os resultados são mostrados na Figura 7.2(b), na qual cada barra corresponde a uma decisão de particionamento diferente. Por exemplo, a barra correspondente à camada *fire1* refere-se ao processamento dessa camada e das anteriores no dispositivo em borda e as demais na nuvem. Consequentemente, o tempo de comunicação nesse caso corresponde ao tempo de comunicação da rede com taxa de envio de 1,1 Mbps da camada *fire1* na Figura 7.2(a). Já a barra da camada *input* corresponde à decisão de particionamento em enviar a imagem bruta à nuvem, portanto, trata-se do cenário de processamento exclusivamente na nuvem. Ressalta-se que a Figura 7.2(b) mostra todas as possíveis decisões de particionamento sem ramos laterais, que apresentam uma acurácia de 92,2%. A camada *saída1*, por sua vez, corresponde ao cenário com um ramo lateral, que resulta em acurácia de 83,33%. Nesse caso, configura-se o limiar de entropia como 1, ou seja, as amostras sempre são classificadas pelo ramo lateral. Os resultados da Figura 7.2(b) mostram que a escolha do particionamento pode afetar significativamente o tempo de inferência. Nota-se que o tempo de processamento na nuvem é desprezível em todos os casos. De todas as camadas, é possível observar que o menor tempo de inferência ocorre na camada *saída1*. Entretanto, esse caso corresponde a uma acurácia inferior às demais, dada a saída obrigatória no ramo lateral. Caso fosse necessária uma maior acurácia, deveriam ser adotadas estratégias de particionamento. Para o cenário estudado, a melhor estratégia é particionar em *max2*. Ou seja, torna-se mais interessante aguardar o processamento na borda nas camadas iniciais até alcançar a *max2* para, então, enviar à nuvem.

## 7.2 Análise do Impacto da Distorção da Imagem

Esta seção avalia, primeiramente, o efeito da distorção da imagem na acurácia em uma rede neural B-SqueezeNet, implementada na borda, e em uma AlexNet, implementada na nuvem. Em seguida, os experimentos mostram o impacto da distorção da imagem no tempo de inferência. Para tanto, esses experimentos são implementados nos três cenários apresentados na Seção 7.1, utilizando o mesmo conjunto de imagens.

### 7.2.1 Impacto da Distorção na Acurácia

Em aplicações IoT com câmeras, os dispositivos, por serem normalmente de baixo custo, eventualmente capturam imagens com diversos tipos de distorção, como contraste, ruído, e *blur*. Essas distorções impactam a qualidade da imagem e, consequentemente, a acurácia das DNNs [57]. Esta dissertação avalia os efeitos do *blur* em DNNs, o qual

atua como um filtro passa-baixa na imagem, atenuando detalhes em alta frequência. O *blur* emula distorções ocorridas devido à distância focal do objeto de interesse, ou ao movimento desse objeto em relação à câmera ou vice-versa. Esse tipo de distorção pode ocorrer mesmo em câmeras de alta resolução. Por exemplo, em aplicações de identificação de placas de trânsito em veículos inteligentes, o movimento da câmera veicular pode causar distorção similar ao *blur*. A inserção de *blur* em uma imagem é implementada pela aplicação da operação de convolução entre a imagem original e um filtro. Este trabalho utiliza um filtro Gaussiano, variando as suas dimensões, enquanto se mantém a variância da imagem original. Em imagens digitais, o valor de um *pixel* tende a estar correlacionado a seus vizinhos. Assim, aplicando filtros com dimensões maiores, altera-se valores de *pixels* cada vez mais distantes e, por sua vez, menos correlacionados. Dessa forma, filtros com dimensões maiores resultam em imagens com maior nível de *blur*. Utilizando o conjunto de imagens da Seção 7.1, aplicam-se diferentes níveis de *blur* (isto é, dimensões do filtro) nas imagens do conjunto testes, resultando em diversos *datasets* com imagens distorcidas. Um exemplo de diferentes níveis aplicados é apresentado na Figura 7.3.

(a) Nível = 0. (b) Nível = 5. (c) Nível = 15. (d) Nível = 65.

Figura 7.3: Exemplos de imagem com diferentes níveis de *blur*.

Para analisar o impacto do *blur* na acurácia, implementa-se uma B-SqueezeNet no Raspberry Pi da borda e uma AlexNet na nuvem. Como na Seção 7.1, ambas as redes neurais são treinadas com um conjunto de imagens sem distorção. Na inferência, aplica-se o conjunto de imagens distorcidas para avaliar a acurácia nas duas redes neurais. Em relação à DNN implementada na borda, avalia-se a acurácia dos ramos laterais e do ramo principal. Para avaliar o primeiro ramo lateral, define-se o valor do limiar de entropia como 1, para que todas as amostras sejam classificadas nesse ramo. Já para o segundo ramo, o limiar de entropia do primeiro ramo é configurado como 0 e o segundo limiar como 1, assim todas as amostras do conjunto de dados são classificadas pelo segundo ramo lateral. Por fim, para o ramo principal, os limiares de entropia dos dois ramos laterais presentes são definidos como 0 para que todas as amostras sejam processadas por todas as camadas da rede neural. É válido frisar que a acurácia independe de condições da rede, então essa análise não considera diferentes tecnologias de acesso ou localidade da nuvem. A Figura 7.4 apresenta a acurácia para diferentes níveis de *blur* para as B-SqueezeNet da borda e a AlexNet da nuvem. A Figura 7.4 mostra que a acurácia das redes neurais é sensível ao aumento do nível de *blur* [57]. Além disso, mostra que o primeiro ramo é ainda mais sensível do que o segundo, que, por sua vez, é mais sensível ao *blur* que o ramo

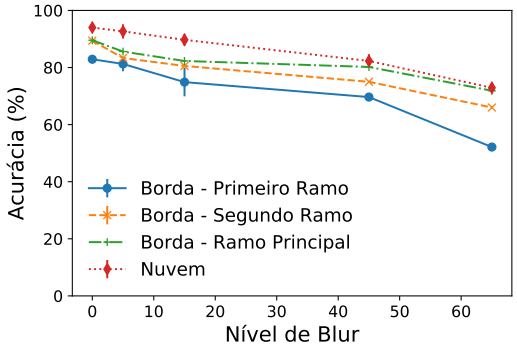



Figura 7.4: Acurácia na classificação das redes neurais B-SqueezeNet na borda e AlexNet na nuvem, utilizando diferentes níveis de *blur*.

principal. Esse comportamento pode ser explicado por duas razões. A primeira deve-se ao *blur* remover texturas da imagem, as quais são as características utilizadas para classificar a imagem nas primeiras camadas [57]. A segunda razão consiste no fato de que quanto menos camadas são percorridas em uma DNN, maior tende a ser o nível de incerteza da classificação e, consequentemente, há uma maior sensibilidade ao *blur*. Dessa forma, é possível estabelecer uma hierarquia de sensibilidade, na qual a acurácia fornecida pelos ramos posicionados mais próximos da camada de entrada é mais sensível do que a de ramos mais próximos da camada de saída do ramo principal. Os resultados apresentados a seguir mostram como é possível reduzir o tempo de inferência se a aplicação aceita uma redução de acurácia.

### 7.2.2 Avaliação do Impacto da Distorção no Tempo de Inferência

O objetivo deste experimento é mostrar que considerar o nível de *blur* em uma imagem permite reduzir o tempo de inferência para sua classificação. Para esse experimento, implementa-se uma rede neural B-SqueezeNet com somente um ramo lateral na borda e seu respectivo ramo principal na nuvem. A escolha de apenas um ramo lateral tem como objetivo facilitar a análise. Já em relação à posição do ramo lateral, o experimento utilizou o primeiro ramo posicionado após a camada *fire1*, como apresentado na Figura 2.9(b), para evitar processamento desnecessário na borda. Diferentemente da Seção 7.2.1, o limiar de entropia do ramo é ajustado em valores menores que 1 e, assim, a amostra pode não ser classificada no ramo lateral. Caso isso ocorra, as amostras são processadas por camadas posteriores e seguem o particionamento de DNNs descrito na Seção 7.1, no qual parte da rede neural é executada na borda e a outra na nuvem. Os resultados também apresentam o cenário de processamento somente na nuvem (isto é, com a AlexNet) para fins comparativos. Nesse caso, não há distorção nas imagens, mas os resultados aplicam-se a imagens com qualquer nível de *blur*, já que não há ramos laterais. A partir do cenário de particionamento, o experimento avalia o tempo de inferência em cada conjunto de

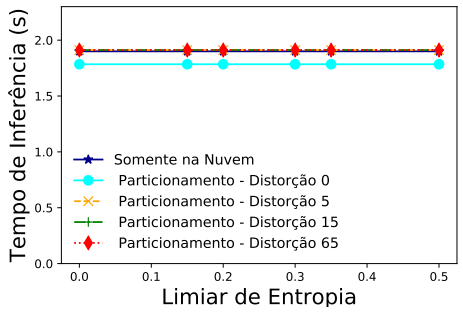
imagens com três níveis de *blur*: 5, 15 e 65. Esses níveis são escolhidos para verificar o comportamento para imagens com distorção baixa, intermediária e alta, respectivamente. Além disso, os resultados apresentados a seguir também consideram o particionamento de DNN em imagens sem nenhum *blur* (isto é, nível 0) para fins comparativos.

A análise realizada consiste em dividir cada conjunto de imagens distorcidas em lotes compostos por 48 imagens. Então, inserem-se tais lotes na rede neural e mede-se o tempo de inferência médio de cada lote. Como visto na Seção 2.3, para que uma dada imagem seja classificada no ramo lateral, essa deve atender a um critério de confiança. Caso atenda, o tempo de inferência corresponde ao tempo de processamento na borda das camadas anteriores ao ramo lateral e ao das camadas pertencentes ao ramo lateral. Caso não atenda, a imagem segue sendo processada pelas próximas camadas até a camada de particionamento, a partir da qual os dados de saída dessa camada são enviados à nuvem. Nesse caso, o tempo de inferência é a soma do tempo de processamento na borda até a camada de particionamento, do tempo de comunicação no envio dos dados de saída dessa camada para a nuvem e do tempo de processamento das camadas restantes na nuvem.

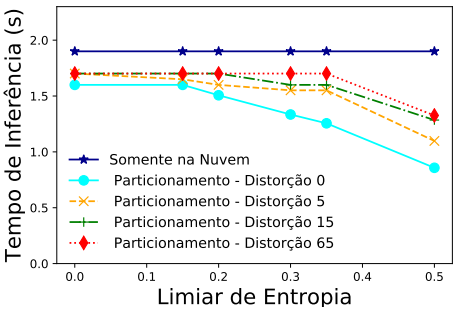
Dado o exposto, o experimento avalia o tempo de inferência para o particionamento após cada camada da rede neural, variando os valores de limiar de entropia entre 0 e 0,5. Este experimento avalia dois casos extremos de condições da nuvem, considerando as medidas de rede da Seção 7.1.1. No pior caso, apresentado na Figura 7.5, utiliza-se uma rede de acesso de 1,1 Mbps e uma nuvem na Rússia. No melhor caso, apresentado na Figura 7.6, utiliza-se uma rede de acesso com taxa de envio de 18,8 Mbps e uma nuvem nos EUA. Os gráficos da Figura 7.5 apresentam os resultados considerando o particionamento após as camadas *conv1*, *fire2*, *fire5* e *fire6*. Essas camadas foram escolhidas para analisar o comportamento do tempo de inferência em camadas anteriores aos ramos (*conv1*), camada posterior mais próxima (*fire2*) e camadas posteriores mais distantes do ramo lateral (*fire5* e *fire6*). Já a Figura 7.6, por questões de concisão, apresenta apenas os resultados de particionamento após as camadas *fire2* e *fire5*.

Considerando o cenário de rede com taxa de envio de 1,1 Mbps e o servidor localizado na Rússia, a Figura 7.5(a) mostra o tempo de inferência de imagens com diferentes níveis de *blur* para o caso de particionamento após a camada *conv1*. Nesse caso, os tempos de inferência de imagens com diferentes níveis de *blur* são iguais e não dependem do limiar de entropia do ramo lateral. Isso ocorre pois, como o particionamento é realizado antes do ramo lateral, então nenhuma amostra é processada e classificada nele e a inferência sempre termina na nuvem. Como a DNN da nuvem não possui ramos laterais, o tempo de inferência não é impactado. Por outro lado, considerando camadas mais profundas (mais próximas da camada de saída), como a camada *fire6*, a Figura 7.5(d) mostra que o cenário sem particionamento (isto é, o "Somente nuvem") possui um tempo de inferência menor em um dado intervalo de probabilidade, mesmo executando a AlexNet, que é uma rede mais complexa do que a B-SqueezeNet. Isso acontece pois, nesse intervalo de probabili-

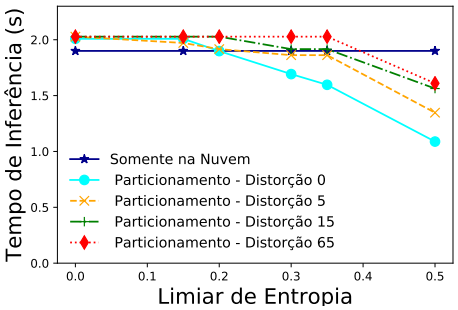
dade, como mostra a Figura 7.2(a), enviar os dados brutos é menos custoso do que enviar dados após a camada *conv1*. À medida que o limiar de entropia aumenta, mais amostras são classificadas no ramo, então se torna menos custoso processar na borda mesmo até camadas mais profundas, como a camada *fire6*.



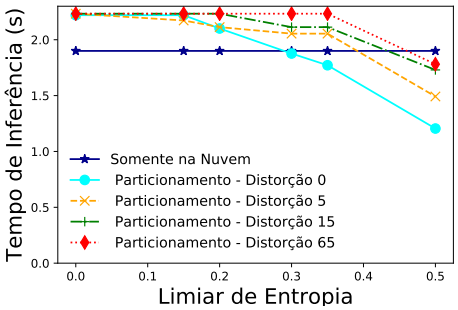

(a) Particionamento na camada *conv1*.




(b) Particionamento na camada *fire2*.




(c) Particionamento na camada *fire5*.




(d) Particionamento na camada *fire6*.

Figura 7.5: Tempo de inferência em função da distorção, para diferentes camadas de particionamento, utilizando uma nuvem na Rússia e taxa de envio de 1,1 Mbps.

De forma geral, a Figura 7.5 ilustra que limiares de entropia cada vez maiores reduzem o tempo de inferência para todos os níveis de distorção da imagem. Esse comportamento ocorre porque quanto maior o limiar de entropia, mais imagens atendem ao critério de confiança e assim são classificadas no primeiro ramo lateral, portanto, não são enviadas à nuvem. Isso está de acordo com os resultados da Figura 7.2, que mostra que o tempo de inferência para classificar uma imagem no primeiro ramo lateral (isto é, o *saída1*) é inferior em comparação ao tempo de inferência com particionamento após qualquer camada e nas duas redes de acesso avaliadas. Para um lote de imagens, considerado na Figura 7.5, a classificação antecipada no ramo lateral de mais amostras, resulta na redução do tempo de inferência médio do lote.

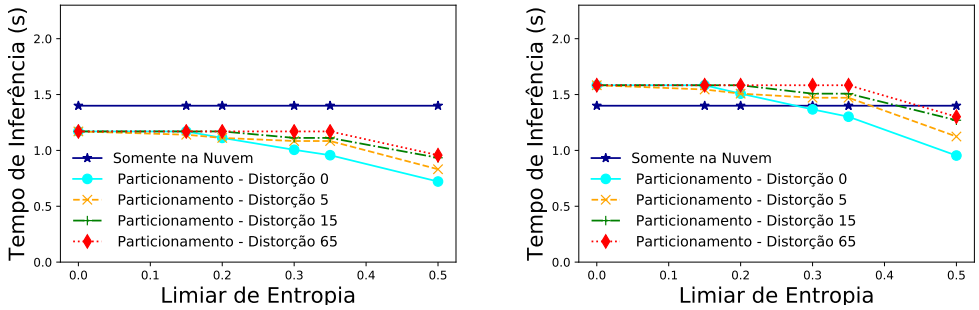
Em relação ao particionamento na camada *fire2*, os resultados da Figura 7.5(b) mostram que o tempo de inferência de imagens com nível de *blur* intermediário (isto é, 15) e alto (isto é, 65) coincidem até um determinado limiar de entropia, porque nenhuma das imagens do conjunto de dados foi suficientemente confiável para ser classificada no primeiro ramo lateral. Contudo, a partir de um dado limiar, as imagens com alto nível de *blur*

seguem sem nenhuma amostra sendo classificada no ramo lateral. Já na curva de nível intermediário, há amostras sendo classificadas no primeiro ramo, o que resulta em redução do tempo de inferência. É importante notar que a classificação antecipada reduz a acurácia, como mostrado na Figura 7.4. Ou seja, há um compromisso entre redução de tempo de inferência e perda de acurácia, conforme apresentado nos experimentos referentes à tarefa de otimização do sistema POPEX.

A Figura 7.5(b) mostra que, em redes com taxa de envio de 1,1 Mbps, o tempo de inferência de imagens com baixo nível de *blur* é até 17,2% inferior em comparação a imagens com alto nível de *blur*, o que corresponde a 228,09 ms. Já em imagens sem nenhum *blur* (isto é, 0), o tempo de inferência é até 21,9% e 35,33% inferior em comparação com as imagens de baixo e alto nível de *blur*, respectivamente. Em relação ao cenário de processamento somente na nuvem, as imagens de baixo nível de *blur* são classificadas 34,25% mais rapidamente. Como imagens com níveis de *blur* diferentes possuem probabilidades diferentes de classificação em cada ramo, o sistema POPEX é capaz de atualizar a probabilidade de classificação dos ramos ao longo do tempo. Dessa forma, o sistema POPEX pode permitir refinar as estratégias de particionamento, conforme a distorção no dado de entrada.

Em relação ao particionamento na camada *fire5*, a Figura 7.5(c) mostra que é mais vantajoso enviar uma imagem à nuvem para ser classificada até o limiar de entropia de 0,15, independentemente do nível de *blur* da imagem. A partir desse limiar, é mais vantajoso adotar a estratégia de particionamento para imagens com baixa distorção, ou seja, sem *blur*. Entretanto, em relação a imagens com baixo e intermediário nível de *blur* ainda é mais vantajoso processar a DNN apenas na nuvem até o limiar de entropia de 0,3, a partir do qual se torna mais vantajoso o particionamento. Assim, esse resultado reafirma que a qualidade da imagem é importante no particionamento. Por exemplo, se for detectado que uma imagem tem baixa qualidade, em alguns casos é desejável enviá-la diretamente para a nuvem ao invés de utilizar o particionamento. Finalmente, considerando o particionamento na camada *fire6*, a Figura 7.5(d) mostra que para limiares de entropia inferiores a 0,35, a estratégia de processamento com o menor tempo de inferência é o processamento somente na nuvem. Nessa camada, em relação a imagens com alto nível de *blur*, a estratégia de particionamento só é vantajosa para limiar de entropia de 0,5, o que pode causar uma redução de acurácia significativa, inviabilizando algumas aplicações.

Considerando uma rede com taxa de envio 18,8 Mbps e o servidor localizado nos EUA, a Figura 7.6(a) mostra que, no particionamento após a camada *fire2*, é mais vantajosa a abordagem de particionamento em comparação a processar a imagem apenas na nuvem, independente do nível de *blur* existente na imagem. Considerando o particionamento, essa figura mostra que o tempo de inferência de imagens sem *blur* é até 48,47% inferior em comparação ao processamento apenas na nuvem e até 31,64% para imagens com maior nível de distorção. Em relação ao particionamento após a camada *fire5*, a Figura 7.6(b)




(a) Particionamento na camada *fire2*. (b) Particionamento na camada *fire5*.

Figura 7.6: Tempo de inferência em função da distorção, para diferentes camadas de particionamento, utilizando uma nuvem nos EUA e taxa de envio de 18,8 Mbps.

ilustra que, diferentemente da Figura 7.5(c), apenas a partir do limiar de entropia 0,3 é mais vantajoso utilizar o particionamento na borda para imagens sem distorção. Na nuvem da Rússia, para a camada *fire5*, isso ocorre a partir do limiar 0,2. Ou seja, para o cenário de rede com taxa de envio de 18,8 Mbps e servidor nos EUA com limiar de entropia de 0,2, a melhor opção é processar na nuvem, enquanto na Rússia, com rede de 1,1 Mbps, é mais vantajoso o processamento baseado em particionamento. Assim, é possível notar que as condições da rede também devem ser consideradas para ajustar o limiar de entropia. O emprego de maior poder computacional na borda faz seu tempo de processamento se aproximar ao da nuvem. Assim, mais camadas são processadas na borda e, no melhor cenário, o processamento pode ser realizado completamente nesse local. Entretanto, esta análise considerou baixo poder computacional na borda, de forma a analisar o pior caso, no qual o particionamento é essencial.

# Capítulo 8

# Conclusões e Trabalhos Futuros

Esta dissertação propôs o sistema POPEX, que aborda o particionamento ótimo de DNNs com saídas antecipadas, equilibrando o compromisso entre acurácia e tempo de inferência. Assim, a otimização determina quais camadas são processadas no dispositivo em borda e quais são processadas na nuvem. Diferentemente das redes neurais tradicionais, as DNNs com saídas antecipadas, como as BranchyNets, possuem ramos laterais que permitem classificar determinadas amostras antecipadamente nas camadas intermediárias, o que reduz o tempo de inferência. Consequentemente, o particionamento da BranchyNet deve considerar a probabilidade de classificar determinadas amostras antecipadamente. Por isso, esta dissertação modelou o tempo de inferência considerando não só os tempos de processamento e o tempo de comunicação, como também a probabilidade de classificação nos ramos laterais. Em seguida, a dissertação modelou a BranchyNet como um grafo, o que permitiu considerar o problema de particionamento de BranchyNet como um problema de particionamento de grafos. Para encontrar o particionamento ótimo do grafo, que equilibre o compromisso entre acurácia e tempo de inferência, a otimização é realizada em duas etapas. A primeira tem como objetivo escolher a combinação dos ramos laterais da BranchyNet que resultem em uma acurácia máxima, executando uma busca exaustiva entre os ramos laterais. Em seguida, a segunda etapa da otimização determina o particionamento que minimiza o tempo de inferência, modelando o problema de particionamento em grafo como um problema de caminho mais curto. Consequentemente, esse problema é resolvido em tempo polinomial usando o algoritmo Dijkstra. O problema de otimização descrito é a principal contribuição do sistema POPEX. O sistema, disponível publicamente na plataforma GitHub, ainda adiciona as funcionalidades de extração dos parâmetros estáticos e dinâmicos que são utilizados no problema de otimização descrito. Além disso, o sistema associa um particionamento ótimo a cada configuração de limiar de entropia.

O particionamento proposto é avaliado por meio de uma análise de sensibilidade, na qual varia-se a probabilidade de classificação nos ramos laterais e o poder de processamento na borda. A partir dos resultados obtidos, mostra-se que a probabilidade de classi-

ficação afeta a escolha da camada de particionamento e, consequentemente, impacta tanto o tempo de inferência quanto a acurácia. Além disso, a dissertação também introduz a probabilidade como um fator a ser considerado no particionamento de BranchyNet. Por fim, os resultados mostram o compromisso entre o tempo de inferência e acurácia em DNNs com saídas antecipadas. De acordo com os resultados obtidos, o limiar de entropia pode ser utilizado para ajustar esse compromisso, com o objetivo de cumprir um requisito de latência predefinido pela aplicação.

Esta dissertação também avaliou, no contexto de DNNs com saídas antecipadas, como o método de calibração *Temperature Scaling* afeta o particionamento. A calibração permite que a confiança da DNN em sua classificação aproxime-se da acurácia. Primeiramente, mostrou-se que os ramos laterais mais próximos da camada de entrada são mais calibrados do que os ramos mais profundos. Em seguida, mostra-se que o método *Temperature Scaling* consegue reduzir os erros de calibração nos ramos laterais e influencia o compromisso entre acurácia e tempo de inferência em BranchyNets. Por um lado, os resultados mostram que ramos laterais bem calibrados podem aumentar o tempo de inferência. Por outro lado, a calibração permite também aumentar a acurácia de cada ramo e, consequentemente, a acurácia final da BranchyNet.

Por fim, este trabalho avaliou o impacto da distorção da imagem no tempo de inferência para classificar imagens. Para isso, foram implementadas redes neurais na borda e na nuvem e comparou-se o tempo de inferência em imagens com diferentes níveis de *blur* nos cenários de processamento somente na borda ou somente na nuvem, como também em um cenário de particionamento no qual a borda e a nuvem são usadas em conjunto. A partir dos resultados obtidos, foi possível concluir que imagens com baixo nível de *blur* tendem a ter uma incerteza de classificação inferior em comparação a imagens com maior nível de *blur*. Dessa forma, mais imagens com baixo nível de *blur* podem ser classificadas no ramo lateral e, consequentemente, na borda da rede. Isso reduz o tempo de inferência. Assim, esta dissertação mostra que fatores relacionados aos dados de entrada, como a distorção na imagem, também afetam a decisão de particionamento.

Os experimentos apresentados, relacionados ao sistema POPEX, avaliam principalmente a otimização executada pelo *Decision Maker*. Em trabalhos futuros, pretende-se avaliar cada componente do sistema POPEX em cenários dinâmicos, ou seja, considerando variações na taxa de envio e na probabilidade de classificação. Além disso, pretende-se investigar heurísticas para o posicionamento de ramos laterais ao longo da BranchyNet, com objetivo de lidar com o erro de *overthinking*. Esse problema ocorre quando o ramo lateral consegue classificar corretamente uma amostra, entretanto o ramo principal classifica erroneamente. Dessa forma, o problema de otimização deve buscar a posição dos ramos laterais para minimizar esse erro. Por fim, pretende-se implementar um estágio de calibração no sistema POPEX para calibrar cada um dos ramos laterais da BranchyNet.

# Apêndice A

# Calibração de confiança com pós-processamento

Este trabalho emprega a calibração em DNNs com saídas antecipadas, considerando um problema de classificação supervisionada com $K$ classes. Nesse contexto, a DNN é modelada como uma função $f_\theta$, sendo $\boldsymbol{\theta}$ os parâmetros aprendidos durante o treinamento. Conforme a Seção, dada uma amostra $\boldsymbol{x}$, a DNN $f_\theta$ gera um vetor de saída $\boldsymbol{z}$. Esse vetor de saída é utilizado para obter o vetor de probabilidades $\boldsymbol{p}$ por meio da função softmax $\sigma(.)$, sendo assim $\boldsymbol{p} = \sigma(\boldsymbol{z})$. O vetor de probabilidade $\boldsymbol{p}$ contém a probabilidade da amostra pertencer a cada uma das $K$ classes. A partir desse vetor de probabilidade $\boldsymbol{p}$, obtém-se uma classe inferida usando a Equação 2.2, e a confiança da classificação $\hat{p}$ é dada pela Equação A.1. A confiança na classificação indica a probabilidade de a amostra de entrada ser de uma determinada classe.

$$\hat{p} = \max_{\{1,\ldots,K\}} (\boldsymbol{p}). \tag{A.1}$$

Uma vez que a dada DNN é treinada, este trabalho utiliza o conjunto de validação para medir a calibração da DNN. Tal conjunto é composto de amostras não vistas pela DNN durante o treinamento. Este trabalho utiliza o ECE (*Expected Calibration Error*) [68] como a métrica para medir a calibração da confiança. Essa métrica mede a calibração calculando a diferença entre confiança $\hat{p}$ e acurácia. Para isso, primeiramente, a métrica ECE divide o intervalo dos valores de confiança ($\hat{p} \in [0, 1]$) em $M$ *bins* igualmente espaçados. Em seguida, as amostras do conjunto de validação são agrupadas nesses $M$ *bins*, conforme a confiança de classificação em cada uma dessas amostras. Portanto, um *bin* $B_m$ contém todas as amostras cuja confiança da classificação pertence a um intervalo $I_m = (\frac{m-1}{M}, \frac{m}{M}]$. Uma vez que essas amostras são agrupadas nos $M$ bins, a acurácia $Acc(B_m)$ em cada *bin* $B_m$ pode ser calculada como

$$Acc(B_m) = \frac{1}{|B_m|} \sum_{i \in I_m} \mathbb{1}[y_i = \hat{y}_i], \tag{A.2}$$

onde $y_i$ e $\hat{y}_i$ são, respectivamente, o rótulo verdadeiro e o inferido pelo modelo da $i$-ésima amostra do conjunto de validação. A função indicadora $\mathbb{1}[y_i = \hat{y}_i]$ assume valor 1 quando $y_i = \hat{y}_i$, e 0 caso contrário. A confiança $Conf(B_m)$ do *bin* $B_m$ é calculada como

$$Conf(B_m) = \frac{1}{|B_m|} \sum_{i \in I_m} \hat{p}_i \tag{A.3}$$

sendo $\hat{p}_i$ a confiança da $i$-ésima amostra.

Após o cálculo de $Acc(B_m)$ e $Conf(B_m)$ para cada um dos *bin*, o valor do ECE é calculado como uma média ponderada da diferença em valor absoluto entre cada $Acc(B_m)$ e $Conf(B_m)$, tal que $m \in \{1, \ldots, M\}$. Assim, o ECE e dado por

$$ECE = \frac{1}{n} \sum_{m=1}^{M} |B_m||Acc(B_m) - Conf(B_m)|, \tag{A.4}$$

onde $n$ é o número de amostras do conjunto de validação.

Após medir a calibração usando a métrica ECE, o objetivo do método de calibração de pós-processamento é gerar uma nova confiança calibrada $\hat{q}_i$ que aproxime a nova confiança $\hat{q}_i$ da acurácia da DNN. Assim, dada uma confiança de classificação $\hat{p}_i$ do modelo da DNN, um método de calibração de pós-processamento pode ser modelado como uma função $g_{\theta'}(\hat{p}_i)$ que mapeia uma confiança não calibrada $\hat{p}_i$ em uma nova confiança de classificação já bem calibrada $\hat{q}_i$, sendo $\theta'$ o conjunto de parâmetros exigidos pelo método de calibração. Este trabalho utiliza o método *Temperature Scaling*, já que Guo *et al.* [17] mostra que esse método supera todos os outros para aplicações de visão computacional. Esse método utiliza um único parâmetro escalar $T > 0$ para obter confiança de classificação bem calibrada da seguinte forma

$$\hat{q} = \max_{\{1,\ldots,K\}} \sigma(\frac{\boldsymbol{z}}{T}), \tag{A.5}$$

onde $\sigma$ é a função *softmax* e $\{1, \ldots, K\}$ é o conjunto de classes predefinidas. Em seguida, esse método aprende o parâmetro $T$, minimizando a função custo *Negative Log-Likelihood* (NLL).

# Referências Bibliográficas


[1] LIU, W., WANG, Z., LIU, X., et al. "A survey of deep neural network architectures and their applications", *Neurocomputing*, v. 234, pp. 11–26, 2017.

[2] BORKAR, T. S., KARAM, L. J. "DeepCorrect: Correcting DNN models against image distortions", *IEEE Transactions on Image Processing*, v. 28, n. 12, pp. 6022–6034, 2019.

[3] KIM, S., PARK, S., NA, B., et al. "Spiking-yolo: Spiking neural network for real-time object detection", *arXiv preprint arXiv:1903.06530*, v. 1, 2019.

[4] LIU, X., HE, P., CHEN, W., et al. "Multi-task deep neural networks for natural language understanding", *arXiv preprint arXiv:1901.11504*, 2019.

[5] WANG, J., SUN, J., LIN, H., et al. "Convolutional neural networks for expert recommendation in community question answering", *Science China Information Sciences*, v. 60, n. 11, pp. 110102, 2017.

[6] TIAN, Y., PEI, K., JANA, S., et al. "Deeptest: Automated testing of deep-neural-network-driven autonomous cars". In: *Proceedings of the 40th international conference on software engineering*, pp. 303–314, 2018.

[7] BECHTEL, M. G., MCELLHINEY, E., KIM, M., et al. "Deeppicar: A low-cost deep neural network-based autonomous car". In: *IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 11–21, 2018.

[8] KIM, H., LEE, Y., YIM, B., et al. "On-road object detection using deep neural network". In: *IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, pp. 1–4, 2016.

[9] LECUN, Y., BENGIO, Y., HINTON, G. "Deep learning", *nature*, v. 521, n. 7553, pp. 436, 2015.

[10] SATYANARAYANAN, M. "The emergence of edge computing", *Computer*, v. 50, n. 1, pp. 30–39, 2017.

[11] KANG, Y., HAUSWALD, J., GAO, C., et al. “Neurosurgeon: Collaborative intelligence between the cloud and mobile edge”. In: *ACM Computer Architecture News (SIGARCH)*, v. 45, pp. 615–629, 2017.

[12] CHEN, Z., HU, W., WANG, J., et al. “An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance”. In: *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, p. 14. ACM, 2017.

[13] HOBERT, L., FESTAG, A., LLATSER, I., et al. “Enhancements of V2X communication in support of cooperative autonomous driving”, *IEEE communications magazine*, v. 53, n. 12, pp. 64–70, 2015.

[14] XU, M., QIAN, F., ZHU, M., et al. “Deepwear: Adaptive local offloading for on-wearable deep learning”, *IEEE Transactions on Mobile Computing*, v. 19, n. 2, pp. 314–330, 2019.

[15] TEERAPITTAYANON, S., MCDANEL, B., KUNG, H.-T. “Branchynet: Fast inference via early exiting from deep neural networks”. In: *IEEE International Conference on Pattern Recognition (ICPR)*, pp. 2464–2469, 2016.

[16] HU, C., BAO, W., WANG, D., et al. “Dynamic Adaptive DNN Surgery for Inference Acceleration on the Edge”. In: *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1423–1431, 2019.

[17] GUO, C., PLEISS, G., SUN, Y., et al. “On calibration of modern neural networks”, *arXiv preprint arXiv:1706.04599*, 2017.

[18] SZE, V., CHEN, Y.-H., YANG, T.-J., et al. “Efficient processing of deep neural networks: A tutorial and survey”, *Proceedings of the IEEE*, v. 105, n. 12, pp. 2295–2329, 2017.

[19] KRIZHEVSKY, A., SUTSKEVER, I., HINTON, G. E. “Imagenet classification with deep convolutional neural networks”. In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097–1105, 2012.

[20] REDMON, J., DIVVALA, S., GIRSHICK, R., et al. “You only look once: Unified, real-time object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[21] HE, S., LAU, R. W., LIU, W., et al. “Supercnn: A superpixelwise convolutional neural network for salient object detection”, *International journal of computer vision*, v. 115, n. 3, pp. 330–344, 2015.

[22] CAI, Z., FAN, Q., FERIS, R. S., et al. "A unified multi-scale deep convolutional neural network for fast object detection". In: *European conference on computer vision*, pp. 354–370. Springer, 2016.

[23] BADRINARAYANAN, V., KENDALL, A., CIPOLLA, R. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation", *IEEE transactions on pattern analysis and machine intelligence*, v. 39, n. 12, pp. 2481–2495, 2017.

[24] LIU, C., CHEN, L.-C., SCHROFF, F., et al. "Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 82–92, 2019.

[25] KHAN, A., SOHAIL, A., ZAHOORA, U., et al. "A survey of the recent architectures of deep convolutional neural networks", *Artificial Intelligence Review*, pp. 1–62, 2020.

[26] HE, K., ZHANG, X., REN, S., et al. "Spatial pyramid pooling in deep convolutional networks for visual recognition", *IEEE transactions on pattern analysis and machine intelligence*, v. 37, n. 9, pp. 1904–1916, 2015.

[27] IANDOLA, F. N., HAN, S., MOSKEWICZ, M. W., et al. "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$ 0.5 mb model size", *arXiv preprint arXiv:1602.07360*, 2016.

[28] SCARDAPANE, S., SCARPINITI, M., BACCARELLI, E., et al. "Why should we add early exits to neural networks?" *arXiv preprint arXiv:2004.12814*, 2020.

[29] LI, E., ZENG, L., ZHOU, Z., et al. "Edge AI: On-demand accelerating deep neural network inference via edge computing", *IEEE Transactions on Wireless Communications*, v. 19, n. 1, pp. 447–457, 2019.

[30] LEE, C.-Y., XIE, S., GALLAGHER, P., et al. "Deeply-supervised nets". In: *Artificial intelligence and statistics*, pp. 562–570, 2015.

[31] BIOOKAGHAZADEH, S., ZHAO, M., REN, F. "Are FPGAs Suitable for Edge Computing?" In: *USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18)*, Boston, MA, jul. 2018. USENIX Association. Disponível em: <https://www.usenix.org/conference/hotedge18/presentation/biookaghazadeh>.

[32] KIM, Y., KIM, J., CHAE, D., et al. "$\mu$Layer: Low Latency On-Device Inference Using Cooperative Single-Layer Acceleration and Processor-Friendly Quantization". In: *Proceedings of the Fourteenth EuroSys Conference 2019*, pp. 1–15, 2019.

[33] ZHANG, C., LI, P., SUN, G., et al. "Optimizing fpga-based accelerator design for deep convolutional neural networks". In: *Proceedings of the ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 161–170, 2015.

[34] WU, J., LENG, C., WANG, Y., et al. "Quantized convolutional neural networks for mobile devices". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4820–4828, 2016.

[35] HAN, S., SHEN, H., PHILIPOSE, M., et al. "Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints". In: *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 123–136, 2016.

[36] HAN, S., MAO, H., DALLY, W. J. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding", *arXiv preprint arXiv:1510.00149*, 2015.

[37] PANDA, P., SENGUPTA, A., ROY, K. "Energy-efficient and improved image recognition with conditional deep learning", *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, v. 13, n. 3, pp. 1–21, 2017.

[38] WANG, M., MO, J., LIN, J., et al. "DynExit: A Dynamic Early-Exit Strategy for Deep Residual Networks". In: *IEEE International Workshop on Signal Processing Systems (SiPS)*, pp. 178–183, 2019.

[39] BOLUKBASI, T., WANG, J., DEKEL, O., et al. "Adaptive neural networks for efficient inference". In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 527–536. JMLR. org, 2017.

[40] STAMOULIS, D., CHIN, T.-W., PRAKASH, A. K., et al. "Designing adaptive neural networks for energy-constrained image classification". In: *Proceedings of the International Conference on Computer-Aided Design*, pp. 1–8, 2018.

[41] WANG, Z., BAO, W., YUAN, D., et al. "SEE: Scheduling Early Exit for Mobile DNN Inference during Service Outage". In: *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 279–288, 2019.

[42] KAYA, Y., HONG, S., DUMITRAS, T. "Shallow-Deep Networks: Understanding and Mitigating Network Overthinking", *arXiv preprint arXiv:1810.07052*, 2018.

[43] LI, H., OTA, K., DONG, M. "Learning IoT in edge: deep learning for the internet of things with edge computing", *IEEE Network*, v. 32, n. 1, pp. 96–101, 2018.

[44] HE, Y., YU, F. R., ZHAO, N., et al. "Software-defined networks with mobile edge computing and caching for smart cities: A big data deep reinforcement learning approach", *IEEE Communications Magazine*, v. 55, n. 12, pp. 31–37, 2017.

[45] BOJARSKI, M., DEL TESTA, D., DWORAKOWSKI, D., et al. "End to end learning for self-driving cars", *arXiv preprint arXiv:1604.07316*, 2016.

[46] XU, H., GAO, Y., YU, F., et al. "End-to-end learning of driving models from large-scale video datasets". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2174–2182, 2017.

[47] CHEN, Z., HUANG, X. "End-to-end learning for lane keeping of self-driving cars". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1856–1860, 2017.

[48] ZHANG, T., CHOWDHERY, A., BAHL, P., et al. "The design and implementation of a wireless video surveillance system". In: *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, pp. 426–438, 2015.

[49] CHEN, J., RAN, X. "Deep learning with edge computing: A review", *Proceedings of the IEEE*, v. 107, n. 8, pp. 1655–1674, 2019.

[50] CUERVO, E., BALASUBRAMANIAN, A., CHO, D.-K., et al. "MAUI: making smartphones last longer with code offload". In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*, pp. 49–62, 2010.

[51] GORDON, M. S., JAMSHIDI, D. A., MAHLKE, S., et al. "COMET: Code Offload by Migrating Execution Transparently". In: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, pp. 93–106, 2012.

[52] KO, J. H., NA, T., AMIR, M. F., et al. "Edge-host partitioning of deep neural networks with feature space encoding for resource-constrained internet-of-things platforms". In: *IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pp. 1–6, 2018.

[53] TEERAPITTAYANON, S., MCDANEL, B., KUNG, H.-T. "Distributed deep neural networks over the cloud, the edge and end devices". In: *IEEE International Conference on Distributed Computing Systems (ICDCS)*, pp. 328–339, 2017.

[54] SZEGEDY, C., LIU, W., JIA, Y., et al. "Going deeper with convolutions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[55] HE, K., ZHANG, X., REN, S., et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[56] PACHECO, R. G., COUTO, R. S. "Inference Time Optimization Using BranchyNet Partitioning". In: *IEEE Symposium on Computers and Communications (ISCC)*, pp. 1–7, 2020.

[57] DODGE, S., KARAM, L. "Understanding how image quality affects deep neural networks". In: *IEEE International Conference on Quality of Multimedia Experience (QoMEX)*, pp. 1–6, 2016.

[58] KRIZHEVSKY, A., NAIR, V., HINTON, G. "The cifar-10 dataset", *online: http://www. cs. toronto. edu/kriz/cifar. html*, v. 55, 2014.

[59] KENDALL, A., GAL, Y. "What uncertainties do we need in bayesian deep learning for computer vision?" In: *Advances in neural information processing systems*, pp. 5574–5584, 2017.

[60] OGAWA, T., SAKAI, H., SUZUKI, Y., et al. "Pedestrian detection and tracking using in-vehicle lidar for automotive application". In: *IEEE Intelligent Vehicles Symposium (IV)*, pp. 734–739, 2011.

[61] CHEN, J., XU, H., WU, J., et al. "Deer crossing road detection with roadside LiDAR sensor", *IEEE Access*, v. 7, pp. 65944–65954, 2019.

[62] KRIZHEVSKY, A. "One weird trick for parallelizing convolutional neural networks", *arXiv preprint arXiv:1404.5997*, 2014.

[63] PLATT, J., OTHERS. "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods", *Advances in large margin classifiers*, v. 10, n. 3, pp. 61–74, 1999.

[64] CARUANA, R., LOU, Y., GEHRKE, J., et al. "Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission". In: *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1721–1730, 2015.

[65] PACHECO, R. G., COUTO, R. S. “Introduzindo a qualidade da imagem como uma nova condição de particionamento de DNN na borda”. In: *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC)*, pp. 1–14, 2020. Aceito para publicação.

[66] PARKHI, O. M., VEDALDI, A., ZISSERMAN, A., et al. “Cats and dogs”. In: *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3498–3505, 2012.

[67] ZHANG, X., WANG, Y., SHI, W. “pcamp: Performance comparison of machine learning packages on the edges”. In: *USENIX Workshop on Hot Topics in Edge Computing (HotEdge)*, 2018.

[68] NAEINI, M. P., COOPER, G. F., HAUSKRECHT, M. “Obtaining well calibrated probabilities using bayesian binning”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, v. 2015, p. 2901. NIH Public Access, 2015.